ACM Computing Surveys

RAID: High-Performance, Reliable Secondary Storage

Peter M. Chen
Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
1301 Beal Avenue
University of Michigan
Ann Arbor, MI 48109-2122

Edward K. Lee
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301-1044

Garth A. Gibson
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Randy H. Katz
Computer Science Division
Department of Electrical Engineering and Computer Science
571 Evans Hall
University of California
Berkeley, CA 94720

David A. Patterson
Computer Science Division
Department of Electrical Engineering and Computer Science
571 Evans Hall
University of California
Berkeley, CA 94720

Abstract: Disk arrays were proposed in the 1980s as a way to use parallelism between multiple disks to improve aggregate I/O performance. Today they appear in the product lines of most major computer manufacturers. **This paper gives a comprehensive overview of disk arrays and provides a framework in which to organize current and future work.** The paper first introduces disk technology and reviews the driving forces that have popularized disk arrays: performance and reliability. It then discusses the two architectural techniques used in disk arrays: striping across multiple disks to improve performance and redundancy to improve reliability. Next, the paper describes seven disk array architectures, called RAID (Redundant Arrays of Inexpensive Disks) levels 0-6 and compares their performance, cost, and reliability. It goes on to discuss advanced research and implementation topics such as refining the basic RAID levels to improve performance and designing algorithms to maintain data consistency. Last, the paper describes six disk array prototypes or products and discusses future opportunities for research. The paper includes an annotated bibliography of disk array-related literature.

Content indicators: (Computing Reviews Classification) B.4.2 and B.4.5 [Input/Output and Data Communications]: Input/Output Devices; Reliability, Testing, and Fault-Tolerance; D.4.2 [Operating Systems]: Storage Management; E.4 [Coding and Information Theory]. Other keywords: disk array, RAID, parallel I/O, storage, striping, redundancy

1 INTRODUCTION

In recent years, interest in RAID, Redundant Arrays of Inexpensive Disks[1], has grown explosively. The driving force behind this phenomenon is the sustained exponential improvements in the performance and density of semiconductor technology. Improvements in semiconductor technology make possible faster microprocessors and larger primary memory systems which in turn require larger, higher-performance secondary storage systems. These improvements have both quantitative and qualitative consequences.

On the quantitative side, Amdahl's Law [Amdahl67] predicts that large improvements in microprocessors will result in only marginal improvements in overall system performance unless accompanied by corresponding improvements in secondary storage systems. Unfortunately, while RISC microprocessor performance has been improving 50% or more per year [Patterson94, pg. 27], disk access times, which depend on improvements of mechanical systems, have been improving less than 10% per year. Disk transfer rates, which track improvements in both mechanical systems and magnetic media densities, have improved at the faster rate of approximately 20% per year, but this is still far slower than the rate of processor improvement. Assuming that semiconductor and disk technologies continue their current trends, we must conclude that the performance gap between microprocessors and magnetic disks will continue to widen.

In addition to the quantitative effect, a second, perhaps more important, qualitative effect is driving the need for higher-performance secondary storage systems. As microprocessors become faster, they make possible new applications and greatly expand the scope of existing applications. In particular, image-intensive applications such as video, hypertext and multi-media are becoming common. Even in existing application areas such as computer-aided design and scientific computing, faster microprocessors make it possible to tackle new problems requiring faster access to larger datasets. This shift in applications, along with a trend toward large, shared, high-perfor-

1. Because of the restrictiveness of "Inexpensive", RAID is sometimes said to stand for "Redundant Arrays of Independent Disks".

mance, network-based storage systems, is causing us to reevaluate the way we design and use secondary storage systems [Katz92].

Disk arrays, which organize multiple, independent disks into a large, high-performance logical disk, are a natural solution to the problem. Disk arrays stripe data across multiple disks and access them in parallel to achieve both higher data transfer rates on large data accesses and higher I/O rates on small data accesses [Salem86, Livny87]. Data striping also results in uniform load balancing across all of the disks, eliminating hot spots that otherwise saturate a small number of disks while the majority of disks sit idle.

Large disk arrays, however, are highly vulnerable to disk failures; a disk array with a hundred disks is a hundred times more likely to fail than a single disk. An MTTF (mean-time-to-failure) of 200,000 hours, or approximately twenty-three years, for a single disk implies an MTTF of 2000 hours, or approximately three months, for a disk array with a hundred disks. The obvious solution is to employ redundancy in the form of error-correcting codes to tolerate disk failures. This allows a redundant disk array to avoid losing data for much longer than an unprotected single disk. Redundancy, however, has negative consequences. Since all write operations must update the redundant information, the performance of writes in redundant disk arrays can be significantly worse than the performance of writes in non-redundant disk arrays. Also, keeping the redundant information consistent in the face of concurrent I/O operations and system crashes can be difficult.

A number of different data striping and redundancy schemes have been developed. The combinations and arrangements of these schemes lead to a bewildering set of options for users and designers of disk arrays. Each option presents subtle tradeoffs between reliability, performance and cost that are difficult to evaluate without understanding the alternatives. To address this problem, this paper presents a systematic tutorial and survey of disk arrays. We describe seven basic disk array organizations along with their advantages and disadvantages and compare their reliability, performance and cost. We draw attention to the general principles governing the design and configuration of disk arrays as well as practical issues that must be addressed in the implementation of disk arrays. A later section of the paper describes optimizations and variations to the seven basic disk-array organizations. Finally, we discuss existing research in the modeling of disk arrays and fruitful avenues for future research. This paper should be of value to anyone interested in disk arrays, including students, researchers, designers and users of disk arrays.

2 BACKGROUND

This section provides basic background material on disks, I/O datapaths, and disk technology trends for readers who are unfamiliar with secondary storage systems.

2.1 Disk Terminology

Figure 1 illustrates the basic components of a simplified magnetic disk drive. A disk principally consists of a set of *platters* coated with a magnetic medium rotating at a constant angular velocity and a set of disk *arms* with magnetic read/write *heads* that are moved radially across the platters' surfaces by an *actuator*. Once the heads are correctly positioned, data is read and written in small arcs called *sectors* on the platters' surfaces as the platters rotate relative to the heads. Although all heads are moved collectively, in almost every disk drive, only a single head can read or write data at any given time. A complete circular swath of data is referred to as a *track* and each platter's surface consists of concentric rings of tracks. A vertical collection of tracks at the same

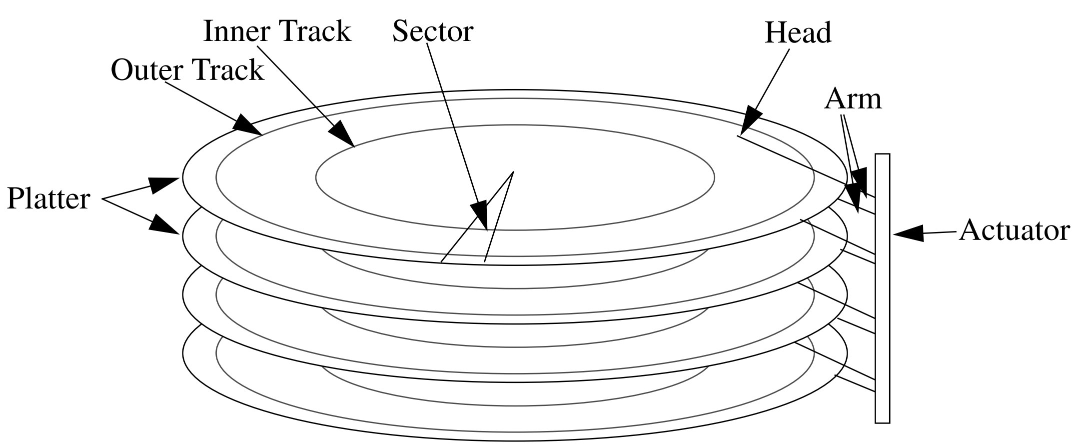


Figure 1: Disk Terminology. Heads reside on arms which are positioned by actuators. Tracks are concentric rings on a platter. A sector is the basic unit of reads and writes. A cylinder is a stack of tracks at one actuator position. An *HDA (head-disk assembly)* is everything in the figure plus the airtight casing. In some devices it is possible to transfer data from multiple surfaces simultaneously, but this is both rare and expensive. The collection of heads that participate in a single logical transfer that is spread over multiple surfaces is called a *head group*.

radial position is logically referred to as a *cylinder*. Sectors are numbered so that a sequential scan of all sectors traverses the entire disk in the minimal possible time.

Given the simplified disk described above, disk service times can be broken into three primary components: *seek time*, *rotational latency*, and *data transfer time*. Seek time is the amount of time needed to move a head to the correct radial position and typically ranges from one to thirty milliseconds depending on the seek distance and the particular disk. Rotational latency is the amount of time needed for the desired sector to rotate under the disk head. Full rotation times for disks currently vary from eight to twenty-eight milliseconds. The data transfer time is dependent on the rate at which data can be transferred to/from a platter's surface and is a function of the platter's rate of rotation, the density of the magnetic media, and the radial distance of the head from the center of the platter—some disks use a technique called zone-bit-recording to store more data on the longer outside tracks than the shorter inside tracks. Typical data transfer rates range from one to five megabytes per second. The seek time and rotational latency are sometimes collectively referred to as the *head positioning time*. Table 1 tabulates the statistics for a typical high-end disk available in 1993.

Form Factor/Disk Diameter	5.25 inch
Capacity	2.8 GB
Cylinders	2627
Tracks Per Cylinder	21
Sectors Per Track	~99
Bytes Per Sector	512
Full Rotation Time	11.1 ms
Minimum Seek (single cylinder)	1.7 ms
Average Seek (random cylinder to cylinder)	11.0 ms
Maximum Seek (full stroke seek)	22.5 ms
Data Transfer Rate	≈ 4.6 MB/s

Table 1: Specifications for the Seagate ST43401N Elite-3 SCSI Disk Drive. Average seek in this table is calculated assuming a uniform distribution of accesses. This is the standard way manufacturers report average seek times. In reality, measurements of production systems show that spatial locality significantly lowers the effective average seek distance [Hennessy90, pg. 559].

The slow head positioning time and fast data transfer rate of disks lead to very different performance for a sequence of accesses depending on the size and relative location of each access. Suppose we need to transfer 1 MB from the disk in Table 1, and the data is laid out in two ways: sequential within a single cylinder or randomly placed in 8 KB blocks. In either case the time for the actual data transfer of 1 MB is about 200 ms. But the time for positioning the head goes from about 16 ms in the sequential layout to about 2000 ms in the random layout. This sensitivity to the workload is why I/O-intensive applications are categorized as *high data rate*, meaning minimal head positioning via large, sequential accesses, or *high I/O rate*, meaning lots of head positioning via small, more random accesses. For example, scientific programs that manipulate large arrays of data fall in the high data rate category, while transaction processing programs fall in the high I/O rate category.

2.2 Data Paths

A hierarchy of industry standard interfaces has been defined for transferring data recorded on a disk platter's surface to or from a host computer. In this section we review the complete datapath, from a disk to a users' application (Figure 2). We assume a read operation for the purposes of this discussion.

On the disk platter's surface, information is represented as reversals in the direction of stored magnetic fields. These "flux reversals" are sensed, amplified, and digitized into pulses by the lowest-level read electronics. The protocol ST506/412 is one standard that defines an interface to disk systems at this lowest, most inflexible, and technology-dependent level. Above this level of the read electronics path, pulses are decoded to separate data bits from timing-related flux reversals. The bit-level ESDI and SMD standards define an interface at this more flexible, encoding-independent level. Then, to be transformed into the highest, most-flexible packet-level, these bits are aligned into bytes, error correcting codes applied, and the extracted data delivered to the host as data blocks over a peripheral bus interface such as SCSI (Small Computer Standard Interface), or IPI-3 (the third level of the Intelligent Peripheral Interface). These steps are performed today by intelligent on-disk controllers, which often include speed matching and caching "track buffers".

SCSI and IPI-3 also include a level of data mapping: the computer specifies a logical block number and the controller embedded on the disk maps that block number to a physical cylinder, track, and sector. This mapping allows the embedded disk controller to avoid bad areas of the disk by remapping logical blocks that are affected to new areas of the disk.

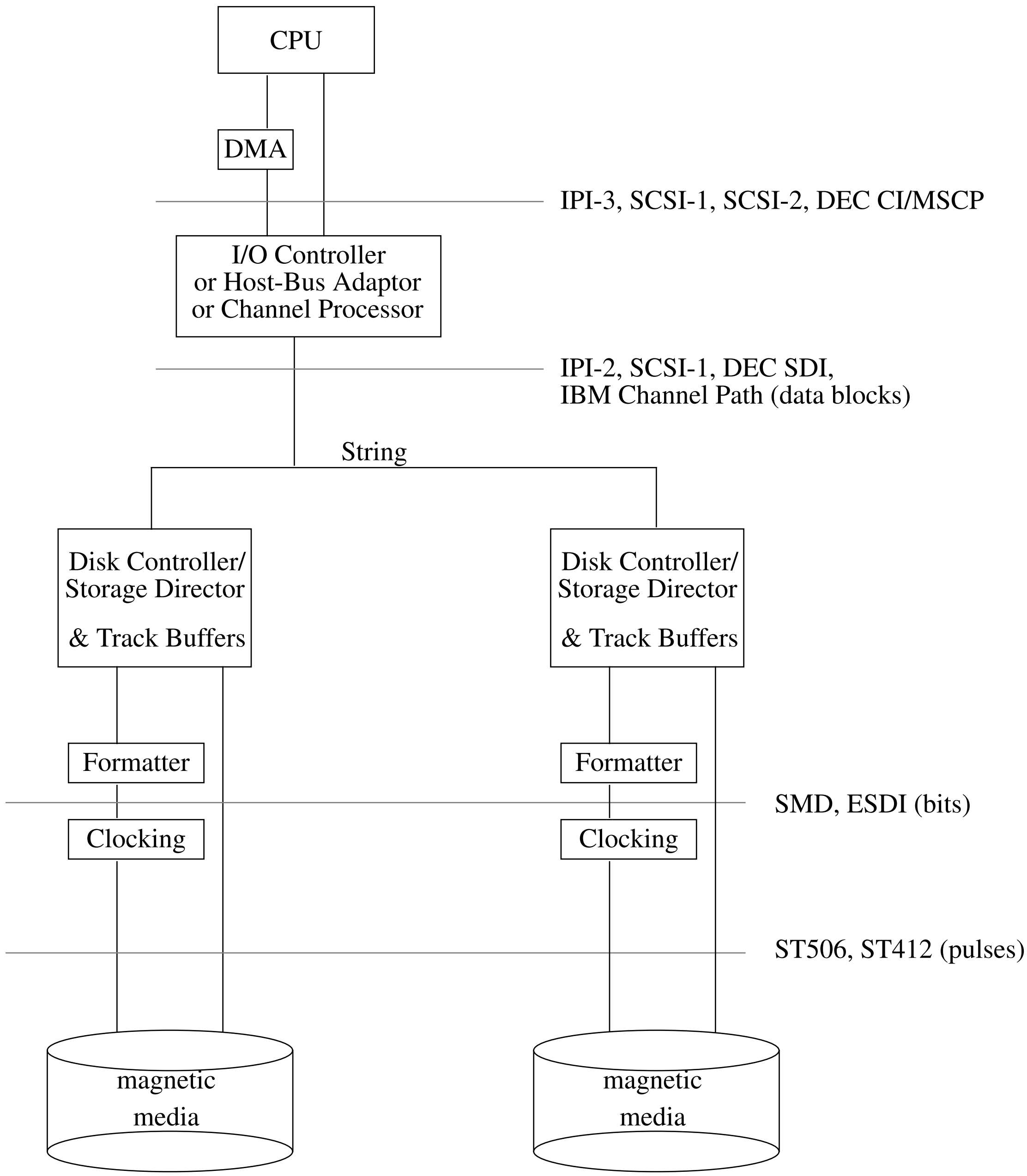


Figure 2: Host-to-Device Pathways. Data that is read from a magnetic disk must pass through many layers on its way to the requesting processor. Each dashed line marks a standard interface. Lower interfaces such as ST506 deal more closely with the raw magnetic fields and are highly technology dependent. Higher layers such as SCSI deal in packets or blocks of data and are more technology independent. A string connects multiple disks to a single I/O controller; control of the string is distributed between the I/O and disk controllers.

Topologies and devices on the data path between disk and host computer vary widely depending on the size and type of I/O system. Mainframes have the richest I/O systems, with many devices and complex interconnection schemes to access them. An IBM *channel path*, which encompasses the set of cables and associated electronics that transfer data and control information between an I/O device and main memory, consists of a *channel*, a *storage director*, and a *head of string*. The collection of disks that share the same pathway to the head of string is called a *string*. In the workstation/file server world, the channel processor is usually called an I/O controller or host-bus adaptor (HBA) and the functionality of the storage director and head of string is contained in an embedded controller on the disk drive. As in the mainframe world, the use of high-level peripheral interfaces such as SCSI and IPI-3 allow multiple disks to share a single peripheral bus or string.

From the host-bus adaptor, data is transferred via direct memory access, over a system bus, such as VME, S-Bus, MicroChannel, EISA, or PCI, to the host operating system's buffers. In most operating systems, the CPU then performs a memory-to-memory copy over a high-speed memory bus from the operating system buffers to buffers in the application's address space.

