A Performance Study of Three High Availability Data Replication Strategies

Hui-I Hsiao†

IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

David J. DeWitt

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

This research was partially supported by the Defense Advanced Research Projects Agency under contract N00039-86-C-0578, by the National Science Foundation under grants DCR-8512862, by a Digital Equipment Corporation External Research Grant, and by a research grant from the Tandem Computer Corporation.

† This work was done while the author was in the University of Wisconsin.

Abstract

Several data replication strategies have been proposed to provide high data availability for database system applications. However, the tradeoffs among the different strategies for various workloads and different operating modes is still not well understood. In this paper, we study the relative performance of three high availability data replication strategies, chained declustering, mirrored disks, and interleaved declustering, in a shared nothing database machine environment. Among the issues that we have examined are (1) the relative performance of different strategies when no failures have occurred, (2) the effect of a single node failure on system throughput and response time, (3) the performance impact of varying the CPU speed and/or disk page size on the different replication strategies, and (4) the tradeoff between the benefit of intra query parallelism and the overhead of activating and scheduling extra operator processes.

Experimental results obtained from a simulation study indicates that, in the normal mode of operation, chained declustering and interleaved declustering perform comparably. Both perform better than mirrored disks if an application is I/O bound (due to disk scheduling), but slightly worse than mirrored disks if the application is CPU bound. In the event of a disk failure, because chained declustering is able to balance the workload while the other two cannot, it provides noticeably better performance than interleaved declustering and much better performance than mirrored disks.

1. Introduction and Motivation

While a number of solutions have been proposed for increasing the availability and reliability of computer systems, the most commonly used technique involves the replication of processors and mass storage [Borr81]. For database applications, the availability of disk-resident data files is perhaps the major concern. Most database management systems employ a combination of a disk-based log together with periodic checkpointing of memory-resident data to insure the integrity and availability of the database in the event of disk or system failures. These techniques cannot, however, satisfy the availability requirements of certain database applications[1] because the recovery time in the event of a media failure can be intolerably long and the fact, that during the recovery period, data is unavailable.

To achieve a very high degree of data availability, two basic techniques are currently being used. In the first, multiple copies (usually two) of the same data item are stored on disks attached to separate processors. When one copy fails, the other copy can continue to be used and, unless both copies fail simultaneously, the failure will be transparent to users of the system and no interruption of service will occur. Examples of this mechanism include mirrored disks [Katz78, Bitt88], interleaved declustering [Tera85], the inverted file strategy [Cope88], and chained declustering [Hsia90a].

In the second approach, the data, along with the redundant error detection/correction information (usually parity bytes), is spread across an array of disk drives. When errors are discovered, the redundant information can be used to restore the data and application programs can continue using the data with minimal interruption. Strategies based on this approach include synchronized disk interleaving [Kim86], redundant array of inexpensive disks (RAID) [Patt88], and parity striping of disk arrays [Gray90].

Both approaches have been used in commercial systems. For example, Tandem’s NonStop SQL database machine and Teradata’s DBC 1012 database machine employ identical copies, while IBM’s AS400 system uses a disk array. Which of the above two approaches is a better choice for database applications? The tradeoffs between these two approaches can be captured by three factors: performance, availability, and cost.

While, traditionally, the performance of a computer system is measured both in terms of response time and throughput, in a multiprocessor system that provides resiliency from hardware and software failures, performance can be measured in two different operating modes: the **normal mode**, with no failed components, and the **failure mode**, in which one or more processors or disks have failed. When operating without any failures, [Gray90]

[1] Some examples of such applications are stock market trading, air defense systems, air traffic control systems, airline reservation-type systems, banking (OLTP), etc.

demonstrates that mirrored disks (an identical copy based strategy) provides better performance than RAID for OLTP applications and [Chen90] demonstrates that for small requests (less than one track of data), a mirrored disk mechanism provides higher disk throughput (MBbyte/sec./disk) than RAID. Since I/O requests in most database applications almost always transfer less than one track of data, these results seem to indicate that identical copy mechanisms will generally provide superior performance for database applications.

In the failure mode of operation, the same conclusion holds because the remaining copy can continue to be used without any interruption in service when one copy fails. There will be little or no service degradation to users and/or application programs. On the other hand, with a disk array, when a query needs to access data on the failed disk/copy, the data must be reconstructed on the fly. This process requires accessing[2] all the remaining disks in the array in order to satisfy a single disk request. In such cases, the failed disk array will be restricted to serve only one request at a time and its performance will degrade significantly.

The availability of a system or data file is greatly influenced by the way data files are placed on disks [Hsia90b] and the time needed to recover or restore a failed disk/node. The longer the recovery time is the higher the probability that a second failure will render data unavailable. With an identical copy approach, data can be copied from the intact disk(s) to the new disk after the broken disk has been repaired or replaced. This process can be done easily and quickly. Consequently, the vulnerable window where a second failure may result in a loss of data availability is shortened. With the disk array approach, on the other hand, all disk pages in the failed array must be read and processed in order to rebuild the data originally stored on the failed drive. This process will take longer than simply copying from one disk to the other. Consequently, when failures occur, systems employing a disk array will remain in the failure mode longer and the possibility of second failure occurring before the first failure is fixed will be higher. As a result, the probability of data being unavailable is higher with the disk array approach. On the other hand, the disk array approach is more attractive if the cost of disk space is a major concern.

Throughout this paper, we focus on multiprocessor database systems that employ a "shared-nothing" architecture [Ston86]. For such systems, the application of horizontal partitioning (i.e. declustering) techniques [Ries78, Tera85, DeWi86, Livn87] facilitates the successful application of inter- and intra-query parallelism in the normal mode of operation [Tand87, DeWi88]. However, when a failure occurs, balancing the workload among the remaining processors and disks can become difficult, as one or more nodes[3] (processor/disk pairs) must assume the workload of the component that has failed. In particular, unless the data placement scheme used allows the workload of

[2] In addition, pages read from all disks would have to be "xor'd" together to reconstruct the failed data.

[3] We assume that, in the absence of special purpose hardware (i.e. dual ported disks and disk controllers), the failure of a processor controlling one or more disks renders the data on these disks unavailable.

the failed node to be distributed among the remaining operational nodes, the system will become unbalanced and the response time for a query may degrade significantly even though only one, out of perhaps a hundred nodes, has failed. In addition, the overall throughput of the system may be drastically reduced since a bottleneck may form.

In this paper, using a simulation model, we study the performance of three identical copy based high availability schemes: chained declustering, mirrored disks, and interleaved declustering. The simulation model is based on the software and hardware architectures of the Gamma database machine [DeWi90]. With this simulation model, the performance of the three data replication strategies is evaluated under a number of different workload assumptions. Among the issues that have been examined are (1) the relative performance of the three mechanisms when no failures have occurred, (2) the effect of a single node failure on system throughput and response time, and (3) the performance impact of varying the CPU speed and/or disk page size on the different replication strategies, and (4) the tradeoff between the benefit of intra-query parallelism and the overhead of activating and scheduling extra operator processes.

