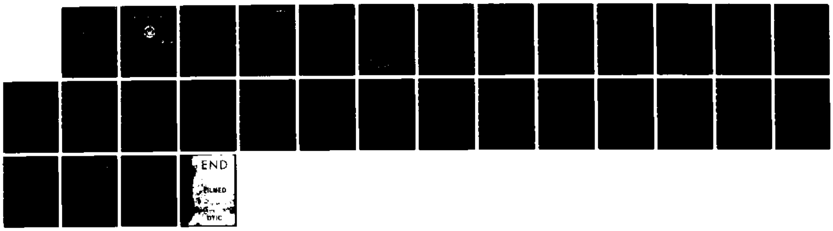
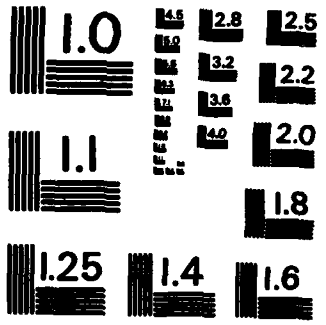
AD-A148 192 PERFORMANCE EVALUATION OF A DATABASE SYSTEM IN A MULTIPLE BACKEND CONFIGURATIONS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S A DEMURJIAN ET AL. OCT 84 1/1

UNCLASSIFIED NPS52-84-019 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

NPS52-84-019

NAVAL POSTGRADUATE SCHOOL

Monterey, California

PERFORMANCE EVALUATION OF A DATABASE SYSTEM IN A MULTIPLE BACKEND CONFIGURATIONS

Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr, Jai Menon, Paula R. Strawser, Robert C. Tekampe, Robert J. Watson

October 1984

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

The work reported herein was supported by Contract N00014-WR-24058 form the Office of Naval Research

Reproduction of all or part of this report is authorized.

This report was prepared by:

[illegible]
DAVID K. HSIAO
Professor of Computer Science

Reviewed by:

[illegible]
BRUCE J. MACLENNAN, Acting Chairman
Department of Computer Science

[illegible]
KNEALE T. MARSHALL
Dean of Information and Policy Sciences

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-84-019	2. GOVT ACCESSION NO. A148 192	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Performance Evaluation Of A Database System In a Multiple Backend Configurations		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr, Jai Menon, Paula R. Strawser, Robert C. Tekampe, Robert J. Watson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N; RR014-08-01 N0001484WR24058
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, Virginia 22217		12. REPORT DATE October 1984 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

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

18. SUPPLEMENTARY NOTES

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

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

The aim of this performance evaluation is twofold: (1) to devise benchmarking methodologies to the measurement of a prototyped database system in multiple backend configurations, and (2) to verify the performance claims as projected or predicted by the designer and implementor of the multi-backend database system known as MBDS.

Despite the limitation of the backend hardware, the benchmarking experiments have proceeded well, producing startling results and good insights. By collecting macroscopic data such as the response time of the request, the external

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

performance measurements of MBDS have been conducted. By collecting microscopic data such as the time entering and leaving a system process, the internal performance measurements of MBDS have been carried out. Methodologies for constructing test databases, directories, and requests have been devised and utilized. The performance evaluation studies verify that (a) when the database remains the same the response time of a request can be reduced to nearly half, if the number of backends and their disks is doubled; (b) when the response set of a request doubles, the response time of the query remains nearly constant, if the number of backends and their disks is doubled. These were the performance claims of MBDS as predicted by its designer and implementor.

i

PERFORMANCE EVALUATION OF A DATABASE SYSTEM IN MULTIPLE BACKEND CONFIGURATIONS *

Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr, Jai Menon,
Paula R. Strawser, Robert C. Tekampe, Robert J. Watson **

October 1984

ABSTRACT

The aim of this performance evaluation is twofold: (1) to devise benchmarking strategies for and apply benchmarking methodologies to the measurement of a prototyped database system in multiple backend configurations, and (2) to verify the performance claims as projected or predicted by the designer and implementor of the multi-backend database system known as MBDS.

Despite the limitation of the backend hardware, the benchmarking experiments have proceeded well, producing startling results and good insights. By collecting macroscopic data such as the response time of the request, the external performance measurements of MBDS have been conducted. By collecting microscopic data such as the time entering and leaving a system process, the internal performance measurements of MBDS have been carried out. Methodologies for constructing test databases, directories, and requests have been devised and utilized. The performance evaluation studies verify that (a) when the database remains the same the response time of a request can be reduced to nearly half, if the number of backends and their disks is doubled; (b) when the response set of a request doubles, the response time of the query remains nearly constant, if the number of backends and their disks is doubled. These were the performance claims of MBDS as predicted by its designer and implementor.

* The work reported herein is supported by Contract N00014-84-WR-24058 from the Office of Naval Research and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

** S. A. Demurjian and D. K. Hsiao are with the Laboratory for Database Systems Research, Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943. D. S. Kerr is with the Department of Computer Science, The Ohio State University, Columbus, OH, 43210. J. Menon is with the IBM San Jose Research Lab, San Jose, CA 95193. P. R. Strawser is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 10598. Robert C. Tekampe is with the Development Center, Marine Corps Development and Education Command, Quantico, VA, 22134. Robert J. Watson is with the Training Division, Headquarters, Marine Corps, Washington D. C., 20380.

1. INTRODUCTION

The multi-backend database system (MBDS) is a database system designed specifically for capacity growth and performance enhancement. MBDS consists of two or more minicomputers and their dedicated disk systems. One of the minicomputers serves as a controller to broadcast the requests to and receive the results from the other minicomputers, which are configured in a parallel manner and are termed as backends. All the backend minicomputers are identical, and run identical software. The database is evenly distributed across the disk drives of each backend by way of a cluster-based data placement algorithm unknown to the user. User access to the MBDS is accomplished either via a host computer, which in turn communicates with the MBDS controller, or with the MBDS controller directly. Communication between the controller and backends is accomplished using a broadcast bus. An overview of the system architecture is given in Figure 1.

