1 Overview

In recent years, interest in RAID, Redundant Arrays of Inexpensive Disks, 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.

Disk arrays, which organize multiple independent disks into a large, high-performance logical disk, are a natural solution to this problem. Disk arrays stripe data across multiple disks and accessing them in parallel to achieve both higher data transfer rates on large data accesses and higher I/O rates on small data accesses. 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 of 200,000 hours (approx. twenty-three years) for a single disk implies an MTTF of 2000 hours (approx. 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.

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 chapter presents a systematic tutorial and survey of disk arrays.

2 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; the reliability of RAID-based systems in the face of system crashes, uncorrectable bit-errors and correlated disk failures; and finally, issues in the implementations of block-interleaved, redundant disk arrays.

2.1 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 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 the granularity of data interleaving and 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 at a relatively small unit such that all I/O requests, regardless or 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 disadvantage 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 at a relatively large unit so that small I/O requests need access only a small number of disks but large requests can access all the disks in the disk array. This allows multiple small requests to be serviced simultaneously but still allows large requests the benefits of using all the disks in the disk array.

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 but there are some that use Hamming codes 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.

2.2 Basic RAID Organizations

This subsection describes the basic RAID, Redundant Arrays of Inexpensive Disks, organizations which will be used as the basis for further examinations of performance, cost and reliability of disk arrays. In addition to presenting RAID levels 1 through 5 which first appeared in [7], we present two other RAID organizations, RAID levels 0 and 6, which have since become generally accepted. 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 1 schematically illustrates the seven RAID organizations.

2.2.1 Non-Redundant (RAID Level 0)

The non-redundant disk array, or RAID level 0, has the lowest cost of any redundancy scheme 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 such as mirroring, which duplicate data, can perform better on reads by selectively scheduling requests on the disk with the shortest expected seek and rotational delays [1]. 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.

2.2.2 Mirrored (RAID Level 1)

The traditional solution, called *mirroring* or *shadowing*, uses twice as many disks as a non-redundant disk array [1]. 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 [2]. If a disk fails, the second copy is used to service requests. Mirroring is frequently used in database applications where availability and transaction rate are more important than storage efficiency [4].

2.2.3 Memory-Style ECC (RAID Level 2)

Memory systems have provided recovery from failed components with much less cost than mirroring by using Hamming codes [8]. 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

