FAULT TOLERANCE IN A VERY LARGE DATABASE SYSTEM: A STRAWMAN ANALYSIS

Amit P. Sheth
UNISYS West Coast Research Center
2400 Colorado Ave
Santa Monica 90406

ABSTRACT

A system for very large databases will require a high degree of parallelism and, therefore, will involve a large number of components. In such a system, many components can fail, affecting the performance of active components and the availability of the system. Designing such a system with a desired degree of fault tolerance is difficult and requires qualitative and quantitative considerations of factors affecting system fault tolerance.

In this paper, a simple model is used to study the effect of fault tolerance techniques and system design on system availability. A generic multiprocessor architecture is used that can be configured in different ways to study the effect of system architectures. Important parameters studied are different system architectures and hardware fault tolerance techniques, mean time to failure of basic components, database size and distribution, interconnect capacity, etc. Quantitative analysis compares the relative effect of different parameter values. Results show that the effect of different parameter values on system availability can be very significant. System architecture, use of hardware fault tolerance (particularly mirroring) and data storage methods emerge as very important parameters under the control of a system designer.

Key Words: availability, fault tolerance, very large database, multiprocessor architecture, disk mirroring, processor pairing, recovery, reconfiguration, mean time to failure, mean time to repair.

1. Introduction

The topic of fault tolerance in computers and database system has received a fair degree of attention. Articles by Kim [Kim 84] and Siewiorek [Siewiorek 84] discuss architectures of fault-tolerant computers and fault tolerance techniques used in many commercial and prototype systems. Some of the publications that discuss fault tolerance techniques used in commercial and prototype computers are [Siewiorek et al 78], [Borr 81], [Kastner 83], [Borr 84], [Bernstein 85], and [Gray 86]. Recently, work continues on larger systems, such as Teradata DBC1012 [Decker 86], but not much has been published on the fault tolerance techniques used in such systems or on evaluation of such techniques. While it seems that many of the techniques developed for smaller distributed systems are applicable to larger multiprocessor systems, designing fault tolerance for large multiprocessor systems is more difficult because they have many more components that can fail and there are more alternatives to provide fault tolerance. [Koon and Ozsu 86] and [Sheth et al 87] evaluate fault tolerant algorithms in a relatively small (~ 10 nodes) and loosely coupled distributed systems. However, there is no published research on evaluating fault tolerance in a tightly coupled multiprocessor system for large databases (> 100 gigabytes).

In this paper, we study the effects of various system parameters and fault tolerance techniques in a system for very large databases. We use a generic multiprocessor architecture which, by choosing different system parameters can represent a range of system architectures, from a loosely coupled multiprocessor with non-shared memory and partitioned database to a tightly coupled multiprocessor with shared database and shared memory. We study the effects of various architectures, fault tolerance techniques, mean time to failure of important components, database size and distribution, interconnect capacity, etc. on the availability of a system. The results show that the choice of different system architectures and some parameter values significantly affect availability. Because fault tolerance is obtained at the cost of additional system resources (in terms of number of redundant components and system time and resources required to maintain redundant information), a designer's task is to minimize such cost to obtain the desired level of fault tolerance. The results can help a designer to understand important trade-offs and choose an appropriate system architecture and fault tolerance techniques.

This paper is organized as follows. Section 2 describes the generic system architecture we evaluate. Section 3 introduces basic concepts and terminology relevant to the fault tolerance techniques. Section 4 is the main part of the paper. It describes the quantitative analysis and the results. Section 5 discusses our conclusions and how this study can be extended.

2. System Description

Achieving good performance in a very large database system (VLDBS) requires a large amount of computing power. This requires a high degree of parallelism, given the limitation of the computing power of a single component. A VLDBS with a high degree of parallelism is expected to have the following characteristics:

1. It has many major components such as processors, disks and memory.
2. The relations are very large and distributed over many components for storage and parallel-processing. Because of the very large size of the database, the amount of replication possible is limited.
3. An update transaction will change many data items. This means that reintegration of failed components can take longer.

We assume a generic multiprocessor architecture for a VLDBS, as shown in Figure 2.1. This architecture consists of a set of "clusters" linked by an interconnect. Each cluster consists of a set of processors, a shared memory bank addressable by all processors in the cluster, and a set of disk storage units and associated controllers. Each cluster has its own power supply. All the components within a cluster (processors, disks/controllers, memory and power supply) are connected by an intracluster bus. The size of a VLDBS is defined by the architectural parameters given in Table 2-1. A *component unit* (e.g., a processor unit or disk unit) consists of one or more components. A component unit consisting of k components is called a *k-redundant* unit in which $k-1$ components are redundant components used to increase fault tolerance. However, all active components in a unit perform the same function, so the redundancy does not add to the computing power (in the case of the processor unit) or storage capability (in the case of the disk unit or memory unit). Our architecture has one power supply unit, one memory unit, and an intracluster bus per cluster. However, each component unit may contain redundant components.

This work was supported in part by Rome Air Development Center (Contract #F30602-85-C-0215) and was performed for the Very Large Parallel Data Flow (VLPDF) project at Honeywell Corporate Systems Development Division, Golden Valley, MN.

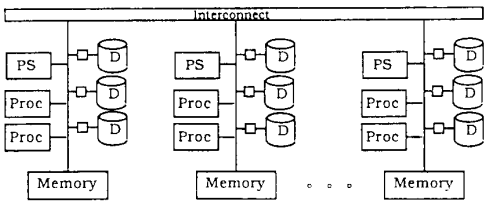

Figure 2.1. Architecture of a VLDBS

NC	Number of clusters
NP	Number of processor units per cluster
ND	Number of disk units per cluster

Table 2-1: Architectural Parameters

3. Fault Tolerance Techniques

A component or a subsystem that functions according to its specifications is called *active*. Three terms are relevant to the operation of the components and the system: fault, error, and failure [Siewiorek and Swarz 82, Avizienis and Kelly 84]. A *fault* is a defect in a component of the system. *Faults* result in *errors*, which are undesired or invalid component states. Errors may result in *failures*, which mean loss of the service expected from the component (or the system of which the component is a part). Errors may be transient, intermittent, or permanent. The first two are also referred to as soft errors, while permanent errors are referred to as hard errors. Siewiorek and Swarz [Siewiorek and Swarz 82] estimate that soft errors account for more than 90% of all faults. Failures from soft errors are called *soft failures*, while those from hard errors are called *hard failures*. The fault tolerance techniques should protect the system against soft as well as hard failures.

After the failure of a component (or subsystem), the system may take a corrective action called *failure recovery*. This may involve *reconfiguring* the system to isolate the failed component and to reorganize the system so that it can be restarted without the failed component. Once the failed component is repaired, it is *reintegrated* with the system. In a multiple-component system the failure of a single component that affects no other component during the recovery is called a *single failure*. Simultaneous failure of more than one component, or a failure of a component while the system is recovering from a previous component failure are called *multiple failures*. Normally, the probability of a multiple failure is very low. After a failure of one or more components, a system that can continue to operate at lower efficiency corresponding to the loss of power associated with the failed components is called a *gracefully degradable* system.

Redundancy is a basic property required for all fault tolerance techniques. It is used to provide information needed to negate the effect of failures [Siewiorek 84]. Redundancy can be obtained by having extra components, or *physical redundancy*, and by having extra time, or *temporal redundancy*. Physical redundancy is used as hot standbys (or backups), as checkers that mask faults via voting, and to reconfigure the system around faulty components. Temporal redundancy is used for retrying operations to recover from transient or soft errors.