There are two basic performance claims of the multi-backend database system, which have been projected in the original design goals [Hsia81a, Hsia81b]. The first claim states that if the database size remains constant, then the response time of requests processed by the system is inversely proportional to the multiplicity of backends. This claim implies that by increasing the number of backends in the system and by replicating the system software on the new backends, MBDS can achieve a reciprocal decrease in the response time for the same requests. The second claim states that the response time of requests is invariant when the response set and the multiplicity of backends increases in the same proportion. This claim implies that when the database size grows, the response set for the same requests will grow. By increasing the number of backends accordingly, MBDS can maintain a constant response time.

In this paper we provide a preliminary evaluation of the validity of the MBDS performance claims. The main focus of this paper is on the external performance measurement of MBDS. The external performance measurement evaluates a system by collecting the response times of requests. External performance measurement is a macroscopic evaluation of the system. Ingres, Oracle, and the Britton-Lee IDM/500, have all been evaluated using external performance measurement techniques [Stra84, Schi84]. We also seek to provide some insight into the internal performance of MBDS. Internal performance measurement provides a microscopic view of a system, by collecting the times of the

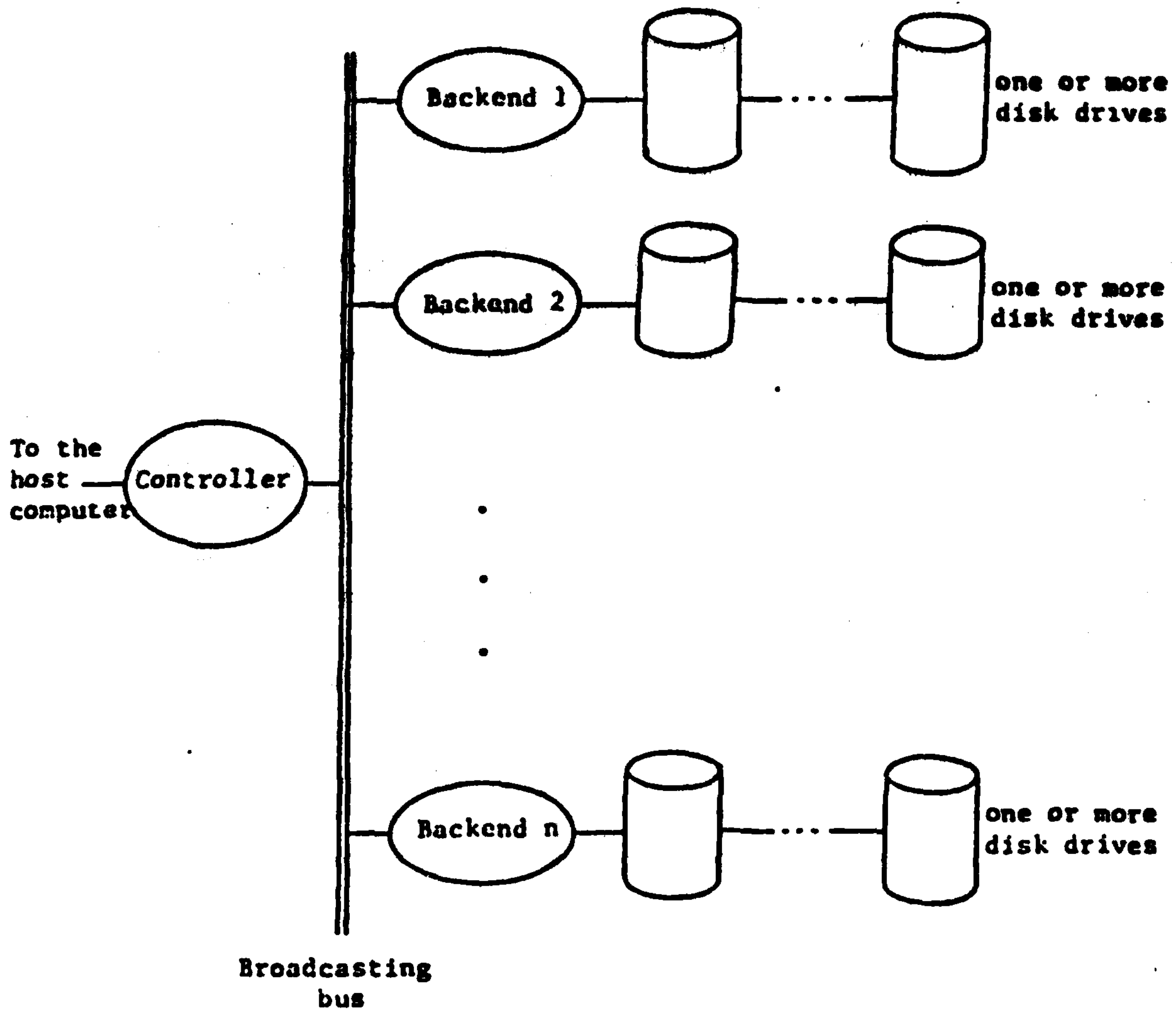


Figure 1. The MDBS Hardware Organization

work distributed and performed by the system components.

The remainder of this paper is organized as follows. In Section 2 we provide a brief overview of the multi-backend database system. In Section 3 we discuss the general testing strategy that was used to evaluate the system. In Section 4 we examine the evaluation results. Finally, in Section 5 we conclude this paper and summarize the results.

2. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

The current hardware configuration of MBDS consists of a VAX-11/780 (VMS OS) running as the controller and two PDP-11/44s (RSX-11M OS) running as backends. Intercomputer communication is supported by three parallel communication links (PCL-11Bs), which is a time-divisioned-multiplexed bus. The implementation efforts are documented in [Kerr82, He82, Boyn83a, Demu84]. MBDS is a message-oriented system (see [Boyn83b]). In a message-oriented system, each process corresponds to one system function. These processes, then, communicate among themselves by passing messages. User requests are passed between processes as messages. The message paths between processes are fixed for the system. The MBDS processes are created at the start-up time and exist throughout the entire running time of the system.

In the rest of this section we begin by discussing the data model of MBDS, the attribute-based data model. Then, we present a brief review of the attribute-based data language (ABDL) of MBDS by focusing on the retrieve request only. Next, we discuss the directory structure used in the system, since it plays an integral role in the specification of the test database (see Section 3). Lastly, we overview the execution of a RETRIEVE request, to provide a general overview of the structure and the operation of MBDS.

2.1. The Attribute-Based Data Model

In the attribute-based data model, data is modeled with the constructs: database, file, record, attribute-value pair, directory keyword, directory, record body, keyword predicate, and query. Informally, a <u>database</u> consists of a collection of files. Each <u>file</u> contains a group of records which are characterized by a unique set of directory keywords. A <u>record</u> is composed of two parts. The first part is a collection of <u>attribute-value pairs</u> or <u>keywords</u>. 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, <POPU-

LATION, 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 addressing the record (file). Those attribute-value pairs which are not kept in the directory for addressing the record (file) are called non-directory keywords. The rest of the record is textual information, which is referred to as the record body. An example of a record is shown below.

(<FILE, Census>, <CITY, Monterey>, <POPULATION, 25000>, { Temperate climate })

The angle brackets, <,>, enclose an attribute-value pair, i.e., keyword. The curly brackets, {,}, include the record body. The first attribute-value pair of all records of a file is the same. In particular, the attribute is FILE and the value is the file name. A record is enclosed in the parenthesis. For example, the above sample record is from the Census file.

The database is accessed by indexing on directory keywords using keyword predicates. A keyword predicate is a tuple consisting of an attribute, a relational operator (=, !=, >, <, >=, <=), and an attribute value, e.g., POPULATION >= 20000 is a keyword predicate. More specifically, it is a greater-than-or-equal-to predicate. Combining keyword predicates in disjunctive normal form characterizes a query of the database. The query

(FILE = Census and CITY = Monterey) or
(FILE = Census and CITY = San Jose)

will be satisfied by all records of the Census file with the CITY of either Monterey or San Jose. For clarity, we also employ parentheses for bracketing predicates in a query.

2.2. The Attribute-Based Data Language

MBDS is designed to perform the primary database operations, INSERT, DELETE, UPDATE, and RETRIEVE. Additionally, an aggregate operation (i.e., AVG, COUNT, SUM, MIN, or MAX) may be applied when using the RETRIEVE operation. Users access MBDS through the host computer or the controller by issuing a request. A request is a primary operation along with a qualification. A qualification is used to specify the information of the database that is to be performed by the primary operation. A user may wish to treat two or more requests as a transaction. In this situation, MBDS executes the requests of the transaction without permuting them, i.e., if T is a transaction containing the requests <R1><R2>, then MBDS executes the request R1 before the request R2. Finally, we define the term traffic-unit to represent either a single request or a transaction in execution.

Now, let us examine the retrieve request, the main focus of our study in this paper. An example of a retrieve request would be:

RETRIEVE ((FILE = Census) and (POPULATION > 10000)) (CITY)

which retrieves the names of all those cities in the Census file whose population is greater than 10000. Notice that the qualification component of a retrieve request consists of two parts: the query which specifies records of the database to be retrieved and the target list which specifies the attribute-value(s) to be returned to the user. Aggregate operators may be applied to attributes listed in the target list. In this example, there is the query of two predicates ((FILE = Census) and (POPULATION > 10000)) and the target list (CITY).

2.3. The Directory Tables

To manage the database (often referred to as user data), MBDS uses directory data. Directory data in MBDS corresponds to attributes, descriptors, and clusters. An attribute is used to represent a category of the user data; e.g., POPULATION is an attribute that corresponds to actual populations stored in the database. A descriptor is used to describe a range of values that an attribute can have; e.g, (10001 < POPULATION < 15000) is a possible descriptor for the attribute POPULATION. The descriptors that are defined for an attribute, e.g., population ranges, are mutually exclusive. Now the notion of a cluster can be defined. A cluster is a group of records such that every record in the cluster satisfies the same set of descriptors. For example, all records with POPULATION between 10001 and 15000 may form one cluster whose descriptor is the one given above. In this case, the cluster satisfies the set of a single descriptor. In reality, a cluster tends to satisfy a set of multiple descriptors.

Directory information is stored in three tables: the Attribute Table (AT), the Descriptor-to-Descriptor-Id Table (DDIT) and the Cluster-Definition

Table (CDT), examples of which are given in Figure 2. The Attribute Table maps directory attributes to the descriptors defined on them. A sample AT is depicted in Figure 2a. The Descriptor-to-Descriptor-Id Table maps each descriptor to a unique descriptor id. A sample DDIT is given in Figure 2b. Note that the pointers shown in Figure 2b are not placed in the DDIT table but are shown here for clarity in order for us to relate to the AT of Figure 2a. The Cluster-Definition Table maps descriptor-id sets to cluster ids. Each entry consists of the unique cluster id, the set of descriptor ids whose descriptors define the cluster, and the addresses of the records in the clusters. A sample CDT is shown in Figure 2c Thus, to access the user data, we must first access directory data via the AT, DDIT, and CDT.