The organization of the rest of the paper is as follows. In the next section, the three high availability strategies are presented. Our simulation model is described in Section 3. The results of our simulation experiments are presented and analyzed in Section 4. Our conclusions and future research directions are contained in Section 5.

2. Existing High Availability Strategies

In this section, we briefly describe the three data replication schemes studied in this paper: mirrored disks [Borr81, Bitt88], interleaved declustering [Tera85, Cope89], and chained declustering [Hsia90a]. Each scheme stores two identical copies of each relation on different disks and each is able to sustain a single node (disk or processor) failure.

2.1. Tandem's Mirrored Disks Architecture

In Tandem's NonStop SQL system [Tand87], each disk drive is connected to two I/O controllers, and each I/O controller is connected to two processors, thus providing two completely independent paths to each disk drive. Furthermore, each disk drive is "mirrored" (duplicated) to further ensure data availability. Relations are generally declustered across multiple disk drives. For example, Figure 1 shows relation R partitioned across four disks. R_i represents the i-th horizontal fragment of the first copy of R and r_i stands for the mirror image of R_i. As shown in Figure 1, the contents of disks 1 and 2 (and 3 and 4) are identical. Read operations can be directed (by the I/O controller) to either drive but write operations must be directed to both drives in order to keep the contents of both disks identical, causing the two disk arms to become synchronized on writes [Bitt89].

sc 0.30
medium 8

1 8
2 9
3 9
4 10
file GRN/fig.tan.d1

Figure 1: Data Placement with Tandem's Mirrored Disk Scheme.

When a disk in a mirrored pair fails, the remaining disk can assume the workload of the failed drive and, unless both disks fail simultaneously, data will always be available. The actual impact of a failure on the performance of the system depends on the fraction of read and write operations. If most I/Os are reads, losing a drive may result in doubling the average I/O time because only one disk arm is available. On the other hand, if most I/Os are write operations, the impact of a failure may be minimal [Bitt89].

The failure of a processor will, however, almost always have a significant negative impact on performance. Consider the failure of processor P1 in Figure 1. While the data on disks 1 and 2 will remain available, processor P2 will have to handle **all** accesses to disks 1 and 2 as well as disks 3 and 4 until P1 is repaired. If P2 is already fully utilized when the failure occurs, the response time for queries that access data on either pair of drives may double if the system is CPU bound.

2.2. Teradata's Interleaved Declustering Scheme

In the Teradata database machine [Tera85], the processors are divided into clusters of 2 to 16 processors (one or two disk drives may be attached to each processor). Tuples in a relation are declustered among the drives in one or more clusters by hashing on a "key" attribute. The tuples of a relation stored on a disk are termed a **fragment**. Optionally, each relation can be replicated. In this case, one copy is designated as the **primary** copy and the other the **backup** copy.

The tuples in each primary fragment are stored on one node. For backup fragments, Teradata employs a special data placement scheme termed **interleaved declustering** [Tera85, Cope89]. If the cluster size is N, each backup fragment will be subdivided into N-1 subfragments each of which will be stored on a different disk within the same cluster — but not the disk containing the primary fragment. In Figure 2, a relation R is declustered across 8 disk drives and N=4. (R_i represents the i-th primary fragment whereas $r_{i.j}$ represents the j-th subfragment of the i backup fragment of R_i.)

When a node failure occurs, interleaved declustering is able to do a better job of balancing the load than the mirrored disk scheme since the workload of the failed node will be distributed among N-1 nodes. However, this improvement in load balancing is not without a penalty. In particular, the probability of data being unavailable increases proportionately with the size of the cluster [Hsia90a].[4] During the normal mode of operation, read requests are directed to the fragments of the primary copy and write operations update both copies. In the event of a CPU or disk failure that renders a fragment of the primary copy unavailable, the corresponding fragment of the backup copy will be promoted to become the primary (active) fragment and all data accesses will be directed to it.

center, expand; c | c s s s | c s s s | c | c c c c | c c c c |. cluster 0 cluster 1

Node	0	1	2	3	4	5	6	7	_	**Primary**
Copy	R0	R1	R2	R3	R4	R5	R6	R7	_	**Backup Copy**
		r0.0	r0.1	r0.2		r4.0	r4.1	r4.2		
	r1.2		r1.0	r1.1	r5.2		r5.0	r5.1		
	r2.1	r2.2		r2.0	r6.1	r6.2		r6.0		
	r3.0	r3.1	r3.2		r7.0	r7.1	r7.2	_		

Figure 2: Interleaved Declustering (Cluster Size N = 4)

2.3. Chained Declustering

With chained declustering [Hsia90a], two physical copies (a **primary** and a **backup**) of each relation are declustered over a set of disks such that the primary and backup copies of a fragment are always placed on different nodes. Nodes are divided into disjoint groups called **relation-clusters** and tuples of each relation are declustered among the drives that form one of the relation-clusters. Optionally, the disks in each **relation-cluster** can themselves be sub-divided into smaller groups termed **chain-clusters**. A small system may consist of only one **relation-cluster**, while a large system may contain several. For purposes of simplicity, in this paper we assume that a relation-cluster contains all of the disks in the system and that it is not subdivided into multiple chain-clusters.

The data placement algorithm for chained declustering operates as follows. Assume that there are a total of M disks numbered from 1 to M. For every relation R, the i-th primary fragment is stored on the {[i+C(R)] mod M}-th disk, and the i-th backup fragment is stored on the {[i+1+C(R)] mod M}-th[5] disk. The function C(R) allows the first fragment of relation R to be placed on any disk within a **relation-cluster** while the "1" in the second formula is used to ensure that the the primary and backup copies of a fragment are placed on different disks. As an example, consider Figure 3 where M, the number of disks in the relation-cluster, is equal to 8 and C(R) is 0. The tuples in the primary copy of relation R are declustered using one of Gamma’s three horizontal partitioning strategies with tuples in the i-th primary fragment (designated R_i) stored on the i-th disk drive. The backup copy is declustered using the same partitioning strategy but the i-th backup fragment (designated r_i) is stored on (i+1)-th disk (except r7 which is stored on 0-th disk). In the figure, R_i and r_i contain identical data. We term this technique **chained declustering** because the disks are "linked" together, by the fragments of a relation, like a chain.

[4] Data will be unavailable if any two nodes in a cluster fail.

[5] A generalized formula is {[i+k+C(R)] mod M} where 0 < k < M and the greatest common divisor of M and k, GCD(M, k), is equal to 1.

center, expand; c | c c c c c c c c |.

Node	0	1	2	3	4	5	6	7 _
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
_								
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

Figure 3: Chained Declustering (Relation Cluster Size = 8)

With the above data placement strategy, a relation will be unavailable only if two (logically) adjacent disks within a relation-cluster fail simultaneously, or if the second disk fails while the first one is being repaired. For example, suppose that node 1 has failed. If node 0 or node 2 also fails before node 1 is repaired, then some data will be unavailable. Any subsequent node failure other than node 0 or node 1 (i.e, nodes 3 to 7) will not compromise the availability of data.

