Analysis of Database System Architectures Using Benchmarks

S. BING YAO, ALAN R. HEVNER, MEMBER, IEEE, AND HÉLÈNE YOUNG-MYERS

Abstract—**Database machine architectures have been proposed as a promising alternative to improve database system performance, control, and flexibility. While many claims have been made for the database machine concept, few studies have been made to test the performance advantages and disadvantages of a database machine in an application environment. A comprehensive benchmark study comparing the performance of database systems on a conventional computer system and a database machine is reported in this paper. The results show the database machine architecture to have superior performance in most cases. The performance advantage is sensitive to the communication line speed between the host computer and the database machine. The effects of line speed are studied and displayed in the benchmark results. A summary of the similarities and differences between the architectures based upon our benchmark results completes the paper.**

Index Terms—**Benchmarking, database machines, database system architectures, performance evaluation.**

I. INTRODUCTION

DATABASE management systems have traditionally run as large software programs on conventional computer system architectures (Fig. 1). The functionality and control requirements of the database system place a great processing requirement on the host computer system. It is well known that database systems tend to be processor-bound rather than input/output-bound for most applications [10]. The presence of a database system can significantly decrease the performance of a host computer system.

The backend database machine architecture has been proposed as a way to offload the majority of database activity onto a separate, specialized computer system (Fig. 2). This offloading releases the resources of the host for use by application programs, at the expense of purchasing a specialized database machine. The database machine architecture executes a database request as follows. An application program generates a database operation. The host computer parses the operation and forms a request for transmission to the database machine. The database machine receives, optimizes, and executes the request on the database. The result is returned to the host system for display to the user.

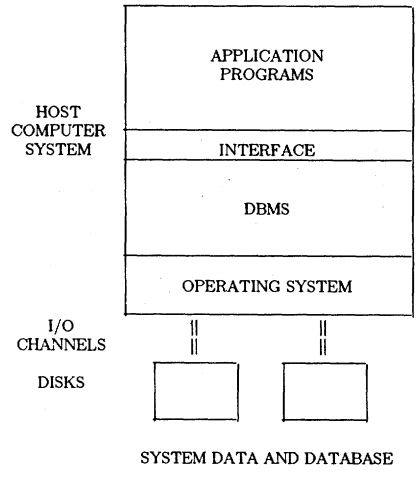


Fig. 1. Conventional database system architecture.

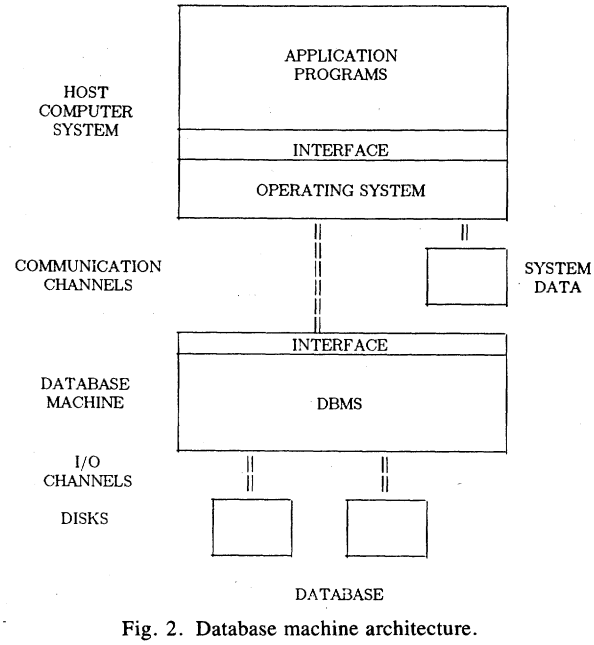


Fig. 2. Database machine architecture.

The database machine concept has created a great deal of interest and research [6], [7]. A database machine can be defined as a specialized software/hardware configuration designed to manage a database system. The simplest database machine implementation consists of a stand-

Manuscript received July 31, 1984; revised April 30, 1985. This work was performed with Software Systems Technology, Inc., College Park, MD, through Contract NB82SBCA1645 with the National Bureau of Standards.

The authors are with the Database Systems Research Center, College of Business and Management, University of Maryland, College Park, MD 20742.

IEEE Log Number 8714439.

alone, general-purpose computer running a database management system [19]. Since the system has a specialized purpose, the operating system can be streamlined to support database requests most efficiently [22].

An analysis of the potential advantages and disadvantages of the database machine architecture includes the following considerations.

1) Performance: Parallel execution in the host system and the database machine should result in improved overall system throughput. This assumes that the host system continues execution on other tasks while the database machine performs its work. The response time of database requests is affected by two factors. Each request will require the transmission of messages and data to and from the database machine. However, the specialized efficiencies of the hardware and software in the database machine should decrease the time required to execute the request.

2) Reliability: The ability to share a database machine among several host systems provides a greater overall distributed system availability if one of the hosts should fail. The database machine architecture, however, contains more system components that can fail independently.

3) Database Sharing: A database machine can service requests from more than one host system. Hosts may be directly connected to a database machine or the machine can act as a server on a communications network.

4) Security: A database machine limits database exposure to database requests from the host system. The database system is isolated from nondatabase users. Increased data sharing, however, places greater requirements on secure user authentication and data access control.

5) Modularity: A database machine can act as a standard, database system module for multiple processor systems. Host systems can be local or remote. Dissimilar hosts require a simple interface protocol to communicate with the database machine. A standard database module allows straightforward data transfer among databases.

Among the above considerations, the key issue for effective use of the database machine architecture is *performance*. Wide acceptance of database machines will only be achieved when its price/performance ratio is significantly better than that of the conventional database system architecture.

Several studies have been performed to evaluate the performance advantages and disadvantages of the database machine architecture. While simulation and analytic modeling have been useful in modeling aspects of database system behavior (e.g., data access costs [13]), *benchmarking* is the preferred method to evaluate performance because it can study a real, complete database system and its functionality. Both simulation and analytic modeling are limited in the scope of their system testing. Database system benchmarks have been used in several ways. On a single database system, different implementation algorithms or different system configurations can be tested. A multiple database system benchmark can compare the performance of the different systems on the same database and workload.

Researchers at the University of Wisconsin have developed a comprehensive strategy for benchmarking database management systems and machines [1], [4], [12]. A synthetic database is used for single-user and multiple-user tests on systems including Oracle, INGRES, SQL/DS, and the IDM-500 database machine. An advantage of using a synthetic database is that it can be automatically generated with precise control of data size and value distributions. Such control is necessary for performing sensitivity analyses. A synthetic database is also easily transportable among different database systems for testing purposes. Database benchmark research at the Naval Postgraduate School has studied the database machine architecture in single and multiple backend configurations [3], [11]. While also using synthetic databases, this research shows promising performance improvements in multiple database machine environments.

Additional benchmark studies have been performed on database machine architectures by other research groups and vendors [25], [9]. In general, the results demonstrate performance advantages for database machines over conventional database management systems.

Previous benchmark research on database systems has been performed primarily in an idealized environment of a synthetic database. This paper presents the results of a comprehensive benchmark study that uses a real database environment to compare a conventional database management system architecture to a database machine architecture. The use of a real database, as opposed to a synthetic one, provides a more realistic environment for the benchmark tests. The problems of designing, loading, querying, updating, and maintaining a real database are a better indication of potential user problems [24]. Further, we consider as performance variables not only the response time of the entire query (time-to-last), but also the time until the first record (time-to-first) is received by the user. Some interesting differences are found between these two performance variables. The benchmark results, while confirming the performance advantages of the database machine architecture, highlight the sensitivity of the database machine to several architectural factors. In particular, the database machine is sensitive to the communications line speed to and from the host. Via a parametric study, we analyze the sensitivity of the database machine architecture to this line speed.

The goal of our research is not to compare the performance of production database systems, but to identify the major distinctions between the different database system architectures that influence performance. Another issue that we do not emphasize in this research is the respective costs of the different architectures. The volatility of system costs and performance capabilities causes any price/performance analysis to be quickly outdated. However, system costs would be a major consideration in the selection of a database system. Section II details our benchmark methodology. The design and execution of the benchmark are described. In Sections III–V, we report the results of our studies. In conclusion, a summary of the performance similarities and differences between the ar-

chitectures are presented. Appendix A analyzes the effect of line speed on the database machine benchmark results presented throughout the paper.

II. Benchmark Experiments

The design and execution of the benchmark experiment are described in this section. The actual benchmark testing was performed during late 1983 and 1984.

A. Benchmark Methodology

Managing a database requires a complex system made up of hardware, software, and data components. A benchmark methodology for database systems must consider a wide variety of system variables in order to fully evaluate performance. Each variable must be isolated as much as possible to allow the effects of that variable, and only that variable, to be evaluated.

The benchmark methodology for database systems consists of three stages [29]:

1) Benchmark Design—Establishing the system environment for the benchmark; designing the system configuration, test data, workload, and variables of the benchmark studies.

2) Benchmark Execution—Performing the benchmark and collecting the performance data.

3) Benchmark Analysis—Analyzing the performance results on individual database systems and, if more than one system is benchmarked, comparing performance across several systems.

Fig. 3 illustrates the methodology.