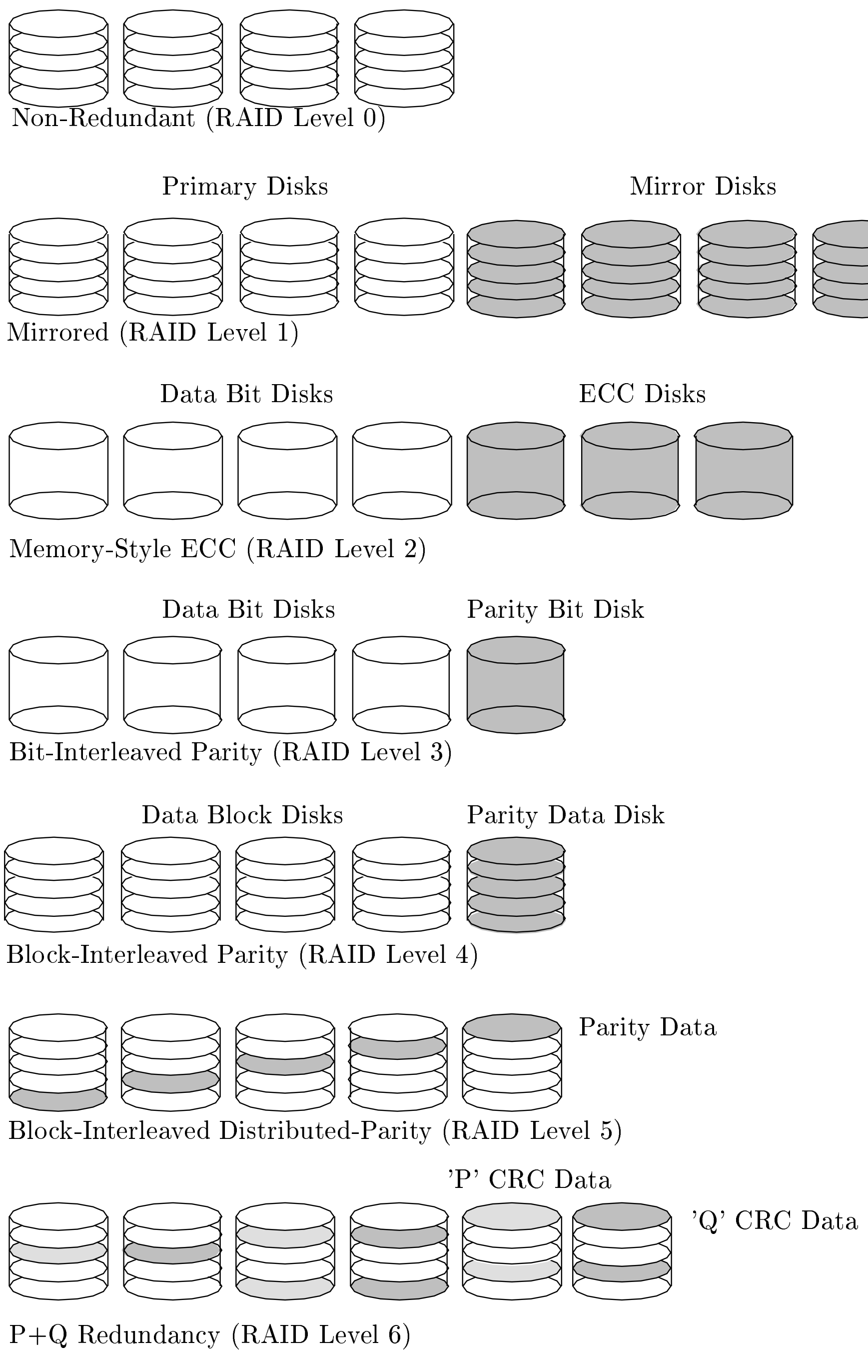

Figure 1: **The seven basic RAID schemes** for a user storage capacity of four disks.

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.

2.2.4 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 which 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 simple to implement.

2.2.5 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* [3]. 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 XORing 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 in order to compute 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.

2.2.6 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 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 one of 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.

The exact method used to distribute parity in block-interleaved distributed-parity disk arrays can have an impact on performance. Figure 2 illustrates the best parity distribution of those investigated in [5], 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.

2.2.7 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, however, it is desirable to use stronger codes that can tolerate multiple disk failures. Moreover, when a disk fails in a parity protected disk array, recovering the contents of the

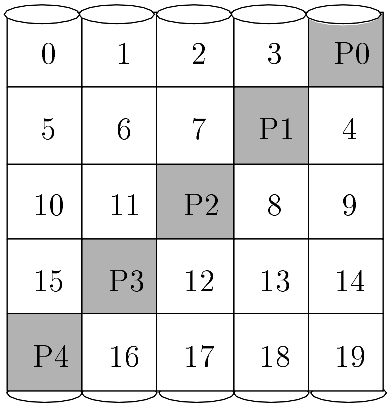

Figure 2: **Left-Symmetric Parity Placement in a RAID 5 array.** Each square corresponds to a stripe unit. Each column of squares corresponds to a disk. P0 computes the parity over stripe units 0, 1, 2 and 3; P1 computes parity over stripe units 4, 5, 6 and 7; etc.

failed disk requires successfully reading the contents of all non-failed disks. As we will see in Section 2.5, the probability of encountering an uncorrectable read error during recovery can be 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.

2.3 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 subsection, we compare RAID levels 0 through 6 based on performance and cost. The following subsection examines reliability.

