In Proceedings of PDIS'91: The IEEE International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, Dec 1991.

IDA-based Redundant Arrays of Inexpensive Disks

Azer Bestavros

Department of Computer Science
Boston University
Boston, MA 02215

(bestavros@cs.bu.edu)

Abstract

Parallel disks can improve I/O performance in a manner analogous to the use of parallel processors to improve computation times. However, due to their data storage function, reliability issues become exceedingly important. Currently proposed schemes use shadowing or parity to achieve reliability in parallel disks. In this paper we introduce the idea of using the Information Dispersal Algorithm (IDA) of Michael O. Rabin [8] to distribute data and redundancy information uniformly *among multiple disks and compare the performance and reliability characteristics of shadowing, parity, and IDA. We discuss some ways to take advantage of the uniformity of data placement and argue that IDA is the algorithm of choice for achieving reliability and performance in parallel disk systems.*

1 Introduction

Recently, parallel disk systems have emerged as a potential solution for achieving ultra-high capacity, performance and reliability at a reasonably low cost. This emerging technology is likely to have a great impact on the market of data-storage devices for mainframe computers, massively parallel computers and large database systems [9]. The basic idea underlying the design of parallel disk systems is to use a large number of *inexpensive* small disks to attain very large capacities. The relatively low cost per megabyte of storage on these disks, combined with their better volumetric and power efficiencies [7], makes them very attractive when compared to larger disks.

A significant drawback of using a large number of parallel disks to improve I/O performance is the effect it has on reliability. More disks means more frequent failures and, thus, an increased probability of data loss. Without taking additional protective measures, it is well known that the probability of losing data increases almost *linearly* with the number of disks in the system. With large disk arrays, [1] the reliability of the system drops to unacceptable levels. For example, even with a small disk array of 32-64 disks, the MTTF (Mean Time To Fail) is expected to drop to few days.

Interest in parallel disk systems has been spurred by the sharp decline in the price of small hard disk drives. The current ratio between the cost per Megabyte for large and small disks is somewhere between 2 and 5, and, with current market trends, this figure is only expected to rise.[2] Previous research [7] used minimal redundancy primarily to ensure acceptable levels of fault-tolerance. Performance gains from the added redundancy (if any) were viewed as a side effect. In our opinion, this should be reversed. With the aforementioned pricing trends, more redundancy should be used, primarily, to boost performance with fault-tolerance being a byproduct.

In this paper, we propose a novel technique for adding redundancy to disk arrays. Our approach is based on the Information Dispersal Algorithm (IDA) of Michael O. Rabin [8]. We show that IDA is superior to current approaches, namely: shadowing [1] and parity [6]. It yields the best reliability and performance gains for every percent of added redundancy.

2 Shadowing

This is the simplest and most straightforward approach for increasing the reliability of a system. The idea is to keep $(r+1)$ replicas of every block of data. Each one of these replicas is striped over a group of $m = n/(r+1)$ disks. This scheme tolerates up to r failures out of the n disks by an r-fold increase in the required storage.

[1] Some of these are estimated to have as many as 1000 disks (see [6]).

[2] The price is roughly 3-10 Dollars per megabyte for 3.5-inch disks (*e.g.* Conners CP3100) compared to 20-45 Dollars per megabyte for the 10.5-inch and the 14-inch disks (*e.g.* DEC RA-81, IBM 3380).

2.1 The Seek Time:

In this section, we analyze the seek time of a shadowed system and how it relates to the parameters n and r.[3] We do not assume that any intelligent arm scheduling algorithm is used. Thus, we assume that the accesses are uniformly distributed over the disks diameter D. Notice that this assumption becomes invalid if a scheduling algorithm like the ones described in [5] is used. Also, we assume that the disks are asynchronous.[4]

Read Access:

To read a data block, the controller will have to wait for the first replica to become available. Since the number of disks involved in reading a replica is m, the expected seek time becomes the expected maximum of these m disk seeks.[5] Using the results in [4], we get:

$$\text{E(Group seek)}_{\text{read}} = \frac{D}{2}\frac{\Gamma(m+1)}{\Gamma(m+1.5)}\sum_{i=1}^{m}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

