A Comparison Of High-Availability Media Recovery Techniques

George Copeland and Tom Keller
MCC
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

We compare two high-availability techniques for recovery from media failures in database systems. Both techniques achieve high availability by having two copies of all data and indexes, so that recovery is immediate. "Mirrored declustering" spreads two copies of each relation across two identical sets of disks. "Interleaved declustering" spreads two copies of each relation across one set of disks while keeping both copies of each tuple on separate disks. Both techniques pay the same costs of doubling storage requirements and requiring updates to be applied to both copies.

Mirroring offers greater simplicity and universality. Recovery can be implemented at lower levels of the system software (e.g., the disk controller). For architectures that do not share disks globally, it allows global and local cluster indexes to be independent. Also, mirroring does not require data to be declustered (i.e., spread over multiple disks).

Interleaved declustering offers significant improvements in recovery time, mean time to loss of both copies of some data, throughput during normal operation, and response time during recovery. For all architectures, interleaved declustering enables data to be spread over twice as many disks for improved load balancing. We show how tuning for interleaved declustering is simplified because it is dependent only on a few parameters that are usually well known for a specific workload and system configuration.

1 Introduction

Most database systems employ a checkpoint and log of the database for recovery from media failure, system failure and user error, and periodically archive older copies for disaster recovery [Gra78]. The problem with this technique when used for media recovery is its slow recovery time during which data is unavailable.

To obtain higher availability via immediate recovery, some systems maintain two identical on-line copies of all data and their indexes, where updates are sent to both copies and reads can use either copy. "Mirroring" (also called "duplexing" or "shadowing") is one example of this, where each disk has a twin containing the same data ([Kat78], [Bit88]). A special case of mirroring for database systems is "mirrored declustering" [Tan87], which spreads two copies of each relation across two sets of disks, such that each disk in one set has a twin in the other set containing exactly the same data. "Interleaved declustering" [Ter85] spreads two copies of each relation across one set of disks, such that the two copies of each tuple are guaranteed to be placed on different disks. Both of these immediate recovery techniques provide higher availability than the checkpoint-and-log technique, because data is available immediately after a media failure, although at reduced performance. Their cost is doubling the size of both the base data and their indexes (i.e., more disk storage) and requiring updates to be applied to both copies (i.e., more disk arms).

Two alternatives for media recovery have been proposed for systems with many disks which require less storage and/or update overhead than the immediate-recovery techniques. [Par86], [Kat88] and [Pat87] use "interleaved parity sectors" within each cluster of d+p disks, such that d data sectors and their p "parity" (actually an error-correcting code) sectors are placed on different disks. This is analogous to the way memory designers spread the bits of each memory word and its error-correcting code across multiple chips to protect against single-chip failures. [Cop88] forms a backup copy of each relation from a set of inverted files plus a

"remainder relation" containing all non-inverted attributes. The relation and its backup are placed on different disks. Recovery time for either of these two techniques is slower than that of the immediate-recovery techniques but typically faster than the checkpoint-and-log technique. However, both alternatives' cost/performance during normal operation can be significantly better than the immediate-recovery techniques for applications whose availability requirements lie between the checkpoint-and-log and immediate-recovery techniques.

Regardless of whether any of the alternative higher-availability techniques are used for media failure, the checkpoint-and-log technique must also be employed. The reason is that the checkpoint-and-log technique insures against a wider range of failures. It provides recovery from system failures, user errors and natural disasters. It also acts as a backup for media failures in case the higher-availability techniques fail.

This paper compares the two immediate-recovery techniques of mirrored and interleaved declustering for various database system architectures having many disks. We first describe the two techniques in Section 2. Then we model their performance in Section 3. Sections 4 and 5 separately compare those aspects of the two techniques that are independent and dependent on the database system hardware architecture. Section 6 provides a summary.

2 Immediate-Recovery Techniques

This section describes the two immediate-recovery techniques. First, we describe the characteristics that are common for both techniques. Then we describe the interleaved and mirrored declustering techniques.

2.1 Common Characteristics

Both techniques maintain two copies of all data and indexes, so that recovery can be immediate. That is, data is available immediately after a failure, although at reduced performance. During normal operation (i.e., without failures), a read operation can use either copy while a write operation must be applied to both copies.

When a disk fails, the following algorithm is used:

1) Notify the system of the Failed Disk. This is needed to trigger the recovery algorithm, as well as to update whatever system tables that are required to prevent using the Failed Disk for user transactions until it is recovered.
2) Resume processing user transactions immediately.
3) Find a replacement for (or repair) the Failed Disk.
4) Begin a background copy algorithm [Att84] from the Backup Disks to the replaced or repaired Failed Disk, where small subsets of the lost data are each copied as a separate transaction, serialized with user transactions, and writes are applied to whatever data has already been copied on the Failed Disk.
5) When copying is complete, notify the system to reinstate the Failed Disk and resume normal operation.

Both techniques use "declustering" to achieve disk load balancing (e.g., [Ter85], [DeW86], [Liv87] and [Tan87]). Declustering spreads subsets of the tuples of each relation across many (not necessarily all) disks, usually using a hash function. The total number of disks containing one complete copy of a particular relation is called the "degree of declustering" (D) of that relation. D is not necessarily the same for all relations. The two techniques differ in their method of declustering.

2.2 Interleaved Declustering Technique

Interleaved declustering divides the disks into clusters, each containing $S \geq 2$ disks. Figure 2.1 illustrates this for S=2 and Figure 2.2 for S=4. One copy of each relation is declustered across all disks within one or more of these clusters. In both figures, relation R is partitioned into R0, R1, ..., each of which is placed on a separate disk. The second copy of each relation further partitions the subset of the first copy on each disk across all other disks within the same cluster. In Figure 2.1, this is trivial because there is only one other disk within each cluster. In Figure 2.2, for example, R0 in D0 is further partitioned to form a second copy into r0.0, r0.1 and r0.2. Each of these S-1 smaller subsets is placed on one of the disks in the same cluster as R0, but not on the same disk as R0. Thus,

$$D = S, 2S, \ldots, N ,$$

for each copy, where N is the total number of disks in the system.

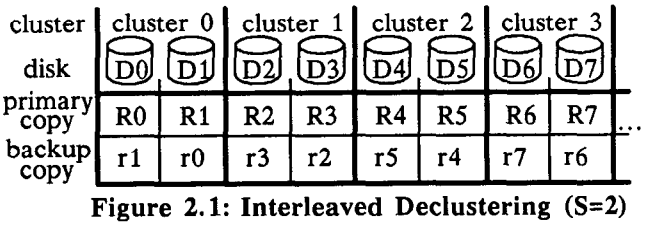

Figure 2.1: Interleaved Declustering (S=2)

[Ter85] implements this by hashing the keys of tuples of a relation onto a large number of buckets, and the mapping of each bucket onto two different Diskids within the same recovery cluster is stored explicitly in a table. This table is subsequently used to find which disk contains a given tuple. An alternative to this table-lookup method is to use an algorithmic mapping, such as the one illustrated in

cluster	cluster 0				cluster 1			
disk	D0	D1	D2	D3	D4	D5	D6	D7
primary copy	R0	R1	R2	R3	R4	R5	R6	R7
backup copy	r1.2	r0.0 r2.2	r0.1 r1.0	r0.2 r1.1 r2.0	r5.2 r6.1	r4.0 r6.2	r4.1 r5.0 r7.2	r4.2 ... r5.1 r6.0
	r2.1	r3.1	r3.2		r7.0	r7.1		
	r3.0							

Figure 2.2: Interleaved Declustering (S=4)

Figure 2.3, which stores only a vector (V) of cluster identifiers for each relation and uses modular arithmetic to compute the particular Diskid. The tradeoff is between less processing for the table-lookup approach vs. less memory and updating for the algorithmic approach.

h = hash(key value)