2.3 Technology Trends

Much of the motivation for disk arrays comes from the current trends in disk technology. As Table 2 shows, magnetic disk drives have been improving rapidly by some metrics and hardly at all by other metrics. Smaller distances between the magnetic read/write head and the disk surface, more accurate positioning electronics, and more advanced magnetic media have dramatically increased the recording density on the disks. This increased density has improved disks in two ways. First, it has allowed disk capacities to stay constant or increase, even while disk sizes have decreased from 5.25" in 1983 to 1.3" in 1993. Second, the increased density, along with an increase in the rotational speed of the disk, has made possible a substantial increase in the transfer rate of disk drives. Seek times, on the other hand, have improved very little, only decreasing from approximately 20 ms in 1980 to 10 ms today. Rotational speeds have increased at a similarly slow rate from 3600 revolutions per minute in 1980 to 5400-7200 today.

3 DISK ARRAY BASICS

This section examines basic issues in the design and implementation of disk arrays. In particular, we examine the concepts of data striping and redundancy; basic RAID organizations; performance and cost comparisons between the basic RAID organizations; reliability of RAID-based systems in the face of system crashes, uncorrectable bit-errors and correlated disk failures; and finally, issues in the implementation of block-interleaved, redundant disk arrays.

3.1 Data Striping and Redundancy

Redundant disk arrays employ two orthogonal concepts: data striping for improved performance and redundancy for improved reliability. Data striping transparently distributes data over multiple disks to make them appear as a single fast, large disk. Striping improves aggregate I/O performance by allowing multiple I/Os to be serviced in parallel. There are two aspects to this parallelism. First, multiple, independent requests can be serviced in parallel by separate disks. This decreases the queueing time seen by I/O requests. Second, single, multiple-block requests can be serviced by multiple disks acting in coordination. This increases the effective transfer rate seen by a single request. The more disks in the disk array, the larger the potential performance benefits. Unfortunately, a large number of disks lowers the overall reliability of the disk array, as mentioned

	1993	**Historical Rate of Improvement**
Areal Density	50-150 Mbits/sq. inch	27% per year
Linear Density	40,000-60,000 bits/inch	13% per year
Inter-Track Density	1,500-3,000 tracks/inch	10% per year
Capacity (3.5” form factor)	100-2000 MB	27% per year
Transfer Rate	3-4 MB/s	22% per year
Seek Time	7-20 ms	8% per year

Table 2: Trends in Disk Technology. Magnetic disks are improving rapidly in density and capacity, but more slowly in performance. Areal density is the recording density per square inch of magnetic media. In 1989, IBM demonstrated a 1 Gbit/sq. inch density in a laboratory environment. Linear density is the number of bits written along a track. Inter-track density refers to the number of concentric tracks on a single platter.

before. Assuming independent failures, 100 disks collectively have only 1/100th the reliability of a single disk. Thus, redundancy is necessary to tolerate disk failures and allow continuous operation without data loss.

We will see that the majority of redundant disk array organizations can be distinguished based on two features: 1) the granularity of data interleaving and 2) the method and pattern in which the redundant information is computed and distributed across the disk array. Data interleaving can be characterized as either fine-grained or coarse-grained. Fine-grained disk arrays conceptually interleave data in relatively small units so that all I/O requests, regardless of their size, access all of the disks in the disk array. This results in very high data transfer rates for all I/O requests but has the disadvantages that only one logical I/O request can be in service at any given time and all disks must waste time positioning for every request. Coarse-grained disk arrays interleave data in relatively large units so that small I/O requests need access only a small number of disks while large requests can access all the disks in the disk array. This allows multiple, small requests to be serviced simultaneously while still allowing large requests to see the higher transfer rates afforded by using multiple disks.

The incorporation of redundancy in disk arrays brings up two somewhat orthogonal problems. The first problem is selecting the method for computing the redundant information. Most redundant disk arrays today use parity, though some use Hamming or Reed-Solomon codes. The second problem is that of selecting a method for distributing the redundant information across the disk array. Although there are an unlimited number of patterns in which redundant information can be distributed, we roughly classify these patterns into two different distributions schemes, those that concentrate redundant information on a small number of disks and those that distributed redundant information uniformly across all of the disks. Schemes that uniformly distribute redundant information are generally more desirable because they avoid hot spots and other load balancing problems suffered by schemes that do not uniformly distribute redundant information. Although the basic concepts of data striping and redundancy are conceptually simple, selecting between the many possible data striping and redundancy schemes involves complex tradeoffs between reliability, performance and cost.

3.2 Basic RAID Organizations

This section describes the basic RAID, Redundant Arrays of Inexpensive Disks, organizations that will be used as the basis for further examinations of the performance, cost, and reliability of disk arrays. In addition to presenting RAID levels 1 through 5 that first appeared in the landmark paper by Patterson, Gibson and Katz [Patterson88], we present two other RAID organizations, RAID levels 0 and 6, that have since become generally accepted[1]. For the benefit of those unfamiliar with the original numerical classification of RAID, we will use English phrases in preference to the numerical classifications. It should come as no surprise to the reader that even the original authors have sometimes been confused as to the disk array organization referred to by a particular RAID level! Figure 3 schematically illustrates the seven RAID organizations.

3.2.1 Non-Redundant (RAID Level 0)

A non-redundant disk array, or RAID level 0, has the lowest cost of any RAID organization because it does not employ redundancy at all. This scheme offers the best write performance since it never needs to update redundant information. Surprisingly, it does not have the best read performance. Redundancy schemes that duplicate data, such as mirroring, can perform better on reads by selectively scheduling requests on the disk with the shortest expected seek and rotational delays [Bitton88]. Without redundancy, any single disk failure will result in data-loss. Non-redundant disk arrays are widely used in supercomputing environments where performance and capacity, rather than reliability, are the primary concerns.

3.2.2 Mirrored (RAID Level 1)

The traditional solution, called *mirroring* or *shadowing*, uses twice as many disks as a non-redundant disk array [Bitton88]. Whenever data is written to a disk the same data is also written to a redundant disk, so that there are always two copies of the information. When data is read, it can be retrieved from the disk with the shorter queueing, seek and rotational delays [Chen90a]. If a disk fails, the other copy is used to service requests. Mirroring is frequently used in database appli-

1. Strictly speaking, RAID Level 0 is not a type of *redundant* array of inexpensive disks since it stores no error-correcting codes.

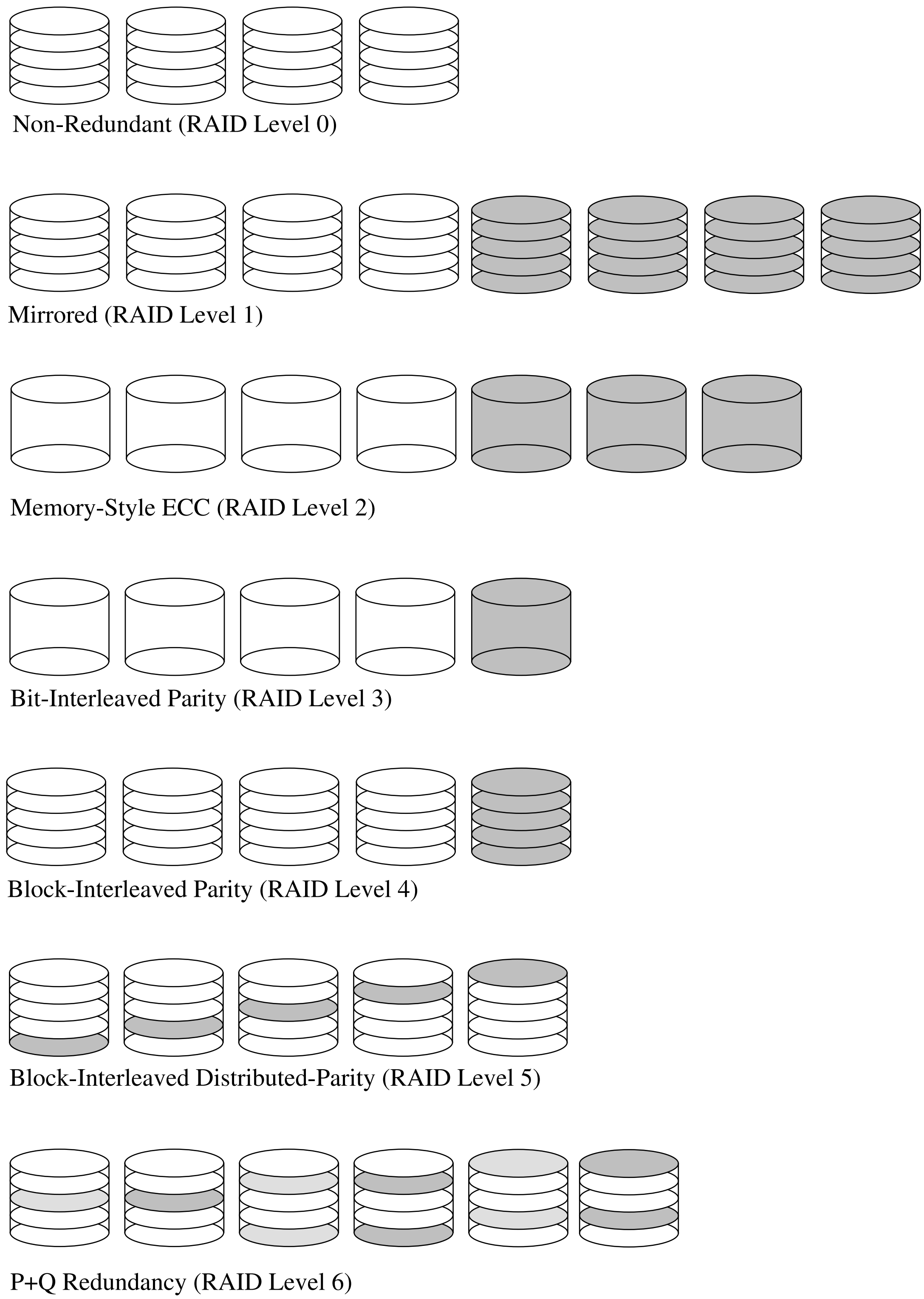


Figure 3:RAID Levels 0 Through 6. All RAID levels are illustrated at a user capacity of four disks. Disks with multiple platters indicate block-level striping while disks without multiple platters indicate bit-level striping. The shaded platters represent redundant information.

cations where availability and transaction rate are more important than storage efficiency [Gray90].

3.2.3 Memory-Style ECC (RAID Level 2)

Memory systems have provided recovery from failed components with much less cost than mirroring by using Hamming codes [Peterson72]. Hamming codes contain parity for distinct overlapping subsets of components. In one version of this scheme, four data disks require three redundant disks, one less than mirroring. Since the number of redundant disks is proportional to the log of the total number of disks in the system, storage efficiency increases as the number of data disks increases.

If a single component fails, several of the parity components will have inconsistent values, and the failed component is the one held in common by each incorrect subset. The lost information is recovered by reading the other components in a subset, including the parity component, and setting the missing bit to 0 or 1 to create the proper parity value for that subset. Thus, multiple redundant disks are needed to identify the failed disk, but only one is needed to recover the lost information.

Readers unfamiliar with parity can think of the redundant disk as having the sum of all the data in the other disks. When a disk fails, you can subtract all the data on the good disks from the parity disk; the remaining information must be the missing information. Parity is simply this sum modulo two.

3.2.4 Bit-Interleaved Parity (RAID Level 3)

One can improve upon memory-style ECC disk arrays by noting that, unlike memory component failures, disk controllers can easily identify which disk has failed. Thus, one can use a single parity disk rather than a set of parity disks to recover lost information.

In a bit-interleaved, parity disk array, data is conceptually interleaved bit-wise over the data disks, and a single parity disk is added to tolerate any single disk failure. Each read request accesses all data disks and each write request accesses all data disks and the parity disk. Thus, only one request can be serviced at a time. Because the parity disk contains only parity and no data, the parity disk cannot participate on reads, resulting in slightly lower read performance than for redundancy schemes that distribute the parity and data over all disks. Bit-interleaved, parity disk arrays are frequently used in applications that require high bandwidth but not high I/O rates. They are also simpler to implement than RAID Levels 4, 5, and 6.

3.2.5 Block-Interleaved Parity (RAID Level 4)

The block-interleaved, parity disk array is similar to the bit-interleaved, parity disk array except that data is interleaved across disks in blocks of arbitrary size rather than in bits. The size of these blocks is called the striping unit [Chen90b]. Read requests smaller than the striping unit access only a single data disk. Write requests must update the requested data blocks and must also compute and update the parity block. For large writes that touch blocks on all disks, parity is easily computed by exclusive-or'ing the new data for each disk. For small write requests that update only one data disk, parity is computed by noting how the new data differs from the old data and applying those differences to the parity block. Small write requests thus require four disk I/Os: one to write the new data, two to read the old data and old parity for computing the new parity, and one to write the new parity. This is referred to as a read-modify-write procedure. Because a block-interleaved, parity disk array has only one parity disk, which must be updated on all write operations, the parity disk can easily become a bottleneck. Because of this limitation, the block-interleaved distributed-parity disk array is universally preferred over the block-interleaved, parity disk array.

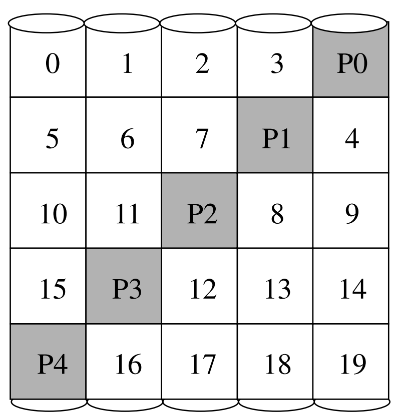
3.2.6 Block-Interleaved Distributed-Parity (RAID Level 5)

The block-interleaved distributed-parity disk array eliminates the parity disk bottleneck present in the block-interleaved parity disk array by distributing the parity uniformly over all of the disks. An additional, frequently overlooked advantage to distributing the parity is that it also distributes data over all of the disks rather than over all but one. This allows all disks to participate in servicing read operations in contrast to redundancy schemes with dedicated parity disks in which the parity disk cannot participate in servicing read requests. Block-interleaved distributed-parity disk arrays have the best small read, large read and large write performance of any redundant disk array. Small write requests are somewhat inefficient compared with redundancy schemes such as mirroring however, due to the need to perform read-modify-write operations to update parity. This is the major performance weakness of RAID level 5 disk arrays and has been the subject of intensive research [Menon93b, Stodolsky93].

The exact method used to distribute parity in block-interleaved distributed-parity disk arrays can affect performance. Figure 4 illustrates the best parity distribution of those investigated in [Lee91b], called the left-symmetric parity distribution. A useful property of the left-symmetric parity distribution is that whenever you traverse the striping units sequentially, you will access each disk once before accessing any disk twice. This property reduces disk conflicts when servicing large requests.

3.2.7 P+Q Redundancy (RAID Level 6)

Parity is a redundancy code capable of correcting any *single*, self-identifying failure. As larger disk arrays are considered, multiple failures are possible and stronger codes are needed [Burkhard93]. Moreover, when a disk fails in a parity-protected disk array, recovering the contents of the failed disk requires successfully reading the contents of all non-failed disks. As we will see in Section 3.4, the probability of encountering an uncorrectable read error during recovery can be



(Left-Symmetric)

Figure 4: RAID level 5 Left-Symmetric Parity Placement. Each square corresponds to a stripe unit. Each column of squares corresponds to a disk. P0 computes the parity over stripe units 0, 1, 2 and 3; P1 computes parity over stripe units 4, 5, 6 and 7; etc. Lee [Lee91b] shows that the left-symmetric parity distribution has the best performance. Only the minimum repeating pattern is shown.

significant. Thus, applications with more stringent reliability requirements require stronger error-correcting codes.

One such scheme, called *P+Q redundancy*, uses Reed-Solomon codes to protect against up to two disk failures using the bare minimum of two redundant disks. The P+Q redundant disk arrays are structurally very similar to the block-interleaved distributed-parity disk arrays and operate in much the same manner. In particular, P+Q redundant disk arrays also perform small write operations using a read-modify-write procedure, except that instead of four disk accesses per write requests, P+Q redundant disk arrays require six disk accesses due to the need to update both the 'P' and 'Q' information.

3.3 Performance and Cost Comparisons

The three primary metrics in the evaluation of disk arrays are reliability, performance, and cost. RAID levels 0 through 6 cover a wide range of tradeoffs between these metrics. It is important to consider all three metrics to fully understand the value and cost of each disk array organization. In this section, we compare RAID levels 0 through 6 based on performance and cost. The following section examines reliability.

3.3.1 Ground Rules and Observations

While there are only three primary metrics in the evaluation of disk arrays (reliability, performance and cost), there are many different ways to measure each metric and an even larger number of ways of using them. For example, should performance be measured in I/Os per second, bytes per second, or response time? Would a hybrid metric such as I/Os per second per dollar be more appropriate? Once a metric is agreed upon, should we compare systems at the same cost, the same total user capacity, the same performance, or the same reliability? The method one uses depends largely on the purpose of the comparison and the intended use of the system. In time-sharing applications, the primary metric may be user capacity per dollar; in transaction processing applications the primary metric may be I/Os per second per dollar; and in scientific applications, the primary metric may be bytes per second per dollar. In certain heterogeneous systems, such as file servers, both I/O per second and bytes per second may be important. In many cases, these metrics may all be conditioned on meeting a reliability threshold.

Most large secondary storage systems, and disk arrays in particular, are throughput oriented. That is, we are generally more concerned with the aggregate throughput of the system than, for example, its response time on individual requests (as long as requests are satisfied within a specified time limit). Such a bias has a sound technical basis: as techniques such as asynchronous I/O, prefetching, read caching and write buffering become more widely used, fast response time depends on sustaining a high throughput.