During normal operation, reads are directed to the fragments of the primary copy and writes update both copies. In the case of a single node (processor or disk) failure chained declustering is able to distribute the workload of the cluster uniformly among the remaining operational nodes. As illustrated by Figure 4, with a cluster size of 8, when a processor or disk fails, the load (read portion of the workload) on each remaining node will increase by 1/7th by using both the primary and backup fragments for read operations. For example, when node 1 fails, the primary fragment R1 can no longer be accessed and thus its backup fragment r1 on node 2 must be used for processing queries that would normally have been directed to R1. However, instead of requiring node 2 to process all accesses to both R2 and r1, chained declustering offloads 6/7ths of the accesses to R2 by redirecting them to r2 at node 3. In turn, 5/7ths of access to R3 at node 3 are sent to r3 instead. This dynamic reassignment of the workload results in an increase of 1/7th in the workload of each remaining node in the cluster. Since the relation-cluster size can be increased without compromising data availability, it is possible to make this load increase as small as desired.

_

center, expand; c | c c c c c c c c |.

Node	0	1	2	3	4	5	6	7 _
Primary Copy	R0	---	$\frac{1}{7}$R2	$\frac{2}{7}$R3	$\frac{3}{7}$R4	$\frac{4}{7}$R5	$\frac{5}{7}$R6	$\frac{6}{7}$R7 _
Backup Copy	$\frac{1}{7}$r7	---	r1	$\frac{6}{7}$r2	$\frac{5}{7}$r3	$\frac{4}{7}$r4	$\frac{3}{7}$r5	$\frac{2}{7}$r6 _

Figure 4: Fragment Utilization with Chained Declustering
After the Failure of Node 1 (Relation-Cluster Size = 8)

An attractive feature of this is that the reassignment of active fragments incurs neither disk I/O nor data movement. Only some bound values and pointers/indices in a memory resident control table must be changed, and these modifications can be done very quickly and efficiently.

The example shown in Figure 4 provides a simplified view of how the chained declustering mechanism actually balances the workload in the event of a node failure. In actual database applications, however, queries cannot simply access an arbitrary fraction of a data fragment because data may be clustered on certain attribute values, indices may exist, and the query optimizer may generate different access plans. For example, in addition to three declustering alternatives (range, hash, and round-robin), Gamma also provides clustered and non-clustered indices on both the partitioning and non-partitioning attributes[6]. [Hsia90a] describe the design of a load balancing algorithm for the chained declustering mechanism that can handle all possible combinations of partitioning methods, storage organizations, and access plans. The keys to the solution are the notion of a **responsible range** for indexed attributes, the use of **query modification techniques** [Ston75], and the availability of an **extent map** for relations stored as a heap.

3. Simulation Model

3.1. Model Overview

To evaluate the three availability mechanisms, we constructed a simulation model of the Gamma [DeWi90] running on a 32 node Intel iPSC/2 hypercube [Inte88]. Figure 5 depicts the overall structure of the model. Each component is implemented as a DeNet [Livn89] discrete event module. The arcs in the figure are discrete event connectors and can be thought of as a combination of a preconstructed message path and a set of predefined message types. The role of each component is described briefly below. (The actual model parameters that we used can be found in Table 4.1).

Database Manager

The database is modeled as a set of relations consisting of a number of data pages. Both clustered and non-clustered indices can be constructed. The system catalog is used to keep track of the relations, indices, and, for chained declustering and interleaved declustering, the location of the primary and backup fragments of each relation.

```
sc 0.35
medium 8
1 6
2 6
3 9
4 10
file GRN/model.xg
```

Figure 5: Architecture of the Simulation Model.

[6] All fragments of a relation must have the same "local" storage organization.

Terminal

This module is responsible for generating queries. A query may select or update any number of tuples and it can be executed using either a sequential file scan or a clustered or a nonclustered index. The model simulates a closed system, so there can be only one outstanding request per Terminal. The number of active Terminals in the system determines the multi-programming level. When a query is completed, a Terminal waits for exactly *ThinkTime* seconds before submitting another query. The simulation runs until the preselected response time confidence interval, *ConfidInt*, is reached.

Query Manager

Given a query request, this module examines the schema to determine which node(s) should execute the query and then constructs an appropriate query plan[7]. If a single node is to be used to execute the query, it will be sent directly to the node. Otherwise, it is be sent to the Scheduler module.

Scheduler

This module is responsible for coordinating the execution of multiple-node queries. For each query, the Scheduler traverses the tree top down activating an operator process on each of the nodes containing relevant fragments. When a failure has occurred, the vertices of a query may be modified first. After initiating the query, the Scheduler waits for an "done" message from each operator process before committing the query and sending a "query done" message to the requesting Terminal.

Network Manager

The network manager encapsulates the operation of the communication network. A key parameter of this module is *PacketThreshold*, which determines how many network packets can be served simultaneously. Network packets are served in first-come, first-served (FCFS) order by the Network Manager. When a packet arrives at the Network Manager, it is served immediately if there are less than PacketThreshold packets outstanding. Otherwise, the new packet will be placed in a queue; as soon as a packet leaves the Network module, the head of the waiting queue will be removed and service for it will begin. While being served, each packet is delayed for a time T in the network module before it is delivered to the destination node. T is proportional to the number of bytes in the packet, which includes a packet header. The size of a network packet ranges from several hundred bytes for a control packet to several thousand bytes for a data packet.

Network Interface

This module models the sending and receiving of network packets (messages) for an operator node. A certain amount of CPU cycles is consumed for each message sent and received, The actual number of cycles consumed is determined by the message type (e.g. data or control packet) and its size.

Operator Manager

The Operator Manager simulates Gamma’s operator processes. This module models two different types of operator processes: selection processes and store processes. Depending on the type of the incoming query packet, the operator process may begin requesting data pages from the Disk Manager (if it is a select) or it may wait for a data packet to arrive from another processor via the network module (if it is a store). Each operator process requests certain amount of CPU time when it initiates an I/O request and when it processes disk or network data pages.

CPU

The CPU module models the sharing of the CPU resource among different processes running on a node. When a process needs CPU cycles, it sends a request to the CPU module with the number of CPU instructions needed. If the CPU is free, the request is served immediately and a reply is sent back to the requester after the requested CPU time has elapsed. Otherwise, the request will be put in a CPU ready queue. A key parameter

[7] The actual plan generated may differ depending on the mode of operation (normal or failure).

of this module is the CPU speed in MIPS.