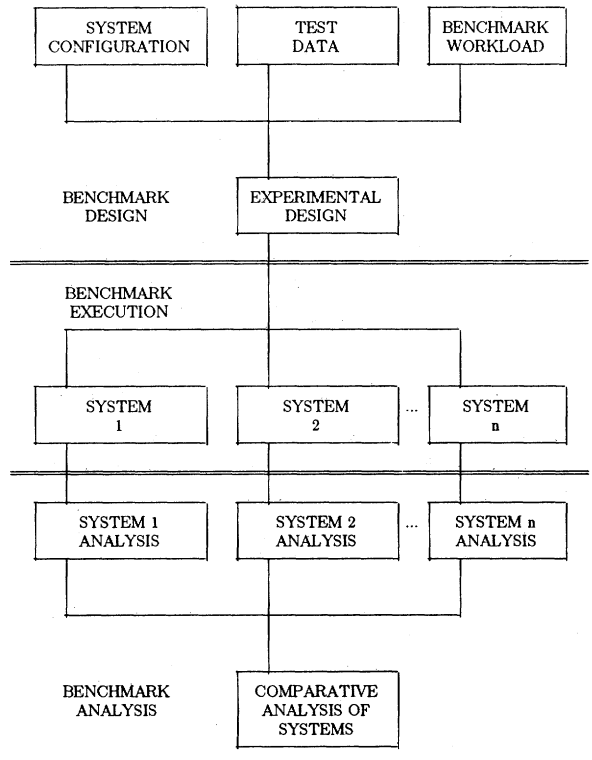


Fig. 3. Database system benchmark methodology.

B. Benchmark Design

The design of a benchmark involves establishing the environment of the database systems to be tested, and developing the actual tests to be performed. The four steps of the benchmark design phase are presented below. For a comparative benchmark over several database systems the benchmark design should be invariant over all systems.

1) System Configuration: We ran the benchmark experiment on a conventional, minicomputer database system and a database machine with a minicomputer front end. It is highly desirable to choose systems of the same data model. Although there are several popular data models, few have been implemented on a database machine. Experimental database machines have been structured on the network data model (e.g., NEC's experiment machine [18]) and the hierarchical data model (e.g., the ADABAS machine [21]). However, only the relational data model has been successfully used as a basis for a production database machine (e.g., Britten-Lee's IDM-500 and IDM-200). In fact, it is shown in [17] that the relational data model provides a more efficient interface to a database machine than other data models. For the experiments reported in this paper, the database machine selected uses the relational data model. For benchmark comparisons, then, the conventional database system selected also used the relational data model.

Oracle [20] was selected to be the database system for the conventional architecture. In its experimental configuration, the Oracle database system was installed on a VAX® 11/750 running VMS 3.0. The VAX system contained 2 Mbytes of main memory. The mass storage available on the VAX 11/750 consisted of a Digital Equipment RL02 system disk, and a 9766 Control Data Corporation disk drive with an unformated capacity of 300 Mbytes.

The IDM-500 database machine [16] represented the database machine architecture in the benchmark study. An IDM-500 database machine with one Mbyte of memory was installed and used for the benchmark. Release 24 of the IDM software was used. Mass storage for the IDM was also a 9766 Control Data Corporation disk drive. The IDM operated with a VAX 11/750 front-end computer running Berkeley UNIX™ 4.1. The data transfer between the VAX and the IDM-500 was through a 9600 baud RS232 communications line. We found the performance of the database architecture to be quite sensitive to the line speed between the host and the database machine. By using a relatively slow line speed in our studies, we were better able to study the performance impact of line speed. From the benchmark data with the 9600 bps line, we have analyzed the performance effects of speeding up the line. Our analysis of line speed sensitivity in the database machine architecture is found in Appendix A. Throughout the remainder of the paper we show the estimated performance of a one Mbps line along with the benchmark results from the 9600 bps line for the database machine architecture.

2) Test Data: For benchmark testing we used a file of personnel data from a large application system. From

®VAX is a registered trademark of Digital Equipment Corporation.
™UNIX is a trademark of AT&T Bell Laboratories.

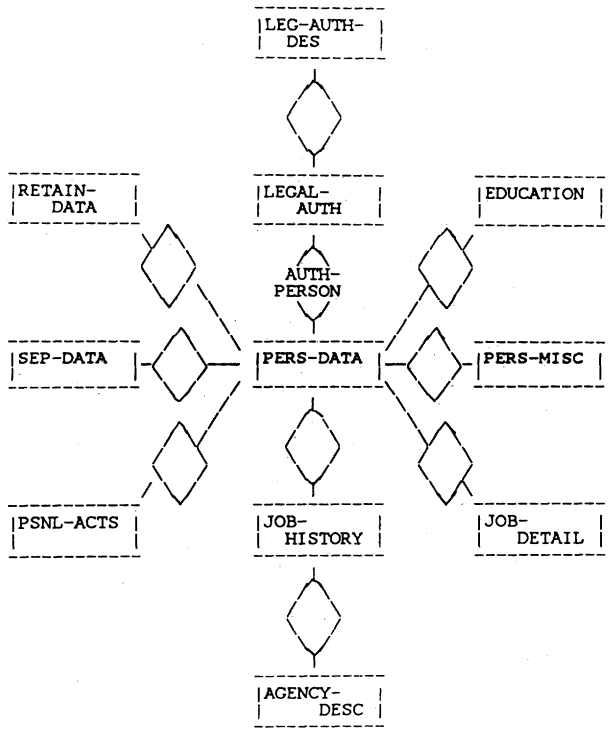


Fig. 4. Personnel database schema.

AGENCY_DESC(AGENCY, SUBELEMENT, NAME, DESCRIPTION)
RECORD SIZE = 33 BYTES

AUTH_PERS(SSN, CODE)
RECORD SIZE = 8 BYTES

LEGAL_AUTH(CODE, NAME)
RECORD SIZE = 22 BYTES

LEGAL_AUTH_DES(CODE, EXPLANATION)
RECORD SIZE = 45 BYTES

EDUCATION(SSN, EDUC_LEVEL, DEGREE_DATE, ACAD_DISC)
RECORD SIZE = 17 BYTES

JOB_DETAIL(SSN, PAY_DETERM, PAY_BASIS, PAY_PLAN, PAY_GRADE, CUR_GRDE_DT, CUR_OCC_DT, STEP, TENURE, PATCO, GS_EQUIV, BARG_UNIT, FLSA_EXEMPT, APPT_TYPE, UPDAT)
RECORD SIZE = 53 BYTES

JOB_HISTORY(SSN, SERV_DATE, AGENCY, SUBELEMENT, STATE, SALARY, STD_METRO, WORK_SCHED, POS_OCCUPIED)
RECORD SIZE = 25 BYTES

PERS_MISC(SSN, SUBMIT_OFF, SPEC_PRG_ID, RETIREMENT, ANNUITANT, FEGLI, PMIP, VET_PREF, VIET_VET)
RECORD SIZE = 17 BYTES

PERS_DATA(SSN, NAME, SEX, CITIZEN, BIRTH_DATE, HANDICAP, RACE, SERV_DATE, OCCUPATION, FUNCT_CLASS, LOCATION, STAT_CODE, MNGR_LEVEL, MNGRS_SSN, UPDAT)
RECORD SIZE = 85 BYTES

PSNL_ACTS(SSN, ACT_NATURE, PSNL_ACT_DT)
RECORD SIZE = 12 BYTES

RETAIN_DATA(SSN, RET_GRADE, RET_STEP, RET_PAY_PLAN)
RECORD SIZE = 15 BYTES

SEP_DATA(SSN, SEP_DATE)
RECORD SIZE = 9 BYTES

Fig. 5. Relational database schema.

these data, we extracted a personnel database schema. The schema, modeled in an entity-relationship diagram, is presented in Fig. 4. The third normal form relations are listed in Fig. 5 along with the record size, in bytes, of each.

Level 0 Indexes:

No indexes.

Level 1 Indexes:

Unique, clustered index on PERS_DATA(SSN)
Unique, clustered index on PERS_MISC(SSN)
Unique, clustered index on SEP_DATA(SSN)
Unique, clustered index on RETAIN_DATA(SSN)
Unique, clustered index on JOB_HISTORY(SSN, SERV_DATE)
Unique, clustered index on JOB_DETAIL(SSN)
Unique, clustered index on EDUCATION(SSN, EDUC_LEVEL)
Unique, clustered index on PSNL_ACTS(SSN)
Unique, clustered index on LEGAL_AUTH(CODE)
Unique, clustered index on LEG_AUTH_DES(CODE)
Unique, clustered index on AGENCY_DESC(AGENCY)
Unique, clustered index on AUTH_PERS(SSN, CODE)
Nonclustered index on JOB_HISTORY(AGENCY)

Level 2 Indexes:

All Level 1 indexes
Nonclustered index on RETAIN_DATA(RET_GRADE)
Nonclustered index on RETAIN_DATA(RET_PAY_PLAN)
Nonclustered index on PERS_DATA(RACE)
Nonclustered index on PERS_DATA(LOCATION)
Nonclustered index on JOB_DETAIL(PAY_GRADE)
Nonclustered index on AGENCY_DESC(SUBELEMENT)
Nonclustered index on EDUCATION(EDUC_LEVEL)
Nonclustered index on JOB_DETAIL(PATCO)
Nonclustered index on PERS_DATA(HANDICAP)
Nonclustered index on PERS_MISC(VET_PREF)
Nonclustered index on JOB_HISTORY(STATE)
Nonclustered index on EDUCATION(ACAD_DISC)

Fig. 6. Three levels of indexes.

TABLE I
DATABASE SIZES

Size	Number of Records in Single Relation
56 Mbyte	189 960
10 Mbyte	33 000
8 Mbyte	26 400
6 Mbyte	20 000
3.5 Mbyte	10 500

Because the use of indexes has a major impact on database system performance, three different levels of indexing were studied in the benchmark. Level 0 contains no indexes on any of the database attributes. Level 1 provides primary indexes on the database. Unique, clustered indexes are built on all primary keys. We also tested the database systems' ability to provide combined indexes. In level 1 we include three combined indexes. Level 2 includes the indexes from level 1 and adds additional secondary indexes on attributes that are used for retrieval in the benchmark query set described in the next section. Further indexing was not tested because of storage constraints. The specific indexes constructed on the personnel database are shown in Fig. 6.

