NPS52-85-011

# NAVAL POSTGRADUATE SCHOOL

Monterey, California

# A Benchmarking Methodology for the Centralized-Database Computer with Expandable and Parallel Database Processors and Stores

Steven A. Demurjian

David K. Hsiao

James R. Vincent

August 1985

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R.H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract from the Office of Naval Research.

This report was prepared by:

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER NPS52-85-011 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) A Benchmarking Methodology for the Centralized-Database Computer with Expandable and Parallel Database Processors and Stores | | 5. TYPE OF REPORT & PERIOD COVERED |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Steven A. Demurjian, David K. Hsiao, James R. Vincent | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School, Monterey, CA 93943 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N; RR014-08-01 N0001485WR24046 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research, Arlington, VA 22217 | | 12. REPORT DATE August 1985 |
| | | 13. NUMBER OF PAGES 46 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper a benchmarking methodology for a new kind of database computers is introduced. The emergence in the research community and in the commercial world of this kind of database computer (known as the multiple-backend database computers), where each computer system is configured with two or more identical processors and their associated stores for concurrent executionof transactions and for parallel processing of a centralized database spread over separate stores, is evident. The motivation and characterization of the multiple-backend database computer are first given. The need and lack of a methodology for benchmarking the new computer with a variable number of backends for the same database or with a fixed number of differenct capacities are also evident. The measures (benchmarks) of the new computer are articulated and established and the design of the methodology for conducting the measurements is then given. Because of the novelty of the database computer architcture, the benchmarking methodology is rather elaborate and somewhat complicated. To aid our understanding of the methodology, a concrete sample is given herein. This sample also illustrates the use of the methodology. Meanwhile, a CAD system which computerizes the benchmarking methodology for systematically assisting the design of test databases and test-transaction mixes, for automatically tallying the design data and workloads, and for completely generating the test databases and test-transaction mixes is being implemented.

# A Benchmarking Methodology for the Centralized-Database Computer with Expandable and Parallel Database Processors and Stores *

Steven A. Demurjian, David K. Hsiao and James R. Vincent **

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943
U. S. A.

August 1985

## ABSTRACT

In this paper a benchmarking methodology for a new kind of database computers is introduced. The emergence in the research community and in the commercial world of this kind of database computer (known as the *multiple-backend database computers*), where each computer system is configured with two or more identical processors and their associated stores for concurrent execution of transactions and for parallel processing of a centralized database spread over separate stores, is evident. The motivation and characterization of the multiple-backend database computer are first given. The need and lack of a methodology for benchmarking the new computer with a variable number of backends for the same database or with a fixed number of backends for different capacities are also evident. The measures (benchmarks) of the new computer are articulated and established and the design of the methodology for conducting the measurements is then given. Because of the novelty of the database computer architecture, the benchmarking methodology is rather elaborate and somewhat complicated. To aid our understanding of the methodology, a concrete sample is given herein. This sample also illustrates the use of the methodology. Meanwhile, a CAD system which computerizes the benchmarking methodology for systematically assisting the design of test databases and test-transaction mixes, for automatically tallying the design data and workloads, and for completely generating the test databases and test-transaction mixes is being implemented.

* The work reported herein is supported by grants from the Department of Defense STARS Program and from the Office of Naval Research and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

** CAPT James R. Vincent is now at HQ/SIPN, Scott AFB, IL 62225.

## TABLE OF CONTENTS

## 1. THE MOTIVATION AND NEED

We need to benchmark a new kind of database computers. The need is accentuated by the claims that the new breed of database computers can achieve performance gains and capacity growth. In other words, unless and until we have benchmarked these computers, we will not be able to verify their claims. These computers are new because they resemble neither the traditional approach to database management by placing the system software in a mainframe computer such as SQL/DS in an IBM 3033 [1], nor the more recent approach to database management by utilizing the dedicated hardware and software in a single backend computer such as the Britton-Lee IDM 500 [2]. Whereas in the *mainframe-based approach* a database system is characterized as an applications program (albeit, a major one), which shares the resources of the mainframe computer with other applications programs (depicted in Figure 1), in the *single-backend approach* a database system has the exclusive control and use of the resources of the entire backend computer (depicted in Figure 2). The term *backend* is meant to be in the 'back' of terminals or general-purpose computers [3], where neither the terminals nor the general-purpose computers (termed the *hosts*) provide the database management services. Instead, the database management services are provided by the backend computer to the user or user programs (*transactions*) via the terminals or the hosts.

The new kind of the database computers (depicted in Figure 3) is of the *multiple-backend approach* where no database system is mainframe-based and each database system consists of one or more backend controllers (starting with one) and two or more backends (beginning with two usually) and their disk systems interconnected by a network. The controller software is mainly dedicated to the communication with the hosts and the terminals, to the scheduling and control of transaction executions by the backends and to the routing of the responses coming from the backends back to the users. The backend software in the multiple backends is identical and is responsible for carrying out the primary database operations such as the retrieve, insert, delete, and update for the transactions. The sort-and-or-merge operations may also be carried out by the backends either with the help of the controller if the communications network is a simple network or with the help of the interconnecting network if the communications network is also a

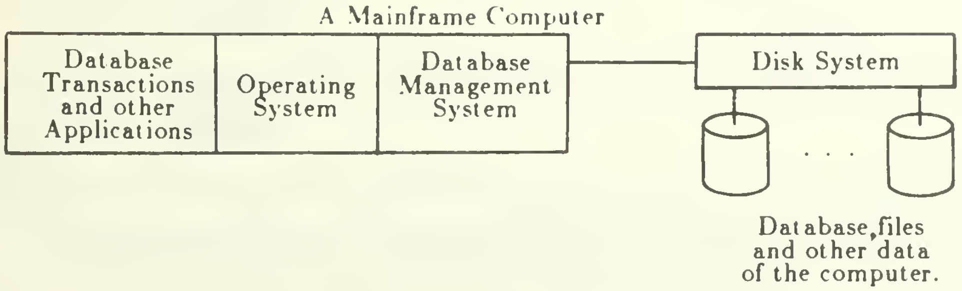
A Mainframe Computer
Database Transactions and other Applications
Operating System
Database Management System
Disk System
. . .
Database files and other data of the computer.

Figure 1. The Mainframe-Based Approach to Database Management.

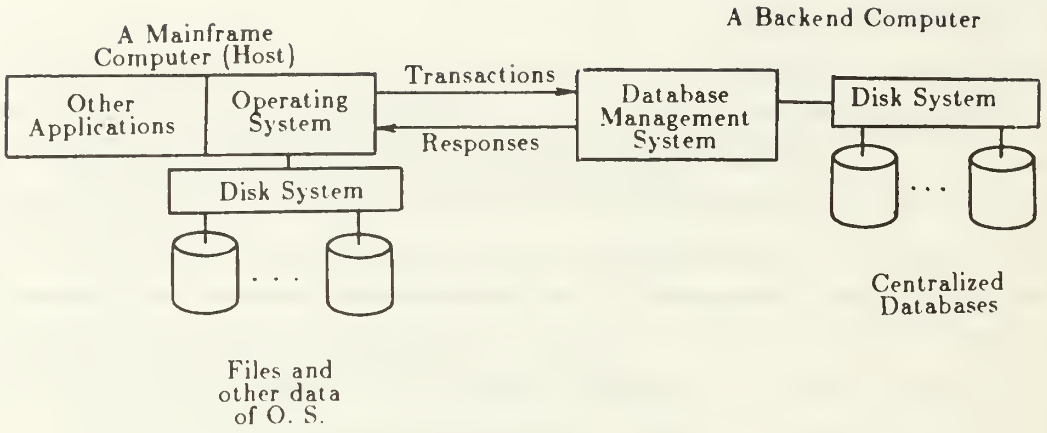

Figure 2. The Single-Backend Approach to Database Management.

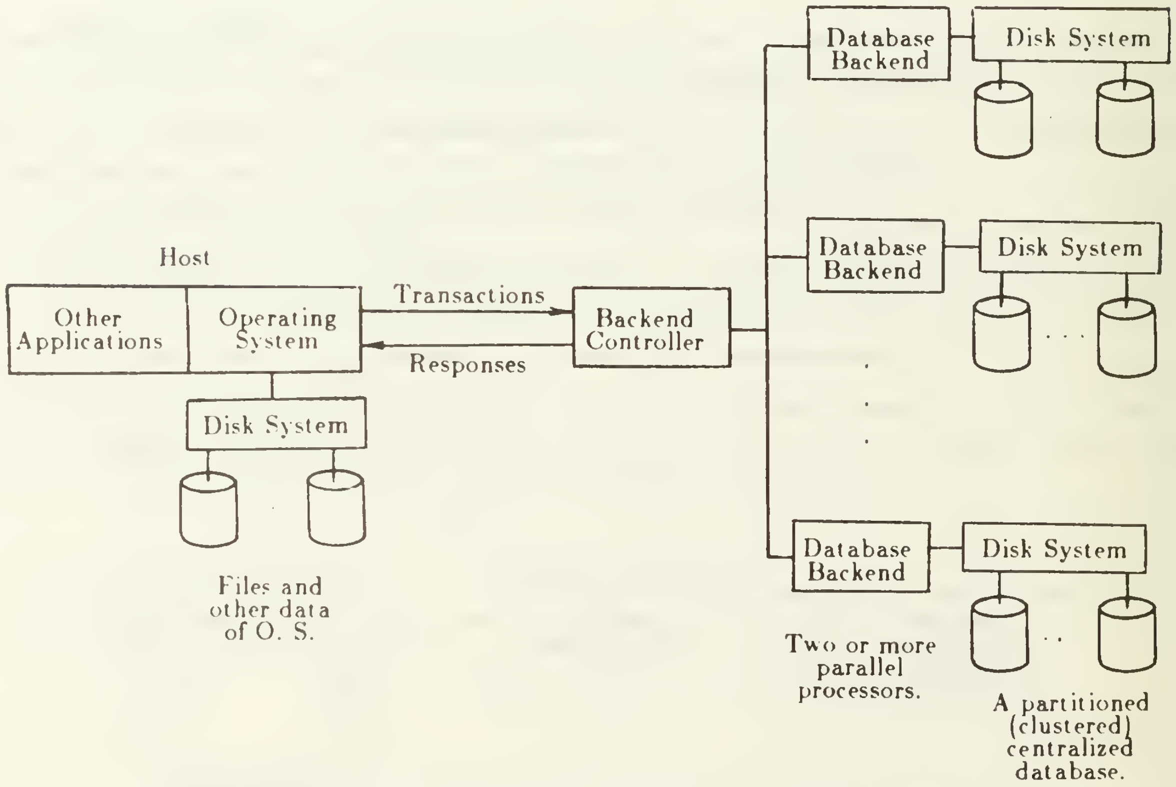

Figure 3. The Multiple-Backend Approach to Database Management.

sorting-and-or-merging network. Examples of the multiple-backend approach to database management can be found in the experimental Multi-Backend Database System (MBDS) utilizing an Ethernet interconnection 4 and the commercial Teradata DBC 1012 system consisting of a communications and sorting network, known as the Y-net [5].

Unlike the mainframe-based and single-backend approaches, the multiple-backend approach emphasizes great-capacity and high-performance database management. Furthermore, it attempts to relate the capacity growth and performance gains to the number of backends used in the system. In other words, when new backends and their disk systems are added to a multiple-backend database computer, an increase in both the capacity and the performance would likely be produced. In the case of MBDS, the system is expandable in terms of tens of backends and associated disk systems, whereas the DBC/1012 is expandable in terms of hundreds of backends and disks. We may then ask whether of not in the former case the capacity growth and performance gains can be measured in tens and in the latter case in hundreds. Clearly, we need a benchmarking methodology for the multiple-backend database computers so that we can verify their growth and gains.

The design of the benchmarking methodology is complicated by the fact that (1) there is the need of test databases which can be used for testing backends of varying numbers, for deriving partitions (clusters) of a database, and for placing the partitions (clusters) on parallel stores; (2) there is the need of test-transaction mixes which can be used for measuring primary database operations in terms of their response times, for verifying the response-time reductions due to additions of backends and the redistribution of the same database, for clarifying the response-time invariance on account of various growths in database capacity with various additions of backends and backend stores; (3) there is the need of systematic ways to generate the test databases and the test-transaction mixes, to conduct the tests, to collect the test results, to interpret the results and to verify the results against established measures (benchmarks). The major portions of the paper consist of the articulation and establishment of the measurement criteria and measures, the design, interpretation, and generation of the test databases, the test-transaction mixes, the test procedures, and the test configurations.

Before presenting our benchmarking methodology, we first outline the architecture and characteristics of the multiple-backend database computers. We also establish the measures (i.e., benchmarks) of the computers These measures (benchmarks) should provide us with precise, quantitative definitions of the notions of capacity growth and performance gains as functions of the numbers of backends.

## 2. THE ARCHITECTURE AND CHARACTERISTICS OF THE MULTIPLE-BACKEND DATABASE COMPUTER

In this section we review the architecture and characteristics of the multiple-backend database computer. From Figure 3, we observe that certain features of the multiple-backend approach to database management must be examined for benchmarking. These features are examined from both hardware and software perspectives. We are also interested in the expandability of the multiple-backend approach.

All of the backends have identical hardware and replicated software which handles concurrent execution of transactions. Consequently, a backend performs directory management, access-path selections, access operations, concurrency control and record (tuple*) processing for insertion, deletion, and update. The backend also controls its own disk system. The number of backends in a given system may be in tens or hundreds.

## 2.1. The Backend Controllers

All of the backend controllers have identical hardware and replicated software which handle preprocessing of the transactions, broadcast the transactions to the backends, keep track of the execution progress of the transactions, assemble the responses from the backends, and route the responses to the users or user transactions originated at the hosts or terminals. The number of backend controllers in a given system is usually one but may be more for redundancy and reliability.

## 2.2. The Interconnecting Network

The interconnecting network can range from a broadcasting network to a cross-bar network. However, since database management involves aggregate functions such as maximum and minimum and sort-and-merge functions such as sequencing and merging (relational joins), the network may have local memories and processors for such functions. Since backends are intended to perform most of the database management operations on their database partitions (clusters) independent of one another, there are minimal communications among the backends and between the controller and its backends. Thus the interconnecting network does not have to be a high-bandwidth communications network. Instead, the network may assist the backends in performing aggregate and sort-and-merge functions. As there is usually only one controller, the use of broadcasting and tree-like networks becomes common.

## 2.3. The Expandability Requirements

The multiple-backend database computer is expandable. The expansion requires the use of the same backend hardware. the replication of the existing backend software on the new hardware. the redistribution of the partitions (clusters) on the old and new disks in order to achieve the desirable effect where multiple transactions being executed in the backends are reading (or writing) and processing multiple data streams of partitions (clusters) coming from (or going to) disks. This effect allows *partition(cluster)-parallel-and-record(tuple)-serial* operations.

