

AD-A212 761

NAVAL POSTGRADUATE SCHOOL
Monterey, California



DTIC FILE COPY

THESIS

PERFORMANCE EVALUATIONS OF A
PARALLEL AND EXPANDABLE
DATABASE COMPUTER --
THE MULTI-BACKEND DATABASE COMPUTER

by

James E. Hall

June 1989

Thesis Advisor:

David K. Hsiao

Approved for public release; distribution unlimited

DTIC
ELECTE
SEP 21 1989
S B D

18 174

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification UNCLASSIFIED		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report Approved for public release; distribution is unlimited.	
2b Declassification/Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol (If Applicable) 52	7a Name of Monitoring Organization Naval Postgraduate School	
6c Address (city, state, and ZIP code) Monterey, CA 93943-5000		7b Address (city, state, and ZIP code) Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number	
8c Address (city, state, and ZIP code)		10 Source of Funding Numbers	
		Program Element Number	Project No Task No Work Unit Accession No
11 Title (Include Security Classification) PERFORMANCE EVALUATIONS OF A PARALLEL AND EXPANDABLE DATABASE COMPUTER -- THE MULTI-BACKEND DATABASE COMPUTER			
12 Personal Author(s) Hall, James E.			
13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report (year, month, day) June 1989	15 Page Count 100
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
17 Cosati Codes		18 Subject Terms (continue on reverse if necessary and identify by block number)	
Field	Group	Database Computer, Parallel Processing, Performance Evaluation, Benchmarking, Multi-Backend Computer.	
19 Abstract (continue on reverse if necessary and identify by block number) ► This study is the actual application of a performance evaluation technique known as benchmarking to an experimental database management system (DBMS). The specific DBMS evaluated is the Multi-Backend Database System (MBDS) which is a software multiple-backend database system. The unconventional nature of a multiple-backend computer system required the development of a special performance evaluation methodology which was the topic of several related theses. A previously developed performance evaluation methodology and the computer assisted benchmarking tools developed to implement the methodology had only been applied to MBDS on a very small scale and had not been used with the current set of modern MBDS hardware. The focus of this thesis is the verification of the performance claims made by the implementor of MBDS. These performance claims were, in fact, validated by conducting a series of relatively large-scale benchmarking experiments in which MBDS performed, generally, as predicted by its implementor. While the results are encouraging, future benchmarking experiments need to be conducted on an even larger database to examine MBDS performance under an extreme load. This will require the development of a high-speed database loading utility which is not the focus of this thesis. Here, we report on the test databases, test transactions and test results (which constitute the benchmarks) used to verify the MBDS implementor's claims of response-time reduction and response-time invariance.			
20 Distribution/Availability of Abstract <input checked="" type="checkbox"/> unclassified/unlimited <input type="checkbox"/> same as report <input type="checkbox"/> DTIC users		21 Abstract Security Classification UNCLASSIFIED	
22a Name of Responsible Individual Prof. David K. Hsiao		22b Telephone (Include Area code) (408) 646-2253	22c Office Symbol Code 52Hq

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted

security classification of this page

All other editions are obsolete

Unclassified

Approved for public release; distribution is unlimited.

Performance Evaluations of a Parallel and Expandable
Database Computer -- the Multi-Backend Database Computer

by

James E. Hall
Captain, United States Marine Corps
B.S., National University, 1984

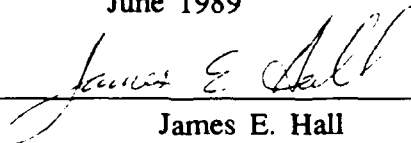
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE


from the

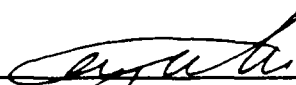
NAVAL POSTGRADUATE SCHOOL
June 1989

Author:

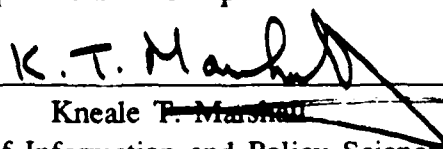

James E. Hall

Approved by:


David K. Hsiao, Thesis Advisor


C. Thomas Wu, Second Reader


Robert B. McGhee, Chairman
Department of Computer Science


Kneale P. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This study is the actual application of a performance evaluation technique known as *benchmarking* to an experimental database management system (DBMS). The specific DBMS evaluated is the Multi-Backend Database System (MBDS) which is a software multiple-backend database system. The unconventional nature of a multiple-backend computer system required the development of a special performance evaluation methodology which was the topic of several related theses. A previously developed performance evaluation methodology and the computer assisted benchmarking tools developed to implement the methodology had only been applied to MBDS on a very small scale and had not been used with the current set of modern MBDS hardware.

The focus of this thesis is the verification of the performance claims made by the implementor of MBDS. These performance claims were, in fact, validated by conducting a series of relatively large-scale benchmarking experiments in which MBDS performed, generally, as predicted by its implementor. While the results are encouraging, future benchmarking experiments need to be conducted on an even larger database to examine MBDS performance under an extreme load. This will require the development of a high-speed database loading utility program which is not the focus of this thesis. Here, we report on the test databases, test transactions and test results (which constitute the benchmarks) used to verify the MBDS implementor's claims of response-time reduction and response-time invariance.



Accession For	
NT	<input checked="" type="checkbox"/>
ETI	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-i	

TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND	1
1. The Field of Study	1
2. Performance Evaluation in General	5
3. Our Area of Research	6
4. The Research Environment	6
5. The Importance of this Research	6
B. RESEARCH QUESTIONS AND OBJECTIVES	7
1. MBDS Performance Claims.	7
2. Primary and Subsidiary Research Questions	8
C. SCOPE, LIMITATIONS AND ASSUMPTIONS	8
1. The Scope	8
2. Limitations	8
a. The Test Database Size	8
b. The Block Size	10
c. The Test Transactions	11
d. The Computer-Aided Tools	11
3. Assumptions	12
D. A LITERATURE REVIEW AND THE METHODOLOGY	12
1. The Previous Research	12
2. Research Methodology	13
E. DEFINITIONS AND ABBREVIATIONS	13
1. Definitions	13
2. Abbreviations	14
F. A SUMMARY OF FINDINGS	15
G. THE ORGANIZATION OF THIS STUDY	15

II. THE MULTIPLE-BACKEND DATABASE SYSTEM (MBDS)	17
A. AN OVERVIEW OF MBDS	17
B. MBDS DESIGN FEATURES	18
1. The Backend Controller	18
2. The Communications Bus	18
3. The Backend Computers	20
4. The Database Layout	20
a. Data Placement	20
b. Clustering	21
c. The Physical Distribution of Records	21
C. THE ATTRIBUTE-BASED DATA MODEL	24
1. The Base Data	25
2. The Meta Data	25
3. The Directory	26
D. THE ATTRIBUTE-BASED DATA LANGUAGE	26
1. The Insert Request	27
2. The Delete Request	27
3. The Update Request	27
4. The Retrieve Request	28
5. The Retrieve-Common Request	28
6. A Summary of ABDL	29
E. THE MBDS HARDWARE	29
1. Generic Unix Computers	29
2. The Disk Drives	30
3. The Broadcast Bus	30
III. THE COMPUTER-AIDED BENCHMARKING SYSTEM (CABS) . .	31
A. AN OVERVIEW	31
B. THE COMPONENTS OF CABS	32
1. The Database Generator	32

a.	The Design of the Test Database	32
b.	The Record File	35
c.	The Template and Descriptor File	36
2.	The Test Transaction Mix	37
3.	The Generation of the Evaluator Reports	39
C.	HOW TO USE CABS	39
1.	Running the program	39
2.	The Output	41
a.	The Database Files	42
b.	The Transaction Mix Files	44
c.	The Report Files	44
D.	THE BENCHMARKING METHODOLOGY	45
1.	The Initial System Setup	45
2.	Operating the Test Interface	45
a.	Starting the System	45
b.	Loading the Test Database	46
c.	Conducting Performance Testing	47
d.	Collecting the Performance Data	50
e.	Exiting the Test Interface	51
IV.	THE MEASURES	52
A.	AN OVERVIEW	52
B.	THE RESPONSE-TIME REDUCTION MEASURE	52
C.	THE RESPONSE-TIME INVARIANCE MEASURE	53
V.	THE PERFORMANCE DATA	55
A.	THE BASE CONFIGURATION	55
B.	THE TWO-BACKEND CONFIGURATION	56
1.	RTR Results	56
2.	RTI Results	57
C.	THE THREE-BACKEND CONFIGURATION	58

1. RTR Results	58
2. RTI Results	59
D. THE FOUR-BACKEND CONFIGURATION	60
1. RTR Results	60
2. RTI Results	61
E. THE FIVE-BACKEND CONFIGURATION	62
1. RTR Results	62
2. RTI Results	63
F. THE SIX-BACKEND CONFIGURATION	64
1. RTR Results	64
2. RTI Results	65
G. THE SEVEN-BACKEND CONFIGURATION	66
1. RTR Results	66
2. RTI Results	67
H. THE EIGHT-BACKEND CONFIGURATION	68
1. RTR Results	68
2. RTI Results	69
I. AVERAGE PERFORMANCE BY TRANSACTION TYPE ...	70
VI. ANALYSIS AND INTERPRETATION OF THE TEST DATA	72
A. AN ANALYSIS OF MBDS RESPONSE-TIME REDUCTION ..	72
1. RTR Performance on Overhead-Intensive Transactions	72
2. RTR Performance on Data-Intensive Transactions	75
B. AN ANALYSIS OF MBDS RESPONSE-TIME INVARIANCE ..	81
1. The RTI Testing in General	81
2. RTI Performance on Overhead-Intensive Transactions	81
3. RTI Performance on Data-Intensive Transactions	83
VII. CONCLUSIONS AND RECOMMENDATIONS	86
LIST OF REFERENCES	88
INITIAL DISTRIBUTION LIST	89

LIST OF TABLES

1. TIME TO LOAD LARGE SCALE DATABASE	9
2. THE SINGLE-BACKEND RESPONSE TIMES	55
3. SINGLE-BACKEND RECORD DISTRIBUTIONS	55
4. THE TWO-BACKEND RTR PERFORMANCE	56
5. TWO-BE RTR RECORD DISTRIBUTIONS	56
6. THE TWO-BACKEND RTI PERFORMANCE	57
7. TWO-BACKEND RTI RECORD DISTRIBUTIONS	57
8. THE THREE-BACKEND RTR PERFORMANCE	58
9. THREE-BACKEND RTR RECORD DISTRIBUTIONS	58
10. THE THREE-BACKEND RTI PERFORMANCE	59
11. THREE-BACKEND RTI RECORD DISTRIBUTIONS	59
12. THE FOUR-BACKEND RTR PERFORMANCE	60
13. FOUR-BACKEND RTR RECORD DISTRIBUTIONS	60
14. THE FOUR-BACKEND RTI PERFORMANCE	61
15. FOUR-BACKEND RTI RECORD DISTRIBUTIONS	61
16. THE FIVE-BACKEND RTR PERFORMANCE	62
17. FIVE-BACKEND RTR RECORD DISTRIBUTIONS	62
18. THE FIVE-BACKEND RTI PERFORMANCE	63
19. FIVE-BACKEND RTI RECORD DISTRIBUTIONS	63
20. THE SIX-BACKEND RTR PERFORMANCE	64
21. SIX-BACKEND RTR RECORD DISTRIBUTIONS	64
22. THE SIX-BACKEND RTI PERFORMANCE	65
23. SIX-BACKEND RTI RECORD DISTRIBUTIONS	65
24. THE SEVEN-BACKEND RTR PERFORMANCE	66
25. SEVEN-BACKEND RTR RECORD DISTRIBUTIONS	66
26. THE SEVEN-BACKEND RTI PERFORMANCE	67
27. SEVEN-BACKEND RTI RECORD DISTRIBUTIONS	67
28. THE EIGHT-BACKEND RTR PERFORMANCE	68
29. EIGHT-BACKEND RTR RECORD DISTRIBUTION	68

30. THE EIGHT-BACKEND RTI PERFORMANCE	69
31. EIGHT-BACKEND RTI RECORD DISTRIBUTIONS	69
32. AVERAGE RTR PERFORMANCE TIMES	71
33. IDEAL RTR PERFORMANCE TIMES	71
34. AVERAGE RTI PERFORMANCE TIMES	71

LIST OF FIGURES

1. Conventional Mainframe Approach	2
2. Single-Backend Approach	2
3. Multi-Backend Approach	3
4. Clusters of Records	22
5. The Structure of the Test Database	38
6. RTR Performance on Transaction #1	73
7. RTR Performance on Transaction #6	74
8. RTR Performance on Transaction #2	76
9. RTR Performance on Transaction #3	77
10. RTR Performance on Transaction #4	78
11. RTR Performance on Transaction #5	79
12. RTR Performance on Transaction #7	80
13. Overhead-Intensive RTI Performance Analysis	82
14. A Comparison Chart of Data-Intensive RTI Performance	84
15. An Analysis of Data-Intensive RTI Performance	85

I. INTRODUCTION

A. BACKGROUND

1. The Field of Study

There was an early warning about the onset of a revolution in the field of database management provided by [Ref. 1]:

Database machines are coming, database machines are coming!

After more than a decade of research and development, it is clear that database machines are here to stay. *Database machines* are special-purpose computers (usually mini-computers) which provide on-line database management services and control their own secondary storage (disk) systems. Database machines have private, high bandwidth I/O subsystems which permit these special-purpose computers to perform the data-intensive (I/O intensive) functions associated with database management in an efficient and uninterrupted manner. By comparison, the general-purpose mainframe computer is geared towards the computation-intensive execution of multiple programs (processes). In addition to sharing the CPU, the processes in a mainframe system also share (*and compete for*) data channels and secondary storage resources. As a result, *mainframe-based database systems* are crippled by the shared nature of the system resources. Figure 1 depicts the conventional mainframe based database approach.

Database machines are ordinarily connected to one or more host (*frontend*) mainframe computers via a two-way communications link. In such a configuration, the database machine is known as a *backend* processor. There are two basic database machine configurations -- single-backend and multiple-backend. Figures 2 and 3 depict both database machine topologies. Both configurations move the database management function from the busy frontend mainframe computer to the dedicated backend database machine. The frontend