Several databases of different sizes were constructed by eliminating a percentage of records from the database. The original personnel file data formed a single, unnormalized relation of size approximately 56 Mbytes and contained 189 960 records. By randomly eliminating records we were able to form several database sizes for our benchmark experiments, as shown in Table I.

Once the appropriate number of records was contained in the single relation, a program to divide the data into normalized relations in the conceptual schema was run to build the test database.

3) Benchmark Workload: We designed a number of queries to test the retrieval and update capabilities of the database systems. The queries are written in the SQL query language for the conventional database system, and in QUEL for the database machine system. The queries were carefully chosen so that equivalent versions could be written in both SQL and QUEL. For expository purposes we use the SQL formation of the queries in this section.

The queries are divided into several categories. For data retrieval we developed ten query sets and several special query sets to test specific database system features. Each query set contains from four to seven queries that vary in complexity based upon the number and type of conditions in the query predicate. We use the complexity classification developed by Cardenas [5] in our benchmark methodology. The range of complexities in each query set can be illustrated by examining query set 1.

Query q1-1 requires retrieval of all records in the database relation.

```
q1-1: select ssn, ret_grade, ret_pay_plan
      from retain_data
```

Query q1-2 contains a single *atomic condition* on the relation.

```
q1-2: select ssn, ret_grade, ret_pay_plan
      from retain_data
      where ret_pay_plan = 'WG'
```

Query q1-3 contains a single *item condition* as a disjunction (OR) of two atomic conditions.

```
q1-3: select ssn, ret_grade, ret_pay_plan
      from retain_data
      where ret_pay_plan = 'WG'
        or ret_pay_plan = 'GM'
```

Query q1-4 contains a single *record condition* as a conjunction (AND) of two item conditions.

```
q1-4: select ssn, ret_grade, ret_pay_plan
      from retain_data
      where (ret_pay_plan = 'WG' or ret_pay_
        plan = 'GM') and ret_grade > '08'
```

Query q1-5 adds another item condition to the record condition.

```
q1-5: select ssn, ret_grade, ret_pay_plan
      from retain_data
      where (ret_pay_plan = 'WG' or ret_pay_
        plan = 'GM')
        and ret_grade > '08'
        and ret_grade < '12'
```

Query q1-6 contains a single *query condition* as a disjunction (OR) of record conditions.

TABLE II
RECORD SIZE OF QUERY RESULTS

Query Set	Bytes Returned per Record
1	13
2	14
3	11
4	4
5	11
6	15
7	11
8	13
9	18
10	37

```
q1-6: select ssn, ret_grade, ret_pay_plan
      from retain_data
      where ((ret_pay_plan = 'WG' or ret_pay_
        plan = 'GM')
        and ret_grade > '08'
        and ret_grade < '12')
        or ret_grade = '07'
```

Each query set is designed so that the complexities of the query predicates increase with increased numbers in the set. The number of records retrieved changes from one query to the next depending on whether an OR condition (increase in records) or an AND condition (decrease in records) is added.

The query sets are designed as follows. Query sets 1–5 test single relation retrieval. The query sets range from retrieval on a small size relation (Query set 1), medium size relations (Query sets 2 and 3), and large size relations (Query sets 4 and 5). Query sets 6–10 test multiple relation retrieval. Query sets 6–8 are two relation queries. Query set 9 is a three relation query and Query set 10 is a four relation query. The basic query sets are numbered for reference as qx-y, where x is the query set number and y is the query number in the set (e.g., q3-4). The number of bytes in each record of the result for each query set is given in Table II. Note that this number reflects the actual data bytes in the result, not the total bytes transmitted to the host (i.e., no overhead bytes are counted).

To test the sorting facility we modified Query sets 1 through 4 by adding an "ORDER BY" clause on an appropriate attribute. These queries are designated with an "o" (e.g., qo2-3). To test aggregates we modified query sets 4 and 5 by adding the "COUNT" aggregate function in the output list. These queries are designated with an "x" (e.g., qx5-2). In addition to the retrieval queries, we tested the performance of several update commands. We designed representative insertions (e.g., qI-1), deletions (e.g., qD-1), and modifications (e.g., qM-2) on the personnel database. A complete listing of all test queries used in the benchmark is in [30].

Benchmark workloads are generated by defining *job scripts* for each benchmark run. The job script is a file of query numbers. For example, a job script for a benchmark run could be {q1-1, qx5-3, q9-4, qI-3, q4-5}. The job

script file is read into a program called "runner." This program is located in the host computer for the database machine and the minicomputer for the conventional database system. "Runner" executes the queries in job script order on the database system. Statistics are recorded on the query's performance in the system. Times are recorded to the sixtieth of a second to three decimal places for the conventional and database machine benchmarks. The statistics gathered are:

first	Time until first record is received at the host.
last	Time until last record is received at the host.
records	Number of records returned.

For the database machine architecture the parse times for the query in the host system are also recorded.

The runner algorithm is outlined as follows—actions in parentheses occur only in the database machine architecture:

Algorithm Runner:

```
Begin
   Read Job-script-file into query-array until EOF:
   Open database system;
   While (query-array not empty) do
      Read next query from query-array;
      Send query to the database management system;
      Parse query;
      (Transmit parsed query to the database machine;)
      Execute query;
      (Transmit results to host computer system;)
      Record time statistics on:
         (time-to-parse)
         time-to-first
         time-to-last
         record-count;
   End while;
   Close database system;
End of Algorithm Runner.
```

For each benchmark test we defined a job script and executed the "runner" on the different database systems. Statistics for each query in the script were collected. For multiuser tests and background load tests on the databases we ran multiple copies of runner simultaneously on separate job scripts and gathered the statistics on each.

4) Experimental Design: For the benchmark we selected response time as the primary performance measure. The "runner" program provides easily accessible timings from which several types of query response times can be calculated. Other potential performance measures, such as throughput and utilization, while useful, were not considered as important. Throughput in a single user environment can be obtained from response time statistics since "runner" initiates a query in the job script as soon as its predecessor completes. Throughput for the multiuser portion of the study can be estimated by timing the completion of all job scripts and dividing by the number of queries completed.

The primary statistics that we will use in our analysis are the time-to-first record value and the time-to-last record value. The time-to-first statistic measures the time from when "runner" calls the database system with the query until the first result record is received at the host system. The time-to-last statistic measures the total response time of the query from initiation until the last record is retrieved.

C. Results Analysis

The following benchmark analyses will present performance data for the benchmark tests that result from varying combinations of several experimental variables. We divide the discussion of the benchmark results into three sections; the single relation results, multiple relation results, and the multiple user results. As we compare the two database system architectures, we classify the experimental results as demonstrating either architectural similarities or architectural differences. A complete description of the benchmark experiments and the resulting data are found in [30].

III. Single Relation Results

We first measure the performance of queries that involve a single relation. The queries in query sets 1–5 are performed on the two database system architectures studied. By varying appropriate parameters we have obtained several interesting observations and comparisons. The following discussion highlights the principal results on single relation queries.

A. Response Time Analysis

One of the most important measures for a database system's performance is the query response time. In most online applications, response time is defined as the time from the submission of the query until the query result is available. Two different timings are important: the time until the first result record is obtained and time until the last record is available. In our benchmark study, both of these timings are analyzed. We will show that these timings are sensitive to the query complexity and type of system architecture.

We first show the dependency of query response times on the query complexity and the amount of data retrieved. It is shown in Fig. 7 that time-to-first increases as the complexity of the queries increases on the database machine. This observation does not differ significantly for different database sizes or line speeds. The conventional database system shows the same performance trend. An intuitive explanation for this result is that when the query complexity increases, fewer records are retrieved. If records are uniformly distributed in the relation, the time for reaching the first record should be greater the fewer records to be retrieved. The breaks in the curves correspond to the introduction of the "OR" operator in the query condition. The addition of "OR" has an effect of *increasing* the number of records retrieved. In order to isolate the "AND" effect we did not connect data points for the "OR" operations in the figures.

Fig. 8 demonstrates the varied effect of line speed on time-to-last with respect to query complexity. For 9600 bps lines, time-to-last decreases as the complexity of the

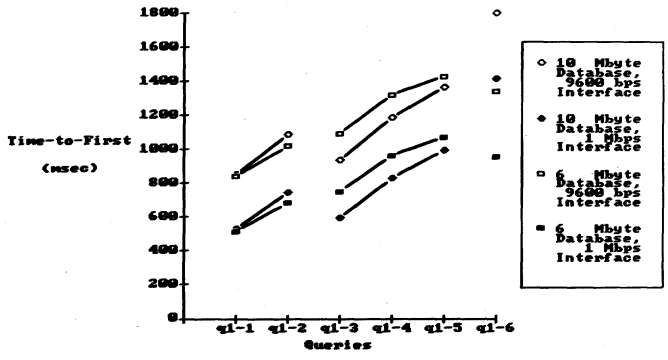


Fig. 7. Query complexity versus time-to-first (database machine, level 1 indexes).

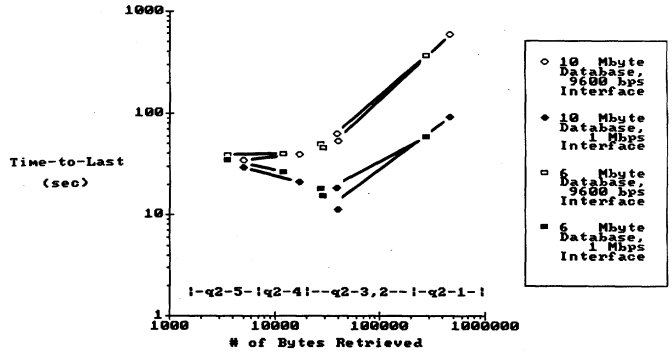


Fig. 9. Result size versus time-to-last (database machine, level 1 indexes).