## 2.4. The Database Organization

A database must be partitioned (clustered) at the database-creation time. Each partition (cluster) must be placed on the respective disks of the separate backends one block (track) at a time. For a round-robin database placement algorithm. if a partition (cluster) is. for example, of 25 blocks (tracks) and the first available disk track to be used is at Backend 2. then for a 10-backend database system Backend 2 through Backend 6 will have 3 blocks (tracks) of records (tuples) on each of their respective disks, while Backend 1 and Backend 7 through Backend 10 will have only two blocks (tracks) of records

*Certain concepts and terminologies of the non-relational database and relational database are similar. Thus, files mean relations; records tuples; partitions clusters; tracks blocks; merging of two files relational joins; and so on. We shall enclose the similar terms in parentheses.

(tuples) on each of their respective disks. (See Figure 4.)

The controller is responsible for determining the first block (track) (i.e., the first backend) to be used for the data placement and the backends are responsible for placing the records (tuples) on their available tracks. Although different partitioning (clustering) schemes and database placement algorithms may be utilized for a given system, the design of the schemes and algorithms is to create the partition(cluster)-parallel-and-block(therefore, record)-serial effect for the subsequent access operations of the system. More specifically, in the above example we can conclude intuitively that the access and process times for the 25 blocks (tracks) of records (tuples) are shortened to the times for 2 or 3 blocks (tracks) of records (tuples). Thus, a 10-backend database computer may have a throughput of, at least, 8 times that of a single-backend database computer or of a mainframe-based database system.

As new records (tuples) are being inserted into a database, the database placement algorithm will be activated frequently to place the new records on the next available blocks (tracks). This does not require any redistribution of the database. However, as the new backends are being added to the system, it becomes necessary to execute the database placement algorithm for the entire existing database in order to maintain the optimal effect of partition(cluster)-parallel-and-block(track)-serial operations. This is termed the *redistribution* of the database. Such redistribution, although time-consuming, is infrequent (i.e., new backends are not added every day) and has the desirable effect on system performance (i.e., new distribution of partitions or clusters allows a higher degree of parallel access operations).

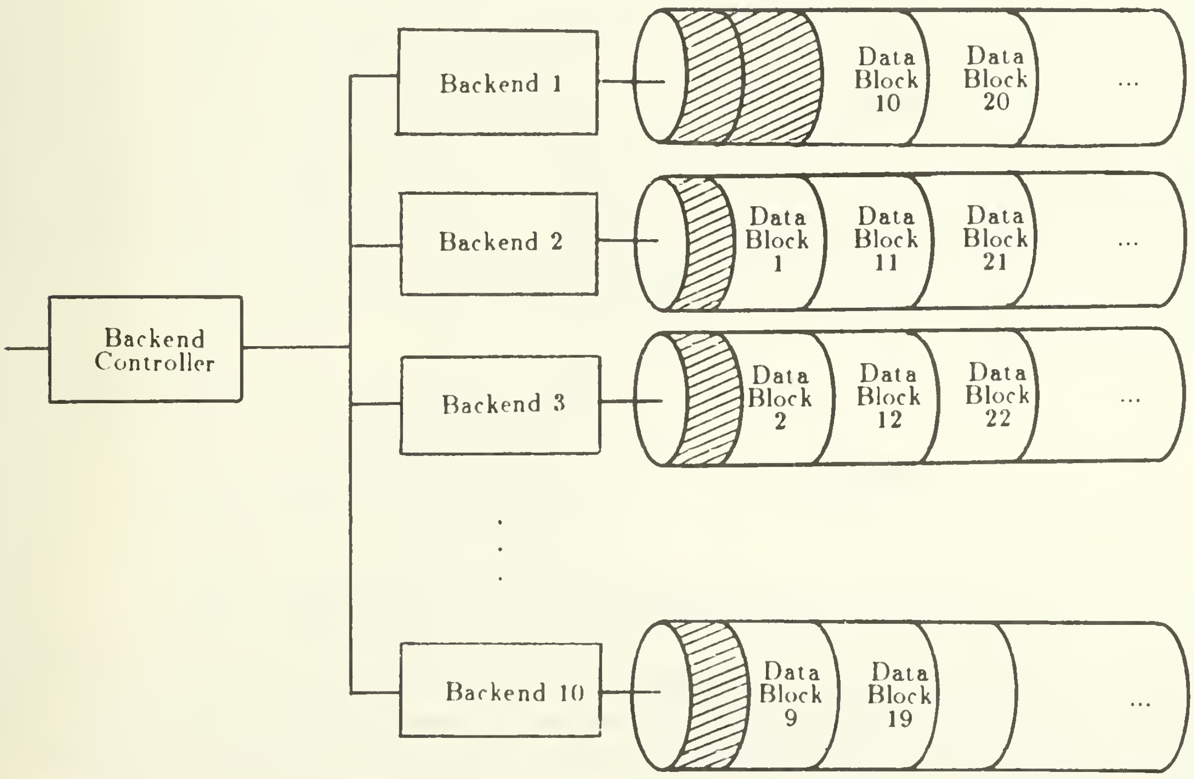

Figure 4. A Round-Robin Database Placement Algorithm for Placing 25 Blocks of Data in a 10-Backend Database Computer.

## 3. THE MEASURES (BENCHMARKS) OF THE MULTIPLE-BACKEND DATABASE COMPUTER

There are two measures (benchmarks) of the multiple-backend database computer. One measure (benchmark) relates to its performance-gains capability, while the second measure (benchmark) corresponds to its capacity-growth potential. Each of these measures (benchmarks) are examined in the following sections. The third part of this section introduces the development and specification of the test (benchmark) transactions and the test (benchmark) databases which are used to conduct the performance-gains and capacity-growth measures (benchmarks).

### 3.1. A Measure (Benchmark) on Performance Gains

Since a multiple-backend database computer may be configured with two or more backends for parallel processing and access, there can be many different configurations. For example, we may want to benchmark a ten-backend configuration versus a twenty-backend configuration. The performance gain of one configuration over the other configuration is measured in the amount of *response-time reductions* from the one computer configuration to the other configuration for the same transaction against the same database. The *response time* of a transaction is defined as the elapsed time between the time that the transaction is issued (i.e., received by the backend controller) and the time that the last response of the transaction is produced (i.e., routed to the transaction) in a single-user, stand-alone environment. Thus, this response time represents the best possible (i.e., shortest) response time that the transaction may incur in a given computer.

Since the contents of both the database and the transaction are not to be changed for this measure, the only changes are the different numbers of backends and the different distributions of the same database in the two configurations. In other words, in a certain configuration X, we have $x$ number of backends and one distribution of the database. In configuration Y, we have $y$ number of backends and a redistribution of the *same* database. Since all of the software and the hardware of the backend, the backend controller and the interconnecting network are the same, the following formula establishes the performance-gains measure (benchmark) of configuration Y with respect to configuration X for a specific transaction and database.

$$\begin{array}{c}\textit{The}\\ \textit{Response-Time}\\ \textit{Reduction}\end{array} = 1 - \left( \frac{\begin{array}{c}\textit{The Response}\\ \textit{Time of}\\ \textit{Configuration } Y\end{array}}{\begin{array}{c}\textit{The Response}\\ \textit{Time of}\\ \textit{Configuration } X\end{array}} \right)$$

Equation 1. The Response-Time-Reduction Formula

Let $x$, the number of backends for configuration X, be 20, and $y$, the number of backends for configuration Y, be 60, then ideally we would like the ratio of response times of configuration Y of 60 backends and configuration X of 20 backends to be 1/3. Consequently, the response-time reduction of the 60-backend database computer over the 20-backend database computer would be 2/3. This example illustrates that if we triple the number of backends of an existing database computer and redistribute the same database on the existing and new disks, we would expect to cut the response time of a transaction by two-thirds. In other words, the response-time reduction is inversely proportional to the ratio of the number of backends of the two configurations.

In reality, the response-time reductions are not likely to reach their ideal proportions. The issue is therefore how close a given multiple-backend database computer can reach its ideal response-time reductions. As shown in Figure 5, we must measure (benchmark) a sufficient number of configurations for a given transaction and database in order to determine the system overheads and their impact on the response times (therefore, on the response-time reductions) of the transaction and the database.

Ideally, we would expect that in Figure 5 $R_i = \frac{1}{i}$ for $i = 1, 2, \ldots, n$, and for large $n$, Typically, $\Delta_1 = 0$ and $\Delta_2 < \Delta_3 < \cdots < \Delta_n$ where $\Delta_i$ is the system overhead incurred in handling the transaction in the $i$-backend configuration. In studying Figure 5, we may expect the benchmarking effort to address the following issues:

(1) What are the values of $\Delta_i$ for the given $i$-backend computers under benchmarking?

(2) How large will $n$ be when there is no further reduction in response time (i.e., $R_n \geqslant R_{n+1}$)?

(3) How large will $n$ be when the system overhead becomes pronounced (i.e., $\frac{\Delta_{n+1}}{n+1} \gg \frac{\Delta_n}{n}$)?

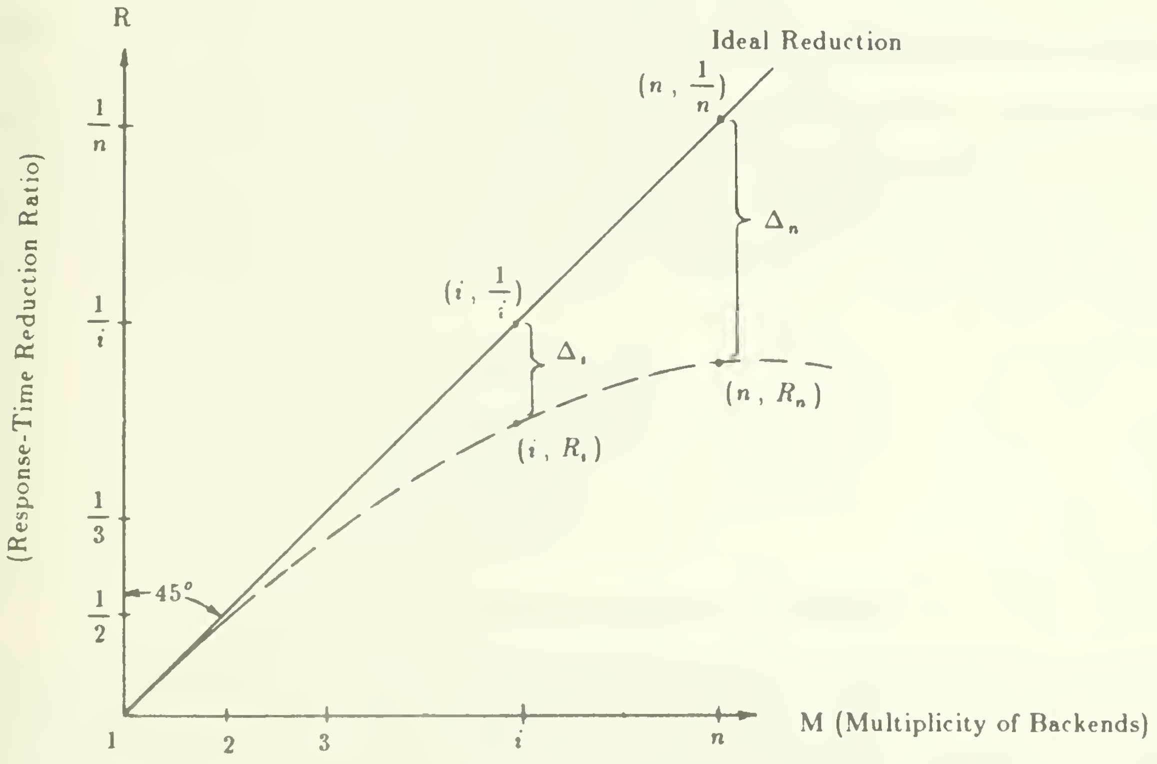

Figure 5. The Response-Time Reduction Measure

## 3.2. A Measure (Benchmark) on Capacity Growth

The capacity growth of one configuration over the other configuration is characterized by the sizes of the responses to the same transaction. As the database grows, the responses to the same transaction increase also. Consequently, the capacity growth of one configuration over the other configuration is also characterized by the sizes of their databases. What we want to measure (benchmark) is whether the response time of a transaction can be held constant despite the capacity growth in a new configuration. To compensate for the extra work necessary in capacity growth, the multiple-backend database computer can offer new configurations with additional backends.

Unlike the previous measure (benchmark) on performance gain where the size of the database has not been changed but the database has only been redistributed in the new configuration, in the capacity-growth measure (benchmark) the size of the database is both changed and redistributed in the new configuration. The change of the database size is deliberate in order to induce a change in the amount of responses to the same transaction. Obviously, if we are to induce, for example, twice the amount of the responses to a transaction in the new configuration over the amount of responses to the transaction in the old configuration, the size of the database in the new configuration is likely to be a multiple of (say, twice) the size of the database in the old configuration. To compensate for the increase in the database size and the response-set size, the new configuration is given a corresponding increase in number of backends and their disk systems. For this example, if the database size and the response-set size are doubled, then the number of backends and disks in the new configuration would be doubled. Therefore, what we want to measure (benchmark) is the invariance of the response time of the same transaction as the size of response sets and the number of backends increase in the same proportions. We characterize the capacity-growth measure (benchmark) as the *response-time invariance* and present the formula in Equation 2 below.

$$\begin{array}{c}\textit{The}\\ \textit{Response-Time}\\ \textit{Invariance}\end{array} = \left( \frac{\begin{array}{c}\textit{The Response}\\ \textit{Time of}\\ \textit{Configuration } Z\end{array}}{\begin{array}{c}\textit{The Response}\\ \textit{Time of}\\ \textit{Configuration } X\end{array}} \right) - 1$$

Equation 2. The Response-Time-Invariance Formula

Again, let $x$, the number of backends for configuration X, be 20, and $z$, the number of backends for configuration Z, be 60, then ideally we would like the ratio of response times of configuration Z of 60 backends and configuration X of 20 backends to be 1. Consequently, the response-time invariance of the 60-backend database computer over the 20-backend database computer would be 0, i.e., no variance. Of course, in this example, the transaction receives three times more responses in the 60-backend database computer than the response to the same transaction in the 20-backend database computer.

This example illustrates that if we triple the number of backends of an existing database computer for the grown and redistributed database, we would expect to maintain the same response time of the transaction despite the fact that the transaction is now processing three times more responses than before. For the purpose of maintaining the same response time of a transaction, it is ideal if the number of backends added to the existing configuration is in proportion to the increase in the amount of the responses. In this example, we need to add three times as many new backends and their disk systems to the existing database computer in order to hold the response time invariant.

In reality, some variances in response times will always exist. The issue is therefore how close a given multiple-backend database computer can maintain its ideal response-time invariances. As shown in Figure 6, we must measure (benchmark) a sufficient number of configurations for a given transaction and a similar number of databases in order to determine the system overheads and their impact of the response times.

Ideally, we would expect that in Figure 6 $S_i = i$ for $i = 1, 2, \ldots, n$ and for large $n$. However, in typical cases, $\delta_1 = 0$ and $\delta_2 < \delta_3 < \cdots < \delta_n$, where $\delta_i$ is the system overhead incurred in handling the transaction in the $i$-backend configuration. In studying Figure 6, we may expect the benchmarking effort to address the following issues:

(1) What are the values of $\delta_i$ for the given $i$-backend computers under benchmarking?