Disk Manager and Disk

The Disk Manager is responsible for handling I/O requests generated by the Operator Manager. When a disk request is received, the Disk Manager maps the logical page number generated by the Operator Manager to a physical disk address (cylinder #, sector #), issues a disk I/O request, and then waits for the completion of the disk request. An elevator disk scheduling discipline is used except in the case of mirrored disks. In this case, a FIFO (with the shortest seek time) scheduling discipline is used [Bitt88, Gray88]. The total time required to complete a disk access is

$$DiskAccessTime = SeekTime + RotationalLatency + SettleTime + TransferTime$$

The Seek Time for seeking across n tracks is modeled by the formula [Bitt88]:

$$\text{Seek Time}(n) = SeekFactor * \sqrt{n}$$

The Rotational Latency is modeled by a random function which returns uniformly distributed values in the range of *MinLatency* to *MaxLatency*. *SettleTime* models the disk head settle time after a disk arm movement. The value of Transfer Time is computed by dividing the disk page size by the disk *Transfer Rate*.

Failure Manager and Log Manager

The Failure Manager has no impact during the normal mode of operation. In the failure mode, this module will randomly select a node to fail. The Log manager is not actually implemented. However, because each scheme has approximately the same overhead for generating and storing log records, we believe that the exclusion of the Log Manager will not significantly affect their relative performance.

3.2. Physical Data Placement in the Simulation Model

Figure 6 illustrates the layout of the primary and backup fragments on four disks for the mirrored disks, chained declustering, and interleaved declustering schemes. As illustrated in Figure 6, relation R is partitioned across four disks (or two mirrored pairs). R_i is the i-th primary fragment of relation R, while r_i is the corresponding backup fragment. With the mirrored disk strategy, the contents of the two disks within a mirrored pair are identical (R_i+R_j represents the union of fragments R_i and R_j). With chained declustering, primary fragments with their associated indices from all relations are placed together on the outer half of the cylinders while backup fragments are stored on the inner half. With interleaved declustering, primary fragments and their associated indices from all relations are placed together on the outer half of a disk drive (as with chained declustering) while all backup subfragments are placed together on the inner half.

sc 0.43
1 7
2 8
3 9
4 12
file GRN/datapla3.xg

Figure 6: Data Placement

With the mirrored disk scheme, a disk read request can be served by either disk in the pair. In the Tandem Non-Stop SQL system, and our model of this architecture, the disk with the shortest seek time is assigned to serve a disk read request. As a result, the expected seek distance for random reads is one-sixth of the tracks [Bitt88, Gray88]. With both chained and interleaved declustering, a primary fragment access scheme is used in our simulation experiments and because the primary fragments are placed together on the outer half of a disk drive, the expected seek distance for random read requests is also reduced from one-third to one-sixth of the tracks. As a result, all three data replication schemes provide improved performance over the non-replicated case for read only queries.

3.3. Alternative Update Mechanisms for Backup Fragments

With chained and interleaved declustering, an update query can be processed in one of three ways. First, each update can be sent to and processed by the two nodes on which the relevant primary and backup fragments are stored. A second approach is to send the update query to only the node containing the primary fragment for processing. After processing is completed, the node sends redo log records to the backup node where they are applied. A variation of this approach is to again direct update queries only to the nodes containing the primary fragments. However, instead of shipping redo records, the updated disk pages are shipped to the nodes containing the corresponding backup fragments where they are written directly to disk. This method incurs additional communications costs but does fewer total disk I/Os than either of first two methods. Because the network message delay is 5.6 ms for an 8K data page (measured on the Intel Hypercube), while the average disk service time for read requests is more than 12 ms, we selected this third method for processing update queries with chained and interleaved declustering.

4. Experiment and Results

This section presents the results of our comparison of the three availability mechanisms under a variety of different workloads, in both the normal and failure modes of operation. Of particular interest was how the load imbalance caused by a disk or processor failure affects system throughput and response time for various types of queries. Besides comparing the performance of the different high availability strategies, two other related issues are also explored: the impact of updating the backup copies and the tradeoff between the benefit of intra query parallelism and the overhead of activating and scheduling extra operator processes.

4.1. Model Validation

In order to evaluate the accuracy of the results produced by the simulation model, we first configured the model to reflect, as accurately as possible, the characteristics of Gamma, and then ran a number of experiments without replication. As described in [Hsia90b], the model predicted the actually measured performance of Gamma with less than a 10% margin of error.

4.2. Experimental Design

Typically, system throughput and average response time are the two key metrics used to evaluate a system. However, since our model simulates a closed system, response time is inversely proportional to system throughput. Thus, in the remainder of this section, throughput will be used as the main performance metric. Several additional metrics will be used to aid in the analysis of the results obtained. The first is the *disk service time*, which is the average time to serve a disk I/O request (not including the time spent waiting in the disk queue). The second metric is the *disk utilization*, which is computed by dividing the total disk service time of a disk by the experiment time. The third metric is the *CPU utilization*, which is measured by dividing the total CPU busy time by the experiment time. Finally, the average number of index and data pages accessed per query is also examined in our experiments.

Table 4.1 specifies the parameter settings used for the experiments. Since the mirrored disk scheme requires at least two disks (a mirrored pair) on each processor, each of the 16 processors has two disks attached. The database consists of eight relations, each with 2 million tuples and relations are fully declustered. The number of terminals (sources) in the model is varied from 1 to 72 and the buffer hit ratio for disk read requests is assumed to be 20%. The cluster size (number of disks) for the interleaved declustering scheme was set to 8 as this is the maximum size recommended by Teradata to its customers.

box,center; lI | l.

Parameter	**Setting**
Number of Processors	16
Disks per Processor	2
Number of Relations	8
Relation Size	2M tuples
Tuple Size	208 bytes
Multi-programming Level	1 - 72
Buffer Hit Ratio	20%
CPU Speed	3 MIPS
ID Cluster Size	8
Seek Factor	0.78
MinLatency	0 msec
MaxLatency	16.667 msec
Settle Time	2.0 msec
Transfer Rate	2M bytes/sec
Disk Page Size	8K bytes
PachetThreshold	999
Think Time	0 sec
ConfidInt	within 5% (95% confidence)

Table 4.1: Parameter Settings for Performance Experiments.

For the four experiments, the relations in the database were declustered over the disks in the system by hashing on the attribute used in the selection predicate of the queries. After the tuples had been declustered, a clustered index was constructed on the partitioning attribute. The motivation for this physical organization was to cover a broad spectrum of the performance space with only a few queries. First, in the case of Experiment 1 (a single-tuple, indexed retrieval on the partitioning attribute) this declustering/indexing strategy allows the query to be directed to a single node for processing where it incurs a minimum number of I/Os. The same is true for Experiment 3 — a single tuple update. On the other hand, the queries in Experiment 2 (1% indexed selection on the partitioning attribute) and in Experiment 4 (update between 10% and 50% of the tuples selected by the query used for Experiment 2) must be sent to all processors for execution because they both involve range selections on a hash partitioned relation.

If the relations had instead been range partitioned on the selection attribute, then queries 2 and 4 could have been directed to a subset of the processors, reducing their response time and improving the overall throughput of the system. The reason that we did not elect to use this alternative, is that we wanted to bracket the performance space with as few queries as possible. Using range declustering would have required us to push the simulations found in Experiments 2 and 4 to even[8] higher multiprogramming levels in order to demonstrate the differences among the various strategies that we could observe at lower MPLs when hash partitioning was used.