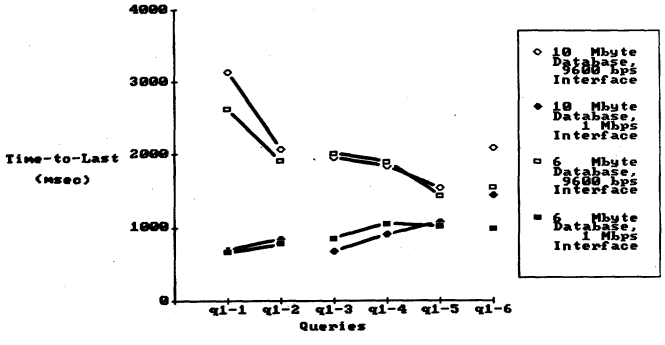


Fig. 8. Query complexity versus time-to-last (database machine, level 1 indexes).

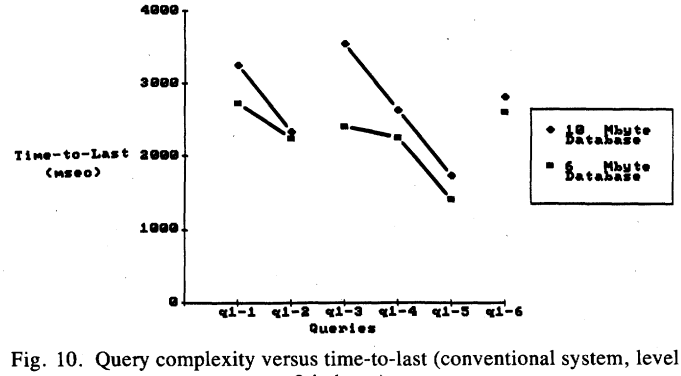


Fig. 10. Query complexity versus time-to-last (conventional system, level 2 indexes).

query increases. Here the factor which dominates time-to-last is the total number of records retrieved, which is inversely proportional to the query complexity. However, when the line is sped up to one Mbps, time-to-last increases as the complexity of the queries increase. Here the factor which dominates the response time is the initial query processing time, which is directly proportional to query complexity. Fig. 9 further reinforces this observation by plotting time-to-last against the number of bytes retrieved. When the number of bytes is small the response time is dominated by the initial query processing time. As more bytes are retrieved, the data transmission time dominates the response time and time-to-last increases, with the faster transmission line providing an increasingly greater benefit.

For the conventional database architecture, time-to-last decreases as query complexity increases as shown in Fig. 10. The factor which dominates the response time is the number of records retrieved from the database.

Figs. 11 and 12 show the time-to-first record and time-to-last record data, respectively, for the single relation queries on the 3.5 Mbyte database using level 2 indexes. The graphs plot the performance advantage of one architecture over the other as the number of bytes retrieved varies. The percentage of performance advantage is calculated as follows:

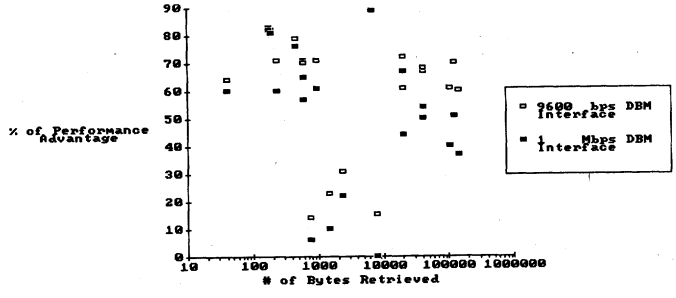


Fig. 11. Performance advantage in time-to-first of conventional system over database machine (single relation queries, 3.5 Mbyte database, level 2 indexes).

Fig. 11 shows that in every query the conventional architecture has better time-to-first performance than the database machine architecture. The performance advantage is greater for queries that retrieve less than 1000 bytes or more than 10 000 bytes. However, by increasing the line speed between the host and the database machine the magnitude of this advantage becomes less.

In Fig. 12, the time-to-last data show different areas of performance advantage. The conventional architecture has a performance advantage for most queries that return less than 1000 bytes. However, for queries that return more

$$\%\ \text{Performance Advantage of Conventional Architecture over DBM Architecture} = 100 * (\text{DBM Response Time} - \text{Conv. Response Time})/\text{DBM Response Time}$$

$$\%\ \text{Performance Advantage of DBM Architecture over Conventional Architecture} = 100 * (\text{Conv. Response Time} - \text{DBM Response Time})/\text{Conv. Response Time}$$

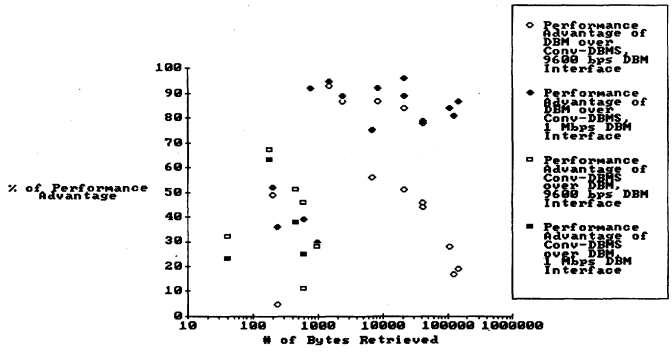


Fig. 12. Performance advantage in time-to-last between database machine and conventional system (single relation queries, 3.5 Mbyte database, level 2 indexes).

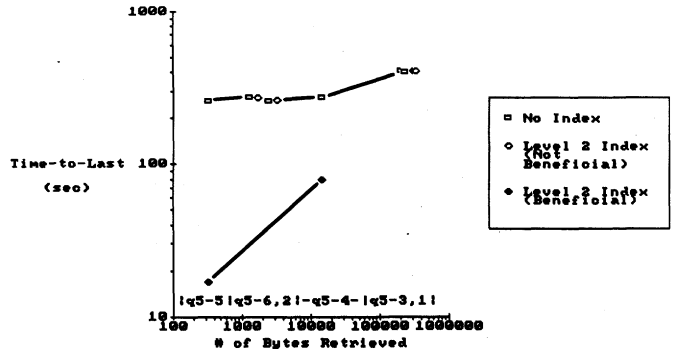


Fig. 13. Effect of indexing (conventional system, single relation query set 5, 6 Mbyte database).

than 1000 bytes the database machine demonstrates a clear superiority in performance. Here the advantages of a specialized hardware and software database system in a back-end machine are apparent. Note that when the communications line is speeded up, the magnitude of the conventional architecture advantage decreases and the magnitude of the database machine architecture advantage increases. For some queries, around the critical region of 1000 bytes, the database machine architecture becomes more advantageous than the conventional database architecture with the one Mbps line.

B. Index Effect

The benchmark tests on single relation indexing lead to the observation that the selection of secondary key indexes provide significant performance advantages for queries that contain selections on indexed attributes. Fig. 13 illustrates this point using query set 5 on a relation with secondary indexes on the conventional database system architecture. In queries q5.1, q5.2, and q5.3 the indexed attributes are not used in the query conditions and there is no significant performance difference between the levels of indexing. In queries q5.4 and q5.5, however, predicates are added that utilize the two indexed attributes. The improved performance of the queries with index level 2 is apparent. This improvement is lost, however, when the query is made more complex by adding another "OR" clause in query q5.6. Although the new condition uses an indexed attribute, the response time actually increases.

The performance of query execution does not always benefit from the use of indexes. This result is valid over both database system architectures. The benefit of indexes is influenced by the following parameters:

1) Hit Ratio: The hit ratio refers to the percentage of records retrieved from a relation. Indexes are useful for retrieving a relatively low percentage of the file because they can avoid the searching of the entire relation sequentially. To retrieve a large amount of data, however, the extra index accesses become a burden and sequential search could be more efficient. This effect is clearly shown in Fig. 13 where the difference between the times for the same query with and without indexes decreases as more records are retrieved.

2) Disjunctive Index: The use of disjunctive conditions (terms connected by "OR") in a query can sometimes make index processing more difficult. It is well known that if one of the terms in the disjunction is not indexed, then any other index in the condition is not useful. However, even if both terms in a disjunction use indexes, the performance still depends on the particular implementation. Many database systems are not able to process the "OR" index efficiently. As an example, performance penalties for disjunctive indexing can be observed for query q5.6 in Fig. 13. The disjunctive indexing is in fact worse than no indexing.

3) Range Index: The use of indexes to solve a range query may have a similar effect as the disjunctive index. This was also observed in the benchmark results [30].

Benchmark tests on indexing are found to have similar results over both architectures. Indexing is a performance enhancement method that is applicable to both conventional and database machine architectures.

C. Order of Query Execution

It is possible to improve the query processing performance by using specialized buffer management algorithms and large buffer size [23]. To determine if there is any ordering effect among queries accessing the same set of relations, two different mixes of queries were run. The first query mix ran similar queries in sequence. The response time can be reduced if the system is able to take advantage of the data and indexes kept in the buffer from previous queries. The second query mix ran queries in random order. The results showed that the response time to the last record retrieved is not reduced significantly by running similar queries together for any number of data retrieved. The difference in response times ranges from 366 ms for 3615 records to 217 ms for 33 000 records—insignificant amounts. We found that the same observation was valid for all query sets, all database sizes tested, and for both database system architectures. It appears that no interquery optimization is performed in either of the benchmarked systems.