(2) How large will $n$ be when there is no invariance in response time (i.e., $S_n \geq S_{n+1}$)?

(3) How large will $n$ be when the system overhead becomes pronounces (i.e., $\frac{\delta_{n+1}}{n+1} \gg \frac{\delta_n}{n}$)?

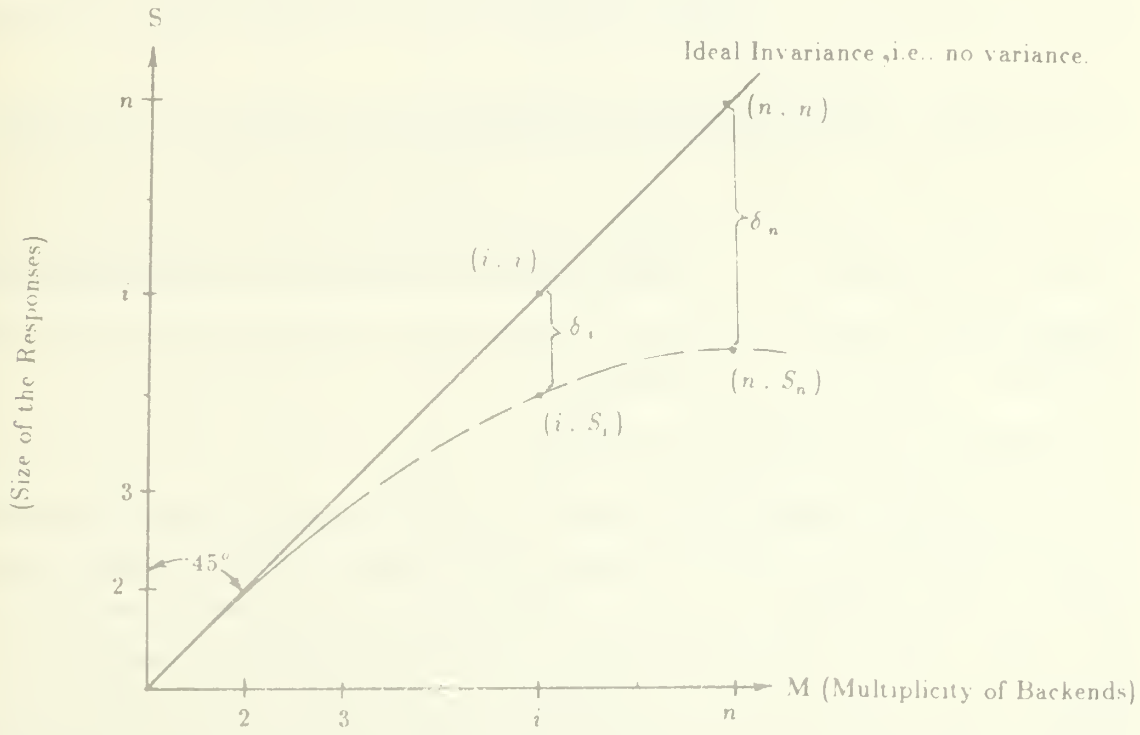

Figure 6 The Response-Time Invariance Measure

## 3.3. The Test (Benchmark) Transactions and the Test (Benchmark) Databases

The aforementioned measures on performance gains and capacity growth have focused on measuring (benchmarking) one transaction at a time. In other words, the same transaction is used to measure (benchmark) the response-time reductions and invariances over a large number of computer configurations. To provide a comprehensive measure of the multiple-backend database computer, we should use a number of 'standard' transactions (benchmarks). Among the primary operations of the database computer, the retrieve, delete and update operations tend to involve all of the backends of the computer, whereas the insert operation does not. Consequently, we must emphasize test transactions (benchmarks) which consist of various retrieve, delete and update operations. In Section 5, we propose a methodology for generating test-transaction (benchmark) mixes involving retrieve, delete, insert and update operations.

The test (benchmark) database for the measures must be distributed 'evenly' on the disks of a number of configurations with different numbers of backends. Consequently, we need a methodology not only to generate test (benchmark) databases but to allow even distribution and redistribution of the same test (benchmark) databases on the new configurations. Furthermore, each test (benchmark) database must induce just the right amount of increase in responses for the test-transaction (benchmark) mix for every new configuration when measuring (benchmarking) response-time invariance. This methodology is expounded in the following section.

## 4. THE DESIGN AND GENERATION OF TEST(BENCHMARK) DATABASES

Let us consider the possible configurations for a database computer with $m$ backends. Let $s$ denote the total number of bytes in the database of the one-backend database computer. Depending on the configuration being chosen, we would like to benchmark the $m$-backend computer against the $i$-backend computer where $i = 1, 2, \ldots, or\ (m - 1)$. Furthermore, we would want to evenly distribute the database of size $s$ to 1, 2, 3, ..., *or* $m$ backends.

### 4.1. The Concept of Database-Size Multiples

To determine a database size which permits an even distribution and redistribution of data to each backend in the configuration, we find the *least common multiple* (LCM) for the possible configurations of 1, 2, 3, 4..., *or* $m$ backends. For example, consider the case where up to four backends are used. The four possible computer configurations are for 1, 2, 3, and 4 backends. To enable us to allocate the database in $s / 1$, $s / 2$, $s / 3$, and $s / 4$ increments, the database size must be a multiple of 12, i.e., the LCM{1, 2, 3, 4}. If we select a database with 24,984 200-byte records for the one-backend database computer (i.e., a total of 4.8 megabytes), the configurations listed in Table 1 are possible. We may measure the performance of a one-backend computer. Then, we distribute the database evenly on the disk systems of a two-backend computer, three-backend computer, and four-backend computer, and measure the performance for each configuration. The distribution of data in bytes for the four configurations is also given in Table 1. An analysis of data for this series of tests may produce a graph similar to Figure 5. Table 2 summarizes the method for determining $s$, the database size, for a computer with $m$ backends.

We note that the expression for calculating the common database-size multiple requires a factor of 32. The need for this factor is explained later when we consider the record sizes of the database in Section 4.5.

| Number of Backends | Number of Megabytes per Backend |
|---|---|
| 1 | 4.8 |
| 2 | 2.4 |
| 3 | 1.6 |
| 4 | 1.2 |

Table 1. Sample Computer Configurations.

| Number of Backends in the Computer | With rec_size expressed in bytes, $s$ is a multiple of |
|---|---|
| 1 | ( 2 x 32 x rec_size) |
| 2 | ( 2 x 32 x rec_size) |
| 3 | ( 6 x 32 x rec_size) |
| 4 | ( 12 x 32 x rec_size) |
| 5 | ( 60 x 32 x rec_size) |
| 6 | ( 60 x 32 x rec_size) |
| 7 | (420 x 32 x rec_size) |
| . . . | . . . |
| $m$ | (LCM{1,2,...,$m$} x 32 x rec_size) |

Table 2. Database-Size Multiples.

## 4.2. The Determination of the Possible Test Configurations

Using $s$ as determined in Table 2, we can easily summarize the database-size requirements for conducting various performance-gains and capacity-growth measurements (benchmarks). Depicted in Table 3, for example, if we benchmark a three-backend computer and a one-backend computer for the performance-gains measure, we must configure the computer first with all of the database on one backend, then with the database distributed evenly on two backends, and finally with the database distributed evenly on three backends. For the capacity-growth measure, we benchmark first all of the database on one backend, then double the size of the database and distribute it evenly on two backends, and finally triple the size of the database and distribute it evenly on three backends.

In general, when we have a database computer which is expandable to a maximum of $m$ backends, then the number of benchmark configurations for performance gains in terms of response-time reductions is $m$, and the number of possible benchmark configurations for capacity growth in terms of response-time invariances is $(m - 1)$, thereby making the total number of distinct benchmark configurations to be $(m + (m - 1))$, i.e., $(2m - 1)$. Using this methodology, a system evaluator may select certain distinct test configurations for the performance-gains and capacity-growth measurements of a database computer with any number of backends.

| Configuration Number | Number of Backends | Megabytes per Backend | Total database Size in Megabytes |
|---|---|---|---|
| 1 | 1 | $s$ | $s$ |
| 2 | 2 | $s/2$ | $s$ |
| 3 | 3 | $s/3$ | $s$ |
| 4 | 2 | $s$ | $2s$ |
| 5 | 3 | $s$ | $3s$ |
| Note:<br>Configurations 1, 2 and 3 are required to benchmark the performance gains.<br>Configurations 1, 4 and 5 are required to benchmark the capacity growth. | | | |

Table 3. Test Configurations with One to Three Backends

## 4.3. The Choice of Database Sizes and Record Sizes

Next, we consider how to determine the database size, $s$. More importantly, the database-size multiple of $s$. To adequately measure the performance characteristics of a multiple-backend database computer, we propose that three different database sizes be selected. One size should represent a small database, one size should represent a large database, and the third should represent an intermediate size between the largest and smallest ones. We decide that the smallest database size is $s$ / 4, while the intermediate size is $s$ / 2.

The database sizes are dependent on the disks used. Therefore, we propose the following methodology which may be easily applied to any disk organization. First, the largest database size is proportional to the maximum formated capacity, in megabytes, of the backend's disk. For example, assume that for a three-backend database computer, each backend has a single disk drive with a maximum formated capacity of 300 megabytes for the database use*. From Table 2 we see that $s$ must be divisible by (6 x 32 x rec_size). Although we have yet to consider the record size, this requirement implies that the largest database must be divisible by 6 x 32, i.e., 192.

Now we consider *record size* before selecting the final value for $s$. Strawser notes that record-size selection is also hardware specific, since it depends on the size of the unit of data accessed by the particular computer [6]. For example, suppose the disk-track size is 4 kilobytes. Using Strawser's scheme for blocking records of four sizes into a 4-kilobyte track, we may select sizes of 2000, 1000, 400, and 200 bytes per record, resulting in a range of 2 to 20 records per track. For a computer which supports a 16-kilobyte track size, we may select record sizes of 4000, 2000, 800, and 400 bytes per record, which results in a range of 4 to 40 records per track.

The key to record-size selection is to ensure that one record size is large and one small, with the other two record sizes representing intermediate values between the largest and smallest values picked. This will enable us to contrast performance for cases where there are many small records per track to cases where there are a few large records per track. In addition, we require that the three smaller record sizes be evenly divisible into the largest record size, since this simplifies the process of determining

*In this study, the Fujitsu Eagle disk drive is used. Out of the 380-megabyte formated capacity, 80 megabytes are reserved for use by the directory. (See Section 4.7 on directory data.) The remaining 300 megabytes are used for the database.

database size. With this requirement, we may concentrate on sizing the database for the largest record size, and be assured that the selected database will accommodate the smaller record sizes as well. Since track sizes differ for various disk installations, each system evaluator may determine unique record sizes which will be compatible with the specific unit of data access and storage.

Assume that we decide to use 4-kilobyte tracks, with record sizes of 2000, 1000, 400, and 200 bytes per record. We use this assumption to continue the development of test databases for a three-backend database computer.

## 4.4. The Calculation of the Database-Size Multiple

We can now determine the required database-size multiple for our sample application as follows:

$$(6 \times 32 \times 2000) = 384{,}000.$$

Therefore, $s$ will be the largest multiple of 384,000 bytes for a maximum formated database capacity of 300 megabytes. For simplicity, let a million bytes be a megabyte. Since 781 x 384,000 = 299.904 megabytes, we have

$$s / 4 = 74.976 \text{ megabytes},$$
$$s / 2 = 149.952 \text{ megabytes, and}$$
$$s = 299.904 \text{ megabytes}.$$

In other words, the large database size, $s$, is 781 multiples of 384,000 bytes.

Tables 4, 5 and 6 show that for our sample the three proposed test databases are feasible, since they permit each database to be distributed evenly as required for all of the feasible test configurations.

| Configuration Number | Number of Backends | Megabytes per Backend | Total database Size in Megabytes |
|---|---|---|---|
| 1 | 1 | 74.976 | 74.976 |
| 2 | 2 | 37.488 | 74.976 |
| 3 | 3 | 24.992 | 74.976 |
| 4 | 2 | 74.976 | 149.952 |
| 5 | 3 | 74.976 | 299.904 |

Note:
Configurations 1, 2 and 3 are required to benchmark the performance gains.
Configurations 1, 4 and 5 are required to benchmark the capacity growth.

Table 4. Test Configurations for the Three-Backend Computer with Small Databases ($s$ / 4 = 74.976 megabytes).

| Configuration Number | Number of Backends | Megabytes per Backend | Total database Size in Megabytes |
|---|---|---|---|
| 1 | 1 | 149.952 | 149.952 |
| 2 | 2 | 74.976 | 149.952 |
| 3 | 3 | 49.984 | 149.952 |
| 4 | 2 | 149.952 | 299.904 |
| 5 | 3 | 149.952 | 449.856 |

Note:
Configurations 1, 2 and 3 are required to benchmark the performance gains.
Configurations 1, 4 and 5 are required to benchmark the capacity growth.

Table 5. Test Configurations for the Three-Backend Computer with Medium Databases ($s$ / 2 = 149.952 megabytes).

| Configuration Number | Number of Backends | Megabytes per Backend | Total database Size in Megabytes |
|---|---|---|---|
| 1 | 1 | 299.904 | 299.904 |
| 2 | 2 | 149.942 | 299.904 |
| 3 | 3 | 99.968 | 299.904 |
| 4 | 2 | 299.904 | 599.808 |
| 5 | 3 | 299.904 | 899.712 |

Note:
Configurations 1, 2 and 3 are required to benchmark the performance gains.
Configurations 1, 4 and 5 are required to benchmark the capacity growth.

Table 6. Test Configurations for the Three-Backend Computer with Large Databases ($s$ = 299.904 megabytes).

## 4.5. The Consideration of Database Formats

Next, we consider how to format the test databases in terms of record sizes. Two options seem feasible. We may use only one record size per database, or we may include all four record sizes in a database. Consider the case where we use only one record size per database. As we have four record sizes, we must create four separate databases, i.e., one for each record size. Further, we also want to test with small, medium and large databases. We therefore have 12 (i.e., 3 x 4) different database configurations to be used for testing. Since there are 5 possible computer configurations for a three-backend database computer, the measurement tests may run as high as 60 times, i.e., 12 x 5. In addition, there is a separate mix of test transactions for each record size and database size. Multiply this number with the number of test-transaction mixes, and the resulting number of tests may be unreasonably large.

Consider the case where four different record sizes appear in a single database. In this case, we require just three test databases instead of twelve, since each database contains four record sizes. This database configuration may be easier to use for testing, since only 15 sets (i.e., 3 x 5) of measurement tests need be run. Each mix of test transactions is larger, since it includes transactions for testing all four record sizes. However, the size of responses for the test-transaction mix is smaller. As records of all four sizes are distributed over the available secondary storage, fewer records per record size are stored. Because the available secondary storage is now shared by records of four different sizes, we must consider how to distribute the records of different sizes. One option would be to use an equal number of records per record size. The disadvantage of this approach is that the resultant database distribution is not even.