The basic physical redundancy techniques used to increase the availability of each of the components include duplication (2-redundancy), triplication (3-redundancy) and voting, and k-redundancy. In this paper, we evaluate effect of duplication of each of the major system components as well as k-redundancy in processors. Duplication strategies for processors, called processor pairing [Kastner 83, Bernstein 85], and for disks, called disk mirroring [Gray 86], are well known. Processor pairing allows masking of soft failures at instruction level. k-redundancy in processors increases tolerance of hard failures. Disk mirroring, or duplexing, is used to increase this MTTF. Duplication is used similarly in memory, power supply and interconnect.

Availability of the database greatly depends on how the data is distributed and duplicated. The data fault tolerance is attained by proper placement of data. Our primary method is to have two copies of each data item. To do this, we first horizontally fragment a relation and assign different horizontal fragments to different clusters. A fragment assigned to a cluster is further vertically fragmented, and different vertical fragments are assigned to different disk units in a cluster. In this scheme, multiple disk units within a cluster are used to improve response time but not fault tolerance.

A good data placement strategy should meet the following three principles:

1. The database is partitioned across the clusters. Thus the failure of a cluster will make only the fragment(s) assigned to it unavailable.
2. Each partition assigned to a cluster is uniformly distributed over all the disk units in that cluster. It may be noted that the purpose of having multiple disks in a cluster is to increase the parallelism and not the fault tolerance.
3. The two copies of data (fragment in our scheme) cannot reside in the same cluster for the sake of fault tolerance.

Many other fragment allocation schemes are possible (e.g., see [Lu et al 87]). One such scheme assumed for the performance evaluation later is illustrated next.

Consider dividing the database into four fragments so that two copies of each fragment are distributed over four clusters. One possible data distribution scheme is shown in Figure 3.1. The database will remain available if there is a single failure of clusters because one copy of every fragment will still be accessible. The database will also be available in two out of a possible six double failures of clusters.

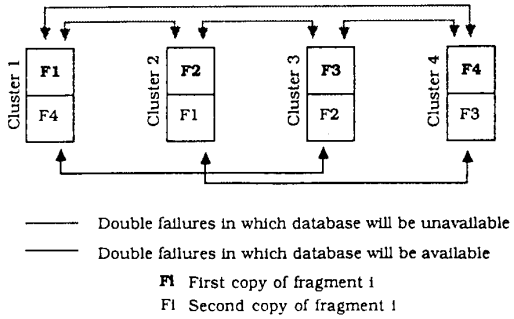

Figure 3.1. Data Placement Scheme

To illustrate how the system can be reconfigured in terms of data placement after one or more single failures of clusters, consider the failure of cluster 1. In this case, one copy each of fragment F1 and fragment F4 will be lost, and the other three clusters have only one copy of fragments F1 and F4. Although it is possible to continue processing the transactions, it is not desirable to do so, because it will not allow the system to be gracefully degradable. Failure of one more cluster may leave the system with no copy of a fragment. For example, failure of cluster 2 would mean that no copy of fragment F1 will be available, and so the system can no longer process transactions.

The solution is to create a second copy of the fragments that are unavailable because of a cluster failure. Following the data placement rules discussed above, the system of three working clusters can be reconfigured as shown in Figure 3.2. Following the same arguments, if cluster 3 were to fail after reconfiguration before cluster 1 failed (as shown in Figure 3.2) but before cluster 1 is reintegrated, the system can be reconfigured as shown in Figure 3.3. Because at least two copies of data must be kept in the working system and the two copies cannot be stored on the same cluster, the system can tolerate faults until two clusters are left in the working system.

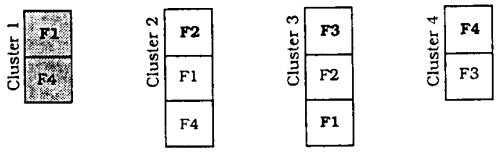

Figure 3.2. Data Placement After Cluster 1 Fails

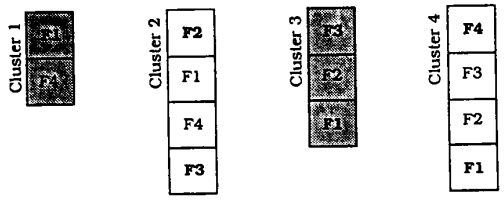

Figure 3.3. Data Placement After Cluster 3 Also Fails

After a failed cluster is repaired, it is reintegrated into the system. This involves replacing (or updating) the copies of the fragments on the repaired cluster with the up-to-date copy in the rest of the working system. For example, if cluster 1 was repaired after the situation shown in Figure 3.3, fragments F1 and F4 will be updated using either of the two up-to-date copies on clusters 2 and 4, fragment F1 will be deleted from cluster 4, and fragment F4 will be deleted from cluster 2. The process of replacing (or updating) can occur in the background for the active clusters (clusters 2 and 4 in this case) for most of the time. However, just before cluster 1 is made active, there may be a brief pause in the system to allow for the reallocating transaction and for updating the global data allocation directories. We leave out some details of cluster reintegration.

In addition to the hardware and data fault tolerance techniques described above, a system needs software fault tolerance techniques, including transaction fault tolerance and system fault tolerance. Table 3-1 summarizes fault tolerance techniques for various components.

Component	Failure Type	Failure Handling Method
Processor	Soft Failure Hard Failure	Processor Pairing Transaction Recovery, or Software Reinitialization, or System Reconfiguration (in extreme cases)
Disk	Soft Failure Hard Failure	Error Correction Codes and Hardware Techniques Disc Mirroring, or System Reconfiguration
Memory	Soft Failure Partial Failure Hard Failure	Parity and Error Correction Codes, or Transaction Recovery, or Software Reintialization Memory Reorganization Memory Mirroring, or System Reconfiguration
Power Supply	Soft Failure Hard Failure	Hardware Techniques System Reconfiguration
Intracluster Bus	Soft Failure Hard Failure	Hardware Techniques Treated as failure of associated component
Interconnect	Soft Failure Hard Failure	Protocol/Retries, and Error Correction Codes Redundant Paths

Table 3-1. Failure Tolerance Techniques As Applied To Various Failures

4. Quantitative Analysis

In this section, we will study the availability of a VLDBS. A VLDBS comprises the subsystems of components. In subsection 4.1, we study the availability of subsystems that consist of multiple non-redundant and redundant components. In subsection 4.2, the parameters used in the quantitative analysis are given. In subsections 4.3 and 4.4, we study the mean time to failure of a cluster and time to recover from a cluster failure, respectively. These two parameters are used to calculate system availability of a VLDBS in subsection 4.5. Subsections 4.3, 4.4 and 4.5 discuss basic quantitative methods to calculate output parameters using input parameters, followed by examples and evaluations using a range of input parameter values.

4.1. Subsystem Availability

The parameter used most widely to characterize the fault tolerance of a system is its availability. To measure the availability of a system and to devise fault tolerance techniques to achieve the desired level of system fault tolerance, we should also be able to measure availability of each of the system components.

Two parameters of a component are relevant in calculating a component's availability: MTTF, mean time to failure, and MTTR, mean time to repair. MTTF refers to the average time that elapses between two consecutive failures of a component. MTTR refers to the average time needed to detect and repair a failed component. The availability of a component can simply be given by the ratio,

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

Estimates of the MTTF and MTTR of each component of a system should be known before system availability can be estimated. Table 4-1 gives the estimates of MTTF for the components that we will use in our analysis.