$H = \mathrm{mod}_D(h)$

first copy:

1st disk within cluster; offset within cluster

$$\mathrm{Diskid}_1 = V\left(\left\lfloor \frac{H}{S} \right\rfloor\right) S + \mathrm{mod}_S(h)$$

second copy:

1st disk within cluster; offset within cluster

$$\mathrm{Diskid}_2 = V\left(\left\lfloor \frac{H}{S} \right\rfloor\right) S + \mathrm{mod}_S\left(\underbrace{\mathrm{mod}_S(h)}_{\text{primary offset within cluster}} + \underbrace{1}_{\text{shift}} + \underbrace{\mathrm{mod}_{S-1}(h)}_{\text{spread within cluster}}\right)$$

Figure 2.3: An Interleaved Declustering Algorithm

2.3 Mirrored-Declustering Technique

Mirrored-declustering divides the disks into clusters, each containing two disks, as illustrated in Figure 2.4. One copy of each relation is declustered across one or more of these clusters but is only stored on one of the two disks within each cluster, while the second copy is stored on the other disk. Thus, D for each copy is

$$D = 1, 2, \ldots, \frac{N}{2} .$$

cluster	cluster 0		cluster 1		cluster 2		cluster 3	
disk	D0	D1	D2	D3	D4	D5	D6	D7
primary copy	R0 R1		R2 R3		R4 R5		R6 R7	...
backup copy		r0 r1		r2 r3		r4 r5		r6 r7

Figure 2.4: Mirrored Declustering

The major difference between mirrored and interleaved declustering is that interleaved declustering allows S>2. Mirrored declustering is similar to interleaved declustering with S=2, except that mirrored declustering limits D for each copy, as can be seen by comparing Figures 2.2 and 2.4.

3 Performance Analysis

In this section, we derive analytic formulas for recovery time, mean time to failure of both copies of any data (causing a checkpoint-and-log recovery), response time during recovery copying, and throughput during normal operation. For the most part, the formulas for mirrored declustering are the same as for interleaved declustering with S=2.

In this analysis, we make the following practical assumptions for the sake of simplicity:

A) All data is disk-resident.

B) Disks are assumed to be the bottleneck; thus only disk performance is analyzed.

C) The load is uniformly balanced among the disks during normal operation.

D) Disk utilization during normal operation is low enough that the normal system load can still be maintained during recovery copying. This means that recovery copying can be done as a background operation.

E) The maximum single disk accesses/sec (μ) is assumed to be the same during recovery copying as during normal operation.

F) Both copies of the data are clustered so that only the actual backup data must be read from the Backup Disks (see Section 5.2).

G) The copy units are copied using a random ordering.

H) The impact of concurrency control is negligible, because disks are assumed to be the bottleneck. Copy transactions involve a small amount of data (e.g., a disk track) and are delayed or aborted whenever they conflict with user transactions.

3.1 Recovery Time

The total recovery time is

$$T_{rec} = T_{rep} + T_c ,$$

where T_{rep} is the time to replace or repair the Failed Disk and T_c is the time to copy the data that was lost on the Failed Disk from the Backup Disks to the replaced or repaired Failed Disk. Note that if spare disks are kept on-line, then replacement is almost immediate, so that $T_{rec} \cong T_c$.

During normal operation, the processing rate of each disk in a recovery cluster has a load of $\lambda_n = \mu \rho_n$ as illustrated in Figure 3.1, where λ_n, μ and ρ_n are the arrival rate, service rate and utilization, respectively, of each disk. The service rate does not distinguish between read and write accesses.

λ_n	λ_n	λ_n	λ_n
D0	D1	D2	D3

Figure 3.1: Load During Normal Operation

For analysis, it is useful to separate the total load λ_n into its read λ_r and write λ_w components:

$$\lambda_n = \lambda_r + \lambda_w$$
$$\rho_n = \rho_r + \rho_w,$$

where $\lambda_r = F_r \lambda_n$ and $\lambda_w = F_w \lambda_n$
$\rho_r = F_r \rho_n$ and $\rho_w = F_w \rho_n$,

and F_r and F_w are the fractions of physical read and write disk accesses, so that $F_r+F_w=1$.

Figure 3.2 illustrates how the loads are different in the Backup Disks and the Failed (and now replaced or repaired) Disk. Each of the Backup Disks must execute the normal load $\lambda_n=\lambda_r+\lambda_w$, plus $1/(S-1)$ of both the Failed Disk's read load λ_r and the copying reads λ_c. The Failed Disk must execute updates to only that fraction of the data which has already been copied $u(t)/U$ (i.e., data that has already been copied before an update occurs are updated in the Failed Disk; all other data is updated in the Backup Disks before being copied to the Failed Disk), plus the copying writes λ_c, where U is the total number of copy units required to recover the Failed Disk and u(t) is the accumulated number of units copied as of time t. When the Failed Disk is the bottleneck, better performance could be achieved by copying the least-frequently-used copy units first because this would reduce the Failed Disk's total updating load so that λ_c could be larger. However, including a distribution of frequency of use in the analysis would be too complex. Instead, we assume that the copy units are randomly ordered, so that the Failed Disk's update load is simply $\lambda_w u(t)/U$.

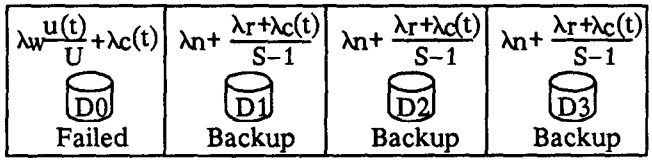

Figure 3.2: Load During Recovery Copying

In general, λ_c is a function of time, because the Failed Disk may be the bottleneck and it has a time-varying load, so that the accumulated number of units copied as of time t is

$$u(t) = \int_0^t \lambda_c(t)dt = \mu\int_0^t \rho_c(t)dt, \text{ for } 0\le t\le T_c \quad \text{(Eqn. 3-1)}$$

and $u(T_c) = U$,

where $\rho_c(t)=\lambda_c(t)/\mu$ and T_c is the time to copy the U total copy units.

We define $\lambda_m=\mu\rho_m$ as the maximum processing rate of any disk, such that $\lambda_m>\lambda_n$ and $\rho_m>\rho_n$. In Section 3.3, we show how ρ_m limits the impact of disk accesses for background copying on the response time of foreground disk accesses. In each of the Backup Disks, the total work must be less than this maximum rate, so that

$$\lambda_m \ge \lambda_n + \frac{\lambda_r+\lambda_c(t)}{S-1}, \text{ for } 0\le t\le T_c$$

or $$\rho_m \ge \rho_n + \frac{\rho_r+\rho_c(t)}{S-1}, \text{ for } 0\le t\le T_c.$$

Likewise, in the Failed Disk, the total load must be less than this maximum rate, so that

$$\lambda_m \ge \lambda_w\frac{u(t)}{U} + \lambda_c(t), \text{ for } 0\le t\le T_c$$

or $$\rho_m \ge \rho_w\frac{u(t)}{U} + \rho_c(t), \text{ for } 0\le t\le T_c.$$

For recovery copying to be done as a background operation, we must have $\rho_c(t)>0$ in the above inequality for the Backup Disks. This implies

$$\rho_n < \frac{S-1}{S-F_w}\rho_m \equiv \rho_{n\max}$$

or conversely $S > \frac{R-F_w}{R-1} \equiv S_{min}$, where $R\equiv\frac{\rho_m}{\rho_n}$.

$\rho_{n\max}$ is the maximum utilization during normal operation (or, conversely, S_{min} is the minimum S) that allows background copying. R is the ratio of maximum utilization to utilization during normal operation, a metric of the additional disk utilization available in the system for recovery copying. Notice the convenient property that S_{min} is only a function of R and F_w.