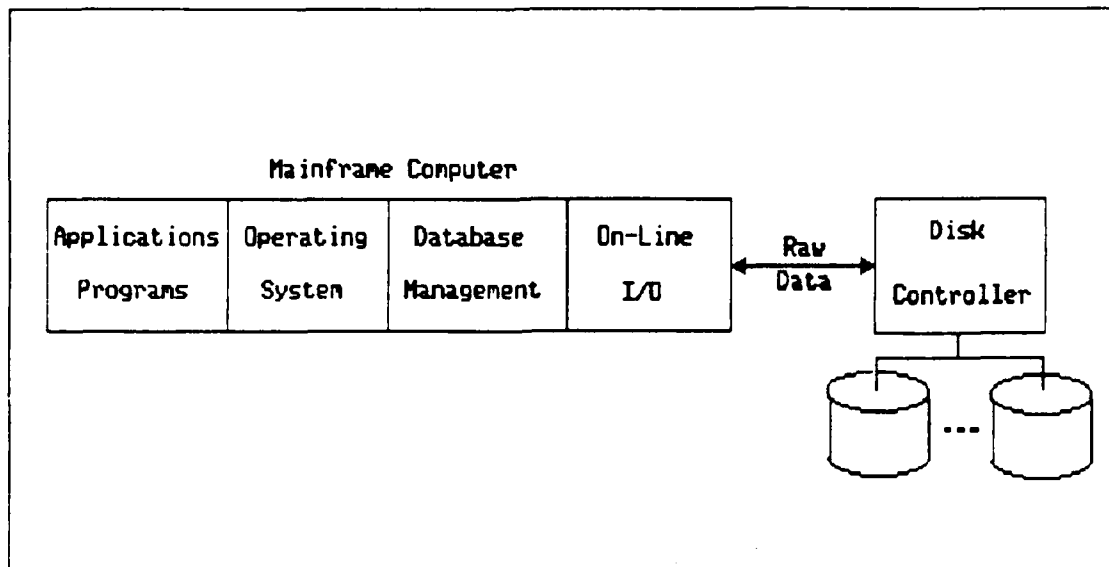


Figure 1. Conventional Mainframe Approach

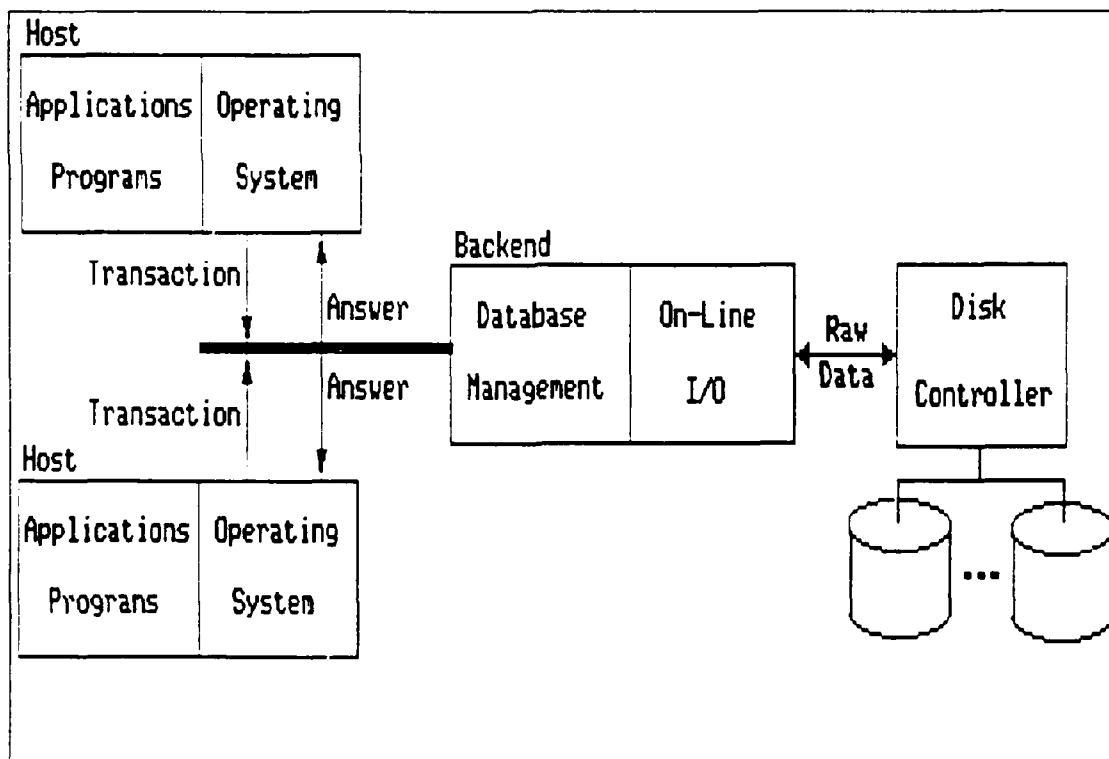


Figure 2. Single-Backend Approach

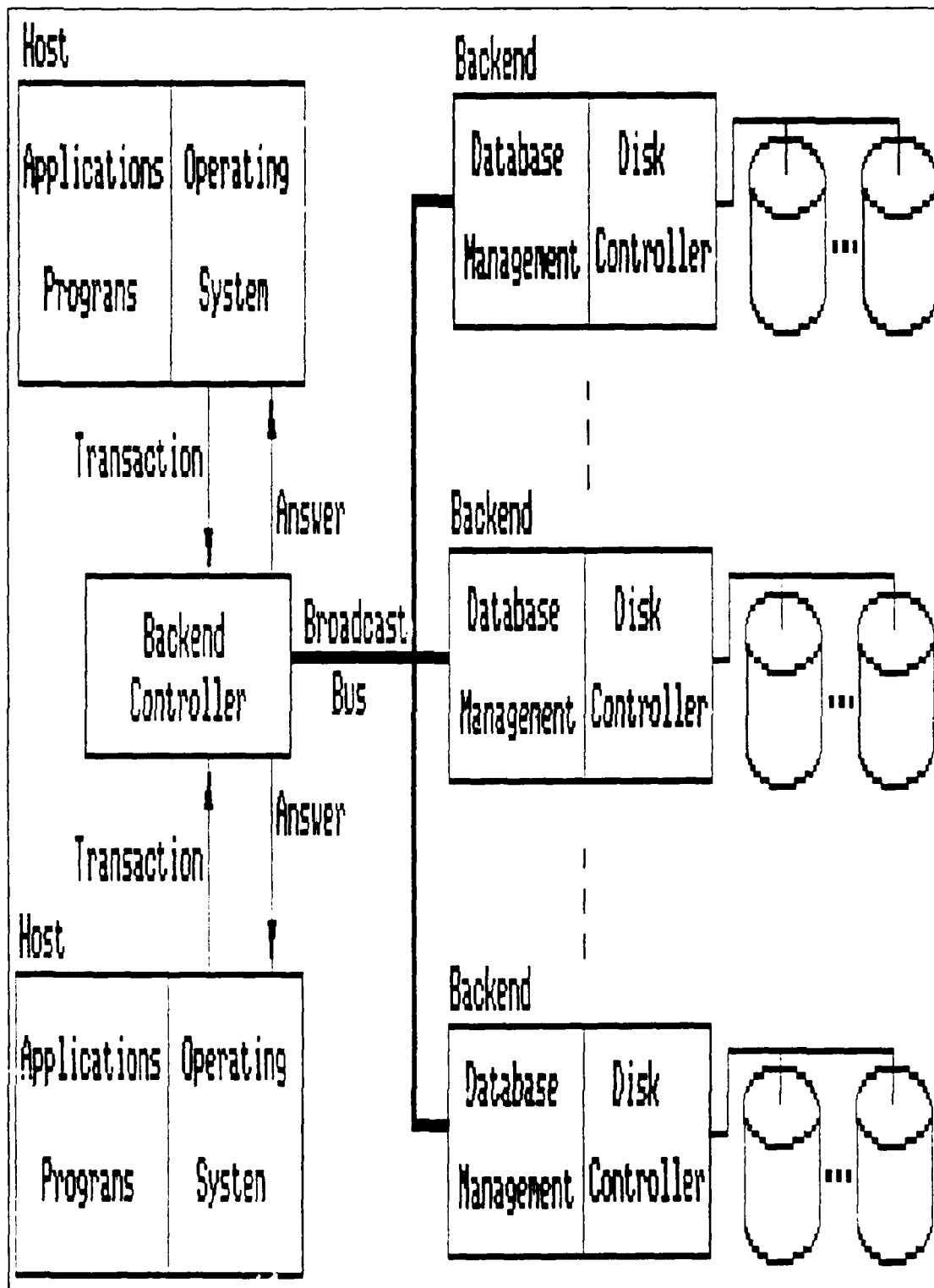


Figure 3. Multi-Backend Approach

computer receives user requests in the form of database transactions which are transmitted to the database machine. These transactions are processed by the database machine which returns the answer for a given transaction to the user over the communications link. Although a *single-backend database machine* is a far more efficient database manager than a conventional mainframe-based database system, the performance of a single-backend system degrades predictably as the database size increases (much like a conventional mainframe-based database system). This analogy between a single-backend and mainframe-based system is also true in the area of performance upgrades. The only performance upgrade option is to buy the next generation of hardware or software. This results in relatively low performance increases on the order of 10-20% for a premium price. Additionally, performance increases of this small magnitude are consumed quickly by any database growth.

Multiple-backend database machines have the potential for both high-performance transaction processing and large capacity growth. Increasing the number of backends and distributing the database evenly over each of the backends should demonstrate predictable performance gains and capacity growth in a multiple-backend system. In order to determine the performance capabilities of a given multiple-backend configuration, the database administrator must have some empirical method for measuring system performance. The performance evaluation of multiple-backend database machines is more complex than the performance evaluation of either the conventional mainframe-based system or the single-backend system. This study is the performance evaluation of the experimental ***Multi-Backend Database System (MBDS)*** which is under development in the Laboratory for Database Systems Research of the Naval Postgraduate School.

2. Performance Evaluation in General

Although computer software engineering techniques have evolved and improved over time, software engineering remains more of an "art" than an engineering discipline. In other engineering disciplines, performance specification and evaluation are more integral parts of the design process. Automotive engineers, for instance, can establish the performance specifications for a sports car early in the design process, assemble the necessary components and measure the car's performance with standardized tests (i.e., horsepower, cornering force, 1/4 mile elapsed time). Unfortunately, computer software engineers tend to handle this process differently. Computer software performance evaluation is often a separate, postdevelopment process. Modern software engineering techniques focus on computer program correctness, modularity and maintainability rather than focusing on efficiency. This difference is due in large part to the enormous (and growing) demand for software, the overall performance increases of the computer hardware, the many problems associated with software maintenance and the need for reliable application software. *If a program doesn't work, it doesn't matter how fast it runs* [Ref. 2].

Modern computer hardware speed and capacity has served to mask the inefficiencies of our software by allowing the programs to run *fast enough* to be acceptable to users. This does not make computer system performance evaluation a dead issue. Computer system (system, for short) performance evaluation is a growing subdiscipline of computer science. Performance evaluation has important applications in the following areas:

- **Procurement** -- The establishment of measurable selection criteria.
- **Improvement** -- The modification of an existing system to improve performance.
- **Capacity Planning** -- The modeling of changes to a system and/or its workload to determine whether or not the system can support the changes.
- **Design** -- The verification of system design claims.

These performance evaluation categories were originally identified in [Ref. 3] and [Ref. 4]. Complete hardware/software systems or individual system components can be the targets of performance evaluation studies.

3. Our Area of Research

This study is the actual application of a performance evaluation technique known as *benchmarking* to an experimental database management system (DBMS) known as the Multi-Backend Database System (MBDS) in an attempt to verify the implementor's performance claims. Because the system under study is a DBMS, the research is in the general area of database systems and machines. Since the DBMS under study employs a parallel and expandable computer architecture, this research is also in the area of parallel processing by computers.

4. The Research Environment

The research was conducted in a controlled environment of the NPS Laboratory for Database Systems Research on a network of modern UNIX workstations. The research was aided by using an existing set of computer-aided design (CAD) tools for performance evaluation known as the Computer-Aided Benchmarking System (CABS). The DBMS under study is the experimental Multiple Backend Database System (MBDS). The maintenance and on-going development of MBDS is conducted by a team of professional programmers. Since MBDS is still under development, a working version was set aside for the benchmarking experiment to prevent side-effects from any enhancements made to MBDS during the testing process.

5. The Importance of this Research

This study is primarily in the performance evaluation category of design verification. The focus of MBDS development has primarily been on software correctness and not on software performance. As a result, MBDS performance had never been tested to any great extent. Therefore, verification of the system's design goals in the areas of performance and capacity growth

was important to the implementors and supporters of MBDS. The results of this study also have significant implications for MBDS in the remaining areas of performance evaluation. The results could be used to evaluate the cost effectiveness of an actual implementation of MBDS, to identify any bottlenecks in MBDS for future optimization efforts or to extrapolate the MBDS capacity for database growth. This research is based on the practical application of a previously developed and largely untested CAD benchmarking methodology. The utility, generality and applicability of CABS as a CAD tool for use in future performance evaluation efforts became apparent through extensive use of CABS during this research.

B. RESEARCH QUESTIONS AND OBJECTIVES

1. MBDS Performance Claims.

As an expandable computer, MBDS may employ one or more database backends. MBDS was designed to exploit the opportunities for parallel processing presented by distributing the database over more than one backend. The following is a summary of the performance claims that have been made by the designer and implementor of MBDS:

- **Response-Time Reduction (RTR).** The response-time reduction of a transaction is inversely proportional to the multiplicity of the backends. This means that as the number of backends increases, the response-time reduction for a given transaction is expected to improve (e.g., moving from one backend to three backends should yield a response-time which is just one-third the original response time).
- **Response-Time Invariance (RTI).** The response-time invariance of a transaction in response to the increase of the database size is maintained by a corresponding increase in the multiplicity of the backends (e.g., when the database doubles in size, doubling the number of backends will yield the original response-time).

2. Primary and Subsidiary Research Questions

The primary research question is to determine to what degree MBDS demonstrates the response-time reduction and/or the response-time invariance. The subsidiary research questions of determining an appropriate test database, test transaction set and test configurations were already solved by the designers and implementor of CABS. Since CABS provided the tools necessary to benchmark MBDS, the objectives of this research were to learn how to use CABS, to learn how to load various configurations of MBDS, to run the benchmark experiments methodically and then to interpret the results.

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

1. The Scope

The scope of the thesis covers the following three areas:

- *Application* of a benchmarking methodology to a configurable computer with a variable number of parallel database processors and stores.
- *Collection* of benchmarking results from the parallel database computer under various transaction loads.
- *Interpretation* of the performance data and correlation of the data with the predicted performance gains and growth capacity.

2. Limitations

a. *The Test Database Size*

The original intent of this study was to test a very large-scale database with as much as *300 megabytes* of base data per backend. The capability of generating such a database was central to the CABS methodology. Unfortunately, neither the designers nor the implementor of CABS considered the *time* necessary to actually build a database of this size -- record by record. At the outset of this study, the MBDS insertion rate was only one record every eight seconds. The insertion process is complex and had never been optimized since MBDS is still under development. Because the scale of the desired

database would require a very rapid record insertion rate, considerable time was spent in optimizing the insert process before starting the testing. A rate as high as two records a second (for small records) was achieved as the result of the optimization effort. This was a marked improvement but what was needed an improvement of several orders of magnitude.

Even at nearly two records a second, the test database could be built at a rate of only one megabyte per hour. The time necessary to load a single 300 megabyte configuration of MBDS would take nearly *two weeks* of constant computer processing time. The CABS methodology calls for three database sizes with 15 different test configurations each. Each database must be loaded a minimum of nine times to set up the test configurations. The non-stop computer processing time required for a full test is shown in Table 1.

TABLE 1. TIME TO LOAD LARGE SCALE DATABASE

<u>Database size</u>	<u>Computer Time</u>
Large (300 Mb)	9 * 2 weeks
Medium (150 Mb)	9 * 1 week
Small (70 Mb)	9 * 1/2 week
	31.5 weeks !!!

Clearly, the amount of time necessary to load a very large-scale database was prohibitive, so the decision was made to reduce the maximum database size per backend to 30 megabytes. Unfortunately, this goal proved unobtainable as well, when a peculiar system problem occurred six to seven hours into the lengthy loading process. At this point in the loading process,

there is a tremendous and inexplicable slowing of MBDS processing by the UNIX operating system. The insertion of records continues correctly but at a very slow rate of about *one record a minute*. This phenomena occurred every time a long duration load was attempted.

The slowing of the insert process is strictly the result of lengthy computer processing time rather than an actual degradation in the performance of MBDS as the database grows. The fact that the MBDS insertion rate remains constant for a given record length was demonstrated by stopping the slowed processes manually and then restarting MBDS. The system would start inserting at the normal rate, adding to the seven plus megabytes of data already loaded. A problem associated with stopping and restarting the system was to determine exactly which record was the last record successfully inserted so that the input file could be altered for reuse. The benchmarking methodology depends on a specific number of records which made the restart procedure time consuming since missing or duplicate records would be unacceptable. As a result of these technical limitations, the decision was made to go forward with only the smallest of the three database sizes provided for testing by CABS. In Chapter III, we discuss CABS in more detail. The small database amounts to nearly seven megabytes of base data per backend and consists of more than 30,000 records per backend. The database size chosen was large enough to put MBDS through its most comprehensive test to date and to provide a clear indication of MBDS performance characteristics.

b. The Block Size

The block size is an integral part of the way MBDS distributes the database. The block size assumed by the CABS methodology was 2000 bytes. Unfortunately, during the optimization process for the INSERT transaction, this assumption was overlooked. The block size is a system constant which can be adjusted by changing the value and recompiling the MBDS source code. The optimum block size arrived at for insertions was eight kilobytes, the basic unit of communication between backends and the

secondary storage devices. The problem is that the larger block size caused MBDS to load the test databases more unevenly than if the smaller block size was used instead. The CABS methodology creates so many clusters (many nearly empty anyway) that the problem was not immediately noticeable and it was not discovered until near the end of the study. Even with the correct block size, MBDS loads the database unevenly. A test was conducted to assess the impact of this oversight. The system performance was virtually even (with the large block size slightly faster), but the insertion rate was much slower. The decision was made to complete the testing with the eight kilobyte block size.

c. The Test Transactions

The original intent was to evaluate all five of the database request types: INSERT, DELETE, RETRIEVE, RETRIEVE-COMMON and UPDATE. However, UPDATE and RETRIEVE-COMMON were not available for this study because these transactions were undergoing modification and optimization during the testing. Additionally, during the course of loading the test databases, it became apparent that testing the INSERT transaction was unnecessary. After literally logging days of constant INSERT transaction processing, the performance of the MBDS INSERT transactions was constant for a given record size.

d. The Computer-Aided Tools

The actual version of CABS available did not provide all of the reports and files that the system is supposed to produce (see Chapter III). Most of the reports generated are based on the incorrect assumption of perfect database record distribution which renders them useless. The most serious flaw discovered in the CABS output was the MBDS descriptor file which is the basis for the clustering and database distribution. The file generated did not implement the CABS design methodology due to a complex programming error. The file generated was huge and grouped all of the records into two record clusters. This presented a lengthy setback in the testing process.

Finally, the logic error was discovered and then corrected by the MBDS system programmers. The current version of CABS generates the correct descriptor files.

3. Assumptions

Since the objective of this benchmarking effort was the design verification of MBDS, the system was always tested in a best-case scenario. For the purpose of this study, best-case means that the only load on the frontend and backend computers was MBDS and that the database was always as evenly distributed as possible. Modeling real-world processing loads and database loads and database distributions is beyond the scope of this study and the capabilities of CABS. However, the make-up of the artificial database generated by CABS is a challenging test for MBDS and is discussed in more detail in Chapter III.

D. A LITERATURE REVIEW AND THE METHODOLOGY

1. The Previous Research

Over the past six years, a considerable amount of thought and work has gone into the development of the methodology and CAD tools used in this study. Below is a chronological list of previous work with a brief summary of the contribution of the work:

- *Performance Evaluation Tools for a Multi-Backend Database System* by Joseph G. Kovalchik, December 1983. This thesis presents a design foundation for a set of ideal database performance evaluation tools.
- *A Methodology for Benchmarking Relational Database Machines* by Paula R. Strawser, March 1984. This dissertation presents standards for the performance evaluation of relational database machines.
- *Internal and External Performance Measurement Methodologies for Database Systems* by Robert C. Tekampe and Robert J. Watson, June 1984. This thesis documents the actual instrumentation of MBDS for internal and external performance evaluation. A small scale benchmarking experiment was performed manually with promising results. Recommendations were made for automated benchmarking tools and a more complete system test.

- *A Performance Measurement Methodology for Software Multiple-Backend Database System* by James R. Vincent, June 1985. This thesis laid the design groundwork for the automated generation of test database and test transaction sets.
- *A Computer Aided Design for the Generation of Test Transactions and Test Databases and for the Benchmarking of Parallel, Multiple-Backend Database Systems* by George Patrick Fenton, Jun 1986. This thesis was the actual implementation of CABS.

2. Research Methodology

Beyond the papers listed above (many of which reference each other), a library subject catalog search was conducted with limited results. Few books exist on computer system performance evaluation in general and database system performance evaluation in particular. [Ref. 3] and the updated [Ref. 4] were useful and are considered standard references in the field of performance evaluation.

E. DEFINITIONS AND ABBREVIATIONS

1. Definitions

- Backend -- A backend processor (or just *backend*) is a dedicated set of computer hardware (including secondary storage) and software used to accomplish specialized tasks, in this case database management. Backend processors are connected to frontend computer which typically perform the routine tasks (such as user interface) and control access to the backends.
- Base Data -- The actual records which makeup the database.
- Benchmark -- A performance measurement which is repeatable and provides a standard for comparison.
- Bottleneck -- The computer system component which is the limiting factor for the entire system. Given the correct circumstances, hardware performance (especially secondary storage), software performance and communications capacities are all candidates for the label of system bottleneck.
- Cluster -- A group of logically related records. An individual record can only map to a single cluster of records.

- **Configure/Configurable** -- The assembly of a set of hardware and associated software to create a new version of a computer system. MBDS is said to be configurable because any number of backends can be added to makeup a particular implementation of the system.
- **Database Machine** -- Special-purpose computers (usually mini-computers) which provide on-line database management services and control their own secondary storage (disk) systems.
- **Data-Intensive** -- A database query transaction which accesses a large percentage of the base data. Since access to secondary storage is very slow when compared to computer processing speed, the time spent accessing the base data is the limiting performance factor, rather than the time spent processing the transaction (looking up the records for the system to access).
- **Frontend** -- The frontend computer is the machine between the user and the backend processors. The frontend has minimal processing responsibilities which are normally limited to relaying user transactions to the backends and backend responses to the user.
- **Meta Data** -- Information about the database which includes indexing data and record format data.
- **Overhead-Intensive** -- A database query transaction which accesses a very small percentage of the base data. The time spent processing the transaction becomes more significant and could become the limiting performance factor.
- **Response Time** -- The elapsed time from the instant a database query is released for processing to the instant the system is ready to report a result. Normally, the time necessary to actually display the result is not included, because this display time could vary greatly.

2. Abbreviations

- **ABDL** -- Attribute-Based Data Language
- **CABS** -- Computer-Aided Benchmarking System
- **DBMS** -- Database Management System
- **MBDS** -- Multi-Backend Database System
- **RTI** -- Response-Time Invariance
- **RTR** -- Response-Time Reduction

F. A SUMMARY OF FINDINGS

While CABS proved useful, the lack of a rapid database loader for CABS limited the size of the test database. Even with a scaled-down database, the performance characteristics of MBDS were clear. MBDS demonstrates strong response-time invariance and response-time reduction. This is especially true for data-intensive transactions since any added overhead due to communications with additional backend processors becomes insignificant. These findings were expected and originally thought to be heavily dependent on perfectly even loading of the backend computers. Due to random selection of the first backend for a given cluster of records, perfectly even loading of the backend computers was not always possible. Surprisingly, nearly ideal performance was still achievable with fairly uneven loading. This gave insight into the performance achievable with a real database which would seldom, if ever, be completely evenly loaded.

G. THE ORGANIZATION OF THIS STUDY

The remainder of this thesis is organized as follows:

- **In Chapter II, we expand on MBDS.** A more detailed description of the target computer on which the benchmarks were performed.
- **In Chapter III, we elaborate on CABS.** A detailed description of the benchmarking methodology and tools used to collect the benchmarking data.
- **In Chapter IV, we establish the measures.** A description of the performance-gain and capacity-growth measures in terms of response-time reduction and response-time invariance, respectively, is given.
- **In Chapter V, the test data are presented.** Tabulation and presentation of the data collected by transaction type and backend multiplicity are included.
- **In Chapter VI, data analysis and interpretation are given.** The correlated test data is related to the response-time reduction and response-time invariance performance claims made about MBDS.

- **In Chapter VII, we make our conclusions and recommendations.**
Summarizes the benchmarking experience and indicates future work.