This partitioning/indexing combination chosen does, however, have a subtle impact on the performance of the ID scheme that the reader should be aware of. As an example, consider a system consisting of 4 processors (P1, P2, P3, and P4) each with 1 disk and assume that each disk page can hold only 2 tuples. Assume also a relation containing 24 tuples with partitioning attribute values 1 to 24 which is hash declustered using "mod 4" as the hash function. Thus, the tuples stored at P1 will have key values 1, 5, 9, 13, 17, and 21. After the tuples have been declustered, a clustered (ie., sorted) index is created at each node on the partitioning attribute. On P1, this step will place the tuples with keys 1 & 5 on page 1, 9 & 13 on page 2, and 17 & 21 on page 3. With ID, there are two ways of making a backup copy of the tuples residing on P1. The approach used by Teradata is to apply a second hash function to the key attribute of each tuple, mapping each tuple on P1 to either P2, P3, or P4 (the other processors do the same for their tuples). The advantage of this approach is that when P1 fails, a single tuple selection query on the partitioning attribute can be routed directly to the backup processor by using the second hash function. Its disadvantage is that updates must be applied in both places (incurring additional disk I/Os and processing overhead for every single update — during the normal as well as the failure mode).

The second approach for constructing the backup fragments is to distribute duplicate copies of the pages of the primary copy among the other nodes in the relation cluster. Thus, page 1 from P1 will be placed on P2, page 2 on P3, and page 3 on P4. The advantage of this approach is that updates to page 1 on P1 can be reflected on P2 by

[8] As it was, the simulation model is so detailed that it ran "forever".

simply shipping a copy of the page to P2. The disadvantage is that when P1 fails, single tuple selections that would normally be handled only by P1 must now be sent to P2, P3, and P4 for processing. For the particular database design that we have chosen, one of P2, P3, and P4 will search their index on the backup fragments to locate the desired tuple. The others will search their index only to find no matching tuple. We think that this is the better of the two alternatives because the path length for updates in the normal mode of operation is much shorter[9].

This indexing/partitioning combination also impacts the performance of ID mechanisms when executing the 1% selection operation of Experiments 2 and 4 in the failure mode of operation. Each relation consists of about 50,000 8Kbyte pages — or about 1500 pages/disk. Since the relation is hash partitioned on the selection attribute, each of the 32 disks will produce approximately 15 pages of result tuples. These 15 pages will overlap one and occasionally two of the backup subfragments (each backup subfragment will contain approximately 50 pages). Therefore, when a disks fails, the subquery originally served by the failed primary fragment will be served by one or two backup subfragments and not by all the processors in the cluster. A similar effect occurs with CD because the division of responsibility between the primary and backup copies is based on attribute value ranges.

4.3. Performance Results for Selection Queries

This section examines the relative performance of chained declustering (CD), interleaved declustering (ID), and mirrored disks (MD) for two different selection queries.

Experiment 1: Single tuple selection on the partitioning attribute

The first query tested was a single-tuple, exact-match selection on the partitioning attribute using an index. Since the selection is on the partitioning attribute the query can be directed to a single node for execution. Figure 7a shows the average throughput obtained by each scheme in the normal mode of operation. All three schemes provide approximately the same performance when the multiprogramming level (MPL) is less than 24. With 32 disks and less than 24 outstanding disk requests, the chance that there is more than one request in a disk queue is very small. Hence, the order in which requests are serviced is likely to be the same for all three schemes and thus so are their disk service times.

When the MPL is greater than 24, the probability of more than one request waiting in the disk queue becomes higher, in turn, increasing the effectiveness of the elevator disk scheduling algorithm used by the CD and ID mechanisms. Consequently, their average seek distance becomes smaller than that of the MD mechanism. For

[9] This design does not preclude the use of record-level locking. Basically, an updated page is sent to the appropriate backup node when the buffer pool manager on the primary site forces the updated page to disk.

```
sc 0.35
1 8
2 9
3 9
4 10
file GRN/n16_nclidx_sel.0_3p8.nf.xg
```

Figure 7a: Single Tuple Selection
Normal Mode

Figure 7b: Single Tuple Selection
Failure Mode

example, at a MPL of 72, the average disk seek (service) time for CD is 7.19 ms (21.62 ms), whereas it is 8.31 ms (22.74 ms) with MD. As a result, CD and ID provide about the same level of throughput and they both process more queries per second than MD when MPL ≥ 48. Henceforth, we shall refer to this effect referred to as the **Disk Scheduling Effect**.

In the failure mode of operation (Figure 7b), all three schemes suffers little (or no) performance degradation at low multiprogramming levels (MPL ≤ 4) because the processors and disk are under utilized. As the MPL increases, however, the impact of a disk failure becomes more and more significant. When MPL > 24, the throughput of the MD scheme levels off because the remaining disk in the failed mirrored pair is fully utilized and becomes a bottleneck. On the other hand, with the CD and ID schemes the throughput continues to increase because both mechanisms do a better job of distributing the workload originally served by the failed disk (henceforth referred to as the **Load Balancing Effect**). Comparing Figures 7a and 7b, one can see that, at a MPL of 72, the decrease in throughput due to a disk failure with CD is about 10%, while it is about 20% with ID. In contrast, the decrease in throughput with mirrored disks is higher than 40%.

The performance differences between the ID and CD mechanisms in Figure 7b is the result of differences in their disk utilizations when a failure occurs. For example, with the ID scheme, at a MPL of 72, the average utilization of the remaining disks in the cluster that suffered a disk failure is around 95% while the utilization of the drives in the other clusters is about 60%. On the other hand for CD, the disk utilization of each of the remaining drives is around 70% at a MPL of 72. With respect to CPU utilization, it is less than 40% for all three mechanisms. One interesting observation is that for CD and ID, the CPU utilization of the processor that has only one operational disk attached is only about 20%. This effect occurs because the CPU utilization is proportional to the number of pages processed by a node which, in turn is proportional to the number of operational disks it has.

Experiment 2: 1% selection query using a clustered index

This experiment considers the performance of three mechanisms while executing an indexed selection query with a 1% selectivity factor. The source relation is assumed to be hash partitioned and thus each query must be sent to all active nodes for processing. With all three schemes, each node produces 1250 result tuples that are returned to the submitting terminal. To process this query using the MD mechanism in the normal mode of operation, each processor will read 2 or 3 index pages[10] and 35 data pages. In the case of CD or ID, each processor will read 2 or 3 index pages and 18 data pages from **each** of its two disks. Both read and process two more index pages and one more data page than with MD because their primary and backup fragments are distributed across both disk drives. In addition, twice as many operator processes are activated with CD and ID. While the CD and ID schemes incur this extra disk overhead, they also benefit from the corresponding higher degree of intra-query parallelism (henceforth referred to as **Query Parallelism Effect**) until the CPU becomes a bottleneck at higher multiprogramming levels.

