DISTRIBUTED RAID --
A NEW MULTIPLE COPY ALGORITHM

by

Michael Stonebraker

Memorandum No. UCB/ERL M89/56

15 May 1989

Distributed RAID -- A New Multiple Copy Algorithm

by
Michael Stonebraker

Memorandum No. UCB/ERL M89/56

15 May 1989

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California
Berkeley, CA 94720

DISTRIBUTED RAID -- A NEW MULTIPLE COPY ALGORITHM

Michael Stonebraker
EECS Dept.
University of California, Berkeley

Abstract

All previous multicopy algorithms require additional space for redundant information equal to the size of the object being replicated. This paper proposes a new multicopy algorithm with the potentially attractive property that much less space is required and equal performance is provided during normal operation. On the other hand, during failures the new algorithm offers lower performance than a conventional scheme. As such, this algorithm may be attractive in various multicopy environments as well as in disaster recovery. This paper presents the new algorithm and then compares it against various other multicopy and disaster recovery techniques.

1. INTRODUCTION

In a recent paper, the concept of a RAID (Redundant Array of Inexpensive Disks) was introduced [PATT88]. Such disk systems have the desirable property that they survive disk crashes and require only 1 extra disk for each group, G of disks. Hence, the space cost of high availability is only 100/G percent, a modest amount compared to traditional schemes which mirror each physical disk at a space cost of 100 percent.

In this paper we extend the RAID concept to a distributed computing system and call the resulting construct RADD (Redundant Array of Distributed Disks). RADDs will be shown to support redundant copies of data across a computer network at the same space cost as RAIDs do for local data. Such copies will be shown to increase availability in the presence of both temporary and permanent failures of single

This research was sponsored by the National Science Foundation under Grant MIP-8715235 and by a Grant from IBM Corp.

computer systems as well as disk failures. As such, RADDs should be considered as a possible alternative to traditional multiple copy techniques such as surveyed in [BERN81]. Moreover, RADDs are also candidate alternatives to high availability schemes such as **hot standbys** [GAWL85].

In Section 2, we begin by briefly reviewing a Level 5 RAID from [PATT88], which is the system we generalize. Then, in Section 3 we discuss our model of a distributed computing system and indicate the basic structure of a RADD. Section 4 continues with an extension of the model to deal with unequal disk capacity at the various sites. Then, Section 5 deals with network unreliability, while Section 6 considers the relationship of RADD to a distributed DBMS. Lastly, Section 7 closes with a performance comparison between RADD and several other multiple copy and high availability schemes.

2. RAID

A RAID is composed of a group of G data disks plus one parity disk and an associated I/O controller which processes requests to read and write disk blocks. All G+1 disks are assumed to be the same size, and a given block on the parity disk is associated with the corresponding data blocks on each data disk. This parity block always holds the bit-wise parity calculated from the associated G data blocks.

On a read to a functioning disk, the RAID controller simply reads the object from the correct disk and returns it to the attached host. On a write to a functioning disk, the processing is somewhat more complex because the controller must update both the data block and the associated parity block. The data block, of course, is simply overwritten. However, the parity block must be updated as follows:

parity-block = old-parity-block XOR (new-data-block XOR old-data-block) (1)

Here XOR is the bitwise exclusive OR of two objects, the old parity block and the XOR between the new data block and its old contents. Intuitively, whenever a data bit is toggled, the corresponding parity bit must also be toggled.

Using this architecture, a read has no extra overhead while a write may cost 2 physical reads-modify-write accesses. However, careful buffering of the old data block can remove one of the reads and prefetching the old parity block can remove the latency delay of the second read. A RAID can support as many as G parallel reads but only a single write because of contention for the parity disk. In order to overcome this last bottleneck, [PATT88] suggests striping the parity over all G + 1 drives so that each physical

drive has 1/(G + 1) of the parity data. In this way, up to G/2 writes can occur in parallel though a single RAID controller. This striped parity proposal is called a Level 5 RAID in [PATT88].

If a head crash or other disk failure occurs, the following algorithm must be applied. First, the failed disk must be replaced with a spare disk either by having an operator mechanically replace the failed component or by having a G + 2nd spare disk associated with the group. Alternately, spare drives can be economized by having RAIDs share spare drives. Then, a background process is performed to read the other G disks and reconstruct the failed disk onto the spare. For each corresponding collection of blocks, the contents of the block on the failed drive is:

failed block = XOR {other blocks in the group} (2)

If a read occurs before reconstruction is complete, then the corresponding block must be reconstructed immediately according to the above algorithm. A write will simply cause a normal write to the replacement disk and its associated parity disk. Algorithms to optimize disk reconstruction have been studied in [COPE89, KATZ89].

In order for a RAID to lose data, a second failure must occur while recovering from the first one. Since the mean time to failure of single disk is typically in excess of 30000 hours (about 4 years) and recovery time can easily be contained to an hour, then the mean time to failure of a RAID with G = 10 exceeds 50 years.