II. THE MULTIPLE-BACKEND DATABASE SYSTEM (MBDS)

A. AN OVERVIEW OF MBDS

The target system of this study is the prototype database machine known as the *Multiple-Backend Database System* (MBDS) which is under development in the Laboratory for Database Systems Research of the Naval Postgraduate School (NPS). MBDS is a configurable database machine which employs one or more backend computers which are connected parallelly via a broadcast bus. This approach to database machine design is known as the *software multiple-backend approach*. This approach is described in [Ref. 5]:

These backends have identical and replicated software and their own disk systems. In a software multiple-backend configuration, there is a backend controller (i.e., master) which is responsible for supervising the execution of database transactions and for interfacing with the hosts and users. The backends (slaves) perform the database operations with the database stored on the disk systems of the backends ... Users access the system either by way of the hosts or through the controller directly.

The ability of MBDS to share the workload among backend computers is the source of potential performance gains and capacity growth in proportion to the number of backends. MBDS was designed to be easily expandable. This design goal is realized through the use of generic computer hardware (UNIX machines) which permits the use of identical software on each backend slave computer. This means there is no new programming requirement when adding backend computers to MBDS. Since special-purpose backend computers are not required, it is possible to use different models of a given generic class of computers for MBDS.

B. MBDS DESIGN FEATURES

1. The Backend Controller

While it is possible to configure a single backend implementation of MBDS, the more effective and interesting MBDS configurations consist of two or more backend computers. Further, one machine acts as the backend controller and the remaining machines manage equal portions of the database. The design of the MBDS backend controller is described by [Ref. 5]:

The verall design goal of a backend controller should focus on minimizing the work done by the controller. The controller receives a user transaction either from a host or through a terminal and sends the transaction to all of the backends for execution. The controller also collects all of the results produced by the backends for the user transaction and routes the results to the host or to the terminal. As such, the controller becomes a prime candidate for the bottleneck of the system. By minimizing the work of the controller, and by offloading all of the database management operations to the backends, the controller may reduce the possibility of becoming the system bottleneck. Overall, the functions of the controller are reduced to the *pre-processing of the user transactions*, the *post-processing of the transaction results*, the *sending and receiving of data* from the backends and hosts, and the *arbitration of data insertion* into the database.

User database transactions are reformatted during pre-processing and placed on the broadcast bus. They are transmitted or *broadcast* to all of the backends simultaneously. The post-processing function combines the records received from the backends in response to a transaction and performs any aggregate operations (AVG, SUM, etc.) requested before forwarding the complete results to the user.

2. The Communications Bus

MBDS uses a *broadcast bus* for the communications chores primarily to ensure easy expansion of the number of backends. The task of adding another backend to a given system requires the simple connection the backend computer's communications transceiver to the local-area network which is connecting the MBDS machines. Other bus topologies are conceivable, but each of them has been disqualified due to its drawbacks as it is applied to a

parallel and expandable architecture like MBDS. For instance, the standard token-ring type of bus does permit easy expansion but as the number of backends increases, the delay in the message passing also grows. Another alternative is a point-to-point or direct connection from the controller to the backends and from each backend to all the other backends. This approach offers the highest speed and capacity potential. However, expansion is limited to the number of physical connection points provided on the computers. For these reasons, the broadcast bus was selected for MBDS.

By using a broadcast bus, MBDS can achieve parallel execution of user transactions. When the controller puts a message on the bus for the backends, the message is *broadcast* to all the backends at once and is received by the backends almost simultaneously. This form of controller-to-backend communication is very fast when compared with the overhead involved in point-to-point communications, which is also in keeping with the MBDS design goal of minimizing the work done by the controller. The backends can make use of the broadcast bus to communicate with other backends when necessary and, mainly, to return the (partial) result of transactions to the controller for postprocessing. Fortunately, the answers to database queries tend to be much smaller than the amount of base data actually accessed so congestion of the bus by the backends is unlikely. The majority of the data is accessed over the backend's own internal, high speed data bus connected to the secondary storage. Even in the worst-case scenario of an exhaustive search or involved merge operations where the broadcast bus would very likely be loaded with output from the backends, the broadcast bus remains acceptable. This is because some delay is expected by the user in such an operation from which the user is expecting considerable operations and results from the system. Ethernet, the industry standard broadcast bus, has a data transfer rate of ten megabits per second which provides a fairly substantial bandwidth. If required by the users or sheer size of the database, higher bandwidth broadcast buses do exist. Such a bus would increase the transfer rate enormously and shift the limiting factor to the speed and memory size of the controller.

3. The Backend Computers

The backends are the workhorses of the system, faithfully listening to the broadcast bus for the next transaction which is executed at the earliest opportunity as rapidly as possible. Each of the backends operates only on its portion of the database and returns a partial result for assembly by the controller. The design of the MBDS backends is described by [Ref. 5]:

... the backends of the system all have *identical software* to allow replication of the software on a new backend. Additionally, the backends must have *complete software* to perform all of the database management functions. These functions include *directory management, concurrency control, record processing, and communications*. The directory management function is responsible for managing indices, calculating record clusters, allocating the secondary-storage addresses for record insertion, maintaining secondary-storage tables of indices, cluster numbers, and addresses, processing transactions against the directory tables, and providing record addresses for subsequent database access operations. The concurrency control function oversees various accesses to the directory tables and the user data facilities the concurrent execution of transaction. The record processing function is used to stage the user data from the secondary storage to the primary memory, to process the staged data, to store data onto the secondary storage, and to return the responses to the controller. Finally, there are communication functions in each backend to control communications among backends and between the backend and the controller. It is necessary to minimize the communications among backends, in order to reduce the communications traffic among them.

4. The Database Layout

a. Data Placement

In order to realize the full performance potential of MBDS, the database must be spread out evenly across the backends. The design of the MBDS database is described by [Ref. 5]:

In a multi-backend database system, a database must be placed on the secondary storage in such a way so that all of the subsequent accesses to the database will result in *block-parallel-and-record-serial operation*. In other words, all of the backends are accessing, in parallel, the secondary-storage blocks of the same database in their respective disk systems, although the records in the blocks which may satisfy the same transaction or different transactions are being accessed by the backends serially. Thus, the issue really focuses on how to ensure an even distribution of the

user database across the disk systems of the backends. Such a distribution requires a *data placement algorithm*. To achieve an even distribution of data, there must be a processor in the multi-backend database system that is responsible for overseeing the record-insertion process. The controller has an overview of the entire system, and is the logical choice for arbitrating the record insertion process, i.e., controlling the data placement.

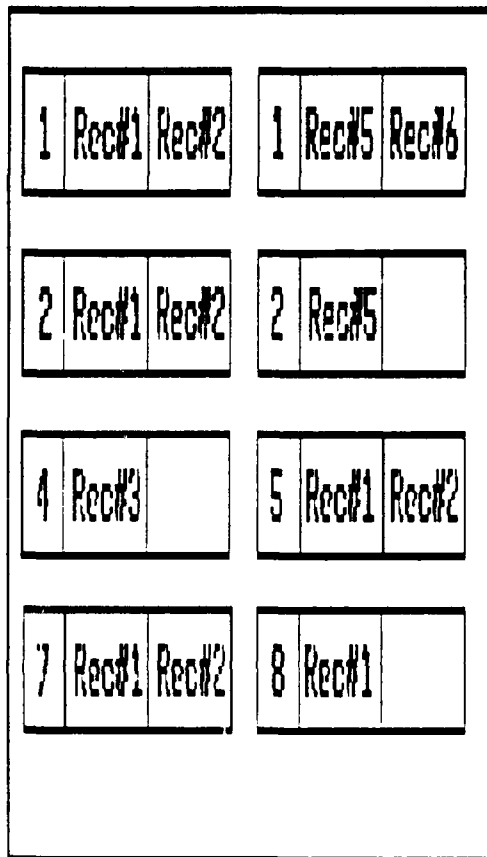
b. Clustering

To achieve block-parallel-and-record-serial operation, MBDS partitions logically related records into *clusters of records* (often referred to as just *clusters*). The *clusters*, in turn, are made up of one or more *blocks* of storage space. A *block* is a preset unit of secondary storage (disk) space which must be large enough to contain the largest record defined in the database, but small enough to allow fragmentation of the cluster into multiple blocks. When the first record of a given cluster is inserted, the data placement algorithm in the controller selects the starting backend for the cluster at *random* and the backend allocates one block of disk space in which to store the record. This block becomes the *active block* of the cluster. As more record insertions into the cluster occur, the active block will eventually run out of free space and the backend with the active block must notify the controller. The controller arbitrates this by simply sending the insert transaction to the next backend in a round-robin manner. Figure 4 depicts the concepts of clusters, blocks and records.

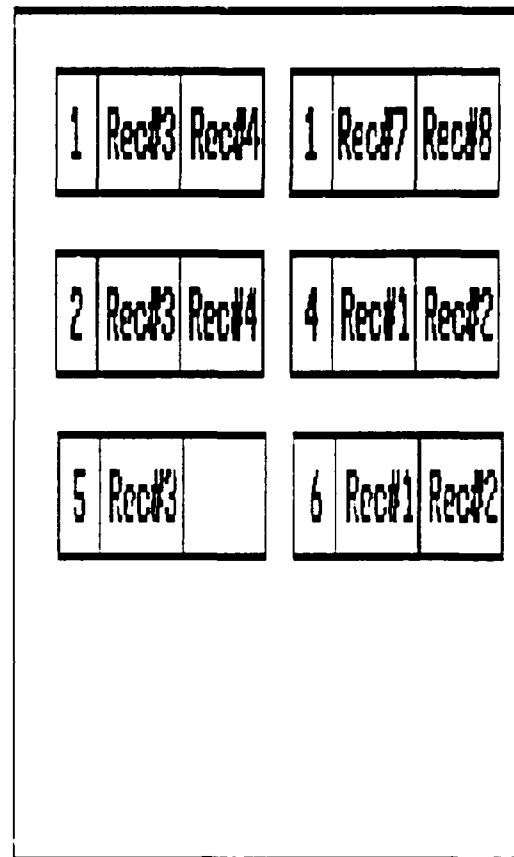
c. The Physical Distribution of Records

With an appropriately small block size, this data placement (clustering) methodology assures a *fairly* even distribution of each cluster of records over the backends. As shown in Figure 4, this clustering methodology can easily cause a certain amount of uneven loading. This is especially true when the block size is much larger than the record size or the number of records in a given cluster is small. This uneven loading phenomena is virtually unavoidable because of the random selection of the first backend of each cluster and the variable number of records possible in each cluster.

Backend #1



Backend #2



Legend:

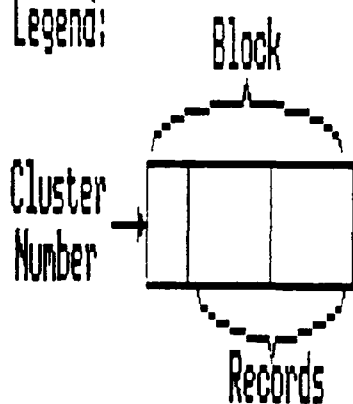


Figure 4. Clusters of Records

Because the selection of the first backend of a given cluster is a random decision, it is possible that certain backends could be selected more often than others. This is especially true in a computer-based system because there is not a *true* random number generator available to the computer programmer. MBDS uses a *pseudo*-random number generator from the UNIX system function library and like any good pseudo-random number generator it tends to pick, on the average, a number near the middle of the range involved (usually zero to unity, so *one half* should be the mean). This has a noticeable effect on the distribution of records. In practice, the controller does tend to pick certain backends for the starting block of a cluster more often than the rest of the backends. Even if the selection of the first backend for each cluster was perfectly fair, the *only time* a cluster of records can be distributed perfectly evenly over all of the backends is the special case with the following characteristics:

- All of the blocks in the cluster are full.
- The total number of blocks in the cluster can be evenly divided by the number of backends (e.g., 30 blocks could be evenly distributed over three backends, but not four backends).

In order to realize perfectly even loading of each of the backends, this special case must be true for *every cluster of records in the database*! Thus, it makes some uneven loading inevitable in any realistic database.

These problems are exacerbated by an excessively large block size. Conversely, these problems are minimized by small block sizes and large numbers of records. Since MBDS is designed to support very large databases and the block size is under the control of the database administrator, these problem areas have little effect on the application of MBDS to a carefully designed database. However, the potential for uneven loading exists and must be addressed early in the database design process. It is interesting to note that no previous study has evaluated the effectiveness of the data placement algorithm. This is especially true of the design of the CABS database where the impression is given that perfectly even loading is both easily accomplished

and necessary for optimal performance. This discussion shows that a perfect distribution must be contrived and would be quite difficult to accomplish. See Chapters III and V for discussion on the database distribution created by CABS. See Chapter VI for discussion on the performance costs of uneven loading.

C. THE ATTRIBUTE-BASED DATA MODEL

The database model used by MBDS is the attribute based data model. The *attribute-based data model* provides a high-level abstraction which permits the user to view the logical properties of the database while concealing implementation details of the database and DBMS. A concise description of the attribute-based data model from [Ref. 5] is provided below:

In the *attribute-based data model*, the data is considered in the following constructs: database, file, record, attribute-value pair, keyword, attribute-value range, directory keyword, non-directory keyword, directory, record body, keyword predicate, and query. Informally, a *database* consists of a collection of files. Each *file* contains a group of records which are characterized by a unique set of directory keywords. A *record* is composed of two parts. The first part is a collection of *attribute-value pairs* or *keywords*. An attribute-value pair is a member of the Cartesian product of the attribute name and the value domain of the attribute. As an example, <POPULATION,25000> is an attribute-value pair having 25000 as the value for the population attribute. A record contains at most one attribute-value pair for each attribute defined in the database. Certain attribute-value pairs of a record (or a file) are called the *directory keywords* of the record (file), because either the attribute-value pairs or their attribute-value ranges are kept in a *directory* for identifying the records (files). Those attribute-value pairs which are not kept in a directory are called *non-directory keywords*. The rest of the record is textual information, which is referred to as the *record body*.

These constructs make up the two kinds of data used by MBDS, base data and meta data.

1. The Base Data

The actual files of records make up the bulk of the data stored by MBDS. The records are stored in secondary storage as a collection of attribute-value pairs followed, optionally, by a variable-length record body which is just textual information attached to the record by the user (Note: this record body feature has not been implemented). An entire block of records for any given cluster is stored together in contiguous disk space. The contents of an example record is depicted below:

(<FILE,CardFile>,<TITLE,PC_World>,<MAG_NO,2356>,
{Windows evaluation})

The convention for denoting an attribute-value pair is enclosing the attribute name and value in angle brackets (e.g., < name, value >). The convention for denoting the textual record body is enclosing the text in curly braces. The concept of an attribute-value pair is equivalent to the conventional database management and data processing term *a field of a record* or just *field*. The term *the field name* is equivalent to *the attribute name* and the term *the field value* is equivalent to *the attribute value*. The first attribute-value pair (FILE) in every record of a given file of records is the file name and is identical to the rest of the records in the file. This first attribute value pair is mandatory and permits a first, coarse partitioning of the database by file name.

2. The Meta Data

MBDS must store enough information about the base data so that the records can be accessed quickly and directly. Information about base data is termed *meta data*. MBDS meta data is made up of *attributes*, *descriptors* and *clusters*. All of the meta data about the database makes up the *directory* of the database. *Attributes* represent the different possible types of base data. *Descriptors* are used to describe the *specific values* (Type-A descriptors) or *ranges of values* (Type-B descriptors) that an attribute can assume. When more than one descriptor is specified for a given attribute, the descriptors must be mutually exclusive. *Clusters* are groups of logically related records. Clusters

are identified by a *set* of different possible descriptor values or value ranges, although the minimum number is just one (the FILE descriptor described above). In order for a record to map to a cluster it must satisfy all of the descriptors specified for the cluster. Because the descriptors are mutually exclusive, the total number of clusters possible can be determined by calculating the Cartesian product of the number of different possible values or value ranges defined for all the descriptors. In effect, the database is *partitioned* into completely separate equivalence classes. This permits the user to specify the desired clusters exactly and the access precision is improved.

3. The Directory

The directory is made up of three user-defined tables:

- The Attribute Table (AT)
- The Descriptor-to-Descriptor-Id Table (DDIT)
- The Cluster Definition Table (CDT)

The attribute table points or *maps* directory attributes to the descriptors defined for them in the descriptor-to-descriptor-id table. The descriptor-to-descriptor-id table associates a unique descriptor identification number to each descriptor. The cluster definition table maps each set descriptor identification numbers to a unique cluster identification number and the identification numbers of the records in that cluster.

D. THE ATTRIBUTE-BASED DATA LANGUAGE

The *attribute-based data language* (ABDL) is the native data manipulation language of MBDS which permits users to write queries to access the database. The definition of ABDL appears in [Ref. 6]. ABDL is a simple language which includes just five primary database operations: INSERT, DELETE, UPDATE, RETRIEVE and RETRIEVE-COMMON. An ABDL request is made up of a primary operation and a qualification. A qualification is a combination of keyword predicates that specifies the records in the

database to which the operation applies. An ABDL transaction is made up of two or more requests.

1. The Insert Request

The ABDL INSERT request adds new records to the database. The qualification part of the request lists the attribute-value pairs to be inserted:

INSERT (<FILE,CardFile>,Title<pcworld>,MagNo<2356>)

This request would insert a record into the FILE called CardFile with a Title attribute value of 'pcworld' and MagNo attribute value of 2356. Keywords need a value, but non-directory attributes and the record body can be omitted if there is no initial value.

2. The Delete Request

The ABDL DELETE request can be used to *mark* one or more records for deletion. MBDS finds the record(s) specified by the qualification and changes the record type code. Since the record is not physically deleted, this opens up the possibility for an UNDELETE operation and creates the need for a *garbage collection* routine. Neither of these utility, production system type operations exist yet because MBDS is still an experimental system. The following is a sample DELETE request:

DELETE ((FILE=CardFile)and(Title=pcweek))

This delete request would delete all the database records in the FILE called CardFile with a Title of 'pc week'.

3. The Update Request

The ABDL UPDATE request can be used to modify the values in the records of the database. The UPDATE request is made up of a query part and a modifier part. The query identifies the records to be changed and the modifier specifies how the records are to be changed. The following is a sample UPDATE request:

UPDATE ((FILE=CardFile) (LastUpdate = 1/13/89))

This UPDATE request would change the LastUpdate attribute value to '1/13/89' in all the records of the FILE called CardFile.

4. The Retrieve Request

The ABDL RETRIEVE request can be used to extract information from the database. The RETRIEVE request is made up of a query, a target-list and an optional by-clause. The *query* part of the request is used to identify the record(s) from which the user needs information. The query can specify a single record, a set of records, or the entire contents of a file. In response to a RETRIEVE request, each backend moves all of the records in the clusters specified by the query to main memory and then forwards only the records which completely satisfy the query (non-directory attributes can also be used in a RETRIEVE request so there is a possibility of fetching records which match on the directory attributes and later screen out because of the non-directory attribute specification in the query). An exhaustive search of the database (which is a collection of files) is considered a pathological case and is not directly supported. The user can, however, perform an exhaustive search of a given file by making the file attribute the only directory attribute in the query. The *target-list* allows the user to specify the attributes needed in the output. An empty target-list is the default which displays all of the attribute-values in the records found by the query. The *by-clause* is an optional operation which is performed by the controller during post-processing of the records retrieved. These operations are termed *aggregate* operations and include the AVG, COUNT, SUM, MIN and MAX functions. The following is a sample RETRIEVE request:

RETRIEVE ((FILE=CardFile) and (Title=pcworld) COUNT(Title))

This request would retrieve all of the CardFile records with a Title of 'pcworld' and count them.

