DISTRIBUTED RAID -- A NEW MULTIPLE COPY ALGORITHM

Michael Stonebraker † and Gerhard A. Schloss ‡

† EEEC Department, CS Division
and
‡ Walter A. Haas School of Business
University of California, Berkeley
Berkeley, CA 94720

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 sequence of recent papers, the concept of a single site RAID (Redundant Array of Inexpensive Disks) was introduced and developed [PATT88, PATT89, GIBS89]. Such disk systems have the desirable property that they survive disk crashes and require only one extra disk for each group of G disks. Hence, the space cost of high availability is only $\frac{100}{G}$ percent, a modest amount compared to traditional schemes which mirror each physical disk at a space cost of 100 percent.

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

The purpose of this research is to extend the RAID concept to a distributed computing system. We call the resulting construct RADD (Redundant Array of Distributed Disks). RADDs are shown to support redundant copies of data across a computer network at the same space cost as RAIDs do for local data. Such copies increase availability in the presence of both temporary and permanent failures (*disasters*) of single site 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** [GAWL87] or other techniques surveyed in [KIM84].

This paper is structured as follows. Section 2 briefly reviews a Level 5 RAID from [PATT88], which is the idea we extend to a distributed environment. Then, in Section 3 we discuss our model of a distributed computing system and describe the basic structure of a RADD. Section 4 deals with performance and reliability issues of RADD as well as several other high-availability constructs, while Section 5 considers miscellaneous RADD topics including concurrency control, crash recovery, distributed DBMSs, and non uniform site capacity. Finally, Section 6 closes with conclusions and mentions several candidate topics for future research.

2. RAID - Redundant Array of Inexpensive Disks

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 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. Denote a parity block by P and a regular data block by D. Then:

$$P = P_{old} \ \mathbf{XOR} \ (D_{new} \ \mathbf{XOR} \ D_{old}) \tag{1}$$

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 two physical read-modify-write accesses. However, since many writes are preceded by a read, 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] suggested striping the parity blocks over all $G+1$ drives such that each physical drive has $\frac{1}{G+1}$ of the parity data. In this way, up to $\frac{G}{2}$ writes can occur in parallel through 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 $(G+2)$-nd spare disk associated with the group. 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:

$$D_{failed} = \mathbf{XOR} \ \{\text{other blocks in the group}\} \tag{2}$$

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 disk failure must occur while recovering from the first one. Since the mean time to failure, *MTTF*, of a single disk is typically in excess of 35,000 hours (about four years) and the recovery time can easily be contained to an hour, the mean time to data loss, *MTTLD*, in a RAID with $G = 10$ exceeds 50 years.

Hence, we 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 in [PATT88] 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 or software failure, etc., 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 - Redundant Array of Distributed Disks

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 N physical disks each with B blocks. These $N * B$ blocks are managed by the local operating system or the I/O controller and have the following composition:

$$\frac{N * B * G}{G + 2} \qquad - \qquad \text{data blocks}$$

$\frac{N * B}{G + 2}$ - parity blocks

$\frac{N * B}{G + 2}$ - spare blocks

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 help 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.

In Figure 1 we show the layout of data, parity and spare blocks for the case of $G = 4$. The i-th 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, as well as a single S, the spare block which will be used to store the contents of an unavailable 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. Note that user reads and writes are directed at data blocks and not parity or spare blocks. We also assume that the network is reliable. Analysis of the case of unreliable networks can be found in [STON89].

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

- disk failures
- temporary site failures
- permanent site failures (disasters).

In the first case, a site continues to be operational but loses one of its N disks. The site remains operational, except for 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

	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

Figure 1: The Logical Layout of Disk Blocks

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 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 effort is made to survive multiple failures.

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 = $(G + 2)$ * quotient (I / G) + 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' = 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.**

4. Performance and Reliability of a RADD

In this section we compare the performance of a RADD against four other possible schemes that give high availability. The first is a traditional multiple copy algorithm. Here, we restrict our attention to the case where there are exactly two copies of each object. Thus, any interaction with the database 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 scheme in which the RADD algorithms are applied to the different sites and 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 plus RADD’’ scheme will be called C-RAID. Finally, it is also possible to utilize a *two-dimensional* RADD. In such a system the sites are arranged into a two-dimensional array and row *and* column parities are constructed, each according to the formulas of Section 3. We call this scheme 2D-RADD, and a variation of this idea was developed in [GIBS89]. The comparison considers the space overhead as well as the cost of read and write operations for each scheme under various system conditions.