Hence, we will assume that a RAID is tolerant to disk crashes. As such it is an alternative to conventional mirroring of physical disks, such as is done by several vendors of computer systems. An analysis of RAIDs is presented in [PATT88] that indicates that a RAID offers performance only slightly inferior to mirroring but with vastly less physical disk space.

On the other hand, if a site fails permanently (because of flood, earthquake, or other disaster), then a RAID will also fail. Hence, a RAID offers no assistance with site disasters. Moreover, if a site fails temporarily, because of a power outage, a hardware failure or a software failure, then the data on a RAID will be unavailable for the duration of the outage. In the next section we extend the RAID idea to a multi-site computer network and demonstrate how to provide space-efficient redundancy that increases availability in the presence of temporary or permanent site failures as well as disk failures.

3. RADD

3.1. Introduction

Consider a collection of G + 2 independent computer systems, S[0], ..., S[G+1], each performing data processing on behalf of its clients. The sites are not necessarily participating in a distributed data base system or other logical relationship between sites. Each site has one or more processors, local memory and a disk system. The disk system is assumed to consist of some number, N, physical disks each with B blocks. These N * B blocks can be managed by the local operating system or I/O controller and have the following composition:

N * B * G / (G + 2)	data blocks
N * B / (G + 2)	parity blocks
N * B / (G + 2)	spare blocks (or a lesser number)

Informally, data blocks are used to store local site data. Parity blocks are used to store parity information for data blocks at other sites. Furthermore, spare blocks are used to reconstruct the logical disk at some site if a site failure occurs. Loosely speaking, these blocks correspond to data, parity and spare blocks in a RAID. Also, it will be possible to allocate a lesser number of spare blocks at each site at the expense of lower availability. This issue is discussed further in Section 7.2.

In the following discussion it may be helpful to refer to Figure 1 where we show the layout of data. parity and spare blocks for the case G = 4. The Ith row of the figure shows the composition of physical block I at each site. In each row there is a single P which indicates the location of the parity block for the remaining blocks. There is also a single S in each row which indicates the location of the spare block which will be used to store the contents of the block if another site is temporarily or permanently down. The remainder of the blocks are used to store data and are numbered 0,1,2,... at each site.

In the rest of this section we will discuss our algorithms in terms of reading and writing physical blocks, noting that they are logically interpreted as in Figure 1. Also, user reads and writes are directed at data blocks and not parity or spare blocks. Lastly, we assume that the network is reliable. In other words, there are no network failures and a message, once delivered to the network manager, will be received by the destination machine. Unreliable network communication is treated in Section 5.

	S[0]	S[1]	S[2]	S[3]	S[4]	S[5]
block 0	P	S	0	0	0	0
block 1	0	P	S	1	1	1
block 2	1	0	P	S	2	2
block 3	2	1	1	P	S	3
block 4	3	2	2	2	P	S
block 5	S	3	3	3	3	P

The Logical Layout of Disk Blocks

Figure 1

We assume that there are three kinds of failures, namely:

1) disk failures
2) temporary site failures
3) permanent site failures (disaster)

In the first case, a site continues to be operational but loses one of its N disks. The other disks continue to function normally and the site remains operational, albeit short B blocks. The second type of failure occurs when a site ceases to operate temporarily. After some repair period the site becomes operational and can access its local disks again. The third failure is a site disaster. In this case the site may be restored after some repair period but all information from all N disks is lost. This case typically results from fires, earthquakes and other natural disasters, in which case the site is usually restored on alternate or replacement hardware.

Consequently, each site in the network is in one of three states:

up	functioning normally
down	not functioning
recovering	running recovery actions

A site moves from the **up** state to the **down** state when a temporary site failure or site disaster occurs. After the site is restored, there is a period of recovery, after which normal operations are resumed. A disk failure will move a site directly from **up** to **recovering**. The protocol by which each site obtains the state of all other sites is straightforward and is not discussed further in this paper [ABBA85].

Our algorithms attempt to recover from single site failures, disk failures and disasters. No attempt is made to survive multiple failures.

3.2. Algorithms

Each site is assumed to have a source of unique identifiers (UIDs) which will be used for concurrency control purposes in the algorithms to follow. The only property of UIDs is that they must be globally unique and never repeat. For each data and spare block, a local system must allocate space for a single UID. On the other hand, for each parity block the local system must allocate space for an array of G + 2 UIDs.

If system S[J] is **up**, then the Ith data block on system S[J] can be read by accessing the Kth physical block according to Figure 1. For example, on site S[1], the Kth block is computed as:

$$K = (G + 2) * \text{quotient}(I / G) + \text{remainder}(I / G) + 2$$

The Ith data block on system S[J] is written by obtaining a new UID and:

W1) writing the Kth local block according to Figure 1 together with the obtained UID.