Component	MTTF Estimates
Processor	100K hours
Memory (64 MB RAM)	100K hours
Disc	10K hours
Communication	100K hours
Power Supply	100K hours

Table 4-1. Component MTTF Estimates

Availability of a fault tolerant database system can be defined in several ways. We define it as the percent of the time the *entire* database is available for access by authorized users, i.e.,

$$Avail = \frac{time\ the\ entire\ database\ is\ available}{total\ time} \times 100 \tag{2}$$

Two reasons for using this definition of the availability are [Kim 84, Smith 86]: it is easy to quantify and it is easier to justify that the specified fault tolerant techniques meet certain availability goals.

Next we calculate the availability of subsystems consisting of multiple nonredundant components and multiple redundant components.

Availability of Subsystems Consisting of Nonredundant Components

A system with no redundant components requires that all the components are active for the system to be active. Such a system is also called a *series system*. Let the mean time to failure of the i th component be $MTTF_i$. Then the mean time to failure of a system of n components, $MTTF_{ser\text{-}sys}$, is given by:

$$MTTF_{ser-sys} = \frac{1}{\sum_{i=1}^{n} \frac{1}{MTTF_i}} \tag{3}$$

Availability of Subsystem Consisting of Redundant Components

Consider a subsystem that has reconfigurable duplication as in disk mirroring. Such a subsystem consists of a pair of identical components working in parallel and performing the same task. The subsystem can perform the required task as long as at least one of the two components is active. Using a combinatorial model of system availability, we can map this subsystem to a *two module, two repairman model* (or alternatively a Markovian queue called M/M/2/2/2 queue). Now availability of this subsystem, A_{pair} can be given as follows [Siewiorek and Swarz 82, page 280]:

$$A_{pair} = \frac{2\lambda\mu + \mu^2}{\lambda^2 + 2\lambda\mu + \mu^2}\ where \tag{4}$$

$$failure\ rate = \lambda = \frac{1}{MTTF}\ \text{and}\ repair\ rate = \mu = \frac{1}{MTTR}$$

Repair time for this subsystem, $MTTR_{pair}$, is given by $MTTR/2$, because the subsystem is modeled as failed when both the components have failed, and it is repaired when either of the two components work. This happens at the rate 2μ. Now, the mean time to failure of the subsystem, $MTTF_{pair}$, is given by using equation 1.

The above two component system can be extended to a k component subsystem with k-1 redundant components. This subsystem fails only when none of its components is active. It can be modeled by a k module, k repairmen model (or an M/M/k/k/k queue). Availability of this system, $A_{k\text{-}redundant}$, can be given by extending equation 4 as follows.

$$A_{k-redundant} = \frac{(\lambda+\mu)^k - \lambda^k}{(\lambda+\mu)^k} \quad (5)$$

Practically, this availability quickly approaches 1 for the values of the mean time to failure given in Table 4-1. We can show that the mean time to repair for this subsystem is inversely proportional to k. Corresponding value of the mean time to failure of the subsystem will be very large as compared with the mean time to failure of the individual component.

Availability of Hardware Components with Redundancy

Now let us look at the effect of some of the hardware fault tolerance techniques. Table 4-2 gives the parameters of interest.

$MTTF_{disk}$	mean time to failure of a single disk
$MTTF^{proc}$	mean time to failure of a single processor
$MTTF^{mem}$	mean time to failure of a single memory
$MTTF^{int}$	mean time to failure of a single interconnect
$MTTF_{ps}$	mean time to failure of a single power supply

Table 4-2. Component Fault Tolerance Parameters

Effect of Disk Mirroring: Disk mirroring is used to tolerate hard failures of disk units. By disk mirroring, the availability of the disks can be greatly improved. However, this fault tolerance is achieved at the cost 100% overhead in number of components.

Example: Consider a subsystem consisting of mirrored disks with $MTTF_{disk}$ to be 10K hours and the mean time to repair, $MTTR_{disk}$ to be 24 hours. By using equation 4, availability of a disk unit of mirrored disks, A_{du} evaluates to 0.999942675 and the mean time to failure, $MTTF_{du}$ to 239 years. A similar duplication scheme can also be used for memory, interconnect and power supply. For $MTTF_{mem}$ of 100K hours, the mean time to failure of a memory unit consisting of mirrored memory will be 23,793 years!

Effect of Processor Pairing: The use of redundancy in processor pairing is quite different from that in disk mirroring. This is because disk mirroring is used to tolerate hard faults, while processor pairing is used to tolerate soft faults. By processor pairing, all the soft faults are detected. Since many of the errors are transient, many of the faults do not recur. For recurring soft faults, the transaction is aborted or the system code is reinitialized. If the soft error is persistent after several retries, it is manifested as a hard failure. The mean time to failure corresponding to the soft faults is estimated to be 10K hours or about ten times as frequent as the hard failures. The processor pairing tolerates these soft faults. For any hard failure, an interrupt is generated and the processor pair detaches itself from the system. Thus, unlike disk mirroring, the processor pair does not continue to work if one of the two processors in a pair fails.

Example: For a processor pair to be active, both the processors should be active. Thus if mean time to failure of each processor, $MTTF_{processor}$, is 100K hours, then by using equation 3, mean time to failure of a processor unit, $MTTF_{pu}$, consisting of processor pair will be 50K hours.

Effect of Processor Redundancy: Processor redundancy is used to tolerate hard failures that disable processor units (which may be a single processor or a processor pair). This is done by using multiple processor units in a cluster (i.e., *k-redundancy*). Since at least one processor unit should be active for the cluster to be active, processor redundancy decreases the probability of a cluster failure due to processor failures. Processor redundancy also contributes to an increase in the processing power.

Example: If a cluster contains two processor units, such that each unit is a processor processor pair with the mean time to failure of each unit, $MTTF_{pu}$ to be 50K hours, then the mean time to failure of the cluster *due to processor failures*, $MTTF_{procs}$ will be 5950 years. If a cluster contains three processor units, $MTTF_{procs}$ will be 8,269,650 years!

4.2. Parameters

Many parameters affect the availability of a system. Table 4-2 defined component parameters. Table 4-3 defines system parameters. The last three parameters in Table 4-3 are called output parameters; the rest are called input parameters. Range or multiple values of input component and system parameters are used to study the effect of parameter values on system availability. Table 4-4 gives range values and default values used in our quantitative evaluations. If the value of a parameter used during a calculation is not explicitly mentioned, then the default values should be assumed.

NC	number of clusters in the system
ND	number of disk units per cluster
NP	number of processor units per cluster
DBS	database size
DF	distribution factor
C	interconnect capacity
PS	page size
I	I/O access time
$MTTF_{cluster}$	cluster MTTF
$MTTR_{cluster}$	MTTR for a cluster failure
Av	system availability

Table 4-3. Important System Parameters

The distribution factor (DF) is given by the average number of clusters on which the second copy of data stored on one cluster is stored. It depends on the data placement scheme used by the system (see [Lu et al 87] for more details). I/O access time is the time to transfer one page from a disk to memory.

Depending on the component fault tolerance techniques used in a system, we get different system configurations. We identify three system configurations of interest to our study. These configurations are given in Table 4-5.