In designing the test database, one of the key concepts is the choice of the directory attributes in order to determine the necessary descriptors and therefore clusters. Thus, we provide a brief introduction to the three classifications of descriptors. A type-A descriptor is a conjunction of a less-than-or-equal-to predicate and a greater-than-or-equal-to predicate, such that the same attribute appears in both predicates. For example, ((POPULATION >= 10000) and (POPULATION <= 15000)) is a type-A descriptor. A type-B descriptor consists of only an equality predicate. (FILE = Census) is an example of a type-B descriptor. Finally, a type-C descriptor consists of the name of an attribute. The type-C attribute defines a set of type-C sub-descriptors. Type-C sub-descriptors are equality predicates defined over all unique attribute values which exist in the database. For example, the type-C attribute CITY forms the type-C sub-descriptors (CITY=Cumberland) and (CITY=Columbus), where "Cumberland" and "Columbus" are the only unique database values for the CITY.

2.4. The Execution of a Retrieve Request

In this section, we describe the sequence of actions for a retrieve request as it moves through MBDS. The sequence of actions will be described in terms of the messages passed between the MBDS processes. The MBDS controller processes are Request Preparation (REQP), Insert Information Generation (IIG), and Post Processing (PP). The MBDS backend processes are Directory Management (DM), Record Processing (RECP) and Concurrency Control (CC). For completeness, we describe the actions which require data aggregation.

Attribute	Ptr
POPULATION	P
CITY	C
FILE	F

Figure 2a. An Attribute Table (AT)

P->	0 < POPULATION < 50000	D11
	50001 < POPULATION < 100000	D12
	100001 < POPULATION < 250000	D13
	250001 < POPULATION < 500000	D14
C->	CITY = Cumberland	D21
	CITY = Columbus	D22
F->	FILE = Employee	D31
	FILE = Census	D32

Dij: Descriptor j for attribute i.

Figure 2b. A Descriptor-to-Descriptor-Id Table

Id	Desc-Id Set	Addr
C1	{D11,D21,D31}	A1,A2
C2	{D14,D22,D32}	A3

Figure 2c. A Cluster-Definition Table (CDT)

Figure 2. The Directory Tables

First the retrieve request comes to REQP from the host computer or the controller itself. REQP sends two messages to PP: the number of requests in the transaction and the aggregate operator of the request. The third message sent by REQP is the parsed traffic unit which goes to DM in the backends. DM sends the type-C attributes needed by the request to CC. Since type-C attributes may create new type-C sub-descriptors, the type-C attributes must be locked by CC. Once an attribute is locked and descriptor search can be performed, CC signals DM. DM will then perform Descriptor Search on m/n predicates, where m is the number of predicates specified in the query, and n is the number of backends. DM then signals CC to release the lock on that attribute. DM will broadcast the descriptor ids for the request to the other backends. DM now sends the descriptor-id groups for the retrieve request to CC. A descriptor-id group is a collection of descriptor ids which define a set of clusters needed by the request. Descriptor-id groups are locked by CC, since a descriptor-id group may define a new cluster. Once the descriptor-id groups are locked and Cluster Search can be performed, CC signals DM. DM will then perform Cluster Search and signal CC to release the locks on the descriptor-id groups. Next, DM will send the cluster ids for the retrieval to CC. CC locks cluster-ids, since a new address may be specified for an existing cluster. Once the cluster ids are locked, and the request can proceed with Address Generation and the rest of the request execution, CC signals DM. DM will then perform Address Generation and send the retrieve request and the addresses toe RECP. Once the retrieval has executed properly, RECP will tell CC that the request is done and the locks on the cluster ids can be released. The retrieval results are aggregated by each backend and forwarded to PP. PP completes the aggregation after it has received the partial results from every backend. When PP is done, the final results will be sent to the user.

3. THE BENCHMARK STRATEGY

In this section we analyze the basic benchmark strategy for the preliminary performance evaluation of the multi-backend database system. The benchmark strategy focuses on collecting macroscopic and microscopic measurements on the systems performance. Macroscopic measurements correspond to the external performance measurement of the system, which collects the response time of requests that are processed by the system. Internal performance measurement involves the detailed measurement of the working processes of the system. In particular, we are measuring the time taken to process a particular message in MBDS. Each MBDS process has a group of functions, called message-handlers, that control and oversee the processing of a message. The time spent in a particular message-handler is collected in internal performance measurement.

To adequately conduct both the internal and external performance measurement of the system, software was developed to collect timing information and data. The performance software was bracketed in conditional compilation statements to facilitate an easy transition between a testing system and a running system. We constructed two software bases of the MBDS. The first consisted of the MBDS code and only the testing software required for external performance measurement. The second had the testing software for both the internal and external performance measurement software compiled in. We hope to use the difference in timings collected from the two bases to calculate the overhead incurred by the addition of internal performance measurement software.

The rest of this section is organized as follows. First, we give a high-level description of the test database organization and system configurations used in the performance evaluation. Next, we present a detailed discussion of the test database organization. Third, we examine the request set used to collect the timings. Finally, we review the relevant tests that are to be conducted, and the measurement statistics that are collected and calculated.

3.1. The Test Database Organization and Testing Configurations

To properly evaluate a database system, various record sizes need to be used. The sizes are chosen based on the size of the unit of disk management. In MBDS, this is the block. MBDS processes information from the secondary memory using a 4Kbyte block. Given a blocksize of 4Kbytes, it was recommended to construct the database with record sizes of 200 bytes, 400 bytes, 1000 bytes, and 2000 bytes [Stra84]. This gives a range of 2 to 20 records per block. Since we are engaged in only the first test of MBDS, we limited the scope of the testing to a database with a 200 byte record size.

In addition, the virtual and physical memory limitations of each backend restricted the database size to a maximum of 1000 records per backend. This limitation, coupled with the two software versions of the system and the need to verify the two performance claims, led us to the specification of five different system configurations for the MBDS performance measurements. Table 1

TEST	No. of Backends	Records/Backend	Database Size
A.E	1	1000	200K bytes
B.E	2	500	200K bytes
C.E	2	1000	400K bytes
A.I	1	1000	200K bytes
B.I	2	500	200K bytes

Table 1. The Measurement Configurations

displays the configurations.