W2) computing A = remainder (K / (G + 2))

W3) sending a message to site A consisting of:

a) the block number K
b) the bits in the block which changed value (the change mask)
c) the UID for this operation

W4) When site A receives the message it will update block K, which is a parity block, according to formula (1) above. Moreover, it saves the received UID in the Jth position in the UID array discussed above.

If System S[J] is **down**, other sites can read the Kth physical block on system S[J] in one of two ways, and the decision is based on the state of the spare block. Each data and spare block has two states:

valid	non-zero UID
invalid	zero UID

Consequently, the spare block is accessed by reading the Kth physical block at site S[A'] determined by:

$$A' = \text{remainder}((K + 1) / (G + 2))$$

The contents of the block is the result of the read if the block is valid. Otherwise, the data block must be

reconstructed. This is done by reading block K at all up sites except site S[A'] and then performing the computation noted in formula (2) above. The contents of the data block should then be recorded at site A' along with a new UID obtained from the local system to make the block valid. Subsequent reads can thereby be resolved by accessing only the spare block.

If site S[J] is down, other sites can write the Kth block on system S[J] by replacing step W1 with:

> W1') send a message to site S[A'] with the contents of block K indicating it should write the block.

If a site S[J] becomes operational, then it marks it state as recovering. To read the Kth physical block on system S[J] if system S[J] is recovering, the spare block is read and its value is returned if it is valid. Otherwise, the local block is read and its value is returned if it is valid. If both blocks are invalid, then the block is reconstructed as if the site was down. As a side effect of the read, the system should write local block K with its correct contents and invalidate the spare block. If site S[J] is recovering, then writes proceed in the same way as for up sites. Moreover, the spare block should be invalidated as a side effect.

A recovering site also spawns a background process to lock each valid spare block, copy its contents to the corresponding block of S[J] and then invalidate the contents of the spare block. In addition, when recovering from disk failures, there may be local blocks that have an invalid state. These must be reconstructed by applying formula (2) above to the appropriate collection of blocks at other sites. When this process is complete, the status of the site will be changed to up.

3.3. Concurrency Control

During normal operations, any concurrency control scheme can be used. However, we will assume that dynamic locking is employed. Hence, reads and writes set the appropriate locks on each data block that they read or write. If a site is down, then read and write locks are set on the spare block which exists at some site which is up. Parity blocks are never locked. If the spare block is valid, no further special treatment must be performed.

If the spare block is not valid, then it must be locked as above and then reconstructed by remote reads of G other blocks. These reads can be performed with no additional locking; however, each read operation must also return the UID of the stored block. The parity block must return its array of UIDs.

Each UID must be compared against the corresponding UID in the array for the parity block. If all UIDs match, then formula (2) constructs the correct contents of the block. If any UIDs fail to match, then the read was not consistent and must be retried.

3.4. Crash Recovery

The algorithms we have described operate at the file system level. Hence, it is necessary to discuss crash recovery in three different contexts:

> 1) the DBMS performs transaction management through write-ahead-log (WAL) techniques. The file system has no knowledge of transactions.
>
> 2) the DBMS performs transaction management through a no-overwrite scheme. Again, the file system is ignorant of transactions.
>
> 3) the operating system performs transaction management.

In the case that the DBMS is using a WAL scheme [GRAY78], there is a significant problem. If site S[J] fails, other sites can employ the algorithms above to reconstruct the contents of the failed file system at the time of the crash onto spare blocks. However, there may well be writes from uncommitted transactions that have been recorded as well as writes from committed transactions that have not yet been recorded.

Consequently, the blocks of S[J] must first be restored to a consistent state by DBMS recovery code before they can be accessed. This will require running the standard two-phase recovery algorithm over the log which was written by the local DBMS [HAER83]. Unfortunately, the log must also be reconstructed according to the algorithms noted above. As a result, each block accessed during the recovery process will require G physical reads at various sites.

In the case of a temporary site failure, local recovery can often be initiated rather quickly. For example, recovery will be commenced immediately for software failures. In this case, only one local read need be done for each block accessed during the recovery process. Therefore, remote recovery is unlikely to finish before local recovery has been completed. Consequently, in the common case of site failures of short duration, a standard WAL technique used in conjunction with a RADD is unlikely to increase availability.

As a result, RADDs are more appropriate for site disasters and disk failures in this environment. To make them useful for site failures, very fast recovery is required. There are a multitude of options to move

in this direction; however the best technique appears to be a no-overwrite storage manager and we now turn to this scenario.

POSTGRES [WENS88, STON86] supports a storage manager in which data is not overwritten [STON87]. In this architecture, there is no concept of processing a log at recovery time. Hence, if a site failure occurs, then remote operations can proceed according to the algorithms in Section 3 with no intervening recovery stage. Hence, a RADD will work well for site failures as well as site disasters and disk failures if a no-overwrite storage manager is used.