Parameter	Values/Range	Default
$MTTF_{disk}$	10K or 30K (hours)	*30K (hours)*
$MTTF^{proc}$	100K (hours)	-
$MTTF^{mem}$	100K (hours)	-
$MTTF^{int}$	100K (hours)	-
$MTTF_{ps}$	100K (hours)	-
NC	1024 to 2	-
ND	1 to 512	-
NP	1, 2 or $ND/2$	$ND/2$
DBS	10**11 (100 Giga) or 10**12 (1 Tera) byte	10**12 (byte)
DF	1, $NC/2$ or NC-1	$NC/2$
C	100M, 500M, or 1000M (bits)	500M (bits)
PS	4K, 16K, or 64K (bytes)	16K (bytes)
I	20 ms	-

Table 4-4. Input Parameter Values

Configuration 1	single disk, processor pair, single memory, single interconnect, single power supply
Configuration 2	mirrored disk, processor pair, single memory, single interconnect, single power supply
Configuration 3	mirrored disk, processor pair, dual memory, dual interconnect, dual power supply

Table 4-5. System Configuration

To keep the analysis simple and apply existing modeling techniques without loosing the focus of the study, following simplifying assumptions are made:

1. Each system component or module is fail fast, i.e., it either functions properly or stops [Schlichting and Schneider 83].

2. Each component fails and recovers independently.
3. Mean time to failure of a component is exponentially distributed with mean MTTF. Similarly, mean time to repair of a component is exponentially distributed with mean MTTR.
4. Components (particularly interconnect) are lightly loaded during recovery. To include the queuing effect in calculating $MTTR_{cluster}$, a more detailed model (e.g. [Sheth et al 85]) needs to be used.
5. All the clusters and components of the same type are identical.

We also assume that the total number of disk units in the system will be 1024 (i.e., $NC * ND = 1024$). If each disk unit uses mirroring, there will be twice as many disks in the system. We will assume the data placement scheme maintains two copies of data, i.e., the total amount of storage capacity required in the system will be twice that of DBS. Thus if the DBS is 100 gigabytes, then each disk unit should have a storage capacity of (100 * 2 / 1024) or nearly 200 megabytes. If the DBS is 1 terabyte, then each disk unit should have a storage capacity of nearly 2 gigabytes.

4.3. Mean Time to Failure of a Cluster

A cluster is active if all of the following hold true:

1. All of the *ND* disk units are active,
2. At least one of the *NP* processor units is active,
3. Shared memory unit is active,
4. Interconnect and connection to it is active, and
5. Power supply unit is active.

Thus, by using equation 3, $MTTF_{cluster}$ is given by:

$$\frac{1}{MTTF_{cluster}} = \frac{1}{MTTF_{disks}} + \frac{1}{MTTF_{procs}} + \frac{1}{MTTF_{mu}} + \frac{1}{MTTF_{int}} + \frac{1}{MTTF_{psu}} \quad (6)$$

where $MTTF_{disks}$ is the MTTF of the system of all the disk units in the cluster, $MTTF_{procs}$ is the MTTF of the processor units in the cluster, $MTTF_{mu}$ is the MTTF of the shared memory unit in the cluster, $MTTF_{int}$ is the MTTF of interconnect and connection to it, and $MTTF_{psu}$ is the MTTF of power supply unit in the cluster. Each of these items are discussed below.

- All of the disk units of a cluster must be active in an active cluster. This is because of the way we distribute the data, namely, the data allocated to a cluster is vertically fragmented and different fragments are stored on different disks. Hence all the fragments are required to construct a copy of a relation. $MTTF_{disks}$ can be given by simplifying equation 3 because it is a system consisting of k identical disk units working in parallel and each of the units should be available for all the system to be available. Thus, $MTTF_{disks} = MTTF_{du} / k$ where $MTTF_{du}$ is the MTTF of a disk unit. Each disk unit may be comprised of a single disk or mirrored disks.
- At least one of the processors units must be active in an active cluster. When a processor fails, transaction fault tolerance aborts the transactions being executed by the failed processor. As long as at least one of the processor units is active, the processing will continue in the cluster, but at reduced speed. Such a property of the system is called graceful degradation. A processor unit is either a single processor or a processor pair.
- The shared memory must be active in an active cluster. The shared memory could be either nonredundant or redundant. If it uses memory mirroring, $MTTF_{mem}$ can be calculated in a way similar to the mirrored disks.
- An interconnect and the cluster's communication to it must be active for an active cluster. The system may have single interconnect or dual interconnects (i.e., two interconnects with independent communication controllers at each cluster). $MTTF_{int}$ of a dual interconnect can be calculated as in the case of mirrored memory.
- The power supply must be active for the cluster to be active. A cluster may have a single or a dual power supply. $MTTF_{psu}$ can be calculated as in the case of dual interconnect and mirrored memory.

4.3.1. MTTF Results

There are many parameters that influence $MTTF_{cluster}$. Among the important ones are the size of the cluster (primarily identified by *ND*), fault tolerance techniques used for each of its components (i.e., configuration), the reliability (or mean time to failure) of each of its components, and the data storage scheme.

The larger the size of a cluster, the more parallelism and storage capacity within a cluster. However, there are more components in a cluster that can fail. Thus the $MTTF_{cluster}$ will decrease. Two important parameters that decide cluster size are *ND* and *NP*. In terms of fault tolerance and our study, *ND* plays a significantly more dominant role because of the following reasons:

- $MTTF_{disk}$ is smaller than $MTTF_{processor}$ because a disk has mechanical components and hence is inherently less reliable.
- Usually *ND* is larger than *NP* since the memory contention limits the number of processors that can be used in parallel.
- *ND* also decides the size of the database.

Because of the importance of *ND* and the sensitivity of all the three output parameters with respect to it, we plot it on the x-axis for all the graphs. The output parameters are plotted on the y-axis.

The fault tolerance techniques require redundancy and hence more components. The system configurations reflect the hardware fault tolerance techniques used in the system. Figure 4.1 compares the $MTTF_{cluster}$ of cluster using different configurations. Configuration 3, which uses hardware fault tolerance techniques for all of its components, performs significantly better.

Because the MTTF of a disk is significantly smaller than the MTTF of other components, $MTTF_{disk}$ has a very significant effect on $MTTF_{cluster}$. This effect is shown in Figure 4.2. Note that the scale on the y-axis is logarithmic (i.e., it doubles every unit).

Figure 4.3 compares $MTTF_{cluster}$ of a cluster that has one processor with a cluster that has more than one processor. The difference between the curves for $NP = 1$ and $NP \geq 2$ is significant for smaller values for *ND*. At larger values for *ND*, $MTTF_{disks}$ becomes a bottleneck for both cases; so the difference is not significant. Also, having more than two processors does not increase the $MTTF_{cluster}$ noticeably because the value of $1 / MTTF_{procs}$ does not contribute significantly in equation 6.

4.4. Response Time of a Cluster Recovery

A cluster recovery takes place after a cluster fails. Estimating $MTTR_{cluster}$, response time for cluster recovery is quite difficult. Before we do so, let us briefly discussed a procedure to perform a cluster recovery (more details are given in [Lu et al 87]).