Each one of these groups can be approximated with a *logical disk* having the same expected seek time. For a single disk, however, the expected seek time is one third its maximum seek distance. Thus, the logical disk would have a maximum seek distance of:

$$D' = 3\frac{D}{2}\frac{\Gamma(m+1)}{\Gamma(m+1.5)}\sum_{i=1}^{m}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

The seek time for reading in a shadowed system is, thus, the minimum of the $r+1$ logical disks. Using the analysis in [4], we get:

$$\text{E(Shadow seek)}_{\text{read}} = \left(\frac{3}{2r+3}\right)\frac{D}{2}\frac{\Gamma(m+1)}{\Gamma(m+1.5)}\sum_{i=1}^{m}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

When tabulated [2], data from the above formulae show that for a constant level of fault-tolerance, the seek time tends to increase as the total number of disks per server increases. This is due to the fact that the number of disks per group increases, resulting in an increase in the expected seek time of the group (logical disk). Notice also that for a given number of disks per server, the expected seek time tends to decrease sharply as the level of fault-tolerance increases. This is due to two factors. First, the number of groups increases making the expected minimum seek time decrease. Second, the number of disks per group decreases making the expected seek time per group decrease as well.

Write Access:

To write a data block, the controller will have to wait for all of the disks (all groups and all disks per group) to respond. Using the analysis from [4], we get:

$$\text{E(Shadow seek)}_{\text{write}} = \frac{D}{2}\frac{\Gamma(n+1)}{\Gamma(n+1.5)}\sum_{i=1}^{n}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

The seek time for writing in a shadowed system is, thus, independent of the level of fault-tolerance. It increases as the total number of disks in the system increases.

When deriving the above estimate, we assumed that *all* the pieces (namely n) have to be written *before* committing the write request. That was necessary to maintain the same level of fault-tolerance throughout the whole operation. Another approach, however, would be to relax this restriction a little bit to improve performance. We illustrate this using the following example.

Assume that a shadowing system tolerates up to two faults by maintaining three replicas of each stripe. Using the usual (conservative) approach, we will have to write all three stripes for every write request. As we have shown above, this will cause the seek and latency delays to increase. In a sense, this performance degradation is the price we have to pay for the increased protection against failures. The question, of course, is whether this price is justified! For instance, if we commit the write request after writing only one copy (the first one to finish), then the seek time would be changed from 0.543 to 0.143, a decrease of about seventy-five percent. One might argue that, by doing so, the level of fault-tolerance is jeopardized, since a failure in the disk with the only copy (before the other two copies get to be written) might result in an unrecoverable error. Although true, this is unrealistic. First, the probability of having a failure in the interval of time between the completion of the first write and the second is extremely small. Second, even if that failure occur, the second and third writes can still go through unless the whole system fails (say as a result of a power failure). As a matter of fact, we can still tolerate such unlikely situation by adopting a more conservative solution where we commit the write request after two of the copies get written (the first two to finish). This will make the seek time 0.314, a decrease of over forty percent from the original write-all approach. Assuming the occurrence of two failures before the third write is completed is unquestionably overly pessimistic.

Read/Write Access:

For a read/write mix of α, the expected seek time is given by:[6]

$$\text{E(Shadow seek)}_{\text{read/write}} = \alpha\text{E}_{\text{read}} + (1-\alpha)\text{E}_{\text{write}}$$

[3] We assume that the seek time is dominant compared to other delays (*e.g.* latency and overhead).

[4] This assumption is pessimistic. In a number of situations, one might argue that other assumptions are more appropriate. For example, seek synchronization [4] is a more appropriate assumption if the head movement of the different disks are correlated.

[5] For seek synchronous systems, the expected seek time should be $\frac{D}{3}$ (see [4]).

[6] In typical applications, read requests tend to be much more frequent than write requests. However, the use of buffer caches makes the actual (physical) read/write mix more balanced.

2.2 The Transfer Time

Since the total number of disks per server is n, it follows that data can be striped into $m = n/(r+1)$ pieces. These pieces can be accessed in parallel, thus reducing the transfer time by a factor of $m = n/(r+1)$.