The second issue is reliability, and we examine two metrics for each system. The first metric is the *mean time to unavailability* of a specific data item, *MTTU*. This quantity is the mean time until a 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*, *MTTLD*. This quantity is the mean time until there exists a data item that cannot be restored.

4.1. Disk 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. Our algorithms 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 and spare blocks is left for future research.

Figure 2 indicates the space overhead of each scheme. Clearly, the traditional multiple copy

System	Space Overhead
RADD	25%
RAID	25%
2D-RADD	50%
C-RADD	56.25%
ROWB	100%

Figure 2: A Disk Space Overhead Comparison

algorithm requires a 100 percent space penalty since each object is written twice. Since $G = 8$ and we are also allocating a spare block for each parity block, the parity schemes (RAID and RADD) require two extra blocks for each 8 data blocks, i.e. 25 percent. In a two-dimensional array, for each 64 disks the 2D-RADD requires two collections of 16 extra disks. Hence, the total space overhead for 2D-RADD 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. Hence, the total space overhead is 56.25 percent.

4.2. 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.

During normal operation 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 and ROWB need a local write plus a remote write. In a 2D-RADD, the RADD algorithm must be run in two dimensions, resulting in one local write and two remote writes. A C-RAID requires a total of four writes. The RADD portion of the C-RAID performs a local write and a remote write as above. However, each will be

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

Table 1: I/O Cost Parameters

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

Figure 3: A Performance Comparison

turned into two actual local writes by the RAID portion of the composite scheme, for a total of three local writes plus one remote 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 C-RAID and remote operations for RADD and 2D-RADD. ROWB is the only scheme that requires less operations, since it needs only to read the value of the other copy of the desired object, a single remote read.

Writes require less operations than reads when a disk failure is present. Each parity scheme writes the appropriate spare block plus the parity block, thereby requiring two (RADD) to four (2D-RADD) remote writes, or a mix of local and remote writes (C-RAID). Of course, ROWB needs only to write to the single copy of the object which is up.

We also 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 disk failure costs must be made for the parity schemes. Specifically, a RAID cannot handle either failure and must block. Furthermore, a C-RAID must use its RADD portion to process read operations. Hence, reconstruction occurs with G remote reads rather than local reads. In a RADD and in ROWB, the decrease in performance is the same as in the case of a disk failure.

Figure 3 shows the number of local and remote operations required by the systems to perform reads and writes under various circumstances. Note that these figures represent *the worst cases*, namely the total number of operations required when the desired data are on a failed component. Other I/O operations are, of course, unaffected.

To summarize, our figures show that during normal operation RAID outperforms all other systems because writes are less costly. RADD and ROWB offer the same performance. During failures, ROWB offers superior performance, as does RAID for the failures that it can tolerate. C-RAID offers good performance in some failure modes, when its RAID portion can be utilized. In contrast, a 2D-RADD offers high costs during all types of failures.

4.3. Availability and Reliability

To make quantitative the reliability of the various schemes, we use the disk and site numbers from Table 2. The first metric is the number of disks at a site. In a RAID environment there may be many small disks, and we have used $N = 25$. On the other hand, in a conventional environment with large capacity 3380 style drives, $N = 10$ might be more representative.

The second metric is the assumptions about failures and disasters. In particular, we consider mean times to failure, *MTTF*, as well as mean times to repair, *MTTR*, for both disks and sites. 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 75 years and will require one day (24 hours) 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 to recover from a disaster of about two weeks of outage.

Table 2 has a column for a cautious user with 25 disks and a conventional user with 10 disks. The interested reader can generate results for other cases by applying the formulas which follow. Both columns share the disk reliability constants from [PATT88]. Hence, a disk failure is assumed to happen about once every four years. In a cautious RAID or RAID environment we assume a *MTTR* of one hour because spare disks exist in the array and one need only reconstruct a disk in background. On the other hand, in a conventional environment, a large capacity disk must be reconstructed, and we have allowed 8 hours for this background task. We generously use the same recovery times for ROWB which has no on-site spares, because exact recovery times would depend on the service contract chosen. Lastly, in all cases we assume a site failure every 2,000 hours, namely about once in 12 weeks, with a mean time to recovery of 30 minutes.

	cautious system	conventional system
	--------	-------------
N	25	10
$MTTF_{disk}$	35,000 hrs	35,000 hrs
$MTTR_{disk}$	1 hr	8 hrs
$MTTF_{site}$	2,000 hrs	2,000 hrs
$MTTR_{site}$	.5 hrs	.5 hrs
$MTTF_{disaster}$	650,000 hrs	650,000 hrs
$MTTR_{disaster}$	24 hrs	350 hrs

Table 2: Reliability Constants.