An even distribution would be to split the database into four equal quarters, with each quarter of the database corresponding to one of the four record-size categories. We apply this technique to our example of the three-backend database computer. First, consider the small database of 74.976 megabytes, i.e., $s / 4$. Then, there are 18.744 megabytes per quarter. Therefore, we have for four record sizes:

(18.744 megabytes)/(2000 bytes/record) = 9,372 records

(18.744 megabytes)/(1000 bytes/record) = 18,744 records

(18.744 megabytes)/(400 bytes/record) = 46,860 records

(18.744 megabytes)/(200 bytes/record) = 93,720 records

Following through with similar calculations for $s / 2$ and $s$, we can derive Tables 7, 8 and 9 for small, medium, and large databases consisting of four record sizes in equal quarters per database. We see from these tables that our database design permits each database to be distributed evenly as required for all of the possible test configurations.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Megabytes per Backend | Database Size in Megabytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 9,372 | 18.744 | 74.976 |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | |
| | | 200 | 93,720 | 18.744 | |
| 2 | 2 | 2000 | 4,686 | 9.372 | 74.976 |
| | | 1000 | 9,372 | 9.372 | |
| | | 400 | 23,430 | 9.372 | |
| | | 200 | 46,860 | 9.372 | |
| 3 | 3 | 2000 | 3,124 | 6.248 | 74.976 |
| | | 1000 | 6,248 | 6.248 | |
| | | 400 | 15,620 | 6.248 | |
| | | 200 | 31,240 | 6.248 | |
| 4 | 2 | 2000 | 9,372 | 18.744 | 149.952 |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | |
| | | 200 | 93,720 | 18.744 | |
| 5 | 3 | 2000 | 9,372 | 18.744 | 224.928 |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | |
| | | 200 | 93,720 | 18.744 | |

Table 7. Small Test Databases for Different Configurations.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Megabytes per Backend | Database Size in Megabytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 18.744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 149.952 |
| | | 200 | 187,440 | 37.488 | |
| 2 | 2 | 2000 | 9,372 | 18.744 | |
| | | 1000 | 18,744 | 18.744 | |
| | | 400 | 46,860 | 18.744 | 149.952 |
| | | 200 | 93,720 | 18.744 | |
| 3 | 3 | 2000 | 6,248 | 12.496 | |
| | | 1000 | 12,496 | 12.496 | |
| | | 400 | 31,240 | 12.496 | 149.952 |
| | | 200 | 62,480 | 12.496 | |
| 4 | 2 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 299.904 |
| | | 200 | 187,440 | 37.488 | |
| 5 | 3 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37.488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 449.856 |
| | | 200 | 187,440 | 37.488 | |

Table 8. Medium Test Databases for Different Configurations.

| Configuration Number | Number of Backends | Record Size in Bytes | Number of Records per Backend | Megabytes per Backend | Database Size in Megabytes |
|---|---|---|---|---|---|
| 1 | 1 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 299.904 |
| | | 200 | 374,880 | 74.976 | |
| 2 | 2 | 2000 | 18,744 | 37.488 | |
| | | 1000 | 37,488 | 37.488 | |
| | | 400 | 93,720 | 37.488 | 299.904 |
| | | 200 | 187,440 | 37.488 | |
| 3 | 3 | 2000 | 12,496 | 24.992 | |
| | | 1000 | 24,992 | 24.992 | |
| | | 400 | 62,480 | 24.992 | 299.904 |
| | | 200 | 124,960 | 24.992 | |
| 4 | 2 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 599.808 |
| | | 200 | 374,880 | 74.976 | |
| 5 | 3 | 2000 | 37,488 | 74.976 | |
| | | 1000 | 74,976 | 74.976 | |
| | | 400 | 187,440 | 74.976 | 899.712 |
| | | 200 | 374,880 | 74.976 | |

Table 9. Large Test Databases for Different Configurations.

We may now explain the requirement for the multiple of 32 in the database-size relation of Table 2. First, recall that, in general, $s$ is a multiple of (LCM{1,2,...,M} x 32 x rec_size). In our methodology for selecting a small, medium, or large database, we decided to select database-size increments of $s / 4$, $s / 2$, and $s$. Thus, $s$ is a multiple of 1, 2, and 4. Since the LCM{1,2,4} is 4, then $s$ must be divisible by 4. Secondly, to enable us to handle the four record sizes in a single database, we must be able to split the database into four even quarters. In the case of the small-size database, $s / 4$ must be divisible by 4 again to yield even quarters. Since, $(s / 4)/ 4$ is the same as $s / 16$, the effect is to require that the total database size, $s$, be divisible by 16. Finally, we require that the small database size represented by $s / 16$ be further divisible by 2. This final requirement is actually related to the partitioning (clustering) mechanism of the multiple-backend database computers which groups records into partitions (clusters). By requiring that the database be divisible by this final factor of 2, we make it possible for each partition (cluster) to hold an even number of records. Thus, we can control the size of partitions (clusters), say, forming a new partition (cluster) of one half of the records of an existing partition (cluster). Although this factor is not a general requirement for a partitioning (clustering) mechanism, and can be eliminated without any loss of generality in our methodology, there will be less work in conducting the benchmarking experiments (i.e., controlling the partition (cluster) size), and in interpreting the benchmarking results. Therefore, we have $(s / 16)/ 2$, which means that $s$ must be a multiple of 32 times the LCM{1,2,...,M} times the record size.

## 4.6. The Formation of Partitions (Clusters)

Although partitioning (clustering) schemes are different in different multiple-backend database computers, they all generate partitions (clusters) with variable numbers of records. The number of records per partition (cluster) is determined by the indices used, which will be elaborated on in a later section. Here we propose a way to form variable-size partitions (clusters) for even distribution. We have selected nine partition (cluster) categories, with each partition (cluster) containing from 2 to 10 blocks of records. This design provides a uniform range of cluster sizes and facilitates the design of extensible and versatile test-transaction mixes. For example, the cluster category with two blocks per cluster has four 2000-byte records per cluster, eight 1000-byte records per cluster, twenty 400-byte records per cluster, and forty 200-byte records per cluster. These values are calculated by multiplying the number of records per block by the number of blocks per cluster. Thus, there are nine categories of clusters and their records per cluster are depicted in Table 10.

Our next consideration is to determine how many partitions (clusters) of each partition (cluster) category should be chosen for each of the four record sizes comprising a test database. Let us return to our three-backend computer configuration, and integrate the data on clusters, records, blocks and others for the small test database, with 74.976 megabytes, i.e., of $s / 4$

Configuration 1 of Table 7 shows that we have 9,372 records for the 2000-byte record size. We wish to distribute these records according to the nine cluster categories of Table 10. Let us consider a simple illustration. We use the nine cluster categories and the corresponding values of the number of records per cluster category for the 2000-byte record size. We again assume a three-backend computer, with four clusters for each of the cluster categories. This results in the distribution shown in Table 11.

| Cluster Category | Blocks per Cluster | Record Size in Bytes: | | | |
|---|---|---|---|---|---|
| | | 2000 | 1000 | 400 | 200 |
| 1 | 2 | 4 | 8 | 20 | 40 |
| 2 | 3 | 6 | 12 | 30 | 60 |
| 3 | 4 | 8 | 16 | 40 | 80 |
| 4 | 5 | 10 | 20 | 50 | 100 |
| 5 | 6 | 12 | 24 | 60 | 120 |
| 6 | 7 | 14 | 28 | 70 | 140 |
| 7 | 8 | 16 | 32 | 80 | 160 |
| 8 | 9 | 18 | 36 | 90 | 180 |
| 9 | 10 | 20 | 40 | 100 | 200 |

Table 10. Number of Records per Cluster Category.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks |
|---|---|---|---|---|---|
| 2,000 | 2 | 4 | 4 | 16 | 8 |
| | 3 | 6 | 4 | 24 | 12 |
| | 4 | 8 | 4 | 32 | 16 |
| | 5 | 10 | 4 | 40 | 20 |
| | 6 | 12 | 4 | 48 | 24 |
| | 7 | 14 | 4 | 56 | 28 |
| | 8 | 16 | 4 | 64 | 32 |
| | 9 | 18 | 4 | 72 | 36 |
| | 10 | 20 | 4 | 80 | 40 |
| Sub-totals: | | | 36 | 432 | 216 |

Table 11. Sample Record Distribution.

Given the distribution of Table 11, we see that the database computer distributes the blocks across three backends to effect an even record distribution. The first cluster category consists of two blocks per cluster, for four clusters, resulting in a total of eight blocks to be distributed across three backends. Since eight is not evenly divisible by three, two backends will receive three blocks of records, while one backend will receive two blocks of records. The database computer distributes the blocks for the rest of the clusters in a similar fashion. Table 12 shows the block and record distribution for this example.

Notice in Table 12 that during block distribution the database computer ensures that each backend ends up with an equal number of blocks. We observe that Backend 3 has received one less block for the first cluster category. During distribution of the blocks for the third cluster category, the database computer has compensated it by inserting six blocks at Backend 3, while inserting only five blocks each at Backends 1 and 2. The same situation occurs between cluster categories four and six, and between categories seven and nine of Table 12. Although it is not possible for the the database computer to distribute blocks equally for *every* individual cluster, it does work to achieve an equal distribution in the long run for the *entire* cluster collection.

| Cluster Category | Number of Blocks per Cluster | Backend 1 | | Backend 2 | | Backend 3 | |
|---|---|---|---|---|---|---|---|
| | | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 1 | 2 | 3 | 6 | 3 | 6 | 2 | 4 |
| 2 | 3 | 4 | 8 | 4 | 8 | 4 | 8 |
| 3 | 4 | 5 | 10 | 5 | 10 | 6 | 12 |
| 4 | 5 | 7 | 14 | 7 | 14 | 6 | 12 |
| 5 | 6 | 8 | 16 | 8 | 16 | 8 | 16 |
| 6 | 7 | 9 | 18 | 9 | 18 | 10 | 20 |
| 7 | 8 | 11 | 22 | 11 | 22 | 10 | 20 |
| 8 | 9 | 12 | 24 | 12 | 24 | 12 | 24 |
| 9 | 10 | 13 | 26 | 13 | 26 | 14 | 28 |
| Sub-totals: | | 72 | 144 | 72 | 144 | 72 | 144 |
| Note: (72 x 3) = 216 blocks. | | | | Note: (144 x 3) = 432 records. | | | |

Table 12. Record/Block Distribution for Table 11 Example.

With this understanding of the cluster distribution process, let us return to the task of determining the required number of clusters for a total of 9,372 2000-byte records. If we sum all of the number of records per cluster, we have 108 records distributed over all nine cluster categories. We therefore simply divide 9,372 by 108. The result is 86, with a remainder of 84. This means that we are 24, (108 - 84), records short of being able to use 87 clusters for each of the 9 cluster categories. This deficit is easily resolved by using 86 clusters for the first and last cluster categories, since 4 + 20 = 24. The other seven categories will each have 87 clusters.

We may use the same computation to arrive at the record and block distributions of the 200-byte, 400-byte, and 1000-byte record sizes for Configuration 1 of Table 7, since 200, 400, and 1000 are all divisors of 2000. The resulting cluster distribution is also shown in Table 13.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1,000 | 2 | 8 | 86 | 688 | 172 | 172 |
| | 3 | 12 | 87 | 1,044 | 261 | 261 |
| | 4 | 16 | 87 | 1,392 | 348 | 348 |
| | 5 | 20 | 87 | 1,740 | 435 | 435 |
| | 6 | 24 | 87 | 2,088 | 522 | 522 |
| | 7 | 28 | 87 | 2,436 | 609 | 609 |
| | 8 | 32 | 87 | 2,784 | 696 | 696 |
| | 9 | 36 | 87 | 3,132 | 783 | 783 |
| | 10 | 40 | 86 | 3,440 | 860 | 860 |
| Sub-totals: | | | 781 | 18,744 | 4,686 | 4,686 |
| 400 | 2 | 20 | 86 | 1,720 | 172 | 172 |
| | 3 | 30 | 87 | 2,610 | 261 | 261 |
| | 4 | 40 | 87 | 3,480 | 348 | 348 |
| | 5 | 50 | 87 | 4,350 | 435 | 435 |
| | 6 | 60 | 87 | 5,220 | 522 | 522 |
| | 7 | 70 | 87 | 6,090 | 609 | 609 |
| | 8 | 80 | 87 | 6,960 | 696 | 696 |
| | 9 | 90 | 87 | 7,830 | 783 | 783 |
| | 10 | 100 | 86 | 8,600 | 860 | 860 |
| Sub-totals: | | | 781 | 46,860 | 4,686 | 4,686 |
| 200 | 2 | 40 | 86 | 3,440 | 172 | 172 |
| | 3 | 60 | 87 | 5,220 | 261 | 261 |
| | 4 | 80 | 87 | 6,960 | 348 | 348 |
| | 5 | 100 | 87 | 8,700 | 435 | 435 |
| | 6 | 120 | 87 | 10,440 | 522 | 522 |
| | 7 | 140 | 87 | 12,180 | 609 | 609 |
| | 8 | 160 | 87 | 13,920 | 696 | 696 |
| | 9 | 180 | 87 | 15,660 | 783 | 783 |
| | 10 | 200 | 86 | 17,200 | 860 | 860 |
| Sub-totals: | | | 781 | 93,720 | 4,686 | 4,686 |

Table 13. Record/Block Distribution. Small Database. Configuration 1.