If transaction management is supported within the operating system, then the above considerations precisely apply. If the operating system uses a WAL, as in the 801 project [CHAN87] or Camelot [SPEC87], then a RADD will not work well on short-duration site failures. On the other hand, if the operating system does not overwrite blocks, as in [OUST88], a RADD will perform well on all three kinds of failures.

4. NON UNIFORM DISK SYSTEM SIZES

In the previous sections we have assumed that each site has N disks each of B blocks capacity. In this section, we indicate that this assumption is straightforward to relax. Specifically, assume that there are L sites, L > G + 2, with numbers of disks N[0], N[1], ..., N[L-1]. Furthermore assume the total number of drives at all sites is equal to A * (G+2), for some constant A. Lastly, assume that no site has more than A drives. The goal is allocate the collection of A * (G+2) drives into A groups of G+2 drives each such that the G+2 drives are all on different sites. In this way, we can run the algorithms of Section 3 on each group individually.

The following simple algorithm shows how to meet this goal. Pick a single drive from each of the G+2 sites with the largest number of drives, and put these drives into the first group. If multiple sites have the same number of drives, then resolve the tie in some arbitrary way. Since no site has more than A drives, there must be at least G+2 sites with a drive. There are now N'[0], ..., N'[L-1] drives at each site totaling (A-1) * (G+2) drives. Moreover, no site has more than A-1 drives. Consequently, the problem definition is the same as before. Repeat the the picking algorithm iteratively until there are no drives left.

This algorithm is also straightforward to extend to cover non uniform disk sizes. Simply group disk blocks at each site into logical drives of size B, B > 0 blocks. Then use the above algorithm to allocate these logical drives to RADD groups. Assuming that B divides the total number of blocks at each site, the algorithm will construct a successful RADD with no wasted blocks.

5. NETWORK FAILURES

We now indicate how to extend the model of Section 3 to deal with network failures of two kinds:

network partitions
lost messages

In the case of network partitions, we assume that the sites divide into two or more mutually exclusive collections that can communicate within individual partitions but not across partition boundaries. If the partition looks like a single failure, e.g. there are two collections with respectively G + 1 and 1 site, then the algorithms of Section 3 apply to the partition with G + 1 members. As long as the singleton site ceases processing, consistency is guaranteed. Any other network partition looks like a multiple site failure and is not addressed by the RADD algorithms. In this case, the system must block by denying access to data blocks until the failure is repaired.

If messages can be lost, then software at each site must ensure that the following conditions are true before a transaction can be committed at that site:

1) the messages updating the parity block for all writes performed on behalf of the transaction have been received at the various parity sites

2) the message corresponding to transaction commit has been received at its parity site

Enforcing these conditions may entail a substantial performance penalty. However, they will allow a RADD to operate correctly when messages can be lost.

Hence, network failures that look like single site failures can be tolerated along with lost packets (at some performance penalty). Other partitions require a RADD to block awaiting reconnection.

6. DISTRIBUTED DATA BASES

If a distributed DBMS is the client of a RADD, then it must take the following actions. First, query optimization can proceed with no consideration of multiple copies. The resulting heuristic plan is meant to

be executed at several sites. If the site at which a plan is supposed to execute is **up** or **recovering**, then the plan is simply executed at that site. If the site is **down**, then the plan is allocated to some other convenient site.

Distributed concurrency control can be done using any of the common techniques. The algorithms above do not appear to impact any of the common algorithms.

A two-phase commit is commonly used to raise the probability that all sites commit or all sites abort a distributed transaction [SKEE81]. Nothing about our algorithm interferes with such commit processing.

Moreover, in a RADD one can question the need for a two phase commit. Commonly, one has a **master** module coordinating a collection of **slave** modules, one at each site where processing needs to take place. The master sends commands to various slaves who reply **done** when they are ready for more input. For each local write that is made by a slave into the local buffer pool, the algorithms of Section 3 require a non-local write to the parity block. If the message for each such write is sent and received reliably before the slave returns **done**, then a slave can crash any time after returning **done**, and the information written in the buffer pool is recoverable. Each slave is thereby **prepared** after each command, and the coordinator can issue a **commit** at any time after he has received a **done** from all slaves. Consequently, if the network is reliable, only a single failure occurs, and the messages to update the parity information are sent before returning **done**, then a two-phase commit does not increase the probability of avoiding inconsistent data. The second and third assumptions are plausible; hence if the network fails infrequently then it may not be worthwhile to use a two-phase commit in a RADD environment.