3 Parity

A major drawback of the shadowing approach is the excessive amount of redundancy needed to protect against failures. Instead of replicating data, redundancy can be added to the system so as to *correct* the erroneous information. In [7], [6] such an approach for tolerating a single failure in a group of N disks was suggested and termed *N+1 RAID*. The idea is to calculate and store parity information of a group of disks on a bit-per-disk basis. One parity block would be needed for every N blocks across the disks. Any single disk failure can be corrected simply by reading the rest of the disks in the group to determine what bit values of the failed disk would result in getting the proper parity.[7]

In this section, we generalize the parity approach to be able to tolerate more than one failure. We analyze the expected performance of such an organization and contrast it to the shadowing approach.

3.1 Level of redundancy and fault-tolerance

Let the number of data disks per server be m and assume that it is required to tolerate up to r simultaneous faults using the parity approach. The parity disks will partition the system into a number of parity groups. Each parity group consists of a number of data disks m and one parity disk.[8] Obviously, each one of the parity groups can tolerate at most one fault. To be able to tolerate up to r faults, we should have every data disk in at least r parity groups. If no more than r faults occur, then the data on any disk is recoverable since that disk is guaranteed to be in at least one parity group with at most one fault.

One way of organizing the data and parity disks to guarantee the above condition is to imagine that the data disks are organized in an r-dimensional space.[9] Obviously, every data disk will have r coordinates. Each one of these coordinates identifies a parity group that the data disk is a member of. For such an arrangement, one parity disk is needed per coordinate for each dimension. Let the number of parity groups (and thus the number of parity disks) needed to tolerate r faults be p. We have:

$$\begin{aligned} p &= r\lceil \sqrt[r]{m} \rceil^{r-1} \\ &\approx r m^{\frac{r-1}{r}} \end{aligned}$$

To tolerate one fault, we get $p = 1$ which corresponds to the *N+1 RAID* of [6]. To tolerate more faults, the number of parity disks grows rapidly. For example, to tolerate 2 simultaneous faults amongst 64 data disks, we need 16 parity disks (25 percent added redundancy). In this case, the parity groups consist of 8 data disks and one parity disk. To tolerate 3 simultaneous faults, however, we need 48 parity disks (75 percent added redundancy). In this case, the parity groups consist of 4 data disks and one parity disk.

3.2 Seek Time

In this section, we analyze the seek time of a parity system. However, before doing that, we will discuss how reading and writing in such a system is accomplished. We assume that data is striped over the m data disks.

To write a data block, all $m+p$ disks are accessed in parallel to write all the data and parity information to the disks. In case of no failures, the access would take as long as the slowest of the $m+p$ disks. In case of failures, data should be written on different sectors or on standby units. Depending on the seriousness of the failure, this process might be more complicated. To read a data block, all m disks are accessed in parallel. In case of no failures, the access would take as long as the slowest of the m disks. If failure(s) are detected, the system will have to:

[1] Lock all parity disks associated with the failed disk(s),
[2] Read all the necessary parity groups,
[3] Compute the data block for a correct parity,
[4] Write back the corrected data,
[5] Unlock the parity disks.

Since failures are not expected to be frequent, this performance overhead can be neglected on the average. In the above discussion, we have assumed that fine grain striping is used, making it necessary for every request to access all data disks. If coarse grain striping is used, it might be possible to fulfill the request by accessing *fewer* disks. For read requests, this might be an advantage, since the seek/latency delays are expected to be smaller. Write requests, however, become very expensive [7]. In particular, to write data to a single disk in a parity system, the system will have to:

[1] Lock all the parity groups associated with the disk,
[2] Read the old data,
[3] Read the old parity,
[4] Perform the write request,
[5] Compute and write the new parity using the formula:
New parity = (old data $\oplus$ new data) $\oplus$ old parity, and
[6] Unlock the parity disks.

[7] The N *information* disks along with the *parity* disk form a *parity group*.
[8] Notice that an information disk can be in more than one parity group.
[9] The cases where $r = 2, 3$ were examined in [6].

For a system with even a small percentage of writes, the parity disks will quickly become a performance bottleneck. Thus, in the following analysis, we will assume that fine grain striping is used.

Read Access:
Using the formula for the expected seek time for the maximum of m seeks [4], we get:

$$\text{E(Parity seek)}_{\text{read}} = \frac{D}{2}\frac{\Gamma(m+1)}{\Gamma(m+1.5)}\sum_{i=1}^{m}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

Write Access:
Using the formula for the expected seek time for the maximum of $n = m + p$ seeks [4], we get:

$$\text{E(Parity seek)}_{\text{write}} = \frac{D}{2}\frac{\Gamma(n+1)}{\Gamma(n+1.5)}\sum_{i=1}^{n}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

3.3 The Transfer Time

The transfer to/from the m data disks can be done in parallel, resulting in decreasing the transfer time by a factor of $m = n - p$.