Tests A.E, B.E, and C.E are conducted without internal performance software in place. Test A.E configures MBDS with one backend and one thousand records in the test database. Test B.E configures MBDS with two backends and one thousand records split evenly between the backends. The transition from Test A.E to Test B.E is used to verify the first performance claim (see Section 1). Test C.E also configures MBDS with two backends, but, the size of the database is doubled to two thousand records. The transition from Test A.E to Test C.E is used to verify the second performance claim (see Section 1). Test A.I and B.I are conducted with internal performance software in place. Test A.I configures MBDS with one backend and one thousand records in the test database. Test B.I configures MBDS with two backends and one thousand records split evenly between the backends. The transitions from A.E to A.I and from B.E to B.I are used to determine the overhead incurred by the addition of internal performance measurement software. Overall, using these five configurations, the verification of the MBDS performance and capacity claims is simplified and the performance measurement methodology of computing the internal measurement overhead is facilitated.

3.2. The Detailed Test Database Organization

We have chosen the test record size to be 200 bytes. The 200-byte record minimizes the primary memory required to store the record template. In actuality, a record of 198 bytes is used. The record consists of 33 attributes, each requiring 6 bytes of storage. The record template is used to specify the attributes, both the directory and the non-directory attributes, of the record. Of the 33 attributes listed (see Figure 3), INTE1 and INTE2 are directory attributes. MULTI and STR00 to STR29 are non-directory attributes.

Attribute Name	Attribute Type
INTE1	integer
INTE2	integer
MULTI	string
STR00	string
STR01	string
.	.
.	.
.	.
STR29	string

Figure 3. The Record Template

The descriptor types and the descriptor ranges for the two directory attributes, INTE1 and INTE2, must also be defined. The values for INTE1 are classified by using five type-A descriptors, each of which represents a range of 200, i.e., the ranges would be [1,200], [201,400], ..., [801,1000], where [a,b] is used to represent the type-A descriptor range. The values for INTE2 are also classified using type-A descriptors. The first twenty-three ranges for INTE2 cover 40 values, with the last range covering 80 values, i.e., the type-A descriptor ranges would be [1,40], [41,80], ..., [881,920], [921,1000]. The INTE2 descriptor ranges are not uniform.

Next, we examine the records which are generated and stored in the test database. INTE1 and INTE2 have identical value, i.e., numbers, being the next sequential number after the previous record, starting at 1. Therefore, the one thousandth record would have the (INTE1, INTE2) pair set to 1000. The MULTI attribute, which is of the type of character string, is set to One for a database of only 1000 records. The intent of this attribute is to increase the number of records per cluster in the database. This is done by setting MULTI to Two, Three, etc., for each (INTE1, INTE2) pair in the database. Therefore, to double the size of the database, every (INTE1, INTE2) pair will have an associated MULTI attribute with values of One and Two. The remaining attributes, STR00 to STR29, are set to Xxxxx as fillers for the rest of the record body. Figure 4 depicts the general layout of the file for 1000 records where MULTI is set to One.

INTE1	INTE2	MULTI	STR00	STR01	...	STR29
1	1	One	Xxxxx	Xxxxx	...	Xxxxx
2	2	One	Xxxxx	Xxxxx	...	Xxxxx
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
1000	1000	One	Xxxxx	Xxxxx	...	Xxxxx

Figure 4. The Generated Records

The cross-product of INTE1 ranges and INTE2 ranges has resulted in the specification of 24 descriptor groups for the INTE1 and INTE2 attributes. Coupled with the record template, they generate a test database that contains 24 clusters. The first 23 clusters contain 40 records each. The last cluster contains 80 records. To maintain consistency in the retrieval requests (discussed in the next section), we avoid any requests that access the last 80 records in the test database using the INTE2 attribute.

3.3. The Request Set

The request set used for our performance measurement is given in Figure 5. The retrievals are a mix of single or double predicate requests. Since the majority of the work done on a database is to retrieve data, we limit our first measurements to only retrieve requests. In every request, 1/2 of the target attribute values for each record is returned. The first request is for only two records from two separate clusters. The second request retrieves 1/4 of the database. Seven of the 24 clusters must be examined. All records in each of the first six clusters are retrieved. Only 1/4 of the seventh cluster, defined by the INTE2 range from 241 to 280, is retrieved. In the third request, 1/2 of the database is retrieved. Thirteen of the 24 clusters must be examined. All records in each of the first twelve clusters are returned. Only 1/2 of the thirteenth cluster, defined by the INTE2 range from 481 to 520, is retrieved. The system searches only for records having values in the INTE2 range from 481 to 500 in this cluster.

The entire database is examined in the fourth request. The fifth request retrieves 2/5 of the database. The query is divided into two predicates, to

Request Number	Retrieval Request
1	(INTE1=10) or (INTE1=230)
2	(INTE2 < 250)
3	(INTE2 < 500)
4	(INTE1 < 1000)
5	(INTE1<200) or (INTE1>801)
6	(INTE1<400) or (INTE1>601)
7	(INTE1 <= 201)
8	(INTE1 <= 401)
9	(INTE1<=201) or (INTE1>=800)
The Target Attribute-Values for Each: (INTE1,INTE2,MULTI,STR00,STR01,STR02,STR03,STR04, STR05,STR06,STR07,STR08,STR09,STR10,STR11,STR12)	

Figure 5. The Retrieval Requests

obtain all records from the first five clusters, and the last four clusters. The sixth request is a retrieval of 4/5 of the database. Again the query is divided into two predicates, to obtain all records from the first 10 clusters, and the last nine clusters.