2.3.1 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, throughput becomes more important relative to response

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

Table 1: **Throughput per dollar metrics of different RAID levels** relative to RAID level 0. G is the size of the parity group.

time. In other words, maximum throughput, especially on random I/O requests, is a "hard" performance metric which is difficult to improve by using software techniques.

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 storage 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 equivalent to RAID level 1 (mirroring) with the exception that in a mirrored disk array, certain disk scheduling optimizations on reads are performed that generally are not implemented for RAID level 5 disk arrays. 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 indicates the use of RAID level 1 disk arrays while a small striping unit size relative to the size of an average request may indicate the use of a RAID level 3 disk array.

2.4 Comparisons

Table 1 tabulates the maximum throughput per dollar relative to RAID level 0 for RAID levels 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 system. For the above disk array organizations, the storage efficiency is equal to the performance/cost metric for large writes.

Figure 3 graphs the performance/cost metrics from Table 1 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 performance/cost of RAID level 3 systems is always less than or equal to the performance/cost of RAID level 5 systems. This is

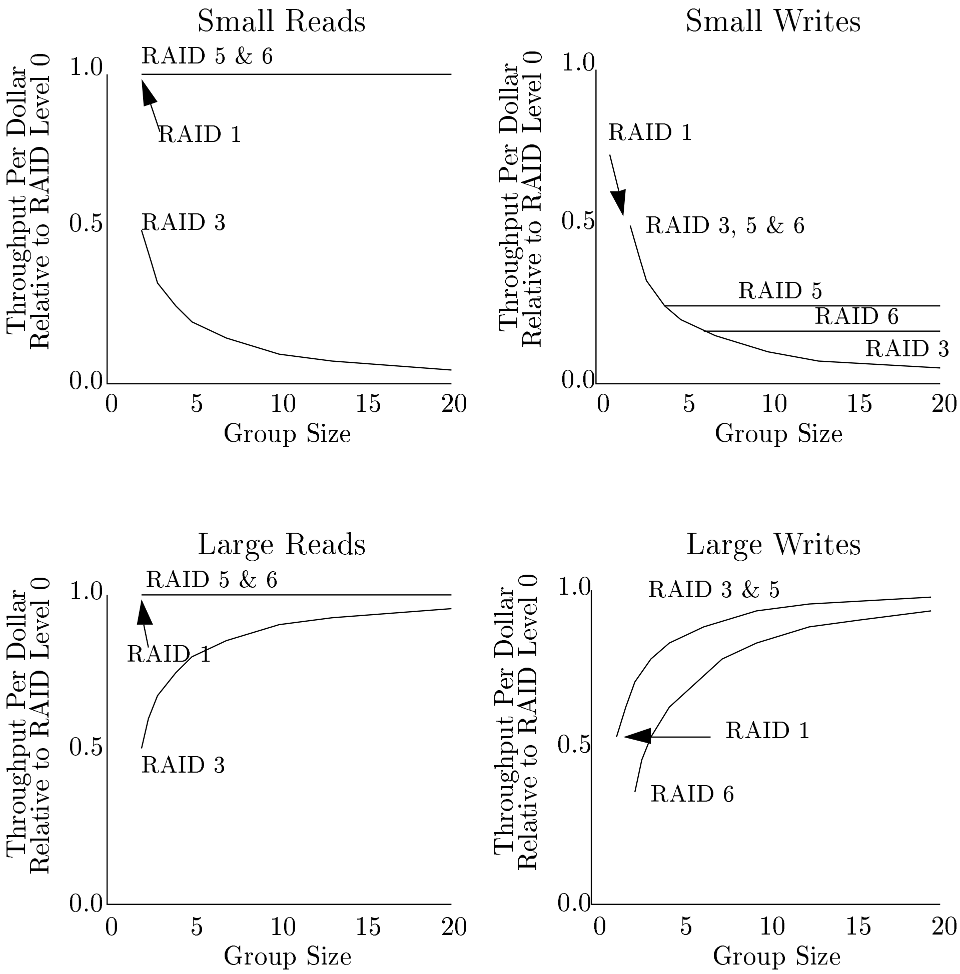

Figure 3: **Throughput per dollar relative to RAID level 0** as a function of the group size

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.