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

$$MTTU = MTTF_{site} \tag{3a}$$

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

$$MTTU = \frac{(MTTF_{site})^2}{MTTR_{site} * (G+1)} \tag{3b}$$

In a C-RAID, *MTTU* is driven by the *MTTU* of the RADD portion, and therefore it will be approximated by (3b) above. For ROWB, *MTTB* is given by (3b) above with *G* = 1:

$$MTTU = \frac{(MTTF_{site})^2}{MTTR_{site} * 2}$$

For a 2D-RADD, the *MTTU* is the expected time until two specific sites fail while the first one is recovering, namely:

$$MTTU = \frac{(MTTF_{site})^3}{(MTTR_{site})^2 * (G+1) * G} \tag{3c}$$

Figure 4 gives the *MTTU* for the various systems. Since all four scenarios give the same *MTTU*, we report the numbers only once.

system	*MTTU* (years)
RAID	.23 (2 wks)
RADD	101
C-RADD	101
ROWB	455
2D-RADD	>500

Figure 4: *MTTU* for the Various Systems

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

$$MTTLD = \frac{MTTF_{disaster}}{(G+2)} \tag{4a}$$

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

(a) a second disaster while recovering from the first

(b) a disaster while recovering from a disk failure

(c) a second disk crash while recovering from the first

(d) a disk failure while recovering from a disaster.

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

$$MTTLD = \frac{MTTF_{disaster} * MTTF_{disk}}{MTTR_{disaster} * (G+2) * (G+1) * N} \tag{4b}$$

For ROWB, *MTTLD* 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. This **optimistic** scenario will result in highest *MTTLD*. 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 this **pessimistic** case, *MTTLD* is given by (4b) above, while in the optimistic case one must multiply (4b) by the ratio of extra disks required for a ROWB. In a real system the actual *MTTLD* would lie in between these two extremes.

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

(a') a double site failure occurs while recovering from a disaster

(b') a second disaster occurs while recovering from the first

(c') a double disk failure occurs while recovering from a disaster.

Each of these events has a mean time to occur of more than 500 years. Figure 5 indicates the *MTTLD* calculations of the various systems. Notice that RADD and ROWB have high reliability in cautious 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 *MTTLD* 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 *MTTLD* for all three systems.

	cautious	conventional
	system	system
RADD	48.1	8.25
ROWB-optimistic	216	37
ROWB-pessimistic	48.1	8.25
RAID	7.5	7.5
C-RADD	>500	>500
2D-RADD	>500	>500

Figure 5: *MTTLD* for the Various Systems (in years)

5. Miscellaneous RADD Topics

5.1. 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 unique identifier, 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.

5.2. Crash Recovery

The algorithms we have outlined in Section 3 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.

5.3. Distributed Databases

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 RADD algorithms do not appear to impact any of the common concurrency control techniques [BERN81, GRAY78].

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 algorithms seems to interfere with such commit processing in a RADD environment.

5.4. Non-Uniform Site Capacity

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.

6. Conclusions and Future Research

We have examined four 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 *MTTLD*. RADD offers dramatically better availability in a conventional environment than RAID, but offers much lower performance during recovery operations. On the other hand, ROWB offers performance intermediate between RADD and RAID. However, it requires a large space overhead.

The two options, C-RAID and the two-dimensional 2D-RADD, require more space than a RADD but less than ROWB. A 2D-RADD provides the highest availability in the presence of site failures and disasters, but offers the lowest performance. Finally, a C-RAID combines the survivability features of a RADD with a better performance during reconstruction due to its RAID capabilities. Although C-RAID seems to be inferior in space utilization, an optimization of the spare blocks allocation between the RADD and the RAID portions can significantly decrease the required overhead. Analysis of this topic is left for a future research.

To make our conclusions more concrete, we calculate the average cost of an I/O during normal operation as well as during disk and site failures, using performance data from Figure 3.