7. PERFORMANCE

7.1. Introduction

In this section we compare the performance of a RADD against five other possible schemes that give higher availability. The first is a traditional multiple copy algorithm. Here, we restrict attention to the case where there are exactly two copies of each object. In this case, any voting scheme reduces to something equivalent to a Read-One-Write-Both (ROWB) scheme [ABBA85]. In fact, ROWB is essentially the same as a RADD with a group size of 1 and no spare blocks. The second comparison is with a Level 5 RAID as discussed in [PATT88]. Third, we examine a composite RAID in which the RADD algorithms are being

performed as in Section 3. In addition, the single site RAID algorithms are also applied to each local I/O operation, transparent to the higher level RADD operations. This combined RAID will be called C-RAID. It is also possible to utilize a two dimensional RADD. In such a system the sites are arranged into a two-dimensional array and a row parity and column parity are constructed, each according to the formulas of Section 3. We call this scheme 2D-RADD, and a variation on this idea was developed in [GIBS89]. The last case which we examine is a RADD in which the group size is one-half as large as normal. Such a system, named 1/2-RADD will require the same amount of disk space as a C-RAID and may be an interesting alternative if extra space is available.

The comparison will use three metrics. First, we consider the space overhead of each scheme. Next, we document the cost of read and write operations for each scheme under various conditions. We conclude the performance discussion by considering network bandwidth issues. The third issue is reliability, and we document two metrics for each system. The first metric is the mean time to the unavailability of a specific data item, MTTU. This quantity is the mean time until the particular data item is unavailable because the algorithms must wait for some site failure to be repaired. The second metric is the mean time until the system irretrievably loses data, MTTF. This quantity is the mean time until there exists a data item that cannot be restored. The next four subsections document our results.

7.2. Space Requirements

Space requirements are determined solely by the group size, G, that is used, and for the remainder of this paper we assume that G = 8. Furthermore, it is necessary to consider briefly our assumption about spare blocks. The algorithms in Section 3 were constructed assuming that there is one spare block for each parity block. During any failure, this will allow any block on the down machine to be written while the site is down. Alternately, it will allow one disk to fail in each disk group without compromising the ability of the system to continue with write operations to the down disks. Clearly, a smaller number of spare blocks can be allocated per site if the system administrator is willing to tolerate lower availability.

In our analysis we assume there is one spare block per parity block. Analyzing availability for lesser numbers of parity blocks is left as a future exercise. Figure 2 indicates the space overhead of each scheme under the assumptions above. Clearly, the traditional multiple copy algorithm requires a 100 percent space

System	Space Overhead
RADD	25 percent
ROWB	100 percent
RAID	25 percent
C-RAID	56.25 percent
2D-RADD	50 percent
1/2-RADD	50 percent

A Space Comparison
Figure 2

penalty since each object is written twice. Since G = 8 and we are also allocating a spare block for each parity block, then the parity schemes (RAID and RADD) require two extra blocks for each 8 data blocks, i.e. 25 percent. The space overhead for a RADD with group size of 4 (1/2-RADD) is twice as high, i.e. 50 percent. For each 64 disks in a two-dimensional array, the 2D-RADD requires two collections of 16 extra disks. Hence, the space overhead is 50 percent. The C-RAID requires two extra disks for each 8 data disks for the RADD algorithm. In addition, the 10 resulting disks need 2.5 disks for the local RAID algorithms. Hence, the total space overhead is 56.25 percent.

7.3. Cost of I/O Operations

In this subsection we indicate the cost of read and write operations for the various systems. In the analysis we use the constants in Table 1 below, and Figure 3 shows the number of local and remote opera-

Parameter	cost
local read	R
local write	W
remote read	RR
remote write	RW

Some Cost Parameters
Table 1

tions required by the systems to perform reads and writes under various circumstances. Under normal circumstances when all sites are up, all systems read data blocks by performing a single local read. A normal write requires 2 actual writes in all cases except C-RAID and 2D-RADD. A local RAID requires two local writes, while RADD, ROWB, and 1/2-RADD all need a local write plus a remote write. A C-RAID requires a total of 4 writes. The RADD portion of the algorithm performs a local write plus a remote write. However, each will be turned into two actual writes by the RAID portion of the algorithm. We count this as 3 local writes plus one remote write. In a 2D-RADD, the RADD algorithm must be run in two dimensions, resulting in two remote writes and one local write.

If a disk failure occurs, all parity systems must reconstruct the desired block. In each case, they must read all other blocks in the appropriate disk group. These are local operations for RAID and remote operations for RADD, 2D-RADD and 1/2-RADD. Lastly, a C-RAID can use the local RAID portion of the

System	RADD	ROWB	RAID	C-RAID	2D-RADD	1/2-RADD
no failure read time	R	R	R	R	R	R
no failure write time	W+RW	W+RW	2*W	RW+3*W	W+2RW	W+RW
disk failure read time	G*RR	RR	G*R	G*R	G*RR	G*RR/2
disk failure write time	2*RW	RW	2*W	2*W+2*RW	4*RW	2*RW
previously reconstructed read time	R+RR	R	2*R	2*R	R+RR	R+RR
site failure read time	G*RR	RR	-	G*RR	G*RR	G*RR/2
site failure write time	2*RW	RW	-	2*RW	4*RW	2*RW

A Performance Comparison
Figure 3

algorithm to reconstruct the desired block locally. The only scheme that requires less operations is ROWB which needs only to read the value of the other copy of the desired object, a single remote read.