2.5 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 disk arrays. This subsection 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.

2.5.1 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 an N disk RAID level 5 with group size G to be $\frac{MTTF^2(disk)}{N\times(G-1)\times MTTR(disk)}$, 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 [7]. 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.

For a disk array with two redundant disk per parity group, such as P+Q redundancy, the mean time to failure is $\frac{MTTF^2(disk)}{N\times(G-1)\times(G-2)\times MTTR^2(disk)}$. 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 of course, but it at least gives us an idea of the potential reliability afforded by disk arrays. The rest of this subsection 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 which can dramatically affect the reliability of disk arrays.

2.5.2 System Crashes and Parity Inconsistency

In this subsection, 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 [6], 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 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. 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 corresponding to any write requests that were interrupted.

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 and every 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.

2.5.3 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 on writes but are detected by reads?

Disk manufacturers 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 $(1 - \frac{1}{2.4 \times 10^{10}})^{2.0 \times 10^8} = 99.2\%$. 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.

2.5.4 Correlated Disk Failures

The simplest model of reliability of disk arrays [7] 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 will 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 which 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.

2.5.5 Reliability Revisited

The previous subsections have described how system crashes, uncorrectable bit errors and correlated disk failures can decrease the reliability of redundant disk arrays. In this subsection, we will calculate mean-time-to-data-loss statistics by 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.

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

Total User Capacity	100 disks (500 GB)	MTTF(disk)	200,000 hours
Disk Size	5 GB	MTTF(disk2)	20,000 hours
Sector Size	512 bytes	MTTF(disk3)	2,000 hours
Bit Error Rate	1 in 10^{14} bits	MTTR(disk)	1 hour
	1 in 2.4×10^{10} sectors		
p(disk)	99.96%	MTTF(system)	1 month
Parity Group Size	16	MTTR(system)	1 hour

Table 2: **Reliability parameters used in the forthcoming calculations.** p(disk) is the probability of reading all sectors of the disks; it can be derived from the disk size, sector size and bit error rate.

is ten times more likely than the previous failure (until the failed disk has been reconstructed). Table 2 tabulates values of the reliability parameters we will use for calculating numeric reliability estimates in this subsection. Note that the reliability estimates will be computed at a constant user capacity of 100 disks, consisting of independent, 16-disk parity groups.

Table 3, 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 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.

Type of Failure	Mean Time To Data Loss	MTTDL (Years)	10 Year Failure Rate
Double Disk Failure	$\frac{MTTF(disk) \times MTTF(disk2)}{N \times (G-1) \times MTTR(disk)}$	285	3.4%
System Crash + Disk Failure	$\frac{MTTF(system) \times MTTF(disk)}{N \times MTTR(system)}$	154	6.3%
Disk Failure + Bit Error	$\frac{MTTF(disk)}{N \times (1-(p(disk)^{G-1}))}$	36	24.4%
Software RAID	(harmonic sum of above)	26	31.6%
Hardware RAID (with NVRAM)	(harmonic sum excluding system crash + disk failure)	32	26.8%

Table 3: **Failure Characteristics for RAID Level 5 Disk Arrays.** The 10 year failure rate is the percent chance of data loss in a 10 year period. Note that the total number of disks in the system, N, is equal to the number of data disks times $G/(G-1)$.

Table 4 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 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.

2.5.6 Summary and Conclusions

This subsection 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

Type of Failure	Mean Time To Data Loss	MTTDL (Years)	10 Year Failure Rate
Triple Disk Failure	$\frac{MTTF(disk) \times MTTF(disk2) \times MTTF(disk3)}{N \times (G-1) \times (G-2) \times MTTR^2(disk)}$	38052	0.03%
System Crash + Disk Failure	$\frac{MTTF(system) \times MTTF(disk)}{N \times MTTR(system)}$	144	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	0.02%
Software RAID	(harmonic sum of above)	143	6.8%
Hardware RAID (with NVRAM)	(harmonic sum excluding system crash + disk failure)	21166	0.05%

Table 4: **Failure Characteristics for RAID Level 6 Disk Arrays.** The 10 year failure rate is the percent chance of data loss in a 10 year period. Note that the total number of disks in the system, N, is equal to the number of data disks times $G/(G-2)$.