There are two backends in configuration 2. three backends in configuration 3. two again in configuration 4 and three again in configuration 5. Each of there configurations have different record block distributions for the same database. The following seven tables are for the remaining four configurations. We do not include the eight tables for the medium database size of the five configurations. Nor do we include the eight tables for the large database size. The interested reader may refer to 7 for the tables.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Backend 1 | Backend 2 |
|---|---|---|---|---|---|---|---|
| | | | | | | Number of Blocks | Number of Blocks |
| 2000 | 2 | 4 | 86 | 344 | 172 | 86 | 86 |
| | 3 | 6 | 87 | 522 | 261 | 131 | 130 |
| | 4 | 8 | 87 | 696 | 348 | 174 | 174 |
| | 5 | 10 | 87 | 870 | 435 | 217 | 218 |
| | 6 | 12 | 87 | 1,044 | 522 | 261 | 261 |
| | 7 | 14 | 87 | 1,218 | 609 | 305 | 304 |
| | 8 | 16 | 87 | 1,392 | 696 | 348 | 348 |
| | 9 | 18 | 87 | 1,566 | 783 | 391 | 392 |
| | 10 | 20 | 86 | 1,720 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 9,372 | 4,686 | 2,343 | 2,343 |
| 1000 | 2 | 8 | 86 | 688 | 172 | 86 | 86 |
| | 3 | 12 | 87 | 1,044 | 261 | 131 | 130 |
| | 4 | 16 | 87 | 1,392 | 348 | 174 | 174 |
| | 5 | 20 | 87 | 1,740 | 435 | 217 | 218 |
| | 6 | 24 | 87 | 2,088 | 522 | 261 | 261 |
| | 7 | 28 | 87 | 2,436 | 609 | 305 | 304 |
| | 8 | 32 | 87 | 2,784 | 696 | 348 | 348 |
| | 9 | 36 | 87 | 3,132 | 783 | 391 | 392 |
| | 10 | 40 | 86 | 3,440 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 18,744 | 4,686 | 2,343 | 2,343 |
| 400 | 2 | 20 | 86 | 1,720 | 172 | 86 | 86 |
| | 3 | 30 | 87 | 2,610 | 261 | 131 | 130 |
| | 4 | 40 | 87 | 3,480 | 348 | 174 | 174 |
| | 5 | 50 | 87 | 4,350 | 435 | 217 | 218 |
| | 6 | 60 | 87 | 5,220 | 522 | 261 | 261 |
| | 7 | 70 | 87 | 6,090 | 609 | 305 | 304 |
| | 8 | 80 | 87 | 6,960 | 696 | 348 | 348 |
| | 9 | 90 | 87 | 7,830 | 783 | 391 | 392 |
| | 10 | 100 | 86 | 8,600 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 46,860 | 4,686 | 2,343 | 2,343 |
| 200 | 2 | 40 | 86 | 3,440 | 172 | 86 | 86 |
| | 3 | 60 | 87 | 5.220 | 261 | 131 | 130 |
| | 4 | 80 | 87 | 6,960 | 348 | 174 | 174 |
| | 5 | 100 | 87 | 8,700 | 435 | 217 | 218 |
| | 6 | 120 | 87 | 10,440 | 522 | 261 | 261 |
| | 7 | 140 | 87 | 12,180 | 609 | 305 | 304 |
| | 8 | 160 | 87 | 13,920 | 696 | 348 | 348 |
| | 9 | 180 | 87 | 15,660 | 783 | 391 | 392 |
| | 10 | 200 | 86 | 17,200 | 860 | 430 | 430 |
| Sub-totals: | | | 781 | 93,720 | 4,686 | 2.343 | 2.343 |

Table 14. Record Block Distribution, Small Database, Configuration 2.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2,000 | 2 | 4 | 86 | 344 | 172 | |
| | 3 | 6 | 87 | 522 | 261 | |
| | 4 | 8 | 87 | 696 | 348 | (See below) |
| | 5 | 10 | 87 | 870 | 435 | |
| | 6 | 12 | 87 | 1,044 | 522 | |
| | 7 | 14 | 87 | 1,218 | 609 | |
| | 8 | 16 | 87 | 1,392 | 696 | |
| | 9 | 18 | 87 | 1,566 | 783 | |
| | 10 | 20 | 86 | 1,720 | 860 | |
| Sub-totals: | | | 781 | 9,372 | 4,686 | |

| Number of Blocks per Cluster | Backend 1 | | Backend 2 | | Backend 3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 116 | 57 | 114 | 57 | 114 |
| 3 | 87 | 174 | 87 | 174 | 87 | 174 |
| 4 | 116 | 232 | 116 | 232 | 116 | 232 |
| 5 | 145 | 290 | 145 | 290 | 145 | 290 |
| 6 | 174 | 348 | 174 | 348 | 174 | 348 |
| 7 | 203 | 406 | 203 | 406 | 203 | 406 |
| 8 | 232 | 464 | 232 | 464 | 232 | 464 |
| 9 | 261 | 522 | 261 | 522 | 261 | 522 |
| 10 | 286 | 572 | 287 | 574 | 287 | 574 |
| Sub-totals | 1,562 | 3,124 | 1,562 | 3,124 | 1,562 | 3,124 |

Table 15a. Record/Block Distribution, Small Database, Configuration 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 1,000 | 2 | 8 | 86 | 688 | 172 | |
| | 3 | 12 | 87 | 1,044 | 261 | |
| | 4 | 16 | 87 | 1,392 | 348 | (See below) |
| | 5 | 20 | 87 | 1,740 | 435 | |
| | 6 | 24 | 87 | 2,088 | 522 | |
| | 7 | 28 | 87 | 2,436 | 609 | |
| | 8 | 32 | 87 | 2,784 | 696 | |
| | 9 | 36 | 87 | 3,132 | 783 | |
| | 10 | 40 | 86 | 3,440 | 860 | |
| Sub-totals: | | | 781 | 18,744 | 4,686 | |

| Number of Blocks per Cluster | Backend 1 | | Backend 2 | | Backend 3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 232 | 57 | 228 | 57 | 228 |
| 3 | 87 | 348 | 87 | 348 | 87 | 348 |
| 4 | 116 | 464 | 116 | 464 | 116 | 464 |
| 5 | 145 | 580 | 145 | 580 | 145 | 580 |
| 6 | 174 | 696 | 174 | 696 | 174 | 696 |
| 7 | 203 | 812 | 203 | 812 | 203 | 812 |
| 8 | 232 | 928 | 232 | 928 | 232 | 928 |
| 9 | 261 | 1,044 | 261 | 1,044 | 261 | 1,044 |
| 10 | 286 | 1,144 | 287 | 1,148 | 287 | 1,148 |
| Sub-totals: | 1,562 | 6,248 | 1,562 | 6,248 | 1,562 | 6,248 |

Table 15b. Record/Block Distribution, Small Database, Configuration 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 400 | 2 | 20 | 86 | 1,720 | 172 | (See below) |
| | 3 | 30 | 87 | 2,610 | 261 | |
| | 4 | 40 | 87 | 3,480 | 348 | |
| | 5 | 50 | 87 | 4,350 | 435 | |
| | 6 | 60 | 87 | 5,220 | 522 | |
| | 7 | 70 | 87 | 6,090 | 609 | |
| | 8 | 80 | 87 | 6,960 | 696 | |
| | 9 | 90 | 87 | 7,830 | 783 | |
| | 10 | 100 | 86 | 8,600 | 860 | |
| Sub-totals: | | | 781 | 46,860 | 4,686 | |

| Number of Blocks per Cluster | Backend 1 | | Backend 2 | | Backend 3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 580 | 57 | 570 | 57 | 570 |
| 3 | 87 | 870 | 87 | 870 | 87 | 870 |
| 4 | 116 | 1,160 | 116 | 1,160 | 116 | 1,160 |
| 5 | 145 | 1,450 | 145 | 1,450 | 145 | 1,450 |
| 6 | 174 | 1,740 | 174 | 1,740 | 174 | 1,740 |
| 7 | 203 | 2,030 | 203 | 2,030 | 203 | 2,030 |
| 8 | 232 | 2,320 | 232 | 2,320 | 232 | 2,320 |
| 9 | 261 | 2,610 | 261 | 2,610 | 261 | 2,610 |
| 10 | 286 | 2,860 | 287 | 2,860 | 287 | 2,860 |
| Sub-totals: | 1,562 | 15,620 | 1,562 | 15,620 | 1,562 | 15,620 |

Table 15c. Record/Block Distribution, Small Database, Configuration 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 200 | 2 | 40 | 86 | 3,440 | 172 | (See below) |
| | 3 | 60 | 87 | 5,220 | 261 | |
| | 4 | 80 | 87 | 6,960 | 348 | |
| | 5 | 100 | 87 | 8,700 | 435 | |
| | 6 | 120 | 87 | 10,440 | 522 | |
| | 7 | 140 | 87 | 12,180 | 609 | |
| | 8 | 160 | 87 | 13,920 | 696 | |
| | 9 | 180 | 87 | 15,660 | 783 | |
| | 10 | 200 | 86 | 17,200 | 860 | |
| Sub-totals: | | | 781 | 93,720 | 4,686 | |

| Number of Blocks per Cluster | Backend 1 | | Backend 2 | | Backend 3 | |
|---|---|---|---|---|---|---|
| | Number of Blocks | Number of Records | Number of Blocks | Number of Records | Number of Blocks | Number of Records |
| 2 | 58 | 1.160 | 57 | 1,140 | 57 | 1.140 |
| 3 | 87 | 1,740 | 87 | 1,740 | 87 | 1,740 |
| 4 | 116 | 2,320 | 116 | 2,320 | 116 | 2,320 |
| 5 | 145 | 2,900 | 145 | 2,900 | 145 | 2,900 |
| 6 | 174 | 3,480 | 174 | 3,480 | 174 | 3,480 |
| 7 | 203 | 4,060 | 203 | 4,060 | 203 | 4,060 |
| 8 | 232 | 4,640 | 232 | 4,640 | 232 | 4,640 |
| 9 | 261 | 5,220 | 261 | 5,220 | 261 | 5,220 |
| 10 | 286 | 5,720 | 287 | 5,740 | 287 | 5,740 |
| Sub-totals: | 1,562 | 31,240 | 1,562 | 31,240 | 1,562 | 31,240 |

Table 15d. Record Block Distribution, Small Database, Configuration 3.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 4 | 8 | 86 | 688 | 344 | 172 |
| | 6 | 12 | 87 | 1,044 | 522 | 261 |
| | 8 | 16 | 87 | 1,392 | 696 | 348 |
| | 10 | 20 | 87 | 1,740 | 870 | 435 |
| | 12 | 24 | 87 | 2,088 | 1,044 | 522 |
| | 14 | 28 | 87 | 2,436 | 1,218 | 609 |
| | 16 | 32 | 87 | 2,784 | 1,392 | 696 |
| | 18 | 36 | 87 | 3,132 | 1,566 | 783 |
| | 20 | 40 | 86 | 3,440 | 1,720 | 860 |
| Sub-totals: | | | 781 | 18,744 | 9,372 | 4,686 |
| 1000 | 4 | 16 | 86 | 1,376 | 344 | 172 |
| | 6 | 24 | 87 | 2,088 | 522 | 261 |
| | 8 | 32 | 87 | 2,784 | 696 | 348 |
| | 10 | 40 | 87 | 3,480 | 870 | 435 |
| | 12 | 48 | 87 | 4,176 | 1,044 | 522 |
| | 14 | 56 | 87 | 4,872 | 1,218 | 609 |
| | 16 | 64 | 87 | 5,568 | 1,392 | 696 |
| | 18 | 72 | 87 | 6,264 | 1,566 | 783 |
| | 20 | 80 | 86 | 6,880 | 1,720 | 860 |
| Sub-totals: | | | 781 | 37,488 | 9,372 | 4,686 |
| 400 | 4 | 40 | 86 | 3,440 | 344 | 172 |
| | 6 | 60 | 87 | 5,220 | 522 | 261 |
| | 8 | 80 | 87 | 6,960 | 696 | 348 |
| | 10 | 100 | 87 | 8,700 | 870 | 435 |
| | 12 | 120 | 87 | 10,440 | 1,044 | 522 |
| | 14 | 140 | 87 | 12,180 | 1,218 | 609 |
| | 16 | 160 | 87 | 13,920 | 1,392 | 696 |
| | 18 | 180 | 87 | 15,660 | 1,566 | 783 |
| | 20 | 200 | 86 | 17,200 | 1,720 | 860 |
| Sub-totals: | | | 781 | 93,720 | 9,372 | 4,686 |
| 2000 | 4 | 80 | 86 | 6,880 | 344 | 172 |
| | 6 | 120 | 87 | 10,440 | 522 | 261 |
| | 8 | 160 | 87 | 13,920 | 696 | 348 |
| | 10 | 200 | 87 | 17,400 | 870 | 435 |
| | 12 | 240 | 87 | 20,880 | 1,044 | 522 |
| | 14 | 280 | 87 | 24,360 | 1,218 | 609 |
| | 16 | 320 | 87 | 27,840 | 1,392 | 696 |
| | 18 | 360 | 87 | 31,320 | 1,566 | 783 |
| | 20 | 400 | 86 | 34,400 | 1,720 | 860 |
| Sub-totals: | | | 781 | 187,440 | 9,372 | 4,686 |

Table 16. Record/Block Distribution, Small Database, Configuration 4.

| Record Size in Bytes | Number of Blocks per Cluster | Number of Records per Cluster | Total Number of Clusters | Total Number of Records | Total Number of Blocks | Number of Blocks per Backend |
|---|---|---|---|---|---|---|
| 2000 | 6 | 12 | 86 | 1,032 | 516 | 172 |
| | 9 | 18 | 87 | 1,566 | 783 | 261 |
| | 12 | 24 | 87 | 2,088 | 1,044 | 348 |
| | 15 | 30 | 87 | 2,610 | 1,305 | 435 |
| | 18 | 36 | 87 | 3,132 | 1,566 | 522 |
| | 21 | 42 | 87 | 3,654 | 1,827 | 609 |
| | 24 | 48 | 87 | 4,176 | 2,088 | 696 |
| | 27 | 54 | 87 | 4,698 | 2,349 | 783 |
| | 30 | 60 | 86 | 5,160 | 2,580 | 860 |
| Sub-totals: | | | 781 | 28,116 | 14,058 | 4,686 |
| 1000 | 6 | 24 | 86 | 2,064 | 516 | 172 |
| | 9 | 36 | 87 | 3,132 | 783 | 261 |
| | 12 | 48 | 87 | 4,176 | 1,044 | 348 |
| | 15 | 60 | 87 | 5,220 | 1,305 | 435 |
| | 18 | 72 | 87 | 6,264 | 1,566 | 522 |
| | 21 | 84 | 87 | 7,308 | 1,827 | 609 |
| | 24 | 96 | 87 | 8,352 | 2,088 | 696 |
| | 27 | 108 | 87 | 9,396 | 2,349 | 783 |
| | 30 | 120 | 86 | 10,320 | 2,580 | 860 |
| Sub-totals: | | | 781 | 56,232 | 14,058 | 4,686 |
| 400 | 6 | 60 | 86 | 5,160 | 516 | 172 |
| | 9 | 90 | 87 | 7,830 | 783 | 261 |
| | 12 | 120 | 87 | 10,440 | 1,044 | 348 |
| | 15 | 150 | 87 | 13,050 | 1,305 | 435 |
| | 18 | 180 | 87 | 15,660 | 1,566 | 522 |
| | 21 | 210 | 87 | 18,270 | 1,827 | 609 |
| | 24 | 240 | 87 | 20,880 | 2,088 | 696 |
| | 27 | 270 | 87 | 23,490 | 2,349 | 783 |
| | 30 | 300 | 86 | 25,800 | 2,580 | 860 |
| Sub-totals: | | | 781 | 140,580 | 14,058 | 4,686 |
| 200 | 6 | 120 | 86 | 10,320 | 516 | 172 |
| | 9 | 180 | 87 | 15,660 | 783 | 261 |
| | 12 | 240 | 87 | 20,880 | 1,044 | 348 |
| | 15 | 300 | 87 | 26,100 | 1,305 | 435 |
| | 18 | 360 | 87 | 31,320 | 1,566 | 522 |
| | 21 | 420 | 87 | 36,540 | 1,827 | 609 |
| | 24 | 480 | 87 | 41,760 | 2,088 | 696 |
| | 27 | 540 | 87 | 46,980 | 2,349 | 783 |
| | 30 | 600 | 86 | 51,600 | 2,580 | 860 |
| Sub-totals: | | | 781 | 281,160 | 14,058 | 4,686 |

Table 17. Record Block Distribution, Small Database, Configuration 5.