If the Backup Disks are the bottleneck during recovery copying instead of the Failed Disk, then ρ_c is a constant wrt time (ρ_{cB}) which is as large as ρ_n will allow on the Backup Disks given the constraint of ρ_m, or

$$\rho_m = \rho_n + \frac{\rho_r+\rho_{cB}}{S-1}.$$

Solving for ρ_{cB},

$$\rho_{cB} = \rho_n\big((R-1)S - R + F_w\big). \quad \text{(Eqn. 3-2)}$$

Notice that ρ_{cB} can be larger than ρ_m, because we have defined λ_c to be the total copy rate seen by the Failed Disk rather than the rate within each Backup Disk.

If the Failed Disk is the bottleneck during recovery copying, then ρ_c is a function of time ($\rho_{cF}(t)$), and grows as large as ρ_m on the Failed Disk will allow:

$$\rho_m = F_w\rho_n\frac{u(t)}{U} + \rho_{cF}(t), \text{ for } 0\le t\le T_c.$$

Solving for $\rho_{cF}(t)$,

$$\rho_{cF}(t) = \rho_m - F_w\rho_n\frac{u(t)}{U}, \text{ for } 0\le t\le T_c. \quad \text{(Eqn. 3-3)}$$

$\rho_c(t)$ is whichever of ρ_{cB} or $\rho_{cF}(t)$ is the bottleneck:

$$\rho_c(t) = \min\big(\rho_{cB}, \rho_{cF}(t)\big), \text{ for } 0\le t\le T_c.$$

To find the total copying time T_c, we solve the following equation relating the total number of copy units copied to the copy time:

$$U = u(T_c) = \mu \int_0^{T_c} \rho_c(t)\,dt \quad . \qquad \text{(Eqn. 3-4)}$$

This requires examining the following three cases, each having different formulas for $\rho_c(t)$:

Case B: The Backup Disks are the bottleneck during the entire recovery-copying process, as illustrated in Figure 3.3. This case occurs when

$$\rho_{cB} \le \rho_{cF}(T_c) = \rho_m - F_w \rho_n \quad .$$

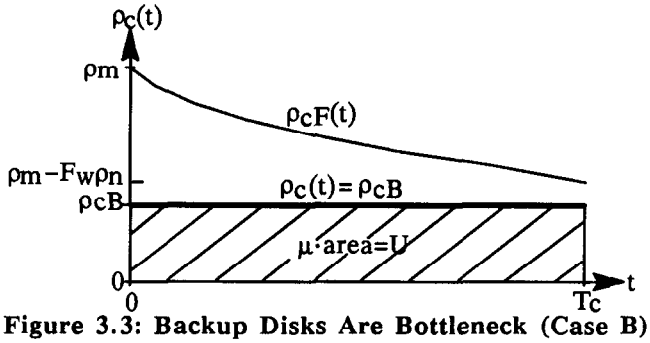

Figure 3.3: Backup Disks Are Bottleneck (Case B)

For this to occur, we must have

$$\rho_n \ge \frac{S-2}{S-2F_w}\rho_m \equiv \rho_{nB}$$

or conversely $\quad S \le 2\frac{R-F_w}{R-1} \equiv S_B \quad .$

ρ_{nB} is the minimum utilization during normal operation (or, conversely, S_B is the maximum S) for the Backup Disks to always be the bottleneck during recovery copying. Notice the convenient property that S_B is only a function of R and F_w. Also notice that interleaved declustering with S=2 (or mirrored declustering) always operates under this case. This is because the single Backup Disk does more work that the Failed Disk throughout the entire recovery process since it does as much copying as the Failed Disk plus normal transaction processing. For this case, $\rho_c=\rho_{cB}$, which is not a function of time. We can find T_c by solving Equation 3-4 using $\rho_c=\rho_{cB}$:

$$U = \mu\,\rho_{cB}\,T_c$$

so that $\quad T_c = \frac{U}{\mu\,\rho_{cB}} \quad .$

Case F: The Failed Disk is the bottleneck during the entire recovery-copying process, as illustrated in Figure 3.4. This case occurs when

$$\rho_{cB} \ge \rho_{cF}(0) = \rho_m \quad .$$

For this to occur, we must have

$$\rho_n \le \frac{S-2}{S-F_w}\rho_m \equiv \rho_{nF}$$

or conversely $\quad S \ge \frac{2R-F_w}{R-1} \equiv S_F \quad .$

ρ_{nF} is the maximum utilization during normal operation (or, conversely, S_F is the minimum S) for the Failed Disk to always be the bottleneck during recovery copying. Notice the convenient property that S_F is only a function of R and F_w. Also note

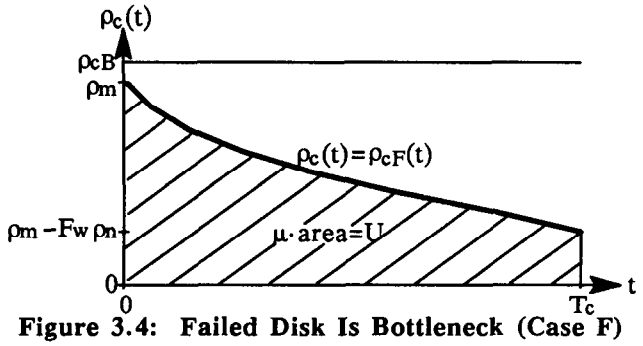

Figure 3.4: Failed Disk Is Bottleneck (Case F)

that interleaved declustering with S=2 (or mirrored declustering) never operates under this case. Since the Failed Disk is the bottleneck, $\rho_c(t)=\rho_{cF}(t)$, which is described by the following differential equation:

$$\rho_{cF}(t) = \rho_m - F_w\rho_n\frac{u(t)}{U} \quad , \text{for } 0 \le t \le T_c$$
$$= \rho_m - \frac{F_w\rho_n}{U}\mu\int_0^t \rho_{cF}(t)\,dt \quad , \text{for } 0 \le t \le T_c$$

which has the solution

$$\rho_{cF}(t) = \rho_m e^{-qt} \quad , \text{where } q \equiv \frac{\mu F_w \rho_n}{U} \quad .$$

We can find T_c by solving Equation 3-4 using this expression for ρ_c:

$$U = \mu\int_0^{T_c} \rho_m e^{-qt}\,dt$$

or $$T_c = \begin{cases} \frac{U}{\mu F_w \rho_n}\log\left(\frac{R}{R-F_w}\right), & \text{for } F_w>0 \\ \frac{U}{\mu\rho_m}, & \text{for } F_w=0 \quad . \end{cases}$$

Case BF: The Backup Disks are the bottleneck at the beginning of the recovery-copying process, and the Failed Disk is the bottleneck during the rest of the recovery-copying process, as illustrated in Figure 3.5. This case occurs when

$$\rho_{cF}(T_c) \le \rho_{cB} \le \rho_{cF}(0)$$

i.e., $\quad \rho_m - F_w\rho_n \le \rho_{cB} \le \rho_m \quad .$

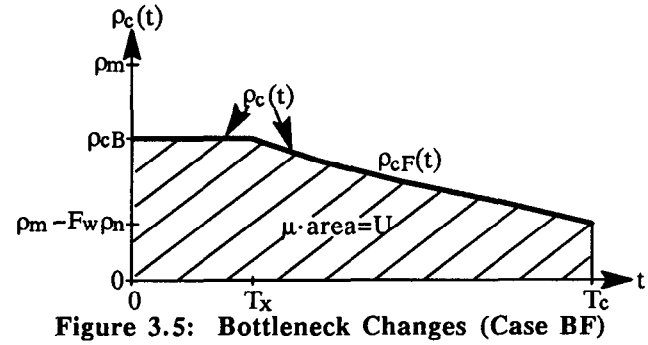

Figure 3.5: Bottleneck Changes (Case BF)