The results obtained are presented in Figures 8a and 8b. In the normal mode of operation, CD and ID provide more throughput than MD until the MPL is greater than 12 at which point the CPU becomes 100% utilized[11]. On the other hand, with the MD scheme, a CPU bottleneck does not form and, the throughput does not level off until the MPL reaches 24. Ultimately, at a MPL of 48, the MD scheme provides about 5% more throughput. Figure 8a illustrates that there is a tradeoff between the benefit of a higher intra query parallelism and the overhead of scheduling more operator processes and processing more index pages. If a system will be consistently operated under high CPU utilization (i.e., its applications are CPU bound), then the partitioning strategy/data placement algorithm used with CD and ID should be modified to use only 16 instead of 32 fragments (by treating the two disks attached to a processor as one "logical" unit).

Figure 8b shows the throughput of the three mechanisms in the event of a disk failure. At a MPL of 1, the throughput provided by CD and ID drops by almost 40%. As discussed in Section 4.2, the principal cause of this drop is that with both schemes the workload of the failed disk ends up being handled by a single disk which ends up

```
sc 0.35
1 8
2 9
3 10
4 12
file GRN/n16_clidx_sel.01_3p8.nf.xg
```

Figure 8a 1% Selection
Normal Mode

Figure 8b 1% Selection
Failure Mode

[10] Normally, 2 index pages are read. However, when the range of the selection predicate overlaps two the range of two leaf pages, 3 index pages will be read.

[11] In our simulation model (and in Gamma), the processor is responsible for transferring data from I/O channel's FIFO buffer to main memory. Without this overhead, the CPU bottleneck would form at a higher MPL.

servicing twice as many requests as the other disks. In addition, when CD and ID are operating in the failure mode, the average seek distance is longer because both the primary and backup copies are being accessed. At higher MPLs, since there are multiple outstanding queries, all generating I/O requests, the work of the failed disk becomes evenly distributed among the remaining disks in the failed cluster. Consequently, the performance degradation with CD and ID will be less drastic. Indeed, as demonstrated by Figure 8b, when the MPL ≥ 12, the reduction in throughput is about 3.5% with CD and about 8% with ID.

With the MD scheme, there is little or no performance degradation at a MPL of 1 because the other disk in the mirrored pair is idle and can assume the workload without penalty. However, at MPL of 2 the utilization of this mirrored pair rises to 95% while the remaining disks remain 30% utilized— causing the overall throughput of the MD mechanism to level off. At a MPL of 4, the remaining disk in the failed mirrored pair is fully utilized and truly becomes a bottleneck. Consequently, the MD throughput levels off when MPL ≥ 4.

With the CD and ID schemes, the throughput continues to increase until a MPL of 12 is reached. At this point, the CPU is fully utilized and becomes the bottleneck. Figure 8b also illustrates that CD provides higher throughput than ID in the failure mode of operation because it does a better job at load balancing (**Load Balancing Effect**) while both schemes incur about the same level of overhead in the event of a disk (node) failure.

4.4. Performance Results for Update Queries

For an update query, each time a data item is updated, the change must be reflected in both the primary and the backup copies of the data item. Since in the case of both CD and ID, the page containing the backup copy of the item is on a disk that is connected to a different processor, each update incurs CPU cycles for packaging, sending, and receiving the page over the communications network, as well as a wire delay in the communication network (henceforth referred to as the **Remote Update Overhead**). While no extra CPU cycles are required with the MD mechanism, the write to the mirrored pair ends up synchronizing both disk arms and the average seek distance becomes 0.47n, where n is the number of cylinders [Bitt88]. This is 0.14n higher than the average seek distance of a single disk. Henceforth, we shall refer to this effect as the **Synchronizing Write Overhead**.

Two query types are studied in this section: a single tuple update query using a clustered index, and a query that selects 1% of the tuples using a clustered index and then updates between 10% and 50% of the selected tuples. Since the relations are hash partitioned, the first query will be sent to a single processor while the second will be sent to all processors.

In our experiments, an update transaction is not committed until both the primary and backup copies have been updated. In addition, we assume that the attribute being updated is not indexed and is not the partitioning attribute.

Experiment 3: Single tuple update query

The results of this experiment are contained in Figures 9a and 9b. All three schemes provide comparable performance at low multiprogramming levels. Since the system resources (CPU, disk, and network) are under utilized, the overhead of updating the backup copies is not a significant factor. As the MPL is pushed higher, differences among the schemes begin to emerge. In particular, with MD the overhead of synchronizing disk writes starts to limit the overall performance of the system. On the other hand, the overhead of a remote update with CD and ID is almost entirely a function of the page size and is not significantly affected by the MPL (unless, of course, the network or network interface becomes a bottleneck). As a result, the CD and ID schemes provide significantly higher throughput when the MPL is greater than 24 in the normal mode of operation.

Figure 9b shows the throughput of the three mechanisms in the event of a disk failure. When the MPL is less than 24, the performance of the CD and MD strategies are not significantly affected because both the CPU and disk are under utilized and the load increase that results from the failure is not significant. ID's performance is affected slightly more because it reads and processes more index pages in the failure mode.

As the MPL is pushed higher significant differences in performance begin to appear. For example, at a MPL of 72, Figures 9a and 9b show that the throughput drops by only 3.3% with the CD strategy, by 14.9% with the ID strategy, and by 9.2% with the MD strategy. There are two reasons why ID suffers a larger drop in performance than CD. First, ID can distribute the workload of the failed disk only among the remaining disks in the cluster containing the failed drive while CD is able to evenly redistribute the workload among the remaining 31 disks. Second, with ID, a query originally served by the failed disk has to be sent to all the other processors in the cluster for processing[12], increasing the number of index pages read and processed.

sc 0.35
1 8
2 9
3 10
4 12
file GRN/n16_nclidx_update.0_3p8.nf.xg

Figure 9a: Single Tuple Update
Normal Mode

Figure 9b: Single Tuple Update
Failure Mode

[12] Only one subfragment finds and updates the matching tuple, however.

With the MD mechanism, the remaining operational disk in the failed mirrored pair must assume the entire workload of the pair. However, unlike the single tuple selection case, the decrease in throughput is less than 50%. There are two major reasons for this behavior. First, since writes always go to both disks, the failed disk must assume only the read requests originally handled by the failed disk. Each query reads 3 pages and updates 1 page. Given two such queries, each disk in a mirrored pair is responsible for 3 disk reads and 2 disk writes (assuming that the workload is uniformly distributed between the two disks). When a disk failure occurs, the remaining disk in the failed pair will be responsible for 6 disk reads and 2 disk writes, resulting in a 60% increase in the number of disk requests. With a 20% buffer hit ratio for read requests (the number used in our experiments), the increase in disk requests decreases further to roughly 54%. Second, in a failed mirrored pair, there is no longer any need to synchronize the two disk arms. Consequently, the remaining disk can process write requests much more efficiently, offsetting some of the impact of the increase in the number of disk requests.

