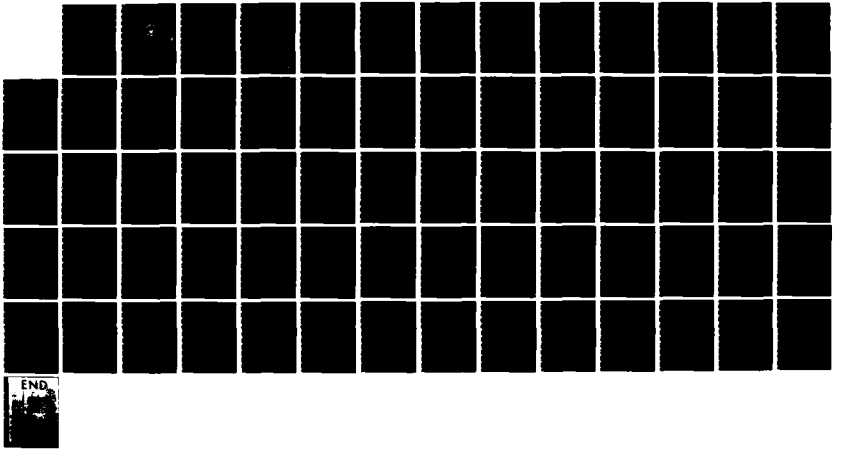
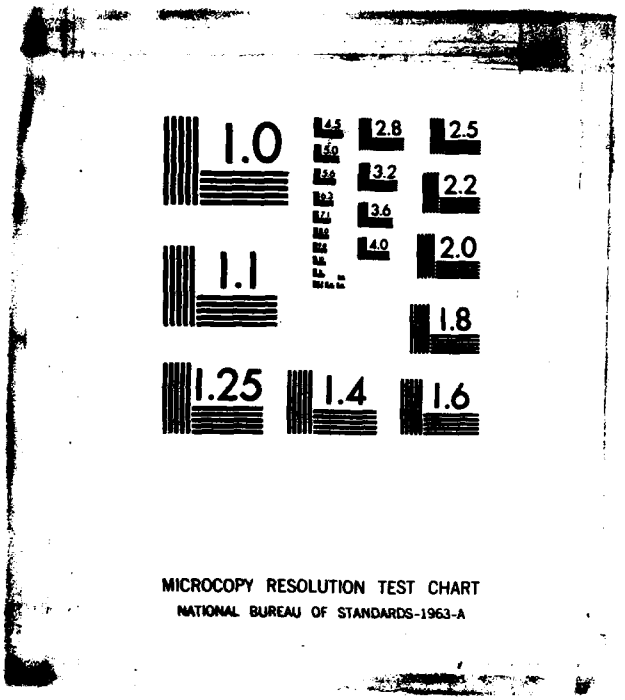
AD-A136 776 BENCHMARKING THE SELECTION AND PROJECTION OPERATIONS AND ORDERING CAPABILITIES OF RELATIONAL DATABASE MACHINES(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA 1/1

UNCLASSIFIED R A BOGDANOWICZ SEP 83 F/G 9/2 NL

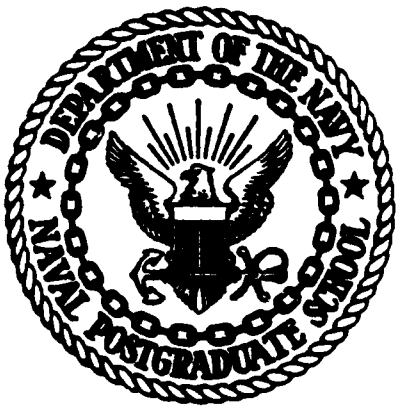


END



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

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

H

BENCHMARKING THE SELECTION AND PROJECTION OPERATIONS, AND ORDERING CAPABILITIES OF RELATIONAL DATABASE MACHINES

by

Robert A. Bogdanowicz

September 1983

Thesis Advisor: David K. Hsiao

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A136776	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Benchmarking the Selection and Projection Operations, and Ordering Capabilities of Relational Database Machines		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis September 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert A. Bogdanowicz		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1983
		13. NUMBER OF PAGES 66
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

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

18. SUPPLEMENTARY NOTES

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

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

benchmarking, relational database machines, database machines

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

This thesis describes the performance-measurement experiments designed for a number of back-end, relational database machine configurations. An in-depth study of the tests and results of the two relational operations, namely, selection and projection, on a specific configuration is presented. In addition, tests are made on the ordering capabilities and performance of the machine configuration. The goal of the work is to lead to a development for a machine-independent methodology for benchmarking the selection and projection operations and on ordering capabilities of database machines.

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)

Approved for public release; distribution unlimited.

Benchmarking the Selection and Projection Operations, and Ordering Capabilities of Relational Database Machines

by

Robert A. Bogdanowicz
Lieutenant, United States Navy
B.S., Illinois Institute of Technology, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1983

Author: Robert A. Bogdanowicz

Approved by: David K. Hsiao
Thesis Advisor

Paula R. Strawser
Second Reader

David K. Hsiao
Chairman, Department of Computer Science

K. T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This thesis describes the performance-measurement experiments designed for a number of backend, relational database machine configurations. An in-depth study of the tests and results of the two relational operations, namely, selection and projection, on a specific configuration is presented. In addition,tasts are made on the ordering capabilities and performance of the machine configuration. The goal of the work is to lead to a development for a machine-independent methodology for benchmarking the selection and projection operations and on ordering capabilities of database machines.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

The experiments described in this thesis are the results of the coordinated efforts of many individuals. Foremost of these individuals is certainly Ms. Paula Strawser of The Ohio State University and the Naval Postgraduate School. Ms. Strawser's diligence, dedication, and guidance have proven invaluable in both the research prior to and subsequent preparation of this thesis.

Likewise, Ms. Doris Mleczko and her staff at the Data Processing Service Center West at Point Mugu, California, have provided much assistance, and they have proven flexible enough to accomodate our sometimes inflexible requirements. Gratitude is also expressed to Mr. Ben Torres and his staff at the Computer Operations Center at Point Mugu.

Finally, to Commander Tom Pigoski of the Naval Security Group Command is offered a special thanks for his continued support in providing the necessary assistance both to keep this project going and to enable the results of this research to be presented at the International Workshop on Database Machines in Munich, September, 1983.

I. INTRODUCTION

A. BENCHMARKING DATABASE MACHINES

Benchmarks have long been a means for making effective comparisons of differing hardware configurations and hardware architectures. As early as 1970 instruction mixes were formed and tested over varying configurations to provide a means of comparison between installations. The early works included the Gibson [Ref. 1], and Flynn [Ref. 2], mixes which consisted of machine instructions ordered by instruction class. The Gibson mix was based on data collected from IBM 7090 installations, while the Flynn mix used programs run at IBM System/360 installations. There has been some work done with similar approaches at the language level, predominantly the work of Knuth [Ref. 3], who used a mix of Fortran statements to obtain his benchmark parameters. All these approaches involved the running of some standardized mix of instructions, either machine instructions or instructions in some high-level language. They used the experimental results from these runs to conduct an analysis of the computer system performance.

1. A Definition

Benchmarking is a term used throughout the industry in a myriad of differing contexts. In each case the ultimate goal is to make an independent measure or relevent comparison of machine capabilities. These comparisons or measures could be anything from the throughput to the speed of calculations by a certain internal component, but in the final analysis some measure or evaluation of performance is desired.

There are many different ways of evaluating machine performance. Many manufacturers provide the capability of attaching monitoring systems to their equipment. These may be either hardware monitors, which physically sense the action occuring in the system and keep statistical records, or they may be software monitors which attempt to perform the same function with software hooks that keep track of the system operation and give the operator a statistical analysis of the machine action and performance. Software monitors have the disadvantage of using a good deal of the system time just for their own operation. Though hardware monitors do not suffer from this disadvantage, they require the wiring of the monitor system into the hardware. The biggest disadvantage to these types of measurements, however, is the inability to make comparisons on differing machine configurations and between different manufacturers. Benchmarks attempt to solve this problem by forming some standardized testing methodology that is easily transportable from one machine to another machine. Most importantly, the measurements made must be relevent regardless of the machines benchmarked and give an accurate means of comparison between these machines.

Therefore, benchmarks are defined to be certain sets of instructions that will test all the capabilities of a machine and yield some generic set of data that will give an accurate measure of that machine in its tested configuration. This data will then give the observer specific guidelines for making relevent and general comparisons with similar machines and configurations.

2. Database Machine Benchmarks

With the advent of special-purpose database machines and backend database machines, a new field of application for benchmarks exists. Previously, benchmark routines have

been used exclusively for the testing and performance evaluation of large general-purpose mainframes. With the proliferation of backend processors to unload specialized tasks from the mainframe, these benchmarks have been ineffective, because the computer system's capabilities of performing the specialized tasks are not benchmarked. Our primary concern is with the benchmarking of specialized backends known as database machines. In this context we mean a specialized processor externally linked to a mainframe, with its own special-purpose hardware and software for database management. Backend refers to this externally-linked and specially-built machine.

3. Objectives

At present the backend database machine is in its infancy in the commercial marketplace. Nevertheless, the database system is extensively utilized in various forms and for different tasks, exclusively in some software configuration operating on a large general-purpose machine. In order to provide effective database functions the software-laden database system consumes a great deal of the mainframe's resources which severly limits the usefulness of the mainframe for other functions.

This has started a trend towards the backend database machine, one that can reduce the time the host spends in searching and updating data in response to user queries. This greatly increases the ultimate usefulness of the host, since these backend database machines are only a small fraction of the total system cost. The database machines now on the market have been implemented using microprocessor technology rather than fully-specialized hardware, thereby keeping their costs down. As the market expands and more progress is made in VLSI technology, we can expect to see more specialized hardware at even lower cost.

Our objective here is to develop some basic testing procedures to benchmark relational database machines. This thesis also gives account of test results performed on a specific backend database machine, the RDM-1100, and its various configurations. It is limited to the results of test queries in the operations of selection and projection and ordering capabilities. In addition to this thesis, there are three other theses, [Refs. 4,5,6], which describe in detail the test procedures and results of join operations, the generation of the databases used in the experiments, and the other test procedures and results. The ultimate goal of the entire project is to develop and identify some sets of queries that can be used in evaluating database machine performance.

B. THE BENCHMARKING ENVIRONMENT

Our primary emphasis is to evaluate the performance of the system/machine under typical operating conditions. In this sense a standardized workload model must be developed. This includes the use of typical user queries (transactions) in addition to the design of a database. In terms of the database, we developed a paramaterized database generator that will generate our databases with attributes according to a specified format and with values from well-defined domains according to specific distributions. We chose this approach so that we could predict or interpret accurately the results of any given query. More details are given on the context and design of the database in Chapter II.

Query streams are developed to test the full range of possible user operations. All queries are in forms of selection, projection, or join operations as may be made by a typical user. The actual query syntax and selection of query streams is discussed further in Chapter III.

In addition, the environment available to us for the test runs is very restricted. There are no hardware or software probes available at the time of testing, nor any statistical information on the backend machine. Our only recourse is to use a built-in retrieve function that will give a readout of the database machine clock. Unfortunately, the clock has a low resolution, 1/60 of a second. A system call is executed to retrieve the time before and after each test query, thereby providing a crude yet consistent time measure.

1. The Host

The actual testing is done using a UNIVAC 1100/42 host system. The system is located at the Pacific Missile Test Center, Point Mugu, California. The basic database machine used is the RDM-1100, which is a Britton-Lee IDM-500 modified to run as a backend to UNIVAC 1100 computers by the Amperif Corp. of Chatsworth, California.

The testing is done using run-stream queries in an interactive environment. These queries are run either on site at Pt. Mugu, or from a remote terminal set up at the Naval Postgraduate School, Monterey. We prefer to run the test queries in a stand-alone, single-user mode in order to minimize the effects of workload variability of the host machine. In the event that the queries are not run stand-alone, the number of coincidental users is very low and little or no difference is observed in the measurement from one run to another.

2. The Host Interface

The interface between Univac and the RDM is via a word channel; the RDM is treated as an I/O device by the UNIVAC mainframe. The standard IDM device is capable of communicating over an RS-232 serial interface or an IEEE-488

parallel interface. The communication board of the IDM at Pt. Mugu has been modified to be compatible with the Univac system. It supports byte/word channel interface with a 200K byte/second capacity.

The driver routines on the Univac host handle the parsing of the user queries, and translate them into the IDM internal format. The host also handles the communication protocol with the backend machine. The backend, in addition to performing the necessary handshakes, will perform the required error checks and cause the host to retransmit in the event that an error is detected.

3. Machine Configurations

The IDM-500 system comes with different amounts of internal cache memory, and has an optional accelerator board. The accelerator is a high-speed processor designed to perform certain common relational functions in order to increase the overall system performance. The machine can be configured to hold 1-6 megabytes of information. We have run tests on the following configurations:

(1) 1/2-megabyte cache without accelerator;

(2) 2-megabyte cache with accelerator;

(3) 2-megabyte cache without accelerator.

The first of these configurations is no longer marketed. The standard package contains 1-megabyte of cache memory and no accelerator. In addition, the machine used in our tests is linked exclusively to the Univac 1100, and is equipped with only one disk controller, with access to two 600-megabyte disks.

C. THE BENCHMARKED MACHINE

We chose to restrict our work to the IDM-500, a relational database machine. This type of machine is relatively new on the database market. Although it is not clear that it will be the predominant database machine architecture, the latest literature and current trends appear to indicate that it may play an important role, at least in the short run.

The relational model is intuitively easier to use and understand than other database models, and it appears that it will significantly contribute to lower software development costs. Nevertheless, fully-implemented software relational database management systems have severe performance problems. The high cost of performing relational operations, most strikingly the join and projection operations, underlies the problem.

With the great interest in the relational database models and the advances in technology that permit the use of special-purpose processors and backend systems to perform the majority of work, we feel that the relational database machine will play an important role in the database management market. The Britton-Lee IDM-500 is one of the first machines to take advantage of this technology and incorporate it into a relational database system which can be used as a backend to a variety of mainframes.

1. Modular Design

The Britton-Lee IDM-500 is a backend relational database machine that can be linked to one or more host computers. Amperif Corp. markets this system under an OEM agreement as the RDM-1100. Essentially, the system is a Britton-Lee IDM-500 with Amperif providing the host and backend interface software to communicate with the Univac

1100 and a host-interface module. Figure 1.1 depicts the architecture of the Britton-Lee machine. From now on we will use IDM-500 and RDM-1100 interchangeably.

The backend is a modular, expandable, microprocessor-based system organized around a central high speed bus. Each module is functionally oriented.

2. Technology and Functionality of Modules

The RDM-1100 is made up of six basic modules organized on a central high speed bus (see Figure 1.1 again). The modules perform the following functions:

a. The database processor

The database processor, a Z8000-based microprocessor, supervises and manages all system resources. This processor executes most of the software in the system.

b. The database accelerator

The database accelerator (an optional processor) is a high-speed processor with an instruction set specifically designed to perform and optimize certain functions. It is activated by the database processor as appropriate. The accelerator has a three-stage pipeline which executes instructions at up to 10 MIPS. This processor can initiate disk activity and process data at disk transfer rates. The accelerator and the RDM software are so configured that the majority of database work is performed by the accelerator under the direction of the database processor.

c. The main memory

The RDM main memory, or cache memory, is composed of 64k-bit dynamic RAM chips. The RDM can be configured with from 1-megabyte to 6-megabytes of memory. This memory is utilized for RDM system code, disk buffering, indices, and user commands.