For this to occur, we must have

$$\rho_{nB} \ge \rho_n \ge \rho_{nF}$$

i.e., $\quad \frac{S-2}{S-2F_w}\rho_m \ge \rho_n \ge \frac{S-2}{S-F_w}\rho_m \quad ,$

or conversely $\quad S_B \le S \le S_F$

i.e., $\quad 2\frac{R-F_w}{R-1} \le S \le \frac{2R-F_w}{R-1} \quad .$

Notice that this case is unnecessary for F_w=0. Figure 3.5 illustrates that at the beginning of the recovery-copying process, $\rho_c=\rho_{cB}$, which is not a function of time; and during the remainder of the recovery-copying process, $\rho_c(t)=\rho_{cF}(t)$. The crossover time T_x occurs when

$$\rho_{cB} = \rho_{cF}(T_x) = \rho_m - F_w\rho_n\frac{u(T_x)}{U} .$$

$u(T_x)$ can be found using Equation 3-1 with $\rho_c=\rho_{cB}$, which does not vary in time:

$$u(T_x) = \mu\int_0^{T_x}\rho_{cB}\,dt = \mu\rho_{cB}T_x .$$

The above equation then becomes

$$\rho_{cB} = \rho_m - F_w\rho_n\frac{\mu\rho_{cB}T_x}{U}$$

or $$T_x = \frac{1}{q}\left(\frac{\rho_m}{\rho_{cB}} - 1\right) = \frac{1}{q}\left(\frac{R}{(R-1)S - R + F_w} - 1\right),$$

using Equation 3-2 for ρ_{cB}. Starting at T_x, u(t) is

$$u(t) = U(T_x) + \mu\int_{T_x}^{t}\rho_{cF}(t)dt, \text{ for } T_x\le t\le T_c .$$

Substituting the above expression for u(t) into Equation 3-3, we find $\rho_{cF}(t)$ described by the following differential equation:

$$\rho_{cF}(t) = \rho_m - \frac{F_w\rho_n}{U}\left(u(T_x) + \mu\int_{T_x}^{t}\rho_{cF}(t)dt\right), \quad T_x\le t\le T_c,$$

whose solution is

$$\rho_{cF}(t) = \rho_{cB}\,e^{-q(t-T_x)}, \text{ for } T_x\le t\le T_c .$$

We can find T_c by solving Equation 3-4 using the correct expression for ρ_c during each of the two time intervals:

$$U = \mu\int_0^{T_x}\rho_{cB}\,dt + \mu\int_{T_x}^{T_c}\rho_{cB}\,e^{-q(t-T_x)}dt .$$

For F_w=0, the solution is simple because $\rho_{cB}=\rho_m$, T_x=0 and q is infinite, so that

$$T_c = \frac{U}{\mu\rho_{cB}} = \frac{U}{\mu\rho_m}, \text{ for } F_w=0 .$$

For F_w>0, we have

$$U = \mu\rho_{cB}\Big|_0^{T_x} + \frac{\mu\rho_{cB}}{q}\,e^{-q(t-T_x)}\Big|_{T_x}^{T_c} .$$

Solving for T_c, we have

$$T_c = \frac{1}{q}\log\left(\frac{1}{qT_x + 1 - \frac{qU}{\mu\rho_{cB}}}\right) + T_x, \text{ for } F_w>0 .$$

Substituting for T_x and q, we have

$$T_c = \frac{U}{\mu F_w\rho_n}\left(\log\left(\frac{(R-1)S}{R-F_w} - 1\right) + \frac{R}{(R-1)S - R + F_w} - 1\right), \text{ for } F_w>0 .$$

We now have solutions for T_c for the three possible cases. We use these equations in Section 4 to compare interleaved and mirrored declustering, and to help determine the optimal value of S for interleaved declustering.

3.2 Mean Time To Checkpoint & Log Recovery

Because a checkpoint-and-log recovery technique is also employed, loss of both copies for the high-availability recovery techniques does not imply a complete loss of data. However, it typically causes a lengthy period during which data is unavailable, because checkpoint-and-log recovery is typically very time consuming. We define the mean time to checkpoint-and-log recovery (MTTCR) as the average time between a loss of both copies of any data. We now derive a formula for MTTCR.

The probability of any particular disk failing during T_{rec} is T_{rec}/mttf, where mttf is the average time between the loss of a particular disk. The probability of any particular disk not failing during T_{rec} is 1-T_{rec}/mttf. The probability of averting a second failure within the same recovery cluster is

$$\left(1 - \frac{T_{rec}}{mttf}\right)^{S-1} .$$

The probability of a second failure within the same cluster is

$$1 - \left(1 - \frac{T_{rec}}{mttf}\right)^{S-1} .$$

Finally, MTTCR is

$$MTTCR = \frac{mttf/N}{1 - \left(1 - \frac{T_{rec}}{mttf}\right)^{S-1}},$$

because mttf/N is the mean time between failure events. Usually T_{rec}/mttf<<1, so that we can expand and truncate the series to obtain

$$MTTCR \cong \frac{mttf/N}{1 - \left(1 - \frac{T_{rec}}{mttf}(S-1)\right)} = \frac{mttf^2}{NT_{rec}(S-1)} .$$

S_{CR} is defined as the value of S such that MTTCR is maximized. Note that T_{rec} is a function of S, so that S_{CR} is not necessarily 2. Because of the complexity introduced by T_{rec} into the MTTCR function, setting to zero the derivative of MTTCR wrt S results in a complex equation that is difficult to solve for S, but provides the insight that S_{CR} has the convenient property of only being a function of R and F_w. We will obtain SCR numerically for particular values of R and F_w in Section 4.

3.3 Response Time During Recovery Copying

In this section, we examine response time for user transactions during recovery copying. As S is increased, the negative impact on responsiveness due to a failure is spread over more transactions. More transactions will involve the larger Failed Disk's recovery cluster and therefore suffer a responsiveness degradation with larger S; however, the amount of this degradation is reduced with larger S. For those

transactions involving the Failed Disk's recovery cluster, we show when the responsiveness of those disks during recovery copying is within a certain percentage (say 25%) of their response time during normal operation.

To model disk response time, we make the following assumptions in addition to the assumptions described at the beginning of Section 3:

a) Interarrival and service times have an exponential distribution, for tractability.
b) Each disk queue has at least two levels of priority, which we call the "foreground queue" (fq) and the "background queue" (bq), with the foreground queue enjoying higher priority.
c) During normal operation, a checkpoint and log technique is used for recovery from system crashes [Gra78], where all reads are put in the foreground queue and all writes are put in the background queue, so that only the responsiveness of the foreground queue impacts user-transaction response time.
d) During recovery copying, reads for recovery copying are also entered in the background queue.
e) Disk accesses are non-preemptive.

Given a two-level priority queue, exponential interarrival and service times, and non-preemptive disk operation, the average response time for the foreground queue per [Kle76] (Vol. 2) is

$$rt = \frac{1}{\mu}\left(\frac{\rho}{1-\rho_{fg}} + 1\right), \text{ where } \rho = \rho_{fg} + \rho_{bg} .$$

During normal operation, $\rho=\rho_n$ and $\rho_{fg}=\rho_r=F_r\,\rho_n$, so that

$$rt_n \equiv \frac{1}{\mu}\left(\frac{\rho_n}{1-\rho_r} + 1\right).$$

During recovery copying, we are only concerned with the mean response time of the Backup Disks, because these disks determine the response time of user transactions. Responsiveness of the Backup Disks is worse during recovery copying for two reasons. One reason is that the read load is increased, so that disk queues will be longer. A second reason is that disk accesses are non-preemptive, so that the background copy operation increases disk-queue wait time. Figure 3.2 makes it apparent that the utilization of the Backup Disks during recovery copying can be characterized as

$$\rho = \rho_n + \frac{\rho_r + \overline{\rho_c(t)}}{S-1}$$

and $$\rho_{fg} = \rho_r + \frac{\rho_r}{S-1} = \rho_r \frac{S}{S-1},$$

where $\overline{\rho_c(t)}$ is defined as some "average" ρ_c within $0 \geq t \geq T_c$. This results in

$$rt_c \equiv \frac{1}{\mu}\left(\frac{\rho_n + \frac{\rho_r + \overline{\rho_c(t)}}{S-1}}{1 - \rho_r \frac{S}{S-1}} + 1\right).$$

Because we are interested in the worst-case response time during recovery copying, we pick the worst-case load on the Backup Disks. For $S \leq S_F$, $\max(\rho_c)=\rho_{cB}$, and for $S \geq S_F$, $\max(\rho_c)=\rho_m$, which is at the beginning of the recovery-copying process (t=0).