In throughput-oriented systems, performance can potentially increase linearly as additional components are added; if one disk provides thirty I/Os per second, two should provide sixty I/Os per second. Thus, in comparing the performance of disk arrays, we will normalize the performance of the system by its cost. In other words we will use performance metrics such as I/Os per second per dollar rather than the absolute number of I/Os per second.

Even after the metrics are agreed upon, one must decide whether to compare systems of equivalent capacity, cost or some other metric. We chose to compare systems of *equivalent file capacity* where *file capacity* is the amount of information the file system can store on the device and excludes the storage used for redundancy. Comparing systems with the same file capacity makes it easy to choose equivalent workloads for two different redundancy schemes. Were we to compare systems with different file capacities, we would be confronted with tough choices such as how a workload on a system with user capacity X maps onto a system with user capacity 2X.

Finally, there is currently much confusion in comparing RAID levels 1 through 5. The confusion arises because a RAID level sometimes specifies not a specific *implementation* of a system but rather its *configuration* and *use*. For example, a RAID level 5 disk array (block-interleaved distributed-parity) with a parity group size of two is comparable to RAID level 1 (mirroring) with the exception that in a mirrored disk array, certain disk scheduling and data layout optimizations can be performed that generally are not implemented for RAID level 5 disk arrays [Hsiao90, Orji93]. Analogously, a RAID level 5 disk array can be configured to operate equivalently to a RAID level

3 disk array by choosing a unit of data striping such that the smallest unit of array access always accesses a full parity stripe of data. In other words, RAID level 1 and RAID level 3 disk arrays can be viewed as a subclass of RAID level 5 disk arrays. Since RAID level 2 and RAID level 4 disk arrays are, practically speaking, in all ways inferior to RAID level 5 disk arrays, the problem of selecting among RAID levels 1 through 5 is a subset of the more general problem of choosing an appropriate parity group size and striping unit size for RAID level 5 disk arrays. A parity group size close to two may indicate the use of RAID level 1 disk arrays; a striping unit much smaller than the size of an average request may indicate the use of a RAID level 3 disk array.

3.3.2 Comparisons

Table 3 tabulates the maximum throughput per dollar relative to RAID level 0 for RAID levels 0, 1, 3, 5 and 6. The cost of each system is assumed to be proportional to the total number of disks in the disk array. Thus, the table illustrates that given equivalent cost RAID level 0 and RAID level 1 systems, the RAID level 1 system can sustain half the number of small writes per second that a RAID level 0 system can sustain. Equivalently, we can say that the cost of small writes is twice as expensive in a RAID level 1 system as in a RAID level 0 system. In addition to performance, the table shows the storage efficiency of each disk array organization. The storage efficiency is approximately inverse the cost of each unit of user capacity relative to a RAID level 0

	Small Read	**Small Write**	**Large Read**	**Large Write**	**Storage Efficiency**
RAID level 0	1	1	1	1	1
RAID level 1	1	1/2	1	1/2	1/2
RAID level 3	1/G	1/G	(G-1)/G	(G-1)/G	(G-1)/G
RAID level 5	1	max(1/G,1/4)	1	(G-1)/G	(G-1)/G
RAID level 6	1	max(1/G,1/6)	1	(G-2)/G	(G-2)/G

Table 3: Throughput Per Dollar Relative to RAID Level 0. This table compares the throughputs of various redundancy schemes for four types of I/O requests. Small here refers to I/O requests of one striping unit; large refers to I/O requests of one full stripe (one stripe unit from each disk in an error-correction group). G refers to the number of disks in an error-correction group. In all cases, the higher the number the better. The entries in this table account for the major performance effects but not some of the second-order effects. For instance, since RAID level 1 stores two copies of the data, a common optimization is to dynamically read the disk whose positioning time to the data is smaller.

system. For the above disk array organizations, the storage efficiency is equal to the performance/cost metric for large writes.

Figure 5 graphs the performance/cost metrics from Table 3 for RAID levels 1, 3, 5 and 6 over a range of parity group sizes. The performance/cost of RAID level 1 systems is equivalent to the performance/cost of RAID level 5 systems when the parity group size is equal to two. The perfor-

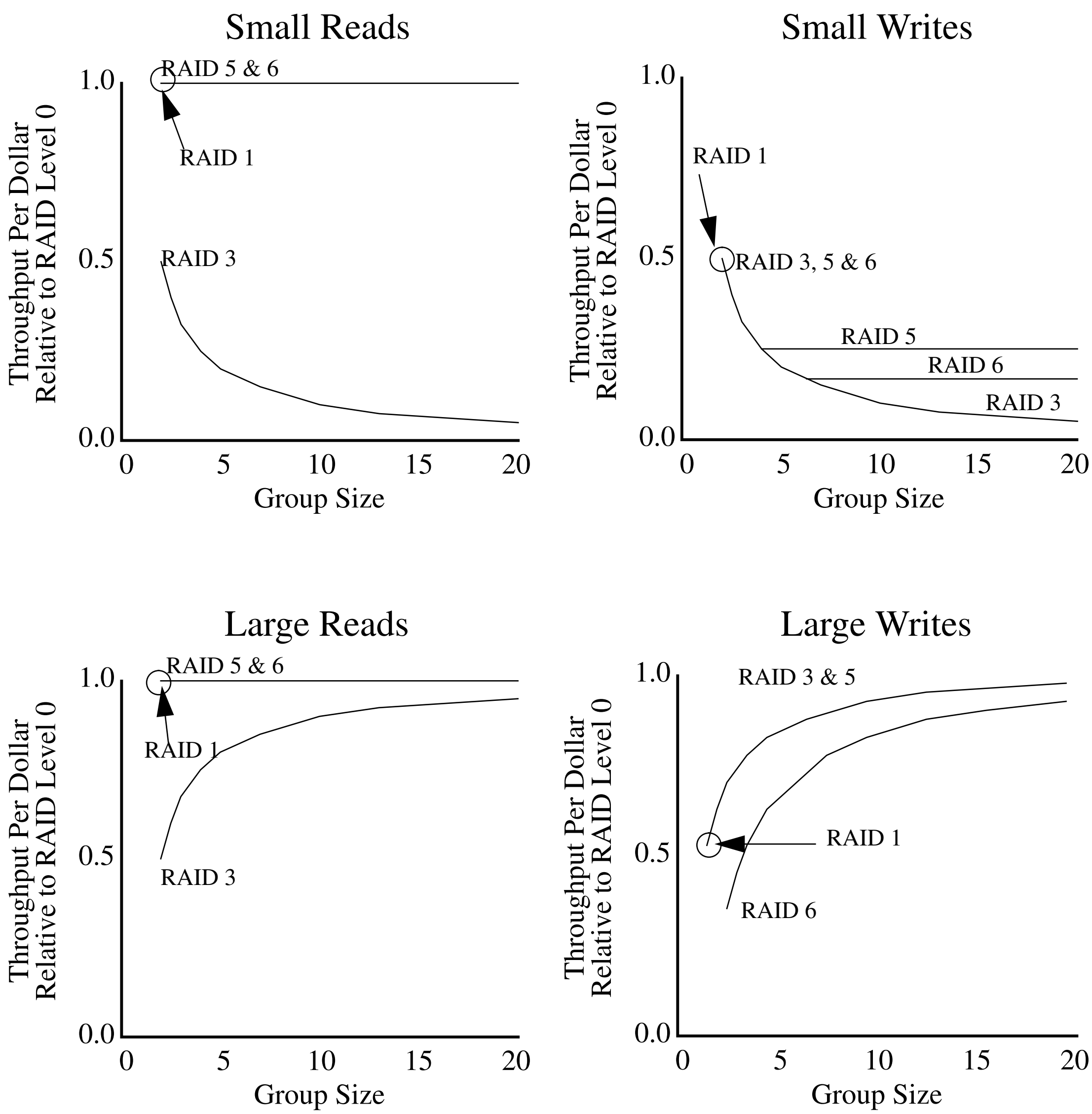


Figure 5: Throughput Per Dollar Relative to RAID Level 0. RAID level 1 performance is approximately equal to RAID level 5 performance with a group size of two. Note that for small writes, the three disk arrays are equally cost effective at small group sizes, but as group size increases, RAID levels 5 and 6 become better alternatives.

mance/cost of RAID level 3 systems is always less than or equal to the performance/cost of RAID level 5 systems. This is expected given that a RAID level 3 system is a subclass of RAID level 5 systems derived by restricting the striping unit size such that all requests access exactly a parity stripe of data. Since the configuration of RAID level 5 systems is not subject to such a restriction, the performance/cost of RAID level 5 systems can never be less than that of an equivalent RAID level 3 system. It is important to stress that these performance/cost observations apply only to the abstract models of disk arrays for which we have formulated performance/cost metrics. In reality, a specific implementation of a RAID level 3 system can have better performance/cost than a specific implementation of a RAID level 5 system.

As previously mentioned, the question of which RAID level to use is often better expressed as more general configuration questions concerning the size of the parity group and striping unit. If a parity group size of two is indicated, then mirroring is desirable. If a very small striping unit is indicated then a RAID level 3 system may be sufficient. To aid the reader in evaluating such decisions, Figure 6 plots the four performance/cost metrics from Table 3 on the same graph for each of the RAID levels 3, 5 and 6. This makes explicit the performance/cost tradeoffs in choosing an appropriate parity group size. Section 4.4 addresses how to choose the unit of striping.

3.4 Reliability

Reliability is as important a metric to many I/O systems as performance and cost, and it is perhaps the main reason for the popularity of redundant disk arrays. This section starts by reviewing the basic reliability provided by a block-interleaved parity disk array then lists three factors that can undermine the potential reliability of disk arrays.

3.4.1 Basic Reliability

Redundancy in disk arrays is motivated by the need to overcome disk failures. When only independent disk failures are considered, a simple parity scheme works admirably. Patterson, Gibson, and Katz derive the mean time between failures for a RAID level 5 to be $\frac{MTTF(disk)^2}{N \times (G-1) \times MTTR(disk)}$, where MTTF(disk) is the mean-time-to-failure of a single disk, MTTR(disk) is the mean-time-to-repair of a single disk, N is the total number of disks in the disk array, and G is the parity group size [Patterson88]. For illustration purposes, let us assume we have 100 disks that each had a mean time to failure (MTTF) of 200,000 hours and a mean time to repair of one hour. If we organized these 100 disks into parity groups of average size 16, then the mean time to failure of the system would be an astounding 3000 years! Mean times to failure of this magnitude lower the chances of failure over any given period of time.

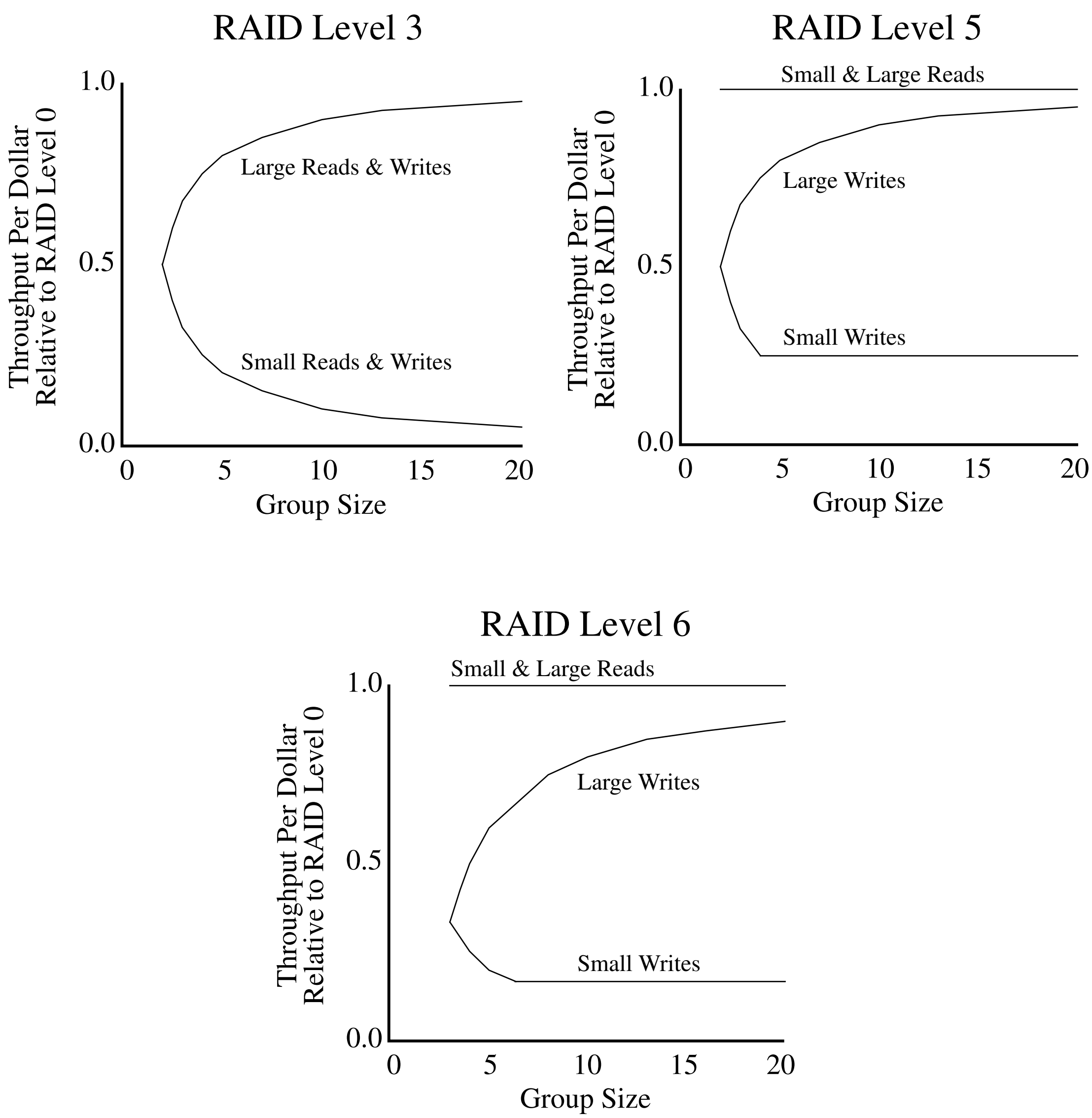


Figure 6: **Throughput Per Dollar Relative to RAID Level 0.** The graphs illustrate the tradeoff in performance/cost versus group size for each specified RAID level. Note that in this comparison, mirroring (RAID level 1) is the same as RAID level 5 with a group size of two.

For a disk array with two redundant disk per parity group, such as P+Q redundancy, the mean tim to failure is $\frac{MTTF^3(disk)}{N \times (G-1) \times (G-2) \times MTTR^2(disk)}$. Using the same values for our reliability parameters, this implies an astronomically large mean time to failure of 38 million years.

This is an idealistic picture, but it gives us an idea of the potential reliability afforded by disk arrays. The rest of this section takes a more realistic look at the reliability of block-interleaved disk arrays by considering factors such as system crashes, uncorrectable bit-errors, and correlated disk failures that can dramatically affect the reliability of disk arrays.

3.4.2 System Crashes and Parity Inconsistency

In this section, the term *system crash* refers to any event such as a power failure, operator error, hardware breakdown, or software crash that can interrupt an I/O operation to a disk array. Such crashes can interrupt write operations, resulting in states where the data is updated and the parity is not updated, or visa versa. In either case, the parity is inconsistent and cannot be used in the event of a disk failure. Techniques such as redundant hardware and power supplies can be applied to make such crashes less frequent [Menon93a], but no technique can prevent systems crashes 100% of the time.

System crashes can cause parity inconsistencies in both bit-interleaved and block-interleaved disk arrays, but the problem is of practical concern only in block-interleaved disk arrays. This is because in bit-interleaved disk arrays, the inconsistent parity can only affect the data that is currently being written. If writes do not have to be atomic, applications cannot assume either that the write during a system crash completed or did not complete, and thus it is generally permissible for the bit-interleaved disk array to store arbitrary data on the updated sectors. In a block-interleaved disk array, however, an interrupted write operation can affect the parity of other data blocks in that stripe that were not being written. Thus, for reliability purposes, system crashes in block-interleaved disk arrays are similar to disk failures in that they may result in the loss of the correct parity for stripes that were being modified during the crash.

In actuality, system crashes can be much worse than disk failures for two reasons. First, they may occur more frequently than disk failures. Second, a system crash in disk arrays using P+Q redundancy is analogous to a double disk failure because both the ‘P’ and ‘Q’ information is made inconsistent. To avoid the loss of parity on system crashes, information sufficient to recover the parity must be logged to non-volatile storage before executing each write operation. The information need only be saved until the corresponding write completes. Hardware implementations of RAID systems can efficiently implement such logging using non-volatile RAM. In software implementations that do not have access to fast non-volatile storage, it is generally not possible to protect against system crashes without significantly sacrificing performance.

3.4.3 Uncorrectable Bit-Errors

Although modern disks are highly reliable devices that can withstand significant amounts of abuse, they occasionally fail to read or write small bits of data. Currently, most disks cite uncorrectable bit error rates of one error in 10^{14} bits read. Unfortunately, the exact interpretation of what is meant by an uncorrectable bit error is unclear. For example, does the act of reading disks actually generate errors, or do the errors occur during writes and become evident during reads?

Disk manufactures generally agree that reading a disk is very unlikely to cause permanent errors. Most uncorrectable errors are generated because data is incorrectly written or gradually damaged as the magnetic media ages. These errors are detected only when we attempt to read the data. Our interpretation of uncorrectable bit error rates is that they represent the rate at which errors are detected during reads from the disk during the normal operation of the disk drive. It is important to stress that there is no generally agreed upon interpretation of bit error rates.