D. Sorting and Aggregations

Next we examined the effect of sorting and aggregation on response time. The performance effect of adding sort-

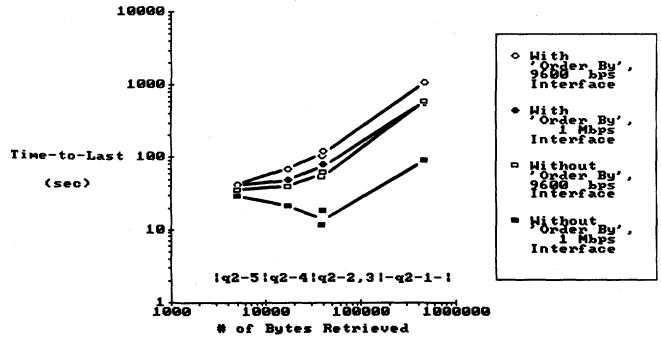


Fig. 14. Effect of sorting (database machine, 10 Mbyte database, level 1 indexes).

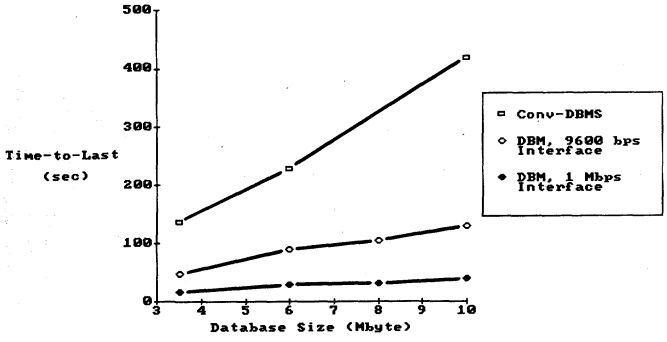


Fig. 15. Database size versus average time-to-last (single relation queries, level 1 indexes).

ing is similar for both architectures. Fig. 14 demonstrates the significant increase in response time when query results must be sorted. The comparison is made between similar query sets on the database machine. One set requires sorting the resulting records, the other does not. The response time to the last record retrieved is significantly greater for the set which requires sorting, regardless of the number of bytes retrieved. Since time-to-last is plotted on a logarithmic scale, it is evident that the magnitude of response time increase is dramatic. Since sorting involves a nonpolynomial complexity, this result is expected.

The inclusion of aggregate functions in query sets 4 and 5 was tested. The cost of performing the aggregate function was significant. We found significant decreases in response times for queries wherein the aggregations are based on groupings by indexed attributes.

E. Average Single Relation Performance

We tested the two database architectures on databases of various sizes up to 10 Mbytes. In Fig. 15 we show the average time-to-last values in seconds for single relation queries for each system over the tested database sizes. The average is taken over all queries tested. It is clearly seen that the database machine has the smaller average time-to-last values. Increasing the communications line speed provides significant decreases in average time-to-last values for the database machine architecture.

IV. Multiple Relation Results

The execution of query sets 6–10 on the two database system architectures provided response time results from which we can make the following observations and comparisons.

A. Response Time Analysis

The *join* operation is the most time consuming operation in relational database processing. A join is required when data from one relation are to be related to data in other relations—a common requirement in most database applications. Nonprocedural relational queries require the database system to establish access paths in the database using the file structures and indexes in the system. The goal of query optimization is to design access strategies that execute joins efficiently. The classic joining techniques of *nested loop* and *sort-merge* [2], [28] can be implemented on both database system architectures. The database machine architecture, however, has the potential for speeding up the join processing by using special hardware [26]. Join processing has significant potential for hardware speedup because of the repetitive nature of the simple comparisons and sorts required.

Our benchmark studies on multiple relation queries demonstrated the critical performance characteristics of the join operation. Joins were considerably more costly than single relation queries. Many of the more complex joins were not able to complete within a reasonable amount of time. Queries from query set 10, four relation queries, would not complete in either architecture under any level of indexes. Otherwise, the presence of indexes had a major effect on join performance. Most joins did not complete when no indexes were defined. Thus, at a minimum, level 1 indexes were required for satisfactory join performance.

The performance differences between the conventional architecture and the database machine showed similar results as in the single relation studies. The performance advantages of the architectures for the multiple relation queries are shown for the 6 Mbyte database with level 2 indexes. The time-to-first data is graphed in Fig. 16 and the time-to-last data is graphed in Fig. 17. We observe that the conventional architecture has a better overall time-to-first performance; while the database machine architecture demonstrates a significantly better time to-last performance. The increased communications line speed between the host and the database machine changes only the magnitude of the performance advantage. The conventional database architecture is not consistently better i.e., time-to-first performance. When less than 1000 bytes are retrieved the database machine shows better performance in most cases. The database machine demonstrates clear time-to-last performance superiority across the entire range of bytes retrieved by multiple relation queries.

B. Index Effect

We are particularly interested in the effect of indexing on the join performance. To process a join without using indexing, most systems use a nested loop technique. That is, for each record in the first relation, the entire second relation is accessed. The complexity for this type of algorithm is $N * N$, where N is the number of records (or

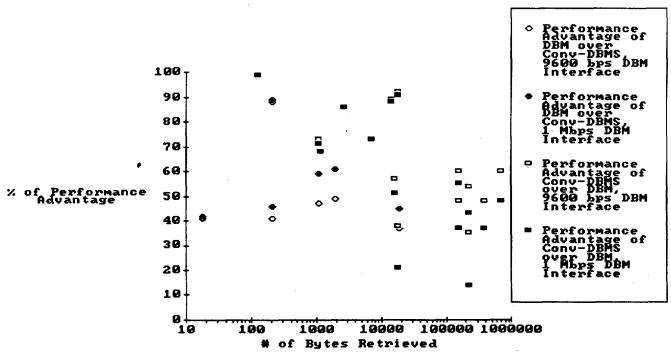


Fig. 16. Performance advantage in time-to-first between database machine and conventional system (multiple relation queries, 6 Mbyte database, level 2 indexes).

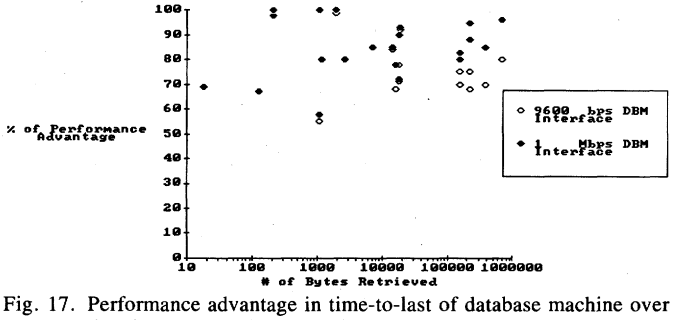


Fig. 17. Performance advantage in time-to-last of database machine over conventional system (multiple relation queries, 6 Mbyte database, level 2 indexes).

data pages) in each relation. Index support improves this algorithm by providing indexed access to the second relation. The complexity of the algorithm is, thus, reduced.

We found that indexing greatly improves the performance of joins on both architectures. Fig. 18 shows on the database machine that the benefit of using clustered indexing is significant when the indexes are involved in the join. For the query set considered, the collected response times for queries using indexes (level 2) are 3984, 3433, 4233, and 4984 ms versus 107 834, 199 066, 693 250, and 617 567 ms, respectively, when no indexes are used. These numbers clearly illustrate the order of performance gained by the use of indexes. The same order of beneficial effect was found in the conventional system tests. Notice that when no indexing is used, so much time is spent on processing that any increase in line speed produces a negligible benefit.

C. Join Methods

Considerable research on query optimization has concentrated on finding efficient methods of performing joins [8]. Since the join is the most complex and expensive operation, efficient join methods are critical for acceptable relational database system performance. Techniques involved in performing joins are largely independent of database system architectures. Differences in join performance result from the selection and quality of implementation of the join optimization algorithms.

In our benchmark tests we ran several experiments that demonstrate the above point. Semantically identical queries, with syntactic variations, were run on both architectures. Although the performance results differed, no architectural differences can be cited. Two examples of our testing are presented.

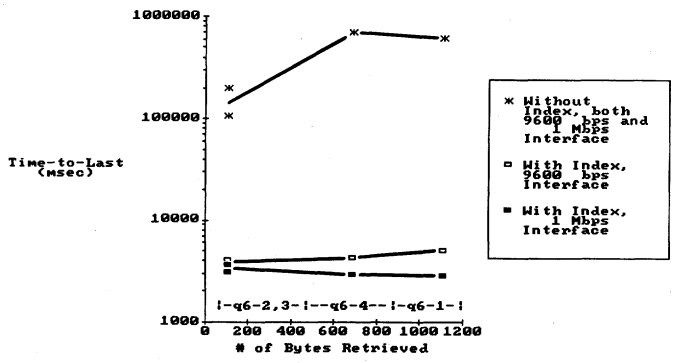


Fig. 18. Effect of indexing (database machine, 6 Mbyte database, level 1 indexes).

1) Arrangement of Query Conditions: We compared the performance of several join queries in which we changed the arrangement of the query conditions. For example:

```
sc1-1: select pers_data.ssn, educ_level, birth_data,
       vet_pref from pers_data, education, pers_
       misc where pers_data.ssn = education.ssn
       and education.ssn = pers_misc.ssn
       and pers_misc.ssn = 300378541

sc1-2: select pers_data.ssn, educ_level, birth_date,
       vet_pref from pers_data, education, pers_
       misc where pers_misc.ssn = 300378541
       and pers_data.ssn = education.ssn
       and education.ssn = pers_misc.ssn
```

For the conventional database system, the rearrangement of clauses in the query condition resulted in a difference in response time. On the 6 Mbyte database query sc1-1 ran in 1250 ms and sc1-2 ran in 680 ms. Similarly, on the 10 Mbyte database sc1-1 ran in 1410 ms and sc1-2 ran in 660 ms. We observe that the tested database system does not seem to recognize when identical clauses in a query are rearranged. Different access strategies are clearly produced.