We define the metric S_{rt} as the minimum value of S such that rt_c does not exceed rt_n by a certain percentage:

$$S_{rt} \equiv S\left|\frac{rt_c}{rt_n} \leq \tau\right. ,$$

where $\tau>1$ is a target response-time ratio. As an example, if we wish rt_c to not exceed rt_n by more than 25%, then $\tau=1.25$.

From the above,

$$S_{rt} = \begin{cases} \dfrac{\tau\frac{1+\rho_n}{1-\rho_r} - 1 - \rho_m}{(\tau-1)(1+\rho_w) + \rho_n - \rho_m} & \text{, for } S \leq S_F \\ \dfrac{\tau\frac{1+\rho_n}{1-\rho_r} - 1 + \rho_m - \rho_w}{(\tau-1)(1+\rho_w)} & \text{, for } S \geq S_F . \end{cases}$$

Notice that S_{rt} is only a function of ρ_m, ρ_n, F_w and τ.

4 Architecture-Independent Comparison

This section uses the formulas and metrics of Section 3 to compare those aspects of the two recovery techniques that are independent of the hardware organization in which the disks are placed.

4.1 Advantages of Interleaved Declustering

The major advantage of interleaved declustering is that it allows S>2, while mirrored declustering has the restriction of S=2. This section illustrates the many performance advantages that accrue to interleaved declustering from this flexibility. Throughout this comparison, we assume a large number (N=500) of large disks (one actuator within an IBM AK4 drive), having 40,000 tracks (U=40,000 assuming a track is the copy unit for recovery and all tracks must be copied), average random seek time of 16msec and rotation time of 16.7msec (μ=30tracks/sec assuming all seeks are random and all accesses are for tracks), and mttf=30,000hrs. These assumptions yield pessimistic values for T_c and MTTCR.

Figures 4.1, 4.2 and 4.3 illustrate how T_c varies as S ranges from 2 through 30 for three different ρ_n's (low: 0.2, medium: 0.4 and high: 0.6), three different F_w's (low: 0.25, medium: 0.5 and high: 0.75)[1], and ρ_m=0.8. F_w is a function of the

[1] Note that the DebitCredit transaction [Ano85] has 3 reads and 4 "logical" writes (physical writes are twice). Because Fw is the fraction of physical writes, Fw=2*4/(3+2*4)=0.73, so that DebitCredit is close to the high Fw value if no caching is assumed.

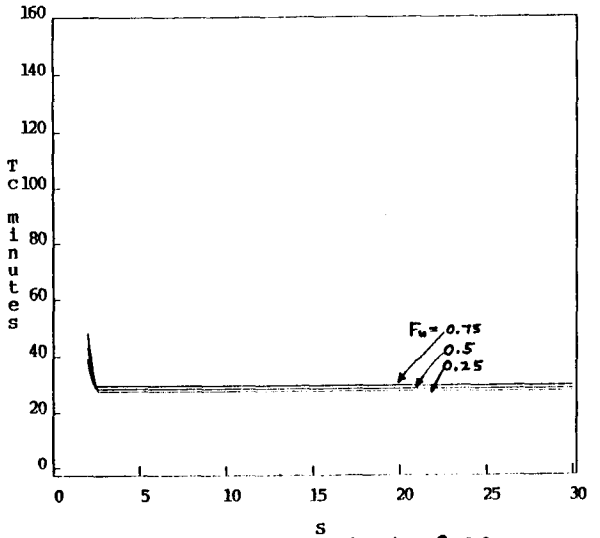

Figure 4.1: Recovery Copying Time (ρn-0.2)

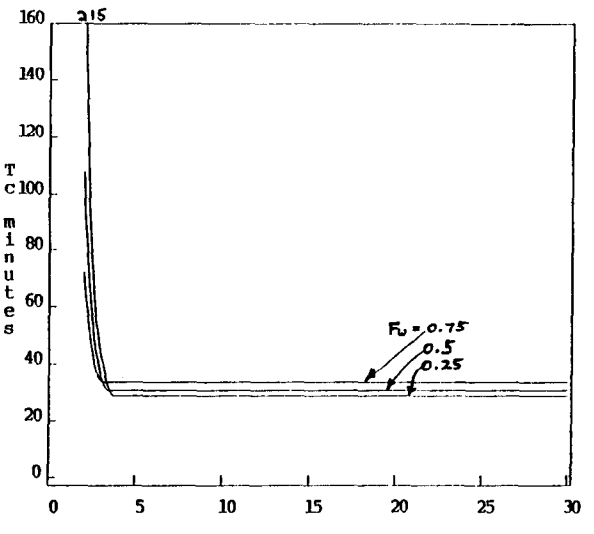

Figure 4.2: Recovery Copying Time (ρn-0.4)

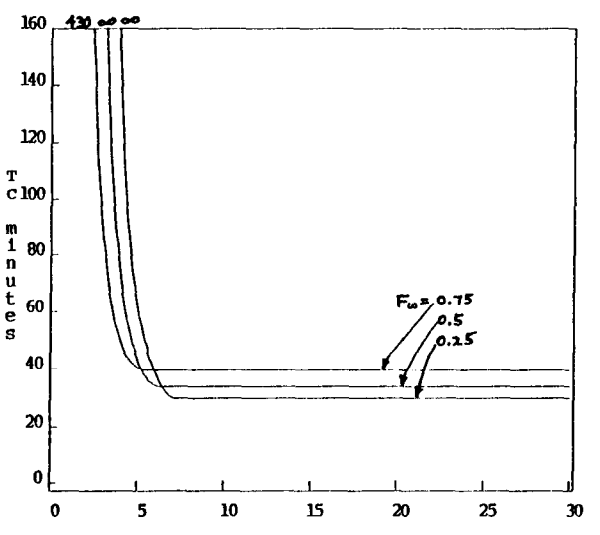

Figure 4.3: Recovery Copying Time (ρn-0.6)

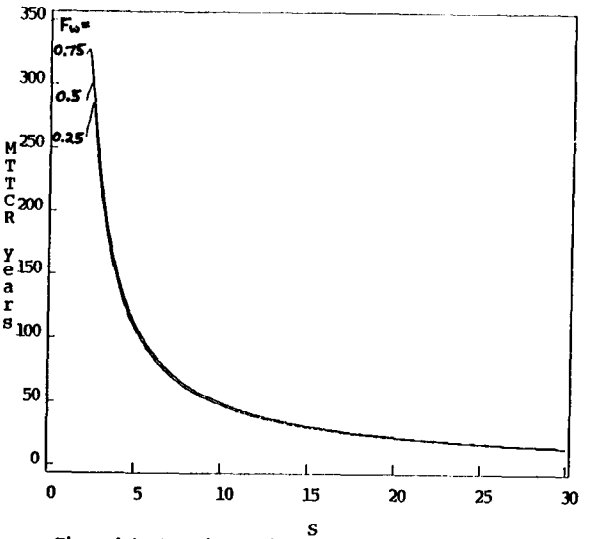

Figure 4.4: Mean Time To Checkpoint-And-Log Recovery (ρn-0.2)