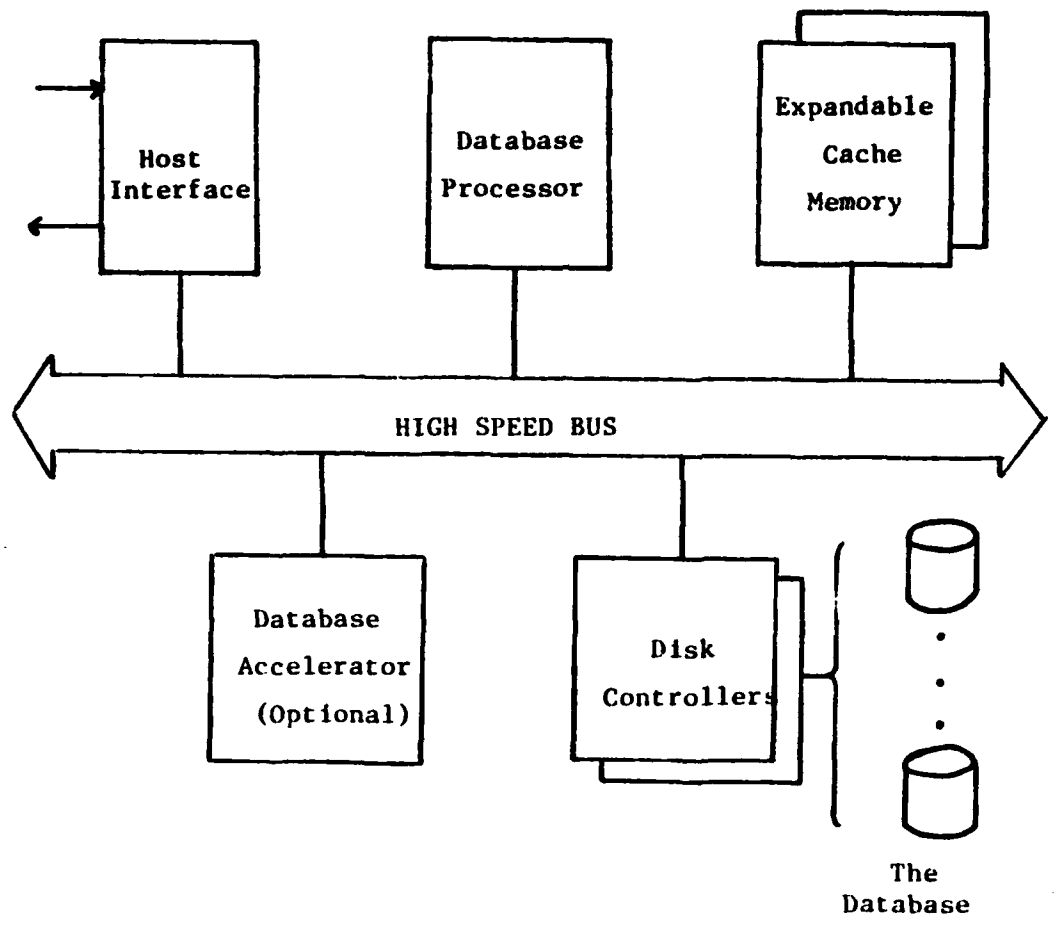


Figure 1.1 The RDM-1100 Relational Database Computer.

d. The internal bus

The entire system uses a common internal bus system for inter-processor communication and data transfer.

e. The disk/tape interfaces

The system can be configured with up to 4 disk controller modules. Each controller can manage from one to four disk drives. The disk controller moves data between external disks and the RDM main memory. The disk controller is designed to work with the accelerator which can process data at disk transfer rates. An optional tape control module supports up to eight tape drives, which can be used for direct disk-to-tape backup, data loading, and RDM software loading.

f. The host interface

The RDM and the host(s) communicate via the host interface module. This module accepts commands from one or more hosts, performs error checking, causes the host to retransmit if an error is detected, and informs the database processor that it is moving a command into the cache. Each host interface module can handle up to eight hosts. Hence, with the full 8 interface modules, a maximum of 64 hosts can be accomodated by the RDM. The standard interface module supports both RS-232 serial interface or an IEEE-488 parallel interface.

II. THE DATABASE

In our benchmark measures on the RDM-1100, it is important to model the queries or transactions to be processed, and to model the database. The performance of any database system depends not only on the characteristics of the database system, but also on the size and structure of the database. Considering this two-dimensional problem, we want to build databases where the values for each attribute may be selected from well-defined domains. In addition, we feel that these values should have specified and well-formed distributions to aid in the prediction of the response set for any given query.

We have built a parameterized relation generator, a software system to generate relations for synthetic databases. These synthetic databases are then used by our query stream to simulate the activity of actual users on the sytem. Several of these databases are built, varying the tuple widths as well as the number of tuples per relation. We then attempt to distribute the databases on the disks to force specific actions on the processor, such as join operations between relations on the same or seperate disks. In this manner we seek to find any significant difference due to the distribution and location of the data on disks.

A. THE USE OF SYNTHETIC DATA

As with any system model, it is important that the synthetic data adequately represent the essential characteristics of real databases. By utilizing the synthetic database, we can represent a subset of the real-world database and save time and space for not accommodating the full

set of the real-world database. However, the organization is general enough to provide an emulation of the real world. The synthetic databases we have designed includ the basic data types that would exist in a real-world database: integer, character, and so on. For attribute values we have incorporated both sequential and random orders, as well as groupings according to specific discrete distributions. These are more fully described in the next section. Using this format we can not only accurately predict the outcome, i.e. amounts of data returned by a query, but we can also easily reproduce the databases on other systems for further tests.

B. GENERATION OF THE SYNTHESIZED DATA

When designing the database, our first concern is with the physical sizes that should be used. The relations must be large enough to test the full capacity of the system, and meaningful enough to include various attributes. For example, we choose tuple widths of 100, 200, 1000, and 2000 bytes with the maximum tuple width being limited at 2000 bytes and the disk access being performed in 2k blocks.

Our second consideration is how large the relations should be, i.e., how many tuples per relation. Again, in order to test the system for both large and small relations, we decide on relations with 500, 1000, 2500, 5000, or 10000 tuples. These are arbitrary decisions. The relation sizes are multiples of the smallest number in order to facilitate comparisons of the test results.

Our next consideration is the actual design and building of the data generation tool. We envision a great many databases with differing configurations. Thus, an interactive interface to a generation program appears to be the most effective approach. Using the locally available IBM 3033

VM/CMS installation and PASCAL/VS as the language, an interactive system is built. For more information on the design, programming, and operation of this tool, please see [Ref. 6].

Using the interactive system, the user is allowed to define the format of a relation in response to sytem prompts, on an attribute-by-attribute basis. The tuple width and relation size are defined. The user is then allowed to 'add' attributes to the tuples one after another until he reaches the desired limit.

The user can choose from several methods of attribute value generation. Integer values can be sequential or random within a specified domain. Uniqueness of the random integer can be assured. The integer can be either one, two, or four bytes. Character-strings can also be chosen, either compressed or uncompressed, in a collating sequence or in some random order. Character string values can also be selected from enumerated domains either randomly or according to a specific discrete distribution. In our prototype the discrete distributions are limited to multiples of 5%. The user is also given the opportunity to set the naming convention for each relation and its attributes. The prototype is designed and implemented with a limited set of alternatives. It is however modular for adding alternatives to the prototype, such as exponential or normal distributions.

We use a standard template for each tuple width. A portion of this template is standard for each relation (see Figure 2.1). Each relation contains: a sequential-integer attribute, a 4-byte-integer,'key'; a character-attribute 'mirror', which is identical in numerical value to 'key' but stored as a character string and not as an integer; a random-integer-attribute 'rand' of 4-byte integers; and a character-string-attribute 'chars', which contains

100 BYTES	
FIELD	TYPE
KEY	I4
MIRROR	C11
RAND	I4
UNIQRAND	I4
CHARS	C4
LETTER	C1
P5	C9
P10	C9
P20	C9
P25	C9
P35	C9
P50	C9
P75	C9
P80	C9

200 BYTES	
FIELD	TYPE
KEY	I4
MIRROR	C11
RAND	I4
UNIQRAND	I4
CHARS	C14
LETTER	C1
P5	C9
P10	C9
P20	C9
P25	C9
P30	C9
P35	C9
P40	C9
P45	C9
P50	C9
P55	C9
P60	C9
P65	C9
P70	C9
P75	C9
P80	C9
P85	C9
P90	C9
P100	C9

1000 BYTES	
FIELD	TYPE
KEY	I4
MIRROR	C11
RAND	I4
CHARS	C63
P5	C9
P10	C9
P20	C9
P25	C9
P30	C9
P35	C9
P40	C9
P45	C9
P50	C9
P60	C9
P65	C9
P70	C9
P75	C9
P80	C9
P85	C9
P90	C9
P100	C9
UP10	UC255
UP25	UC255
UP50	UC255

2000 BYTES	
FIELD	TYPE
KEY	I4
MIRROR	C11
RAND	I4
CHARS	C79
P5	C9
P10	C9
P20	C9
P25	C9
P30	C9
P40	C9
P50	C9
P60	C9
P70	C9
P75	C9
P80	C9
P90	C9
P100	C9
UP10	UC255
UP20	UC255
UP25	UC255
UP50	UC255
UP75	UC255
UP80	UC255
UP100	UC255

FIELD TYPES

C- COMPRESSED CHARACTER STRING
(MAXIMUM OF 255 CHARACTERS)

UC - UNCOMPRESSED CHARACTER STRING
(MAXIMUM OF 255 CHARACTERS)

I4 - FOUR-BYTE INTEGER
THIS FIELD MAY CONTAIN ANY INTEGER VALUE BETWEEN
-2,147,483,648 AND +2,147,483,647

Figure 2.1 Tuple Templates.

characters in a collating sequence. The number of characters in 'chars' is dependent on the tuple width, in order to ensure that tuples are exactly 100, 200, 1000, and 2000 bytes wide. The length of 'chars' is set to the precise number of characters required to ensure that the tuple is of the proper width. The random field is present to aid in randomizing the order of the tuples and the purpose of the mirror field is to compare the performance of identical retrieve operations based on queries qualified on the sequential-integer-attribute,'key', and the character-attribute, 'mirror'. The 100-byte and 200-byte tuples also contain a sequential-unit-letter field of 1-byte character in collating sequence, 'letter', and a unique random-integer-attribute of 4-byte integers, 'uniqrand'.

Each template is then filled out with attributes for which the values are chosen from a number of enumerated values. For example, the P10 attribute specifies attribute values with a uniform distribution over ten unique values. A retrieve statement with one qualifier could then be written to retrieve 10% of the tuples in the relation. The number of such fields is dependent on the tuple width.

Once the design of the databases is complete, multiple instances of each relation are built using the interactive generation tool on the IBM 3033. The relations are then transferred to tape storage for transport to Pt. Mugu and the UNIVAC 1100. The data is loaded onto the UNIVAC 1100 disks and then loaded to the backend database machine using a bulk-load utility.

Tests are planned on the basis of an assumed capability to control the distribution of the data on the RDM 1100 disks. The capability to direct a relation to a specific disk is not implemented, although the space allocation for a database can be split across multiple disks. The pattern of block allocation for relations within the database is controlled within the database machine, and is not predictable.

III. THE QUERY LANGUAGE

The interaction between the user and the RDM-1100 is through the software interface, RQL (relational query language), provided by Amperif. The interface translates the user's RQL command into the backend-machine's internal format and sends the formatted command to the RDM-1100. The software requirement for the host is minimal, and the backend machine is independent of the host.

When performing the test runs, the test queries are grouped into run-streams in order to make more efficient use of the available time. The time provided for our test runs has been very restricted. Since we prefer to make our test runs in a stand-alone, single user environment to minimize the host workload variability, we are forced to execute our run streams during the evenings and on weekends. In addition we want to run sets of tests over several system configurations. This again reduces the overall time for us to run our performance tests on each configuration.

Additional constraints are imposed by the nature of the interface software provided by Amperif and by the configuration of the machine at Pt. Mugu. Pre-compilation of the queries is not supported. We therefore have chosen to use the stored-commands facility of the backend machine to reduce varability in the parsing time. The stored-commands facility allows the user to store the parse-trees produced by the interpreter as named commands in a relation in the user's database. When these stored commands are invoked at a later time, the parsing is reduced to a minimum. Using the stored-command facility also eliminates the time required to look up target-list and qualification attributes in the data dictionary.

A. SYNTAX AND SEMANTICS

The basic operations involved in retrieving data in a relational system are selection, projection and join. This section will provide a basic overview of the syntax of the Relational Query Language (RQL), with pertinent examples. For a more detailed explanation of the language as well as the database administrator functions, please refer to [Ref. 5]. This thesis focuses exclusively on the selection and projection operations. The interested reader is encouraged to read [Ref. 4], for an explanation and evaluation of the join operations as performed on the RDM-1100 and its various configurations.

Simple selection in RQL is expressed as follows:

```
RETRIEVE ( A.ALL ) WHERE A.CITY = "CHICAGO"
```

The keyword to the selection operation is RETRIEVE. The relation referred to in this case is A and the qualifier ALL indicates that all attribute values, i.e. the entire tuple, are to be returned for each qualifying tuple. In this example an optional qualifier consisting of a single predicate has been added, WHERE A.CITY = "CHICAGO". This qualifier restricts the tuples returned to only those tuples in which the city attribute has a value of "CHICAGO". The qualifier could have multiple predicates, related by any of the boolean operators, such as AND, OR, = (EQUAL), != (NOT EQUAL), etc. An example is:

```
RETRIEVE (A.ALL) WHERE A.CITY="CHICAGO" OR A.CITY="MONTEREY"
```

In this case the backend machine will return all the tuples in the relation A in which the city attribute has either the value "CHICAGO" or the value "MONTEREY".

The selection operation restricts the tuples to be returned. The projection operation restricts the attribute values of a tuple; only a portion of the attribute values of each tuple are returned. For example:

```
RETRIEVE (A.CITY,A.NAME)
```

In this case, the target list (A.CITY,A.NAME), specifies the attribute values to be projected out of the tuple and returned to the user. Only the values of attributes CITY and NAME for each of the tuples in the relation A will be returned. A qualifier (not shown) could be added as in a previous example to limit the number of tuples returned to a specific subset of the relation.

Commands like these make up the bulk of the queries used in the selection and projection tests, with varying qualifiers attached. RQL has many more capabilities, such as the aggregate functions and the BY clause. For further details, again refer to [Ref. 5],

B. TEST QUERIES

The test queries used are all selection and projection operations in the form of the previous two examples. Qualifications are used on these queries to select given percentages of the attribute values, as well as given percentages of the tuples in each relation. As described in Chapter II, single qualifiers are used on the attribute values having discrete distributions to select only a given percentage of each relation. Comparisons are made on the

backend database machine's performance as the percentage of data retrieved is varied. This variation covers two dimensions: the percentage of tuples in a relation and the percentage of attribute values in a tuple. Additional testing is done on single-tuple retrieves and queries using range predicates on the key field. Each of these experiments is described in further detail in Chapers IV and V along with a detailed description of the commands used to retrieve the data.

1. Timing Considerations

As mentioned before, the most critical restriction placed on the performance tests is the lack of measurement tools. There are no monitors available to keep track of CPU or I/O activities in the backend database machine. The only available measurement capability is a measurement of elapsed time that could be extracted from the backend database machine clock, which has a resolution of 1/60th of a second. Our prime concern in this performance evaluation is to determine the effects of varying certain parameters on a backend database machine and gather some gross overall measures. In this sense, therefore, we feel that the rough measurements afforded by the backend machine are still acceptable for our purpose.

In order to determine the elapsed time in processing a query, a retrieve command to extract the time from the backend database machine clock is executed before and after each query. The retrieve command is of the form:

```
RETRIEVE ( TIME = GETTIME()) GO
```

GETTIME is a system function of the backend machine. This command is used to print a time, in 1/60 second increments,

before and after our queries. Using this throughout our experiments we can get gross, yet consistent measurements of total time required to execute the queries. Even with this poor resolution, the comparison of identical queries will yield relevent performance comparisons of the response time of the backend machine.