1. **Detect Cluster Failure.** Some of the hard failures result in the cluster failure. There are many strategies possible to detect a cluster failure. One strategy is to use an *adjacency* feature where a cluster sends 'I am alive' message to its left adjacent cluster. If a cluster does not receive this message duing a predetermined period, it inquires to find out whether the right adjacent cluster is functioning properly. For simplicity, we assume a centralized reconfiguration algorithm in which the adjacent cluster that detects the cluster failure coordinates the reconfiguration.
2. **Prepare for reconfiguration.** Upon detection of a cluster failure, the system enters a *pause state*. During this step, all the committed transactions are completed. All the transactions, which are active but not yet committed, are aborted. This step requires accessing and updating appropriate transaction tables and lock tables. The aborted transactions are restarted when the system leaves the pause state. The system remains in the pause state until the reconfiguration is completed. While the system is in the pause state, newly arriving transactions are queued behind the transactions to be restarted.
3. **Find what is missing and locate the redundant copy.** The coordinator accesses its global data allocation directory to find out the data (fragments) that was (were) stored on the failed cluster and to locate

the respective redundant data (fragments). We call the cluster with redundant data source clusters.

4. **Decide the Destination Clusters.** These are the clusters where the second copy of the data has to be created from the copy of data on the source clusters.
5. **Send the Data to the Destination Clusters.** The coordinator instructs the source clusters to send the relevant data to destination clusters. A destination cluster decides how to store the data on its disks.
6. **Update the Global Data Directories.** The coordinator sends appropriate information to all the active clusters to update their respective global directories.
7. **Restart.** The coordinator signals *all O.K.* to all the clusters. The system then leaves the pause state and starts processing transactions in its queues.

After the repair of one or more failed components, a cluster may be ready to be reintegrated with the rest of the active system. The reintegration process requires that the repaired cluster is updated with respects to the data. Most of this process can be performed in the background. We now calculate an optimistic estimate for the mean time to repair.

The step of the recovery procedure that may contribute the most to the recovery time is step 5, "send the data to the destination cluster." This step can be divided into three substeps.

Step 5a. Read the data from the source clusters. The DF is the same as the number of source clusters; so the recovery data will be read from the disks of each of the source clusters. Because we assume that vertical fragments of the parts of relations are stored on a cluster, it is possible to retrieve data from all the disks in a cluster in parallel. The following equations give the time to read the recovery data.

total data lost due to cluster failure, $DBC = 2 \times DBS/NC$
total recovery data on a source cluster, $DBR = DBC/DF$
data on a disk in source cluster, $DBD = DBR/ND$
number of pages to be read from a disk, $PG = DBD/PS$
time to read PG pages or the total read time, $RT = PG \times I$

Step 5b. Transmit data from source clusters to destination cluster. We assume that the data received in a one page fetch cycle is packeted together and transmitted to a destination cluster. Transmission time includes the time required for physical channel transmission and communication software overhead. Optimistically, we assume a communication software overhead factor (CSF) of 2.0. The time to transmit recovery data can be calculated as follows.

packet size, $PKS = ND \times PS$
time to transmit a packet, $TP = (PKS/C) \times CSF$
total transmission time, $TT = TP \times PG$

Step 5c. Write data on the destination cluster. The time to perform this step can be calculated as in step 5a.

We assume that as far as possible, the system will read, transmit and write data in parallel. Thus, if I < TP (i.e., the time to read a page is less than the time to transmit a packet), the total time for step 5 is (I + TT + I), where the first I is due to the time to read a page and the second I is due to the time to write the data received in a packet in parallel on all the disks in a destination cluster. On the other hand, if I > TP, then the total time for step 5 is (TR + TP + I).

Table 4-6 summarizes the time taken by various steps of the recovery procedure. The time taken for other steps of the recovery procedure is our best guesstimate.

	Step of Recovery Procedure	Time Taken
1.	Detect failure	2 sec
2,3,4.	Prepare to send data	10 sec
5.	Send data	max(2I + TT, TR + TP + I)
6,7.	Update dictionary and restart	2 sec

Table 4-6: Factors Contributing to MTTR

4.4.1. MTTR Results

As in case of $MTTF_{cluster}$, $MTTR_{cluster}$ is influenced by many parameters. Among the important parameters are size of the database, distribution factor, interconnect capacity and page size.

The larger the size of the database, the more data will become unavailable when a cluster fails. Thus more data will need to be transferred during the recovery procedure especially if the system has fewer large clusters. Figure 4.4 shows that $MTTR_{cluster}$ is significantly larger for database size of 1 terabyte as compared to the database size of 100 gigabytes. Most of the $MTTR_{cluster}$ is contributed by the transmission time (TT).

A higher *DF* means more clusters have the recovery data and hence less recovery data per cluster and disk in a source cluster. Thus it is possible to have more parallelism for systems with a higher *DF*. *DF* =1 means that only one cluster has all the second copy of data required for recovery, $DF = NC / 2$ means on average half of the cluster in the system have part of the recovery data, and $DF = NC-1$ means all the active clusters have some part of the recovery data. Figure 4.5 shows that effect of the *DF* is very significant. Since the *DF* depends on the data storage scheme, the effect of data storage scheme on $MTTR_{cluster}$ is very significant.

We found that TP > I in most cases. Thus the transmission time is more dominant than read time, especially for large clusters (i.e., higher *ND*). Because of this, effect of interconnect capacity, *C*, on $MTTR_{cluster}$ is significant, particularly for clusters with $ND \geq 16$. See Figure 4.6.

The larger the page size, the more recovery data is read in a page fetch cycle. This results in smaller I/O read and write time and comparatively larger transmission time per packet. For smaller clusters, the read time is dominant and since large page size results in fewer page fetches, a system with larger *PS* has a smaller $MTTR_{cluster}$. See Figure 4.7. For $ND \geq 64$, transmission time dominates; so the PS does not affect $MTTR_{cluster}$.

4.5. System Availability

The mean time to a cluster failure in a system is $NC * MTTF_{cluster}$. The mean time to repair from such a failure is $MTTR_{cluster}$. Thus the system availability can be calculated as follows using equation 1.

$$Av = \frac{(NC \times MTTF_{cluster})}{(NC \times MTTF_{cluster} + MTTR_{cluster})} \quad (7)$$

4.5.1. Availability Results

To study the effect of various parameters on a system, we evaluate same-size systems. We do this by fixing the total number of disk units in a system to 1024 (i.e., $NC * ND = 1024$). Thus if the system has large clusters (i.e., *ND* is large), then the system has fewer such clusters. However, we will need fewer large clusters in a system for a given database size. Depending on the capacity of each of the disk, the total database size can be varied from 100 gigabytes to 1 terabyte. This size, in our opinion, defines a very large database system.

System availability is affected by all the parameters that affect $MTTF_{cluster}$ and $MTTR_{cluster}$. The parameters that have more significant effect are system configuration, $MTTF_{disk}$, *NP*, *DBS*, *DF*, *C* and *PS*.

Table 4-7 relates *Av* to system down time. The desired level of availability depends on the type of system application. We feel that modern systems will be expected to give availability of 0.999 or more. It is comforting to note that the fault tolerance techniques we already know may be able to give such availability even for very large systems. It may be noted that all the *Av* curves that follow are drawn to logarithmic scale. Thus towards the higher end of the scale, even small vertical separation of curves may mean very significant differences in *Av*.

Av	unavailability of the system
0.99	unavailable for one hour in 4 days
0.999	unavailable for one hour in 41 days (> a month)
0.9999	unavailable for one hour in 416 days (> a year)
0.99999	unavailable for one hour in 4,166 days (> 11 years)

Table 4-7. Availability vs Unavailability

Figure 4.8 compares the Av of different configurations This shows that the effect of system configuration (i.e., the hardware fault tolerance techniques) is very significant. For example, a 64-cluster system with 8 disk units each will be unavailable for one hour in approximately 79 days if it uses configuration 1, unavailable for 1 hour in approximately 767 days (2.1 years) if configuration 2 is used, and unavailable for 1 hour in 1250K hours (143 years) if configuration 3 is used.