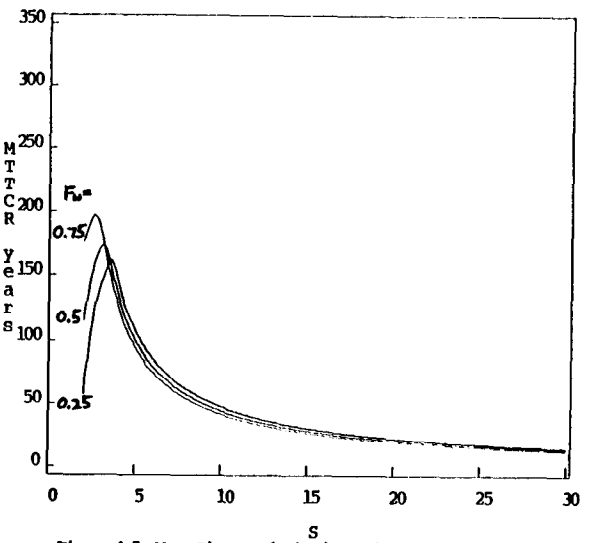

Figure 4.5: Mean Time To Checkpoint-And-Log Recovery (ρn-0.4)

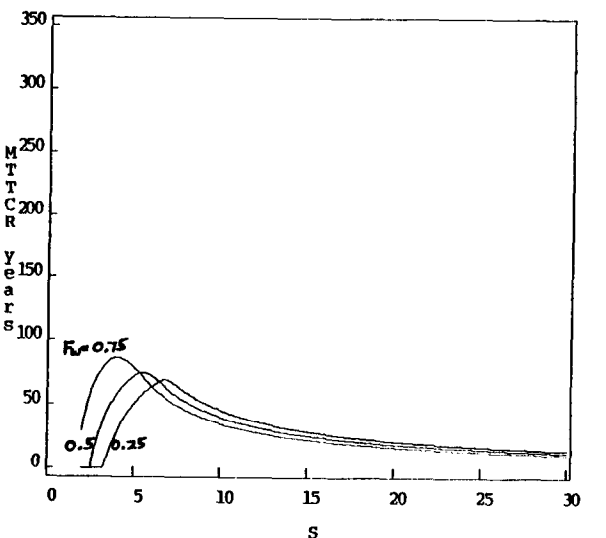

Figure 4.6: Mean Time To Checkpoint-And-Log Recovery (ρn-0.6)

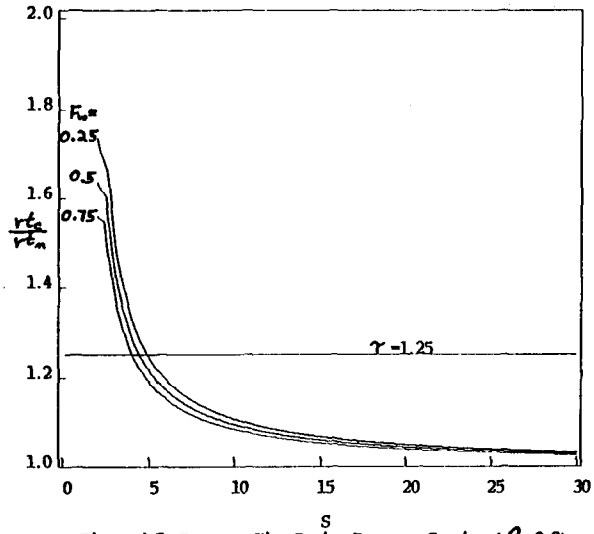

Figure 4.7: Response Time During Recovery Copying (ρ_n=0.2)

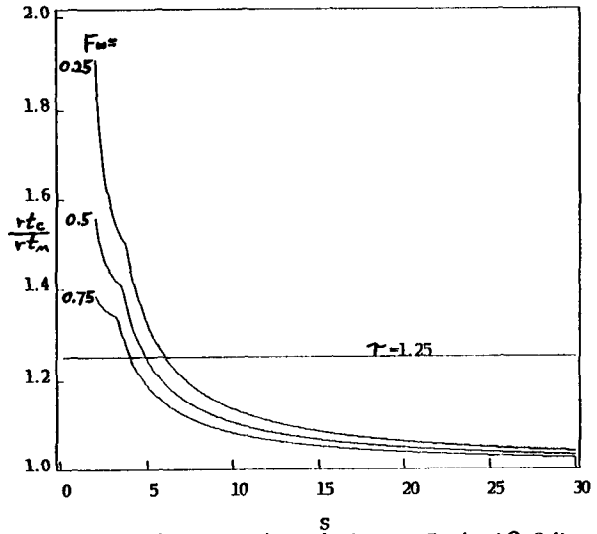

Figure 4.8: Response Time During Recovery Copying (ρ_n=0.4)

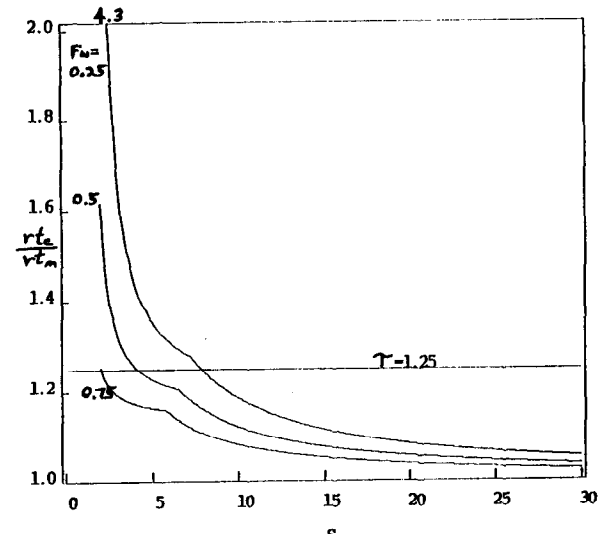

Figure 4.9: Response Time During Recovery Copying (ρ_n=0.6)

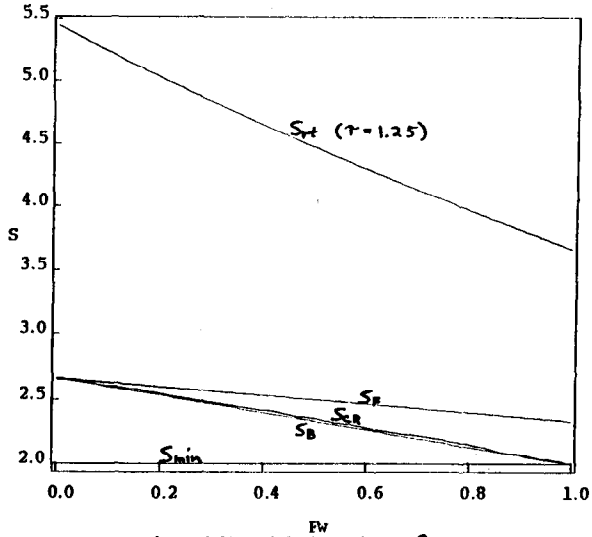

Figure 4.10: Critical S Values (ρ_n= 0.2)

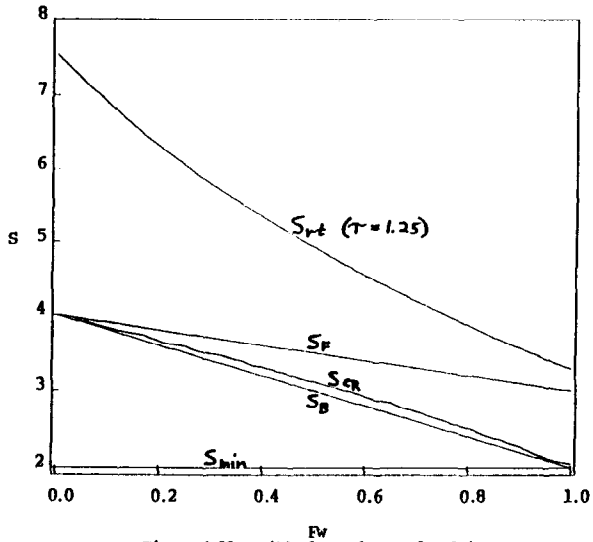

Figure 4.11: Critical S Values (ρ_n= 0.4)