4 The IDA Approach

Recently, Michael O. Rabin [8] devised a new algorithm for the secure and fault-tolerant communication and storage of information. The basic idea of this algorithm is to disperse the contents of a data block into n different pieces so that recombining *any* m of these pieces, $m <= n$, is sufficient to reconstruct the original data block. Figure 1 illustrates the dispersal of m stripes into n stripes using IDA. Obviously, such an approach would tolerate up to $n - m$ faults. The algorithm uses *irreducible polynomial arithmetic* to disperse or reconstruct the data. If the size of the data to be dispersed is S then the size of each of the pieces is $(1/m)S$, making the total required storage $(n/m)S$. The major aspect of IDA is that the added redundant information is not identifiable (as is the case with the parity approach). Rather, it is distributed among the data blocks. This makes the different pieces of information in the system *uniform*. That is, there is no distinction between data and parity.

4.1 Redundancy and Fault-tolerance

To tolerate up to r simultaneous faults, IDA requires that the total number of dispersed stripes exceeds the minimum number of stripes needed for reconstruction by r. Thus, a total of $n = m + r$ stripes is needed for every m stripes of data, a redundancy of $100(n-m)/m$ percent. Obviously, IDA requires the least amount of redundancy compared to shadowing and parity.

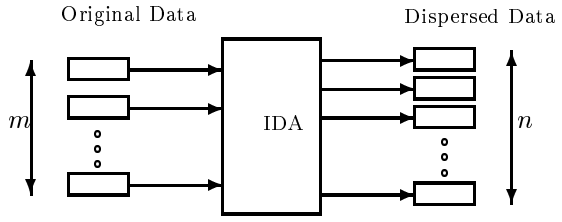

Figure 1: Data dispersal using IDA

4.2 The Seek Time

In this section, we analyze the seek time of IDA and how it relates to the parameters n and r. The estimates we get should be thought of as upper bounds. A number of optimizations (using intelligent layout of data stripes on the different cylinders and appropriate arm scheduling algorithms) are possible which would result in decreasing the seek times considerably.

Read Access:
To read a data block, all n disks are accessed in parallel and the first m of these to reply are used to reconstruct the data block. If no more than r failure(s) are detected, the system will have to recompute the missing stripes and rewrite them to different sectors or standby disks. Notice that this will not cause any delays for the read request. Using the formula for the expected seek time [4] we get:

$$\text{E(IDA seek)}_{\text{read}} = \frac{D}{2}\frac{\Gamma(n+1)}{\Gamma(n+1.5)}\sum_{i=n-m+1}^{n}\frac{\Gamma(i+0.5)}{\Gamma(i+1)}$$

Write Access:
To write a data block, all the n disks are accessed in parallel to write all the stripes. In case of no failures, the access would take as long as the slowest of the n disks. In case of a failure, data should be written on different sectors or on standby units. This, however, should not delay the commitment of the write request. Using the formula for the expected seek time for the maximum of the n disks [4], we get:

$$\text{E(IDA seek)}_{\text{write}} = \frac{D}{2}\frac{\Gamma(n+1)}{\Gamma(n+1.5)}\sum_{i=1}^{n}\Gamma(i+0.5)\Gamma(i+1)$$

Read/Write Access:
When tabulated [2], the seek time for the IDA approach for different read/write mixes reveals an important property. The larger the number of pieces dispersed, n , the worse the seek time of both read and write accesses for a constant level of redundancy. Table 1 shows this trend for different values of n. The write seek time depends only on n , whereas the read seek time depends on the available redundancy (we show the

read seek time for redundancy levels of 25, 50, and 75 percent). Despite this deterioration in the seek time, we might still favor a larger n to cut on the transfer time, especially with fairly large block sizes. Obviously, an optimum level of dispersion can be determined once the system parameters (block size, transfer rate, maximum seek time,... *etc.*) are given.