The primary ramification of an uncorrectable bit error is felt when a disk fails and the contents of the failed disk must be reconstructed by reading data from the non-failed disks. For example, the reconstruction of a failed disk in a 100 GB disk array requires the successful reading of approximately 200 million sectors of information. A bit error rate of one in 10^{14} bits implies that one 512 byte sector in 24 billion sectors cannot be correctly read. Thus, if we assume that the probability of reading sectors is independent of each other, the probability of reading all 200 million sectors successfully is approximately (1-1/(2.4×10^{10}))^(2.0×10^{8}) = 99.2%. This means that on average, 0.8% of disk failures would result in data loss due to an uncorrectable bit error.

The above example indicates that unrecoverable bit errors can be a significant factor in designing large, highly-reliable disk arrays. This conclusion is heavily dependent on our particular interpretation of what is meant by an unrecoverable bit error and the guaranteed unrecoverable bit error rates as supplied by the disk manufactures; actual error rates may be much better.

One approach that can be used with or without redundancy is to try to protect against bit errors by predicting when a disk is about to fail. VAXsimPLUS, a product from Digital Equipment Corporation, monitors the warnings given by disks and notifies an operator when it feels the disk is about to fail. Such predictions can significantly lower incident of data loss [Emlich89, Malhotra93].

3.4.4 Correlated Disk Failures

The simplest model of reliability of disk arrays [Patterson88] assumes that all disk failures are independent when calculating mean time to data loss. This resulted in very high mean time to data loss estimates, up to millions of years. In reality, common environmental and manufacturing factors can frequently cause correlated disk failures. For example, an earthquake might sharply increase the failure rate for all disks in a disk array for a short period of time. More commonly, power surges, power failures, and simply the act of powering disks on and off can place simultaneous stress on the electrical components of all affected disks. Disks also share common support hardware; when this hardware fails, it can lead to multiple, simultaneous disk failures.

Aside from environmental factors, the disks themselves have certain correlated failure modes built into them. For example, disks are generally more likely to fail either very early or very late in their lifetimes. Early failures are frequently caused by transient defects which may not have been detected during the manufacturer’s burn-in process; late failures occur when a disk wears out. A systematic manufacturing defect can also produce bad batches of disks that can fail close together in time. Correlated disk failures greatly reduce the reliability of disk arrays by making it much more likely that an initial disk failure will be closely followed by additional disk failures before the failed disk can be reconstructed.

3.4.5 Reliability Revisited

The previous sections have described how system crashes, uncorrectable bit errors and correlated disk failures can decrease the reliability of redundant disk arrays. In this section, we will calculate mean-time-to-data-loss statistics after incorporating these factors.

The new failure modes imply that there are now three, relatively common ways to lose data in a block-interleaved parity-protected disk array:

- double disk failure,
- system crash followed by a disk failure, and
- disk failure followed by an uncorrectable bit error during reconstruction.

Total User Capacity	100 disks (500 GB)
Disk Size	5 GB
Sector Size	512 bytes
Bit Error Rate (BER)	1 in 10^14 bits 1 in 2.4 10^10 sectors
p(disk) The probability of reading all sectors on a disk. (Derived from disk size, sector size, and BER.)	99.96%
Parity Group Size	16 disks
MTTF(disk)	200,000 hours
MTTF(disk2)	20,000 hours
MTTF(disk3)	2,000 hours
MTTR(disk)	1 hour
MTTF(sys)	1 month
MTTR(sys)	1 hour

Table 4: Reliability Parameters. This table lists parameters used for reliability calculations in this section.

As mentioned above, a system crash followed by a disk failure can be protected against in most hardware disk array implementations with the help of non-volatile storage, but such protection is unlikely in software disk arrays. The above three failure modes are the hardest failure combinations, in that we are currently unaware of any techniques to protect against them without significantly degrading performance. To construct a simple model of correlated disk failures, we will assume that each successive disk failure is ten times more likely than the previous failure (until the failed disk has been reconstructed). Table 4 tabulates values of the reliability parameters we will use for calculating numeric reliability estimates in this section. Note that the reliability estimates will be given per a constant user capacity of 100 disks, consisting of independent, 16-disk parity groups.

Table 5, which tabulates reliability metrics for RAID level 5 disk arrays, shows that the frequency of the three failure combinations are within an order of magnitude of each other. This means that none of the three failure modes can be ignored in determining reliability. This makes it difficult to improve the overall reliability of the system without improving the reliability of several

	MTTDL	MTTDL	Probability of Data Loss over 10 Year Period
Double Disk Failure	$\frac{MTTF(disk) \times MTTF(disk2)}{N \times (G-1) \times MTTR(disk)}$	285 yrs.	3.4%
Sys Crash + Disk Failure	$\frac{MTTF(sys) \times MTTF(disk)}{N \times MTTR(sys)}$	154 yrs.	6.3%
Disk Failure + Bit Error	$\frac{MTTF(disk)}{N \times (1-(p(disk))^{G-1})}$	36 yrs.	24.4%
Software RAID	(harmonic sum of above)	26 yrs.	31.6%
Hardware RAID (NVRAM)	(harmonic sum excluding sys crash+disk failure)	32 yrs.	26.8%

Table 5: Failure Characteristics for RAID Level 5 Disk Arrays. MTTDL is the mean time to data loss. The 10 year failure rate is the percent chance of data loss in a 10 year period. For numeric calculations, the parity group size, G, is equal to 16 and the user data capacity is equal to 100 data disks. Note that the total number of disks in the system, N, is equal to the number of data disks times G/(G-1).

components of the system; a more reliable disk will greatly reduce the frequency of double disk failures but its protection against the other two failure combinations is less pronounced. Frequencies of both system crashes and bit error rates must also be reduced before significant improvements in overall system reliability can be achieved. Note also the deceptively reassuring MTTDL numbers. Even with a MTTDL of 285 years, there is a 3.4% chance of losing data in the first ten years.

Table 6 tabulates the reliability metrics for P+Q redundant disk arrays. As can be seen, system crashes are the Achilles's heel of P+Q redundancy schemes. Since system crashes invalidate both the P and Q information, their effect is similar to a double disk failure. Thus, unless the system provides protection against system crashes, as is assumed in the calculation of the reliability for hardware RAID systems, P+Q redundancy does not provide a significant advantage over parity-

	MTTDL	MTTDL	Probability of Data Loss over 10 Year Period
Triple Disk Failure	$\frac{MTTF(disk) \times MTTF(disk2) \times MTTF(disk3))}{N \times (G-1) \times (G-2) \times MTTR^2(disk)}$	38052 yrs.	0.03%
Sys Crash + Disk Failure	$\frac{MTTF(sys) \times MTTF(disk)}{N \times MTTR(sys)}$	144 yrs.	7.7%
Double Disk Failure + Bit Error	$\frac{MTTF(disk) \times MTTF(disk2))}{N \times (G-1) \times (1-(1-p(disk)))^{(G-2)}) \times MTTR(disk)}$	47697 yrs.	0.02%
Software RAID	(harmonic sum of above)	143 yrs.	6.8%
Hardware RAID (NVRAM)	(harmonic sum excluding sys crash+disk failure)	21166 yrs.	0.05%

Table 6: Failure Characteristics for a P+Q disk array. MTTDL is the mean time to data loss. The 10 year failure rate is the percent chance of data loss in a 10 year period. For numeric calculations, the parity group size, G, is equal to 16 and the user data capacity is equal to 100 data disks. Note that the total number of disks in the system, N, is equal to the number of data disks times G/(G-2).

protected disk arrays. In general, P+Q redundancy is most useful for protecting against unrecoverable bit errors that occur during reconstruction and against multiple, correlated disk failures.

3.4.6 Summary and Conclusions

This section has examined the reliability of block-interleaved redundant disk arrays when factors other than independent disk failures are taken into account. We see that system crashes and unrecoverable bit errors can significantly reduce the reliability of block-interleaved, parity-protected disk arrays. We have shown that P+Q redundant disk arrays are very effective in protecting against both double disk failures and unrecoverable bit errors but are susceptible to system crashes. In order to realize the full reliability advantages of P+Q redundant disk arrays, non-volatile storage must be used to protect against system crashes.

Numeric reliability calculations serve as useful guidelines and bounds for the actual reliability of disk arrays. It is infeasible, however, to compare the reliability of real system based on such numbers. Reliability calculations frequently ignore important implementation-specific factors that are difficult to quantify such as the reliability of software components. What is useful to know, however, and what we have presented here, is the types of common failures that a disk array can tolerate, how they limit the reliability of the system, and thus its approximate reliability in comparison to other disk array organizations of similar complexity.

3.5 Implementation Considerations

Although the operation of block-interleaved redundant disk arrays is conceptually simple, a disk array implementer must address many practical considerations for the system to function correctly and reliably at an acceptable level of performance. One problem is that the necessary state information for a disk array consists of more than just the data and parity stored on the disks. Information such as which disks are failed, how much of a failed disk has been reconstructed, and which sectors are currently being updated must be accurately maintained in the face of system crashes. We will refer to such state information that is neither user data nor parity as *meta state* information. Another problem, addressed in Section 3.5.4, is that multiple disks are usually connected to the host computer via a common bus or string.

3.5.1 Avoiding Stale Data

The only piece of meta state information that *must* be maintained in redundant disk arrays is the validity of each sector of data and parity in a disk array. The following restrictions must be observed in maintaining this information.

- When a disk fails, the logical sectors corresponding to the failed disk must be marked *invalid* before any request that would normally access to the failed disk can be attempted. This invalid mark prevents users from reading corrupted data on the failed disk.

- When an invalid logical sector is reconstructed to a spare disk, the logical sector must be marked *valid* before any write request that would normally write to the failed disk can be serviced. This ensures that ensuing writes update the reconstructed data on the spare disk.

Both restrictions are needed to ensure that users do not receive stale data from the disk array. Without the first restriction, it would be possible for users to read stale data from a disk that is considered to have failed but works intermittently. Without the second restriction, successive write operations would fail to update the newly reconstructed sector, resulting in stale data. The valid/invalid state information can be maintained as a bit-vector either on a separate device or by reserving a small amount of storage on the disks currently configured into the disk array. If one keeps track of which disks are failed/operational, one only needs to keep a bit-vector for the failed disks. It is generally more convenient to maintain the valid/invalid state information on a per striping unit rather than a per sector basis since most implementations will tend to reconstruct an entire striping unit of data at a time rather than a single sector. Because disk failures are relatively rare events and large groups of striping units can be invalidated at a time, updating the valid/invalid meta state information to stable storage does not present a significant performance overhead.

3.5.2 Regenerating Parity after a System Crash

System crashes can result in inconsistent parity by interrupting write operations. Thus, unless it is known which parity sectors were being updated, all parity sectors must be regenerated whenever a disk array comes up from a system crash. This is an expensive operation that requires scanning the contents of the entire disk array. To avoid this overhead, information concerning the consistent/inconsistent state of each parity sector must be logged to stable storage. The following restriction must be observed.

- Before servicing any write request, the corresponding parity sectors must be marked *inconsistent*.
- When bringing a system up from a system crash, all inconsistent parity sectors must be regenerated.

Note that because regenerating a consistent parity sector does no harm, it is not absolutely necessary to mark a parity sector as consistent. To avoid having to regenerate a large number of parity sectors after each crash, however, it is clearly desirable to periodically mark parity sectors as consistent.

Unlike updating valid/invalid information, the updating of consistent/inconsistent state information is a potential performance problem in software RAID systems, which usually do not have access to fast, non-volatile storage. A simplistic implementation would require a disk write to mark a parity sector as inconsistent before each write operation and a corresponding disk write to mark the parity sector as consistent after each write operation. A more palatable solution is to maintain a most recently used pool that keeps track of a fixed number of inconsistent parity sectors on stable storage. By keeping a copy of the pool in main memory, one can avoid accessing stable storage to mark parity sectors that are already marked as inconsistent. By varying the size of the pool, one can tradeoff the hit-rate of the pool against the amount of parity information that needs to be regenerated when recovering from a system crash.

The above method should work efficiently for requests that exhibit good locality of reference. If the disk array must service a large number of random write requests, as in transaction processing environments, we recommend incorporating a group commit mechanism so that a large number of parity sectors can be marked inconsistent with a single access to stable storage. This solves the throughput problem but results in higher latencies for random write requests since the parity sectors must be marked inconsistent before the writes can proceed.

3.5.3 Operating with a Failed Disk

A system crash in a block-interleaved redundant disk array is similar to a disk failure in that it can result in the loss of parity information. This means that a disk array operating with a failed disk can potentially lose data in the event of a system crash. Because system crashes are significantly more common in most systems than disk failures, operating with a failed disk can be risky.

While operating with a failed disk, some form of logging must be performed on every write operation to prevent the loss of information in the event of a system crash. We describe here two elegant methods to perform this logging. The first method, called *demand reconstruction*, is the easiest and most efficient but requires stand-by spare disks. With demand reconstruction, accesses to a parity stripe with an invalid sector immediately trigger reconstruction of the appropriate data onto a spare disk. Write operations thus never deal with invalid sectors created by disk failures. A background process scans the entire disk array to ensure that all the contents of the failed disk is eventually reconstructed within an acceptable time period.

The second method, called *parity sparing* [Reddy91], can be applied to systems without stand-by spares but requires additional meta state information. Before servicing a write request that would access a parity stripe with an invalid sector, the invalid sector is reconstructed and relocated to overwrite its corresponding parity sector. The sector is then marked as relocated. Since the corresponding parity stripe no longer has parity, a system crash can only affect the data being written. When the failed disk is eventually replaced, the relocated sector is copied to the spare disk, the parity is regenerated and the sector is no longer marked as relocated.

3.5.4 Orthogonal RAID

To this point in the paper, we have ignored the issue of how to connect disks to the host computer. In fact, how one does this can drastically affect performance and reliability. Most computers connect multiple disks via some smaller number of strings. A string failure thus causes multiple, simultaneous disk failures. If not properly designed, these multiple failures can cause data to become inaccessible.

For example, consider the 16-disk array in Figure 7 and two options of how to organize multiple, error-correction groups. Option 1 combines each string of four disks into a single error-cor-

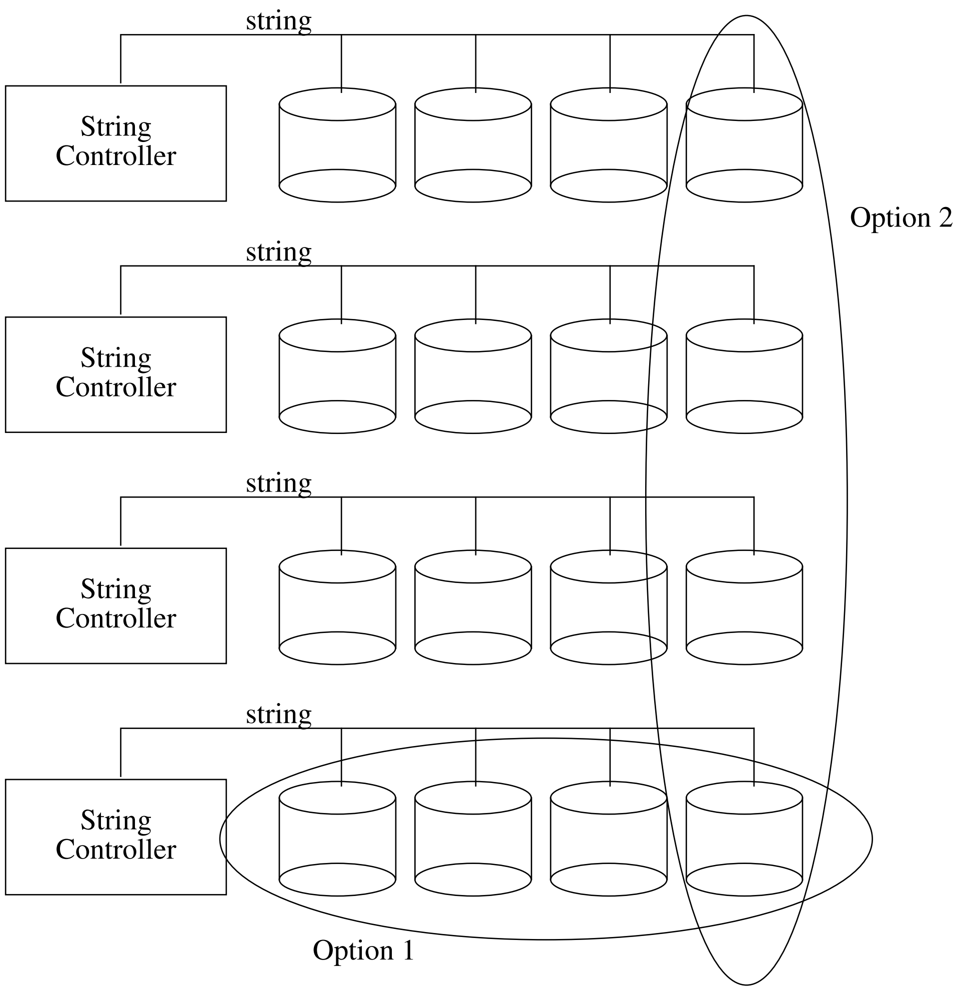


Figure 7: Orthogonal RAID. This figure present two options of how to organize error-correction groups in the presence of shared resources, such as a string controller. Option 1 groups four disks on the same string into an error-correction group; Option 2 groups one disk from each string into a group. Option 2 is preferred over Option 1 because the failure of a string controller will only render one disk from each group inaccessible.

rection group. Option 2 combines one disk on each string into a single error-correction group. Unfortunately for Option 1, if a string fails, all four disks of an error-correction group are inaccessible. Option 2, on the other hand, loses one disk from each of the four error-correction groups and still allows access to all data. This technique of organizing error-correction groups orthogonally to common hardware (such as a string) is called *orthogonal RAID* [Schulze89, Ng94]. Orthogonal RAID has the added benefit of minimizing string conflicts when multiple disks from a group transfer data simultaneously.