5. The Retrieve-Common Request

The ABDL RETRIEVE-COMMON can be used for relational database operations such as merging or comparing files in the database by shared attribute-values. The format of the RETRIEVE-COMMON is given below:

RETRIEVE (Query #1)(Target-List#1)

COMMON (Attribute #1, Attribute #2)

RETRIEVE (Query #2)(Target-List #2)

This simple format of two RETRIEVE requests joined by the COMMON clause provides considerable database processing power to the user. The attributes in the COMMON clause are the attribute names shared between the first RETRIEVE request and the second RETRIEVE request. The records in each file that satisfy both the query and the COMMON clause are selected.

RETRIEVE ((FILE=CardFile) and (Keyword=MSDOS)(MagNo))

COMMON (Keyword, Keyword)

RETRIEVE ((FILE=Abstracts) and (Keyword=MSDOS)(AbstractNo))

This request would retrieve and report all the CardFile MagNo's and Abstracts AbstractNo's which have a Keyword attribute value of 'MSDOS'.

6. A Summary of ABDL

ABDL provides a complete set of the basic tools necessary to perform database management operations. The RETRIEVE-COMMON transaction in particular provides an easy mechanism for the user to specify complex, multi-file queries. ABDL is a low level language and has no procedural constructs. Presently, MBDS uses a large test interface (TI) program to parse higher-level DBMS language (like SQL) commands into a series of ABDL commands. TI also permits users to submit ABDL commands directly. This is the mode in which TI was used for this study.

E. THE MBDS HARDWARE

1. Generic Unix Computers

The computers used for this study are the Integrated Solutions, Inc. (ISI) mini-computers. Each of these machines uses a 16.67 MHz Motorola CPU and have four megabytes of main memory. The operating system used is 4.3 BSD UNIX. A total of nine ISI computers were available for this study, so the largest configuration possible was an eight-backend system.

2. The Disk Drives

All of the computers have a small, hard disk drive to support the UNIX file system. In addition to this UNIX-system disk drive, each backend computer has two hard disk drives dedicated to supporting MBDS operations. The "small" disk is used to store the MBDS meta data and has a formatted capacity of 100 megabytes. The "big" disk is used to store the MBDS base data and has a formatted capacity of 400 megabytes. These disk drives are connected to the high-speed, internal VME data bus of their respective backend computers.

3. The Broadcast Bus

In keeping with the spirit of the use of generic system components, the prototype MBDS uses the industry standard Ethernet communications bus. All of the MBDS computers are connected to a private local area network with only one machine (the controller) acting as a gateway to other, external computers in the School. The Ethernet has a relatively large capacity and greatly simplifies the task of adding backends to MBDS configurations.

III. THE COMPUTER-AIDED BENCHMARKING SYSTEM (CABS)

A. AN OVERVIEW

It is the flexible nature of MBDS that makes the configurable and parallel database machine so powerful and unique. It is the same flexibility and uniqueness that makes designing a test database for performance evaluation of the system such a headache. There are several system-configuration parameters available to the MBDS database administrator. Changes to key system-configuration parameters can have a profound effect on the performance of a given MBDS configuration. The key parameters include:

- The number of backends used.
- The number of clusters possible.
- The number of records per cluster.
- The block size.

The previous work of Strawser [Ref. 7], Tekampe and Watson [Ref. 8], and Vincent [Ref. 9] culminated in Fenton's implementation of CABS [Ref. 10]. With just three essential parameters from the user, CABS will generate a test database and a mix of test transactions. The test database is designed to be partitioned evenly over each of the possible MBDS test configurations. The maximum number of backends is input by the user. Special descriptor files are created by CABS to accomplish this distribution, along with the raw data records to insert in order to build the database. The database generated, although tailored for MBDS, is also general enough to permit testing on any relational DBMS. This is a key performance evaluation requirement in [Ref. 3] and [Ref. 4]. *System independence is the extent to which a model can be transported from system to system while remaining sufficiently representative* [Ref. 3]. Although not included in this study, this

generality of the database would permit a valid comparison of MBDS with another relational DBMS. To use the database in a relational DBMS, the evaluator would merely discontinue the use the MBDS template and descriptor files. The other output of CABS is the set of test transactions which are closely linked to the user-input database size. The automatic generation of the database input files and test transactions removes a great deal of tedious setup work from the system evaluator. CABS could be included as a production MBDS tool enabling the database administrator an efficient means of comparing different possible database configurations. In a more general role, the CABS database could be used to compare different DBMS products or different hardware running the same DBMS.

B. THE COMPONENTS OF CABS

1. The Database Generator

a. *The Design of the Test Database*

Partitioning a database for optimal performance is conventionally the realm of the database administrator. MBDS is unique because the partitioning or clustering of the database records is an integral function of the system. To design a test database that can be evenly split (by the MBDS data placement algorithm) among a given number of backends and is *generic* enough to work on other database systems is a complex task and presented a significant challenge to the designers of CABS. To make a comprehensive performance evaluation of MBDS, there were three different database sizes recommended by [Ref. 9]:

- Large (N bytes)
- Medium (N/2 bytes)
- Small (N/4 bytes)

Each of the two smaller database sizes is a multiple of the original database. To make the test more challenging, CABS uses Strawser's recommendation [Ref. 7] to use four different record sizes:

- Large (N)
- Medium-Lg (N/2)
- Medium (N/4)
- Small (N/10)

Again, each of the smaller record sizes must be able to divide the original, large record size evenly. All three of the test databases are made up of:

- 25% Large records
- 25% Medium-Large records
- 25% Medium records
- 25% Small records

This distribution is designed to give insight into DBMS performance when the number of bytes of base data is kept constant and the number of records is changed. The first step in finding a database of size N, which can be distributed evenly between a given number of backends, is determining the least common multiple (LCM) of the number of backends to be tested. CABS uses a lookup table of precalculated LCMs. Using the LCM in the database calculation ensures that for each configuration, the number of backends can be divided evenly into the number of records used.

CABS uses the LCM to calculate the size in bytes of the smallest database building block which can be split evenly between the backends. This building block is known as the database multiple (DBM). The DBM is a multiple of 32 [Ref. 9], because the database must be divisible by four since the database has to be quartered into the four different record sizes. The small database size is one quarter the original (N) size database. Finally, the implementation of CABS was simplified by using database size divisible by

two. The calculation to arrive at 32 as the factor was simply $(4 * 4 * 2) = 32$, which ensures the divisibility of the DBM. The last element of the DBM calculation is the large record size which is completely system dependent. In Strawser's original scheme [Ref. 7], the large record size is based on the disk track size. Unfortunately, modern disk drive capacity (30 or more kilobytes/track) makes such a record size impractical and unrealistic, as well, since database records are normally much smaller than 30,000 bytes. For this study a large record size of 1,000 bytes was chosen because it was large enough to force MBDS to fragment the records on the communications bus, but small enough to permit the use of a scaled-down database. The actual database multiple for this study then became:

- $DBM = LCM(1,2,3,4,5,6,7,8) * 32 * 1000 \text{ bytes}$
- $DBM = 840 * 32 * 1000 \text{ bytes}$
- $DBM = 26,880,000 \text{ bytes}$

This was, in fact, as close as CABS could get to the scaled-down database target size of 30 megabytes. The three database sizes provided by CABS were:

- Large 26,880,000 bytes
- Medium 13,440,000 bytes
- Small 6,720,000 bytes

Note that these calculations are actually carried out in base ten by CABS, apparently to simplify the actual calculations. There are several example charts in [Ref. 10] and in the actual output of CABS that are incorrectly labeled *megabytes*. If values in actual megabytes are required, a simple conversion is necessary (e.g., $26,880,000 \text{ bytes} / 1,024,000 \text{ bytes/MB} = 26.25 \text{ MB}$).

b. The Record File

Obviously, the largest output file is the record file needed to build the database record-by-record. The current version of CABS produces a single record file of the size requested by the user (large, medium or small) in order to conserve disk space. According to [Ref. 10], CABS is supposed to generate two types of record files. One for use in response-time reduction (RTR) test. The other, a set of files for use in response-time invariance (RTI) testing. RTR testing requires only one such input file which is reused for each test configuration tested regardless of the number of backends. RTI testing requires an extra input file for each of the backends in the test (i.e., an eight backend test requires eight input record files for the RTI testing). The goal is to double (triple, etc.) the total size of the database as the number of backends is doubled (tripled, etc.) to make the load on each backend equivalent to a one backend configuration. Unfortunately, this feature was not implemented in the actual software that resulted from [Ref. 10]. The only record file generated is the RTR file.

Contained in the record file generated by CABS are the input codes and raw attribute-value data necessary to build the database for all four record sizes. The record file is the input for the Test Interface (TI) mass load utility. The first three fields of each record are directory attributes which are used to cluster the records. The first attribute is a Type-B attribute which means the values the attribute can take on are limited to a *specific set* of values. This attribute is the mandatory "FILE" attribute, as well. The four possible "FILE" values are:

- Temp1g -- the large records
- Tempmed1g -- the medium-large records
- Tempmed -- the medium records
- Tempsmall -- the small records

The second and third attributes in each record are Type-A attributes which means there is a *range* of possible values for each attribute. These attributes are also defined as integer values. CABS simply inserts the consecutive record number starting from one for each record class. The second field is titled INTONE and is used to coarsely partition the database into nine cluster categories. The third field is titled INTTWO and is used to partition the database into hundreds of smaller clusters. The fourth attribute is titled the MULTIPLE attribute. The original intent in [Ref. 10] was to use the MULTIPLE attribute to differentiate one RTI file from another. The RTR file has a MULTIPLE value of "One". The RTI record file for two BEs would be "Two" and so on. Again, this feature is not implemented in the current version of CABS since the RTI files are not generated. This field was not used during our study. The test transactions do not check this field and MBDS does not protect against duplicate records, so the RTR file was inserted repeatedly to build RTI test databases.

c. The Template and Descriptor File

While the record file is fairly generic and could be used as the input file for other DBMS tests, the template and descriptor files are specifically created to support the loading of the various MBDS test configurations. Every DBMS needs a data definition language (DDL). MBDS uses the template file to load the data definitions to the system. A single template file is generated by CABS which is shared by each of the three test databases (large, medium-large, medium and small). The record format remains the same for each of the three database sizes. The *number* of records is the difference between database sizes. The template file lists the names of both directory and non-directory attributes and their associated data types.

The descriptor file provides MBDS with the indexing information necessary to define the record clusters of the database. The name of each directory attribute is listed, its classification (Type-A or Type-B), and its datatype. Type-B descriptors are followed by a list of the allowable values.

Type-A descriptors are followed by a list of the attribute-value ranges that specify each cluster category. Taken together as a set of values for any given record, the descriptors identify a unique cluster of records. The clustering methodology of CABS creates a realistic and challenging database for the benchmarking experiment. Hundreds of clusters are created and the clusters are loaded unevenly. Figure 5 shows the effect of the CABS generated descriptor file cluster definitions on the structure of the test database used for this study. The first clusters in each file (large, med-large, medium and small) are lightly loaded and the number of records per cluster is incrementally raised to five times as many records per cluster in the last clusters in the file. This uneven loading of the database was meant to further tax the system and add some realism to this artificial workload model.

2. The Test Transaction Mix

The test transaction mix was designed to allow the system performance evaluator to use one set of test transactions for all of the test configurations of each of the three database sizes. So three sets of test transactions are generated by CABS, each accessing a proportional number of clusters and records. Each set of transactions has four subsets of transactions, one for each record size (large, medium-large, medium and small). Each of these subsets of transactions is made up of 24 different ABDL transactions made up from each of the five primary database operations. Each of which falls into the following categories:

- Overhead-intensive
- Data-intensive
- Multi-file (relational)

The number of transactions generated is fairly large and considerable overlap is apparent. The system evaluator can pick an effective subset of these transactions by selecting representative transactions from the three query categories above.

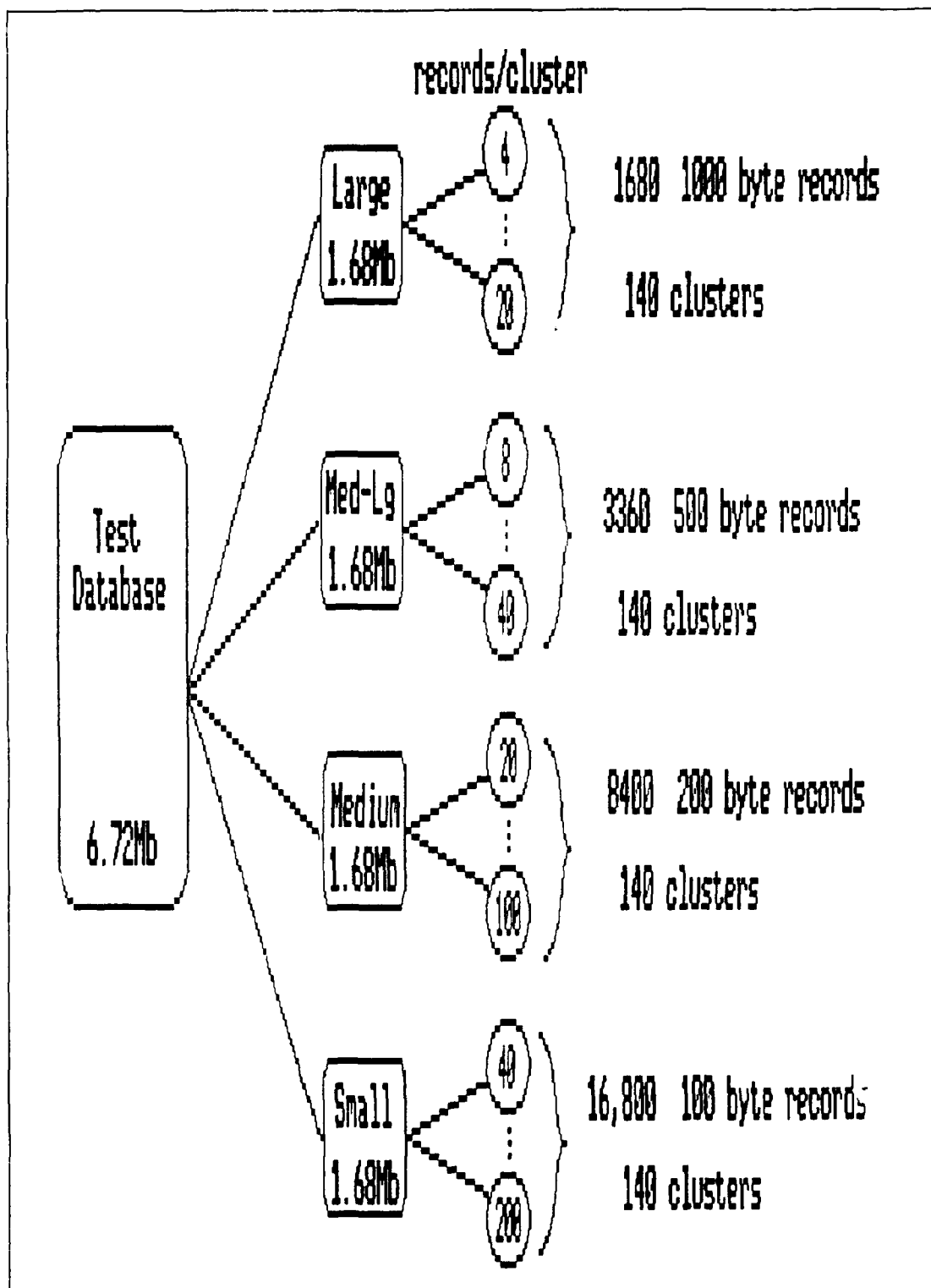


Figure 5. The Structure of the Test Database

3. The Generation of the Evaluator Reports

A considerable number of formatted tables are generated by CABS. Some of the reports proved useful, especially since the reports can be generated without necessarily having to actually produce the record file. This permits the evaluator to see the results of his input and make changes if necessary before actually building a huge record file. A minor problem with the reports is the column titles which read *Megabytes* with column values in *base ten* which should have been converted to megabytes. The problem which rendered the majority of the reports useless was the underlying assumption that MBDS would distribute the records perfectly evenly. The ideal distributions projected by CABS are far from the actual distributions achieved during testing.

Another problem with the CABS reports is the lack of an evaluator's guide. According to [Ref. 10], CABS is supposed to produce a complete set of narrative instructions along with the tables:

The second set of files comprises a number of reports describing the test databases and the test-transaction mixes. In conjunction with the second set of files, the CAD system interleaves a number of standard text files that present a narrative for the evaluator providing instructions on how to interface the CAD generated test-database and test-transaction-mix with MBDS. The text files also present a discussion for interpreting and analyzing the empirical data calculated by the CAD system.

Such an evaluator's report is not generated by the version of CABS used for this study.

C. HOW TO USE CABS

1. Running the program

The program is executed by simply issuing the command "cad" at the UNIX system command line in the "Bench" subdirectory of the current MBDS system. The first user prompt is:

```
*****  
Input the number of backends in the system> 8  
*****
```

The evaluator must input the maximum number of backends to be included in the test. In this case that value is eight. This is an important entry because expanding a benchmarking experiment using the CABS methodology initially setup for a lower number of backends requires a new set of files and repeating the testing. This study was a good example of the need for this type of planning. Initially, only six of the eight possible backend computers were operational due to back-ordered hardware components. Even though it was unlikely that all eight would become available for testing, the test database files generated by CABS were based on an eight-backend maximum configuration. Fortunately, the repair parts arrived in time for the inclusion of a seven and eight backend configuration to the existing data. There would not have been time to repeat the test from the beginning, had the initial data have been based on a maximum of six backends.

The next piece of information decides the record sizes used by CABS for the test database. The number entered at the the prompt below represents double the number of bytes the evaluator has set as the large record size. This value is also the *block size* assumed by CABS (track size is synonymous with block size as far as CABS is concerned). To permit splitting the large record size evenly into the three smaller record sizes, the large record size must be divisible by two, five and ten. This is the track size prompt:

Input the disk track size in the system in bytes > 2000

Next, the evaluator must set the maximum amount of data to be loaded to any single backend computer at the prompt given below:

Input the max disk storage of a single backend
in whole mega bytes (MBYTES) > 30

There is a possibility of a run-time error due to insufficient disk space for the record file. The evaluator must ensure there is adequate disk space in the UNIX file system for the record file size specified.