Write operations require less operations when a disk failure is present. Each parity scheme writes the appropriate spare block plus the parity block. The various schemes thereby require a mix of local and remote writes. Only C-RAID and 2D-RADD which employ the algorithm more than once have higher overhead. ROWB, of course, needs only to write the single copy of the object which is up.

Lastly, we consider the case of read operations to a block which has already been written onto the spare block or which has been previously reconstructed. In this case, all parity schemes must read the spare block and perhaps also the normal block. Counting both reads yields the fifth row of Figure 3.

In the case of a site failure or site disaster, modifications of the the disk failure costs must be made for two of the parity schemes. Specifically, a RAID cannot handle either failure and must block. Furthermore, a C-RAID must use the RADD portion of its algorithm to process read operations. Hence, reconstruction occurs with G remote reads rather than G local reads. The final two rows indicate these changes.

To make the performance of the various options more specific, we assume the R = W = 30 msec. and that RR and RW are 2.5 times more costly. These numbers are approximately the same as the ones reported in [LAZO86]. Figure 4 gives a numerical comparison of the options by evaluating the formulas in Figure 3. During normal operation RAID outperforms all other systems because writes are less costly. RADD, 1/2-RADD and ROWB all offer the same performance while the composite schemes further escalate the cost of writes. During failures, ROWB offers superb performance, as does RAID for the failures that it can tolerate. Among the other parity systems, C-RAID and 1/2-RADD offer good performance in some failure modes. Only 2D-RAID offers high costs during all failures.

7.4. Network Performance

RAID, of course, requires no extra messages between sites. Hence it is unaffected by network costs or bandwidth. On the other hand, the implementation of ROWB that consumes the least bandwidth in a WAL environment is probably to copy the DBMS log from the first site to that of the back-up. Then, the log is simply restored onto the second system. This scheme is usually referred to as a **hot standby** [GAWL85]. A hot standby will usually result in reduced network bandwidth because the log can be a

System	RADD	ROWB	RAID	C-RAID	2D-RADD	1/2-RADD
no failure read time	30	30	30	30	30	30
no failure write time	105	105	60	165	180	105
disk failure read time	600	75	240	240	600	300
disk failure write time	150	75	60	165	300	150
previously reconstructed write time	105	30	60	60	105	105
site failure read time	600	75	-	600	600	300
site failure write time	150	75	-	105	300	150

A Numerical Cost Comparison
Figure 4

logical log of events and not a physical log of changes to secondary storage [HAER83]. RADD will require the changed bits for each block to be sent between sites in step W3. Moreover, suppose we have an encoding scheme that allows one to transmit an insert into a page by simply sending the insert and its location. At the receiving site the bits after the insert are moved down to make room for the insert. Similarly, delete operations can be efficiently encoded. Such encoding will allow B-tree inserts and deletes to be processed with minimal bandwidth requirements. Using these techniques, it appears that a RADD should approximate the bandwidth requirements of a hot standby.

Under normal circumstances, a DBMS will change only a small fraction of each data block. For example, if blocks are 4K in size and records are 100 bytes, then an update of all fields of a data record will cause 2.5 percent of the block to be changed. If every block is brought into memory, changed, and then written out, then there will be 8K of local disk bandwidth for each 100 byte change record sent over the

network. In the case that locality of reference results in the average block being changed four times in memory before it is returned to disk, then 8K of disk I/O will result in 400 bytes of network traffic. Hence, the aggregate network bandwidth needs to be only 1/20 of the aggregate disk bandwidth. This relative performance approximates what is possible today. As a result a RADD should not unduly stress a network during normal operation.

If a site is down, then a read to the down site results in G remote reads. If a single site fails, then (G-1) / G of the read operations are unaffected while 1/G of them require G physical reads. Hence, on average, each read requires two physical read operations during failures. On the other hand, write performance is only modestly affected by failures. Under the assumption that reads are half the total I/O load, then the aggregate network bandwidth and disk bandwidth at the up sites must increase by 50 percent during failures. Hence, response time and throughput would, no doubt, suffer during failures, but normal operations could usually continue under this sort of load increase.

7.5. Reliability

To make quantitative the reliability of the various schemes, we use the disk and site numbers from Table 2 and consider four environments composed of all combinations of each of two metrics. The first metric is the number of disks at a site. In a RAID environment or one where disk servers abound, there may be many disks, and we have used N = 100. On the other hand, in a conventional environment N = 10 might be more representative.

The second metric is the assumptions about disasters. It is difficult to discuss the mean time to a disaster because they are rare events. Hence we use two sets of numbers, one for a cautious user and one for a normal user. The cautious numbers suggest the mean time to disaster is about 25 years and will require 1 day of outage. This might be the assumption of a user who has spent the time to formulate a serious disaster recovery plan. On the other hand, a normal user might assume a mean time to disaster of about 100 years and two weeks of outage might result.