Figure 4.9 compares Av for systems with different $MTTF_{disk}$. Since $MTTF_{disks}$ is always the bottleneck for configurations 1 and 3 and usually the bottleneck for configuration 2, the effect of this parameter is very significant. It should, however, be noted that because both configuration 2 and 3 use disk mirroring, the improved fault tolerance results in a 100% increase in disk costs.

Figure 4.10 shows the effect of NP on Av. This figure shows the curves for configuration 2, in which $MTTF_{procs}$ is the bottleneck for $NP = 1$. Thus the effect of processor redundancy is significant in this case. However in all other cases, because disk availability is the bottleneck for Av, the effect of processor redundancy is not significant if $NP \geq 2$. By comparing the curves for $NP = 2$ and $NP = ND / 2$, we also note that the effect of more than two processor on fault tolerance is insignificant. Thus, the primary purpose of having more than two processors in a cluster will be efficiency in query processing and not fault tolerance. Also note that processor redundancy costs less than disk redundancy.

Figure 4.11 shows the effect of DBS on Av. The difference between DBS of 100 gigabytes and 1 terabyte is less when the system consists of many small clusters because of the smaller difference between values for $MTTR_{cluster}$. This difference is very significant for the system consisting of fewer large cluster for both $MTTR_{cluster}$ (see Figure 4.4) and Av.

Figure 4.12 shows the effect of DF on Av. A higher DF means more clusters have the recovery data and take part in recover procedure, thus providing more parallelism. This effect is more significant for a system consisting of a large number of small clusters.

Figure 4.13 shows the effect of C on Av. The differences for a system of many small clusters is insignificant because the time to read recovery data is dominant (i.e., the system is node bound). However, the transmission time is dominant (i.e., the system is communication bound) for a system with few large clusters, so the effect of C is more significant.

Figure 4.14 shows the effect of PS on Av. The effect is more significant for a system with many small clusters. In general, Av is higher for higher PS. A higher PS means fewer pages need to be fetched and larger chunks of data need to be transmitted. This results in the system becoming communication bound sooner.

5. Conclusions and Future Work

In this paper, we identified basic fault tolerance techniques that are required to achieve a high degree of fault tolerance in a very large database system. We also used quantitative methods to evaluate availability. The quantitative evaluation of the effect of various fault tolerance techniques helped us to understand the relative importance of various parameters that affect system fault tolerance. Our study shows that system availability is very sensitive to the following parameters:

1. System Architecture. By varying the number of clusters, the number of disks per cluster, and the number of processors per cluster, we were able to study different system architectures. For example, one extreme of our parameterized architecture presents a loosely coupled, non-shared memory architecture (i.e., case of $ND = 1$, $NP = 1$). We found that better availability is obtained when the system has many small clusters. In most cases, $ND \leq 8$ when availability peaks.
2. Fault Tolerance Techniques. We studied the effect of various hardware fault tolerance techniques by varying system configurations that differ in the fault tolerance techniques used for different components. We found that the effect of fault tolerance techniques on system availability is very significant. We also found that reliability of a disk ($MTTF_{disk}$) is usually a bottleneck, and performance gain due to disk mirroring is very substantial. Processor redundancy significantly helps if disks are mirrored (or other disk redundancy methods are used). However, $NP = 2$ is sufficient for fault tolerance. A higher NP does not improve availability significantly. Fault tolerance techniques for other components are useful in conjunction with fault tolerance techniques for disks and processors.
3. Database Size. The size of the database has a significant effect on system availability. If the database is larger, more data will be lost when a hard failure occurs; so recovery takes longer. This degrades availability. Also, as the database size increases, the system will require more components of given capacity for storage and efficient processing. This can have a very significant affect on system availability.
4. Component Reliability and Capacity. Since disk reliability is usually the bottleneck, we studied availability for disks with two different MTTF values. Higher $MTTF_{disk}$ improves availability significantly. System availability will also improve significantly if disks with higher capacities are used (provided $MTTF_{disk}$ of a high capacity disk is not much lower than that for a low capacity disk). We also studied the effect of using interconnects with different capacities. A system with a higher capacity interconnect has a significantly better fault tolerance when the system consists of a few large clusters.
5. Data Storage and Access Method. We studied only a limited aspect of this issue. We used one basic data storage method in our study. By varying the distribution factor, we were able to study how a data storage method can affect recovery time and hence system availability. By showing the dependence of the quantitative evaluation on the data storage method and by using a distribution factor, the study shows that the data storage method greatly affects system availability. We studied the access method only with respect to page size. A larger page size helps significantly in a system with many small clusters.

We have used a simple model to evaluate the effect of a number of parameters. To study more detailed effect of some of the parameters, more detailed model should be used. Our study also helped us understand the complexity involved in making the system fault tolerant. We feel the following issues need to be addressed in the future.

1. Query processing and optimization techniques and the system fault tolerance are very intimately related. Query processing techniques and data storage schemes depend on the type and frequency of queries asked (i.e., the application). As we saw in our study, the data storage scheme affects fault tolerance significantly. Thus, in designing a real system, the designers of query processing software and fault tolerance have to understand the application and devise a storage scheme that is acceptable from the query processing as well as fault tolerance viewpoint. Required levels of response time, fault tolerance and reliability are application dependent.
2. Define and evaluate additional fault tolerance and performance parameters. The availability measure evaluated in this report is some time called strong availability. Using other availability definitions, such as percentage of transaction that can complete despite failures, may give further insight into system behavior. We also think that the availability measure gives only a part of the story. Other parts of system behavior are captured by response time measures. To evaluate the system more thoroughly, additional parameters that combine no-failure response time and availability could be evaluated. Examples of such measures are average response time with failures and average system throughput [Sheth et al 87]. Evaluating these parameters may be difficult but required in light of the previous issue.
3. Soft failures are tolerated using hardware methods such as processor pairing, and software methods such as transaction management and software reinitialization. Although conceptually these techniques are not difficult to understand, their implementation may be significantly difficult. These techniques and their effect on overall system performance should be studied in more detail.

REFERENCES

[Avizienis and Kelly 84] A. Avizienis and P. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *IEEE Computer*, Vol. 17, no. 8, August 1984.

[Bernstein 85] P.A. Bernstein, "Sequoia: A Fault-Tolerant Tightly-Coupled Computer For Transaction Processing," Technical Report TR-85-03, Wang Institute of Graduate Studies, School of Information Technology, May 2, 1985.

[Borr 81] A. Borr, "Transaction Monitoring in ENCOMPASS (TM): Reliable distributed transaction processing," *Proc. 9th VLDB*, September 1981.

[Borr 84] A. Borr, "Robustness to Crash in a Distributed Database: A Non Shared Memory Approach," *Proc. 9th VLDB*, September 1981.

[Decker 86] J. Decker, "C^3I Teradata Study," *Tech. Rep. RADC-TR-85-273*, Rome Air Development Center, N.Y., March 1986.

[Gray 86] J. Gray, "Why Do Computers Stop and What Can Be Done About It?," *Proc. 5th Symposium on Reliability in Distributed Software and Database Systems*, 1986, pp. 3-11.

[Gray 86b] J. Gray, Personal communication, November 20, 1986.

[Haeder and Reuter 83] T. Haeder and A. Reuter, "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, Vol. 16, No. 4, December 1983.