The update query used in this experiment is I/O bound. If an update query is CPU bound, the MD scheme may provide better performance than CD and ID in the normal mode of operation. This is because the **Remote Update Overhead** incurred by CD and ID consumes extra CPU cycles while the **Synchronizing Write Overhead** associated with MD increases the disk service time. Since advances in CPU technology have occurred much faster than advances in disk technology [Joy85, Fran87], we believe that such operations will almost certainly be I/O bound in the future and thus the additional cost of doing remote updates will not be that significant.

Experiment 5: 1% selection with X% update using a clustered index

In this experiment, we again assume that the relation being updated is hash partitioned[13] and that the query has to be sent to all operational nodes for processing. The query in this experiment uses a clustered index to sequentially read 1% of the tuples, randomly updating X% of the ones read. Figures 10a and 10b show, respectively, the throughput of the three consistency mechanisms in the normal and failure modes for update frequencies of 10%, 30%, and 50%. In the normal mode of operation, CD provides slightly higher throughput than ID and much higher throughput than MD except at a MPL of 1. At a MPL of 1, both the CD and ID schemes provide higher throughput than the MD scheme when X is equal to 10, whereas MD provides higher throughput than CD and ID when X is equal to 30 or 50. Two factors interact to cause this switch. First, at a MPL of 1 both CD and ID benefit from the effect of intra-query parallelism (see Experiment 2). Second, with ID and CD, as the update frequency is increased, more and more disk contention occurs between reads/writes of the primary fragments and writes to the backup fragments. At an update frequency of 10% the Query Parallelism Effect dominates, and CD and ID provide better

[13] The results will be the same if the relation is range partitioned and the clustered index is constructed on a nonpartitioning attribute.

overall throughput. However, at an update frequency of 30% or 50%, the overhead of a longer disk service time dominates and MD has the best performance. For example, at a MPL of 1, the average disk service times for CD/ID is 19.1 ms. when X = 10 and 23.0 ms. when X = 50 (the corresponding disk service times for MD are 16.5 ms. and 18.2 ms., respectively)

Beginning with a MPL of 2, the update portion of the query begins to cause disk contention with MD as well, resulting in a higher disk service time. Like the previous case with CD and ID, the higher the update percentage is, the more severe the disk contention will be. In addition, having to synchronize the disk heads when performing each write, also increases the average service time. For example, at a MPL of 2, the average disk service time with MD is 19.4 ms. when X = 10 and 22.8 ms. when X = 50.

When X = 10, the throughput of MD continues to increase until a MPL of 4 at which point one or more of the disks becomes fully utilized and forms a bottleneck. The continued increase in disk service times with MD results in a slight decrease in throughput from MPLs of 4 to 12. When MPL ≥ 12, the throughput with MD levels off. With X= 30 and 50, throughput decreases slightly from MPLs of 2 to 4 and levels off after MPL ≥ 4. With both CD and ID, the throughput increases significantly from a MPL of 1 to 4 for all three update levels. When MPL > 4, the rate of increase drops significantly because the disks are nearly 100% utilized. The small increase in throughput is mainly due to a decrease in disk service time as the result of the elevator scheduling algorithm employed by the disk controller. At a MPL of 24, CD and ID provide, respectively, 46%, 55%, and 57%, more throughput at X = 10, 30, and 50 than the MD scheme.

Figure 10b shows the throughput provided by the three mechanisms in the event of a disk failure. Overall, CD and ID provide significantly better performance for all update frequencies at all multiprogramming levels except 1 because they both do a better job of balancing the load in the event of a disk failure. With MD, as the MPL is increased beyond 1, the failed mirrored pair becomes the bottleneck. While CD and ID exhibit a fairly significant drop in performance at a MPL of 1 (when compared with their normal performance), at a MPL of 24, the drop is less than 5% with CD and less than 10% for ID.

sc 0.35
1 8
2 9
3 10
4 12
file GRN/fig10

Figure 10a: 1% Selection with X% Update
Normal Mode

Figure 10b: 1% Selection with X% Update
Failure Mode

4.5. Varying CPU Speed and/or Page Size

In addition to the experiments presented in the previous two sections, we also studied the effect of increasing the CPU speed from 3 to 14 MIPS and decreasing the page size from 8 Kbytes to 4 Kbytes. Except for the 1% selection using a clustered index (Experiment 2), the other queries remained I/O bound and the relative performance of the different replication schemes did not change significantly. On the other hand, in the case of Experiment 2, MD no longer performs better than CD or ID at high MPLs in the normal mode of operation. For example, with a 14 MIP CPU, the throughput of CD and ID is 32% higher with MD at a MPL of 48. At the same MPL and a 3 MIP CPU, MD provide 5% more throughput.

5. Conclusions

In this paper, we have studied the performance of the chained declustering, interleaved declustering, and mirrored disk schemes using a simulation model of Gamma database machine. In particular, we have examined (1) the relative performance of different strategies when no failures have occurred, (2) the effect of a single node failure on system throughput and response time, and (3) the performance impact of varying the CPU speed and/or disk page size on the different replication strategies, and (4) the tradeoff between the benefit of intra query parallelism and the overhead of activating and scheduling extra operator processes.

Experiments were conducted using both read-only, selection queries and update queries requiring both reads and writes. For selection queries, chained declustering and interleaved declustering were shown to perform comparably in the normal mode of operation. Both performed better than mirrored disks if an application is I/O bound (due to disk scheduling), but slightly worse than mirrored disks if the application is CPU bound. In the event of a failure, chained declustering was indeed able to balance the workload among the remaining disks, while interleaved declustering redistributed the workload within the failed cluster; mirrored disks cannot do any load redistribution, so the mirror image of the failed disk had to process all requests originally served by the failed disk. As a result, chained declustering provided slightly better performance than interleaved declustering and much better performance than mirrored disks in a failure mode of operation.

To update the backup copy of a data item, chained declustering and interleaved declustering incur the CPU overhead of packaging and sending the updated data to the remote node where the backup copy is stored. In addition, the remote node consumes extra CPU cycles to receive the network packet (containing the updated page) and to initiate an extra disk write operation to write the updated page to disk. With mirrored disks, both copies of a data item are stored on disks attached to the same processor. Consequently, no extra CPU cycles are needed for updating the backup copy. However, with mirrored disks each write operation ends up synchronizing the read/write heads of both disks in the mirrored pair [Bitt88]. Therefore, the disk service time for a write becomes the maximum service time of two writes. In addition, both disk arms in a mirrored pair will be at the same cylinder after each write operation, effectively reducing the number of disk arms available for serving the next read request to one. Consequently, with mirrored disks, the average disk service time per request is longer for update queries than for select queries.

For update queries, because chained declustering and interleaved declustering incur CPU overhead while mirrored disks incurs overhead in disk seek distance, the relative performance of the three schemes depends on the relative performance of the processor and the disk drive. In the normal mode of operation, if an update query is I/O bound, chained declustering and interleaved declustering perform better than mirrored disks. On the other hand, if an update query is CPU bound, the mirrored disk mechanism will perform somewhat better. However, because advances in CPU technology occur much faster than those of disk drive technology, we believe that most future database applications will be disk bound. Consequently, chained declustering should provide better performance than mirrored disks even with update intensive applications.