The evaluator can run the program repeatedly without actually generating the record file. This permits evaluation of the reports before the lengthy record file generation is activated. Report generation can also be suppressed. The remaining CABS prompts and run-time output are provided below:

```
*****
Do you want to generate the reports> y
Reports will be generated
Do you want the record files generated ?(y/n) y
PERFORMING INITIAL CALCULATIONS
GENERATING THE TEMPLATE FILES
GENERATING THE DESCRIPTOR FILES
GENERATING THE RECORD FILE
Create (s)mall, (m)edium, or (l)arge record file? s
Creating a small record file.
GENERATING THE TRANSACTION MIX FILES
GENERATING THE REPORT FILES
*****
```

2. The Output

CABS produces a total of 49 different files during execution and deposits the files in the directory from which the program is executed. There are three main groups of files produced:

- Database input files
- Transaction mix files
- Report files

The first two groups of files in the directory can be listed with the standard UNIX "ls" command:

```
*****
LDB_LGR#1  LDB_SMR#1  MDB_MLR#1  SDB_MDR#1  TEST.dl  TEST.r
LDB_MDR#1  MDB_LGR#1  MDB_SMR#1  SDB_MLR#1  TEST.dm  TEST.t
LDB_MLR#1  MDB_MDR#1  SDB_LGR#1  SDB_SMR#1  TEST.ds  cad*
```

The report files are *hidden* but can be listed by using the optional UNIX "ls -a" command. The report files now appear as listed below:

```
*****
./          .tm_mdb_lgr_rpt      LDB_MLR#1
../         .tm_mdb_lgr_wl      LDB_SMR#1
.ev_be_tbl_1 .tm_mdb_mdr_rpt      MDB_LGR#1
.ev_be_tbl_m .tm_mdb_mdr_wl      MDB_MDR#1
.ev_be_tbl_s .tm_mdb_mlr_rpt      MDB_MLR#1
.ev_rcd_blk_rel_tbl .tm_mdb_mlr_wl      MDB_SMR#1
.ev_rcd_per_clus_cat_tbl .tm_mdb_smr_rpt      SDB_LGR#1
.ev_test_config_ldb .tm_mdb_smr_wl      SDB_MDR#1
.ev_test_config_mdb .tm_sdb_lgr_rpt      SDB_MLR#1
.ev_test_config_sdb .tm_sdb_lgr_wl      SDB_SMR#1
.tm_ldb_lgr_rpt .tm_sdb_mdr_rpt      TEST.dl
.tm_ldb_lgr_wl .tm_sdb_mdr_wl      TEST.dm
.tm_ldb_mdr_rpt .tm_sdb_mlr_rpt      TEST.ds
.tm_ldb_mdr_wl .tm_sdb_mlr_wl      TEST.r
.tm_ldb_mlr_rpt .tm_sdb_smr_rpt      TEST.t
.tm_ldb_mlr_wl .tm_sdb_smr_wl      cad*
.tm_ldb_smr_rpt LDB_LGR#1          cadrun
.tm_ldb_smr_wl LDB_MDR#1
```

a. The Database Files

The CABS files necessary to build the test databases are the following:

- TEST.t - the MBDS template file
- TEST.dl - the MBDS descriptor file (large database)
- TEST.dm - the MBDS descriptor file (medium database)
- TEST.ds - the MBDS descriptor file (small database)
- TEST.r - the base data to be inserted

In order to load the test database, the evaluator has to manually copy or move the above files to the appropriate directories. The "UserFiles" directory on the backend controller needs to contain the following files:

- TEST.t
- TEST.d
- TEST.r

Note that there is only one descriptor file listed above. The evaluator must select the appropriate descriptor file for the database (small, medium or large) and rename it to TEST.d. The UserFiles directory on each of the backends must contain the following files:

- TEST.t
- TEST.d

The descriptor file is the same as the one on the controller. This is a key area for errors on the part of the evaluator, especially when moving from one database size to another. MBDS will *still work* with, for example, the small database descriptor files and the medium or even the large database record file. The problem is that the clustering will be completely wrong. Once MBDS runs out of the defined clusters in the small database descriptor file, the system will deposit all of the remaining records in a *catch all* cluster which exists to store records which do not match any set of defined directory attributes. Using the example of the medium database size generated by CABS which is double the small database size, one half of the database would end up in the catch all cluster. The effect on the performance is devastating which is a good argument for clustering in itself.

b. The Transaction Mix Files

The ABDL test-transaction files created by CABS:

- LDB_LGR#1, LDB_MLR#1, LDB_MDR#1, LDB_SMR#1
- MDB_LGR#1, MDB_MLR#1, MDB_MDR#1, MDB_SMR#1
- SDB_LGR#1, SDB_MLR#1, SDB_MDR#1, SDB_SMR#1

Each file contains 24 transactions which is more than enough for a complete test. These files are text files which can be modified in a text editor, if a smaller set of transactions can meet the evaluator's needs. The file names can be changed by the evaluator, as well, but the naming convention using a pound sign followed by a number must be maintained. The transaction files are the input for the test interface (TI) and are imported by using the TI "select transaction unit" option. These files must also be moved to the controllers UserFiles subdirectory.

c. The Report Files

All the remaining files in the directory are report files. The evaluator report files can be printed out by using the Unix command:

```
tbl <filename> | psroff -me
```

There are a large number of reports, but because they do not reflect the actual distribution of records, the reports are not very useful. The CABS files listed below do provide a good description of the "ideal" database topology which can be used as a comparison with the actual database:

- .ev_test_config_ldb
- .ev_test_config_mdb
- .ev_test_config_sdb

D. THE BENCHMARKING METHODOLOGY

1. The Initial System Setup

Before the loading of a test database can take place, the evaluator must ensure that the secondary storage devices are clear of any other data. The backend controller has a subdirectory named "test". This subdirectory is further subdivided into directories for each possible MBDS configuration and are appropriately titled 1, 2, 3, 4, 5, 6, 7 and 8. By selecting the directory appropriate to the configuration under study, an executable script file named "zero" is available to the evaluator. Entering the command "zero" will cause the meta data and base data disks used by MBDS to be cleared. This process takes about one hour per backend.

2. Operating the Test Interface

a. Starting the System

To run the test interface (TI), the evaluator must manually move (with the UNIX "cd" command) to the test subdirectory appropriate to the configuration under study. Along with the "zero" script file is another executable script file named "run". Issuing the "run" command will start TI, the MBDS processes on the controller and the processes on the appropriate backends. The initial start-up takes some time because communications between the controller and backends must be established. Once MBDS is online and the TI main menu is presented, select menu choice "a":

The Multi-Lingual/Multi-Backend Database System

Select an operation:

- (a) - Execute the attribute-based/ABDL interface
- (r) - Execute the relational/SQL interface
- (h) - Execute the hierarchical/DL/I interface
- (n) - Execute the network/CODASYL interface
- (f) - Execute the functional/DAPLEX interface
- (x) - Exit to the operating system

Select-> a

b. Loading the Test Database

To load the test database, the evaluator needs to select the *Load a database option* from the menu below:

The attribute-based/ABDL interface:

- (g) - Generate a database
- (l) - Load a database
- (r) - Request interface
- (x) - Exit to the previous menu

Select-> l

The next step is to load the test database template file by selecting *Use a database* from the menu below and responding to the prompt for the database name with "test" which is the test database name:

Select an operation:

- (u) - Use a database
- (r) - Mass load a file of records
- (x) - Exit, return to previous menu

Select-> u

Enter the name of the database: test

At this point, the system is ready to begin the mass-loading of the record file. To accomplish this, select the *Mass load a file of records* option from the menu below and respond to the prompt with the record file name "TEST.r".

Select an operation:

- (u) - Use a database
- (r) - Mass load a file of records
- (x) - Exit, return to previous menu

Select-> r

Enter the record file name: TEST.r

The loading process will start and provide feedback on the progress of the mass-loading utility every ten records. Initially, he records load at a rate of about one megabyte an hour and after six to seven hours the UNIX operating system slows the entire process considerably. The only practical solution to this problem, currently, is to split the input record file into a series of six megabyte files. Since MBDS must run for many hours to build the database, it is advisable to enter the following key strokes once the loading process is running to cause MBDS to end the session normally and save the meta data:

```
u      <CR>
test  <CR>
x      <CR>
x      <CR>
x      <CR>
```

The operating system will buffer this input and accept the commands when the loading process finishes. This frees the evaluator from monitoring the progress of the mass-load process and limits the damage should one of the MBDS computers crash after a successful load. Ending normally writes the meta data to secondary storage which permits restarting the system at a later date. An abnormal end (such as a crash) after a successful mass-load could be costly because reloading the database is the only way to build the meta data.

c. Conducting Performance Testing

Once the test database is loaded, the evaluator can begin the actual performance evaluation. The first step is to ensure that there are no other users on the controller nor on any of the backends. Next, the evaluator must start TI from the same system subdirectory used to load the test configuration. From the main menu below, select the *Execute the attribute-based/ABDL interface* option:

Multi-Lingual/Multi-Backend Database System

Select an operation:

- (a) - Execute the attribute-based/ABDL interface
- (r) - Execute the relational/SQL interface
- (h) - Execute the hierarchical/DL/I interface
- (n) - Execute the network/CODASYL interface
- (f) - Execute the functional/DAPLEX interface
- (x) - Exit to the operating system

Select-> a

Next, select the *Request interface* option from the menu below and respond to the prompt for the database name with "test":

The attribute-based/ABDL interface:

- (g) - Generate a database
- (l) - Load a database
- (r) - Request interface
- (x) - Exit to the previous menu

Select-> r

Enter the database id: test

Next, select the *Performance Testing* option from the menu below:

Select a subsession:

- (s) SELECT: select traffic units from an existing list
(or give new traffic units) for execution
- (n) NEW LIST: create a new list of traffic units
- (d) NEW DATABASE: choose a new database
- (p) * PERFORMANCE TESTING
- (r) * REDIRECT OUTPUT: select output for answers
- (m) * MODIFY: modify an existing list of traffic units
- (o) * OLD LIST: execute all the traffic units in an
existing list
- (x) EXIT: return to previous menu

Refer to the MLDS/MBDS user manual before choosing
subsessions marked with an asterisk (*)

Select-> p

The next step is to enable the system timers, select the *Turn on external timer* option from the menu below and then select the *Exit to previous menu* option as shown below:

Select an operation:

- (e) Turn on external timer.
- (i) Turn on internal timers.
- (a) ABORT..Abandon all requested actions.
- (x) Exit to previous menu.

Select-> e

External Timer On

Select an operation:

- (e) Turn on external timer.
- (i) Turn on internal timers.
- (a) ABORT..Abandon all requested actions.
- (x) Exit to previous menu.

Select-> x

The next step is to load a set of test transactions to run against the database. To accomplish this, select the first menu choice from the menu below and respond to the prompt with the name of one of the four transaction sets appropriate for the size of the database under study:

Select a subsession:

- (s) SELECT: select traffic units from an existing list
(or give new traffic units) for execution
- (n) NEW LIST: create a new list of traffic units
- (d) NEW DATABASE: choose a new database
- (p) * PERFORMANCE TESTING
- (r) * REDIRECT OUTPUT: select output for answers
- (m) * MODIFY: modify an existing list of traffic units
- (o) * OLD LIST: execute all the traffic units in an
existing list
- (x) EXIT: return to previous menu

Refer to the MLDS/MBDS user manual before choosing
subsessions marked with an asterisk (*)

Select-> s

Enter the name for the traffic unit file
It may be up to 40 characters long including the .ext.
Filenames may include only one '#' character
as the first character before the version number.

FILE NAME-> SDB_SMR#1

d. Collecting the Performance Data

Once the transaction set is loaded, the evaluator just needs to enter the transaction number (they are numbered from zero) at the menu below:

Select Options:

- (d) redisplay the traffic units in the list
- (n) enter a new traffic unit to be executed
- (num) execute the traffic unit at [num]
 from the above list
- (x) exit from this SELECT subsession

Option-> 0

(<CNT(INTONESMALL), 320>)

Start: 08:32:31 Stop: 08:32:33 Elapsed Time : 2.167

The response time of the transaction is displayed as shown above. The evaluator must, unfortunately, manually transcribe the times for later analysis. To move to the next record size test transaction set, exit from this submenu, choose the select option again and change to the next test transaction file name.

The technique used during this study was to enter transaction "0" several times in rapid succession followed by the number of the next transaction to be timed. There were two important reasons for this process:

- UNIX will automatically swap processes which are inactive for several seconds out of memory. This creates a considerable delay as MBDS processes are re-activated. A stream of transactions keeps all of the necessary processes active.
- If a given transaction is executed more than once in succession, the response time tends to be improved by the location of the hard disk read/write heads which stay in the locality of the last read operation. The stream of transaction number "0" entries ensures the fact that each

time a transaction is tested the disk drive heads start from the same spot.

In addition to the above technique, the first time for any test transaction was always thrown out because it often varied widely from subsequent times for the same transaction. The timing procedure above was repeated for every transaction until a constant result was obtained (e.g., within 0.050 seconds for data-intensive transactions and 0.017 seconds for overhead-intensive transactions).

e. Exiting the Test Interface

To exit TI, use the menu selections to end normally. When the user ends normally, all of the MBDS controller and backend processes are also stopped. Although exiting TI by using the CONTROL-C keypress is possible, this leaves most of the MBDS processes running which will interfere with the processes started the next time TI is run. If it is necessary to end abnormally, there is a script file in each configuration subdirectory named "burn" which will stop all MBDS processes.

IV. THE MEASURES

A. AN OVERVIEW

The MBDS software can be conditionally compiled to include the code necessary to instrument the system for the timing experiments. The added code creates *checkpoints* which are used to clock the response time of a given MBDS request. The *response time* of a request is defined as the time which elapses between the time the request is released by the user and the time the system is ready to display the response to the request. Actual display time is not included since this could vary greatly. The readings are based on times taken from the controller's computer system clock. Once MBDS is compiled with the timing flags on, the request timer is activated within the test interface as outlined in Chapter III. The timing data is displayed on the terminal screen and must be manually recorded and analyzed off-line using the performance measures.

B. THE RESPONSE-TIME REDUCTION MEASURE

Adding backends to a given MBDS configuration and redistributing the database evenly should, intuitively, improve the performance of the system. The increase in system performance should, in turn, be observable in request response time. This increase in performance is attributable to the increased parallelism in request processing. In other words, the work is shared between more backends. With the increase in the number of backends, there is also a possibility that the overhead of coordinating communications could cause a measurable impact on system performance and could actually become the limiting factor when considering the optimal number of backends.

The MBDS response-time reduction (RTR) performance claim made by the designer and implementor of MBDS was introduced in Chapter I and is repeated here for completeness:

- **Response-Time Reduction (RTR).** The response-time reduction of a transaction is inversely proportional to the multiplicity of the backends. This means that as the number of backends increases, the response-time reduction for a given transaction is expected to improve (e.g., moving from one backend to three backends should yield a response-time which is just one-third the original response time).

To measure the degree to which MBDS demonstrates response-time reduction, the actual performance of the system as measured during the benchmarking of each multi-backend configuration must be compared with the base times of a single-backend configurations. This means that the *ideal* time for a given transaction is defined as the single-backend transaction response time divided by the number of backends used by the multi-backend test configuration. This measure permits direct, graphical comparison of an easily calculated ideal time with the actual performance data collected.

C. THE RESPONSE-TIME INVARIANCE MEASURE

The most important claim made about MBDS is the large capacity for database growth while maintaining an acceptable level of performance. The MBDS response-time invariance (RTI) performance claim made by the designer and implementor of MBDS was introduced in Chapter I and is repeated here for completeness:

- **Response-Time Invariance (RTI).** The response-time invariance of a transaction in response to the increase of the database size is maintained by a corresponding increase in the multiplicity of the backends (e.g., when the database doubles in size, doubling the number of backends will yield the original response-time).

Response-time invariance testing requires maintaining the original single-backend (base system) load on each backend of the multi-backend configuration. This is accomplished by inserting the batch of database records repeatedly, twice for a two-backend test, three times for a three backend test and so on. To measure the degree to which MBDS demonstrates response-time invariance, the actual performance of the system as measured during the benchmarking of each fully loaded multi-backend configuration must be

compared with the base times of a single-backend configuration. This means that the *ideal* time for a given transaction is defined as the single-backend transaction response time. This measure permits direct, graphical comparison of the ideal time with the actual performance data collected.

Half of the multi-backend configurations were tested with record distributions as distributed by MBDS. The remainder of the multi-backends were artificially loaded perfectly even. This was accomplished by loading one backend and copying the meta data and base data to the remaining backends. This process was much faster than repeated batch loading and permitted the analysis of even loading.

V. THE PERFORMANCE DATA

A. THE BASE CONFIGURATION

The *ideal times* for both the response-time reduction and response-time invariance testing are based on the single-backend or *baseline* configuration. Table 2 lists the transaction response times in seconds recorded for the single-backend configuration. Table 3 lists the number of records of each record size.

TABLE 2. THE SINGLE-BACKEND RESPONSE TIMES

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.067	1.900	1.830	1.817
TR 2	13.366	11.833	11.216	10.933
TR 3	25.899	23.149	22.299	21.082
TR 4	46.815	43.015	41.648	40.632
TR 5	25.549	23.716	22.766	22.282
TR 6	2.600	2.483	2.567	2.417
TR 7	79.347	79.897	80.763	77.714

TABLE 3. SINGLE-BACKEND RECORD DISTRIBUTIONS

Rec Size	BE #1
Large	1680
Med-Large	3360
Medium	8400
Small	16800
Total	30240

B. THE TWO-BACKEND CONFIGURATION

1. RTR Results

Table 4 lists the transaction response times in seconds recorded for the two-backend RTR configuration. Table 5 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 4. THE TWO-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	1.650	1.200	1.200	1.250
TR 2	8.283	6.750	6.750	7.350
TR 3	13.661	11.833	11.833	11.833
TR 4	25.482	22.366	22.366	23.382
TR 5	13.416	11.850	11.850	11.566
TR 6	2.050	1.633	1.633	1.550
TR 7	41.265	39.748	39.748	41.315

TABLE 5. TWO-BE RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	Total
Large	870	810	1680
Med-Large	1712	1648	3360
Medium	4206	4194	8400
Small	7734	9066	16800
Total	14522	15718	30240

2. RTI Results

Table 6 lists the transaction response times in seconds recorded for the two-backend RTI configuration. Table 7 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 6. THE TWO-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in seconds			
	SMR	MDR	MLR	LGR
TR 1	2.783	2.117	1.95	1.867
TR 2	14.049	12.166	11.816	12.400
TR 3	25.916	23.282	22.932	22.232
TR 4	47.798	44.415	44.098	44.198
TR 5	26.082	24.582	23.916	23.632
TR 6	3.383	2.917	2.583	2.483
TR 7	77.197	76.647	80.397	77.730

TABLE 7. TWO-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	Total
Large	1672	1688	3360
Med-Large	3368	3352	6720
Medium	8396	8404	16800
Small	17421	16179	33600
Total	30857	29623	60480

C. THE THREE-BACKEND CONFIGURATION

1. RTR Results

Table 8 lists the transaction response times in seconds recorded for the three-backend RTR configuration. Table 9 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 8. THE THREE-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in seconds			
	SMR	MDR	MLR	LGR
TR 1	1.200	1.183	1.017	0.933
TR 2	6.383	5.116	4.983	4.916
TR 3	9.983	8.233	7.900	8.300
TR 4	19.016	15.916	15.899	15.499
TR 5	9.966	8.333	8.016	8.233
TR 6	1.450	1.433	1.317	1.200
TR 7	31.132	27.966	27.966	27.082

TABLE 9. THREE-BACKEND RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	Total
Large	528	582	570	1680
Med-Large	1168	1008	1184	3360
Medium	2776	2644	2980	8400
Small	4626	5478	6696	16800
Total	9098	9712	11430	30240

2. RTI Results

Table 10 lists the transaction response times recorded for the three-backend RTI configuration. Table 11 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 10. THE THREE-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.650	2.367	2.167	2.283
TR 2	14.433	12.150	12.016	12.933
TR 3	26.532	23.132	22.482	22.516
TR 4	48.765	45.931	47.115	46.648
TR 5	27.982	24.099	22.582	22.482
TR 6	3.450	3.200	3.017	2.967
TR 7	80.063	79.597	79.607	80.230

TABLE 11. THREE-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	Total
=====	=====	=====	=====	=====
Large	1660	1668	1712	5040
Med-Large	3357	3311	3412	10080
Medium	8256	8373	8571	25200
Small	15964	16665	17771	50400
Total	29237	30017	31466	90720

D. THE FOUR-BACKEND CONFIGURATION

1. RTR Results

Table 12 lists the transaction response times in seconds recorded for the four-backend RTR configuration. Table 13 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 12. THE FOUR-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	1.200	0.833	0.817	0.833
TR 2	4.600	4.017	3.950	4.233
TR 3	7.450	6.233	6.550	7.283
TR 4	14.049	12.233	12.200	13.566
TR 5	7.450	6.200	6.550	7.333
TR 6	1.450	1.067	0.950	1.067
TR 7	22.499	21.299	21.149	23.849

TABLE 13. FOUR-BACKEND RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	Total
Large	440	364	370	506	1680
Med-Large	876	820	772	892	3360
Medium	1906	2152	2288	2054	8400
Small	4124	3810	4942	3924	16800
Total	7346	7146	8372	7376	30240

2. RTI Results

Table 14 lists the transaction response times in seconds recorded for the four-backend RTI configuration. Table 15 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 14. THE FOUR-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.517	2.267	2.200	2.133
TR 2	13.683	11.866	11.700	12.600
TR 3	26.199	23.549	22.849	22.700
TR 4	47.015	44.519	44.498	45.948
TR 5	26.249	24.282	24.016	23.749
TR 6	3.217	2.933	2.933	2.850
TR 7	77.997	78.130	78.130	80.647

TABLE 15. FOUR-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	Total
Large	1672	1616	1672	1760	6720
Med-Large	3344	3409	3295	3392	13440
Medium	8327	8166	8444	8663	33600
Small	16571	16276	17452	16901	67200
Total	29914	29467	30863	30716	120960

E. THE FIVE-BACKEND CONFIGURATION

1. RTR Results

Table 16 lists the transaction response times in seconds recorded for the five-backend RTR configuration. Table 17 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 16. THE FIVE-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	0.900	0.833	0.850	0.750
TR 2	3.900	3.500	3.450	3.833
TR 3	6.566	5.566	5.266	5.416
TR 4	10.950	10.883	10.883	10.183
TR 5	6.566	5.583	5.300	5.983
TR 6	1.083	1.117	1.200	1.033
TR 7	17.660	18.716	18.849	19.066

TABLE 17. FIVE-BACKEND RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	Total
Large	404	310	296	322	348	1680
Med-Large	488	760	672	628	812	3360
Medium	1794	1596	1398	2052	1560	8400
Small	3378	3940	3302	2804	3376	16800
Total	6064	6606	5668	5806	6096	30240

2. RTI Results

Table 18 lists the transaction response times in seconds recorded for the five-backend RTI configuration. Table 19 lists the number of records of each record size distributed evenly on each backend.