On the database machine, however, the rearrangement of clauses in the query condition resulted in no significant change in response time. On the 6 Mbyte database query sc1-1 ran in 1384 ms and sc1-2 ran in 1100 ms.

2) Implicit Versus Explicit Query Conditions: With the same query we tested the performance when the join conditions were made explicit. For example:

```
sc2-1: select pers_data.ssn, educ_level, birth_date,
       vet_pref from pers_data, education, pers_
       misc where pers_data.ssn = education.ssn
       and education.ssn = pers_misc.ssn
       and pers_misc.ssn = 300378541
```

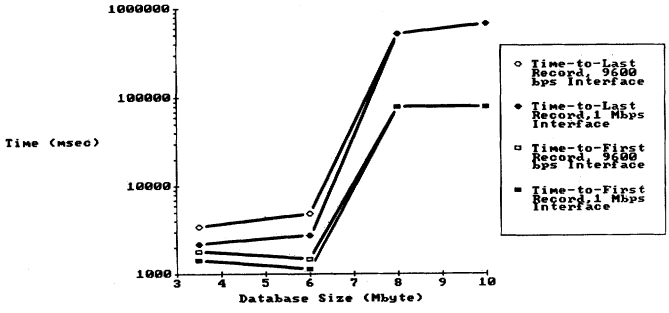


Fig. 19. Database size versus response time (database machine, multiple relation query q6-1, level 1 indexes).

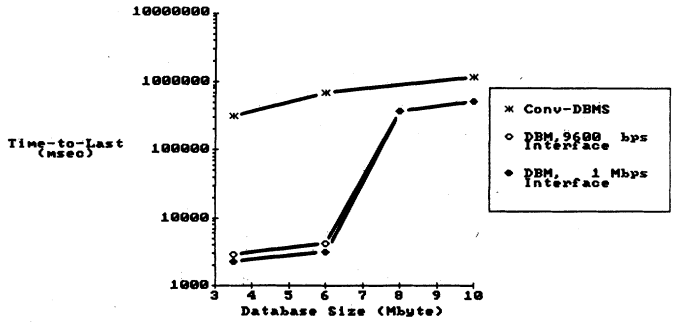


Fig. 20. Database size versus average time-to-last (multiple relation query set 6, level 1 indexes).

sc2-2: select pers_data.ssn, educ_level, birth_date, vet_pref from pers_data, education, pers_misc where pers_data.ssn = 300378541 and education.ssn = 300378541 and pers_misc.ssn = 300378541

We again observed that the conventional database system did not appear to execute the two queries in the same way, while the database machine did. For the 6 Mbyte database, query sc2-1 was run in 1420 ms by the conventional system and 1400 ms by the database machine. However, query sc2-2 required 547 050 ms on the conventional database system and only 1017 ms on the database machine. We do not attribute these performance differences to the system architectures, but to a system implementation that failed to generate the same access path.

D. Average Multiple Relation Query Performance

If the goal of improved performance for database machines is to be met, the join performance should be significantly better than for conventional database system architectures. In benchmarking multiple relation queries, we found a performance anomaly for the database machine as the database size increased. Fig. 19 shows that there is a significant jump for both time-to-first and time-to-last between database sizes 6 and 8 Mbytes. This figure shows the data for query q6-1; however, all multiple relation queries have a similar behavior. This could reflect the particular buffer management techniques being used by the system and a lack of sufficient buffer space. Such a point will be dependent on the workload variables and machine configuration. While we did not observe a similar anomaly in the conventional database system, we do not consider this result to be dependent upon the system architecture.

The average time-to-last performance for query set 6 with respect to database size is shown in Fig. 20. Even with the significant increase in time-to-last values, the database machine demonstrated consistently better performance.

V. Job Script Results

We tested the response time of job scripts under conditions of multiple database system users and nondatabase background workloads.

TABLE III
CONVENTIONAL SYSTEM DATA FOR MULTIUSER TESTS

	Response Time in Seconds	
No. of Users	Order 1	Order 2
1	8435.95	8339.69
2	15 696.23	15 029.96
3	23 626.54	23 639.76

TABLE IV
DATABASE MACHINE DATA FOR MULTIUSER TESTS

	Response Time in Seconds	
No. of Users	Order 1	Order 2
1	1024.25	1024.25
2	1732.78	1823.65
3	—	2721.61

A. Multiple User Tests

The tested database systems are designed to accommodate multiple, concurrent database users. The sharing of resources increases the response time of each user. The throughput of the total system should improve up to a saturation point. In addition, provisions for ensuring the integrity and consistency of the database in the presence of concurrent users add overhead to each user's performance. Response times should increase in proportion to the number of users in the database system. We found exactly this result in our benchmark. This performance trend was noted in both of the database system architectures.

We ran multiple user tests with 1, 2, and 3 users. We selected a random job script with 5 single relation queries and 4 multiple relation queries. The runs were made on the 6 Mbyte database with level 1 indexes. We varied the job script in two ways to test the effect of potential conflict and data locking. In order 1, the job scripts for each user were identical and in the same order. In order 2, the job scripts for each user were scrambled. The response times are recorded for the completion of all job scripts. The data are recorded in Tables III and IV.

The performance data from running the job scripts in

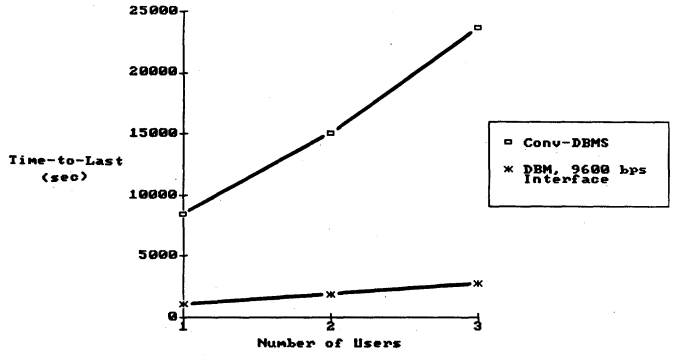


Fig. 21. Multiuser architecture comparison.

order 2 are graphed in Fig. 21. The following observations are made:

1) Rearranging the order of the queries in the job script had no effect in the conventional database system. On the database machine, however, in the three user run with the job scripts in the same order, one of the users became "locked" and did not complete. We are unable to explain this phenomenon.

2) The graph clearly shows the linear increase of the response times over the range of users in our study. This linear performance trend is similar in both the conventional system and database machine multiple user tests.

B. Background Load Tests

One of the clear architectural differences of the database machine architecture is the ability to offload database jobs from the host computer. Thus, the presence of nondatabase jobs should not effect the performance of jobs in the database machine and vice versa. In a conventional database system, however, all jobs reside in the host computer and must contend for common resources, such as processing time. Thus, we would expect to observe an increase in response time proportional to the total number of jobs, database and nondatabase, in the computer system. Our benchmark results verify this analysis.

To test the effect of a background load on database performance, we designed and programmed a sort routine on a medium size data file. We estimate that this job is 75 percent CPU intensive and 25 percent I/O intensive. Running alone on a dedicated VAX 11/750 the sort routine had an average response time of 1106.29 seconds.

We built two job scripts for this benchmark. Job script 1 contained all of the single relation query sets (1–5). Job script 2 contained the multiple relation query sets 6, 7, and 9. We ran each job script with 0, 1, 2, and 3 background jobs and measured the response times of the job scripts and the background jobs. The benchmarks were run on the 6 Mbyte database with level 1 indexes.

We measured the performance effects of running a background load in two ways. We first studied the effect of the background jobs on the database job scripts. We also studied the effect of the database load on the response times of the sort job.

The database machine tests clearly demonstrated that

TABLE V
CONVENTIONAL SYSTEM DATA FOR JOB SCRIPTS WITH BACKGROUND JOBS

No. of Background Jobs	Response Time in Seconds: Single Relation Job Script	Response Time in Seconds: Multiple Relation Job Script
0	6369.68	17 110.22
1	7353.20	19 231.50
2	8428.52	20 426.44
3	9351.03	21 592.07

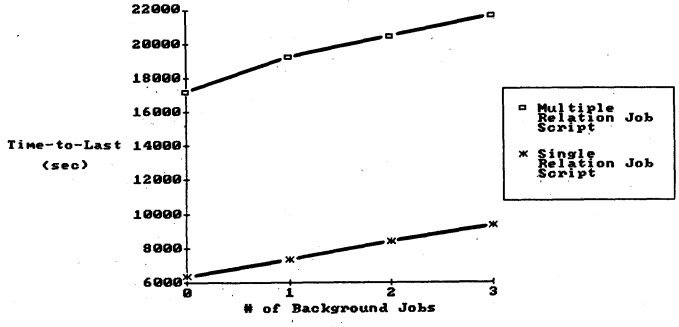


Fig. 22. Background load results (conventional system, 6 Mbyte database).

TABLE VI
CONVENTIONAL SYSTEM DATA FOR BACKGROUND JOBS

No. of Background Jobs	Response Time in Seconds: Single Relation Job Script	Response Time in Seconds: Multiple Relation Job Script
1	4243.75	2983.56
2	6024.80	4223.21
3	7839.60	5352.08

background jobs on the host computer system had negligible effect on the performance of the database job scripts. Response times did not vary significantly based upon the number of nondatabase jobs.

For the conventional database system Table V lists the response times of the two database job scripts as the background load was increased from 0 to 3 jobs. Fig. 22 graphs this data. The linear effect that adding more background jobs has on the performance of the database job scripts can clearly be seen.

For the reverse benchmark on the conventional database system, the average response times for each of the nondatabase sort jobs are presented in Table VI.