4 ADVANCED TOPICS

This section discusses advanced topics in the design of redundant disk arrays. Many of the techniques are independent of each other, allowing designers to mix-and-match techniques.

4.1 Improving Small Write Performance for RAID Level 5

The major performance problem with RAID level 5 disk arrays is the high overhead for small writes. As described in Section 3.2, each small write generates four separate disk I/Os, two to read the old data and old parity, and two to write the new data and new parity. This increases the response time of writes by approximately a factor of two and decreases throughput by approximately a factor of four. In contrast, mirrored disk arrays, which generate only two disk I/Os per small write, experience very little increase in response time and only a factor of two decrease in throughput. These performance penalties of RAID level 5 relative to non-redundant and mirrored disk arrays are prohibitive in applications such as transaction processing that generate many small writes.

This section describes three techniques for improving the performance of small writes in RAID level 5 disk arrays: buffering and caching, floating parity, and parity logging.

4.1.1 Buffering and Caching

Buffering and caching, two optimizations commonly used in I/O systems, can be particularly effective in disk arrays. This section describes how these optimizations can work to minimize the performance degradations of small writes in a RAID level 5.

Write buffering, also called *asynchronous writes*, acknowledges a user's write before the write goes to disk. This technique reduces the response time seen by the user under low and moderate load. Since the response time no longer depends on the disk system, RAID level 5 can deliver the same response time as any other disk system. If system crashes are a significant problem, non-volatile memory is necessary to prevent loss of data that are buffered but not yet committed. This technique may also improve throughput in two ways: 1) by giving future updates the opportunity to overwrite previous updates, thus eliminating the need to write the first update [Menon93a], and 2) by lengthening the queue of requests seen by a disk scheduler and allowing more efficient scheduling [Seltzer90].

Barring these overwrites, however, this technique does nothing to improve throughput. So under high load, the write buffer space will fill more quickly than it empties and response time of a RAID level 5 will still be four times worse than a RAID level 0.

An extension of write buffering is to group sequential writes together. This technique can make writes to all types of disk systems faster, but it has a particular appeal to RAID level 5 disk arrays. By writing larger units, small writes can be turned into full stripe writes, thus eliminating altogether the Achilles heel of RAID level 5 workloads [Rosenblum91, Menon93a].

Read caching is normally used in disk systems to improve the response time and throughput when reading data. In a RAID level 5 disk array, however, it can serve a secondary purpose. If the old data required for computing the new parity is in the cache, read caching reduces the number of disk accesses required for small writes from four to three. This is very likely, for example, in transaction processing systems where records are frequently updated by reading the old value, changing it and writing back the new value to the same location.

By also caching recently written parity, the read of the old parity can sometimes be eliminated, further reducing the number of disk accesses for small writes from three to two. Because parity is computed over many logically-consecutive disk sectors, the caching of parity exploits both temporal and spatial locality. This is in contrast to the caching of data which, for the purposes of reducing disk operations on small writes, relies on the assumption that recently read sectors are likely to be written rather than on the principle of spatial locality. Of course, caching parity blocks reduces the space available for caching data, which may increase the number of data misses.

4.1.2 Floating Parity

Menon and Kasson proposed a variation on the organization of parity in RAID level 5 disk array, called *floating parity,* that shortens the read-modify-write of parity updated by small, random writes to little more than a single disk access time on average [Menon93b]. Floating parity clusters parity into cylinders, each containing a track of free blocks. Whenever a parity block needs to be updated, the new parity block can be written on the rotationally-nearest unallocated block following the old parity block. Menon and Kasson show that for disks with 16 tracks per cylinder, the nearest unallocated block immediately follows the parity block being read 65% of the time, and the average number of blocks that must be skipped to get to the nearest unallocated block is small, between 0.7 and 0.8. Thus, the writing of the new parity block can usually occur immediately after the old parity block is read, making the entire read-modify-write access only about a millisecond longer than a read access.

To efficiently implement floating parity, directories for the locations of unallocated blocks and parity blocks must be stored in primary memory. These tables are about 1 MB in size for each disk array containing four to ten, 500 MB disks. To exploit unallocated blocks immediately following the parity data being read, the data must be modified and a disk head switched to the track containing the unallocated block before the disk rotates though an inter-sector gap. Because of these constraints, and because only a disk controller can have exact knowledge of it's geometry, floating parity is most likely to be implemented in the disk controller.

Menon and Kasson also propose floating data as well as parity. This makes the overhead for small writes in RAID level 5 disk arrays comparable to mirroring. The main disadvantage of floating data is that logically sequential data may end up discontiguous on disk. Floating data also requires much more free disk space than floating only the parity since there are many more data blocks than parity blocks.

4.1.3 Parity Logging

Stodolsky and Gibson propose an approach called *parity logging* to reduce the penalty of small writes in RAID level 5 disk arrays [Stodolsky93, Bhide92]. Parity logging reduces the overhead for small writes by delaying the read of the old parity and the write of the new parity. Instead of immediately updating the parity, an *update image*, which is the difference between the old and new parity, is temporarily written to a log. Delaying the update allows the parity to be grouped together in large contiguous blocks that can be updated more efficiently.

This delay takes place in two parts. First, the parity update image is stored temporarily in non-volatile memory. When this memory, which could be a few tens of KB, fills up, the parity update image is written to a log region on disk. When the log fills up, the parity update image is read into memory and added to the old parity. The resulting new parity is then written to disk. Although this scheme transfers more data to and from disk, the transfers are in much larger units and are hence more efficient; large sequential disk accesses are an order of magnitude more efficient than small random accesses (Section 2.1). Parity logging reduces the small write overhead from four disk accesses to a little more than two disk accesses, the same overhead incurred by mirrored disk arrays. The costs of parity logging are the memory used for temporarily storing update images, the extra disk space used for the log of update images, and the additional memory used when applying the parity update image to the old parity. This technique can also be applied to the second copy of data in mirrored disk arrays to reduce the cost of writes in mirrored disk arrays from two to a little more than one disk access [Orji93].

4.2 Declustered Parity

Many applications, notably database and transaction processing, require both high throughput and high data availability from their storage systems. The most demanding of these applications requires continuous operation—the ability to satisfy requests for data in the presence of disk failures while simultaneously reconstructing the contents of failed disks onto replacement disks. It is unacceptable to fulfill this requirement with arbitrarily degraded performance, especially in long-lived real-time applications such as video service; customers are unlikely to tolerate movies played at a slower speed or having their viewing terminated prematurely.

Unfortunately, disk failures cause large performance degradations in standard RAID level 5 disk arrays. In the worst case, a workload consisting entirely of small reads will double the effective load at non-failed disks due to extra disk accesses needed to reconstruct data for reads to the failed disk. The additional disk accesses needed to completely reconstruct the failed disk increase the load even further.

In storage systems that stripe data across several RAIDs, the *average* increase in load is significantly less than in RAIDs with one large parity group, but the RAID with the failed disk still experiences a 100% increase in load in the worst case. The failed RAID creates a hot spot that degrades the performance of the entire system. The basic problem in these large systems is that although inter-RAID striping distributes load uniformly when no disk is failed, it non-uniformly distributes the increased load that results from a failed disk; the small set of disks in the same parity group as the failed disk bear the entire weight of the increased load. The *declustered parity* RAID organization solves this problem by uniformly distributing the increased load over all disks [Muntz90, Merchant92, Holland92, Holland93, Ng92].

Figure 8 illustrates examples of standard and declustered parity RAIDs for systems with an array size of eight disks and a parity group size of four. In this case, a multiple RAID system is constructed by striping data over two RAIDs of four disks each with non-overlapping parity groups. The declustered parity RAID is constructed by overlapping parity groups. If Disk 2 fails, each read to Disk 2 in the standard, multiple RAID generates a single disk access to Disks 0, 1 and

3 and no disk access to Disks 4, 5, 6 and 7. In the declustered parity RAID, a random read to Disk 2 generates an access to Disks 4, 5, and 7 one-quarter of the time; to Disks 0, 1 and 3 half of the time; and to disk 6 three-quarters of the time. Although the increased load is not uniform, it is more balanced than in the standard RAID. Slightly more complex declustered parity RAIDs exist that uniformly distribute the load such that each read to disk 2 generates an average of 3/7 disk accesses to all non-failed disks.

The simplest way to create a declustered parity RAID that uniformly distributes load is to create a set of parity groups including every possible mapping of parity group members to disks. In our example, this would result in $\binom{8}{4} = 70$ distinct mappings of parity groups to disks. For nearly all practical array and parity group sizes, declustered parity RAID organizations are possible that

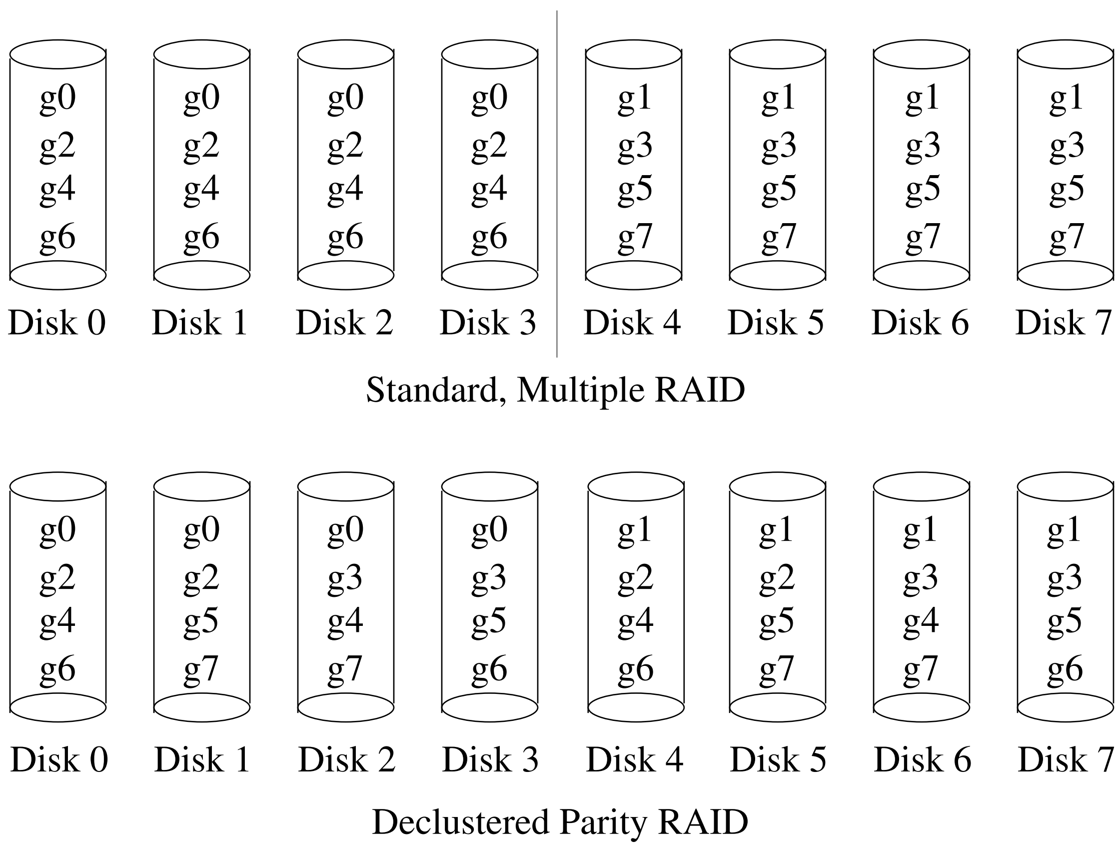


Figure 8: Standard Versus Declustered Parity RAID. This figure illustrates examples of standard and declustered parity RAID with eight disks and a parity group size of four. Identically labeled blocks belong to the same parity group. In the standard RAID organization, parity groups are composed of disks from one of two non-overlapping subsets of disks. In the declustered parity RAID, parity groups span many overlapping subsets of disks.

uniformly distribute reconstruction load with much fewer than the combinatorial number of parity groups. Such organizations can be devised using the theory of *balanced incomplete block designs* [Hall86]. In practice, the load does not need to be absolutely balanced and a close approximation is sufficient.

To summarize, a declustered parity RAID is often preferable to a standard, multiple RAID because it uniformly distributes load during both the normal and failed modes of operation. This allows a more graceful degradation in performance when a disk fails and allows the failed disk to be reconstructed more quickly since all disks in the disk array can participate in its reconstruction. In addition, unlike the example in Figure 8, the disk array size in a declustered parity RAID does not have to be a multiple of the parity group size. Any combination of array and parity group sizes such that the array size is greater than the parity group size is feasible. Declustered parity RAID has two main disadvantages. First, it can be somewhat less reliable than standard, multiple RAID; any two disk failures will result in data loss since each pair of disks has a parity group in common. In a standard, multiple RAID, the parity groups are disjoint, so it is possible to have more than one disk failure without losing data as long as each failure is in a different parity group. Second, the more complex parity groups could disrupt the sequential placement of data across the disks. Large requests are thus more likely to encounter disk contention in declustered parity RAID than in standard multiple RAID. In practice, it is difficult to construct workloads where this effect is significant.

4.3 Exploiting On-Line Spare Disks

On-line spare disks allow reconstruction of failed disks to start immediately, reducing the window of vulnerability during which an additional disk failure would result in data loss. Unfortunately, they are idle most of time and do not contribute to the normal operation of the system. This section describes two techniques, *distributed sparing* and *parity sparing*, that exploit on-line spare disks to enhance performance during the normal operation of the system.

As Figure 9 illustrates, distributed sparing distributes the capacity of a spare disk across all the disks in the disk array [Menon91]. The distribution of spare capacity is similar to the distribution of parity in RAID level 5 disk arrays. Instead of N data and one spare disk, distributed sparing uses N+1 data disks that each have 1/(N+1)th spare capacity. When a disk fails, the blocks on the failed disk are reconstructed onto the corresponding spare blocks. Distributed sparing obviates dedicated spare disks, allowing all disks to participate in servicing requests, and thereby improving performance during the normal operation of the disk array. Additionally, because each disk is partially empty, each disk failure requires less work to reconstruct the contents of the failed disk. Distributed sparing has a few disadvantages. First, the reconstructed data must eventually be copied onto a permanent replacement for the failed disk. This creates extra work for the disk array, but, since the copying can be done leisurely, it does not significantly affect performance. Second, because the reconstructed data is distributed across many disk whereas it was formerly on a single disk, reconstruction disturbs the original data placement, which can be a concern for some I/O-intensive applications. In disk arrays with dedicated spares, the data placement after reconstruction is identical to the data placement before reconstruction.

Parity sparing is similar to distributed sparing except that it uses the spare capacity to store parity information [Reddy91, Chandy93]. As with distributed sparing, this eliminates dedicated spare disks, improving performance during normal operation. The second set of parity blocks can be used in a variety of ways. First, they can be used to logically partition the disk array into two separate disk arrays, resulting in higher reliability. In Figure 10, for example, P0a might compute the parity over blocks 1 and 2 while P0b computes the parity over blocks 3 and 4. Second, the additional parity blocks can be used to augment the original parity groups. In Figure 10, if one

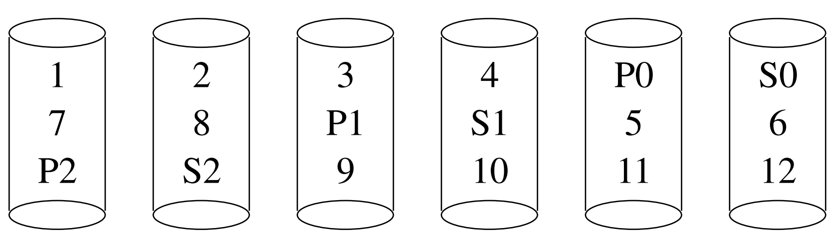


Figure 9: Distributed Sparing. Distributed sparing distributes the capacity of the spare disk throughput the array. This allows all disks, including the disk that would otherwise have been a dedicated spare, to service requests. This figure illustrates a RAID level 5 disk array with distributed sparing. The ‘P’s denote parity blocks and ‘S’s denote spare blocks.

assumes that the parity of blocks 1, 2, 3, 4, P0a and P0b is always zero, write operations need update only one of P0a *or* P0b. This has the benefit of improving small write performance by allowing each small write to choose the parity block it will update based on information such as the queue length and disk arm position at the two alternative disks. Third, the extra parity blocks can be used to implement P+Q redundancy. When a disk fails, the disk array converts to simple parity. By logical extension, a second disk failure would result in a RAID level 0 disk array.

Both distributed sparing and parity sparing offer interesting ways to exploit on-line spares for improved performance. They are most effective for disk arrays with a small number of disks where the fraction of spare disks to non-spare disks is likely to be large. As disk arrays become larger, a smaller fraction of spare disks is needed to achieve the same level of reliability [Gibson91].

4.4 Data Striping in Disk Arrays

Distributing data across the disk array speeds up I/Os by allowing a single I/O to transfer data in parallel from multiple disks or by allowing multiple I/Os to occur in parallel. The disk array designer must keep in mind several tradeoffs when deciding how to distribute data over the disks in the disk array to maximize performance, balancing two conflicting goals:

- Maximize the amount of useful data that each disk transfers with each logical I/O. Typically, a disk must spend some time seeking and rotating between each logical I/O that it services. This positioning time represents wasted work—no data is transferred during this time. It is hence beneficial to maximize the amount of useful work done in between these positioning times.