Expected seek time				
	Read access			Write access
n	$\frac{n-m}{n} = \frac{1}{4}$	$\frac{n-m}{n} = \frac{1}{2}$	$\frac{n-m}{n} = \frac{3}{4}$	
4	0.3905	0.2381	0.1111	0.5937
8	0.4384	0.2629	0.1216	0.7005
16	0.4671	0.2771	0.1275	0.7835
32	0.4830	0.2848	0.1307	0.8451
64	0.4914	0.2888	0.1323	0.8899
128	0.4956	0.2903	0.1331	0.9219
256	0.4978	0.2919	0.1336	0.9447

Table 1: Trend of read and write seek times

Notice that the figures given in Table 1 should be thought of as *upper bounds* since, as it will be explained later, much better write access performance can be achieved through intelligent arm scheduling, optimized layout of data blocks and adoption of a delayed write policy.

4.3 The Transfer Time

The transfer to (or from) the n (or m) disks can be done in parallel, resulting in decreasing the transfer time by a factor of m.

4.4 The Overhead Time

To assemble/disassemble blocks of data using IDA, some computations have to be done. If fine grain striping is used, this overhead can be kept very small. In [3], we presented an architecture for a VLSI chip to implement IDA in real-time. The chip has been fabricated by MOSIS and tested in the VLSI lab, Harvard University. The performance of the chip was measured to be about 1 megabyte per second. This corresponds, to an overhead of 1 micro-second for a stripe of 8 bits. By using proper pipelining and more elaborate designs, we believe that the overhead delay can be reduced significantly. Even, if hardware support is not available to assemble/disassemble the data, the overhead time will still be minimal compared to seek and latency delays. As a matter of fact, the overhead time of IDA becomes an issue only when it exceeds the transfer time of that block.

4.5 Arm Scheduling

In [5], arm scheduling algorithms were proposed for shadowed systems in order to decrease the seek delays for read requests. These algorithms are impossible to use with parity systems since there is no *choice* with respect to which disks to be accessed. With the IDA approach the system can select out of the n disks the best m disks to use for accessing the requested block. Notice that using such an algorithm for arm scheduling does not change our previous estimate of the seek time since the system is, in a way, *predicting* which disks will have the smallest seek time.

4.6 Delayed Write Optimization

The main advantage of the IDA approach is the symmetry of the different stripes. There is no distinction between data and redundancy. As we have shown earlier, this property can be exploited to enhance the performance of read requests. However, when writing the dispersed information to the different disks, we have assumed that the system has to wait until *all* the pieces are written before committing the write request. This restriction results in a degradation of the performance of the write requests compared to the read requests. In systems where disk cashing is used, the percentage of write requests is high. This means that the overall system performance is likely to deteriorate.

The requirement of writing all the pieces before committing the write request stems from the need to sustain the level of fault-tolerance. This might be necessary if that level is low. However, in a system where redundancy is introduced to boost performance as well as fault-tolerance, this restriction becomes questionable. To illustrate this point, consider a system where data blocks are dispersed into 16 pieces, of which any 8 are sufficient to reconstruct the original blocks. In such a system, the read seek time will be as low as $0.27D$, whereas the write seek time will be as high as $0.78D$. Such a system, tolerates up to 8 simultaneous faults. If we are willing to sacrifice this ultra-high level of fault-tolerance a little bit for a very short period of time, then a much better write seek time can be achieved. For instance, if we commit any write request after physically writing only 9 pieces, then the write seek time becomes $0.31D$, an improvement of more than half (actually over 60 percent). One might argue that, by doing so, the level of fault-tolerance is jeopardized since the system can only tolerate one failure right after the write request is committed. This concern is true but unrealistic. First, the probability of having two failures in the interval of time between the completion of the 9^{th} and 10^{th} writes is extremely small. Second, unless the failures are coordinated, the rest of the writes should proceed normally, thus restoring the level of fault-tolerance to its original level.

The flexibility of not writing all the pieces before committing a write request can also be exploited to increase the achievable concurrency. Again, we illustrate this by an example. Consider a system consisting of 64

disks where data blocks are dispersed into 16 pieces of which only 8 are sufficient to reconstruct the original data. If we insist on having all the physical writes go through before committing a write request then, obviously, the system can respond to *at most* 4 write requests at any time. On the other hand, if we adopt the policy of writing (say) only 9 of the pieces before committing a write request, then the system can service up to 7 write requests at any time. This would result in boosting the throughput of system by up to 75 percent and almost doubling the achievable concurrency. Moreover, the delayed writes can be done cheaply – for instance by waiting until the head of a disk is at a favorable position (using a scheduling algorithm similar to C-scan).