2. Objectives

The final objective of these tests is not to generate large volumes of data with figures of retrieval times for particular queries. Our primary goal is to make relevent comparisons of the machine performance as the queries are varied inside specific parameters. To this end we hope to make some judgements of the overall performance of this particular backend database machine, but more importantly to gain some insight into the testing methodology for backend database machines in general. In the next chapters, examples of the run-streams used in the experiments are given along with graphical representations of the test results.

IV. PERFORMANCE EVALUATION OF THE SELECTION OPERATION

A. DEFINITION OF A SELECTION

Selection is a means for the user to retrieve and examine pertinent information from a relation. The user may select the entire relation or he may restrict the information returned to him in two ways. He may limit the number of tuples returned by adding a qualification to the selection operation. The qualification will limit the tuples retrieved to those whose attribute values satisfy the conditions of the qualification. Qualification consists of predicates, assertions on the attribute values of the tuple or tuples. Multiple predicates may be combined with boolean operators, such as AND, OR, EQUAL, NOT EQUAL, etc. The user may also restrict the attribute values returned by explicitly listing those attributes which he desires, a projection of the relation. This is further described in the following sections of this chapter.

B. SELECTIONS IN THE QUERY LANGUAGE

In RQL the user is given considerable power of selection through use of the RETRIEVE command. Using the 100-byte relation described in Table 2.1 as a format for a relation A, a typical RQL selection command might be:

```
RETRIEVE ( A.ALL ) WHERE A.KEY = 25
```

In this command the keyword RETRIEVE is used to signify selection, the A.ALL indicates that all attribute values i.e., entire tuples, are to be returned, and the keyword

WHERE identifies the quantifier. The A.ALL may be replaced with an explicit listing of those attributes desired. The attributes may be listed in any order the user desires. Using the key word WHERE and a qualification, the user may then indicate which of the tuples are to be returned. In this example, only those in which the KEY field is equal to 25 are returned. The user may use other operators such as < or >, and is given the option to use more than one predicate. For example:

```
RETRIEVE ( A.ALL) WHERE A.KEY > 25 AND A.KEY < 100
```

would return all tuples with the KEY field in the range 26 through 99. The user is given great latitude in delimiting the subset of the relation he desires. For more detailed information concerning the capabilities and syntax, the reader is encouraged to read [Ref. 5].

C. AN ENVIRONMENT FOR THE MEASUREMENTS

The results discussed in this chapter are from tests performed on the system configuration with 2-megabyte cache memory and the optional accelerator. Lack of time prevented a significant number of tests on alternate configurations for comparison. However, these tests can be conducted on other configurations without modifications.

As described in Chapter III, the timing measurements are the backend system's response to a retrieve for its internal system clock time in 1/60-second resolution. In most cases the measurements are based on single queries due to the time involved, Some measurements are averages over several query responses; these are differentiated in the sections which follow. In all cases the tests are runs performed in the

evenings and weekends with virtually no other users on the sytem.

D. SELECTION MEASUREMENTS

The figures in the first section represent results gathered for selections with and without indicies. The number of tuples returned is restricted to a fixed proportion of the total number of tuples in the relation; no projection is involved. The final sections give comparisons of the system ordering capabilities on the frontend as well as the backend, and the effects of data compression.

1. The Percentage of Selection

Figures 4.1 and 4.2 show the system response time for selection. Figure 4.1 shows measurements on a database with no indicies; Figure 4.2 shows measurements on a database with a non-clustered index on the P5 and P10 attributes. As described in Chapter II, the P5 and P10 attributes are attributes whose values are in a uniform distribution over the corresponding percentage. The P5 attribute values will be 20 unique values each appearing in 5% of the tuples and the P10 values are 10 unique values each appearing in 10% of the tuples. The queries used are qualified on the P5 attribute. Therefore, for each query the system will return exactly 5% of the tuples in the relation.

As evident in Figure 4.1 the system response time increases nearly linearly as the amount of data returned increases. As expected, the larger is the tuple size; the steeper is the slope, since the volume of the data increases more rapidly for the larger tuple size.

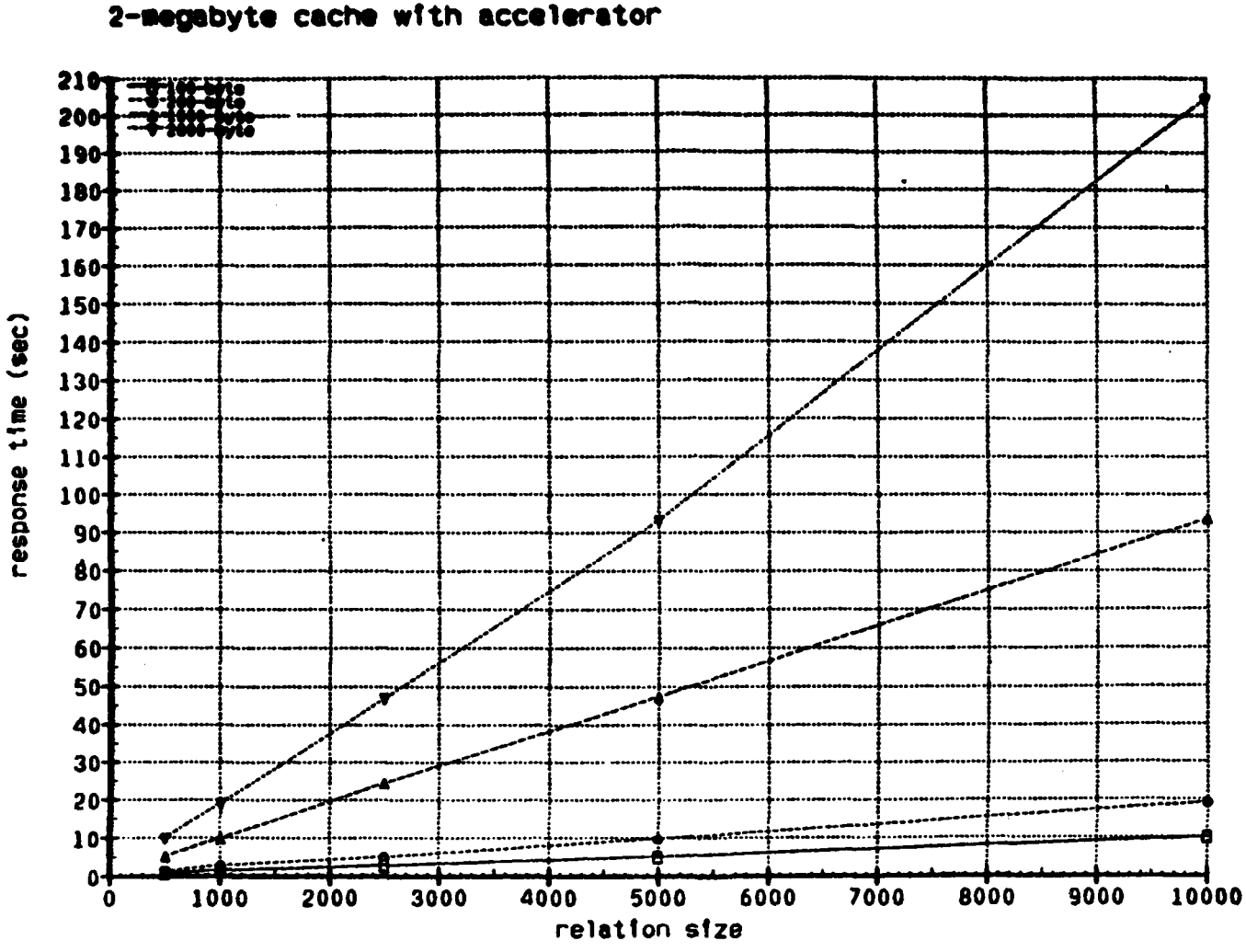


Figure 4.1 Simple Selects with no Indicies.

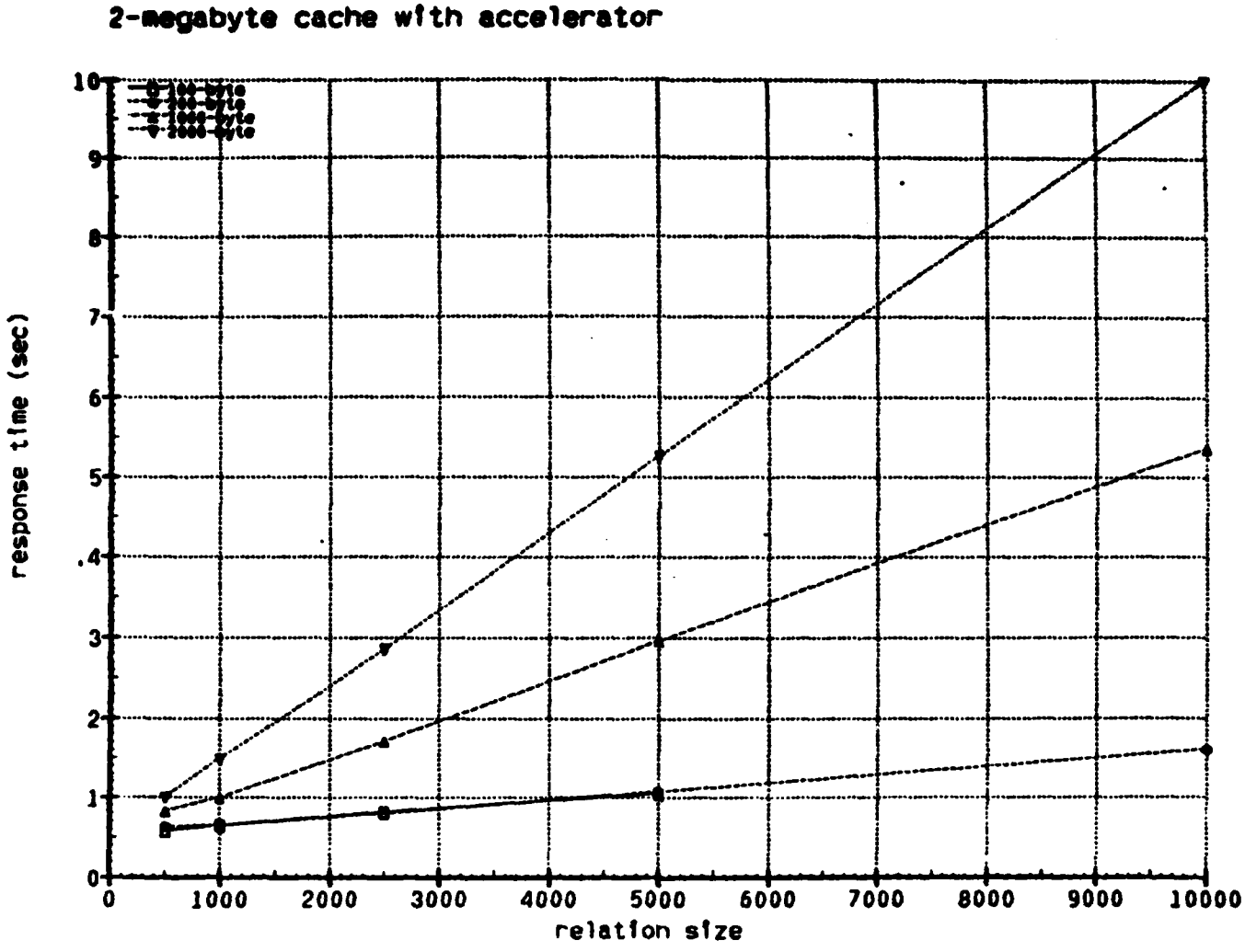


Figure 4.2 5%-Selects with Non-Clustered Index on P5 and P10.

Figure 4.2 shows the results of the same queries run against a database with indicies on the P5 and P10 attributes. Comparing Figure 4.2 and 4.1, we notice that the overall times are greatly reduced. The graph still shows nearly linear relationship of the increasing response time and of the increasing volume of data. Further discussions of the effects of indicies follow in the next section.

The linearity of the response time appears to indicate that the system performance is bound by the speed of the channel between the host and the backend. The larger the volume of data is to be returned; the longer the channel will be active in order to transfer the data.

2. Effects of Clustered and Non-Clustered Indicies

The RDM-1100 supports two types of indicies, clustered and non-clustered. Creating a clustered index causes the tuples to be ordered by KEY for storage. A sparse index containing one entry per block is built. A non-clustered index, on the other hand, contains a unique entry for each tuple in the relation. No ordering of tuples within the relation is implied.

Figure 4.3 shows response times for the retrieval query with no qualification, but with an ordering specification. The queries are of the form:

```
RETRIEVE (A.ALL) ORDER BY A.KEY
```

where A is the relation name and KEY is an attribute in A. In an ordered retrieve, the tuples are sorted in the backend machine and then sent to the host for display. Similar queries are run against a relation with no index, a relation with a non-clustered index on the KEY attribute, and a relation with a clustered index on the KEY attribute. The

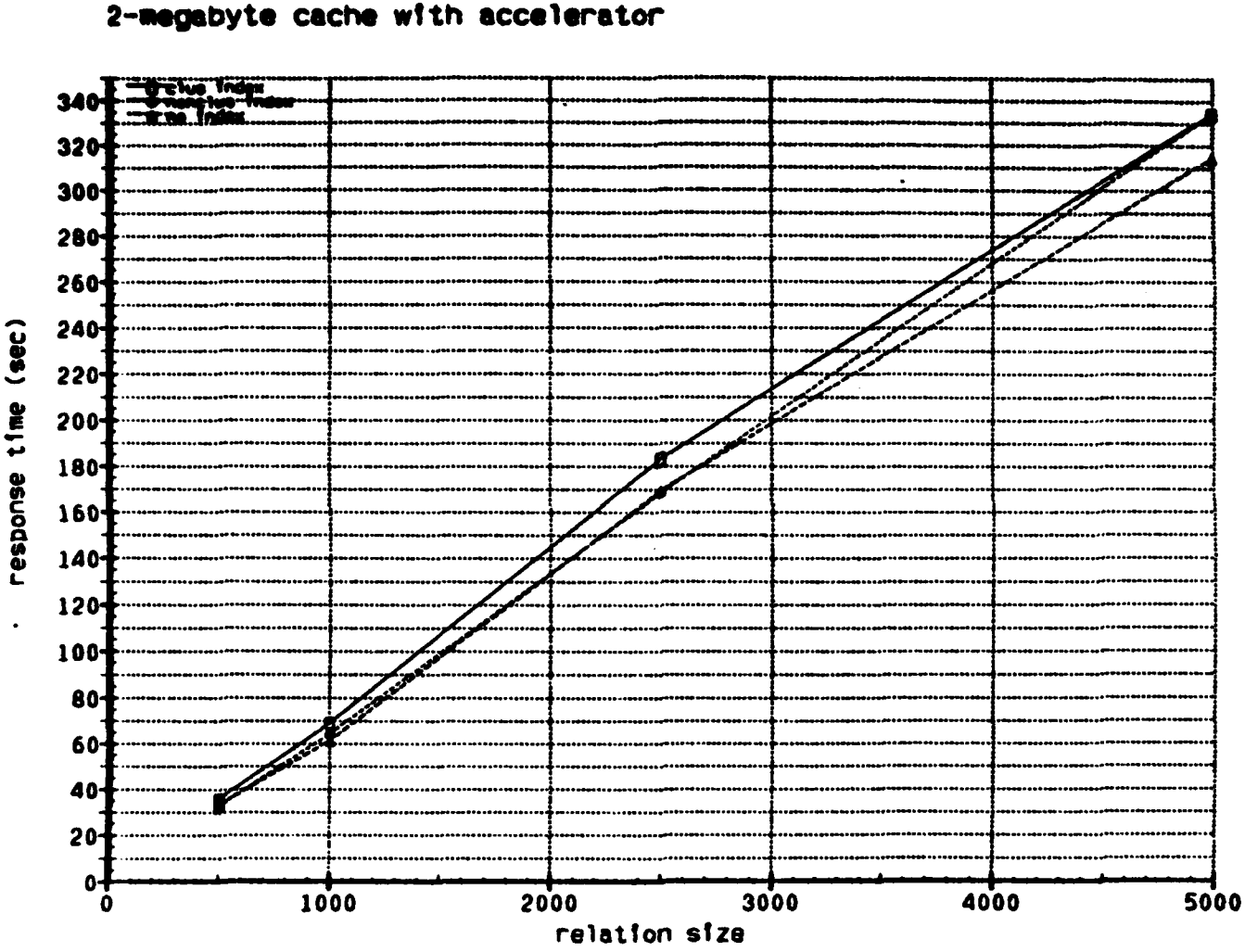


Figure 4.3 Ordered Retrieves with Indicies on KEY.

response times are similar throughout the range of relation sizes. The indicies, clustered or non-clustered, provide no significant improvement for this range of relation sizes. The expected results would have shown a significant improvement for the relation with a clustered index. The similarity in response times may indicate that the RDM sorts the the tuples, even though the tuples have been in sorted order due to the use of a clustered index on the ordering attribute.