The response times of the background jobs increase linearly as the number of background jobs is increased. The most interesting observation is that the sort job runs significantly slower while the single relation job script is running than when the multiple relation job script is running. This is true even though the multiple relation job script requires nearly 2.5 times more time to complete. We feel that this is due to the number of individual queries in the single relation job script (30 versus 18 in the multiple re-

lation job script). The CPU contention caused by activating, parsing, and optimizing each single relation query affected the sort programs to a greater extent than the handling of the fewer multiple relation queries.

VI. Summary

The benchmark experiments reported in this paper allow us to make several significant performance comparisons between conventional database system architectures and database machine architectures. To summarize the experimental results, we classify our findings into performance differences or performance similarities.

A. Performance Differences

Based upon our benchmark results, we found three performance areas in which the database system architecture had a significant effect upon the capabilities and performance of the database system. These areas are as follows.

1) Response Time Analysis: In general, the conventional database system had better time-to-first record performance (Figs. 11 and 16), while the database machine provided better time-to-last record performance (Figs. 12 and 17). The performance of the database machine is influenced by the speed of the communications line between the host and the database machine. As part of our study we analyzed the effect of increasing the data rate from 9600 bps to one Mbps on all of our results. The relative comparisons of performance hold over the complete range of line speeds.

2) Background Load: One of the major advantages of the database machine architecture is that it relieves the front-end computer system of database processing. We found the effect of the load on the front-end system to have negligible performance effect on the database machine processing. On the other hand, increasing background loads on the conventional database system resulted in a significant increase in the job script response time.

3) Reverse Benchmark: A reverse benchmark measures the effect of database processing on the performance of nondatabase jobs in the host computer system. In Table VI we can see this effect on the sort program used in our benchmark as background load. The performance of the sort job running in the front-end system of the database machine was not noticeably affected under the same database loads.

In addition to the performance differences that we found, our experiences in executing the benchmarks lead us to make observations on two nonperformance differences between the database system architectures.

4) Set-Up and Loading: The database system loading and set-up procedures varied between the two systems. The loading of the database machine is done through the front-end computer and therefore is sensitive to the channel line speed. Using the 9600 bps communications line to load the 56 Mbyte database required over 26 hours. A faster communication line is needed for more reasonable database load times.

5) Reliability: The basic architectural issue for reliability is the presence of more than one computer system in a database machine architecture. In the common configuration of one host and one database machine, both systems must function correctly for the database machine to be operative. This effectively decreases the reliability of a database machine architecture. For example, if the probability of failure of the host is p and the probability of failure of the back-end system is r, then the probability of failure for the front-end/back-end system would be

$$1 - (1 - p) * (1 - r) = p + r - p * r.$$

Such an increase in failure probability would be significant in an environment that requires high reliability of its database system. To improve upon this reliability, the database machine can operate as a database server to two or more hosts. The multihost configuration is more reliable than the single-host since if one host failed, the database machine can still be accessed by another host. The only way for the entire system to be inoperative is for all hosts to fail at the same time, or for the database machine to fail. The probability of failure for the multihost system, if p_i is the probability of failure for host i, $i = 1, \cdots, n$, would be:

$$1 - \left(1 - \prod_{i=1}^{n} p_i\right) * (1 - r) = \prod_{i=1}^{n} p_i + r - \prod_{i=1}^{n} p_i * r.$$

B. Performance Similarities

It is important to note that a majority of our benchmark studies found similar performance patterns over the two architectures. The following general observations can be made for each of the benchmarked systems.

1) Query Type: Each system supports all the query types that we tested; single relation queries, multiple relation queries, updates, sort queries, and aggregation queries.

2) Query Complexity: For the conventional database system and the database machine, query complexity directly affected the number of records retrieved for single and multiple relation queries. Increasingly complex queries retrieved fewer records because of the more selective conditions. The time-to-first is found to increase as the query complexity increases. The overhead is greater and because fewer records are retrieved, the system takes longer to find the first record.

3) Indexing: The use of indexing is shown to be valuable for queries that used the indexes effectively. Cases are recognized, however, where indexing does not improve system performance and, in some cases, actually decrease performance. These observations are made across both systems.

4) Join Methods: While several interesting observations can be made from the special case studies of rearrangement of query conditions and implicit versus explicit

join conditions, we judge the performance to be a result more of the particular system implementation than of the system architecture. Therefore, no major architectural differences are reported.

5) Number of Database Users: Both architectures can support multiple database users. The performance of both systems degrades linearly with the number of users in the system (Fig. 21). Thus no architectural differences are observed.

6) Updates: Similar performance is shown on the conventional database system and the database machine for the tested update operations [30].

In addition to the above architectural comparisons, the benchmark study reported in this paper provides several important results. Executing the benchmarks on an actual personnel database gives a realistic examination of the database systems in an application environment. The performance analysis extends database benchmarking methodologies by including consideration of the time-to-first record performance in addition to the time-to-last record performance of queries. The communications line speed between the host computer and the database machine is found to have a significant performance impact. A thorough sensitivity study of this impact was performed.

The database machine architecture has further potential to increase the performance of database system processing. Current research and development is being done to find hardware support for expensive database operations, such as joins, aggregations, and sorting [15]. For example, studies have shown that the accelerator for the IDM database machine improves performance across many query types [4], [14].

Appendix A
Analysis of Line Speed Effects in the Database Machine Architecture

The delay caused by the 9600 bps communications line between the database machine and the host computer had a significant influence on the response time of the benchmarked queries. Intuitively, by increasing the speed of the line, the effects of transmission can be minimized so that the database machine operations are the dominating factors in the timings. We studied the effects of increasing the line speed on all of our benchmark data. Following is a detailed description of our analysis.

A.1 Sequential Versus Concurrent Operations

When processing queries in a database machine architecture, there are operations that must be processed sequentially, while others can be processed concurrently. Sequential operations consist of parsing the query request, transmission of query codes from the host to the database machine, execution of the code, and retrieval of the first block of data from the database for transmission back to the host. In the database machine benchmarked, the host asks the database machine to notify it when there are results ready to transmit. Once this call is received by the host, the host initiates the read request. As soon as the database machine receives the read request, it transmits all of the results available at that time, up to 2048 bytes, unless the host requests a smaller transmission packet. The size of the data sent to the host is the smaller of the two values; the data ready to transmit or the requested packet size [16], [27]. Data transmission can start as soon as the first block of results is available, and concurrent operations between the communications line and the database machine also begin at this time. Thus, concurrency between the line and the database machine can potentially dominate the entire query processing time for any query that retrieves more than a few blocks of data.

Time-to-first data consist of only the timings from the initial sequential operations, while time-to-last data consist of the timings from both the sequential and concurrent operations. Timing for transmissions that are performed sequentially can always be improved by increasing line speed, while timing for transmissions that are performed concurrently with the database machine operations can only be improved until the database machine's utilization reaches 100 percent. This is discussed in more detail later.

A.2 Query Response Times with Increased Line Speed

The query processing time improvements as the line speed increases can be calculated if we know the amount of data transmitted through the communications line for a particular query. Knowing the query, its results retrieved from the database, and the communications protocol, the exact amount of data exchanged between the host and the database machine can be calculated. For time-to-first, only the first result data packet sent by the database machine is included in the calculation. However, all result transmission data packets are included in the calculation for time-to-last.

During the processing of a query, the line is idling mainly during the parsing of the query by the host and the execution of the query in the IDM to retrieve the first block of results. All other idling times are caused by the communications protocol. Since the transmission line is only idling during either host or database machine processing, and this processing is invariant with respect to line speed, the idle time remains the same regardless of the speed of the line.

The transmission line idle time can be easily calculated from the query processing time collected at 9600 bps, provided that the amount of data exchanged between the host and the database machine is known. The idle time can be computed by $I = Q_{9600} - T_{9600}$, where Q is the query processing time and T is the transmission time. The query processing time at any line speed x can then be calculated by $Q_x = I + T_x$. For example, query q1-1 took 2617 ms to complete with a 9600 bps line on the 6 Mbyte database, and approximately 19 000 bits are transmitted. The transmission line idle time is then 2617 −

(19 000/9.6) = 638 ms. If the line speed is increased to 1 Mbps, the estimated query processing time would be 638 + (19 000/1000) = 657 ms, assuming that the database machine can accommodate this increased speed.

A.3 Effect on Time-to-First Data

The differences in time-to-first performance between the conventional database system architecture and the database machine architecture are primarily caused by the need to send the first data packet across the communications line in the latter case. The major variables affecting this time are the size of the first block of data retrieved from the database, the size of the first result data packet returned to the host, and the speed of the communications line. By minimizing the effects of these factors, the time-to-first values between the architectures are similar.

To minimize the effects of the amount of data that must be retrieved for the first block and the first transmission packet size, the block and packet sizes should be as small as possible. The block size is determined during database machine installation and has a minimum size of 256 bytes. The host specifies the maximum number of bytes to transmit from the database machine to the host in the result request [16]. In our benchmark, 256 byte data packets were requested. Although small block and packet sizes would give the best time-to-first data, large sizes would give the best time-to-last data due to the overhead incurred with a larger number of packets. Thus, the selection of the block and packet sizes are important design considerations for database machine performance.

To minimize the effects of transmission speed, the communications line speed between the host and the database machine must be increased. Time-to-first data can always be improved by increasing line speed because all operations timed here are sequentially executed. However, the database machine's time-to-first data may not necessarily become better than the conventional database's data by increasing the line speed. For example, in the comparison of time-to-first values between the architectures, as shown in Fig. 11, as the transmission speed is increased from 9600 bps to 1 Mbps, the conventional database still outperforms the database machine in all of the cases.