TABLE 18. THE FIVE-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.150	1.983	1.967	1.933
TR 2	13.733	12.466	12.150	11.966
TR 3	26.266	23.432	22.932	22.699
TR 4	48.481	47.998	47.231	47.665
TR 5	26.282	23.366	22.816	22.799
TR 6	2.617	2.583	2.583	2.467
TR 7	80.863	81.147	80.780	80.530

TABLE 19. FIVE-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	Total
Large	1680	1680	1680	1680	1680	8400
Med-Large	3360	3360	3360	3360	3360	16800
Medium	8400	8400	8400	8400	8400	42000
Small	16800	16800	16800	16800	16800	84000
Total	30240	30240	30240	30240	30240	151200

F. THE SIX-BACKEND CONFIGURATION

1. RTR Results

Table 20 lists the transaction response times in seconds recorded for the six-backend RTR configuration. Table 21 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 20. THE SIX-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	0.984	0.833	0.767	0.733
TR 2	4.183	3.150	3.133	3.333
TR 3	5.283	5.016	4.683	4.866
TR 4	10.350	8.016	8.400	8.450
TR 5	5.333	5.016	4.683	4.850
TR 6	1.217	1.117	0.883	0.817
TR 7	16.333	13.949	14.683	16.083

TABLE 21. SIX-BACKEND RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	Total
Large	288	350	290	240	232	280	1680
Med-Large	564	492	568	604	516	616	3360
Medium	1446	1384	1488	1330	1260	1492	8400
Small	2226	2332	3694	2400	3146	3002	16800
Total	4524	4558	6040	4574	5154	5390	30240

2. RTI Results

Table 22 lists the transaction response times in seconds recorded for the six-backend RTI configuration. Table 23 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 22. THE SIX-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.600	2.383	2.400	2.200
TR 2	14.516	12.200	12.583	12.616
TR 3	26.532	22.882	22.932	22.949
TR 4	47.581	44.065	44.982	44.248
TR 5	26.332	24.749	23.216	22.982
TR 6	3.117	3.233	3.283	2.917
TR 7	78.230	76.914	77.947	80.463

TABLE 23. SIX-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	Total
=====	=====	=====	=====	=====	=====	=====	=====
Large	1676	1704	1680	1664	1676	1680	10080
Med-Large	3264	3288	3416	3401	3409	3382	20160
Medium	8297	8343	8578	8500	8368	8314	50400
Small	16037	16066	17668	16839	17397	16793	100800
Total	29274	29401	31342	30404	30850	30169	181440

G. THE SEVEN-BACKEND CONFIGURATION

1. RTR Results

Table 24 lists the transaction response times in seconds recorded for the seven-backend RTR configuration. Table 25 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 24. THE SEVEN-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	0.833	0.833	0.767	0.733
TR 2	3.467	2.650	3.683	2.783
TR 3	4.966	4.266	4.500	3.950
TR 4	8.116	7.366	7.350	7.483
TR 5	4.966	4.283	4.500	3.967
TR 6	0.967	0.950	0.850	0.867
TR 7	12.333	12.933	12.733	13.916

TABLE 25. SEVEN-BACKEND RTR RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	BE #7	Total
Large	258	288	250	218	216	190	260	1680
Med-Large	488	448	520	540	376	464	524	3360
Medium	1276	1302	1184	1328	1360	1018	932	8400
Small	2224	2592	2268	2402	2230	2800	2284	16800
Total	4246	4630	4222	4488	4182	4472	4000	30240

2. RTI Results

Table 26 lists the transaction response times in seconds recorded for the seven-backend RTI configuration. Table 27 lists the number of records of each record size distributed evenly on each backend.

THE TABLE 26. THE SEVEN-BACKEND RTI PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.167	2.017	1.983	1.967
TR 2	13.916	12.533	12.150	12.383
TR 3	27.266	23.516	22.932	22.649
TR 4	51.698	48.281	47.181	47.265
TR 5	27.399	23.516	22.432	22.666
TR 6	2.650	2.600	2.533	2.667
TR 7	82.663	80.130	80.413	84.147

TABLE 27. SEVEN-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	BE #7	Total
Large	1680	1680	1680	1680	1680	1680	1680	11760
Med-Large	3360	3360	3360	3360	3360	3360	3360	23520
Medium	8400	8400	8400	8400	8400	8400	8400	58800
Small	16800	16800	16800	16800	16800	16800	16800	117600
Total	30240	30240	30240	30240	30240	30240	30240	211680

H. THE EIGHT-BACKEND CONFIGURATION

1. RTR Results

Table 28 lists the transaction response times in seconds recorded for the eight-backend RTR configuration. Table 29 lists the number of records of each record size distributed by MBDS on each backend.

TABLE 28. THE EIGHT-BACKEND RTR PERFORMANCE

Trans #	Response-Times by Record Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	1.117	0.700	0.833	0.617
TR 2	2.883	2.383	2.983	2.650
TR 3	4.716	3.583	3.783	4.400
TR 4	7.500	6.516	7.083	7.533
TR 5	4.716	3.600	3.767	4.400
TR 6	1.267	0.917	0.950	0.750
TR 7	11.100	11.333	12.366	13.049

TABLE 29. EIGHT-BACKEND RTR RECORD DISTRIBUTION

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	BE #7	BE #8	Total
Large	248	224	172	228	192	140	198	278	1680
Med-Large	460	356	312	380	416	464	460	512	3360
Medium	897	1065	1087	973	1010	1079	1204	1085	8400
Small	2300	1962	2355	1584	1817	1856	2578	2348	16800
Total	3905	3607	3926	3165	3435	3539	4440	4223	30240

2. RTI Results

Table 30 lists the transaction response times in seconds recorded for the eight-backend RTI configuration. Table 31 lists the number of records of each record size distributed evenly on each backend.

TABLE 30. THE EIGHT-BACKEND RTI PERFORMANCE

Trans #	Response-Time by Record-Size in Seconds			
	SMR	MDR	MLR	LGR
TR 1	2.167	2.017	2.017	1.983
TR 2	13.966	12.483	12.316	12.316
TR 3	27.466	23.599	23.032	22.732
TR 4	50.115	48.448	47.598	47.098
TR 5	27.882	23.582	23.066	22.766
TR 6	2.667	2.650	2.650	2.583
TR 7	87.363	83.480	83.880	80.197

TABLE 31. EIGHT-BACKEND RTI RECORD DISTRIBUTIONS

Rec Size	BE #1	BE #2	BE #3	BE #4	BE #5	BE #6	BE #7	BE #8	Total
Large	1680	1680	1680	1680	1680	1680	1680	1680	13440
Med-Large	3360	3360	3360	3360	3360	3360	3360	3360	26880
Medium	8400	8400	8400	8400	8400	8400	8400	8400	67200
Small	16800	16800	16800	16800	16800	16800	16800	16800	134400
Total	30240	30240	30240	30240	30240	30240	30240	30240	241920

I. AVERAGE PERFORMANCE BY TRANSACTION TYPE

To permit the evaluation of MBDS performance across all four record sizes, the response times of the all four record sizes were averaged for each test-transaction number. This process helped consolidate and smooth the data collected. Table 32 presents the average RTR test-transaction response times for each test-transaction number and MBDS configuration tested. Table 33 presents the *ideal* RTR times calculated for use in evaluating system performance. The ideal times are based on the average performance of the baseline, single-backend MBDS configuration on all four test-transaction record sizes. Table 34 presents the average RTI test-transaction response times for each test-transaction number and MBDS configuration tested.

TABLE 32. AVERAGE RTR PERFORMANCE TIMES

Average Response-Times by Transaction Number in Seconds								
Trans#	ONE BE AVG	TWO BE AVG RTR	THREE BE AVG RTR	FOUR BE AVG RTR	FIVE BE AVG RTR	SIX BE AVG RTR	SEVEN BE AVG RTR	EIGHT BE AVG RTR
TR1	1.909	1.325	1.083	0.921	0.833	0.829	0.791	0.817
TR2	11.837	7.283	5.349	4.200	3.671	3.450	3.146	2.725
TR3	23.107	12.290	8.604	6.879	5.704	4.962	4.420	4.120
TR4	43.028	23.399	16.582	13.012	10.725	8.804	7.579	7.158
TR5	23.578	12.171	8.637	6.883	5.858	4.970	4.429	4.121
TR6	2.517	1.716	1.350	1.133	1.108	1.008	0.908	0.971
TR7	79.430	40.519	28.536	22.199	18.573	15.262	12.979	11.962

TABLE 33. IDEAL RTR PERFORMANCE TIMES

Average Response-Times by Transaction Number in Seconds								
Trans#	ONE BE	1BE/2	1BE/3	1BE/4	1BE/5	1BE/6	1BE/7	1BE/8
TR1	1.909	0.950	0.633	0.475	0.380	0.317	0.271	0.237
TR2	11.837	5.917	3.944	2.958	2.367	1.972	1.690	1.479
TR3	23.107	11.575	7.716	5.787	4.630	3.858	3.307	2.894
TR4	43.028	21.508	14.338	10.754	8.603	7.169	6.145	5.377
TR5	23.578	11.858	7.905	5.929	4.743	3.953	3.388	2.965
TR6	2.517	1.242	0.828	0.621	0.497	0.414	0.355	0.310
TR7	79.430	39.949	26.632	19.974	15.979	13.316	11.414	9.987

TABLE 34. AVERAGE RTI PERFORMANCE TIMES

Average Response-Times by Transaction Number in Seconds								
Trans#	ONE BE AVG	TWO BE AVG RTI	THREE BE AVG RTI	FOUR BE AVG RTI	FIVE BE AVG RTI	SIX BE AVG RTI	SEVEN BE AVG RTI	EIGHT BE AVG RTI
TR1	1.903	2.179	2.367	2.279	2.008	2.396	2.034	2.046
TR2	11.837	12.608	12.883	12.462	12.579	12.979	12.745	12.770
TR3	23.107	23.590	23.766	23.824	23.832	23.824	24.091	24.207
TR4	43.028	45.127	47.110	45.495	47.844	45.219	48.606	48.315
TR5	23.578	24.553	24.286	24.574	23.916	24.320	24.003	24.324
TR6	2.517	2.841	3.159	2.983	2.563	3.137	2.612	2.638
TR7	79.430	77.993	79.874	78.726	80.830	78.388	81.838	83.730

VI. ANALYSIS AND INTERPRETATION OF THE TEST DATA

A. AN ANALYSIS OF MBDS RESPONSE-TIME REDUCTION

1. RTR Performance on Overhead-Intensive Transactions

Test transactions number one and number two are overhead-intensive transactions which access (read) only 4% of the database. Of the database records retrieved, only about half satisfy the query or 2% of the database. Very little time is spent actually reading the database, most of the time is spent looking up the clusters and on communications between the controller and the backends.

Figure 6 shows the RTR performance of MBDS on test transaction number one, a RETRIEVE transaction. Figure 7 shows the RTR performance of MBDS on test transaction number six, a DELETE transaction. Clearly, the actual performance lags behind the *ideal* estimate of the performance improvement for both overhead-intensive transactions. Redistributing the data across additional backends ceases to be beneficial at the four or five backend mark. The addition of backends beyond this point becomes futile because the *system overhead has become the limiting factor*. It is interesting to note that while the performance levels off at this point, it does not seem to worsen, even though more communications overhead is introduced. Note that the ideal performance curve, too, goes asymptotic with the X axis as the number of backends increases. The scale on the Y axis should also be noted to put the (sub-second) difference between actual and ideal performance in perspective.

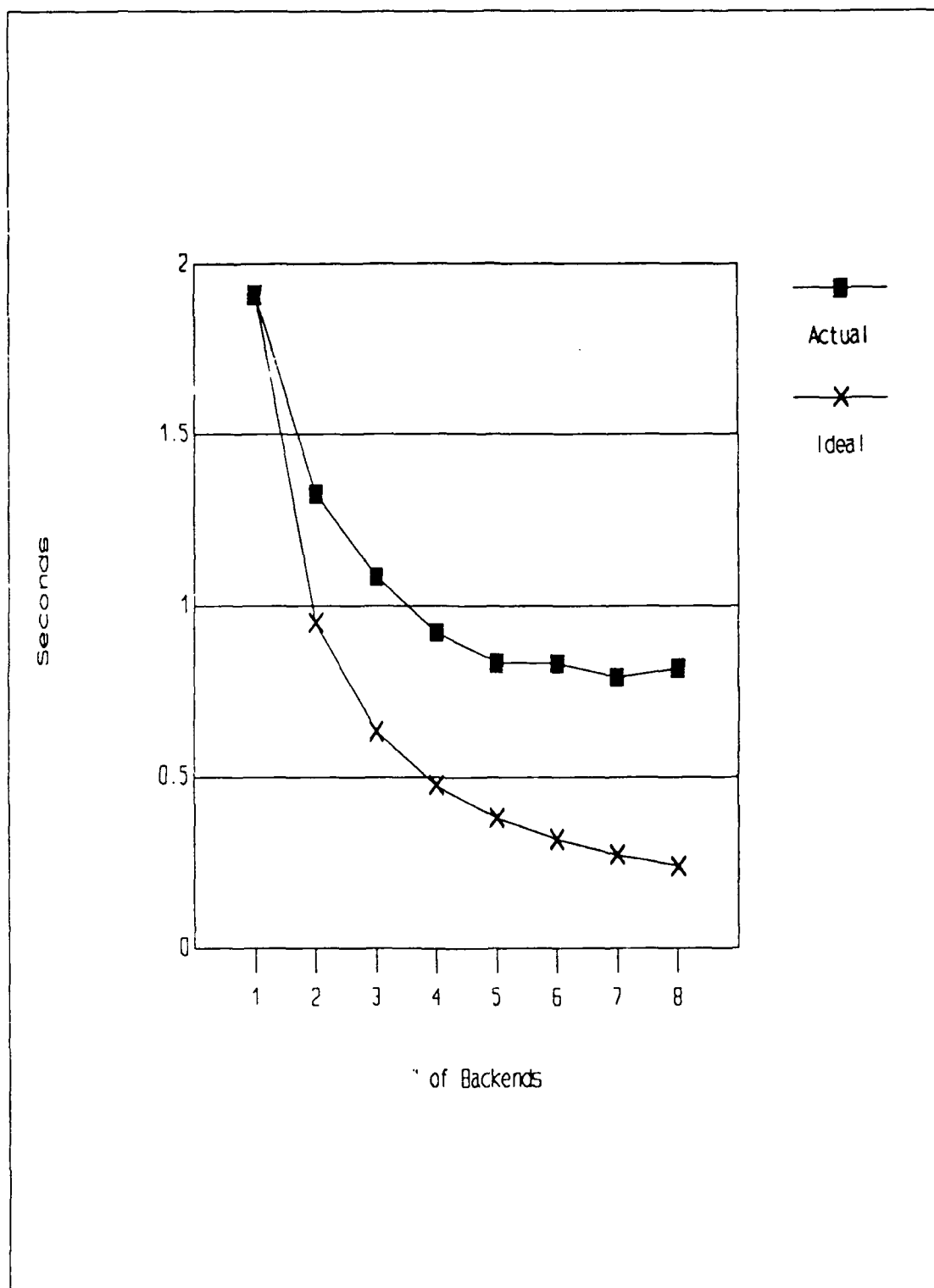


Figure 6. RTR Performance on Transaction #1

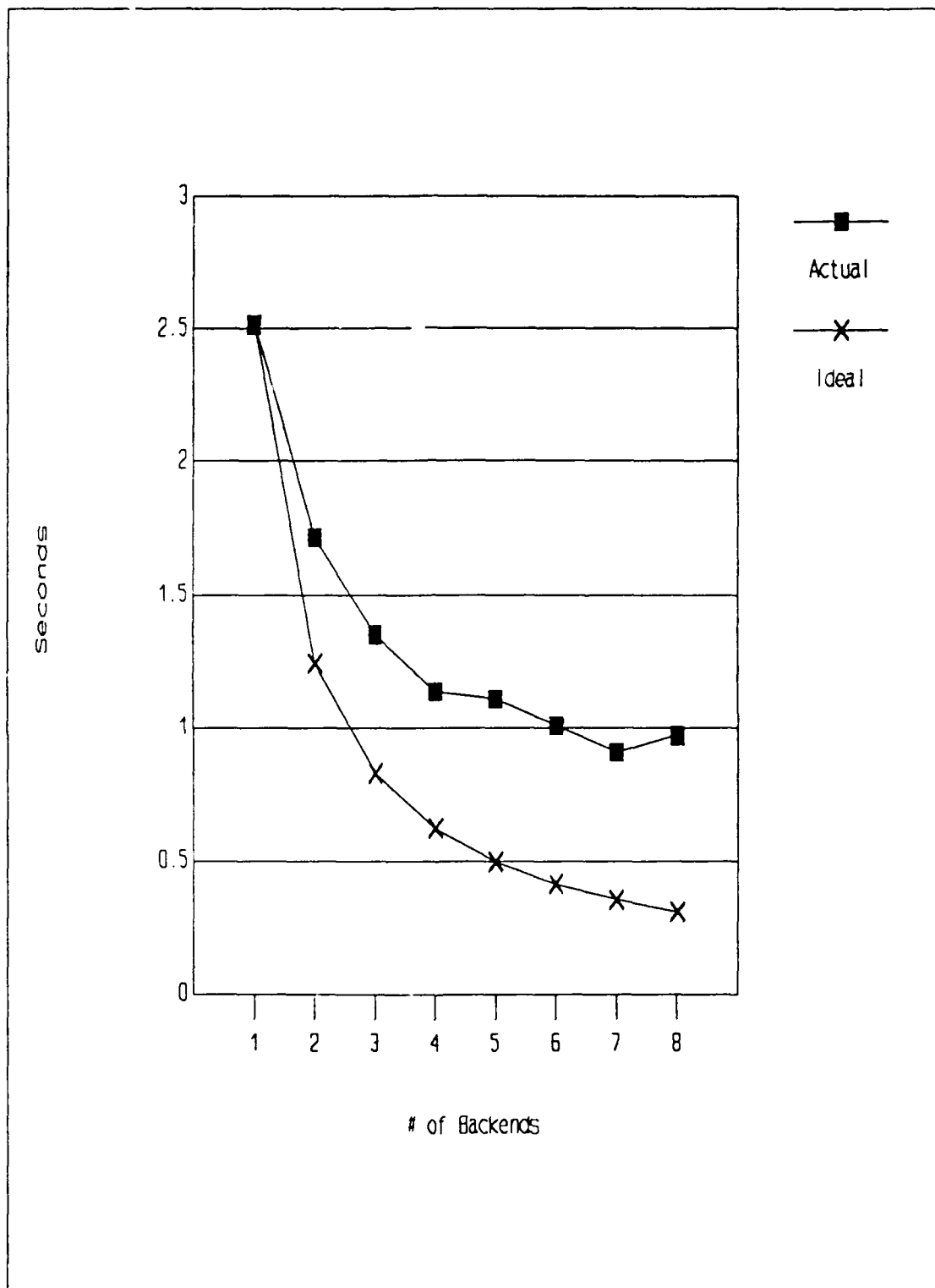


Figure 7. RTR Performance on Transaction #6

2. RTR Performance on Data-Intensive Transactions

The remaining five test transactions are data-intensive transactions. These transactions cause the backends to access as much as 100% of the database records. In data-intensive cases, accesses to secondary storage are prevailing and overshadow data communications time and computer processing time. This means the secondary storage access time is the limiting factor, not the system overhead. This can be seen in the response times which are in the tens of seconds. Sharing this sort of workload is ideally suited for MBDS as the RTR performance results demonstrate:

- Test transaction number two is a data-intensive RETRIEVE transaction which accesses 26% of the database with 96% of the records retrieved (25% of the database) satisfying the query. Figure 8 shows the RTR test results for test transaction number two.
- Test transaction number three is a data-intensive RETRIEVE transaction which accesses 50% of the database with half of the records retrieved (25% of the database) satisfying the query. Figure 9 shows the RTR test results for test transaction number three.
- Test transaction number four is a data-intensive RETRIEVE transaction which accesses 100% of the database with half of the records retrieved (50% of the database) satisfying the query. Figure 10 shows the RTR test results for test transaction number four.
- Test transaction number five is a data-intensive RETRIEVE transaction which accesses 50% of the database with half of the records retrieved (25% of the database) satisfying the query. Figure 11 shows the RTR test results for test transaction number five.
- Test transaction number seven is a data-intensive DELETE transaction which accesses 100% of the database with half of the records retrieved (50% of the database) satisfying the query. Figure 12 shows the RTR test results for test transaction number seven.

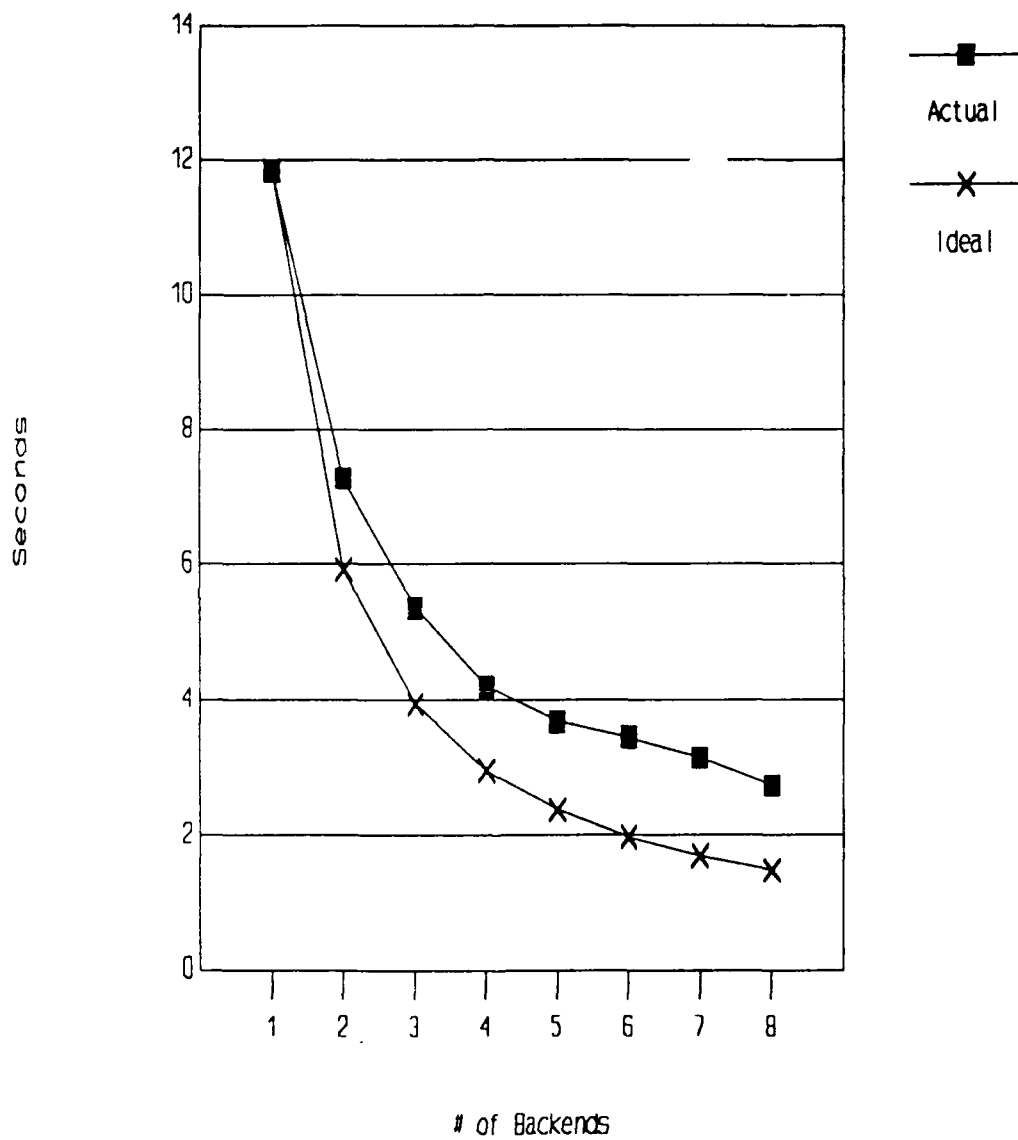


Figure 8. RTR Performance on Transaction #2

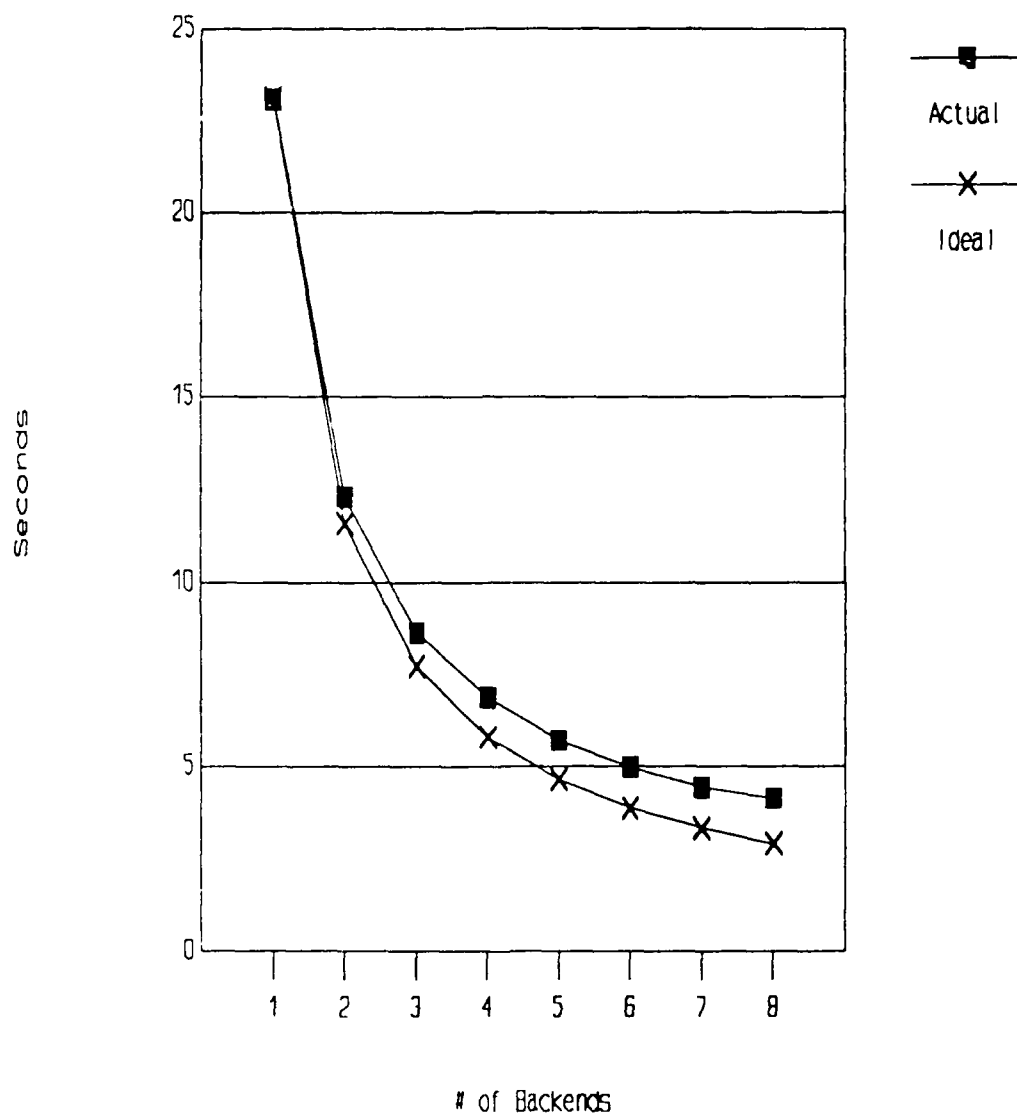


Figure 9. RTR Performance on Transaction #3

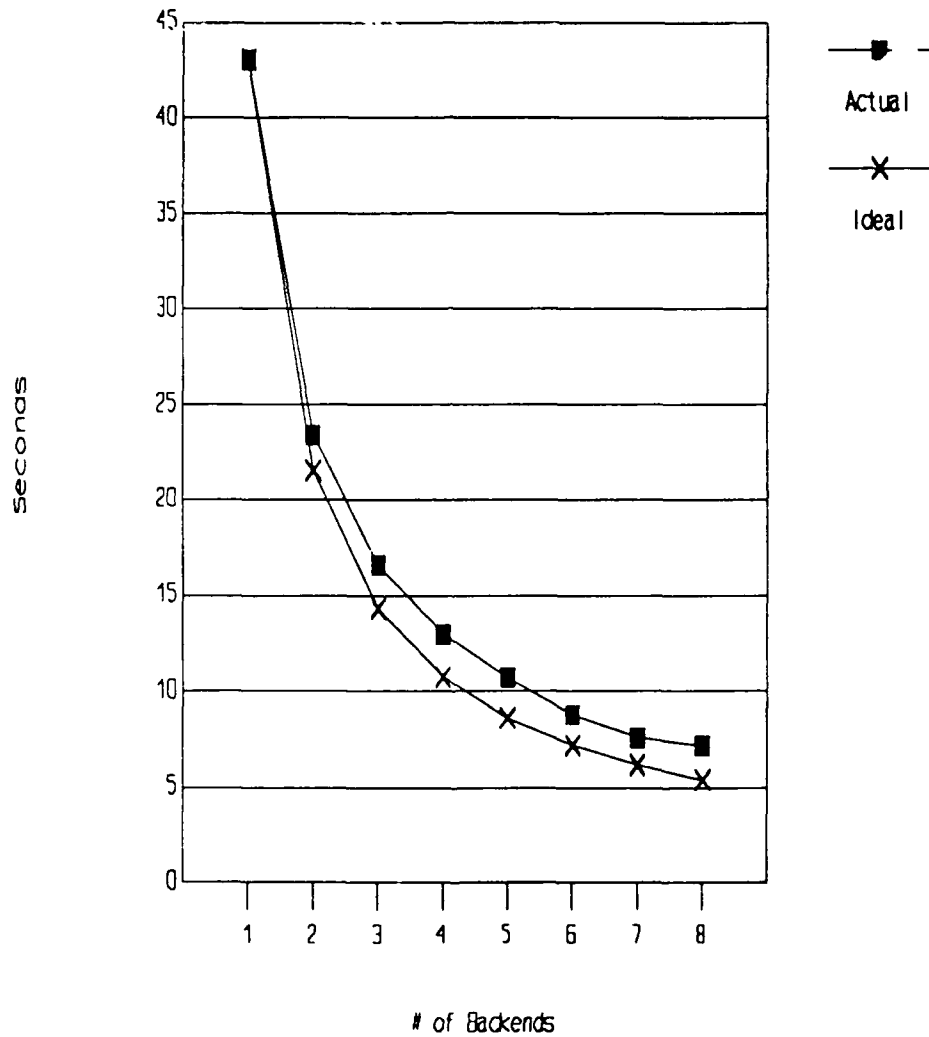


Figure 10. RTR Performance on Transaction #4

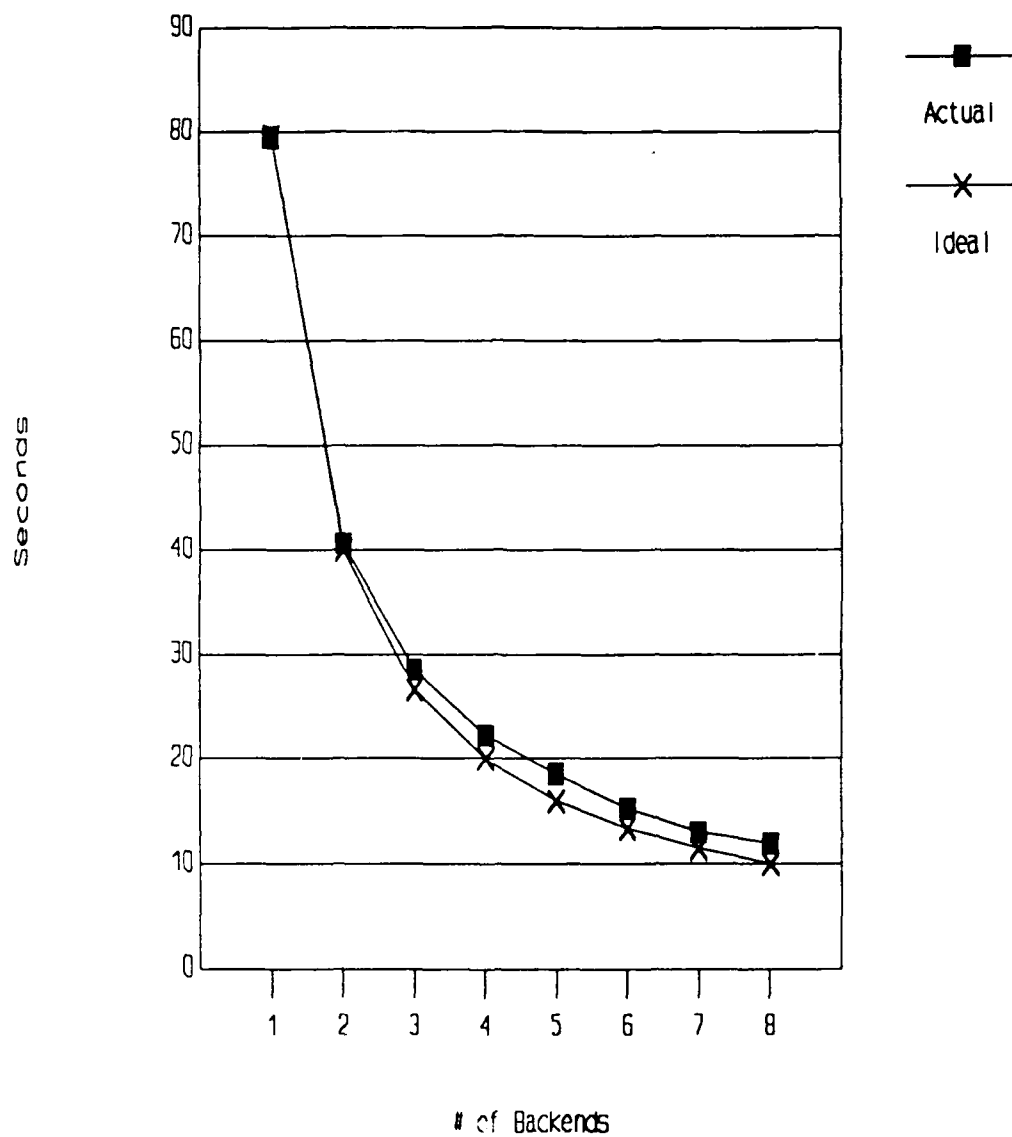


Figure 11. RTR Performance on Transaction #5

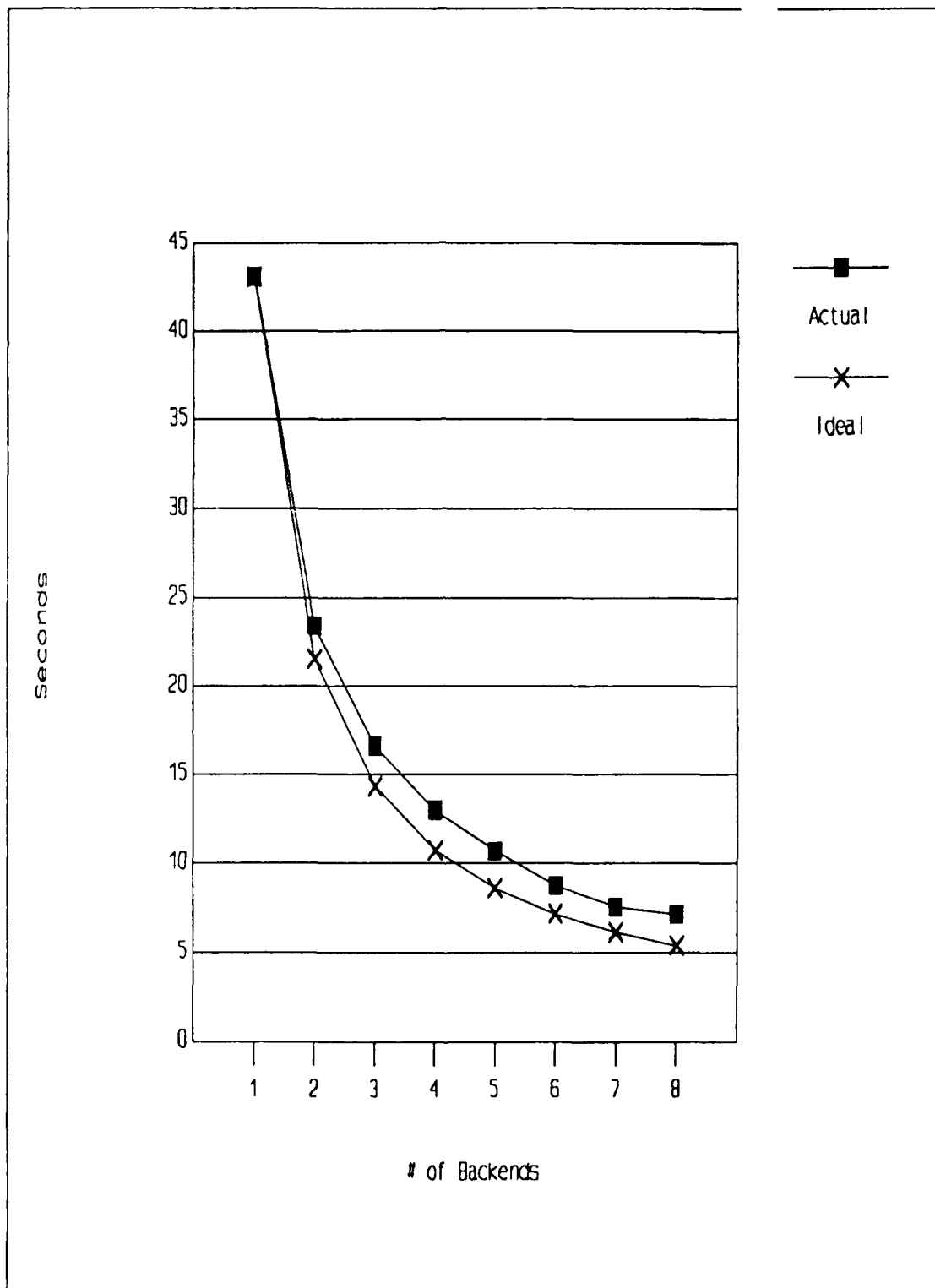


Figure 12. RTR Performance on Transaction #7

B. AN ANALYSIS OF MBDS RESPONSE-TIME INVARIANCE

1. The RTI Testing in General

The RTI testing uses the same test-transaction set as the RTR test, so the *percentage* of the database records accessed remains the same. It is the *size* of the test databases that makes the RTI a demanding test. The number of database records (and therefore the size of the database) is doubled, tripled and so on as the number of backends is incremented proportionally. The goal is to maintain baseline performance in spite of the increased load. It is virtually guaranteed that a conventional database system's response time would be doubled if the same test-transaction is subjected to the test by doubling the database size and number of records retrieved. MBDS attempts to overcome the conventional DBMS's response-time increases.