Figure 4.4 shows the results of test runs on relations with and without non-clustered indicies on the P5 and P10 attributes. The graph shows a significant improvement in response times for the relations with the non-clustered index. Looking at Figure 4.5, the improvement ratio is made more evident for simply qualified retrieves when the index is on the attributes used in the predicates of the qualification. The larger is the tuple size; the greater becomes the improvement. The 200-byte tuple shows a nearly 95% increase in the response time. The other tuple sizes show similar improvements.

3. Effects of Data Compression on Selection Queries

The backend database machine has the capability of storing character strings in either compressed or uncompressed format. A character string in compressed format is stored on the disk with no trailing blanks. The advantage is a savings in disk space. The tradeoff is the increased CPU time required to compress and uncompress the strings as data is moved to and from the disk. Figure 4.6 shows the results of test runs on relations having only uncompressed attribute values and on relations having only compressed attribute values. In the initial test runs the relations have both compressed and uncompressed attributes as specified in Table 2.1, in order to ensure the correct byte-width of the tuple.

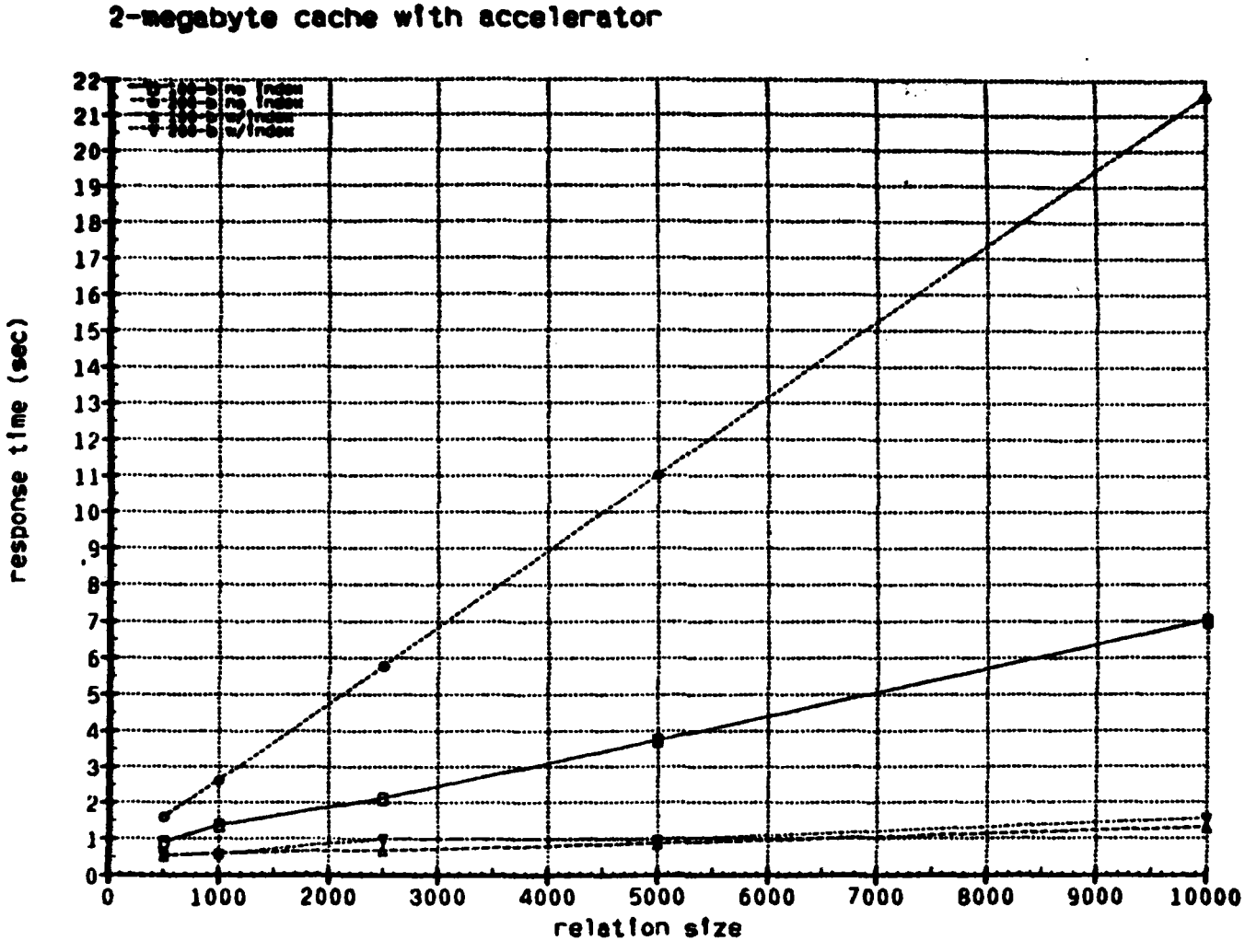


Figure 4.4 Retrieves with and without Indicies.

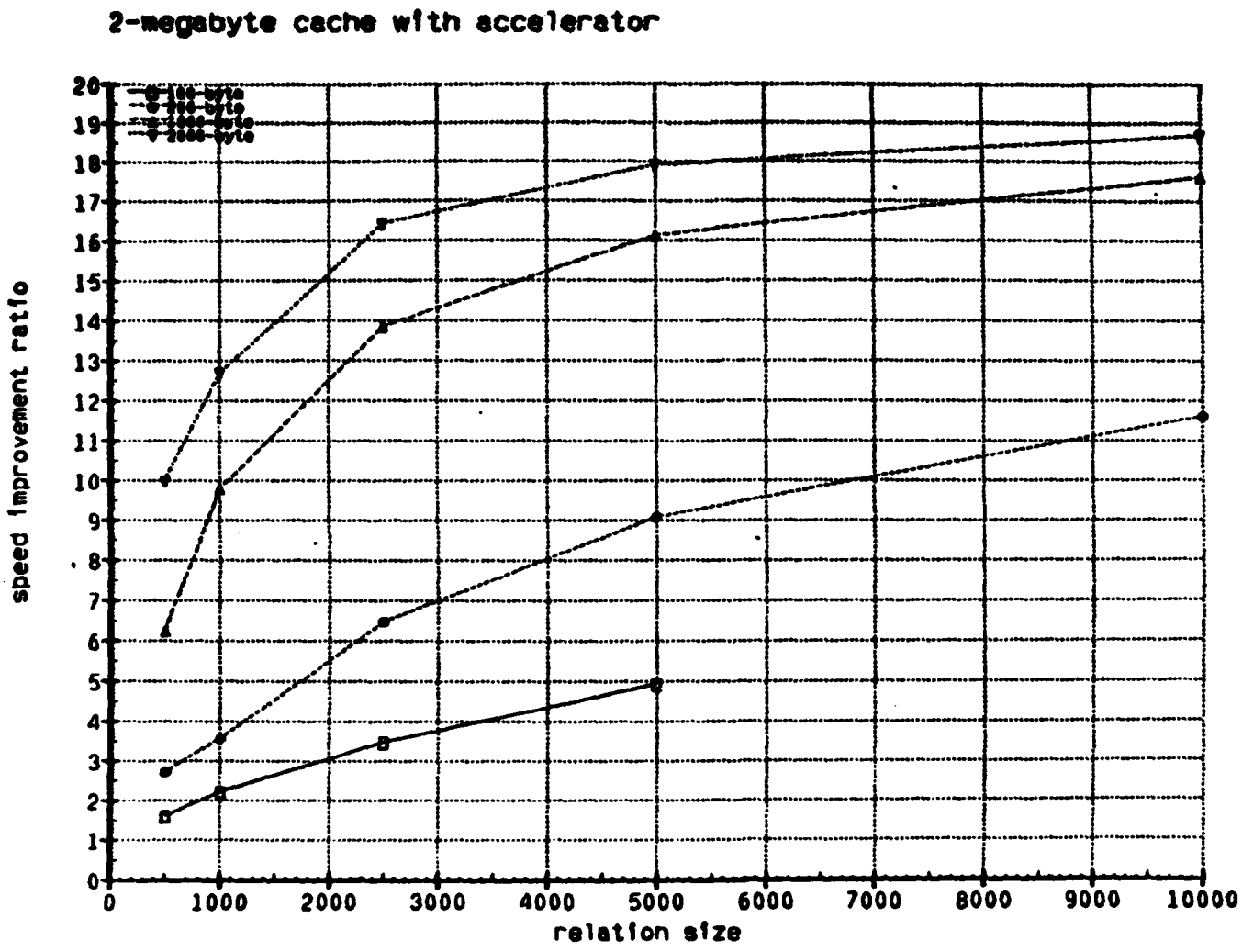


Figure 4.5 Speed Improvement using Non-Clustered Index on P5 and P10.

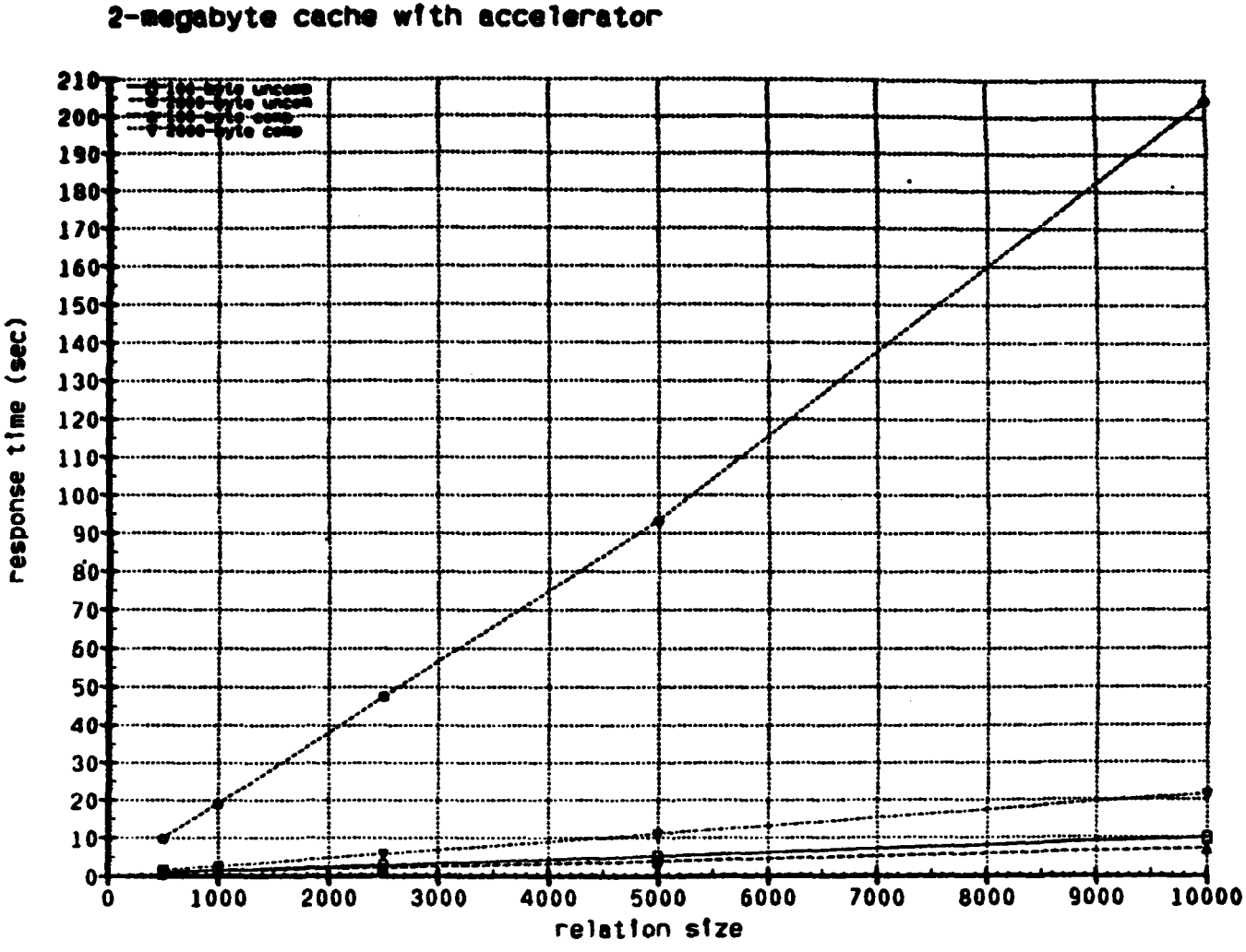


Figure 4.6 Effects of Data Compression.

More specifically, Figure 4.6 shows the results of the tests for the relations of 100-byte tuple size and the 2000-byte tuple size, respectively. For the 100-byte tuple the storage requirement is reduced by approximately 50% when all attributes are fully compressed. In the case of the 2000-byte tuple size, the savings in storage is approximately 90%.

The graph shows a major improvement in the response time for compressed relations. From the steep slope of the line it appears evident that the greatest impact on system speed is the amount of data that must pass over the internal bus. The large reductions in tuple size for the compressed relation shows a clear advantage over the uncompressed relation. The delay becomes increasingly significant for relations of larger tuple sizes. Approximately, a delay factor of 10 for the larger tuple size and 10000-tuple relation is observable.

4. Effects of Ordering and Randomizing the Database Entries

Figure 4.7 shows the results of tests to measure the backend system's sorting capabilities. The relations used are stored in the backend; their tuples are ordered on their KEY attributes. The graph depicts retrieves with and without ordering specifications on the KEY attribute. There is a slight increase in the response time for the ordered retrieves, as might be expected. The differential line depicts the extra time necessary for the ordering, which increases as the relation size increases.