Table 2 indicates a column for each combination of the two metrics. All columns share the disk reliability constants from [PATT88]. Hence, a disk failure is assumed to happen about once every four years. In a RAID environment we assume a MTTR of 1 hour because it seems reasonable to have a spare disk on

site, and repair requires only a board swap. On the other hand, in a conventional environment, it would be more normal to require 8 hours to recover from a disk failure. Lastly, in all cases we assume a site failure every 150 hours, about once a week, with a mean time to recovery of 30 minutes.

In a RADD, MTTU is the expected time until a specific second site fails while the first one is down. Using the standard assumptions of exponential distributions and independent failures, this time is:

MTTU = (site-MTTF) ** 2 / (site-MMTR * (G + 1)) (3)

For ROWB, MTTB is given by (3) above with G = 1:

MTTU = (site-MTTF) ** 2 / site-MMTR * 2

For RAID, the MTTB is the expected time to the failure of a specific site, i.e:

MTTU = site-MTTF

In a C-RAID, MTTU is driven by the MTTU of the D-RAID portion, and will be approximated by (3) above. For a 2D-RAID the MTTU is the time for two specific sites to fail while the first one is recovering, namely:

MTTU = site-MTTF) ** 3 / site-MTTR * (G + 1) * (G + 1)

Consequently, Figure 5 gives the MTTU for the various systems. Since all four scenarios give the same MTTU, we report the numbers only once.

	cautious RAID	cautious conventional	normal RAID	normal conventional
disk-MTTF	30,000 hours	30,000 hours	30,000 hours	30,000 hours
disk-MTTR	1 hour	8 hours	1 hour	8 hours
site-MTTF	150 hours	150 hours	150 hours	150 hours
site-MTTR	30 minutes	30 minutes	30 minutes	30 minutes
disaster-MMTF	150,000 hours	150,000 hours	600,000 hours	600,000 hours
disaster-MTTR	24 hours	24 hours	300 hours	300 hours
N	100	10	100	10

Reliability Constants
Table 2

system	MTTU
RADD	5000 hours
ROWB	22,500 hours
RAID	150 hours
C-RAID	5000 hours
2D-RADD	83,333 hours
1/2-RADD	10,000 hours

MTTU for the Various Systems
Figure 5

MTTF can also be calculated by a collection of formulas. For RAID, MTTF is simply the mean time to the first site disaster, i.e:

MTTF = disaster-MTTF / (G + 2)

On the other hand, for RADD data loss will be caused by one of the following events:

1) a second disaster while recovering from the first
2) a disaster while recovering from a disk failure
3) a second disk crash while recovering from the first
4) a disk failure while recovering from a disaster

For all our combinations of reliability constants, it turns out that 4) is much more frequent than the other three events. Hence, MTTF can be approximated by the MTTF of 4) alone, which is:

MTTF = (site-MTTF) * (disk-MTTF) / site-MMTR * (G + 1) * N (4)

For ROWB, MTTF depends on the physical distribution of copies. At one extreme, one can allocate a specific second site to be the backup for all data at a specific site. On the other hand, each object can be backed up at a random site, in which case every disk is likely to have backup information from all other systems. In the latter case, MTTF is given by (4) above, while in the former case one must multiply (4) by N. As a result, we will use (4) as a conservative estimate for MTTF for ROWB.

Lastly, a C-RAID and a 2D-RADD will not fail unless

1) a double failure occurs on each of two systems
2) a second disaster occurs while recovering from the first
3) a double disk failure occurs while recovering from a disaster

Each of these events has a mean time to occur of more than 500 hundred years. Consequently, Figure 6 indicates the MTTF calculations of the various systems. Notice that RADD and ROWB have high reliability in conventional environments, but offer no better reliability than RAID when there are a large number of disks at each site. The explanation for this fact is that MTTF is driven by a disk failure during recovery from a disaster. With a large number of disks, the probability of one failing during disaster recovery is essentially 1.0, resulting in the same MTTF for all three systems.

8. CONCLUSIONS

We have examined six different approaches to high availability in this study, and each can be seen to offer specific advantages. RAID is the highest performance alternative during normal operations. However, it offers no assistance with site failures or disasters, and therefore has very poor MTTU and MTTF. RADD offers dramatically better availability in a conventional environment than RAID, but offers much lower performance during both normal and recovery operations. On the other hand, ROWB offers performance intermediate between RADD and RAID; however, it requires a large space overhead.

The other three options, C-RAID, 2D-RADD and 1/2-RADD, all require more space than a RADD but less than ROWB and have different good points. A 2D-RAID offers highest availability in the presence of site failures and disasters but offers the lowest performance. On the other hand, a C-RAID offers comparable availability in the presence of disasters and very good performance but suffers low availability

	cautious RAID	cautious conventional	normal RAID	normal conventional
RADD	1.71	28.5	6.84	20.0
ROWB	1.71	28.5	6.84	20.0
RAID	1.71	1.71	6.84	6.84
C-RAID	>500	>500	>500	>500
2D-RADD	>500	>500	>500	>500
1/2-RADD	3.42	>100	13.7	>100

MTTF for the Various Systems (in years)
Figure 6

during site failures. Lastly, a 1/2-RADD offers intermediate availability and performance.