[Kastner 83] P. S. Kastner, "A Fault-Tolerant Transaction Processing Environment," *Database Engineering Bulletin*, Vol. 6, No. 2, June 1983.

[Kim 84] W. Kim, "Highly Available Systems for Database Applications," *ACM Computing Surveys*, Vol. 16, No. 1, March 1984, pp. 71-98.

[Koon and Ozsu 86] T. Koon and M. Ozsu, "Performance Comparison of Resilient Concurrency Control Algorithms for Distributed Databases," *Proc. 2nd Int. Conf. on Data Engineering*, Los Angeles, February 1986.

[Lu et al 87] H. Lu, K. Mikkilineni, R. Ramnarayan, A. Sheth, Very Large Parallel Data Flow, Interim Technical Report, Honeywell Corporate Systems Development Division, February 1987.

[Schlichting and Schneider 83] R. Schlichting and F. Schneider, "Fail-Stop Processors, an Approach to Designing Fault-Tolerant Computing Systems," *ACM TOCS*, Vol. 1, No. 3, August 1983.

[Sheth et al 85] A. Sheth, A. Singhal and M. Liu, "An Analysis of the Effect of Network Parameters on the Performance of Distributed Database Systems," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 10, October 1985.

[Sheth et al 87] A. Sheth, A. Singhal and M. Liu, "Performance Analysis of Resiliency Mechanisms in Distributed Database Systems," *Proc. 3rd Intl. Conf. on Data Engineering*, February 1987.

[Siewiorek and Swarz 82] D. Siewiorek and R. Swarz, "The Theory and Practice of Reliable System Design," *Digital Press*, Bedford, MA, 1982.

[Siewiorek et al 78] D. Siewiorek et al, "A Case Study of C.mmp, Cm*, and C.vmp: Part 1- Experiences with Fault Tolerance in Multiprocessor Systems," *Proc. IEEE*, Vol. 66, No. 10, October 1978.

[Siewiorek 84] D. Siewiorek, "Architecture of Fault-Tolerant Computers," *Computer*, Vol. 17, No. 8, August 1984.

[Smith 86] M. Smith, MCC, personal communication, October 1986.