Figure 4.8 shows the cost of performing the ordering on the backend versus the host. In this case batch runs on the host are used to perform the queries. In general, the batch retrieves show a marked improvement in response time for identical queries over the run-stream queries used in

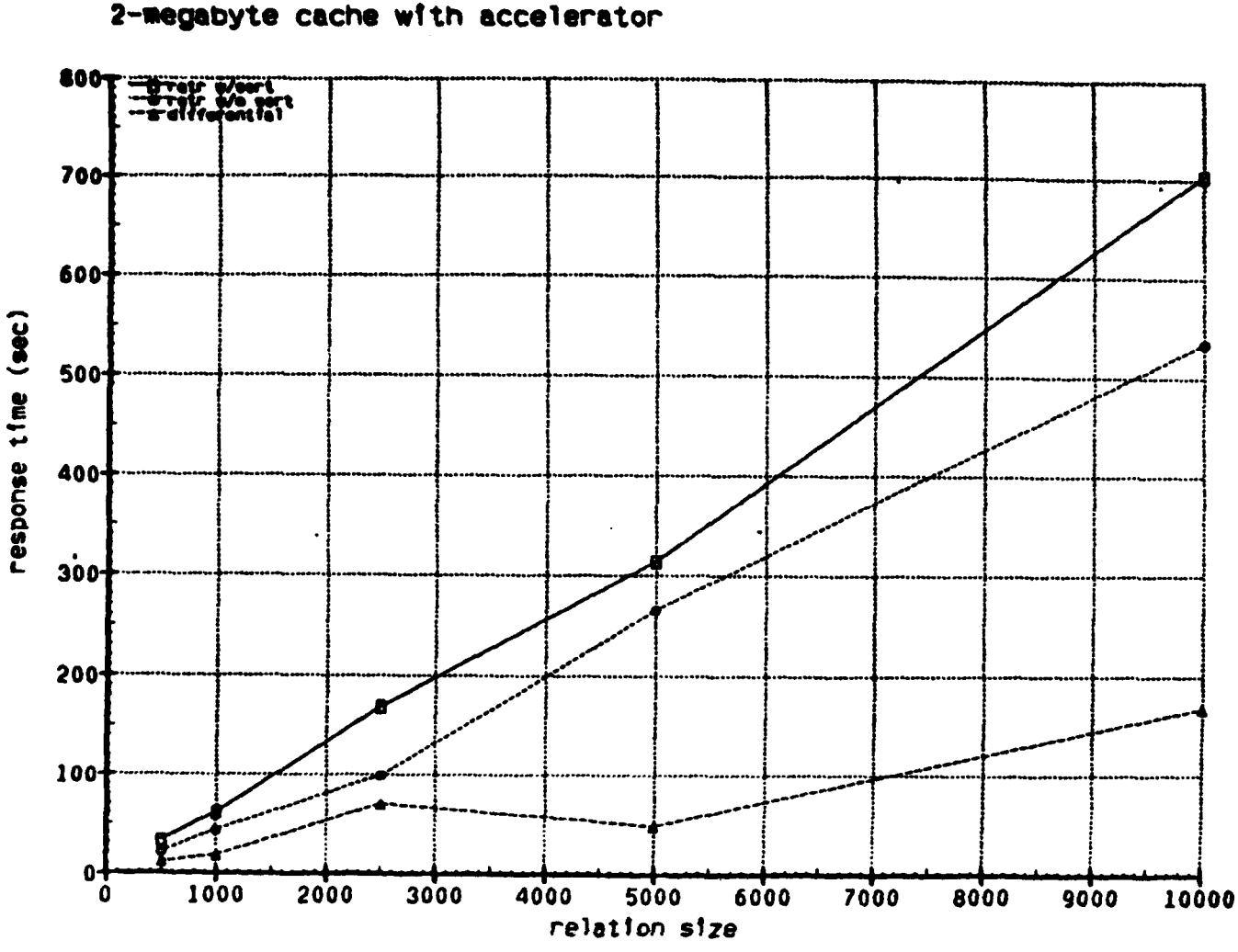


Figure 4.7 Effects of Ordering on the Response Time.

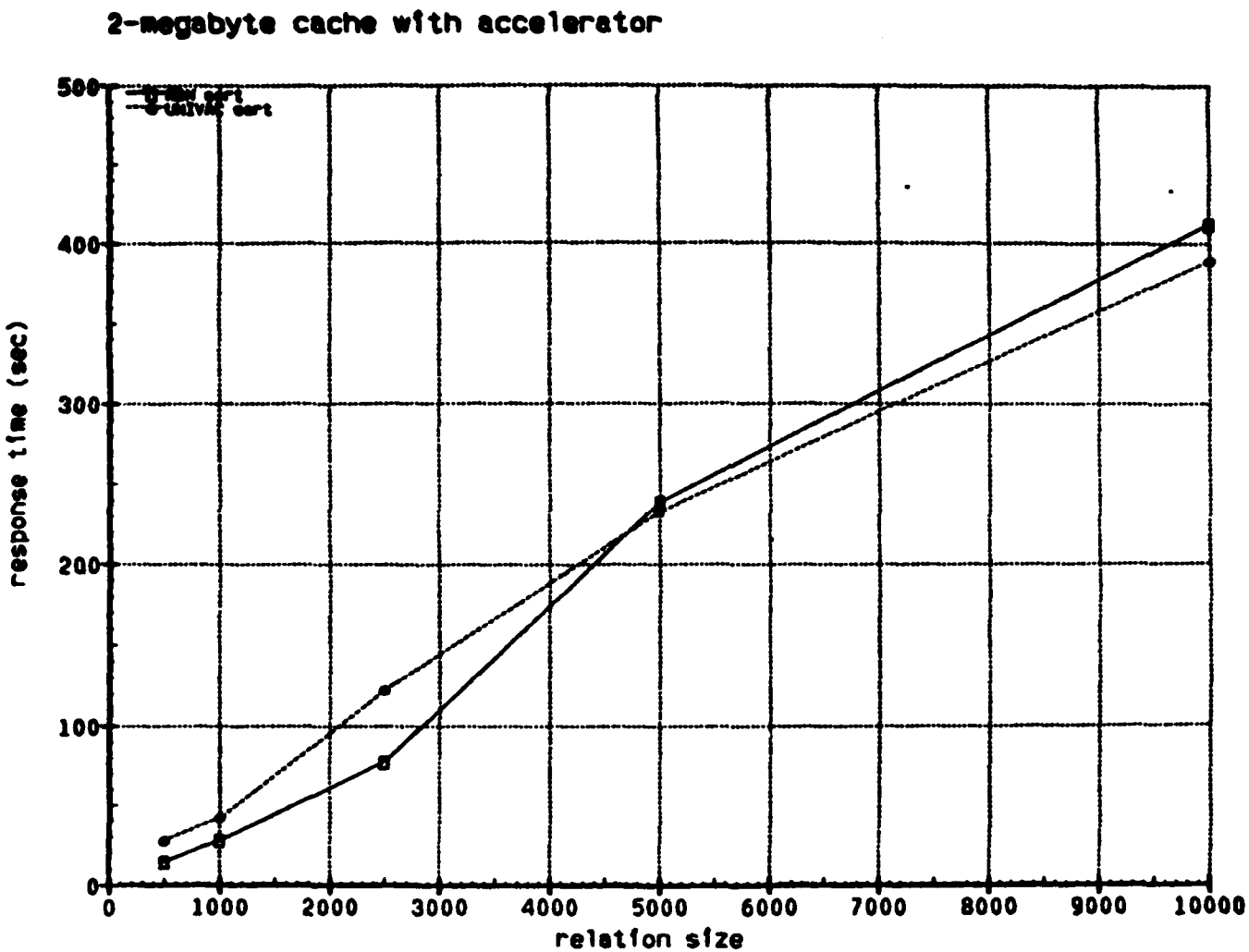


Figure 4.8 A Comparison of Host's and Backend's Ordering Capability.

Figure 4.7. This may be due to the decreased overhead cost for batch versus an interactive environment. Figure 4.8 also shows that for smaller-size relations the backend performs a more efficient ordering than the host does. Even for larger relations the sort time of the host and the sort time of the backend are comparable.

Finally, Figure 4.9 shows the effect of randomizing the order of the tuples in the relation. Using the random-number attribute to scatter the tuples in the relation, similar retrieves are performed on the ordered and randomized relations. In this case there is a non-clustered index on the KEY attribute for the relations. The graph shows minor variances in response times between the two, clearly indicating that the order in which the tuples are stored is not a significant factor in response time for the ordered retrieves.

E. CONCLUSIONS

The response times are generally linear, increasing as the amount of data to be returned is increasing. The amount of data may be varied as the number of tuples in a relation or the width of the tuples.

The creation of indicies on tuples shows significant improvement in response times when the retrieve command is qualified on the indexed attributes. The indicies provide marked improvement as the tuple size increases.

The effects of data compression shows some interesting results. Figure 4.6 has shown a very large improvement for compressed tuples. This improvement is most likely attributable to the decrease in the number of disk blocks accessed. In fact, the difference in time is proportional to the decrease in the number of blocks used for the tuples.

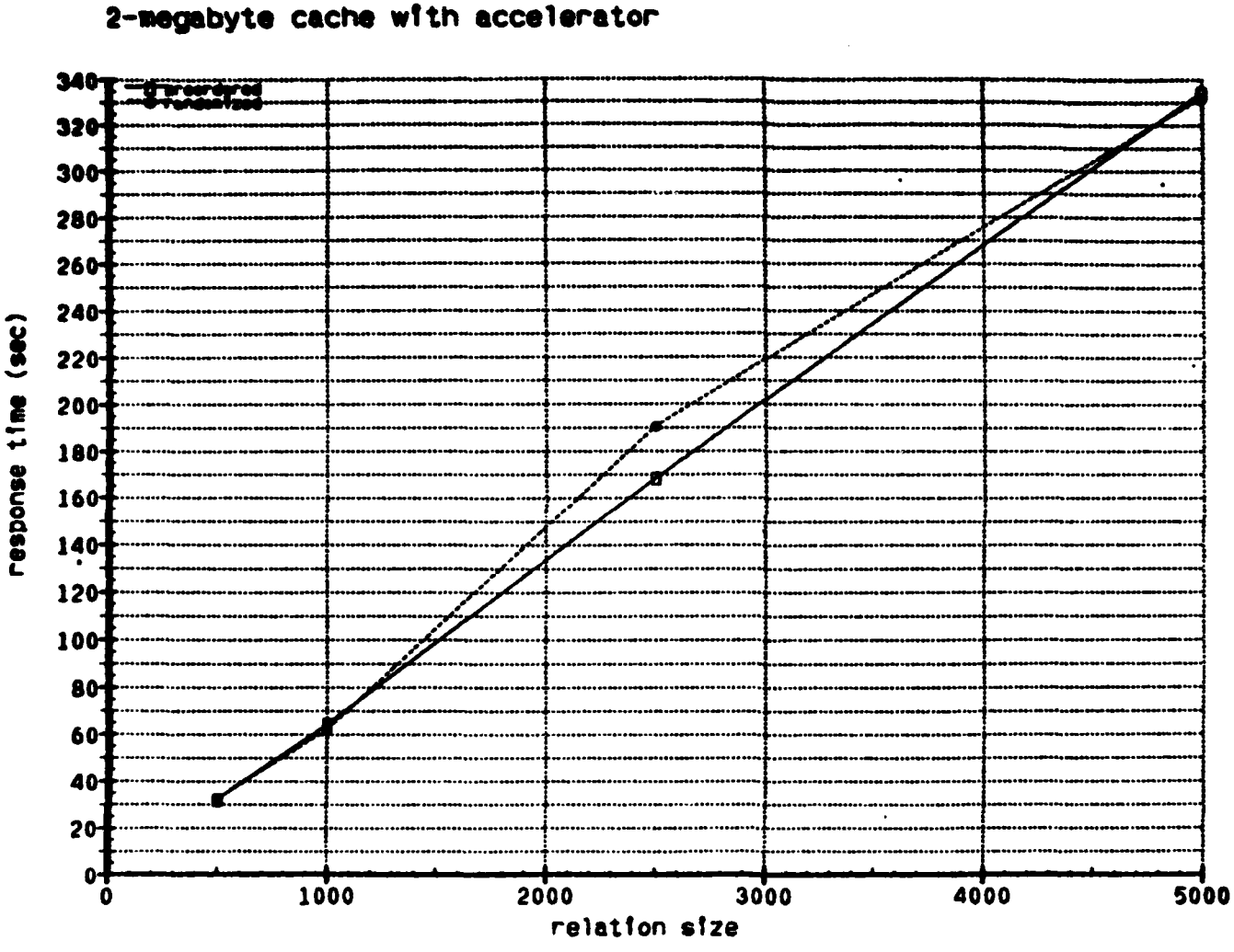


Figure 4.9 Effects of Ordering on Ordered and Random Relations.

Finally, the ordering test shows that the backend can sort tuples at least as fast as the host can. Naturally, the major portion of the time is spent in transferring the data from the disk to either the host or the backend; but, nevertheless, the backend proves more efficient for the smaller size of relations.

V. PERFORMANCE EVALUATION OF PROJECTION OPERATION

A. DEFINITION OF A PROJECTION

Projection is a means to restrict the amount and to order the sequence of information returned to the user in a retrieval operation. More specifically, projection will restrict the attribute values that will be returned from each tuple selected. Projection and selection can be combined to limit the range of values returned. In addition, a user can rearrange the ordering of the attribute values as the relation is displayed by varying the order of the attribute names in the target list. This is not to say that the actual order of the stored relation is altered but that the subset displayed to the user is ordered according to his specifications.

B. PROJECTIONS IN THE QUERY LANGUAGE

In RQL the user is given considerable latitude to describe precisely which attribute values that he wants to be returned. Using the 100-byte relation described in Table 2.1 as a format for a relation A, the RQL command:

```
RETRIEVE ( A.KEY,A.MIRROR )
```

will return to the user only those attribute values in the relation A whose attribute names are KEY and MIRROR. The user can list as many attribute names as he desires and place them in any order in the target list of the RETRIEVE command. In the case where all attribute values of a relation are to be listed, the user may simply use A.ALL. All

attribute values, i.e., entire tuples, will be returned in order as they are stored. The user can also add qualifiers to restrict the number of tuples returned. These qualifiers need not be on the attributes listed. For example,

```
RETRIEVE ( A.KEY,A.MIRROR ) WHERE A.P5 = "RED"
```

will again return to the user only those attribute values in the relation A whose attribute names are KEY and MIRROR. In addition, the qualifier will restrict the tuples returned to those whose P5 attribute value is RED. This RETRIEVE command also illustrates the means to perform a percentage selection. The P5 attribute values are colors selected from an enumerated set. Each different color value in the P5 attribute is present in 5% of the tuples in the A relation. Using these known percentages, the P5 qualification will select exactly 5% of the tuples in relation A.

C. AN ENVIRONMENT FOR THE MEASUREMENTS

The projection measurements discussed here are all on the same system configuration with 2-megabyte cache memory and the optional accelerator. Lack of time has prevented us from obtaining measurements on other configurations.

The projection measurements are conducted for four tuple sizes, i.e. 100-byte, 200-byte, 1000-byte, and 2000-byte, in three percentages of returns, 25%, 50%, and 75%. These percentages refer to the number of attribute values in the tuple that is returned. With the exception of the 100-byte tuple size, these are exact percentages; in the 100-byte case, the number of attributes returned was 29% and 71%. This is due to the tuples in the 100-byte relation having 14 attributes. A strict percentage of 25% and 75% was not

attainable. Nevertheless, they are still referred to as 25% and 75% projections. Further, the retrieval commands are qualified by 5% and 10% selections in order to reduce further the amount of data to be returned. Each query is executed 10 times, each time with a different qualification. This is done to eliminate any effects due to the location of the data in the relation and provides a better average response time.

D. PROJECTION MEASUREMENTS

The test queries used are qualified on the P5 and P10 fields of the relation to perform the aforementioned selection. Each query is then repeated 10 times with a different qualifier. The figures represent the average response time for those ten tests. Each graph shows the response time in seconds plotted against the number of tuples in the relation.