## 4.7. The Definition of Record Templates and Directory Data

To complete the development of the test databases, we must specify the record templates for each of the four record sizes. A *record template* is the formal specification of the *directory* and *non-directory attributes* which make up the record structure and determine the intended attribute-value ranges and attribute values (for short, *descriptors*) of directory attributes. Often, these descriptors are also known as *indices*. Since the four record sizes we have chosen are all divisible by 10, we set the attribute size to 10-

bytes per attribute. Thus, the number of attributes per record is one tenth of the record size.

We specify the record templates for each record class in Table 18. For the four record templates, the attributes TEMPLATE, INT2001, INT1001, INT401, INT201, INT2002, INT1002, INT402, and INT202 are directory attributes, while the remaining attributes of each template are non-directory attributes. We also note that TEMPLATE is an attribute with unique values, whereas the attributes beginning with INT have value ranges.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT2001 | integer |
| 3 | INT2002 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 199 | STRING195 | string |
| 200 | STRING196 | string |

Table 18a. Record Template for 2000-Byte Records.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT1001 | integer |
| 3 | INT1002 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 99 | STRING095 | string |
| 100 | STRING096 | string |

Table 18b. Record Template for 1000-Byte Records.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT401 | integer |
| 3 | INT402 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 39 | STRING035 | string |
| 40 | STRING036 | string |

Table 18c. Record Template for 400-Byte Records.

| Attribute Number | Attribute Name | Attribute Type |
|---|---|---|
| 1 | TEMPLATE | string |
| 2 | INT201 | integer |
| 3 | INT202 | integer |
| 4 | MULTIPLE | string |
| 5 | STRING001 | string |
| 6 | STRING002 | string |
| . | . | . |
| . | . | . |
| . | . | . |
| 19 | STRING015 | string |
| 20 | STRING016 | string |

Table 18d. Record Template for 200-Byte Records.

Next, we must describe the range of values for each of the record attributes listed in Table 18. Again, we use the term *descriptor* for the attribute-value pair or the attribute-value range and the notation Di-j to identify the j-th descriptor for the i-th directory attribute. The TEMPLATE attribute is used to correlate each record with its corresponding record template. This attribute may take on the four values listed in Table 19, corresponding to the four record sizes. In each record template, the range of values for the attributes, INT2001, INT1001, INT401, and INT201, is a function of the individual record size, (2000, 1000, 400, or 200-bytes), the database-size category, (small, medium, or large), and the test configuration, (1, 2, 3, 4, or 5 for a maximum of three backends, for example). This means that a total of nine test databases are required for benchmarking the database computer with a maximum of three backends. In the discussion to follow, we will refer to these nine databases by their acronyms DB1 to DB9, as described in Table 20.

| TEMPLATE Value | Descriptor Identifier |
|---|---|
| TEMP2000 | D1-1 |
| TEMP1000 | D1-2 |
| TEMP400 | D1-3 |
| TEMP200 | D1-4 |

Table 19. The Values and Ids of the Attribute, TEMPLATE.

| Test Database Acronym | Database Size Category | Database Size in Megabytes |
|---|---|---|
| DB1 | Small | s = 74.976 |
| DB2 | Small | 2s = 149.952 |
| DB3 | Small | 3s = 224.928 |
| DB4 | Medium | s = 149.952 |
| DB5 | Medium | 2s = 299.904 |
| DB6 | Medium | 3s = 449.856 |
| DB7 | Large | s = 299.904 |
| DB8 | Large | 2s = 599.808 |
| DB9 | Large | 3s = 899.712 |

Table 20. List of Test Database Acronyms.

We use database DB1, which is used for configurations 1. 2, and 3 of Table 7, to develop the value ranges for the remaining record attributes. The entries for configuration 1 of Table 7 specify 9.372 2000-byte records, 18,744 1000-byte records, 46,860 400-byte records, and 93,720 200-byte records. We use nine descriptors to classify the value ranges for these attributes, corresponding to the nine cluster categories of Table 10. For the DB1 database, column 5 of Table 13 (Total Number of Records) shows the pertinent values to use for these nine descriptors. The range of values for the nine descriptors for each of the attributes, INT2001, INT1001, INT401, and INT201, are listed in Table 21.

The third directory attribute, INTxx2, enables us to group the records in each of the nine cluster categories (identified by the INTxx1 attributes) into subsets. Referring to column 4 of Table 21, we see that the easiest way to subset each cluster category is to subdivide it into individual clusters. If, for example, we consider attribute INT2002 for the 2000-byte records, we see that we have 86 clusters with 4 records per cluster for a total of 344 records. Therefore, we use 86 descriptors, one per cluster, which is identified by the INT2001 descriptor, D2-1.

The INTxx2 attribute-value ranges are calculated via the relationship [w + xy - (x-1); w + xy], which is described in Figure 7. The lower bound of the range is represented by the term (w + xy - (x-1)), while the second term, w + xy, represents the upper bound. Applying this relationship for the first cluster of the 2000-byte records for the DB1 database, we use w = 0, x = 4, and y = {1,...,86}. Therefore, the range of values for INT2002 becomes [1;4], [5;8], [9:12], ..., [341;344]. For the second cluster, we have w = 344, x = 6, and y = {1,...,87}, to derive the ranges [345;350], [351;356], ..., [861;866].

| Directory Attribute | Descriptor Identifier | Range of Values | Number of Records whose Attribute Values are in the Range |
|---|---|---|---|
| INT2001 | D2-1 | [1;344] | 344 |
| | D2-2 | [345;866] | 522 |
| | D2-3 | [867;1,562] | 696 |
| | D2-4 | [1,563;2,432] | 870 |
| | D2-5 | [2,433;3,476] | 1,044 |
| | D2-6 | [3,477;4,694] | 1,218 |
| | D2-7 | [4,695;6,086] | 1,392 |
| | D2-8 | [6,087;7,652] | 1,566 |
| | D2-9 | [7,653;9,372] | 1,720 |
| INT1001 | D3-1 | [1;688] | 688 |
| | D3-2 | [689;1,732] | 1,044 |
| | D3-3 | [1,733;3,124] | 1,392 |
| | D3-4 | [3,125;4,864] | 1,740 |
| | D3-5 | [4,865;6,952] | 2,088 |
| | D3-6 | [6,953;9,388] | 2,436 |
| | D3-7 | [9,389;12,172] | 2,784 |
| | D3-8 | [12,173;15,304] | 3,132 |
| | D3-9 | [15,304;18,744] | 3,440 |
| INT401 | D4-1 | [1;1,720] | 1,720 |
| | D4-2 | [1,721;4,330] | 2,610 |
| | D4-3 | [4,331;7,810] | 3,480 |
| | D4-4 | [7,811;12,160] | 4,350 |
| | D4-5 | [12,161;17,380] | 5,220 |
| | D4-6 | [17,381;23,470] | 6,090 |
| | D4-7 | [23,471;30,430] | 6,960 |
| | D4-8 | [30,431;38,260] | 7,830 |
| | D4-9 | [38,261;46,860] | 8,600 |
| INT201 | D5-1 | [1;3,440] | 3,440 |
| | D5-2 | [3,441;8,660] | 5,220 |
| | D5-3 | [8,661;15,620] | 6,960 |
| | D5-4 | [15,621;24,320] | 8,700 |
| | D5-5 | [24,321;34,760] | 10,440 |
| | D5-6 | [34,761;46,940] | 12,180 |
| | D5-7 | [46,941;60,860] | 13,920 |
| | D5-8 | [60,861;76,520] | 15,660 |
| | D5-9 | [76,521;93,720] | 17,200 |

Table 21 The Attributes, Their Ids and Their Value Ranges.

Continuing in this manner, we derive the entries shown in Table 22 for the INT2002 range of values. We do not present tables for the corresponding attribute values for the INT1002, INT402, and INT202, since the procedure for deriving these values is identical to that shown for Table 22. Note, however, that the INT1002 descriptor range from D7-1 to D7-781; the INT402 descriptor range from D8-1 to D8-781; and the INT202 descriptor range form D9-1 to D9-781.

The MULTIPLE attribute is a character string which enables us to easily increase the number of records within each cluster. This is required when we need to double or triple the database size to test configurations 4 and 5. For configurations 1, 2, and 3, which use the DB1 database, MULTIPLE is set to 'One'. To double the database size for configuration 4, each (INTxx1, INTxx2) pair must match up with

```
[lower-bound; upper-bound] = [w + xy - (x - 1); w + xy]

where:

    w = sum of records from all previous clusters.
          => Initially, w = 0;
          => At end of each cluster category, before advancing
             to the next INTxx1 descriptor, reset w.
             => w = w + xy,
                where y is the max value for this INTxx1 descriptor.

    x = Number of record per cluster
        {4, 6, 8, 10, 12, 14, 16, 18, 20} for 2000-byte records.
        {8, 12, 16, 20, 24, 28, 32, 36, 40} for 1000-byte records.
        {20, 30, 40, 50, 60, 70, 80, 90, 100} for 400-byte records.
        {40, 60, 80, 100, 120, 140, 160, 180, 200} for 200-byte records.

    y = {1, .... z}

    z = {{86,87}, {172, 174}, {344, 348}}
          => z = {86, 87} for small database, (s/4).
          => z = {172, 174} for medium database, (s/2).
          => z = {344, 348} for large database, (s).
```

Figure 7. INTxx2 Attribute-Value Range Relationship.

MULTIPLE attribute values of 'One' and 'Two'. To triple the database size for configuration 5, each (INTxx1, INTxx2) pair must match up with MULTIPLE attribute values of 'One', 'Two', and 'Three'. This relationship is shown in Table 23.

Finally, the attributes STRINGxxx are used as filler fields, and are all set to the character-string value Xxxxxxxxx. Note that this represents a nine-character string, requiring nine-bytes of storage, whereas the allocated attribute size is ten-bytes. The reason that only nine characters are used is that the C language compiler inserts a null character, (i.e., a backslash-zero), to mark the end of each character string. Therefore, although we use ten-byte attribute, we only have nine usable bytes for our character-string values. The STRINGxxx attributes are also used to allow flexibility in retrieving portions of the database. For example, in the test-transaction mixes we present in this section, we use UPDATE operations to change certain STRINGxxx attributes to values such as OneEighth, One-Qtr, and One-Half. We then use RETRIEVE operations to key on the applicable STRINGxxx fields in order to retrieve 1/8, 1/4, and 1/2 of the database, respectively.

We have now described all of the attributes for the record templates of Table 18. The general layout of the 2000-byte record file for the DB1 database is shown in Table 24.

| INT2001 Descriptor Identifier | INT2001 Range of Values | INT2002 Descriptor Identifier | INT2002 Range of Values |
|---|---|---|---|
| D2-1 | [1;344] | D6-1 | [1;4] |
| | | D6-2 | [5;8] |
| | | ... | ... |
| | | D6-86 | [341;344] |
| D2-2 | [345;866] | D6-87 | [345;350] |
| | | D6-88 | [351;356] |
| | | ... | ... |
| | | D6-173 | [861;866] |
| D2-3 | [867;1,562] | D6-174 | [867;874] |
| | | D6-175 | [875;882] |
| | | ... | ... |
| | | D6-260 | [1.555;1,562] |
| D2-4 | [1,563;2,432] | D6-261 | [1,563;1,572] |
| | | D6-262 | [1.573;1,582] |
| | | ... | ... |
| | | D6-347 | [2,423;2.432] |
| D2-5 | [2,433;3,476] | D6-348 | [2,433;2,444] |
| | | D6-349 | [2.445;2,456] |
| | | ... | ... |
| | | D6-434 | [3,465;3,476] |
| D2-6 | [3.477;4,694] | D6-435 | [3.477;3,490] |
| | | D6-436 | [3.491;3.504] |
| | | ... | ... |
| | | D6-521 | [4.681;4,694] |
| D2-7 | [4.695;6.086] | D6-522 | [4.695;4,710] |
| | | D6-523 | [4.711;4,726] |
| | | ... | ... |
| | | D6-608 | [6.071;6,086] |
| D2-8 | [6,087;7,652] | D6-609 | [6,087;6,104] |
| | | D6-610 | [6.105;6,122] |
| | | ... | ... |
| | | D6-695 | [7.635;7.652] |
| D2-9 | [7.653:9.372] | D6-696 | [7.653:7.672] |
| | | D6-697 | [7.673;7.692] |
| | | ... | ... |
| | | D6-781 | [9.353:9.372] |

Table 22. The Value Ranges of the Attribute, INT2002.

| TEMPLATE | INT2001 | INT2002 | MULTIPLE |
|---|---|---|---|
| TEMP2000 | 1 | 1 | One |
| | 2 | 2 | One |
| | . . . | . . . | . . . |
| | 9,372 | 9,372 | One |
| | 1 | 1 | Two |
| | 2 | 2 | Two |
| | . . . | . . . | . . . |
| | 9,372 | 9,372 | Two |
| | 1 | 1 | Three |
| | 2 | 2 | Three |
| | . . . | . . . | . . . |
| | 9,372 | 9,372 | Three |

Table 23. Use of the Attribute, MULTIPLE, for Database DB3 (of 3s-Megabytes).

| | Attributes | | | | | | |
|---|---|---|---|---|---|---|---|
| | TEMPLATE | INT2001 | INT2002 | MULTIPLE | STRING001 | . . . | STRING196 |
| V | TEMP2000 | 1 | 1 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| a | TEMP2000 | 2 | 2 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| l | TEMP2000 | 3 | 3 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| u | TEMP2000 | 4 | 4 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| e | . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| s | TEMP2000 | 9,371 | 9,371 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |
| | TEMP2000 | 9,372 | 9,372 | One | Xxxxxxxxx | . . . | Xxxxxxxxx |

Table 24. File Layout of the 2000-Byte Records for DB1.

## 4.8. A Summary of the Test-Database Methodology.

Let us summarize these database-design considerations. First, we decide to test with three database sizes. (small = $s$ 4. medium = $s$ 2. and large $s$). The largest database is approximately the maximum formated capacity of a backends' disks for the database storage.

Second. to determine the largest feasible upper-bound for each test (benchmark) database. we find the corresponding multiple of database size from Table 2. If the system being evaluated has $m$ backends. $s$ must be divisible by (LCM{1. 2...., $m$ } x 32 x rec_size). The (LCM{1. 2...., $m$ } x 32) portion of the database-size multiple, not including rec_size, is used to determine an upper-bound for the large database size, $s$ .

Third. we consider the record-size parameter. We select four record sizes on the basis of the size of the data-storage-and-access unit used by the particular computer. We select one large and one small size, with two intermediate sizes. We require that the largest record size be divisible by each of the three smaller record sizes to simplify the database sizing process.

Fourth. we calculate the required database multiple in accordance with Table 2, using the largest record size selected in the previous step for the rec_size parameter value. Since the other three record sizes are divisors of the rec_size parameter, we are assured that the database we create using this database multiple will be divisible by all four record sizes.