The seventh and eighth requests are similar in intent. The seventh request examines 10 clusters, requiring only 1 record to be retrieved from the 6th cluster and needing all records from the first five clusters. The eighth request examines 15 clusters, requiring only 1 record to be retrieved from the 11th cluster and needing all records from the first ten clusters. The ninth and final request is similar to the fifth request. But unlike the fifth request, ten additional clusters must be examined. Only two of the records with INTE1 values of 201 and 801, are retrieved from the ten additional clusters. All records in the remaining nine clusters, like the fifth request, are also obtained by this retrieval. Table 2, a presentation of the number of clusters examined versus the percent of the database retrieved, is a synopsis of the previous discussion in tabular form.

3.4. The Measurement Strategy, Statistics and Limitations

The basic measurement statistics used in the performance evaluation of MBDS is the response time of request(s) that are processed by the database system. The <u>response time</u> of a request is the time between the initial

Request Number	Number of Clusters Examined	Volume of Database Retrieved
1	2	2 records
2	7	25%
3	13	50%
4	24 (all)	100%
5	9	40%
6	19	80%
7	10	20% + 1 record
8	15	40% + 1 record
9	19	40% + 2 records

Table 2. The Number of Clusters Examined and the Percent of the Database Retrieved

issuance of the request by the user and the final receipt of the entire request set for the request. The response times are collected for the request set (see Figure 5) for each of the five configurations (see Table 1). Each request is sent a total of ten times per database configuration. The response time of each request is recorded. We determine that ten repetitions of each request produce an acceptable standard deviation. Upon completion of the ten repetitions for a request, we calculate the mean and the standard deviation of the ten response times. There are two main statistics that we calculate to evaluate the MBDS performance claims, the response-time improvement and the response-time reduction.

The <u>response-time improvement</u> is defined to be the percentage improvement in the response time of a request, when the request is executed in n backends as opposed to one backend and the number of records in the database remains the same. Equation 1 provides the formula used to calculate the response-time improvement for a particular request, where Configuration B represents n backends and Configuration A represents one backend. The response-time improvement is calculated for the configuration pairs (A.E, B.E) and (A.I, B.I), respectively. The configuration pair (A.E, B.E) is evaluated for the retrieve requests (1) through (9) (see Tables 5 and 6). The pair (A.I, B.I) is evaluated only for the retrieve requests (1) through (6). Overall, the difference in the collected times of the two configurations, i.e., (A.I - A.E), and (B.I - B.E), respectively, should provide us with a measure of the overhead incurred when internal performance measurement software is present in the system.

$$\text{The Response-Time Improvement} = 100\% - \frac{\text{The Response Time of Configuration A}}{\text{The Response Time of Configuration B}} * 100\%$$

Equation 1. The Response-Time-Improvement Calculation

The <u>response-time reduction</u> is defined to be the reduction in response time of a request, when the request is executed in n backends containing nx number of records as opposed to one backend with x number of records. Equation 2 provides the formula used to calculate the the response-time reduction for a particular retrieval request, where configuration A represents one backend with x records and configuration B represents n backends, each with x records. The response-time reduction is calculated for the configuration pair (A.E, C.E), for the retrieve requests (1) through (9).

$$\text{The Response-Time Reduction} = 100\% * 1 - \frac{\text{The Response Time of Configuration B}}{\text{The Response Time of Configuration A}}$$

Equation 2. The Response-Time-Reduction Calculation

The internal processing times of the message-handling routines which are used to process a retrieval request are also timed. Retrieval (1) and Retrieval (2) are selected to conduct internal timing. These requests are selected since they retrieve the smallest portion of the test database and the processing time for each request is minimal. Each message-handling routine is timed independently of all others and each routine must process multiple requests so that an accurate average may be computed for the time required to process that request type. Sixteen message-handling routines are required to process a retrieve request. If we send twenty requests to each routine, a total of 320 requests must be processed by MBDS. Based on these figures, the time required to conduct the internal performance measurement of a retrieval that has a response time of twenty seconds will be approximately 107 minutes. This figure does not include the administrative time required to process the internal measurement data. For this reason, we limited the internal performance measurement requests to requests (1) and (2).

Additionally, we also limited the number of repetitions per message handler to twenty. This is done to reduce the processing time per message handler. However, this decision reduces the accuracy of the internal performance measurement, from ten-thousands to hundredths of a second. Thus, the internal performance measurement times provide only a rough estimate of the time required to handle the respective messages. There are additional limitations. The last two versions of MBDS differ in the implementation of the directory tables, i.e., the AT, the DDIT, and the CDT. The newest version of the system, called Version F, implements the directory tables on the secondary storage. The previous version, called Version E, stored the directory tables in the primary memory. The major roadblock that we have encountered in the performance measurement of MBDS has been the hardware limitations of the backend processors (PDP-11/44). With only 64K of virtual memory per process and a total of 256K physical memory, we found that we could not increase the MBDS system parameters to permit an extensive test of the system on a large database. These restrictions have forced us to benchmark the primary-memory-based directory management version of the system which, excluding the directory table management routines, is nevertheless equivalent in functionality to Version F.

4. THE BENCHMARKING RESULTS

In this section, we present the results obtained from the performance measurement of MBDS. We also review the results of external performance measurement, overhead incurred by internal performance measurement software and internal performance measurement. One final note, the units of measurement presented in the tables of this section are expressed in seconds.