1. Percentage of Projections on Non-Key Attributes

In general the difference in response times for the five-percent and ten-percent selections is negligible, this is particularly true for the smaller-size relations. Doubling the number of tuples returned in a query can result in approximately a 20% increase (i.e., 1/3 second increase in the response time on the average) in the smaller tuples and a 10% increase (i.e., 7 seconds on the average) in the larger tuples. Figures 5.1 and 5.2 show the results of a 25% projection over varying tuple widths, with Figure 5.1 for a 5% selection and Figure 5.2 for a 10% selection. As can be seen, the graphs in these two figures are nearly identical. This is also the case for the graphs on the 50% and 75% projections. For example, in Figures 5.3 and 5.4, similar graphs for the 5% selection with 50% and 75%

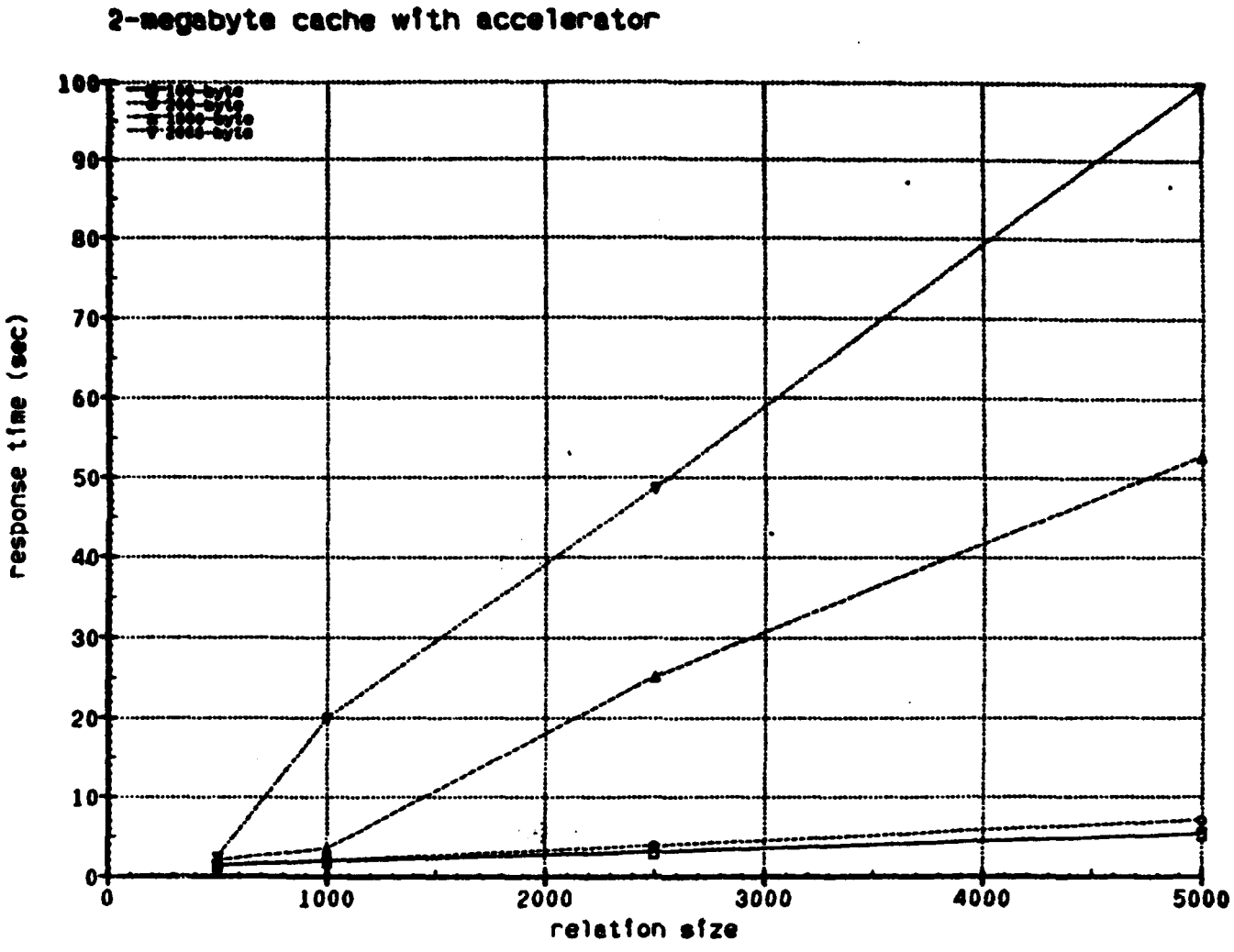


Figure 5.1 25% Projections on 5% Selections.

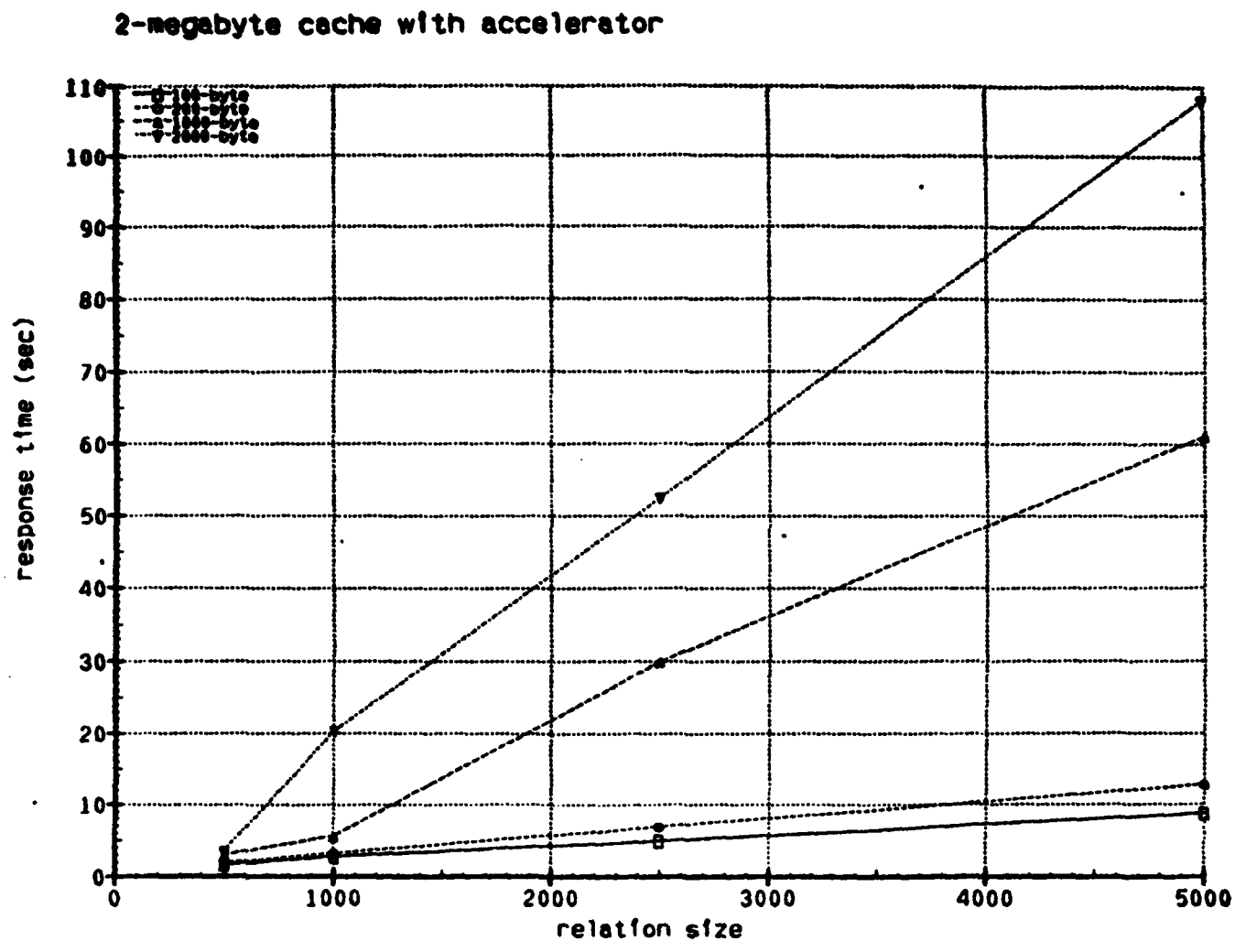


Figure 5.2 25% Projections on 10% Selections.

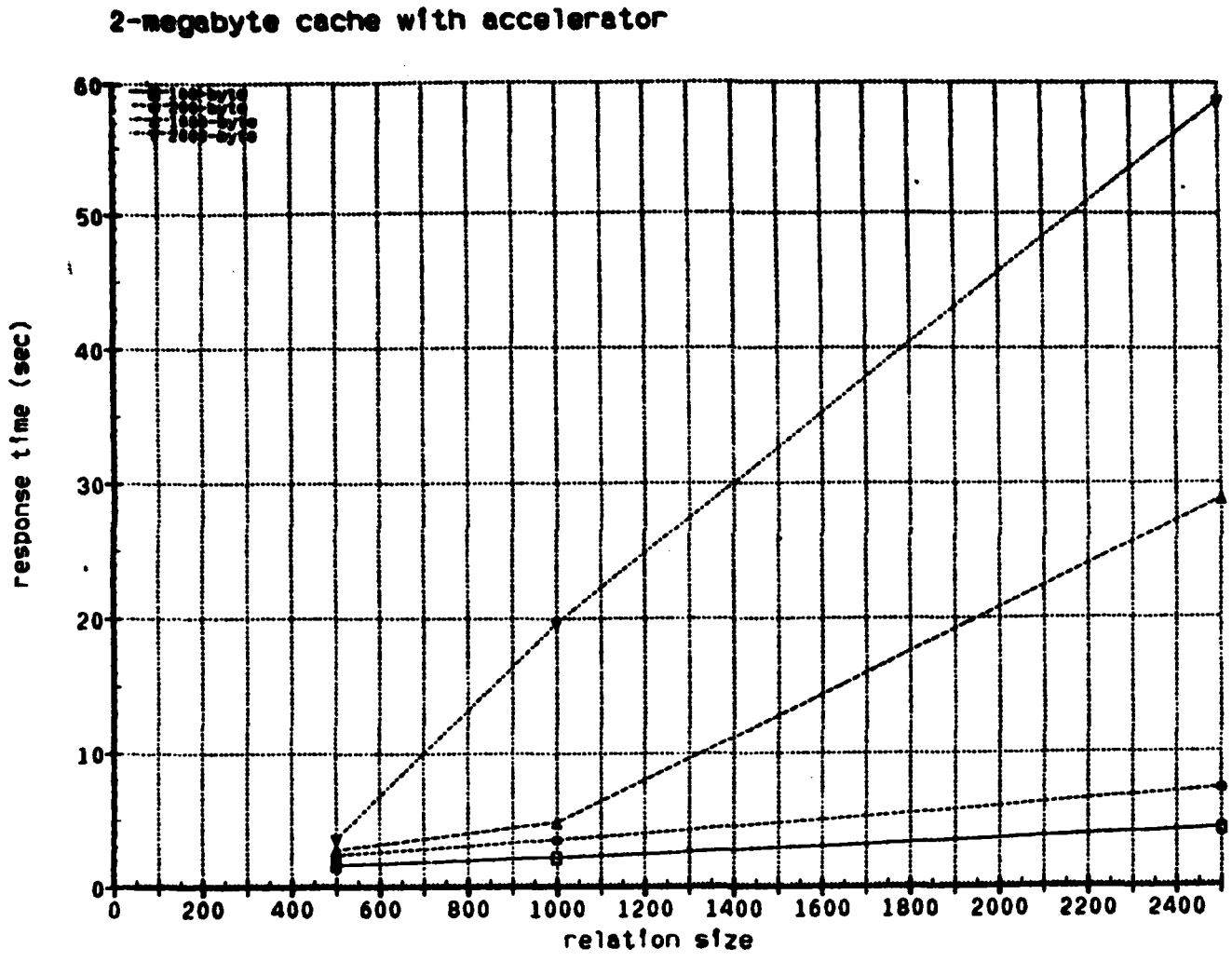


Figure 5.3 50% Projections on 5% Selections.

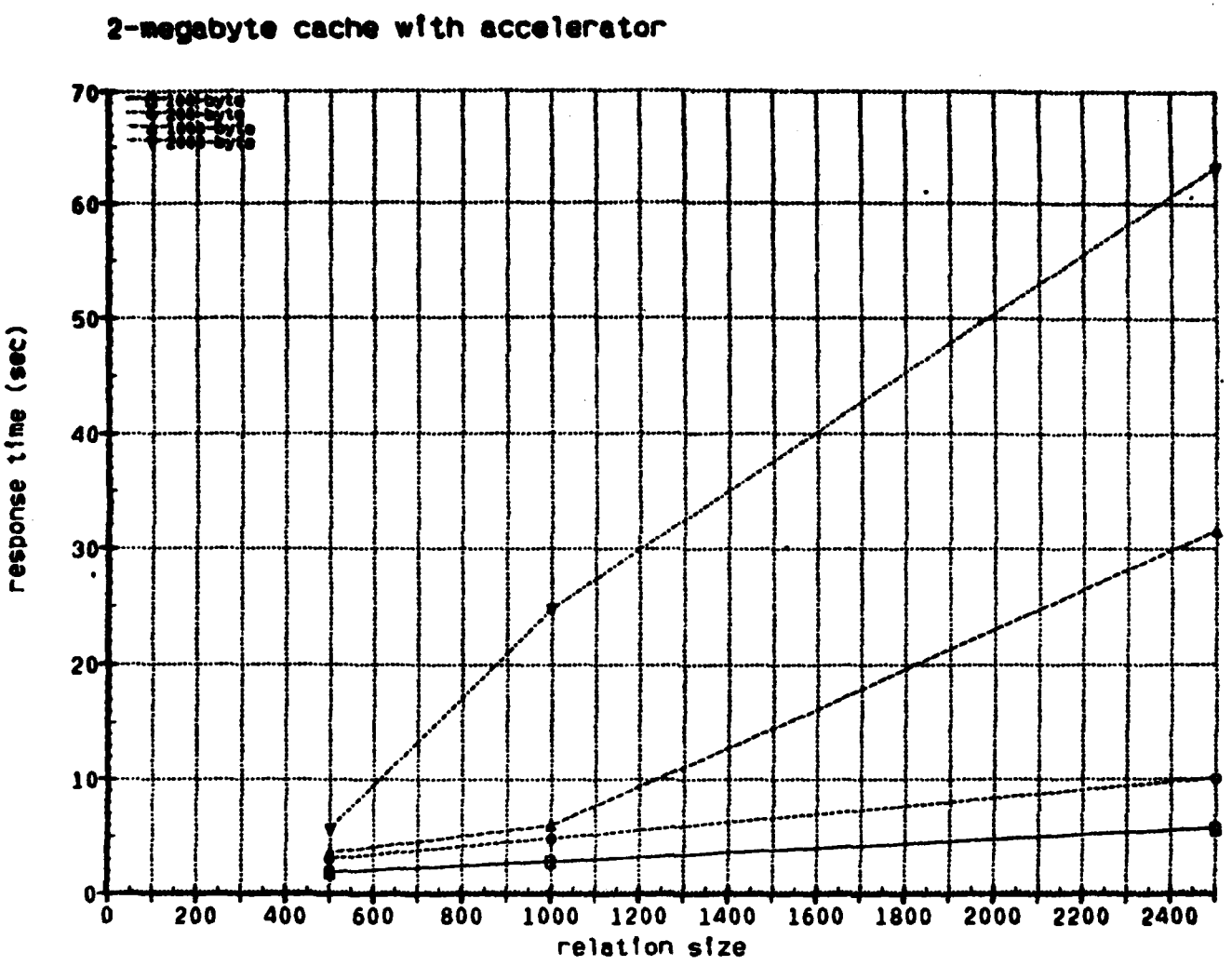


Figure 5.4 75% Projections on 5% Selections.

projections are displayed respectively. In each graph of the aforementioned figures response times increase almost linearly as the relation size increases, and increase dramatically as the number of attribute values returned increases.

Figures 5.5 and 5.6 give a different perspective on the same data. In this case the time for differing projection sizes is graphed over a constant tuple width. As expected, the greater the number of attribute values returned, the larger the response time. Again a much steeper slope is evident in Figure 5.6 for the bigger-width tuple.

2. Comparison of the Equivalent Queries on Selection

Figures 5.7, 5.8, 5.9, and 5.10 show the differences in the response time as the number of attribute values returned per query is varied. In each graph, the tuple size remains constant. In addition to the varied projection percentages, a fourth line representing a selection, in which all attribute values in each tuple, (i.e., the entire tuple) are returned, is added. The test queries used for the line marked 'full select' use the ALL specification to return all attribute values in each tuple. As in the projection measures, each such query is repeated 10 times. The 5% selections are done on the P5 field and a different value is used in the qualifier for each of the 10 queries.