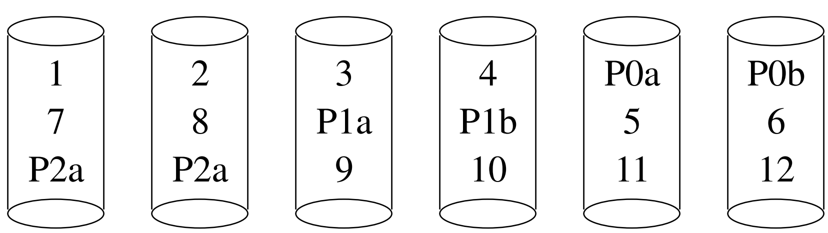


Figure 10: Parity Sparing. Parity sparing is similar to distributed sparing except that the spare space is used to store a second set of parity information.

- Utilize all disks. Idle times are similar to positioning times in that during idle times, no useful work is done. Idle times can arise in two different situations. First, hot spots can exist, where certain disks (the hot disks) are more heavily used than other disks (the cold disks) [Friedman83, Wilmot89]. Second, it is possible that all disks could be used evenly when viewed over a long period of time but not evenly at every instant. For example, if there is only one request to the disk array and that request only uses one disk, then all other disks will remain idle.

These goals are in conflict because the schemes that guarantee use of all disks spread data widely among more disks and hence cause each disk to transfer less data per logical I/O. On the other hand, schemes that maximize the amount of data transferred per logical I/O may leave some disks idle. Finding the right balance between these two goals is the main tradeoff in deciding how to distribute data among multiple disks and is heavily workload dependent.

Data striping, or interleaving, is the most common way to distribute data among multiple disks. In this scheme, logically contiguous pieces of data are stored on each disk in turn. We refer to the size of each piece of data as the striping unit. The main design parameter in data striping is the size of this striping unit. Smaller striping units cause logical data to be spread over more disks; larger striping units cause logical data to be grouped, or clustered, together on fewer disks. Consequently, the size of the striping unit determines how many disks each logical I/O uses.

Because the interaction between workload and striping unit can have a substantial effect on the performance of a disk array with block-interleaved striping, Chen and Patterson developed rules of thumb for selecting a striping unit [Chen90b]. Their simulation-based model evaluated a spindle-synchronized disk array of 16 disks. The stochastic workload applied to the disk array had two main parameters: average request size (varied from 4-1500 KB) and the number of concurrent, independent logical requests (varied from 1-20). Their goal was to find the size of a striping unit that gave the largest throughput for an incompletely specified workload. They found that the most important workload parameter was concurrency. When the concurrency of the workload was known, a simple formula specified a striping unit that provided 95% of the maximum throughput possible for any particular request distribution:

1 sector + 1/4 * average positioning time * data transfer rate * (concurrency-1)

where the average positioning time is the disk's average seek time for the workload plus an average rotational delay. A striping unit selected by this expression is small when the concurrency is low so that every access can utilize all disks, and larger when the concurrency is high so that more different accesses can be serviced in parallel. Intuitively, the product of average positioning time and data transfer rate balances the benefits and the costs of striping data. The benefit of striping is the decreased transfer time of a single request, which saves approximately the transfer time of a stripe unit. The cost of striping is the increased disk utilization that arises from an additional disk positioning itself to access the data. The constant, 1/4, is sensitive to the number of disks in the array [Chen93].

If nothing is known about a workload's concurrency, Chen and Patterson found that a good compromise size for a striping unit is

2/3 * average positioning time * data transfer rate

The constant, 2/3, is sensitive to the number of disks in the array; research needs to be done quantifying this relationship.

Lee and Katz [Lee91a] use an analytic model of non-redundant disk arrays to derive an equation for the optimal size of data striping. The disk array system they model is similar to that used by Chen and Patterson [Chen90b] described above. They show that the optimal size of data striping is equal to $\sqrt{\frac{PX(L-1)Z}{N}}$ where P is the average disk positioning time, X is the average disk transfer rate, L is the concurrency, Z is the request size, and N is the array size in disks. Their results agree closely with those of Chen and Patterson. In particular, note that their equation also predicts that the optimal size of data striping is dependent only the relative rates at which a disk positions and transfers data, PX, rather than P or X individually. Lee and Katz show that the optimal striping unit depends on request size; Chen and Patterson show that this dependency can be ignored without significantly affecting performance.

Chen and Lee [Chen93] conducted a follow-up study to [Chen90b] to determine the striping unit for RAID Level 5 disk arrays. Reads in a RAID Level 5 are similar to reads (and writes) in a RAID Level 0, causing the optimal striping unit for a read-intensive workload in a RAID Level 5 to be identical to the optimal striping unit in a RAID Level 0. For write-intensive workloads, however, the overhead of maintaining parity causes full-stripe writes (writes that span the entire parity group) to be more efficient than read-modify writes or reconstruct writes (writes that do not span an entire parity group). This additional factor causes the optimal striping unit for RAID Level 5 to be smaller for write-intensive workloads than the striping unit for RAID Level 0 by a factor of 4 for a 16-disk array. They also explored the relationship between the optimal striping unit and the number of disks and found that the optimal striping unit for reads varies *inversely* to the number of disks, but that the optimal striping unit for writes varies *with* the number of disks. Overall, they found that the optimal striping unit for workloads with an unspecified mix of reads and writes was *independent* of the number of disks and recommended (in the absence of specific workload information) that the striping unit for RAID Level 5 disk arrays with any number of disks be set to

1/2 * average positioning time * data transfer rate

Researchers are currently investigating ways to distribute data other than a simple round-robin scheme. Some variations are choosing a different striping unit for each file and distributing data by hashing or heat-balancing [Weikum92, Scheuermann91, Copeland88].

4.5 Performance and Reliability Modeling

This section presents a brief summary of work that has been done in modeling the performance and reliability of disk arrays. General performance models for block-interleaved disk arrays are very difficult to formulate due to the presence of queueing and fork-join synchronization. That is, a disk array request consists of multiple component disk requests that must be queued and serviced independently, then joined together to satisfy the disk array request. Currently, exact solutions exist for certain two server fork-join queues, however, the general k server fork-join queue is an open research problem. In addition, the bursty nature of most real I/O workloads is difficult to model using existing performance models, which generally deal only with the steady state behavior of the system. Thus, most performance models of block-interleaved disk arrays place heavy restrictions on the configuration of the disk array or the types of workloads that can be modeled. So far, a satisfactory performance model for RAID level 5 disk arrays that models both reads and writes over a wide range of system and workload parameters has yet to be formulated.

Kim [Kim86] derives response time equations for synchronous byte-interleaved disk arrays by treating the entire disk array as an M/G/1 queueing system. That is, the entire disk array is modeled as an open queueing system with an exponential interarrival distribution, general service time distribution, and a single server consisting of all the disks in the disk array. The study compares the performance of an n disk synchronous byte-interleaved disk array with n independent disk with uniform load and n independent disks with skewed load. She concludes that byte interleaving results in reduced transfer time due to increased parallelism in servicing requests and better load balancing but dramatically reduces the number of requests that can be serviced concurrently.

Kim and Tantawi [Kim91], derive approximate service time equations for asynchronous (disks rotate independently of one another), byte-interleaved disk arrays. Disk seeks are assumed to be distributed exponentially and rotational latencies are assumed to be distributed uniformly. The results of the analytic equations are compared with the results of both synthetic and trace-driven simulations. An important conclusion of the paper is that for a wide range of seek time distributions, the sum of the seek and rotational latency can be approximated by a normal distribution.

Chen and Towsley [Chen91] analytically model RAID level 1 and RAID level 5 disk arrays for the purpose of comparing their performance under workloads consisting of very small and large requests. Bounds are used to approximately model the queueing and fork-join synchronization in RAID level 1 disk arrays. Small write requests in RAID level 5 disk arrays are handled by ignoring the fork-join synchronization overhead, resulting in a somewhat optimistic model. Large requests are modeled by using a single queue for all the disks in the disk array. The results of the model are compared against simulation.

Lee and Katz [Lee93, Lee91a] derive approximate throughput and response time equations for block-interleaved disk arrays. Their model is the first analytic performance model for general block-interleaved disk arrays that takes into account both queueing and fork-join synchronization. Previous models have ignored either the queuing or fork-join synchronization component of the system. Lee and Katz[Lee91a] also provide a simple application of the analytic model to determine an equation for the optimal unit of data striping in disk arrays.

In addition to analytic models specifically for disk arrays, work dealing with the modeling of fork-join queueing systems in general [Baccelli85, Flatto84, Heidelberger82, Nelson88] is useful in modeling disk arrays. Most of these papers, however, model highly restrictive systems that are not easily applied to disk arrays.

The reliability of disk arrays is most frequently modeled using continuous time Markov chains. The failure and recovery of components in the system cause transitions from one state to another. Generally, the most useful information derived from such models is the average time to system failure and the equilibrium state probabilities from which one can determine the fraction of failures caused by each type of failure mode. A disadvantage of Markov reliability models is that the number of states necessary to model even simple disk arrays increases exponentially as new failure modes and system components are introduced. Fortunately, because the repair/replacement rates for components of most disk arrays are much higher than the failure rates, it is usually possible to greatly simply the Markov models by eliminating states that very rarely occur. To date, Gibson [Gibson91] presents the most complete reliability study of disk arrays.

5 CASE STUDIES

Since the first publication of the RAID taxonomy in 1987, the disk drive industry has been galvanized by the RAID concept. At least one market survey, prepared by Montgomery Securities in 1991 [Mon91], (optimistically) predicted that the disk array market would reach $7.8 billion by 1994. Companies either shipping or having announced disk array products include: Array Technology Corporation (a subsidiary of Tandem), Ciprico, Compaq, Data General, Dell, EMC Corporation, Hewlett-Packard, IBM, MasPar, Maximum Strategies, Microtechnologies Corporation, Micropolis, NCR, StorageTek, and Thinking Machines. RAID technology has found application in all major computer system segments, including supercomputing, mainframes, minicomputers, workstation file servers, and PC file servers. We highlight some of these systems in the following subsections.

5.1 Thinking Machines Corporation ScaleArray

The TMC ScaleArray is a RAID level 3 for the CM-5, which is a massively parallel processor (MPP) from Thinking Machines Corporation (TMC). Announced in 1992, this disk array is designed for scientific applications characterized by high-bandwidth for large files. Thinking Machines also provides a file system that can deliver data from a single file to multiple processors from multiple disks [LoVerso93].

The base unit consists of eight IBM Model 0663E15 disks. These 3.5 inch disks contain 1.2 GB of data and can transfer up to 2 MB/second for reads and 1.8 MB/second for writes. A pair of disks is attached to each of four SCSI-2 strings, and these four strings are attached to an 8 MB disk buffer. Three of these base units are attached to the backplane, so the minimum configuration is 24 disks. TMC expects the 24 disks to be allocated as 22 data disks, 1 parity disk, and one spare, but these ratios are adjustable.

Perhaps the most interesting feature of the ScaleArray is that these base units are connected directly to the data routing network of the CM-5. Massively-parallel processors normally reserve that network to send messages between processors, but TMC decided to use the same network to give them a scalable amount of disk I/O in addition to a scalable amount of processing. Each network link offers 20 MB/second, and there is a network link for each base unit. As a consequence of communicating with the data network and the small message size of the CM-5, the interleaving factor is only 16 bytes. Parity is calculated by an on-board processor and sent to the appropriate disk.

Using the scalable MPP network to connect disks means there is almost no practical limit to the number of disks that can be attached to the CM-5, since the machine was designed to be able to scale to over 16,000 nodes. At the time of announcement, TMC had tested systems with 120 disks.

Using their file system and 120 disks (including a single parity disk), TMC was able to demonstrate up to 185 MB/second for reads and up to 135 MB/second for writes for 240 MB files. In another test, TMC demonstrated 1.5 to 1.6 MB/second per disk for reads and 1.0 to 1.1 MB/second per disk for writes as the number of disks scaled from 20 to 120. For this test, TMC sent 2 MB to each disk from a large file.

5.2 StorageTek Iceberg 9200 Disk Array Subsystem

StorageTek undertook the development of disk array-based mainframe storage products in the late 1980s. Their array, called *Iceberg*, is based on collections of 5.25” disk drives yet appears to the mainframe (and its IBM-written operating system) as more traditional IBM 3380 and 3390 disk drives. Iceberg implements an extended RAID level 5 and 6 disk array. An array consists of 13 data drives, P and Q drives, and a hot spare. Data, parity, and Reed-Solomon coding are striped across the 15 active drives within the array. A single Iceberg controller can manage up to four such arrays, totalling 150 GB of storage.

Iceberg incorporates a number of innovative capabilities within its array controller, called *Penguin*. The controller itself is organized as an 8 processor system and executes its own real-time operating system. The controller can simultaneously execute eight channel programs and can independently transfer on four additional channels.

The controller manages a large, battery-backed semiconductor cache (from 64 MB up to 512 MB) in front of the disk array. This “extra level of indirection” makes possible several array optimizations. First, the cache is used as a staging area for compressing and decompressing data to and from disk. This compression can double the effective storage capacity of the disk array. Second, when written data is replaced in the cache, it is not written back to the same place on disk. In a manner much like Berkeley’s Log Structured File System [Rosenblum91], data is written opportunistically to disk in large track-sized transfer units, reducing random access latencies and performing adaptive load balancing. And third, the cache makes it possible to translate between the variable-length sectors used by most IBM mainframe applications and the fixed-size sectors of commodity small disk drives. StorageTek calls this process *dynamic mapping*. The controller keeps track of free space within the array and must reclaim space that is no longer being used. The free space data structures and track tables mapping between logical IBM 3380 and 3390 disks and the actual physical blocks within the array is maintained in a separate, 8 MB, non-volatile controller memory.

Due to the complexity of the software for a system as ambitious as Iceberg, the product is over a year behind schedule, though at the time of this writing it is in beta test.

5.3 NCR 6298

The NCR 6298 Disk Array Subsystem, released in 1992, is a low cost RAID subsystem supporting RAID levels 0, 1, 3 and 5. Designed for commercial environments, the system supports up to four controllers, redundant power supplies and fans, and up to 20 3.5” SCSI-2 drives. All components—power supplies, drives, and controllers—can be replaced while the system services requests. Though the system does not allow on-line spares, built-in diagnostics notify the host when a drive has failed, and reconstruction occurs automatically when a replacement drive is inserted.

The array controller architecture features a unique lock-step design (Figure 11) that requires almost no buffering. For all requests except RAID level 5 writes, data flows directly through the controller to the drives. The controller duplexes the data stream for mirroring configurations and generates parity for RAID level 3 synchronously with data transfer. On RAID level 3 reads, the system can optionally read the parity along with the data, proving an additional check of data integrity. This lock-step nature also means that RAID level 3 performance does not degrade when a single drive fails.

The RAID level 5 implementation does not support full-stripe writes. Instead, the write path uses an intermediate SRAM buffer. When a write occurs, the old data and parity are read (in lock-step) from disk, exclusive-or’ed together, and stored into a 64K SRAM parity buffer. As a side effect of data transfer from the host, the contents of the parity buffer are exclusive-or’ed with the data to generate the up-to-date parity and the parity is written to the parity drive. While this design prohibits the overlap of data transfer for RAID level 5, the controller overlaps the drive positioning operations. This parsimonious use of buffer, in contrast with architectures such as RAID-II, lowers the cost of the controller.

The lock-step data path is also used for reconstruction. Data and parity are read synchronously from the surviving drives, exclusive-or'ed together, and written to the replacement drive. Reconstruction is therefore quite fast, approaching the minimum time of writing a single drive.

The host interface is fast, wide, differential SCSI-2 (20 MB/S), while the drive channels are fast, narrow SCSI-2 (10 MB/S). Because of the lock-step architecture, transfer bandwidth to the host is limited to 10 MB/S for RAID level 0, 1 and 5. However, in RAID level 3 configurations,

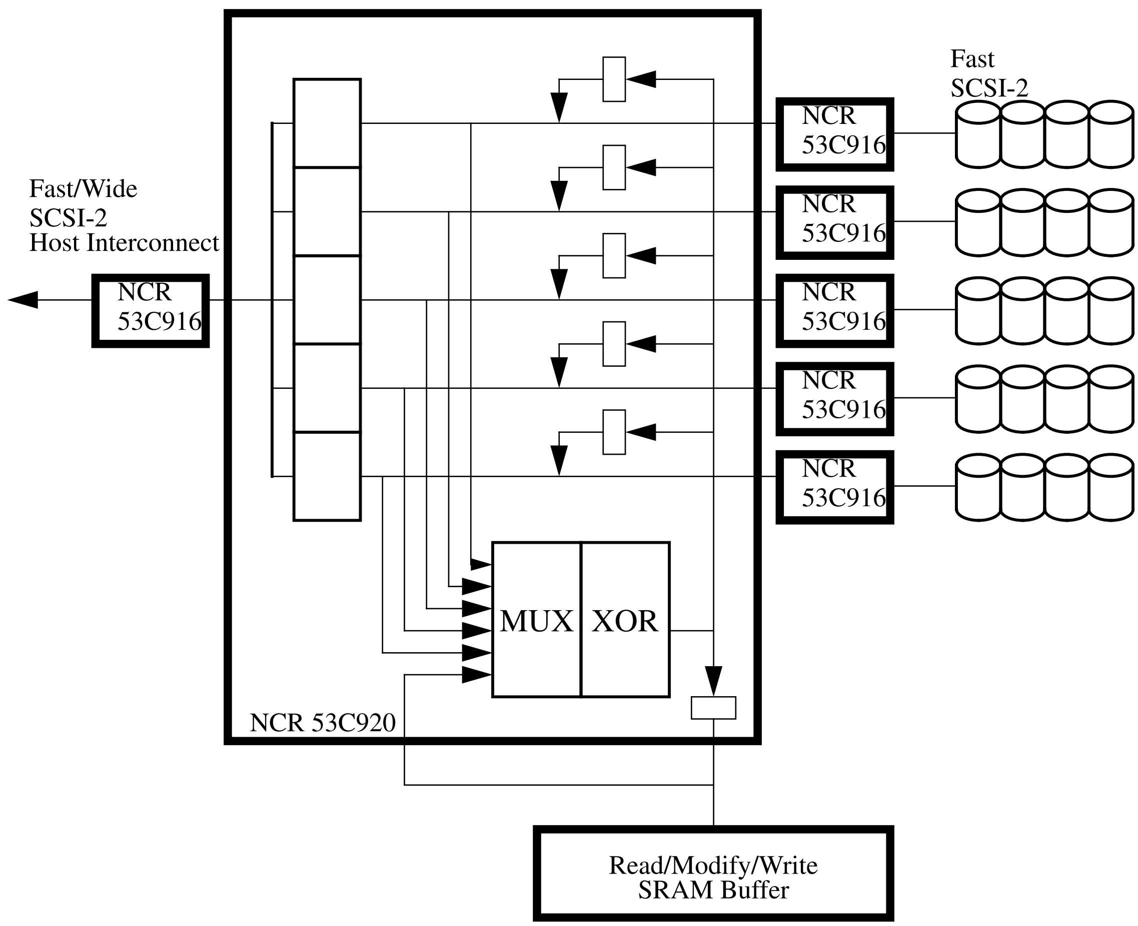


Figure 11: NCR 6298 Controller Datapath. The lock-step datapath of the 6298 requires no memory for any operations except RAID level 5 writes. By placing the XOR and MUX directly in the data path, the controller can generate parity or reconstruct data on the fly.

performance on large transfers has been measured at over 14 MB/S (limited by the host's memory system).