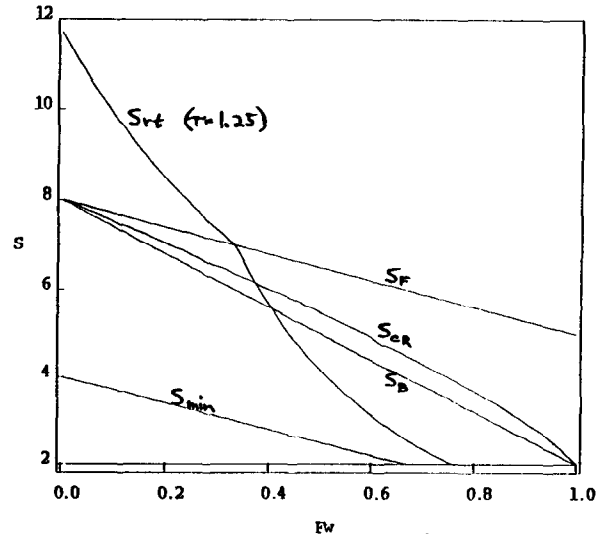

Figure 4.12: Critical S Values (ρ_n= 0.6)

workload. ρ_n is usually chosen to be just low enough to achieve the rt_n requirements of the application. These curves show that T_c for mirroring and other low S values can be unacceptably large. There are two ways to remedy this. One way is to further reduce ρ_n below what would be required to achieve the rt_n requirements of the application, which has a high cost because proportionately more disk arms are required to service the same load. A second way is to increase S, whose only cost is that MTTCR will get worse. Moderate values of S ensure near-optimal T_c, incurring the sole cost of increased MTTCR. We show in the next paragraph that moderate values of S cause MTTCR to be on the order of *decades*, so that this is a small cost to pay. The figures also illustrate that for larger S, T_c has low sensitivity to either ρ_n or F_W. Notice that larger F_W causes shorter T_c when the Backup Disks are the bottleneck (smaller S) because their read load is smaller, while larger F_W causes longer T_c when the Failed Disk is the bottleneck (larger S) because its update load is larger.

Figures 4.4, 4.5 and 4.6 illustrate how MTTCR varies with S for the same ρ_n, ρ_m and F_W values. We assume that any system designed for high availability would keep spare disks on-line, so that $T_{rec} \tilde{=} T_c$. These curves show that for low S, MTTCR gets worse quickly because T_c becomes large, and for high S, MTTCR gets worse gradually and has low sensitivity to either ρ_n or F_W. Even for S=30, MTTCR is more than 10 years. This means that in the worst case, the checkpoint-and-log recovery technique will be required for media failure once every 10 years, on the average. This indicates that MTTCR is not a critical design issue, so that increasing S is an effective way to decrease T_c.

Figures 4.7, 4.8 and 4.9 illustrate how rt_c/rt_n varies with S for the same ρ_n, ρ_m and F_W values. These curves show that the impact of recovery on response time can be large. For a 500 disk system with mttf of 30,000hrs, a recovery is required on average every 60hrs (typically more frequently during peak loads), and have a duration (T_c) which depends upon S. There are two ways to minimize this impact. One way is to further reduce ρ_n below what would be required to achieve the rt_n requirements of the application. This has a high cost because more disk arms are required, and the figures illustrate that ρ_n has a weak effect on rt_c/rt_n. A second way is to increase S. This has an insignificant cost because MTTCR is so large, and the figures illustrate that it has a strong effect on rt_c/rt_n as well as reducing the duration of the impact (T_c).

Figures 4.10, 4.11 and 4.12 illustrate how the formulas derived in this paper can be used as a practical guide in the choice of S. S_{min}, S_B, S_F, S_{CR} and S_{rt} are each dependent only on a few basic parameters that are usually well known for a specific system and workload. S_{min}, S_B, S_F and S_{CR} are dependent on R (ρ_m/ρ_n) and F_W, while S_{rt} is dependent on ρ_n, ρ_m, F_W and τ. F_W is a function of the type of operations in the workload. ρ_n is a function of both the workload's performance requirements and the system configuration. ρ_m is a function of the system's policy for limiting the impact of background disk accesses on foreground disk accesses. τ is a function of the workload's response time requirements. We use τ=1.25 in these figures; that is, mean disk response time is no more than 25% higher during recovery copying than during normal operation. For the sake of performance, S should be at least S_{CR}, because there is no cost to make S this large and T_c and MTTCR improve. Because MTTCR at larger S values is still quite large, S should be further increased to S_F to further improve T_c, or even further to S_{rt} (with some appropriate τ) to keep the impact on response time low.

Mirroring does not have the flexibility of allowing S>2. As a result, the only choice to improve T_c, MTTCR and rt_c/rt_n is to further reduce ρ_n below what would be required to achieve the rt_n requirements of the application. This has a high cost because proportionately more disk arms are required to service the same load. *This forces a choice between poor performance during normal operation, poor performance during recovery with long duration, or higher cost.* Interleaved declustering offers a low cost way to achieve both good performance during normal operation and good performance during recovery with short duration by allowing S>2.

Another advantage of interleaved declustering is that D, the number of disks containing one copy of a relation, can be twice as large for improved load balancing.

4.2 Advantages Of Mirroring

The major advantage of mirrored declustering is that it is much simpler to implement, because all recovery software can be kept within lower levels of the system software, so that recovery is transparent to upper software layers. For example, recovery can be kept entirely within the disk controller. This includes writes to both copies during normal operation, failure notification and recovery copying.

Another advantage of mirrored declustering is that D, the number of disks containing one copy of a relation, can be lower than for interleaved

declustering. For mirroring, D can be in increments of 1. For interleaved declustering, D must be in increments of S. Thus, unlike interleaved, declustering is not required for mirroring. This makes mirroring more universally applicable. It works for any computer application, not just databases with sets of tuples.

5 Architecture-Dependent Comparison

This section compares those aspects of the two recovery techniques that behave differently in different hardware organizations.

5.1 Architecture Classifications

This section describes six types of database system hardware organizations. This set of architectures is necessarily incomplete; however, we hope to have included enough types to allow the reader to readily analyze other types.

Figure 5.1 illustrates the six architectures. Each has processors (p), memories (m), disks (d), local buses and a global interconnect. Different architectures would require different types of global interconnects (message-passing vs. bus). We assume that shared disks have some sort of centralized control that is aware of the status of the different disk queues. That is, SD and SE have globally centralized disk-queue status, and CD and CE have centralized disk-queue status within each cluster.

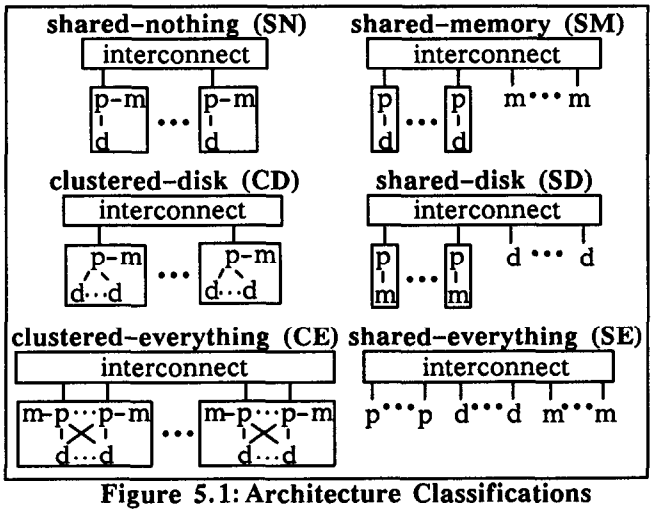

Figure 5.1: Architecture Classifications

[Ter85], [DeW86] and [Cop88] are examples of SN, although CD is a natural extension of these as processors grow faster wrt disks; [Tan87] is an example of CE usually with a cluster size of two; most networks are examples of SD; and most mainframe systems are examples of SE usually with a small number of processors and memories.