As would be expected each figure shows a marked difference in the response time as the number of attribute values returned is increased. The smaller-width tuples in Figures 5.7 and 5.8 show a nearly linear increase in the response time as the relation size (the number of tuples of the same tuple width) increases, and an increase in the slope of the line as the projection size increases. In Figure 5.7 the response time for full select is strictly

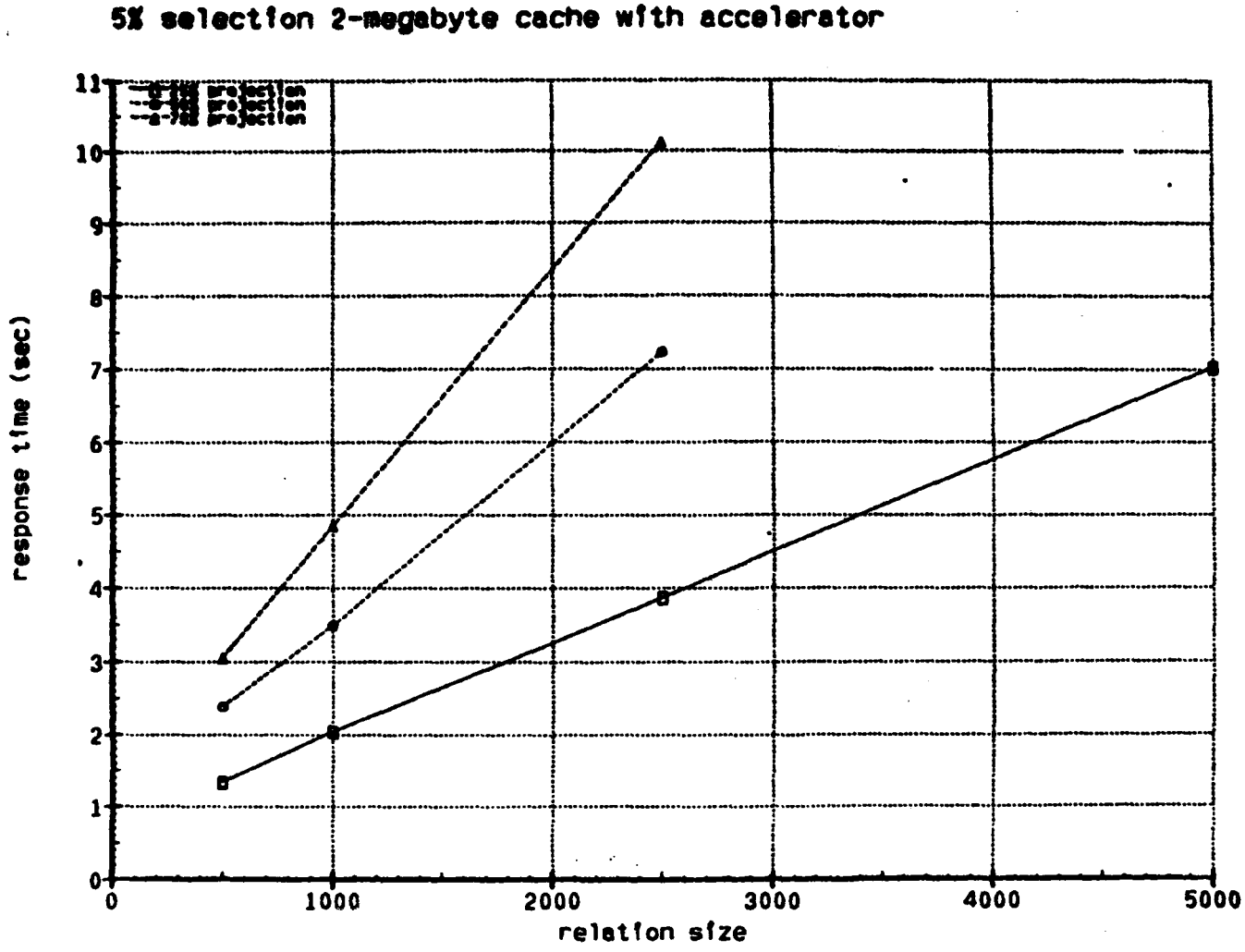


Figure 5.5 Effects of Differing Projection Percentages for 200-byte Tuples.

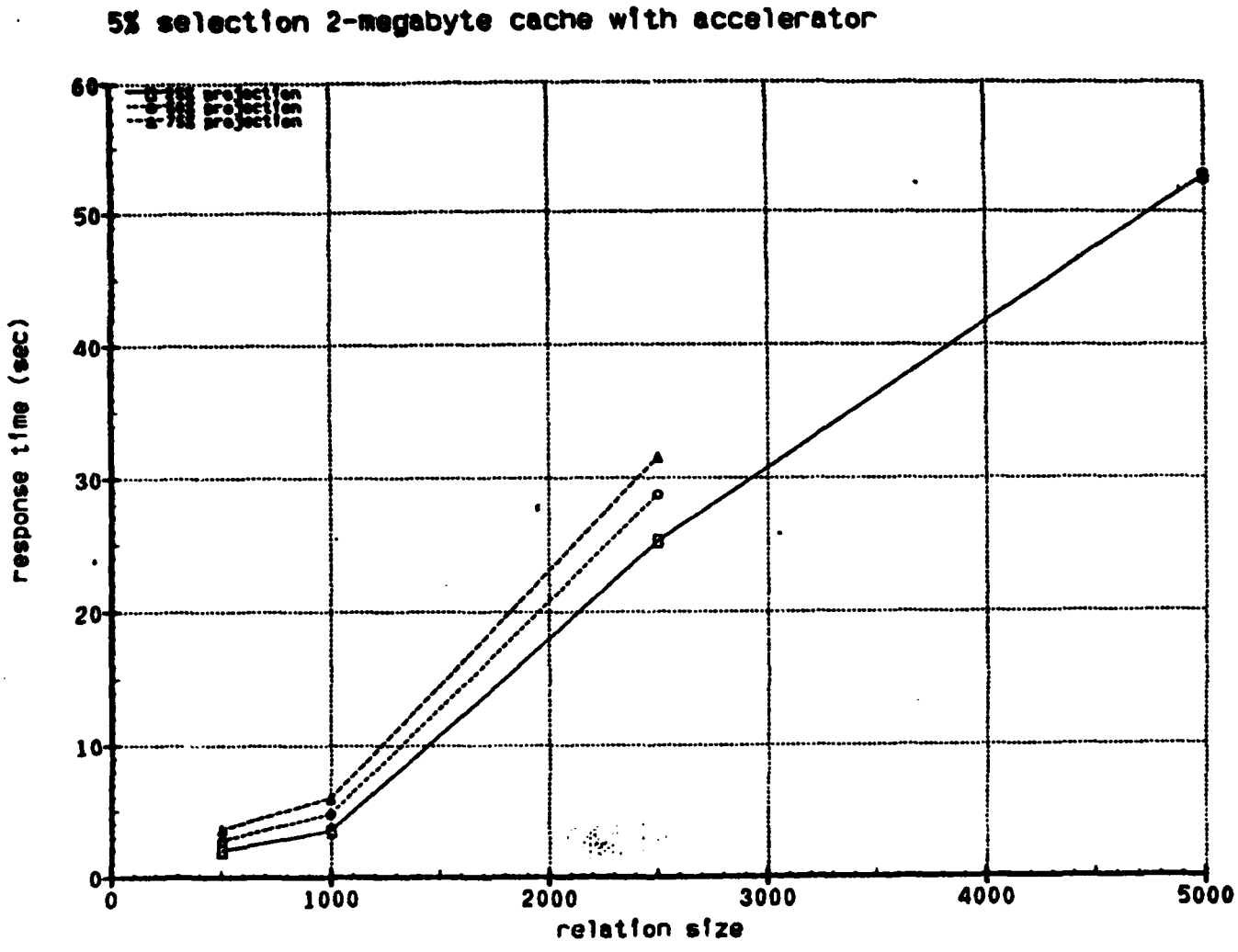


Figure 5.6 Effects cf Differing Projectin Percentages for 1000-byte Tuples.

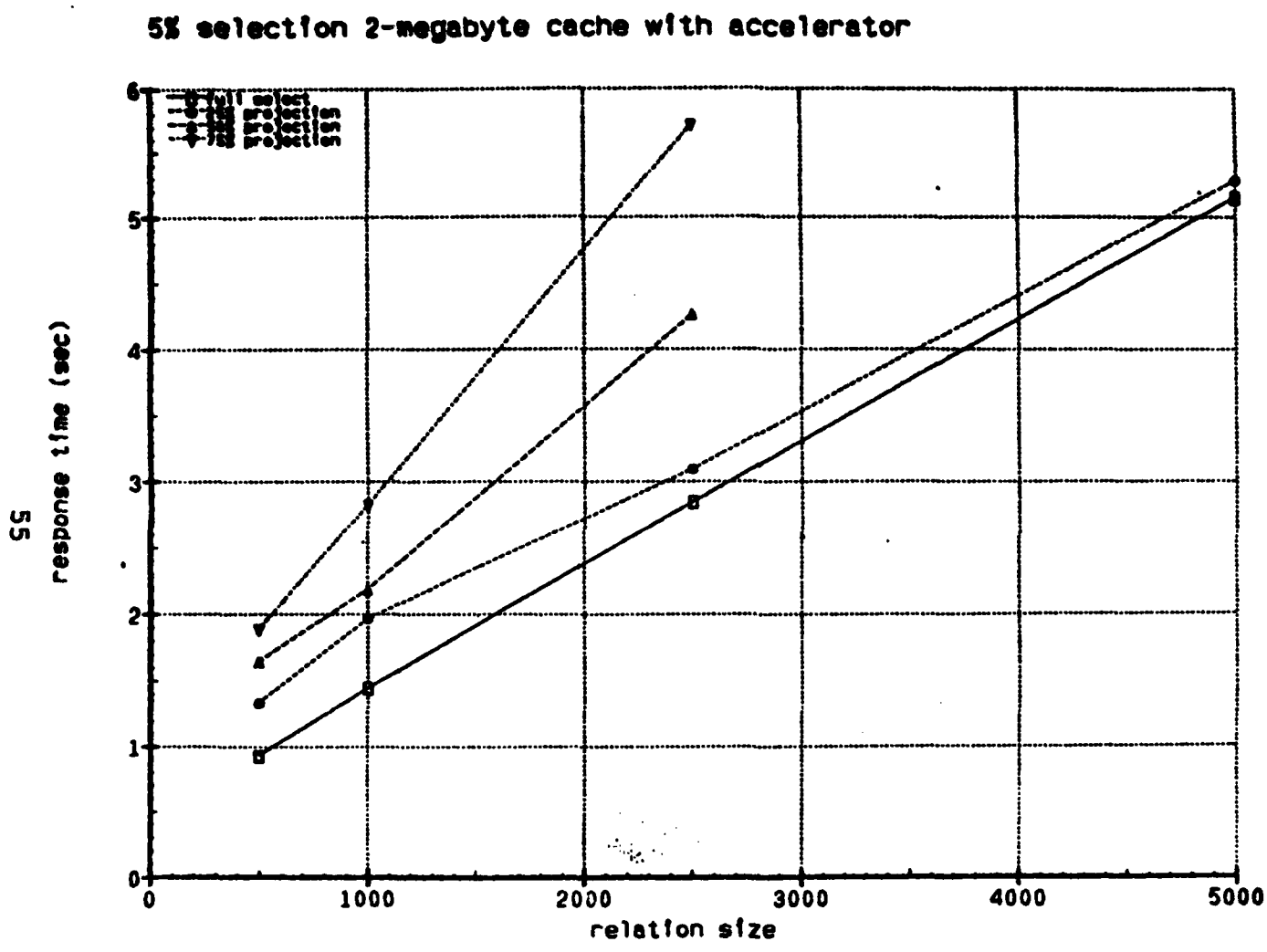


Figure 5.7 Comparison of Projection and Selection for 100-byte Tuples.

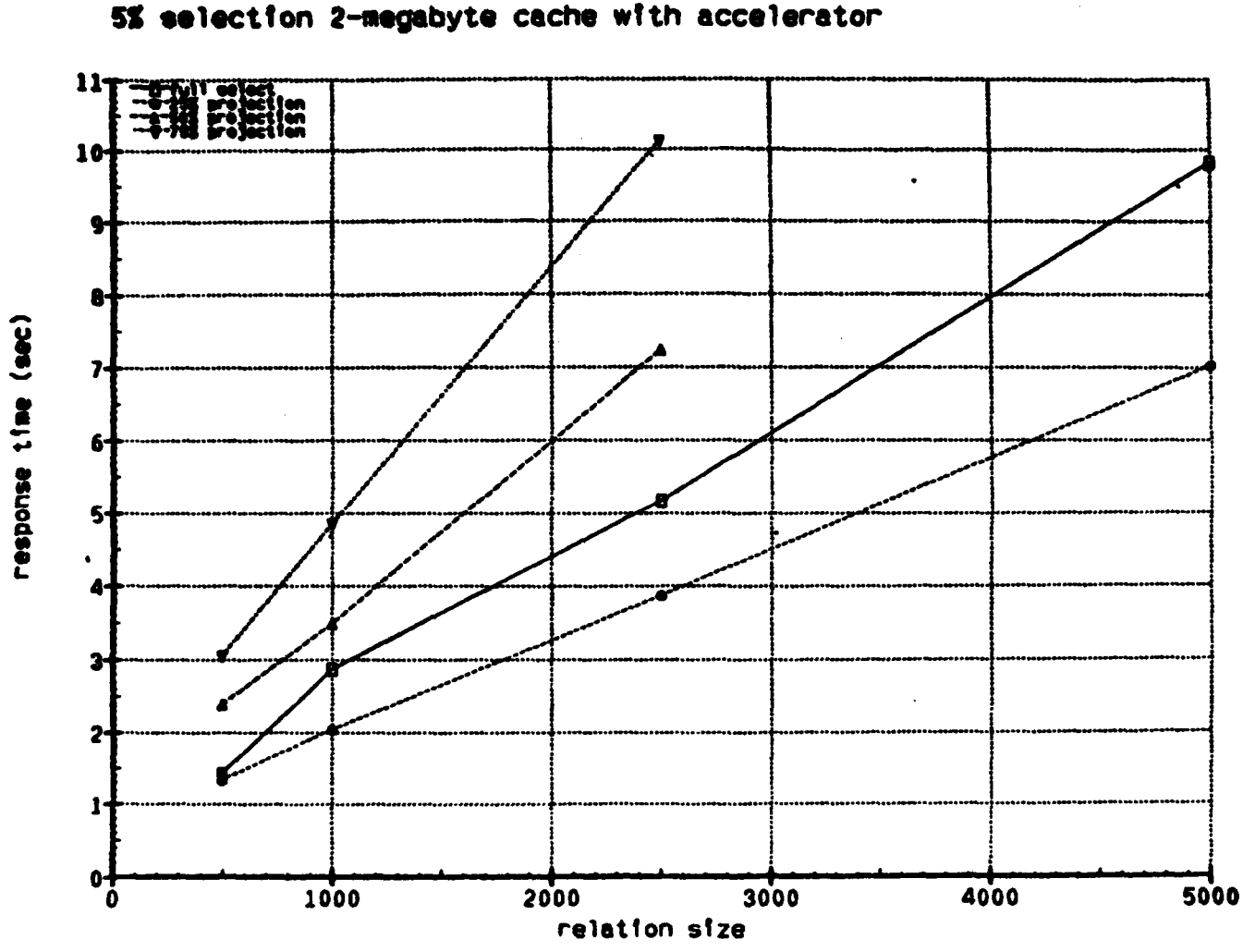


Figure 5.8 Comparison of Projection and Selection for 200-byte Tuples.