5.4 TickerTAIP/DataMesh

TickerTAIP/DataMesh is a research project at Hewlett-Packard Labs whose goal is to develop an array of "smart" disk nodes linked by a fast, reliable network [Cao93] (Figure 12). Each node contains a disk, a CPU, and some local memory. Disk array controller operations such as parity computation are distributed among these smart disk nodes, and the nodes communicate by message-passing across the internal interconnect.

A unique feature of the TickerTAIP architecture is the close association of a CPU to each disk drive in the array (Figure 12). This association allows each node to perform some of the processing needed to perform a disk array operation. In addition, a subset of nodes are connected to the host computers that are requesting data. Because more than one node can talk to the host computers, TickerTAIP can survive a number of node failures. Many other disk arrays, in contrast, have only one connection to host computers and hence cannot survive the failure of their disk array controller.

Currently, TickerTAIP exists as a small, 7-node prototype. Each node consists of a T800 transputer, 4 MB of local RAM, and one HP79560 SCSI disk drive. The TickerTAIP project is

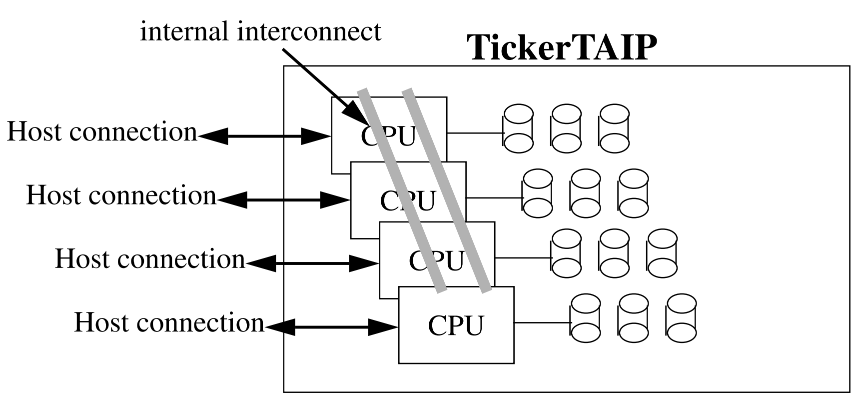


Figure 12: The TickerTAIP/DataMesh Hardware Architecture. A unique feature of the TickerTAIP architecture is the close association of a CPU to each disk drive in the array. This association allows each node to perform some of the processing needed to perform a disk array operation.

now developing software to make the multiple, distributed processing nodes appear as a single, fast storage server. Early results show that, at least for computing parity, TickerTAIP achieves near linear scaling [Cao93].

5.5 The RAID-II Storage Server

RAID-II (Figure 13) is a high-bandwidth, network file server designed and implemented at the University of California at Berkeley as part of a project to study high-performance, large-capacity, highly-reliable storage systems [Chen94, Drapeau94, Katz93]. RAID-II interfaces a SCSI-based disk array to a HIPPI network. One of RAID-II's unique features is its ability to provide high-bandwidth access from the network to the disks without transferring data through the relatively slow file server (a Sun4/280 workstation) memory system. To do this, the RAID project designed a custom printed-circuit board called the *XBUS car*d.

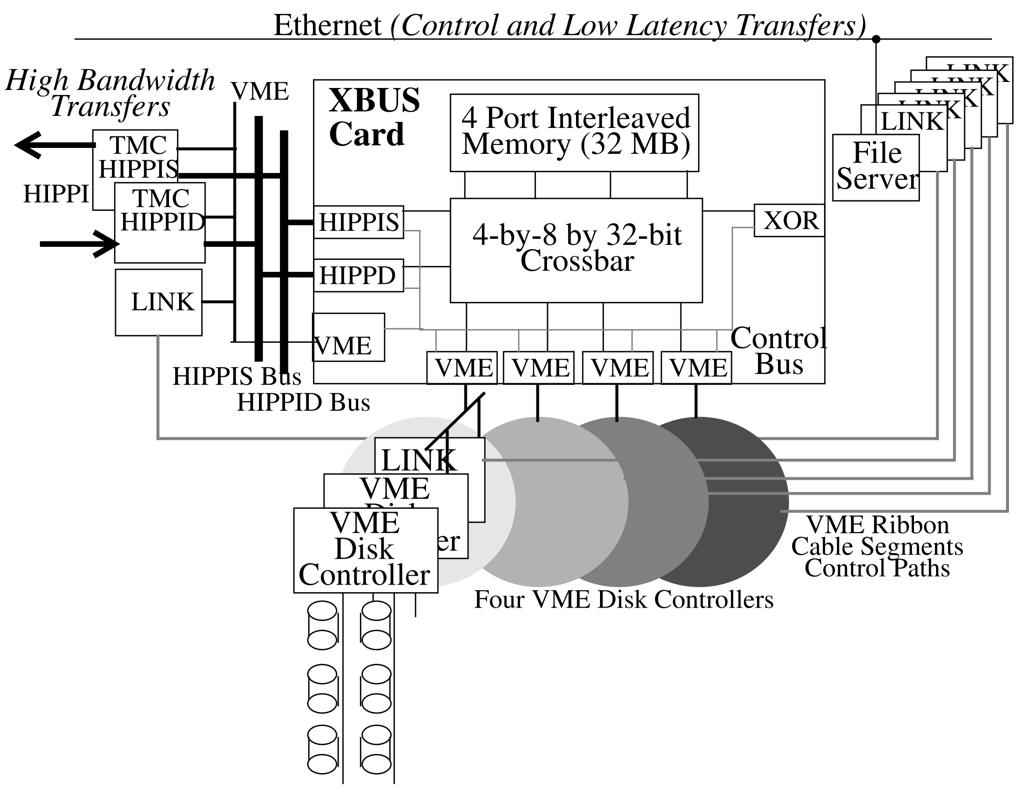


Figure 13: RAID-II Architecture. A high-bandwidth crossbar connects the network interface (HIPPI), disk controllers, multiported memory system, and parity computation engine (XOR). An internal control bus provides access to the crossbar ports, while external point-to-point VME links provide control paths to the surrounding SCSI and HIPPI interface boards. Up to two VME disk controllers can be attached to each of the four VME interfaces.

The XBUS card provides a high-bandwidth path between the major system components: the HIPPI network, four VME busses that connect to VME disk controllers, and an interleaved, multi-ported semiconductor memory. The XBUS card also contains a parity computation engine that generates parity for writes and reconstruction on the disk array. The data path between these system components is a 4×8 crossbar switch that can sustain approximately 160 MB/s. The entire system is controlled by an external Sun 4/280 file server through a memory-mapped control register interface. Figure 13 shows a block diagram for the controller.

To explore how the XBUS card enhances disk array performance, Chen, et al. [Chen94] compare the performance of RAID-II to RAID-I (Table 7). RAID-I is basically RAID-II without the XBUS card [Chervenak91]. They find that adding a custom interconnect board with a parity engine improves performance by a factor of 8 to 15 over RAID-I. The maximum bandwidth of RAID-II is between 20 and 30 MB/s, enough to support the full disk bandwidth of approximately 20 disk drives.

5.6 IBM Hagar Disk Array Controller

Hagar is a disk array controller prototype developed at the IBM Almaden Research Center [Menon93a]. Hagar was designed for large capacity (up to 1 TB), high bandwidth (up to 100 MB/

	Disk Array Read Performance	**Disk Array Write Performance**	**Write Performance Degradation**
RAID-I	2.4 MB/s	1.2 MB/s	50%
RAID-II	20.9 MB/s	18.2 MB/s	13%
RAID-II speedup	8.7	15.2	

Table 7: Performance Comparison between RAID-II and RAID-I. This table compares the performance of RAID-II to that of RAID-I. Because RAID-II has a special purpose parity engine, disk array write performance is comparable to disk array read performance. All writes in this test are full-stripe writes [Lee91b]. For RAID-II reads, data is read from the disk array into XBUS memory, then sent over the HIPPI network back to XBUS memory. For RAID-I reads, data is read from the disk array into Sun4 memory, then copied again into Sun4 memory. This extra copy equalizes the number of memory accesses per data word. For RAID-II writes, data starts in XBUS memory, is sent over HIPPI back into XBUS memory, parity is computed, and the data and parity are written to the disk subsystem. For RAID-I writes, data starts in Sun4 memory, gets copied to another location in Sun4 memory, then is written to disk. Meanwhile, parity is computed on the Sun4 and later written to disk. RAID-I uses a 32 KB striping unit with 8 disks (and is performance-limited by the Sun4's VME bus); RAID-II uses a 64 KB striping unit with 24 disks.

s), and high I/O rate (up to 5000 4-KB I/Os per second). In addition, Hagar provides high availability through the use of redundant hardware components, multiple power boundaries, and on-line reconstruction of data.

Two design features of Hagar are especially noteworthy. First, Hagar uses battery-backed memory to allow user writes to provide safe, asynchronous writes (as discussed in Section 4.1.1). The designers of Hagar require each write to be stored in two separate memory locations in two different power regions to further increase reliability.

Second, Hagar incorporates a special-purpose parity computation engine inside the memory of the controller. This is in contrast to the RAID-II architecture, which places the parity engine as a port on the controller bus (Figure 13). The Hagar memory system supports a special store operation that performs an exclusive-or on the current contents of a memory location with the new data, then writes the result to that location. Incorporating the parity engine in the memory complicates the memory system, but it reduces the data traffic on the controller's internal data bus.

Hagar was never fully operational; however, IBM is working on future disk array products that use ideas from Hagar.

6 OPPORTUNITIES FOR FUTURE RESEARCH

Redundant disk arrays have rejuvenated research into secondary storage systems over the past five to seven years. As this survey highlights, much has been proposed and examined, but much is left to do. This section discusses the classes of research not adequately understood with particular attention to specific open problems.

6.1 Experience with Disk Arrays

As an over five year old research area that has sported products for at least six years, redundant disk arrays have remarkably few published measurement results and experience. In addition to validating models and techniques found in the literature, such experience reports can play an important role in technology transfer [Buzen86]. Furthermore, measurements frequently form the basis for developing new optimizations.

6.2 Interaction among New Organizations

As this survey describes, there are many new and different disk array organizations. Most of these, including double failure correction, declustered parity, parity logging, floating parity, distributed sparing, log-structured file systems, and file-specific data striping, have only been studied in isolation. Unquestionably among these there will be significant interactions, both serious new problems and obvious simplifications or optimizations.

As more is understood about the interactions among disk array technologies, designers and managers of disk arrays will be faced with the task of configuring and tuning arrays. As Section 4.5 discusses, redundant disk array performance and reliability modelling is largely incomplete and unsophisticated. Work needs to be done in the application of fundamental modelling to the problem of disk arrays as well as the development of that fundamental modelling, fork-join queueing models in particular. A good goal for of this work is graphical, interactive analysis tools exploiting low overhead monitoring data to guide configuration and tuning.

One objection commonly lodged against redundant disk arrays, particularly some of the newly proposed technologies, is their relatively high complexity. Storage systems are responsible for more than just the availability of our data, they are responsible for its integrity. As the complexity goes up, the opportunity for disastrous latent bugs also rises. This is compounded by the desire to increase performance by continuing computation as soon as storage modifications are delivered to storage server memory; that is, before these modifications are committed to disk. Inexpensive and highly reliable mechanisms are needed to control the vulnerability to increased software complexity of storage systems.

6.3 Scalability, Massively Parallel Computers, and Small Disks

One of the key motivations for redundant disk arrays is the opportunity to increase data parallelism in order to satisfy the data processing needs of future generations of high-performance computers. This means that arrays must scale up with the massively parallel computers that are being built and the even more massively parallel computers being planned. Massively parallel disk arrays introduce many problems: physical size, connectively, delivery-system bottlenecks, and storage control processing requirements to name a few. The most compelling approach to ever larger disk arrays is to embed storage based on the new generations of small diameter disks into the fabric of massively parallel computers, use the computer's interconnection network for data distribution and redundancy maintenance, and distribute the storage control processing throughout the processors of the parallel computer.

Though compelling, this approach has substantial problems to be overcome. Primary among these are the impact on the interconnection network of distributing the redundancy computations [Cao93], the impact on the processors of distributing storage control, and the viability of allocating data on storage devices near the processors that will use it.

6.4 Latency

Redundant disk arrays are fundamentally designed for throughput, either high transfer rates for large, parallel transfers or large numbers of concurrent small accesses. They are only effective for reducing access latency when this latency is limited by throughput. For lower throughput workloads, disk arrays enhance storage performance only slightly over traditional storage systems.

Caching is the main mechanism for reducing access latency, but caching can be ineffective either because data is too large, too infrequently accessed, or too frequently migrated among caches. For these workloads, data prefetching is essential. Research into aggressive prefetching systems is beginning to examine opportunities to extract or predict future accesses and provide mechanisms to efficiently utilize available resources in anticipation of these accesses [Korner90, Kotz91, Gibson92, Patterson93, Tait91].

7 CONCLUSIONS

Disk arrays have moved from research ideas in the late 1980's to commercial products today. The advantages of using striping to improve performance and redundancy to improve reliability have proven so compelling that most major computer manufacturers are selling or intending to sell disk arrays. Much research and implementation have been accomplished, both in industry and universities, but many theoretical and practical issues remain unresolved. We look forward to the many more fruitful years of disk array research.

8 ACKNOWLEDGEMENTS

We thank Bill Courtright, Mark Holland, Jai Menon, and Daniel Stodolsky for reading an earlier draft of this paper and for their many helpful comments. We are especially indebted to Bill Courtright and Daniel Stodolsky for writing the section of this paper describing the NCR disk array.

9 ANNOTATED BIBLIOGRAPHY

[Amdahl67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings AFIPS 1967 Spring Joint Computer Conference*, volume 30, pages 483–485, April 1967.

Three page paper that eloquently gives case for traditional computers by pointing out that performance improvement is limited by portion of the computation that is not improved.

[Baccelli85] Francois Baccelli. Two Parallel Queues Created by Arrivals with Two Demands. Technical Report 426, INRIA–Rocquencourt France, 1985.

Derives an exact solution for the two-server, M/G/1 fork-join queue.

[Bhide92] Anupam Bhide and Daniel Dias. Raid Architectures for OLTP. Technical Report RC 17879 (#78489), IBM, March 1992.

Increases throughput for workloads emphasizing small, random write accesses in a redundant disk array by logging changes to parity for efficient application later. Parity changes are logged onto a separate disk which must be externally sorted before application to the disk array's parity.

[Bitton88] Dina Bitton and Jim Gray. Disk Shadowing. In *Very Large Database Conference XIV*, pages 331–338, 1988.

Describes disk mirroring and derives an analytical equation for read and write seek distances as a function of the number of data copies.

[Burkhard93] W. Burkhard and J. Menon. Disk Array Storage System Reliability. In *23rd Annual International Symposium on Fault-Tolerant Computing*, pages 432–441, June 1993.

Argues need for multiple-error-correcting disk arrays, discusses how to use maximal distance separable codes to protect against multiple errors in a space-efficient manner.

[Buzen86] Jeffrey P. Buzen and Annie W.C. Shum. I/O Architecture in MVS/370 and MVS/XA. *CMG Transactions*, 54:19–26, Fall 1986.

Overview of the MVS/370 and MVS/XA I/O architecture. Describes channel paths, storage directors, string controllers, rotational position sensing, static and dynamic reconnect.

[Cao93] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP parallel RAID architecture. In *Proceedings of the 1993 International Symposium on Computer Architecture*, May 1993.

Describes the TickerTAIP architecture, software implementation issues, and the performance of different methods of distributing parity computation among multiple processors.