5.2 Indexing

Because SN, CD, CE and SM do not globally share disks, they typically have both global and local cluster indexes. The "global index" is typically replicated in each node and describes which node contains which data. The "local index" is unique to each node and describes where data lives within the node. When interleaved declustering is used in such architectures, there is a policy decision of whether to force the top level of the local index to use the same clustering as the global index or to allow the two indexes to be independent. Using the same clustering has the advantage of fewer disk accesses to find the data contained in another node during recovery. Using different clusterings requires all data on every Backup Disk to be read.

To illustrate this, let us assume that D0 in Figure 2.2 (S=4) fails. To recover, D1, D2 and D3 must each access what was lost in D0. If the local and global indexes are independent, then their clustering will differ, so that all disk pages in D1, D2 and D3 must be accessed. Using the same clustering is illustrated in Figure 5.2. In this case, only disk pages containing r0.0 and R1.2 must be accessed in D1, with a similar savings in D2 and D3. In general, this approach requires only 1/(S-1) of each disk to be accessed (notice that for S=2, this is not an issue), so that recovery can be made faster with larger S. The same clustering of indexes was assumed in the performance modeling in Section 3 (the λ_c term in the load for the Backup Disks in Figure 3.2 was divided by S-1). Without this assumption, larger S would still have the advantage of spreading out the additional read load across the S-1 Backup Disks.

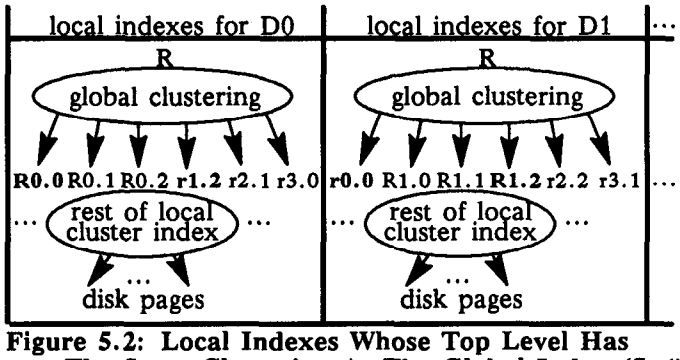

Figure 5.2: Local Indexes Whose Top Level Has The Same Clustering As The Global Index (S=4)

The disadvantage of using the same clustering is that it forces more centralization on SN, CD, CE and SM, because local indexing must be synchronized with global indexing. However, if the size of CD and CE clusters matches that of recovery (S), then this issue is moot because the appropriate level of centralization already exists.

5.3 Performance During Normal Operation

Because reads require access to only one copy, there is a policy decision regarding which copy to access for reads. One policy, used in [Ter85], is to always read the same "primary" copy during normal operation, which has the advantage of disk

independence. A second policy is to read the copy from the disk with the shortest queue, which has the advantage of better throughput via dynamic load balancing. A third policy, used in [Tan87], is to read the copy from the disk with the shortest seek time, which has the advantage of improved response time for reads [Bit88].

Use of either the shortest-queue or shortest-seek policy with CE results in "cache dilution", where both copies of the same hot data exists in different caches. Either the shortest-queue or shortest-seek policy is easy to implement on CD, CE, SD and SE because disks are centrally controlled, but is difficult on SN and SM because disks are controlled independently. Interleaved declustering has the advantage of improving load balancing because it allows twice as much declustering. Thus, interleaved declustering may offer sufficient load balancing for CE (without causing cache dilution by using the shortest-queue policy) and for SN and SM (where the shortest-queue policy is difficult). Because the throughput and response time can be traded, it is difficult to judge which combination of recovery techniques and disk-read policies are most cost-effective without a more thorough cost-performance analysis using knowledge of workload requirements.

6 Summary

Mirroring offers greater simplicity and universality. Recovery can be implemented at lower levels of the system software (e.g., the disk controller). For architectures that do not share disks globally, it allows global and local cluster indexes to be independent. Also, mirroring does not require data to be declustered.

Interleaved declustering allows S>2, which can significantly improve recovery time, mean time to loss of both copies of some data, throughput during normal operation, and responsiveness during recovery. As S is increased, the negative impact on responsiveness due to a failure is spread over more transactions with less impact on each transaction. The mirroring restriction of S=2 forces a choice between poor performance during normal operation, poor performance during recovery with long duration, or higher cost. For all architectures, interleaved declustering allows data to be declustered over twice as many disks for improved load balancing at the expense of imposing a larger increment for declustering. We showed how the choice of S is simplified because it is dependent only on a few parameters that are usually well known for a specific workload and system configuration.

Acknowledgements

Thanks to Marc Smith and Bill Alexander for their many helpful comments.

References

[Ano85] Anon. et al, "A Measure of Transaction Processing Power," *Datamation*, Vol. 31.7, April 1 (1985).

[Att84] R. Attar, P. Bernstein and N. Goodman, "Site Initialization, Recovery And Backup In A Distributed Database System," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 6 (November 1984).

[Bit88] D. Bitton and J. Gray, "Disk Shadowing," *Proceedings of VLDB*, Los Angeles (August 1988).

[Cop88] G. Copeland, W. Alexander, E. Boughter and T. Keller, "Data Placement In Bubba," *Proceedings of ACM SIGMOD* (May 1988).

[DeW86] D.J. DeWitt, R.H. Gerber, G. Graefe, M.H. Heytens, K.B Kumar and M. Muralikrishna, "GAMMA---A High Performance Dataflow Database Machine," *Proceedings of VLDB*, Japan (August 1986).

[Gra78] J.N. Gray, "Notes on Database Operating Systems," in *Operating Systems: An Advanced Course*, Springer-Verlag, New York (1978).

[Kat88] R. Katz, J. Ousterhout, D. Patterson and M. Stonebraker, "A Project On High Performance I/O Subsystems," *Database Engineering*, IEEE Computer Society, Vol. 11, No. 1 (March 1988).

[Kat78] J.A. Katzman, "A Fault-Tolerant Computing System," *Proceedings of the Eleventh Hawaii Conference on System Sciences,"* (January 1978).

[Kle76] L. Kleinrock, *Queuing Systems, Volume 1: Theory*, and *Queuing Systems, Volume 2: Computer Applications*, John Wiley & Sons, New York, pp. 119-126 (1976).

[Liv87] M. Livny, S. Khoshafian, H. Boral, "Multi-Disk Management Algorithms," *ACM SIGMETRICS Conference* (1987).

[Par86] A. Park and K. Balasubramanian, "Providing Fault Tolerance In Parallel Secondary Storage Systems," *Computer Science TR 057-86*, Princeton University (November 1986).

[Pat87] D.A. Patterson, G. Gibson and R.H. Katz, "A Case For Redundant Arrays Of Inexpensive Disks (RAID)," *Report No. UCB/CSD 87/391*, U.C. Berkeley Computer Science Division (December 1987) and *Proceedings of ACM SIGMOD*, Chicago (May 1988).

[Sto86] M. Stonebreaker, "The Case For Shared Nothing," *Database Engineering*, Vol. 9, No. 1 (1986).

[Tan87] The Tandem Database Group, "NonStop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL," *Proceedings of 2nd International Workshop on High Performance Transaction Systems*, Asilomar, CA (September 1987).

[Ter85] "DBC/1012 Data Base Computer System Manual, Release 1.3," *C10-0001-01, Teradata Corp.*, Los Angeles (February 1985).