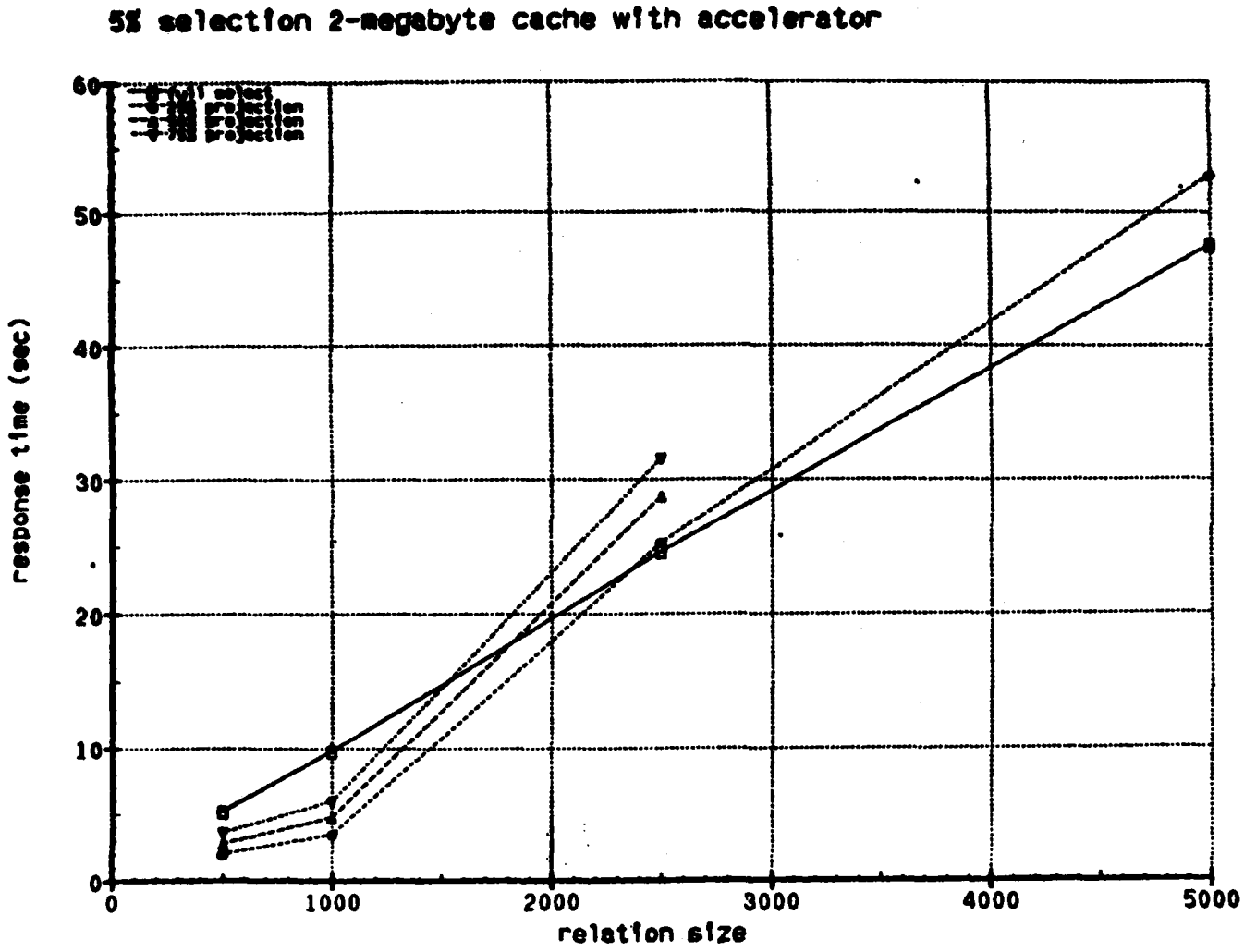


Figure 5.9 Comparison of Projection and Selection for 1000-byte Tuples.

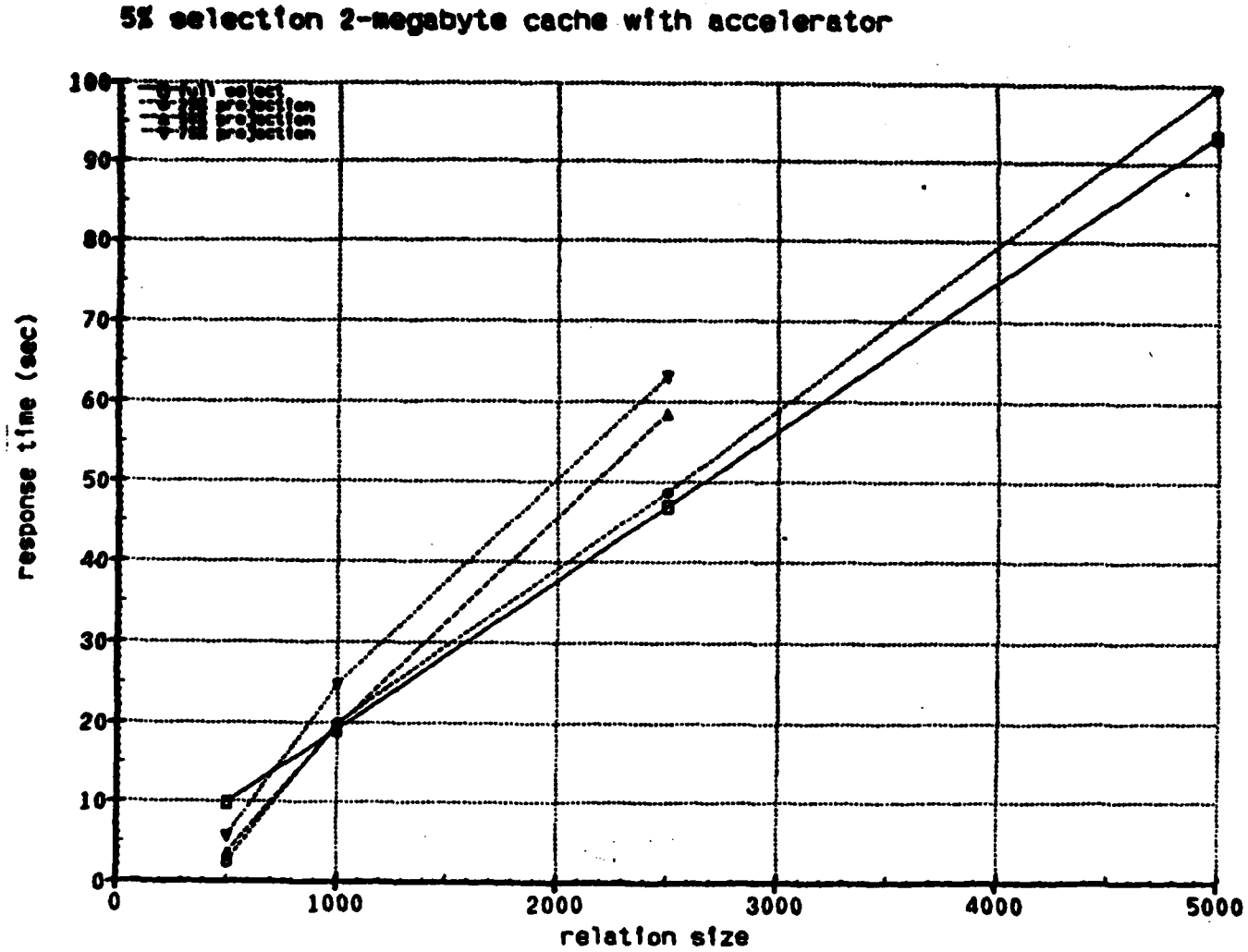


Figure 5.10 Comparison of Projection and Selection for 2000-byte Tuples.

smaller than any projection time, which indicates that for the smaller tuples the backend does a strict selection prior to extracting the attribute values specified in the projection qualifier. As the tuple width increases, the full select may take more time than that of the projection. For the 200-byte tuple in Figure 5.8, the full select time is again nearly linear, and the times are slightly more than the times for a 25% projection. The difference in response between the full select and the 25% projection steadily increases as the relation size increases, but even so the full select is faster than the 50% and 75% projections.

For much-bigger-width tuples, Figures 5.9 and 5.10 show that the full select time is higher than the projection time for the small percentage projections. The full select, however, has a much smaller slope, thereby crossing the line of the projection time and eventually showing a trend of quicker response as the relation size increases. Also of particular note is the uniformity of the curves for the varying projections in the 1000-byte and 2000-byte tuples in Figures 5.9 and 5.10. In contrast, for the smaller tuples the lines are nearly linear with increasing slopes. The lines for the larger tuples are not linear and the slopes are very even.

E. CONCLUSIONS

In general, the projection results are very predictable in that the response time is nearly linear and the response time increases as the amount of data returned increases. The amount of data may be determined by either the relation size or the projection size.

The full select comparisons in Figures 5.7, 5.8, 5.9, and 5.10, on the other hand, show some unanticipated results. Instead of showing a clear advantage in the

response time for full select in all relation sizes, as might be expected, the results vary with the tuple widths. In the smaller tuple width as depicted in Figure 5.7, the full select appears to run faster even though the amount of data returned is greater. For the 200-byte tuples as depicted in Figure 5.8, the relationship is markedly different. For the larger tuples as graphed in Figures 5.9 and 5.10, the full select requires more time for the smaller relations. Nevertheless, its advantage becomes evident as the relation size increases. In summary, the full-select operation is sensitive to the width of the tuples. In other words, the greater is the tuple width; the higher is the select time. The full-select operation is also sensitive to the size of the relations, although in an opposite way. That is, the larger is the relation; the smaller is the select time in proportion to the projection time.

It is difficult to determine what effect the cache and accelerator with other configurations may play in these tests. A need exists for more research in this area to verify the figures and collect more data over a wider range of tuple widths and relation sizes in hopes of obtaining a clearer trend to the relationship of the full select and the projections as the widths and sizes varies.

VI. CONCLUDING REMARKS

A. OVERALL OBSERVATIONS OF THE MACHINE PERFORMANCE

The experiments described in Chapters IV and V show some predictable results as well as some unexpected surprises. Generally the simple select operations, with or without indicies, display expected trends. The response time increases as the amount of data to be returned to the host increases, as shown in Figures 4.1 and 4.5. A similar trend is seen for relations with compressed attribute values. As Figure 4.6 illustrates, reduction in the response time can be significant for the large tuple widths where the degree of compression is high. The relations with indicies also show expected improvements in the response time for retrieves qualified on these attribute values.

Some unexpected results, however, are seen for the test results dealing with ordered retrieves, Figure 4.8. The backend shows an unexpected superiority in sorting over the host for smaller-size relations. Even for the large relations, up to 10000 tuples, the backend maintains a response time comparable with the host. One would expect that the mainframe would have a significant advantage in computing power and show a major improvement when the relation is ordered in the host instead of in the backend.

Another interesting result is the effect of clustered and non-clustered indicies on ordered retrieves. Creating a clustered index on a relation will cause the tuples to be stored in a specific order while a non-clustered index does not imply any ordering of the tuples. Figure 4.3 shows very similar response times throughout the range of relation sizes, regardless of whether the index is clustered or

non-clustered. This implies that the retrieved tuples are sorted even when a clustered index exists for the qualifier attributes.

The tests concerning projection of tuple attributes in Chapter V again show predictable results. Through all the figures for differing projection percentages and tuple widths, the graphs display near linearity in both dimensions. The response time increases as the tuple width or the number of tuples returned increases. But surprising results are evident when comparing projection to full selection.

Consider Figures 5.7, 5.8, 5.9, and 5.10 again. As explained in Chapter V, the overlay of the full select on the varying projection sizes shows no positive trend. The projection measurements are consistent throughout the figures, yet the full selects relationship to the projections varies from one figure to the next. Two of the four figures indicate that it is cheaper to retrieve entire tuples than to project attribute values from the tuple. One figure indicates that beyond a fixed relation size, it is cheaper to retrieve entire tuples. The fourth figure seems to indicate that some degree of projection is always cheaper than retrieving the entire tuple. No clear conclusion can be drawn. More tests over a wider range of tuple widths are required to identify an overall trend or relationship between projection percentage and the full selection retrieves.

B. DATABASE AND MACHINE LIMITATIONS

When considering the test environment, two specific limitations stand above all else. The first of these is the low resolution of the clock from which measurements are taken. The standardized use of the GETTIME function

throughout the tests has made comparison of various test results over differing periods meaningful. Even so, the low resolution makes the need for average times over many similar test runs a necessity. This greatly limits the amount of time that one can spend in running more meaningful tests and in verifying previous results. A great effort has been made to find some other timing mechanism. In the end, GETTIME proves to be the easiest to use, the most consistent, and, most importantly, the easiest to control.

The second limitation concerns the system configuration and the inability to control the environment of both the host and the backend. The performance of these tests has not been a very high priority of the parent command at Pt. Mugu. This is to be expected, since the host machine is in a production environment. Gaining exclusive use is very difficult and extremely costly. With this restriction, our tests are limited to weekend and evening runs, at times of relatively low activity. This significantly reduced the time of system availibility. Also, in terms of the environment, the backend system we used is a relatively new piece of equipment. Lastly, the sytem configuration has been changing frequently during the experimentation period. The time each configuration becomes available has been short. Consequently, not enough data can be collected to make any significant comparisons.

C. RECOMMENDATIONS FOR FUTURE BENCHMARKING EFFORTS

In light of the test results discussed here, the direction of future work should be toward effects of various indicies and ordering capabilities. The results of tests on various types of indicies and the ordering of relations show the most startling results. In addition, some work is required over a wider range of tuple widths to refine previous results.

Another aspect that warrants research is a mix of tests to simulate a more realistic system load, specifically tests with multiple users of the backend and a more realistic host workload. The tests in this thesis are runs on an unloaded system. In actuality, the use of the system will most likely occur closer to peak loading. Perhaps different trends may develop when the host and/or backend are subjected to different load conditions.

Even though these tests are on a specific system, they are general enough in nature to provide insight for tests on other relational machines and to aid in making a comparison of different backends.

LIST OF REFERENCES

1. Gibson, J. C., The Gibson Mix, IBM Tech. Rept. TR00.2043, June 1970

2. Flynn, M. J., "Trends and problems in computer organizations", Information Processing (Proc. IFIP Congress 74) 1974

3. Knuth, D. E., An emperical study of FORTRAN programs, Software - Practice and Experience 1, 1971

4. Crocker, M. D., Benchmarking the Join Operation of a Relational Database Machine, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1983

5. Ryder, C. J., Benchmarking Relational Database Machines' Capabilities in Supporting the Database Administrators' Functions and Responsibilities M.S. Thesis, Naval Postgraduate School, Monterey, California, Sept 1983

6. Stone, V. C., Design of Relational Database Benchmarks M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1983

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4. Curricula Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93940	1
5. Dr. D. K. Hsiao, 52, Hq Computer Science Department Naval Postgraduate School Monterey, California 93940	1
6. Ms. Paula Strawser, 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
7. LT Robert A. Bogdanowicz, USN Computer Science Department Naval Postgraduate School Monterey, California 93940	2
8. LT Michael D. Crocker, USN Computer Science Department Naval Postgraduate School Monterey, California 93940	1
9. LCDR Vincent C. Stone, USN NAVGMSCOL DAHLGRCK ATTN: Code 513 Virginia Beach, Virginia 23461	1
10. LCDR Curtis J. Ryder, USN Computer Science Department Naval Postgraduate School Monterey, California 93940	1
11. Ms. Doris Mleczko Data Processing Service Center West (Code 0340) Naval Air Station Pt. Mugu, California 93042	1
12. LT Linda Widmaier, USN 3016 Eromley Ct. Woodbridge, Virginia 22192	1

13. Mr. Joseph Majkszak (666-14) 1
Staff Consultant
Blue Cross/ Blue Shield Association
676 N. St. Clair
Chicago, Illinois 60611

14. LT Ernest L. Valdes, USN 1
491 Hawthorne St. Apt. 15
Monterey, California 93940

15. LT Warren W. Williams, USN 1
Route 1, Box 8
Dryden, Virginia 24243

END

FILMED

2-84

DTIC