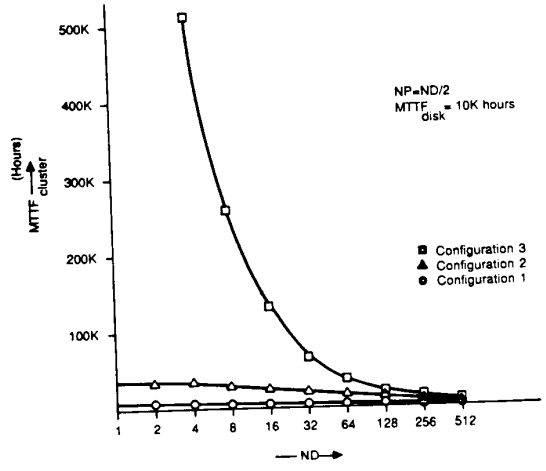

Figure 4.1. Effect of Configuration on $MTTF_{cluster}$

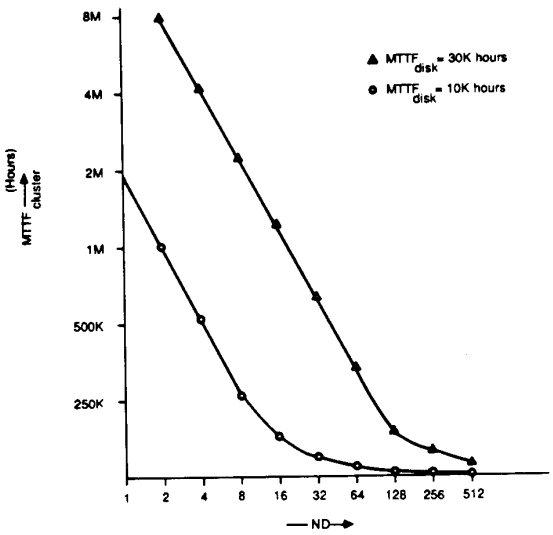

Figure 4.2. Effect of $MTTF_{disk}$ on $MTTF_{cluster}$

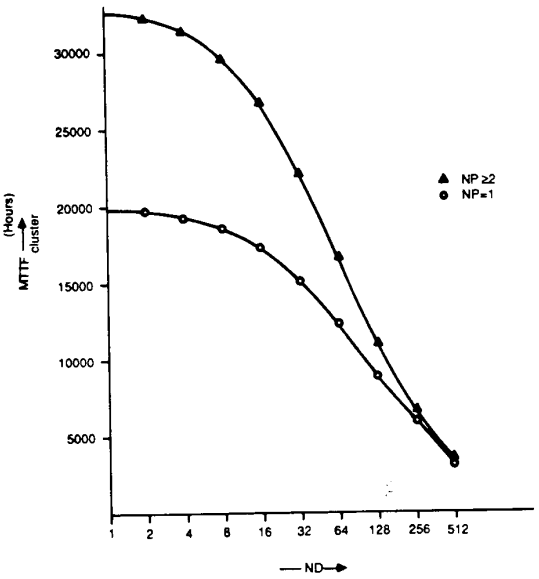

Figure 4.3. Effect of NP on $MTTF_{cluster}$

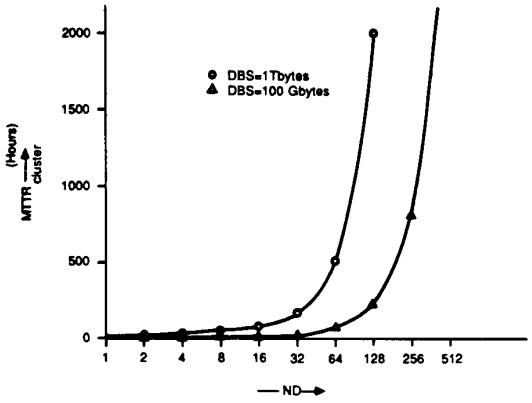

Figure 4.4. Effect of *DBS* on $MTTR_{cluster}$

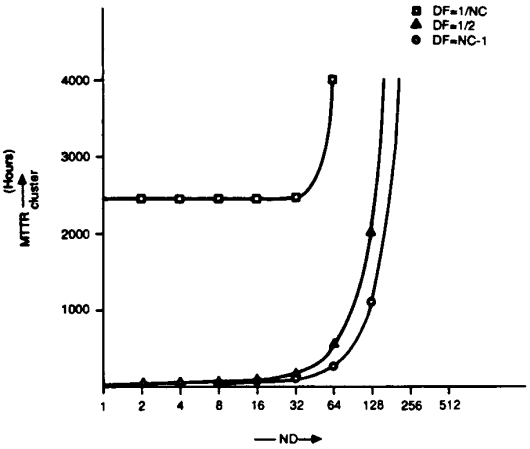

Figure 4.5. Effect of *DF* on $MTTR_{cluster}$

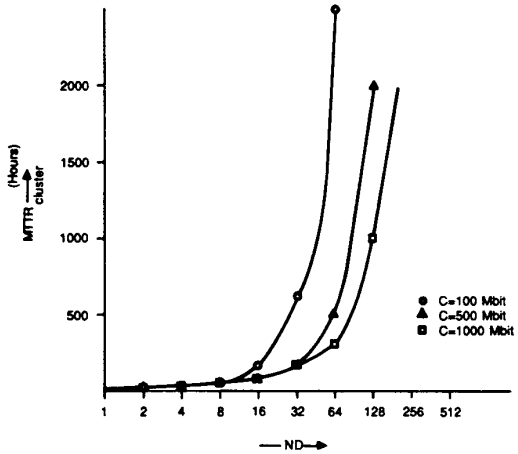

Figure 4.6. Effect of *C* on $MTTR_{cluster}$

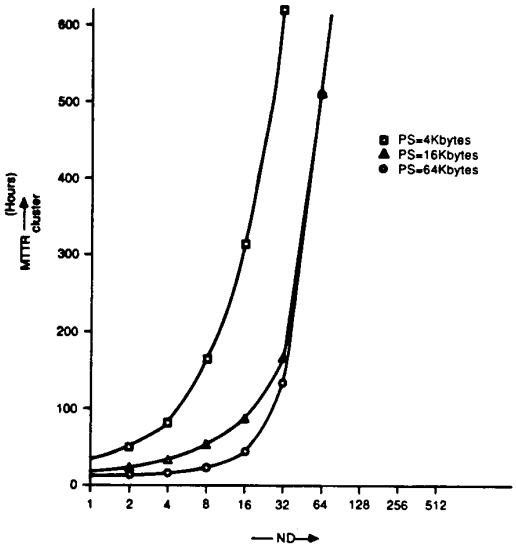

Figure 4.7. Effect of *PS* on $MTTR_{cluster}$

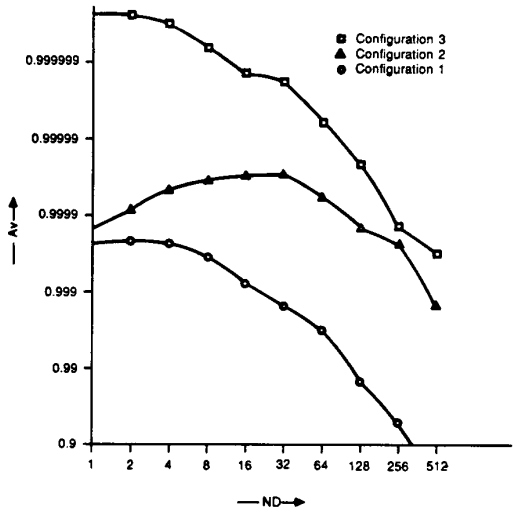

Figure 4.8. Effect of Configuration on *Av*

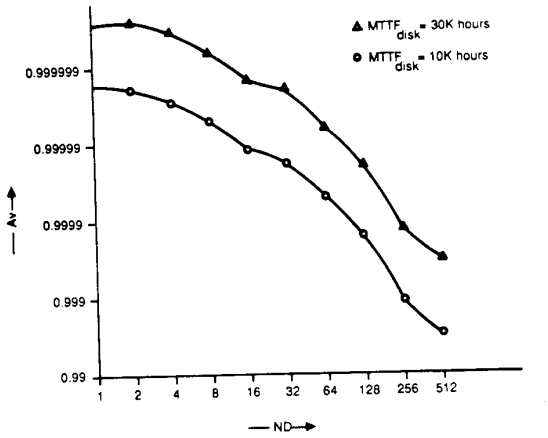

Figure 4.9. Effect of $MTTF_{disk}$ on Av

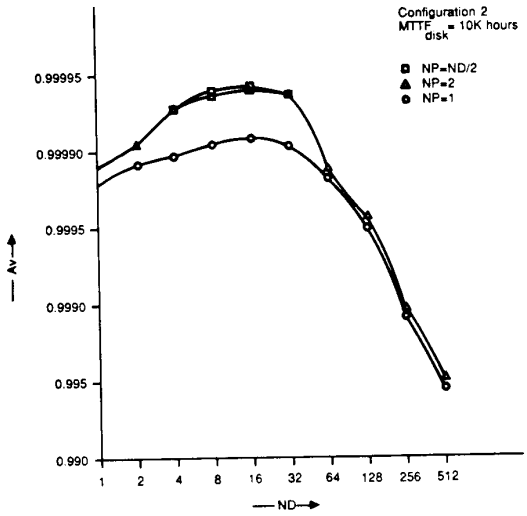

Figure 4.10. Effect of *NP* on *Av*

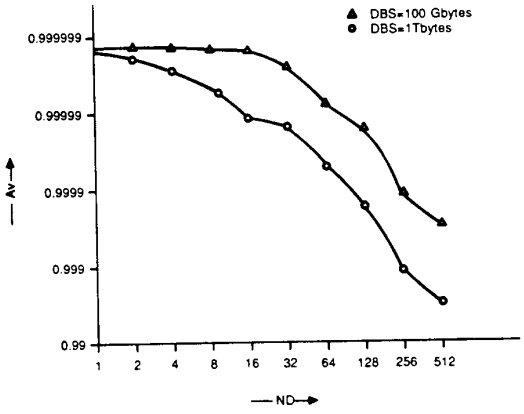

Figure 4.11. Effect of *DBS* on *Av*

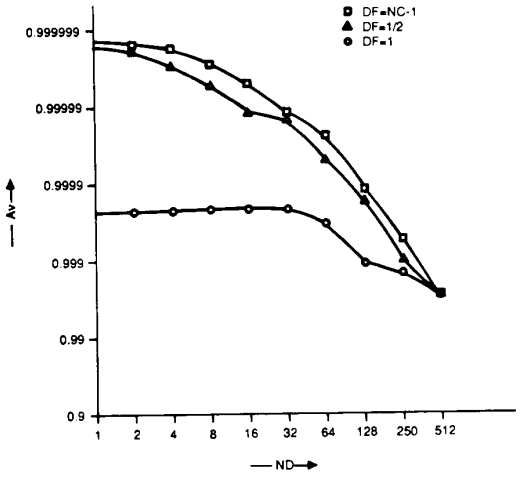

Figure 4.12. Effect of *DF* on *Av*

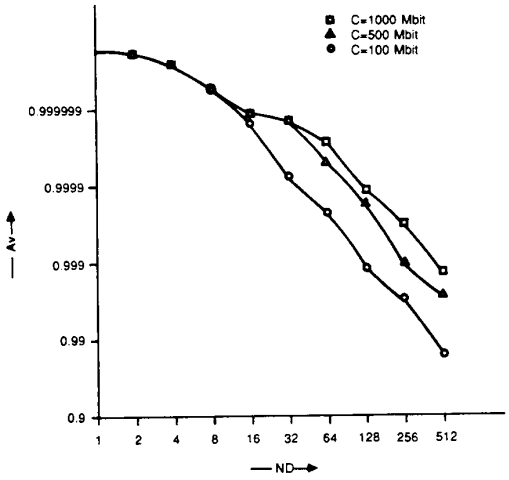

Figure 4.13. Effect of *C* on *Av*

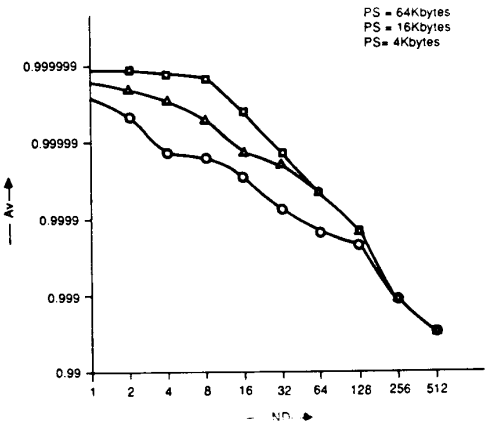

Figure 4.14. Effect of *PS* on *Av*