4.1. The External Performance Measurement Results

Table 3 provides the results of the external performance measurement of MBDS without the internal performance measurement software. There are three parts to Table 3. Each part contains the mean and the standard deviation of the response times for requests (1) through (9), which are outlined in Section 3.3. The three parts of Table 3 represent three different configurations of the MBDS hardware and the database capacity. The first part has configured MBDS with one backend and the database with 1000 records on its disk. The second part has configured MBDS with two backends, with the database of 1000 records, split evenly between the disks of the backends. The third part has configured MBDS with two backends and with a database doubled of 2000 records, where the disk of each backend has 100 records. In Table 3 we notice one data anomaly, the standard deviation for request (9) in the one-backend-with-1000-records configuration. Since we did not conduct an internal performance measurement on this request, we are not sure what causes this skewed standard deviation, and hence will not attempt to offer an explanation of this anomaly.

Given the data presented in Table 3, we can now attempt to verify or disprove the two MBDS performance claims. We begin by calculating the response-time improvement for the nine requests. In Table 4 we present the response-time improvement for the data given in Table 3. Notice that the response-time improvement is lowest for request (1), which represents a retrieval of two records of the database. On the other hand, the response-time

Request Number	One Backend 1K Records (A.E)		Two Backends 1K Records (B.E)		Two Backends 2K Records (C.E)	
	mean	stdev	mean	stdev	mean	stdev
1	3.208	0.0189	2.051	0.0324	3.352	0.0282
2	13.691	0.0255	7.511	0.0339	14.243	0.0185
3	26.492	0.0244	14.164	0.0269	26.737	0.0405
4	52.005	0.0539	26.586	0.0294	52.173	0.0338
5	21.449	0.0336	11.309	0.0375	21.550	0.0237
6	42.235	0.0326	21.622	0.0424	42.287	0.0400
7	12.285	0.0408	6.642	0.0289	12.347	0.0371
8	22.532	0.0296	11.764	0.0300	22.583	0.0110
9	23.913	0.1115	12.624	0.0350	24.169	0.0181

Table 3. The Response Time Without Internal Performance Evaluation Software

Request Number	Response-Time Improvement (A.E,B.E)
1	36.07
2	45.14
3	46.53
4	48.94
5	47.27
6	48.81
7	45.93
8	47.79
9	47.21
No Internal-Measurement Software	

Table 4. The Response-Time Improvement Between Configurations A.E and B.E.

improvement of request (4), which retrieves all of the database information is highest, approaching the upper bound of fifty percent. In general, we find that the response-time improvement increases as the number of records retrieved increases. This seems to support a hypothesis that even if the database is larger, the response-time improvement will remain at a relatively high (between 40 an 50 percent) level.

Next, we calculate the response-time reduction for each of the nine request. In Table 5 we present the response-time reductions for the data given in Table 3. Notice that the response-time reduction is worst for request (1),

Request Number	Response-Time Reduction (A.E,C.E)
1	4.49
2	4.03
3	0.92
4	0.32
5	0.47
6	0.12
7	0.50
8	0.23
9	1.07
1K Records on each Backend No Internal-Measurement Software	

Table 5. The Response-Time Reduction Between Configurations A.E and C.E

which represents a retrieval of two records of the database. On the other hand, the response-time reductions for the requests which access larger portions of the database, requests (4) and (6), have only a small response-time reduction. In general, we found that the response-time reduction decreases as the number of records retrieved increases, i.e., the response time remains virtually constant. Again we seem to have evidence to support the hypothesis that, as the size of the response set increases for the same request, the response-time reduction will decrease to a relatively low (0.1% or less) level.

Table 6 provides the results of external performance measurement of MBDS with internal performance measurement software in place. There are two parts to Table 6. Each part contains the mean and the standard deviation of the response-times for the requests (1) through (6). The two parts of Table 6 represent two different configurations of the MBDS hardware and the database capacity. Part one has configured MBDS with one backend and with the database of 1000 records. Part two has configured MBDS with two backends, with the database of 1000 records, split evenly between the disks of the backends. We did not measure the response times with two thousand records distributed over two backends. We felt that no additional information would be gained by conducting the measurements.

Request Number	One Backend 1K Records (A.I)		Two Backends 1K Records (B.I)	
	mean	stdev	mean	stdev
1	3.205	0.0436	2.219	0.0474
2	13.418	0.0172	7.401	0.0277
3	25.903	0.0119	13.854	0.0361
4	50.750	0.0374	26.402	0.0596
5	20.972	0.0271	11.244	0.0528
6	41.262	0.0331	21.517	0.0575

Table 6. The Response Time (in seconds) With Internal Performance Measurement Software

4.2. The Internal Performance Measurement Overhead

An interesting anomaly is discovered when we compare the response times of the external and internal performance measurement tests, i.e., parts one and two of Tables 3 and 6 for requests (1) through (6). We had anticipated that the addition of internal performance measurement software would add an overhead to the response time of requests. In the transition from A.E to A.I and from B.E to B.I we expected there to be in an increase in the response times for the common requests. We actually found a general improvement, from 0.1% to 5%, in the response times of the requests when the internal performance measurement software is part of the MBDS code. What could have caused the anomaly? One hypothesis is that this is due to the manner in which MBDS is implemented on the backends. Currently, there is not sufficient virtual memory per process available on each backend. The result is that disk overlays are used to organize the code for each process in MBDS. The additional internal performance measurement code may cause the operating system to overlay differently, thereby benefiting the overall performance of MBDS. We still believe that there is an overhead induced by the internal measurement code and Table 7 provides evidence by demonstrating that the response-time improvement achieved by adding a backend is inferior to the corresponding figures in Table 4.

Request Number	Response Time Improvement (A.I,B.I)
1	30.76
2	44.84
3	46.52
4	47.98
5	46.39
6	47.85
1K Records Evaluation Software	

Table 7. The Response Time Improvement Between Configurations A.I and B.I.