[Chandy93] John Chandy and A. L. Narasimha Reddy. Failure Evaluation of Disk Array Organizations. In *Proceedings of the International Conference on Distributed Computing Systems*, May 1993.

Contrasts four previously described schemes for minimizing data reconstruction time in small (7 and 16 disks) redundant disk arrays: RAID 5 with a single spare disk, RAID 5 with a single spare whose space is distributed across all disks, a special case of Muntz and Lui's parity clustering organization, and a method of dynamically converting a redundant data disk to a spare disk by merging two redundancy groups into one larger group. The second, distributed sparing, is generally preferred because of its performance and simplicity, but the Muntz scheme is better for minimal impact of user performance during recovery.

[Chen90a] Peter M. Chen, Garth Gibson, Randy H. Katz, and David A. Patterson. An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1990.

The first experimental evaluation of RAID. Compares RAID levels 0, 1, and 5.

[Chen90b] Peter M. Chen and David A. Patterson. Maximizing Performance in a Striped Disk Array. In *Proceedings of the 1990 International Symposium on Computer Architecture*, pages 322–331, May 1990.

Discusses how to choose the striping unit for a RAID level 0 disk array.

[Chen91] Shenze Chen and Don Towsley. A Queueing Analysis of RAID Architectures. Technical Report COINS Tech. Report 91-71, University of Massachusetts, Amherst, Department of Computer and Information Science, 1991.

Analytically models RAID level 1 and RAID level 5 disk arrays to compare their performance on small and large requests. Bounds are used to model the queueing and fork-join synchronization in RAID level 1 disk arrays. Small write requests in RAID level 5 disk arrays are handled by ignoring the fork-join synchronization overhead. Large requests are modeled by using a single queue for all the disks in the disk array.

[Chen93] Peter M. Chen and Edward K. Lee. Striping in a RAID Level 5 Disk Array. Technical Report CSE-TR-181-93, University of Michigan, November 1993.

Discusses how to choose striping unit for RAID Level 5 disk arrays. Quantifies effect of writes and varying number of disks.

[Chen94] Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, and Randy H. Katz. Performance and Design Evaluation of the RAID-II Storage Server. *Invited to the Journal of Distributed and Parallel Databases, to appear*, 1994. also appeared in The 1993 International Parallel Processing Symposium Workshop on I/O in Parallel Computer Systems.

Summarizes major architectural features of RAID-II and evaluates how they impacts performance.

[Chervenak91] Ann L. Chervenak and Randy H. Katz. Performance of a Disk Array Prototype. In *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 19, pages 188–197, May 1991. Performance Evaluation Review.

Evaluates the performance of RAID-I, a U.C. Berkeley disk array prototype.

[Copeland88] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data Placement in Bubba. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 99–108, 1988.

Discusses data allocation in a large database.

[Drapeau94] Ann L. Drapeau, Ken Shirriff, Edward K. Lee, Peter M. Chen, Garth A. Gibson, John H. Hartman, Ethan L. Miller, Srinivasan Seshan, Randy H. Katz, Ken Lutz, and David A. Patterson. RAID-II: A High-Bandwidth Network File Server. In *Proceedings of the 1994 International Symposium on Computer Architecture*, April 1994.

Describes the architecture, file system, and performance of RAID-II, a disk-array file server prototype.

[Emlich89] Larry W. Emlich and Herman D. Polich. VAXsimPLUS, A Fault Manager Implementation. *Digital Technical Journal*, 8, February 1989.

Describes Digital Equipment Corporation's tool for predicting an avoiding disk failures.

[Flatto84] L. Flatto and S. Hahn. Two Parallel Queues Created by Arrivals with Two Demands I}. *SIAM Journal of Computing*, pages 1041–1053, October 1984.

Derives an exact solution for the two server, M/M/1, fork-join queue.

[Friedman83] Mark B. Friedman. DASD Access Patterns. In *14th International Conference on Management and Performance Evaluation of Computer Systems*, pages 51–61, 1983. CMG XIV.

Looks at how much disk accesses are skewed towards particular disks in several transaction processing sites.

[Gibson91] Garth Alan Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage.* PhD thesis, University of California at Berkeley, December 1991. also available from MIT Press, 1992.

Award winning dissertation that describes RAIDs in detail, with emphasis on reliability analysis of several alternatives.

[Gibson92] Garth A. Gibson, R. Hugo Patterson, and M. Satyanarayanan. Disk Reads with DRAM Latency. *Third Workshop on Workstation Operating Systems*, April 1992.

Proposes that applications give hints about their future file accesses so that the buffer cache can prefetch needed data and provide low-latency file access. The hints could also be exploited to improve cache management and disk scheduling.

[Gray90] Jim Gray, Bob Horst, and Mark Walker. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. In *Proceedings of the 16th Very Large Database Conference*, pages 148–160, 1990. VLDB XVI.

Describes an data and parity layout for disk arrays called parity striping. Parity striping is essentially RAID level 5 with an infinite striping unit and manual load balancing.

[Hall86] M. Hall. *Combinatorial Theory (2nd Edition)*. Wiley-Interscience, 1986.

Textbook on combinatorial theory. The section on balanced, incomplete block designs are most relevant for readers of this paper.

[Heidelberger82] Philip Heidelberger and Kishor S. Trivedi. Queueing Network Models for Parallel Processing with Asynchronous Tasks. *IEEE Transactions on Computers*, C-31(11):1099–1109, November 1982.

Derives approximate solutions for queueing systems with forks but no joins.

[Hennessy90] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc., 1990.

Likely the most popular general book in computer architecture today, the discussion on technology trends, general I/O issues, and measurements of seek distances are most relevant to readers of this paper.

[Holland92] Mark Holland and Garth Gibson. Parity Declustering for Continuous Operation in Redundant Disk Arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 23–35, October 1992.

Describes parity declustering, a technique for improving the performance of a redundant disk array in the presence of disk failure. Analyzes the proposed solution using detailed simulation and finds significant improvements (20-50%) in both user response time and reconstruction time. Also analyzes a set of previously-proposed optimizations that can be applied to the reconstruction algorithm, concluding that they can actually slow the reconstruction process under certain conditions.

[Holland93] Mark Holland, Garth Gibson, and Daniel Siewiorek. Fast, On-Line Failure Recovery in Redundant Disk Arrays. In *Proceedings of the 23rd International Symposium on Fault Tolerant Computing*, 1993. FTCC-23.

Compares and contrasts two data reconstruction algorithms for disk arrays: "parallel stripe-oriented reconstruction" and "disk-oriented reconstruction". Presents an implementation of the disk-oriented algorithm and analyzes reconstruction performance of these algorithms, concluding that the disk oriented algorithm is superior. Investigates the sensitivity of the reconstruction process to the size of the reconstruction unit and the amount of memory available for reconstruction.

[Hsiao90] H. Hsiao and D. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proceedings of the 1990 IEEE International Conference on Data Engineering*, pages 456–465, February 1990.

Introduces a variation of mirroring, where the secondary copy of data is distributed across the disks in a different manner than the primary copy of data.

[Katz92] Randy H. Katz. High Performance Network and Channel-Based Storage. *Proceedings of the IEEE*, 80(8):1238–1261, August 1992.

Presents overview of network-based storage systems. Reviews hardware and software trends in storage systems.

[Katz93] Randy H. Katz, Peter M. Chen, Ann L. Drapeau, Edward K. Lee, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, and David A. Patterson. RAID-II: Design and Implementation of a Large Scale Disk Array Controller. In *1993 Symposium on Integrated Systems*, 1993. University of California at Berkeley UCB/CSD 92/705.

Describes the design decisions and implementation experiences from RAID-II.

[Kim86] Michelle Y. Kim. Synchronized Disk Interleaving. *IEEE Transactions on Computers*, C-35(11):978–988, November 1986.

Simulates the performance of independent disks versus synchronized disk striping. Derives an equation for response time by treating the synchronized disk array as an M/G/1 queueing system.

[Kim91] Michelle Y. Kim and Asser N. Tantawi. Asynchronous Disk Interleaving: Approximating Access Delays. *IEEE Transactions on Computers*, 40(7):801–810, July 1991.

Derives an approximate equation for access time in unsynchronized disk arrays when seeks times are exponentially distributed and rotational latency is uniformly distributed.

[Korner90] K. Korner. Intelligent Caching for Remote File Service. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 220–226, 1990.

Uses traces to generate hints based on the program running and the directory and name of files accessed. The file server uses the hints to pick a caching algorithm: LRU, MRU, none. Simulation showed significant benefits from intelligent caching but not from read-ahead which delayed demand requests since it was not preemptable.

[Kotz91] David Kotz and Carla Schlatter Ellis. Practical Prefetching Techniques for Parallel File Systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 182–189, December 1991.

File access predictors use past accesses to prefetch data in idle nodes of a parallel file system. Simulation studies show that practical predictors can often significantly reduce total execution time while the penalty for incorrect predictions is modest.

[Lee91a] Edward K. Lee and Randy H. Katz. An Analytic Performance Model of Disk Arrays and its Applications. Technical Report UCB/CSD 91/660, University of California at Berkeley, 1991.

Derives an analytic model for non-redundant disk arrays and uses the model to derive an equation for the optimal size of data striping.

[Lee91b] Edward K. Lee and Randy H. Katz. Performance Consequences of Parity Placement in Disk Arrays. In *Proceedings of the 4rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 190–199, April 1991.

Investigates the performance of different methods of distributing parity in RAID level 5 disk arrays.

[Lee93] Edward K. Lee and Randy H. Katz. An Analytic Performance Model of Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 98–109, May 1993.

Similar to earlier technical report with similar name except with better empirical justifications and a more detailed study of the model's properties.

[Livny87] Miron Livny, Setrag Khoshafian, and Haran Boral. Multi-Disk Management Algorithms. In *Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 69–77, May 1987.

Compares performance of disk arrays with track-sized and infinite striping units. Concludes that striping can improve performance for many multi-disk systems.

[LoVerso93] Susan J. LoVerso, Marshall Isman, Andy Nanopoulos, et al. sfs: A Parallel File System for the CM-5. In *Proceedings USENIX Summer Conference*, June 1993.

A description of the I/O hardware and the file system of the massively parallel processor from Thinking Machines. Their RAID-3 disk array has excellent performance for large file accesses.

[Malhotra93] Manish Malhotra and Kishor S. Trivedi. Reliability Analysis of Redundant Arrays of Inexpensive Disks. *Journal of Parallel and Distributed Computing*, 17:146–151, January 1993.

Uses Markov models to derive exact, closed-form reliability equations for redundant disk arrays. Analysis accounts for failure prediction and sparing.

[Menon91] Jai Menon, Dick Mattson, and Spencer Ng. Distributed Sparing for Improved Performance of Disk Arrays. Technical Report RJ 7943, IBM, January 1991.

Explores the use of an on-line spare disk in a redundant disk array analytically. It examines multiple configurations, but fundamentally it distributes the spare's space over the whole array so that every disk is N/(N+2) data, 1/(N+2) parity, and 1/(N+2) spare. This gives an extra 1/(N+2) performance, but, more significantly, it distributes the recovery-write load (the reconstructed data) over all disks to shorten recovery time. The benefits, not surprisingly, are largest for small arrays.

[Menon93a] Jai Menon and Jim Cortney. The Architecture of a Fault-Tolerant Cached RAID Controller. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 76–86, May 1993.

Describes the architecture of Hagar and several algorithms for asynchronous writes that reduce susceptibility to data loss,

[Menon93b] Jai Menon, James Roche, and Jim Kasson. Floating Parity and Data Disk Arrays. *Journal of Parallel and Distributed Computing*, 17:129–139, 1993.

Introduces floating data and floating parity as an optimization for RAID level 5 disk arrays. Discusses performance and capacity overheads of methods.

[Merchant92] A. Merchant and P. Yu. Design and Modeling of Clustered RAID. In *Proceedings of the International Symposium on Fault Tolerant Computing*, pages 140–149, 1992. FTCC.

Presents an implementation of parity declustering, which the authors call "clustered RAID", based on random permutations. Its advantage is that it is able to derive a data mapping for any size disk array with any size parity stripe, and the corresponding disadvantage is that the computational requirements of the mapping algorithm are high compared to the block-design based approaches. Analyzes response time and reconstruction time using this technique via an analytic model, and finds substantial benefits in both.

[Mon91] RAID: A Technology Poised for Explosive Growth. Technical Report DJIA: 2902, Montgomery Securities, December 1991.

Industry projections of market growth for RAID systems from 1990 to 1995.

[Muntz90] Richard R. Muntz and John C. S. Lui. Performance Analysis of Disk Arrays under Failure. In *Proceedings of the 16th Conference on Very Large Data Bases*, 1990. VLDB XVI.

Proposes and evaluates the "clustered RAID" technique for improving the failure-recovery performance in redundant disk arrays. It leaves open the problem of implementation: no technique for efficiently mapping data units to physical disks is presented. Analyzes via an analytical model the technique and two potential "optimizations" to the reconstruction algorithm, and finds significant benefits to all three.

[Nelson88] R. Nelson and A.N. Tantawi. Approximate Analysis of Fork/Join Synchronization in Parallel Queues. *IEEE Transactions on Computers*, 37(6):739–743, June 1988.

Approximates response time in fork-join queueing systems with k >= 2 servers where each logical request always forks into k requests.

[Ng92] Spencer Ng and Dick Mattson. Maintaining Good Performance in Disk Arrays During Failure via Uniform Parity Group Distribution. In *Proceedings of the First International Symposium on High Performance Distributed Computing*, pages 260–269, 1992.

Uses balanced, incomplete block designs to distribute the extra load from a failed disk equally among other disks in the array.

[Ng94] Spencer Ng. Crossbar Disk Array for Improved Reliability and Performance. In *Proceedings of the 1994 International Symposium on Computer Architecture*, April 1994.

Introduces schemes to protect against multiple failures of disk support hardware such as disk controllers and strings.

[Orji93] Cyril U. Orji and Jon A. Solworth. Doubly Distorted Mirrors. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1993.

Describes a technique called distorted mirrors that partitions each of two mirrored disks into two halves, one of which lays out the data in a standard fashion, one of which "distorts" the data layout. This accelerates writes to the distorted copy while preserving the ability to sequentially read large files.

[Patterson88] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *International Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

The first published Berkeley paper on RAIDs, it gives all the RAID nomenclature.

[Patterson93] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A Status Report on Research in Transparent Informed Prefetching. *ACM Operating Systems Review*, 27(2):21–34, April 1993.

Expands on using application hints for file prefetching in [Gibson92]. Hints should disclose access patterns, not advise caching/prefetching actions. Greatest potential from converting serial accesses into concurrent accesses on a disk array. Presents preliminary results of user-level prefetching tests.

[Patterson94] David A. Patterson and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann Publishers, 1994.

A popular undergraduate book in computer architecture, the discussion on technology trends are most relevant to readers of this paper.

[Peterson72] W. Wesley Peterson and E. J. Weldon. *Error-Correcting Codes, Second Edition*. MIT Press, 1972.

A general textbook on the mathematics of error-correcting codes.

[Rosenblum91] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, October 1991.

Describes a Log-Structured File System that makes all writes to disk sequential. Discusses efficient ways to clean the disk to prevent excessive fragmentation.

[Salem86] Kenneth Salem and Hector Garcia-Molina. Disk Striping. In *Proceedings of the Second International Conference on Data Engineering*, pages 336–342, 1986.

Early paper discussing disk striping.

[Scheuermann91] Peter Scheuermann, Gerhard Weikum, and Peter Zabback. Automatic Tuning of Data Placement and Load Balancing in Disk Arrays. *Database Systems for Next-Generation Applications: Principles and Practice*, 1991. DBS-92-91.

Describes heuristics for allocating files to disks to minimize disk skew.

[Schulze89] Martin Schulze, Garth Gibson, Randy Katz, and David Patterson. How Reliable is a RAID? In *Procedures of the IEEE Computer Society International Conference (COMPCON)*, March 1989. Spring COMPCON 89.

Gives a reliability calculation for the electronics as well as the disks for RAIDs.

[Seltzer90] Margo I. Seltzer, Peter M. Chen, and John K. Ousterhout. Disk Scheduling Revisited. In *Proceedings of the Winter 1990 USENIX Technical Conference*, pages 313–324, January 1990.

Re-examines the problem of how to efficiently schedule a large number of disk accesses when accounting for both seek and rotational positioning delays.

[Stodolsky93] Daniel Stodolsky and Garth A. Gibson. Parity Logging: Overcoming the Small Write Problem in Redundant Disk Arrays. In *Proceedings of the 1993 International Symposium on Computer Architecture*, May 1993.

Increases throughput for workloads emphasizing small, random write accesses in a redundant disk array by logging changes to the parity in a segmented log for efficient application later. Log segmentation allows log operations that are large enough to be efficient yet small enough to allow in-memory application of a log segment.

[Tait91] C. D. Tait and D. Duchamp. Detection and Exploitation of File Working Sets. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 2–9, May 1991.

Dynamically builds and maintains program and data access trees to predict future file accesses. The current pattern is matched with previous trees to prefetch data and manage the local cache in a distributed file system. Trace-driven simulation shows reduced cache miss rates over a simple LRU algorithm.

[Weikum92] Gerhard Weikum and Peter Zabback. Tuning of Striping Units in Disk-Array-Based File Systems. In *Proceedings of the 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pages 80–87, 1992.

Proposes file-specific striping units instead of a single, global one for all files.

[Wilmot89] Richard B. Wilmot. File Usage Patterns from SMF Data: Highly Skewed Usage. In *20th International Conference on Management and Performance Evaluation of Computer Systems*, pages 668–677, 1989. CMG 1989.

Reports on how files are accessed on four large data centers and finds that a small number of files account for most of all disk I/O.