Fifth, since the database multiple is now known, we calculate the required values of $s$ , $s$ / 2, and $s$ / 4, respectively. This calculation enables us to verify that these three databases are feasible, since they permit each database to be distributed evenly as required for all of the feasible test configurations.

Sixth, we format the test (benchmark) databases. We include all four record sizes in each database to streamline the benchmarking task. These databases are formated in Tables 7, 8 and 9, for example. These formats show that whichever database size and four record sizes are selected, our design generates test (benchmark) databases in which each database may be evenly distributed on the disks of the backends as required for all of the feasible test (benchmark) configurations.

Seventh, we specify the record templates for each of the four record sizes for each test database, and we develop the descriptors for the 2000-byte record file for the DB1 database, for example. The development of the descriptors for the rest of the files is straightforward, and follows the steps presented for the case of the 2000-byte record file exactly. The descriptors for the database DBi (where i=2 or 3 in our samples) are also developed for the capacity-growth measures similarly as for DB1. The only change is that the number of records per cluster doubles, for example in DB2, or triples. for example, in DB3. Therefore the corresponding ranges of values for the descriptors of INTxx1 and INTxx2 must be also doubled and tripled, respectively.

For the databases DBj (where j=4, 5 or 6 in our sample), there are 1,562 descriptors of INTxx2 for each record template, since the number of clusters doubles for the medium-size database. Similarly, there are 3,124 descriptors of INTxx2 per record template for the databases DBk (where k=7, 8 or 9 in our sample), since the number of clusters doubles again from the medium to large database set.

Although the methodology presented in this section is straightforward, the amount of work involved in the design and generation of a specific set of test databases for benchmarking a multiple-backend database computer is still heavy. Consequently, much of the methodology is being computerized as a CAD system.

Having established these test databases, we may now turn the attention to the test-transaction mixes to be used with these test databases for measuring the performance and growth of the three-backend database computer. In Section 5, we present the methodology for designing and generating test-transaction mixes. In order to provide the reader and system evaluator with a feel of the workload generated by the mixes for the multiple-backend database computer, we need to focus on a sample set of test databases. Thus, our application of the methodology for the design and generation of test databases for a specific three-backend database computer reported herein becomes a timely exercise. Although the application is rather long and tedious, it is important to our understanding of the methodology and it is necessary for our calculation of the workloads of the test-transaction mixes presented in the following section.

## 5. THE DESIGN AND GENERATION OF TEST-TRANSACTION MIXES

As noted earlier, if we are to test the response-time invariance of multiple-backend database computers, we must ensure that any increase in the size of the response set returned by the test-transaction mix is accompanied by a proportional increase in the number of backends in the computer. To induce the increase in responses, we have introduced a methodology in the previous section to increase the size of the test databases. However, the selection of a test-transaction mix which will permit the database size to increase in the same proportion as the increase in the response-set size is much more complex. The selection requires an understanding of the characteristics and features of the data model and data manipulation language. Also, the directory structure and storage strategies of the computer play a major role. Nevertheless, we must show how to design a test-record organization, a test-database structure, and a test-transaction-mix set which enables the system evaluator to use the same organization, structure, and set for all system configurations *without modification!* Furthermore, the transactions that we select must ensure the system evaluator that the database size will increase in exactly the same proportion as the increase in the response-set size.

In addition to the development of test-transaction mixes for the two established measures, we also develop test-transaction mixes to measure the overall performance of the multiple-backend database computer. In [8], Hawthorn and Stonebraker, suggest that three types of test transactions be used to measure the overall performance. One type consists of *overhead-intensive transactions* for which the actual time required to process the required data is much less than the system overhead required to carry out the transaction. The *data processing time* is defined as the time required for the computer to fetch and manipulate the required data, whereas *system overhead* involves both the times spent by operating and database management systems for such tasks as user communication, transaction parsing and validity checking. In essence, overhead-intensive transactions reference very little data. The second type of transaction is *data-intensive* where the data processing time is much greater than the system overhead. Therefore, data-intensive transactions reference large quantities of data. Finally, the last type of transaction, is for *multi-relation* or *multi-file* transactions. They are intended for relational joins or file merges for transactions involving more than one relation (file). These types of operations correspond to the relation join in the DBC/1012 and the retrieve-common operation of MBDS. We consider all of these factors in selecting transactions to measure the performance gains and capacity growth of the multiple-backend database computer as well as its overall performance.

### 5.1. The Emphasis on Generic, Primary Database Operations

Instead of focusing our discussion of database operations on the basis of a specific set of data model and data language of the database computer, we refer to the primary database operations generically. All database computer, whether or not they are multiple-backend and whether or not they are relational, have these five primary database operations, namely, DELETE, INSERT, RETRIEVE, UPDATE and RETRIEVE-COMMON.

The RETRIEVE and DELETE operations have very similar processing steps. Let us first consider the RETRIEVE operation. The search for indices or index ranges for the predicates of the RETRIEVE commences first. This is the *descriptor-search phase*. These indices enable the computer to determine the partitions (clusters) which contain records satisfying the predicates. This is the *cluster-search phase*. Once the clusters are identified, the addresses of the partitioned (clustered) records can then be found. This is the *address-generation phase*. Finally, in the *record-processing phase* the backends fetch the clustered records from their respective disks. Record processing selects from the staged data set the records that satisfy the predicates, extracts the relevant values from the selected records, performs the required aggregate operations, and then forwards the results to the controller for post processing.

The DELETE operation follows almost the same phases. Following the descriptor search, cluster search, and address generation, record processing fetches the selected records from the disks. Record processing selects from the staged data set the records that satisfy the predicates, marks the selected records for deletion, and then returns them to the disks. Record processing then sends a completion message to the controller. We expect that the RETRIEVE and DELETE operations will provide important statistics for verifying the performance-gains and capacity-growth measures. Therefore, we design a diverse mixture of overhead-intensive and data-intensive transactions involving RETRIEVEs and DELETEs.

The RETRIEVE-COMMON operations provide the opportunity to test multi-file or multi-relation operations, i.e., relational joins. Logically, record processing handles two RETRIEVE operations, and fetches two sets of records from the disks. Record processing then selects from the two staged record sets in the primary memory the records whose attribute values are common and whose attributes are specified in the COMMON clause, and returns the results to the controller.

To test the INSERT operation, we propose two sets of transactions. One set inserts new records into *existing* partitions (clusters), while the second set inserts records into *new* partitions (clusters). Similarly, three types of UPDATE operations are possible. One type of UPDATE operation returns the modified records to the *same, existing* partitions (clusters). The second type of UPDATE causes the modified records to change partitions (clusters). The "old" records are deleted, and the "new" records are inserted into *different, existing* partitions (clusters), or to *new* partitions (clusters). Finally, the third type of UPDATE is a blend of the first two types. That is, some of the modified records stay in the same, existing partitions (clusters), while other records change partitions (clusters). We include all three types of UPDATEs in our test-transaction mix.

We anticipate that the primary operation to be performed on a multiple-backend database computer will be to retrieve data from the database store. Therefore, benchmarks which focus on the RETRIEVE operation will provide useful data for conducting the performance-gains and capacity-growth measures. To measure the overall computer performance, we propose test-transaction mixes which include a complete set of the five generic and primary database operations, i.e., DELETE, INSERT, RETRIEVE, RETRIEVE-COMMON, and UPDATE.

## 5.2. The Test-Transaction Mixes

Due to the presence of sample test databases in Section 4, we are able to estimate the workload of each and every test-transaction mix to be introduced in the following section. These workload estimates do provide precise and quantitative measurements of the test transactions intended.

### 5.2.1. Data-Intensive and Overhead-Intensive Retrievals

Table 25 displays the predicates used for our first three retrievals, while Table 26 represents an analysis of the workload incurred by the transactions in Table 25. Let us briefly analyze the intent of each of these transactions.

| Transaction Number | The Predicates of a RETRIEVAL |
|---|---|
| 1 | ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 121) and (INT2001 $\leqslant$ 132)) |
| 2 | (((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 4,823) and (INT2001 $\leqslant$ 4,870)) or ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 6,087) and (INT2001 $\leqslant$ 6,122))) |
| 3 | ((TEMPLATE = TEMP2000) and (INT2002 $\leqslant$ 2,343)) |

Table 25. Transaction Mix 1.

| Transaction Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Retrieved |
|---|---|---|---|
| 1 | 86 | 344 records | 12 records |
| 2 | 174 | 31.56% | 84 records |
| 3 | 339 | 25.09% | 25.00% |

Table 26. Transaction-Mix-1 Workload.

Transaction 1 examines the small portion of the database represented by the attribute INT2001 and its descriptor-id D2-1. (See Table 21 again.) This transaction causes 344 records to be staged from the disks to the primary memory. However, only the 12 records from clusters C30, C31, and C32 are answers of the transaction. Therefore, the transaction evaluates how well the database computer performs when it examines a small amount of data (344/9372 records, or 3.67% of the database), and retrieves only a small amount of data from the set examined (12/344 records, or 3.49%). (See Table 14 again.) We classify transaction 1 as overhead-intensive.

Transaction 2 is designed to examine a large portion of the database (31.56%), but to retrieve only a small portion of the data examined. Although the transaction causes 2,958 records to be staged from the disks to the primary memory, only 84 records (48 from clusters C530, C531, and C532, and 36 from clusters C609 and C610) participate in the response set. Thus, this transaction evaluates how well the database computer performs when it retrieves only a small amount of data from a large amount of data (84/2958 records, or 2.84%) which must be examined. Although the amount of data retrieved is small, the database computer must access a large amount of data to satisfy the predicates. Therefore, we classify transaction 2 as data-intensive.

Transaction 3 retrieves 25% of the database. The transaction examines a large portion of the database (25.09%, or 2,352 records). Of the 2,352 records which are staged to the primary memory, 99.62% (2343/2352) are relevant to the response set. Therefore, this transaction evaluates how well the database computer performs when nearly all of the data examined are answers to the transaction. We classify transaction 3 as data-intensive.

### 5.2.2. Simple, Data-Intensive Updates

Table 27 displays the predicates for transactions 4, 5, and 6. They are all UPDATEs which will return the updated records to their same, existing clusters. Table 28 depicts an analysis of the workload associated with each of these UPDATEs. The intent of transactions 4, 5, and 6 is to update 1/8, 1/4, and 1/2 of the database, respectively.

| Transaction Number | The Predicates of an UPDATE |
|---|---|
| 4 | ((TEMPLATE = TEMP2000) and (INT2002 $\leq$ 1,172)) (STRING001 = OneEighth) |
| 5 | ((TEMPLATE = TEMP2000) and (INT2002 $\leq$ 2,343)) (STRING005 = OneQuartr) |
| 6 | ((TEMPLATE = TEMP2000) and (INT2002 > 4,686)) (STRING010 = One-Half) |

Table 27. Transaction Mix 2.

| Transaction Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Updated |
|---|---|---|---|
| 4 | 212 | 12.57% | 12.50% |
| 5 | 339 | 25.09% | 25.00% |
| 6 | 261 | 50.06% | 50.00% |

Table 28. Transaction-Mix-2 Workload.

Transaction 4 updates one-eighth of the database causing 1,178 records from 212 clusters to be staged from the disks to the primary memory. Then, 1,172 records (1/8 of 9,372) have the values of the attribute STRING001 changed to the character-string value OneEighth. These records are then returned to their original, existing clusters in the disks. This transaction evaluates how well the database computer performs when nearly all of the accessed data (1172/1178 records, or 99.49%) is updated. Since most of the workload for this transaction involves accessing and processing data records, we classify transaction 4 as data-intensive.

Transaction 5 updates one-quarter of the database. With this transaction, 2,343 of the 2,352 accessed records are updated and returned to the same, existing clusters on the disks. This transaction updates the values of the attribute STRING005 to the new character-string value One-Quartr. Similarly, transaction 6 updates one-half of the database. The transaction updates 4,686 of the 4,692 accessed records, and returns them to their original, existing clusters on the disks. Transaction 6 changes the STRING010 value to One-Half. We classify transactions 5 and 6 as data-intensive. We note that the attribute STRING001 is a non-directory attribute. In general, updates of the values of non-directory attributes require no change of the clusters for the records.

### 5.2.3. Data-Intensive and Overhead-Intensive Common Retrievals

Table 29 depicts the transaction specifications for transaction 7, 8, and 9, which are all RETRIEVE-COMMON operations. The corresponding workload statistics are shown in Table 30.

| Transaction Number | RETRIEVE-COMMON Specification |
|---|---|
| 7 | RETRIEVE ((TEMPLATE = TEMP2000) and (INT2001 ⩾ 121) and (INT2001 ⩽ 132)) (INT2001)<br>COMMON(INT2001, INT1001)<br>RETRIEVE ((TEMPLATE = TEMP1000) and (INT1001 ⩽ 264)) (INT1001) |
| 8 | RETRIEVE ((TEMPLATE = TEMP2000) and (STRING010 = One-Half)) (INT2002)<br>COMMON(INT2001, INT1001)<br>RETRIEVE ((TEMPLATE = TEMP1000) and (STRING010 = One-Half)) (INT1002) |
| 9 | RETRIEVE ((TEMPLATE = TEMP2000) and (INT2001 ⩽ 4,686)) (INT2001)<br>COMMON(INT2002, INT1002)<br>RETRIEVE ((TEMPLATE = TEMP1000) and (INT1001 ⩾ 3,515) and (INT1001 ⩽ 4,686)) (INT1001) |

Table 29. Transaction Mix 3.

| Trans. Number | Number of Clusters Examined by the Source Trans. | Number of Records Accessed by the Source Trans. | Number of Records Relevant to the Source Trans. | Number of Clusters Examined by the Target Trans. | Number of Records Accessed by the Target Trans. | Number of Records Relevant to the Target Trans. | Size of the Result Record Set in Records |
|---|---|---|---|---|---|---|---|
| 7 | 86 | 344 | 12 | 86 | 688 | 264 | 12 |
| 8 | 781 | 9.372 | 4.686 | 781 | 18,744 | 9,372 | 4,686 |
| 9 | 261 | 4.692 | 4.686 | 87 | 1,740 | 1,172 | 1,172 |

Table 30. Transaction-Mix-3 Workload.

We interpret transaction 7 as follows. The first RETRIEVE on the 2000-byte record file of database DB1 is called the source transaction. This source transaction causes 344 records to be staged from the disks to the primary memory. The 12 records which satisfy this source transaction are retrieved and stored in a buffer area which we refer to as the *source record set*.

The second RETRIEVE, which retrieves records from the 1000-byte record file, is called the target transaction. When it processes this target transaction, the database computer stages 688 records to the primary memory, selecting the 264 records which satisfy the target transaction and saves them in a second buffer area which we call the *target record set*.