To tighten up the conclusions, we assume that reads happen twice as frequently as writes and calculate the average cost of an I/O during normal operation from Figure 4. Figure 7 then restates the numerical comparison between the solutions using this I/O cost and the reliability numbers for the cautious conventional environment. There are two solutions at 25 percent overhead, and RADD clearly dominates RAID. For a modest performance degradation, RADD reliability is more than one order of magnitude better than RAID. In addition, there are three solutions near 50 percent in overhead, 1/2-RADD, C-RAID and 2D-RADD. All offer MTTF over 100 years and better MTTU than RADD. 1/2-RADD offers the best performance of the three while 2D-RADD offers the best MTTU. C-RAID seems unattractive because it is much less reliable than 2D-RADD with only slightly better performance. Lastly, ROWB requires a space overhead of 100 percent. It is somewhat more reliable than RADD and somewhat higher performance than 2D-RADD, but overall does not seem like an attractive option. Consequently, RADD, 1/2-RADD and 2D-RADD appear to be the dominant alternatives.

If we consider instead the normal RAID environment from Table 2, then RADD, ROWB and RAID all offer the same 6.84 year MTTF . All seem to be insufficiently reliable to be serious candidates for disaster recovery. Hence, 1/2-RADD and 2D-RADD remain as the desirable options.

	space overhead (percent)	I/O cost (msec)	MTTU (years)	MTTF (years)
RAID	25	40	.017	1.71
RADD	25	58.3	.57	28.5
1/2-RADD	50	58.3	1.14	>100
C-RAID	50	75	.57	>500
2D-RADD	56.25	80	9.51	>500
ROWB	100	58.3	2.57	28.5

MTTF for the Various Systems (in years)
Figure 6

In summary, note that RADD and its variants, 1/2-RADD and 2D-RADD, offer an attractive combination of performance, space overhead and reliability. They appear to dominate RAID and ROWB as reliability enhancers in multi-site environments.

REFERENCES

[ABBA85] Abbadi. A. et. al., "An Efficient Fault-Tolerant Protocol for Replicated Data Management," Proc. 1985 ACM-SIGACT SIGMOD Conference on Principles of Database Systems, Waterloo, Ontario, March 1985.

[BERN81] Bernstein, P. and Goodman, N., "Concurrency Control in Distributed Database Systems," Computing Surveys, June 1981.

[CHAN87] Chang, A. and Mergen, M., "801 Storage: Architecture and Programming," Proc 11th SOSP, November 1987.

[COPE89] Copeland, G. and Keller, T., "A Comparison of High-Availability Media Recovery Time," Proc. 1989 ACM-SIGMOD Conference on Management of Data, Portland, OR, June 1989.

[GAWL85] Gawlich. D., "High Availability with Large Transaction Systems," Proc. International Workshop on High Performance Transaction Systems, Asilomar, CA, Sept. 1985.

[GIBS89] Gibson, G. et. al., "Failure Correction Techniques for Large Disk Arrays," Proc. Third Int. Conf. on Architectural Support for Programming Languages and Operating Systems, March 1989.

[GRAY78] Gray, J., "Notes on Database Operating Systems," IBM Research, San Jose, CA, RJ2188, Feb 1978.

[HAER83] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Database Recovery," ACM Computing Surveys, December 1983.

[KATZ89] Katz, R., "Algorithms for RAID Reconstruction," (in preparation).

[LAZO86] Lazowska, E. et. al., "File Access Performance of Diskless Workstations," ACM TOCS, August 1986.

[OUST88] Ousterhout, J. and Douglis, F., " Beating the I/O Bottleneck: A Case for Log-Structured File Systems," University of California, Computer Science Division, Technical Report UCB/CSD 88/467, October 1989.

[PATT88] Patterson, D. et. al., "RAID: Redundant Arrays of Inexpensive Disks," Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, Ill., June 1988.

[SKEE81] Skeen, D., "Non Blocking Commit Protocols," Proc. 1981 ACM SIGMOD Conference on Management of Data, Ann Arbor, Mich., June 1981.

[SPEC87] Spector, A. et. al., "Camelot: A Distributed Transaction Facility for Mach and the Internet," CMU Dept. of Computer Science, Report CMU-CS-87-129, June 1987.

[STON86] Stonebraker, M. and Rowe, L., "The Design of POSTGRES," Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., May 1986.

[STON37] Stonebraker, M., "The POSTGRES Storage System," Proc. 1987 VLDB Conference, Brighton, England, Sept. 1987.

[WENS88] Wensel, S. (ed.), "The POSTGRES Reference Manual," Electronics Research Laboratory, University of California, Berkeley, CA, Report M88/20, March 1988.