Figure 6 then restates numerical comparison between the solutions using these I/O costs and the reliability numbers for the cautious (RAID) environment. Note that the I/O costs are calculated using the following assumptions:

(a) $R = W = 30$ msec, $RR = RW$

(b) a remote operation is 2.5 times more costly than a local operation [LAZO86]

(c) reads happen twice as frequently as writes.

There are two solutions at 25 percent disk space overhead, and RADD clearly dominates RAID. For a modest performance degradation, RADD reliability is almost one order of magnitude better than RAID. There are two solutions with about 50 percent overhead, 2D-RADD and C-RAID. Both offer even higher reliability but much poorer performance during normal operation. Lastly, ROWB requires a space overhead of 100 percent; however, it has the lowest cost

	space	I/O cost -	I/O cost -	MTTU	MTTLD
	overhead	normal op.	failures		
	(%)	(msec)	(msec)	(years)	(years)
RAID	25	40	180†	.23	7.5
RADD	25	55	450	101	48.1
2D-RADD	50	80	500	>500	>500
C-RAID	56.25	75	340	101	>500
ROWB-optimistic	100	55	75	455	216
ROWB-pessimistic	100	55	75	455	48.1

Figure 6: Summary of Performance Parameters for the Various Systems

† for disk failures only

during failures and offers high reliability..

In summary, note that RADD and its variants, the 2D-RADD and the C-RAID, offer an attractive combination of performance, space overhead and reliability. They appear to dominate RAID as reliability enhancers in multi-site systems. If disk space is an important consideration, they may also be attractive relative to ROWB. However, all alternatives examined may be found desirable, depending on the requirements of a specific environment.

Several important issues were left out in this research. First, we assumed identical *MTTF*'s and *MTTR*'s, only one group size and equal significance for all data. Second, we have not examined in detail the various data reconstruction algorithms. Finally, no attempt has been made to optimize C-RAID. We hope to address all these topics in future research.

REFERENCES

[ABBA85] El Abbadi. A., Skeen, D. and Cristian, F., 'An Efficient Fault-Tolerant Protocol for Replicated Data Management', *Proc. 1985 ACM-SIGACT SIGMOD Conf. on Principles of Database Systems*, Waterloo, Ontario, March 1985.

[BERN81] Bernstein, P. and Goodman, N., 'Concurrency Control in Distributed Database Systems', *ACM 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. of 1989 ACM SIGMOD Conf. on Management of Data*, Portland, OR, June 1989.

[GAWL87] Gawlich, D., 'High Availability with Large Transaction Systems', *Proc. 2nd Int. Workshop on High Performance Transaction Systems*, Asilomar, CA, September

1987.

[GIBS89] Gibson, G. et. al., ‘Error Correction in Large Disk Arrays’, *Proc. 3rd Int. Conf. on ASPLOS*, Boston, MA, April 1989.

[GRAY78] Gray, J., ‘Notes on Database Operating Systems’, Research Report RJ2188, IBM Research Lab., San Jose, CA, February 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).

[KIM84] Kim, W., ‘Highly Available Systems for Database Applications’, *ACM Computing Surveys*, September 1984.

[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, Berkeley, Computer Science Division, Technical Report UCB/CSD 88/467, October 1988.

[PATT88] Patterson, D., Gibson, G. and Katz, R., ‘A Case for Redundant Arrays of Inexpensive Disks (RAID)’, *Proc. of 1988 ACM SIGMOD Conf. on Management of Data*, Chicago, IL, June 1988.

[PATT89] Patterson, D., Chen, P., Gibson, G. and Katz, R., ‘Introduction to Redundant Arrays of Inexpensive Disks (RAID)’, *Proc. 1989 IEEE Conf. on Data Engineering*, Los Angeles, CA, April 1989.

[SKEE81] Skeen, D., ‘Non Blocking Commit Protocols’, *Proc. of 1981 ACM SIGMOD Conf. on Management of Data*, Ann Arbor, MI, 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.

[STON87] Stonebraker, M., *The POSTGRES Storage System*, Proc. 1987 VLDB Conference, Brighton, England, Sept. 1987.

[STON88] Stonebraker, M., Patterson, D., Katz, R. and Ousterhout, J., ‘The Design of XPRS’, *Proc. of 14th VLDB Conf.*, Long Beach, CA, August 1988.

[STON89] Stonebraker, M., ‘Distributed RAID - A New Multiple Copy Algorithm’, Technical Memo. No. UCB/ERL M89/56, Electronics Research Lab., College of Engineering, UC Berkeley, May 1989.

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