4.7 Dynamic Reconfiguration

One of the major advantages of using IDA is that the amount of redundancy (and thus the level of fault-tolerance and the performance gains) can be adjusted for the different needs of the system. For instance, critical files would be assigned a higher level of redundancy thus insuring ultra-high accessibility and better performance. On the other hand, less critical data can be assigned lower levels of redundancy (or even no redundancy at all). This flexibility is impossible to insure with parity systems. Another main advantage of using IDA is that, with a careful design, it is possible to add new disks to the system (and thus increase its capacity) dynamically. This is also possible with shadowed systems. For parity systems, however, the distinction between parity information and data makes such a reconfiguration more difficult.

5 Comparison

We will compare the three approaches we have presented (shadowing, parity and IDA) using typical examples. The examples and figures we will be presenting are only *suggestive*. As we have discussed in [4], the actual performance depends heavily on other factors as well.[10]

First, we will examine all three approaches for a fixed level of fault-tolerance. We will compare the required redundancy and the achievable performance. Next, we will fix the total capacity and the percentage of redundancy and compare the achievable performance and fault-tolerance.

5.1 Fixed Fault-tolerance

Consider an example system consisting of $n = 24$ disks. We will examine two situations. First, we will assume that it is required to tolerate one failure out of the 24 disks ($r = 1$). Next, we will assume that it is required to tolerate up to two simultaneous failures ($r = 2$). The conclusions we will draw out of this example can be easily extended for larger values of n or r.

Example-1: ($n = 24$,$r = 1$)
Table 2 shows the level of redundancy, the transfer speedup and the expected seek time for all three striping approaches. The expected seek time is calculated for different read/write mixes of 0, 20, 50, 80 and 100 percent reads.

Example-1 ($n = 24, r = 1$)			
	Striping approach		
	Shadowing	Parity	IDA
Redundancy (percent)	100.0	4.35	4.35
Transfer speedup	12.0	23.0	23.0
Seek Time (α= 0.0)	0.82	0.82	0.82
Seek Time (α= 0.2)	0.75	0.80	0.80
Seek Time (α= 0.5)	0.64	0.78	0.78
Seek Time (α= 0.8)	0.52	0.75	0.75
Seek Time (α= 1.0)	0.45	0.73	0.73

Table 2: Comparison for $n = 24$,$r = 1$

From this table, we notice that the amount of redundancy for a shadowed system is much higher than that of a parity system or a system using IDA. This means a much smaller capacity out of the available space. This also means that less parallelism is available for striping. This is reflected in the lower transfer speedup of shadowed systems when compared to either parity systems or IDA systems. For write accesses, the seek time is the same for all three approaches. The seek time for read accesses, however, is different; shadowing is the best followed by IDA and then parity.

Example-2: ($n = 24$,$r = 2$)
Table 3 shows the level of redundancy, the transfer speedup and the expected seek time for all three striping approaches. The expected seek time is calculated for different read/write mixes of 0, 20, 50, 80 and 100 percent reads.

5.2 Fixed Redundancy

In the above examples, we have fixed the level of fault-tolerance. Our conclusion was that IDA requires a much lesser amount of redundancy compared to the other approaches. What if IDA is allowed to use as much redundancy as does shadowing or parity? What would be the achievable performance and fault-tolerance gains?

Example-3: ($n = 8$,$m = 4$)
Consider an example system consisting of 8 disks where up to 4 can be used for redundant storage. When comparing the cost per megabyte for small disks (5 to 10 Dollars/MB) with that for large disks (20 to 45 Dol-

[10] Organization, Operating system optimizations, Tuning of parameters, *etc.*

Example-2 ($n = 24$,$r = 2$)			
	Striping approach		
	Shadowing	Parity	IDA
Redundancy (percent)	200.0	50.0	9.1
Transfer speedup	8.0	16.0	22.0
Seek Time (α= 0.0)	0.82	0.82	0.82
Seek Time (α= 0.2)	0.71	0.81	0.79
Seek Time (α= 0.5)	0.56	0.80	0.74
Seek Time (α= 0.8)	0.40	0.79	0.70
Seek Time (α= 1.0)	0.29	0.78	0.67