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 reap the full 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, hopefully, its approximate reliability in comparison to other disk array organizations of similar complexity.

2.6 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 paragraph 2.6.4, is that multiple disks are usually connected to the host computer via a common bus or string.

2.6.1 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 serviced. 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.

2.6.2 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 which 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 still results in higher latencies for random write requests since the parity sectors must be marked inconsistent before the writes can proceed.

2.6.3 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 of 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 need 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.

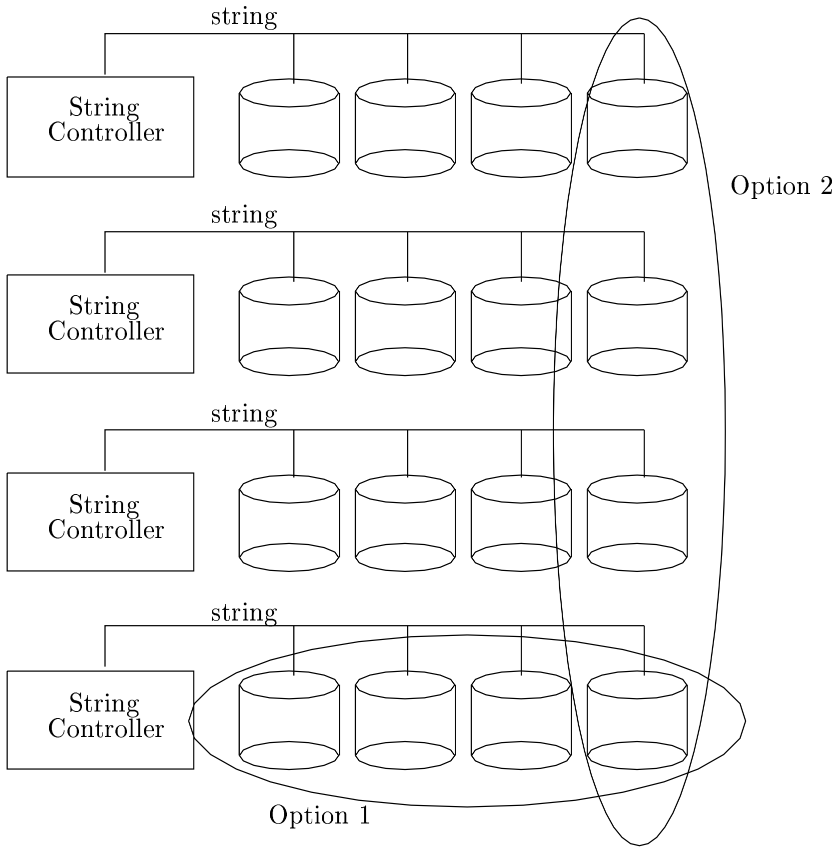

Figure 4: **Orthogonal RAID.** This figure present two options of how to organize error-correction groups in the presence of shared resources, such as a string controller.

The second method, called *parity sparing*, 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, it is not affected by system crashes. 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.

2.6.4 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 loss of data.

For example, consider the 16-disk array in Figure 4 and two options of how to organize multiple error-correction groups. Option 1 combines each string of four disks into a single error-correction 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* [9]. Orthogonal RAID has the added benefit of minimizing string conflicts when multiple disks from a group transfer data simultaneously.

References

[1] DINA BITTON AND JIM GRAY: *Disk Shadowing.* In Very Large Database Conference XIV, pages 331-338, 1988.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] W. WESLEY PETERSON AND E. J. WELDON: *Error-Correcting Codes,* Second Edition. MIT Press, 1972.

[9] 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.