2. RTI Performance on Overhead-Intensive Transactions

RTI testing of MBDS during this study demonstrated the system's sensitivity to uneven loading when subjected to overhead-intensive transactions. Recall from Chapter V that the configurations with five, seven and eight backends were artificially loaded with perfectly even databases. Figure 13 gives a clear indication of the performance results. Perfect or ideal RTI performance would mean the matching of the one-backend configuration times. The unevenly loaded configuration response-times were close, but clearly the response-times suffered. These differences were measurable, but take note of the time scale in Figure 13. The variances were under one second for a given transaction.

Without the evenly loaded configurations, an incorrect conclusion might have been drawn by blaming the heavy increase in the database size for the loss of performance on overhead-intensive transactions. The evenly loaded configurations nearly matched the one-backend times with five, seven and eight times as large a database! The test-transactions still only accessed four percent of the database. However, much more meta data had to be searched. The

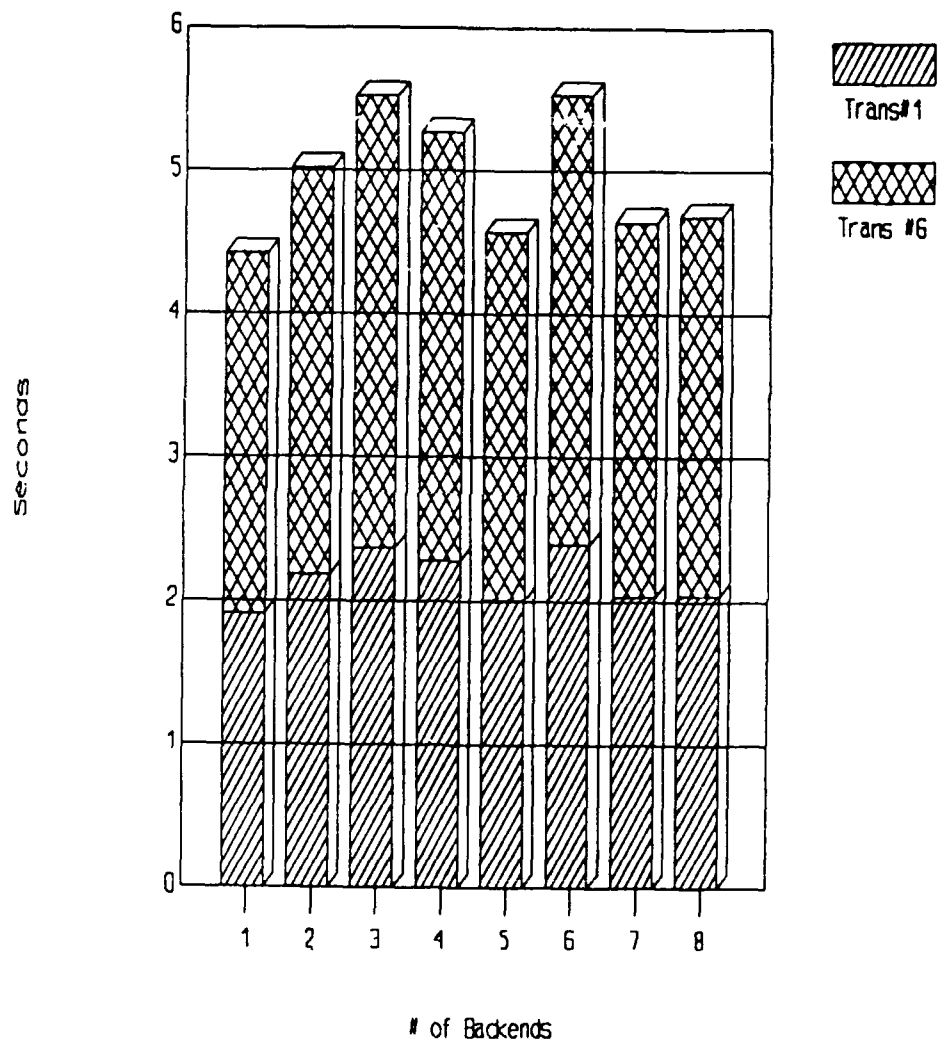


Figure 13. Overhead-Intensive RTI Performance Analysis

unevenly loaded configurations lagged behind the perfectly evenly loaded configurations because of the uneven loading and *not* because of an increase in system overhead. *A parallel system such as MBDS can only be as fast as the slowest backend.* By loading the backends unevenly, overall MBDS performance is limited to the performance of the backend with the heaviest load of records which match a given query. If maximum overhead-intensive transaction processing speed is a priority, a MBDS database administrator must take steps to ensure even database loading. If overhead-intensive transaction processing is infrequent and data-intensive transaction processing is the priority, the sub-second variances on overhead-intensive transactions are a small price to pay for a phenomenal increase in database size.

3. RTI Performance on Data-Intensive Transactions

Where the RTI testing demonstrated sensitivity to uneven loading on overhead-intensive transactions, the opposite was true of data-intensive transactions. Again, ideal RTI performance would mean matching the response-times of the one-backend or *baseline* configuration. Figure 14 shows MBDS test transaction performance during RTI testing (test transactions three and five access the same amount of the database and, therefore, have nearly identical response times and plotted over the top of one another). An ideal RTI test transaction performance would be a horizontal line on this graph. The actual RTI performance on the test transactions proved to be only slightly worse than the baseline times and seemingly oblivious to uneven loading. This indicates that minor imbalances between backends have little impact on the response-time of data-intensive transactions. The reason for this behavior is the relatively small number (100s to 1000s) of "extra" records the heavily loaded backend has to process compared to the total number of records read to meet the query (tens of thousands). Additional fractions or whole seconds have little effect when the query runs for *tens* of seconds. In general, the response-time remained relatively constant. Figure 15 is provided to show the minimal overall variance from the ideal baseline time.

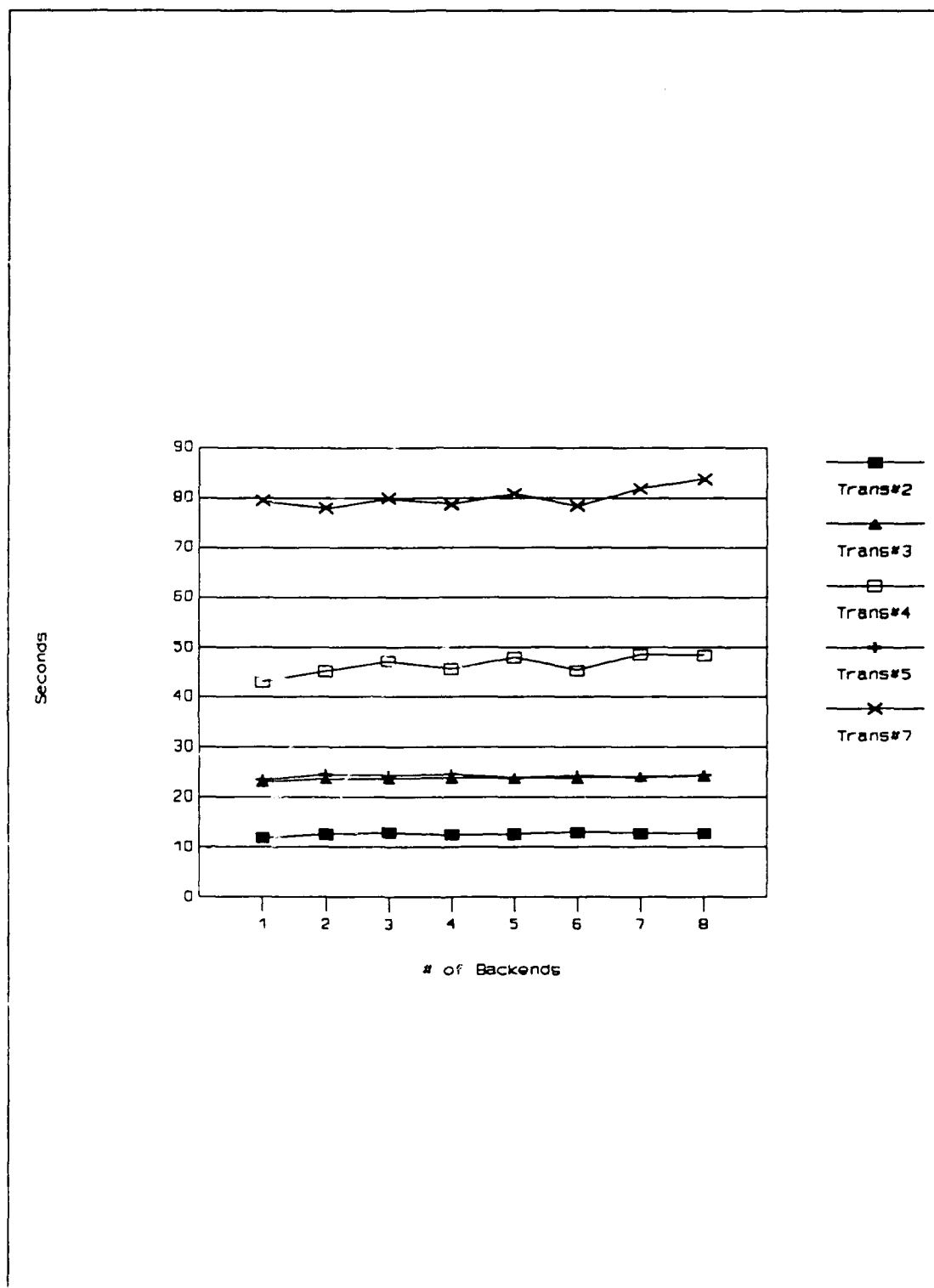


Figure 14. A Comparison Chart of Data-Intensive RTI Performance

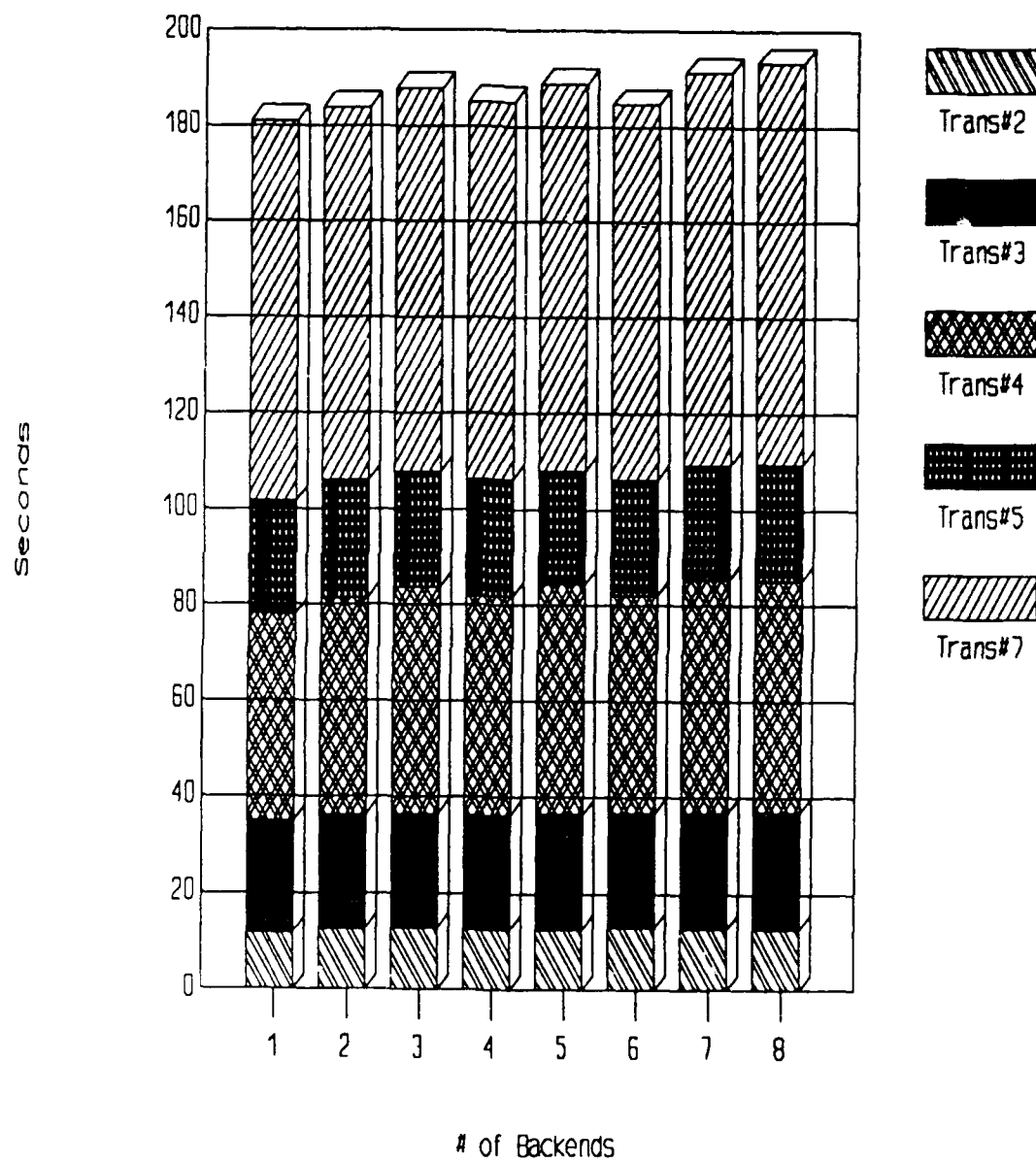


Figure 15. An Analysis of Data-Intensive RTI Performance

VII. CONCLUSIONS AND RECOMMENDATIONS

The focus of this thesis was the verification of performance claims made about MBDS. The system was designed to be configurable, to demonstrate response-time reductions and to demonstrate response-time invariances. To conduct the performance evaluation of MBDS, a previously designed and implemented computer-aided benchmarking methodology and system (CABS) was utilized for the first time. Difficulty in loading a truly large-scale test database required the use of a scaled-down database. However, the database size used was large enough to put MBDS through the most comprehensive test to date and to provide clear insight into the performance potential of MBDS.

During the overhead-intensive portion of the performance evaluation, MBDS demonstrated acceptable levels of response-time reductions and response-time invariances. Overhead-intensive transactions are the most difficult type of transaction for a parallel computer system to process, but MBDS did perform well in these tests. A notable sensitivity to uneven database distribution was observed during the overhead-intensive transaction testing. Optimal overhead-intensive transaction processing depends on even distribution of the database records among the backends.

During the data-intensive portion of the performance evaluation, MBDS demonstrated *strong* response-time reductions and response-time invariances. Data-intensive transactions are exactly what MBDS is designed to perform most effectively. A remarkable tolerance of uneven database distribution when processing data-intensive transactions was observed. Nearly optimal data-intensive transaction processing is achievable by MBDS in spite of moderate uneven database record distribution.

Future work in this area of study should begin with a study of the UPDATE and RETRIEVE-COMMON transactions. Next, an even larger-scale test should be attempted but this will require an off-line, high-speed database

mass-loading utility program. CABS would also have to be extended to produce output to tapes rather than to hard disks which limit file size. Another possible research topic is the analysis of the data placement algorithm used by MBDS. The random selection of the first backend which contains an available block of secondary storage for placing the first block of a given cluster of records causes (somewhat) uneven distribution of the database. Alternative first-backend selection criteria could be developed and tested. Overall, the benchmarking test results were encouraging and enlightening. MBDS has met its designed performance goals.

LIST OF REFERENCES

1. Hsiao, D.K., Guest editor's introduction: Database machines are coming, database machines are coming! *Computer*, Vol. 12, No.3.
2. Kernigan and Plaüger, *The Elements of Programming Style*, McGraw-Hill, 1978.
3. Ferrari, D., *Computer Systems Performance Evaluation*, Prentice-Hall, Englewood Cliffs, NJ, 1978.
4. Ferrari, et al., *Measurement and Tuning of Computer Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
5. Demurjian, Hsiao, and Menon, A Multi-Backend Database System for Performance Gains, Capacity Growth and Hardware Upgrade, 2nd International Conference on Data Engineering, IEEE Computer Society, Feb. 1986.
6. Banerjee, J. and Hsiao, D.K., DBC Software Requirements for Supporting Relational Databases, The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-77-7, November 1977.
7. Strawser, P. R., A Methodology for Benchmarking Relational Database Machines. Ph.D. Dissertation, The Ohio State University, Columbus, OH, 1984.
8. Tekampe, R. C. and Watson R. J., Internal and External Performance Measurement Methodologies for Database Systems, M.S. Thesis, Naval Post Graduate School, Monterey, CA, June 1984.
9. Vincent, J. R., A Performance Measurement Methodology for Software Multiple-Backend Database Systems, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1985.
10. Fenton, G. P., A Computer Aided Design for the Generation of Test Transactions and Test Databases and for the Benchmarking of Parallel, Multiple Backend Database Systems, M.S. Thesis, Naval Post Graduate School, Monterey, CA, June 1986.

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 0142
Naval Postgraduate School
Monterey, CA 93943-5002 | 2 |
| 3. | Commandant of the Marine Corps
Code TE 06
Headquarters, U.S. Marine Corps
Washington, D.C. 20380-0001 | 1 |
| 4. | Department Chairman, Code 52
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5002 | 1 |
| 5. | Curriculum Officer, Code 37
Computer Technology Programs
Naval Postgraduate School
Monterey, CA 93943-5002 | 1 |
| 6. | Professor David K. Hsiao, Code 52Hq
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5002 | 5 |
| 7. | Captain James E. Hall
MCTSSA
Communications and Intelligence Branch
Camp Pendleton, CA 92055-5018 | 1 |