Finally, the database computer does a pairwise-merge operation between the records of the source and target record sets. During this merge, the computer selects the 12 records from the source and target record sets which share common INT2001 and INT1001 attribute values, and returns them to the controller. Note that we retrieve the smallest number of records from the source file, while the larger file to be searched against is designated as the target file. This feature is intrinsic to the efficient merge operation. The purpose of this transaction is to see how well the computer performs a RETRIEVE-COMMON (relational join) operation, when it examines a small amount of data for both the source and target transactions, for which only a small amount of the staged data is relevant to the answer. Relative to the next two RETRIEVE-COMMONs, transaction 7 may be categorized as an overhead-intensive transaction.

Transaction 8 causes all 9,372 records to be accessed from the 2000-byte record file. Of these, 4,686 records (50%) are relevant to the source transaction, and are selected for the source record set. The target transaction accesses all 18,744 records from the 1000-byte record file, of which 9,372 records (50%) are relevant to the target transaction, and are selected for the target record set. The database computer performs the merge operation between the source and target record sets, and returns the 4,686 records which have common INT2001 and INT1001 attribute values to the user via the controller. The purpose of this transaction is to gauge the performance of the computer when it stages large quantities of data from the disks, for which 50% of the staged data is relevant for both the source and the target transactions. Thus, transaction 13 exemplifies a data-intensive query, which also experiences a significant amount of overhead.

The number of records in the source record set for transactions 7 and 8 directly corresponds to the relevant data to be returned to the user. We assume the opposite approach with transaction 9. The source transaction for transaction 9 causes 4,692 records from the 2000-byte record file to be staged to the primary memory. Of these records, 4,686 are relevant to the source transaction, and enter into the source record set. The target transaction stages 1,740 records from the 1000-byte record file, of which 1,172 records are relevant to the target transaction. Here, we force the database computer to execute an inefficient merge operation by using a source record set which is much larger than the target record set. As a result of the merge operation on the source and target record sets, the 1,172 records which share common INT2002 and INT1002 attribute values are returned to the user via the controller. Transaction 9 gauges the database computer performance for the case where nearly all of the records staged for the source transaction are relevant to the source transaction, while only 25% of the records staged for the target transaction are relevant. We categorize transaction 9 as being overhead-intensive and data-intensive.

### 5.2.4. Simple and Complex Inserts

Table 31 shows the records to be inserted by transactions 10 and 11, respectively. The intent of transactions 10 and 11 is to see if a single INSERT experiences a response-time variance as the number of backends in the test configuration increases. Transaction 10 inserts a record into an *existing* partition (cluster) (i.e., C1), while transaction 11 inserts a record into a *new* partition (cluster). We term the former a simple INSERT, the latter a complex INSERT, since the calculation and creation of a new cluster is a complex process.

| Transaction Number | Record to be INSERTed |
|---|---|
| 10 | (<TEMPLATE,TEMP2000>,<INT2001,1>,(INT2002,1>,<MULTIPLE,Four>, <STRING001,Xxxxxxxxx>, ..., <STRING196,Xxxxxxxxx>) |
| 11 | (<TEMPLATE,TEMP2000>,<INT2001,1>,(INT2002,400>,<MULTIPLE,One>, <STRING001,Xxxxxxxxx>, ..., <STRING196,Xxxxxxxxx>) |

Table 31. Transaction Mix 4.

### 5.2.5. Overhead-Intensive and Data-Intensive Deletes

We expect to note that performance-gains statistics from DELETEs will be comparable to those collected by RETRIEVEs, since the processing steps associated with each of these database operations are very similar. Consequently, we select the three DELETEs shown in Table 32 which are designed to imitate the workload of the transactions 1, 2 and 3 for retrievals. Table 33 depicts the workload analysis corresponding to these DELETE operations.

| Transaction Number | The Predicates of a DELETE |
|---|---|
| 12 | ((TEMPLATE = TEMP2000) and (INT2001 $\geq$ 121) and (INT2001 $\leq$ 132)) |
| 13 | ( ( (TEMPLATE = TEMP2000) and (INT2001 $\geq$ 4,823)and (INT2001 $\leq$ 4,870)) or ((TEMPLATE = TEMP2000) and (INT2001 $\geq$ 6,087)and (INT2001 $\leq$ 6,122))) |
| 14 | ((TEMPLATE = TEMP2000) and (INT2002 $\geq$ 7,030)) |

Table 32. Transaction Mix 5.

| Transaction Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Deleted |
|---|---|---|---|
| 12 | 86 | 344 records | 12 records |
| 13 | 174 | 31.56% | 84 records |
| 14 | 121 | 25.07% | 25.00% |

Table 33. Transaction-Mix-5 Workload.

The DELETE operation for transaction 12 will cause the database to stage 344 records to the primary memory, but will only delete the 12 records from clusters C30, C32, and C32. Therefore, this transaction gauges the computer performance when it examines a small amount of data (344/9,372 records), and deletes only a small amount of data from the set examined (12/344 records). We classify transaction 12 as primarily overhead-intensive.

Similarly, transaction 13 causes 2,958 records to be staged to the primary memory, but only deletes 84 of the records accessed. Thus, the transaction evaluates how well the database computer performs when it deletes only a small amount of data from a large amount of data which must be accessed (84/2958 records, or 2.84%). We classify this transaction as both overhead-intensive and data-intensive, since it must examine a large-number of records, although only a small number of records are relevant to the answer.

Transaction 14 causes the database computer to examine a large portion of the database (25.09%, or 2,352 records), and delete 99.62% (2,343/2,352) of the records examined. Thus, transaction 14 gauges the computer performance when nearly all of the data examined is deleted. Transaction 14 is therefore a data-intensive transaction.

### 5.2.6. Complex Data-Intensive and Overhead-Intensive Updates

Table 34 specifies the predicates for our set of complex UPDATEs, while Table 35 depicts the corresponding workload analysis. By complex, we mean again that the operation will involve the calculation and creation of new partitions (clusters) or the migration to other existing partitions (clusters). Transaction 15 will cause the database computer to update 12 records, causing the records to switch to brand new partitions (clusters). Therefore, the 12 "old" records will be deleted from the existing partitions (clusters), and the 12 "new" records will be inserted into newly created partitions (clusters). This transaction will gauge how well the database computer performs when it must examine a small amount of data (344/9372 records), and update a small amount of data from the set accessed (12/344 records), resulting in 12 record deletions and 12 record insertions. We classify transaction 15 as overhead-intensive.

| Transaction Number | The Predicates of an UPDATE and their Update Expression |
|---|---|
| 15 | ((TEMPLATE = TEMP2000) and (INT2001 $\geqslant$ 121) and (INT2001 $\leqslant$ 132))<br>(INT2001 = INT2001 + 2,312) |
| 16 | ((TEMPLATE = TEMP2000) and (INT2002 $\leqslant$ 2,343))<br>(INT2001 = INT2001 + 4,694) |
| 17 | ((TEMPLATE = TEMP2000) and (INT2002 > 7,653) and (INT2002 $\leqslant$ 9,352))<br>(INT2002 = INT2002 + 20) |
| 18 | ((TEMPLATE = TEMP2000) and (INT2002 > 3,477) and (INT2002 $\leqslant$ 3,504))<br>(INT2002 = INT2002 + 14) |
| 19 | ((TEMPLATE = TEMP2000) and (INT2002 > 5,287) and (INT2002 $\leqslant$ 5,350))<br>(INT2002 = INT2002 + 8) |
| 20 | ((TEMPLATE = TEMP2000) and (INT2001 > 7,029))<br>(INT2002 = INT2002 + 10) |

Table 34. Transaction Mix 6.

| Transaction Number | Number of Clusters Examined | Volume of Database Accessed | Volume of Database Updated |
|---|---|---|---|
| 15 | 86 | 344 records | 12 records |
| 16 | 339 | 25.09% | 25.00% |
| 17 | 86 | 18.35% | 18.14% |
| 18 | 2 | 28 records | 28 records |
| 19 | 4 | 64 records | 64 records |
| 20 | 172 | 35.06% | 25.00% |

Table 35. Transaction-Mix-6 Workload.

Transaction 16 is designed to update 25% of the database, causing the records to migrate to brand new partitions (clusters). This transaction will cause 2,352 records to be staged into the primary memory.

Of these, 2,343, or 99.62% (2,343/2,352) will be updated. This will result in 2,343 record deletions, accompanied by an identical number of record insertions into newly created partitions (clusters). Thus, the transaction will test the database computer performance when it must access a large amount of data, and then update nearly all of the accessed records, resulting in a sizable migration of records into newly created partitions (clusters). We classify the transaction as data-intensive.

In contrast, the UPDATE operations of transactions 17 and 18 are designed to cause a migration of records into *existing* clusters. Transaction 17 accesses 1,720 records, and causes 1,700 records, or 98.84% of the records examined to switch to different, existing partitions (clusters). Therefore, the database computer will delete 1,700 "old" records, and insert 1,700 "new" records into existing partitions (clusters). Transaction 17 is data-intensive. Transaction 18 causes the database computer to examine just 28 records. However, all 28 records are updated and forced to migrate to different, existing partitions (clusters). Transaction 18 is primarily overhead-intensive.

The purpose of the last two UPDATE operations is to have some records remain in the same partition (cluster). some migrate to different, existing partitions (clusters), and others migrate to newly created partitions (clusters). Transaction 19 causes MBDS to examine just 64 records. However, all 64 records accessed are updated. One-half of the updated records remain in their same, existing partitions (clusters), while the others migrate to different, existing partitions (clusters). Transaction 19 is primarily overhead-intensive.

Finally, transaction 20 updates 25% (2,343/9,372 records) of the database which causes 3,286 records to be staged into the primary memory. Of these staged records, 2,343, or 71.30% (2,343/3,286) are updated. Some of these records stay in the same partition (cluster), others migrate to different, existing partitions (clusters), while the last 10 records migrate to a newly created partition (cluster). We classify transaction 20 as data-intensive. It is important to observe that in these updates the attribute values being updated are the values of the directory attributes.

### 5.2.7. The Benchmarking Sequence

The execution order of the benchmarks is an important factor to consider. We present a way to sequence the test transactions and minimize the need to reload the test database. Transactions 1 through 14 may be executed in sequence. Transactions 15 through 20 do affect each other. since the various UPDATE operations act on overlapping record sets. They should be executed separately.

### 5.2.8. A Summary of the Test-Transaction Methodology

The reader should note that these test-transaction mixes are only for the DB1 database of Table 20, which is used for test configurations 1, 2, and 3 for the small-size database set. However, the same transactions may be used to test with the medium and large database. DB2 and DB3, respectively.

Although the number of records doubles from DB1 to DB2. and triples from DB1 to DB3, attribute-value ranges for INT2001 and INT2002 remain the same. For each pair of INT2001 and INT2002. the MULTIPLE attribute produces two unique records for the DB2 database. and three unique records for the DB3 database. Since the test-transaction mixes are all keyed on the attribute values of INT2001 and INT2002, the effect is that the number of records retrieved by transaction 1, for example, will double to 24 with the DB2 database, and triple to 36 with the DB3 database. Similar changes occur with the number of records retrieved, deleted, or updated by the other test transactions.

Therefore, we have achieved the effect of increasing the size of the responses in the same proportion to corresponding increases in the database size, using the same set of test-transaction mixes. In other words, we have a test-record organization, a test-database structure, and a set of test-transaction mixes which enable the system evaluator to use the same organization, structure, and set for all configurations for a particular database size *without modification!*

The system evaluator must keep the following factors in mind, nevertheless. The test-transaction mixes presented so far must be run for all four record (tuple) files (relations) for each test database, for all three database sizes (small, medium, and large), and for all configurations, i.e., five, when testing a computer with a maximum of three backends. Since the same mix of transactions may be used for all configurations for a given database size, we require only 12 different mixes of test transactions (one each for each record file, per database size).

Obviously, the required number of tests (benchmarks) grows considerably if a database computer with more than three backends is to be benchmarked. Therefore, the system evaluator may choose a subset of these test-transaction mixes for a quick estimation of the performance-gains and capacity-growth of the multiple-backend database computer.

## 6. CONCLUDING REMARKS

The use of a simple 3-backend database computer for the illustration of design and generation of the test databases and test-transaction mixes is our attempt (1) to limit the length of this paper, (2) to provide a concrete example for the application of the methodology, (3) to facilitate a quick understanding of the general methodology for the reader, and (4) to demonstrate the computation of the design data and workload characterizations without resorting to our CAD system for the methodology, since the CAD system has not yet been completed.

Despite the overwhelming amount of tabulated design data and our focus on a simple multiple-backend database computer with three backends, the benchmarking methodology presented herein is general and effective. It is *general* because it works for any number of backends. It is also *effective* because we are able to use the methodology for benchmarking an 8-backend database computer initially and a 16-backend database computer later. Since we are in the process of computerizing the methodology as a CAD system, the test databases and test-transaction mixes can be designed and generated with ease, in the future. Furthermore, the tabulated design data and workload estimations can also be provided automatically. Such a benchmark-design-and-generation system can indeed apply the methodology to a $m$-backend database computer for large $m$.

## REFERENCES

[1] "SQL/Data System - Concepts and Facilities," IBM, GH24-5013-2, Third Edition, August 1983.

[2] "IDM 500 Series - The Logical Approach to Intelligent Database Management," Britton-Lee, Inc. (This is a recent brochure on Britton-Lee Database Computers.)

[3] Canaday, R. E., Harrison, R. D., Ivie, E. L., Ryder, J. L., and Wehr, L. A., "A Back-end Computer for Data Base Management," *Communications of the ACM,* Vol. 17, No. 10, October 1974.

[4] Kerr, D.S., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MBDS," and He, X., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part II - The First Prototype MBDS and the Software Engineering Experience," *Advanced Database Machine Architectures,* Hsiao, D. K., (Editor), Prentice-Hall, 1983.

[5] Neches, P. M., "Hardware Support for Advanced Data Management Systems," *IEEE Computer*, Vol. 17, No. 11, November 1984.

[6] Strawser, P. R., "A Methodology for Benchmarking Relational Database Machines," Ph.D. Dissertation, The Ohio State University, Columbus, Ohio, March 1984.

[7] Vincent, J. R., "A Performance Measurement Methodology for Software Multi-Backend Database Systems," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1985.

[8] Hawthorn, P. B., and Stonebraker, M., "Performance Analysis of a Relational Data Base Management System," *Proceedings of the ACM SIGMOD Conference on Management of Data,* 1979.

## INITIAL DISTRIBUTION LIST

| | |
|---|---|
| efense Technical Information Center<br>ameron Station<br>lexandria, VA 22314 | 2 |
| udley Knox Library<br>ode 0142<br>aval Postgraduate School<br>onterey, CA 93943-5100 | 2 |
| ffice of Research Administration<br>ode 012A<br>aval Postgraduate School<br>onterey, CA 93943-5100 | 1 |
| hairman, Code 52Ml<br>omputer Science Department<br>aval Postgraduate School<br>onterey, CA 93943-5100 | 40 |
| avid K. Hsiao<br>ode 52Hq<br>omputer Science Department<br>aval Postgraduate School<br>onterey, CA 93943-5100 | 150 |
| hief of Naval Research<br>rlington, VA 22217 | 1 |
| enter for Naval Analyses<br>2000 N. Beauregard Street<br>Alexandria, VA 22311 | 1 |