Table 3: Comparison for $n = 24$,$r = 2$

lars/MB), this level of redundancy is acceptable. Assume that the data is to be striped on the other 4 disks so as to achieve a transfer speedup of 4. With shadowing, 4 disks will be used to hold the redundant copy of the striped blocks. Thus every stripe will be stored on two different disks, thus, tolerating one erasure. To read a data block, we will have to wait for1out of every 2 copies to become available. With parity, the four data disks will be arranged in two dimensions, with two parity disks for row-parity and two parity disks for column-parity, thus, tolerating up to2erasures. To read a data block, we will have to wait for *all* of the 4 data disks to respond. To write a data block, we will have to write *all* of the data and parity disks. With IDA, data blocks will be dispersed into 8 pieces such that *any* 4 will suffice to reconstruct the original data. To read a data block, we will have to wait for the first 4 pieces (*any* 4 pieces) to be available. To write a data block, we will have to write all of the 8 pieces. Table 4 shows a comparison of the relative performance and fault-tolerance of the three approaches.

Example-3 ($n = 8$,$m = 4$)			
	Striping approach		
	Shadowing	Parity	IDA
Tolerable Faults	1	2	4
Seek Time (α= 0.0)	0.70	0.70	0.70
Seek Time (α= 0.2)	0.63	0.68	0.61
Seek Time (α= 0.5)	0.53	0.65	0.48
Seek Time (α= 0.8)	0.43	0.61	0.35
Seek Time (α= 1.0)	0.36	0.59	0.26

Table 4: Comparison for $n = 8$ and $m = 4$

From this table, we notice that IDA is superior to both parity and shadowing in terms of level of fault-tolerance. Its performance is better (although close) to shadowing and a lot better than parity (especially when the read percentage is high).

5.3 Performance optimization

In the example above, we have fixed the level of redundancy and sought the highest possible level of fault-tolerance with any improvement in the performance seen as a *side effect*. As we have discussed earlier, one might add redundancy to the system to enhance its performance in the first place! What would be the achievable performance for a given level of redundancy and a given level of fault-tolerance?

Example-4: ($n = 8$,$m = 4$)
Consider an example system consisting of 8 disks where up to 4 disks can be used for redundant storage. Also, assume that the system is required to tolerate at least one erasure at any point in time. With a pure shadowing approach, the best we can do is to use all of the available redundancy to guarantee the required level of fault-tolerance in an organization identical to that used in Example-3 (4 disks for data and 4 disks for the shadow). With parity, one can suggest an organization similar to the one in Example-3, except that a delayed write policy is used with one of the groups of parity disks (row-parity or column-parity). In this case, to write a data block, we will have to wait until 6 writes terminate (instead of 8). With IDA, we can use a delayed write policy with an arrangement identical to that of Example-3. To write a data block, we will have to write any 5 pieces.

Table 5 shows a comparison of the relative performance and fault-tolerance of the three approaches. We tabulate two measures for fault-tolerance: transient and steady state. The transient level of fault-tolerance reflects the reliability of the system before the delayed writes terminate, whereas the steady-state level of fault-tolerance reflects the eventual reliability once the delayed writes are finished. From Table 5, we notice that IDA is superior to both parity and shadowing in terms of the achievable performance.

Example-4 ($n = 8$,$m = 4$)			
	Striping approach		
	Shadowing	Parity	IDA
Tolerable Faults (transient)	1	1	1
Tolerable Faults (eventual)	1	2	4
Seek Time (α= 0.0)	0.70	0.66	0.34
Seek Time (α= 0.2)	0.63	0.65	0.32
Seek Time (α= 0.5)	0.53	0.63	0.30
Seek Time (α= 0.8)	0.43	0.60	0.28
Seek Time (α= 1.0)	0.36	0.59	0.26

Table 5: Comparison for $n = 8$ and $m = 4$

In all of the above examples, we have assumed that only one of the three approaches is used. Is it possible that a combination of (say) parity and shadowing might be superior (in terms of performance and/or reliability) when compared to IDA ? The answer to this question

is not obvious. For instance, we have established that when tolerating up to 1 erasure, IDA and parity are identical in terms of the required redundancy and the achievable performance. What if we use parity to tolerate failures and shadowing to improve performance ? Our answer is *No*. To illustrate this, we consider the following example:

Example-5: ($n = 10$,$m = 4$)
Consider a system consisting of 10 disks where up to 6 disks can be used for redundant storage. Also, assume that the system is required to tolerate at least one erasure at any point in time. Using IDA, the solution is obvious: disperse data blocks into 10 pieces of which any 4 would be sufficient and use a delayed write strategy. One can imagine another arrangement where data is striped over 4 disks and shadowed onto another 4 disks. To tolerate 1 erasure, a parity disk will be used with every group of 4 data disks. Table 6 shows the achievable performance and fault-tolerance for these two alternatives.

Example-5 ($n = 8$,$m = 4$)		
	Striping approach	
	Shadowing & Parity	IDA
Tolerable faults (transient)	1	1
Tolerable faults (eventual)	2	6
Seek Time (α= 0.0)	0.350	0.270
Seek Time (α= 0.2)	0.356	0.260
Seek Time (α= 0.5)	0.365	0.240
Seek Time (α= 0.8)	0.374	0.220
Seek Time (α= 1.0)	0.380	0.210

Table 6: IDA compared to parity + shadowing

From this table, we notice that IDA is still superior to the combination of parity and shadowing in terms of the achievable performance and reliability.

6 Conclusion and Future Work

IDA-based design of RAID systems is provably optimal with respect to the amount of added redundancy and the achievable level of fault-tolerance. Its main advantage is its indifference in dealing with data and redundancy. This makes it possible to make more use of the added redundancy to enhance performance. Other advantages of using IDA include: intelligent arm scheduling, dynamic reconfiguration and controllable redundancy.

Previous RAID research [7] used redundancy primarily to ensure acceptable levels of fault-tolerance. Performance gains (if any) were viewed as a side effect. In our opinion, this should be reversed. Redundancy should be used primarily to boost performance with fault-tolerance being a byproduct. In this respect, we have demonstrated that IDA is the right choice. With an affordable level of redundancy[11] IDA can be used to reduce disk access delays and improve concurrency while insuring ultra-high levels of reliability.

More work remains to be done in order to incorporate IDA with a RAID design. This includes designing and possibly implementing appropriate hardware prototypes, investigating the potential support available from operating systems, and analyzing the requirements of different applications in order to fine tune system parameters for an optimized performance.

Acknowledgments: I would like to thank Prof. Michael O. Rabin from Harvard University and Wing Wong and Danny Chen from AT&T Bell Labs for their help and suggestions.

References

[1] K.H. Bates and M. TeGrotenhuis. Shadowing boosts system reliability. *Computer Design*, April 1985.

[2] Azer Bestavros. IDA-based disk arrays. Technical Memorandum 45312-890707-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, July 1989.

[3] Azer Bestavros. SETH: A VLSI chip for the real-time information dispersal and retrieval for security and fault-tolerance. In *Proceedings of ICPP'90, The 1990 International Conference on Parallel Processing*, Chicago, Illinois, August 1990.

[4] Azer Bestavros, Danny Chen, and Wing Wong. The reliability and performance of parallel disks. Technical Memorandum 45312-891206-01TM, AT&T, Bell Laboratories, Department 45312, Holmdel, NJ, December 1989.

[5] Dina Bitton. Arm scheduling in shadowed disks. In *Proceedings of COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference*, March 1989.

[6] Garth Gibson, Lisa Hellerstein, Richard Karp, Randy Katz, and David Patterson. Coding techniques for handling failures in large disk arrays. Technical Report UCB/CSD 88/477, Computer Science Division, University of California, July 1988.

[7] David A. Patterson, Peter Chen, Garth Gibson, and Randy H. Katz. Introduction to redundant arrays of inexpensive disks (RAID). In *Proccedings of COMPCON-89, the Thirty-fourth IEEE Computer Society International Conference*, March 1989.

[8] Michael O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[9] Pascal Zachary. Disk arrays may reshape storage of computer data. *The Wall Street-Journal (Technology)*, July 19th 1989.

[11] comparable to what shadowing or even parity will require to achieve a modest level of fault-tolerance.