When failures occur in the mirrored disk scheme, a bottleneck forms at the failed mirrored pair; throughput is then limited by the rate at which the failed pair can service requests. On the other hand, with chained declustering the workload of a failed disk is again evenly redistributed among the remaining disks. Consequently, chained declustering provides much higher throughput than mirrored disks in the event of a failure. The relative performance of chained declustering and interleaved declustering (ID) in the event of a disk failure depends on the query type and the size of an ID cluster. With an ID cluster size of 8, our experiments showed that chained declustering can provide as much as 14% more throughput than interleaved declustering for a single tuple update query and as little as a 3% improvement for a 1% update query. Notice, however, that with an ID cluster size of 8, besides providing lower throughput, the interleaved declustering scheme is 3.5 times more likely to have data unavailable than the chained declustering scheme [Hsia90a].

Our future work includes studying the performance tradeoffs of the three replication schemes with skewed data access and the possibility of dynamic load balancing for the chained declustering scheme. Without data replication, data partitioning (or declustering) is commonly used with multiprocessor multi-disk database machines to break hot spots and achieve load balancing. Hot spots, however, may be dynamic in nature, and the database may need to be reorganized periodically. With chained declustering, a query (subquery) can be processed at the node storing either the primary or backup copy of the matching tuples. In addition, the work of a node may be shifted to its neighbor without physically moving data because nodes are "chained" together through the primary and backup copies of a fragment. These two characteristics of the chained declustering scheme provide a good opportunity for dynamic load balancing [Care86] when a hot spot changes over time. Consequently, a reorganization of the database may not be required.

Acknowledgements

We would like to thank Mike Carey for his invaluable help while developing the simulation model of Gamma and in interpreting the results obtained.

References

[Anon85] Anon et. al, "A Measure of Transaction Processing Power," TR# 85.1, Tandem Computer, Cupertino, CA, 1985.

[Bitt88] Bitton, D. and J. Gray, "Disk Shadowing," *Proceedings of the 14th International Conference on Very Large Data Base,* Los Angeles, August 1988.

[Bitt89] Bitton, D., "Arm Scheduling in Shadowed Disks," *COMPCON,* IEEE Press, March 1989.

[Borr81] Borr, A., "Transaction Monitoring in Encompass [TM]: Reliable Distributed Transaction Processing," *Proceedings of the 7th International Conference on Very Large Data Base,* 1981.

[Care86] Carey, M. and H. Lu, "Load Balancing in a Locally Distributed Database System," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* 1986.

[Care89] Carey, M., Livny, M., "Parallelism and Concurrency Control Performance in Distributed Database Machines," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* Portland, Oregon June 1989.

[Chen90] Chen, P., Gibson, G., Katz, R., and Patterson D., "An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890," *Proceedings of ACM SIGMETRICS conference,* Colorado, May 1990.

[Cope88] Copeland, G., Alexander, W., Boughter, E., and T. Keller, "Data Placement in Bubba," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* Chicago, May 1988.

[Cope89] Copeland, G. and T. Keller, "A Comparison of High-Availability Media Recovery Techniques," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* Portland, Oregon June 1989.

[DeWi86] DeWitt, D., Gerber, R., Graefe, G., Heytens, M., Kumar, K., and M. Muralikrishna, "GAMMA - A High Performance Dataflow Database Machine," *Proceedings of the 12th International Conference on Very Large Data Base,* Japan, August 1986.

[DeWi88] DeWitt, D., Ghandeharizadeh, S., and Schneider, D., "A Performance Analysis of the Gamma Database Machine," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* Chicago, May 1988.

[DeWi90] DeWitt, D., Ghandeharizadeh, S.,Schneider, D., Bricker, A., Hsiao, H, and Rasmussen, R, "The Gamma Database Machine Project," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 2, No 1, March 1990.

[Fran87] Frank, P., "Advances in Head Technology," Presentation at *Challenges in Disk Technology Short Course,* Institute for Information Storage Technology, Santa Clara University, Santa Clara, CA, December 15-17, 1987.

[Gerb87] Gerber, R. and D. DeWitt, "The Impact of Hardware and Software Alternatives on the Performance of the Gamma Database Machine", Computer Sciences Technical Report #708, University of Wisconsin-Madison, July, 1987.

[Gray88] Gray, J., Sammer, H., and Whitford, S., "Shortest Seek vs Shortest Service Time Scheduling of Mirrored Disc Reads" Tandem Computers, December 1988.

[Gray90] Gray, G., Horst, B., and Walker, M., "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput," *Proceedings of the 16th Onternational Conference on Very Large Data Base,* Brisbane, Australia, August 1990.

[Hsia90a] Hsiao, H. and DeWitt, D., "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines," *Proceedings of the 6th International Conference on Data Engineering,* Los Angeles, CA, February 1990.

[Hsia90b] Hsiao, H., "Performance and Availability in Database Machines with Replicated Data," Computer Sciences Technical Report #963, University of Wisconsin-Madison, August, 1990.

[Inte88] Intel Corporation, **iPSC/2 User's Guide,** Intel Corporation Order No. 311532-002, March 1988.

[Joy85] Joy, B., Presentation at ISSCC '85 panel session, February 1985.

[Katz78] Katzman, J., "A Fault-Tolerant Computing System," *Proceedings of the 11th Hawaii Conference on System Sciences,* January 1978.

[Kim86] Kim, M., "Synchronized Disk Interleaving," *IEEE Transactions on Computers,* Vol. C-35, No. 11, November 1986.

[Livn87] Livny, M., S. Khoshafian, and H. Boral, "Multi-Disk Management," *Proceedings of ACM SIGMETRICS Conference,* Alberta, Canada, 1987.

[Livn89] Livny, M., "DeNet User's Guide," Version 1.5, Computer Sciences Department, University of Wisconsin-Madison, 1989.

[Patt88] Patterson, D., Gibson, G., and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* Chicago, May 1988.

[Ries78] Ries, D. and Epstein, R., "Evaluation of Distribution Criteria for Distributed Database Systems," UCB/ERL Technical Report M78/22, UC Berkeley, May 1978.

[Ston75] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," *Proceedings of the SIGMOD Workshop on Management of Data*, San Jose, Calif., May 1975.

[Ston86] Stonebraker, M., "The Case for Shared Nothing," *Database Engineering,* Vol. 9, No. 1, 1986.

[Tand87] Tandem Database Group, "NonStop SQL, A Distributed, High-Performance, High-Reliability Implementation of SQL," *Workshop on High Performance Transaction Systems,* Asilomar, CA, September 1987.

[Tera85] Teradata, "DBC/1012 Database Computer System Manual Release 2.0," Document No. C10-0001-02, Teradata Corp., NOV 1985.

[Triv82] Trivedi, K., in *Probability and Statistics with Reliability, Queueing, and Computer Science Applications,* Prentice-Hall Inc., New Jersey, 1982.