4.3. The Internal Performance Measurement Results

Table 8 provides the results of the internal performance measurement of MBDS for a retrieval request. The times measured for each message-handling routine are given for both request (1) and (2). The message-handling routines are listed with the MBDS process which contains the routine. Although the results are given to four decimal places, we only trust the accuracy to the second decimal place (see Section 3.4). Basically, what can we observe about the collected message-handling times? We see that the controller processes, i.e., Request Preparation and Post Processing, spend very little time in processing the retrieval request. This is a major design goal of MBDS and is necessary to prevent a bottleneck at the controller when the number of backends increases substantially. It appears that this goal is met successfully. We also observe that the results obtained from Concurrency Control are consistent and of short duration. This is expected since there is only one request in the system at a time and no access contention can occur. These tables should then be considered as containing the best-case times. The majority of work done in the backend is at Record Processing. Observing the process timings in Record Processing, we see that, for both requests, the addition of an extra backend reduces the record processing time by nearly half.

5. CONCLUSIONS AND FUTURE WORK

We have shown that the two basic performance claims of the multi-backend database system are valid. While these results are preliminary, they are encouraging. Overall, the response-time improvement ranged from 36.07 percent to 48.94 percent, when the number of backends and their disks is doubled for

MBDS Process	Message Handling Routine	Request Number	One Backend 1K Records	Two Backends 1K Records
Request Preparation	Record Count To Post Proc	1 2	0.0005 0.0000	0.0015 0.0000
	Parse Traffic Unit	1 2	0.0200 0.0180	0.0190 0.0185
	Broadcast Results	1 2	0.0025 0.0065	0.0025 0.0030
Post Processing	Collect Results	1 2	0.0465 0.0890	0.0250 0.0813
Directory Management	Parsed Traffic Unit	1 2	0.0699 0.0925	0.0450 0.0491
	Did Sets Locked	1 2	0.0516 0.0566	0.0510 0.0566
	Cid Sets Locked	1 2	0.0533 0.0450	0.0349 0.0433
	Descriptor Ids	1 2	na na	0.0391 0.0558
Concurrency Control	Cids for Traffic Unit	1 2	0.0424 0.0425	0.0433 0.0433
	Did Sets Traffic Unit	1 2	0.0566 0.0508	0.0408 0.0516
	Did Sets Released	1 2	0.0025 0.0008	0.0016 0.0008
Record Processing	Entire Process	1 2	2.6462 12.7100	1.3775 6.5716
	Request with Disk Address	1 2	0.0466 0.0433	0.0433 0.0383
	Old Request	1 2	0.0130 0.0131	0.0148 0.0168
	PIO Read	1 2	0.0844 0.8593	0.0865 0.8863
	Disk Input/Output	1 2	0.0799 0.0783	0.0741 0.0725

Table 8. Message Handling Routine Processing Times for a Retrieval Request

the same database. The low end of the scale represented a request which involved the actual retrieval of only two records. The high end represents a request which has to access all of the database information. The response-time reductions were also impressive, ranging from a 4.49 percent change to a 0.12 change. In other words, when we double the number of backends and their disks, the response time of a request is nearly invariant despite the fact that the response set for the request is doubled. Another crucial discovery that we made was that the results were consistent and reproducible. The tests were conducted at least twice for most of the request set, with the testing done on different days by different people. The resulting data was consistent and reproducible. The data presented in this paper represents the last set of tests for the request set.

The next logical step in the performance of the multi-backend database system is to extend the testing to include the other request types, update, insert and delete. Additionally, there are still some more tests to run on the retrieval request. We should also investigate the effect of the directory structure on performance. In particular, we should try to determine how much of an effect the descriptor definitions for a directory attribute have on the performance data. Finally, we should conduct some tests on the secondary-memory version of the directory tables to evaluate just how much an effect this version will have on the performance data.

Because MBDS is intended for microprocessor-based backends, winchester-type disks and an Ethernet-like broadcast bus, we will not continue our benchmark work on the present VAX-PDPs configuration. Instead, we plan to download MBDS to either MicroVaxs or Sun Workstations. With either choice, we can utilize a broadcast bus, which was not available when the project began. We may also eliminate all the physical and virtual memory problems. In the new environment we can perhaps obtain a more accurate measurement of the internal performance measurement software overhead, conduct a more thorough benchmarking of MBDS, and study various benchmarking strategies.

REFERENCES

[Boyn83a] Boyne, R., et al., "A Message-Oriented Implementation of a Multi-Backend Database System (MDBS)," in Database Machines, Leillick and Missikoff (eds), Springer-Verlag, 1983.

[Boyn83b] Boyne, R., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management," Technical Report, NPS-52-83-003, Naval Postgraduate School, Monterey, California, March 1983.

[Demu84] Demurjian, S. A., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part IV - The Revised Concurrency Control and Directory Management Processes and the Revised Definitions of Inter-Process and Inter-Computer Messages" Technical Report, NPS-52-84-005, Naval Postgraduate School, Monterey, California, March 1984.

[He82] He, X., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part II - The First Prototype MDBS and the Software Engineering Experience," Technical Report, NPS-52-82-008, Naval Postgraduate School, Monterey, California, July 1982.

[Hsia81a] Hsiao, D.K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

[Hsia81b] Hsiao, D.K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

[Kerr82] Kerr, D.S., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MDBS," Technical Report, OSU-CISRC-TR-82-1, The Ohio State University, Columbus, Ohio, January 1982.

[Schi84] Schill, J., "Comparative DBMS Performance Test Report," Naval Ocean System Center, San Diego, CA, August 1984.

[Stra84] Strawser, P. R., "A Methodology for Benchmarking Relational Database Machines," Ph. D. Dissertation, The Ohio State University, 1984.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52ML Computer Science Department Naval Postgraduate School Monterey, CA 93943	10
Prof. David K. Hsiao, Code 52Hq Computer Science Department Naval Postgraduate School Monterey, CA 93943	130
Chief of Naval Research Arlington, VA 22217	1