A.4 Effect on Time-to-Last Data

Time-to-last data for any query that retrieved more than a few data packets are dominated by concurrent operations between the result transmission and the database machine operations. If the database machine is not utilized to capacity, then the database machine idles while waiting on the communications line to empty the buffer. If the communications line is not utilized to capacity, then the line idles while waiting for traffic. Thus, there exists a point where the database machine's processing capacity and the communications line speed are at an equilibrium, where neither one is waiting on the other. This is a system

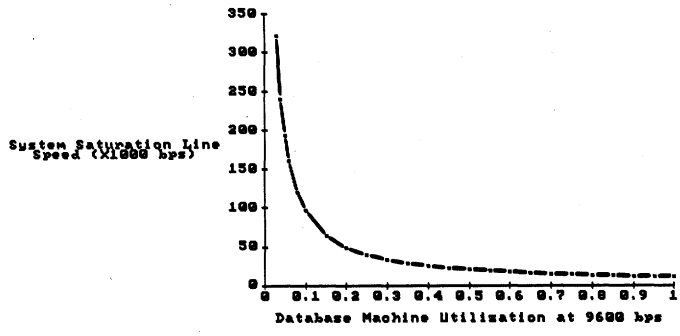


Fig. 23. System saturation line speed versus database machine utilization.

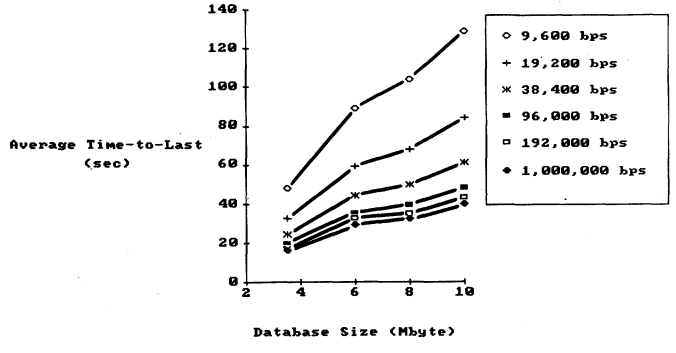


Fig. 24. Effects of line speed with respect to database size (database machine).

equilibrium point, and the transmission line speed at this point will be referred to as the *system equilibrium line speed*.

The speed at which the system benchmark was performed, 9600 bps, is the basis used here to illustrate the relationship between the equilibrium line speed and the database machine's utilization. Since 9600 bps is a relatively slow line speed, we assume that the database machine is processing at a rate such that it is not being used to full capacity. The line speed at which the database machine is utilized at 100 percent is the system equilibrium line speed. This speed can be calculated by dividing 9600 bps by the assumed database machine's utilization with a 9600 bps line (i.e., 9600 bps/u_{DBM}). For example, if the database machine is utilized 50 percent with a 9600 bps line, then by doubling the line speed to 19 200 bps, the database machine would theoretically by utilized at 100 percent. The relationship between the system equilibrium line speed and database machine's assumed utilization at 9600 bps is plotted in Fig. 23. Note that even if the database machine is utilized as little as 10 percent of the time at 9600 bps, only a 96 000 bps line is needed to reach system equilibrium.

Time-to-last data are minimally affected by increasing line speed if the database machine and the line are at equilibrium. Otherwise, the time-to-last data can be significantly improved. Fig. 24 shows the average query processing times over single relation queries as the line speed

is increased. The speeds plotted correspond to the system saturation line speeds assuming 100, 50, 25, 10, 5, and 0.96 percent database machine utilization at 9600 bps. For the majority of the time-to-last results from the benchmark, the database machine architecture is superior to the conventional database architecture. Thus, by increasing the line speed, the magnitude of the superiority would be increased.

A.5 Summary

All of the trends and the majority of the relative comparisons between the conventional database architecture and the database machine architecture shown in this paper remain valid in the presence of increased transmission line speed. Only the magnitude of the differences will increase, further reinforcing the advantages of the database machine architecture. We use the analyses described here to study the effects of increased line speed throughout the paper. The database machine benchmark data is presented as measured with a 9600 bps line and as estimated with a one Mbps line. The effects of the line speed increase are discussed in the body of the paper.

ACKNOWLEDGMENT

At the National Bureau of Standards (NBS) we worked with M. Skall and J. Collica in setting up the benchmark and evaluating the results. The research assistants who ran the benchmarks were R. Berkow on the IDM-500 and S. Baur on ORACLE. The System Development Corporation (SDC) of Santa Monica, CA, served as a subcontractor on the project, assisting in the design of the database and the runner program. From SDC, we acknowledge the contributions of M. Edwards, E. Lund, and R. MacGregor to this work. Finally we wish to thank the vendors of the database systems used in this project. They cooperated fully in the installation and operation of their systems and in the review of the benchmark results. We also acknowledge the helpful comments received from the editor and referees in the preparation of this paper.

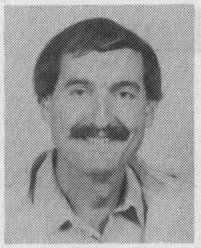
REFERENCES

[1] H. Bitton, D. Dewitt, and C. Turbyfill, "Benchmarking database systems: A systematic approach," Dep. Comput. Sci., Univ. Wisconsin, Tech. Rep. 526, Jan. 1983.
[2] M. Blasgen and K. Eswaran, "Storage and access in relational databases," *IBM Syst. J.*, vol. 16, no. 4, 1977.
[3] R. Bogdanowicz, M. Crocker, D. Hsaio, C. Ryder, V. Stone, and P. Strawser, "Experiments in benchmarking relational database machines," in *Proc. Third Int. Workshop Database Machines*, Munich, West Germany, Sept. 1983.
[4] H. Boral and D. Dewitt, "A methodology for database system performance evaluation," in *Proc. SIGMOD Conf.*, Boston, 1984.
[5] A. Cardenas, "Evaluation and selection of file organization—A model and system," *Commun. ACM*, vol. 16, no. 9, Sept. 1973.
[6] *Computer* (*Special Issue on Database Machines*), vol. 12, no. 3, Mar. 1979.
[7] *Database Eng.* (*Special Issue on Database Machines*), vol. 4, no. 2, Dec. 1981.
[8] *Database Eng.* (*Special Issue on Query Optimization*), vol. 5, no. 3, Sept. 1982.
[9] *Database Eng.* (*Special Issue on DBMS Performance*), vol. 8, no. 1, Mar. 1985.
[10] C. Date, *An Introduction to Database Systems*, vol. II. Reading, MA: Addison-Wesley, 1983.
[11] S. Demurjian *et al.*, "Performance evaluation of a database system in multiple backend configurations," in *Proc. 1985 Int. Workshop Database Machines*, 1985.
[12] D. DeWitt, "Benchmarking database systems: Past efforts and future directions," *Database Eng.*, vol. 8, no. 1, Mar. 1985.
[13] P. Hawthorn and D. DeWitt, "Performing analysis of alternative database machine architectures," *IEEE Trans. Software Eng.*, vol. SE-8, Jan. 1982.
[14] P. Hawthorne, "Variations on a benchmark," *Database Eng.*, vol. 8, no. 1, Mar. 1985.
[15] D. Hsiao, Ed., *Advanced Database Machine Architecture*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[16] *IDM-500 Installation and Operation Manual*, 201-0500, Mar. 1982; also *IDM-500 Software Reference Manual*, Version 1.4, 202-0500, 1982.
[17] D. Luo, D. Xia, and S. Yao, "Data language requirements for a database machine," in *Proc. NCC*, 1982.
[18] T. Makimo, *et al.*, "An evaluation of a centralized database subsystem," *J. Inform. Processing*, vol. 5, no. 1, Mar. 1982.
[19] F. Maryanski, "Backend database systems," *ACM Comput. Surveys*, vol. 12, no. 1, Mar. 1980.
[20] *ORACLE Database System Manuals*, Version 3.1, 1983.
[21] *The Database Machine*, Software AG, Product Announcement, 1980.
[22] M. Stonebraker, "Operating system support for database management," *Commun. ACM*, vol. 24, no. 7, July 1981.
[23] M. Stonebraker, J. Woodfill, J. Ranstrom, M. Murphey, M. Meyer, and E. Allman, "Performance enhancements to a relational database system," *ACM Trans. Database Syst.*, vol. 8, no. 2, June 1983.
[24] M. Stonebraker, "Tips on benchmarking data base systems," *Database Eng.*, vol. 8, no. 1, Mar. 1985.
[25] M. Templeton, I. Kameny, D. Kogan, E. Lund, and D. Brill, "Evaluation of ten data management systems," SDC Document TM-7817/000/00, 1982.
[26] F. Tong and S. B. Yao, "Design of a two-dimensional join processor array," in *Proc. Sixth Workshop Computer Architecture for Non-Numeric Processing*, Hyeres, France, 1981.
[27] M. Ubell, personal communication, Mar. 1986.
[28] S. B. Yao, "Optimization of query evaluation algorithms," *ACM Trans. Database Syst.*, vol. 4, no. 2, June 1979.
[29] S. B. Yao and A. R. Hevner, *A Guide to Performance Evaluation of Database Systems*, Nat. Bureau Standards Special Publ. 500-118, D. Benigni, Ed., U.S. Dep. Commerce, Dec. 1984, 48 pp.
[30] ——, *Benchmark Analysis of Database Architectures: A Case Study*, Nat. Bureau Standards Special Publ. 500-132, D. Benigni, Ed., U.S. Dep. Commerce, Oct. 1985, 187 pp.

S. Bing Yao received the Ph.D. degree from the University of Michigan, Ann Arbor, in 1974.

He is currently an Associate Professor of Business Management and Computer Science and is Director of the Database Systems Research Group at the University of Maryland, College Park. He is involved in the design and implementation of several database management systems and tools including XQL, a relational database system. He has authored over 50 technical papers on database storage structures, database performance optimization, database design, distributed database systems, database machines, and office automation. He is Editor of *Data Base Design Techniques* (Springer-Verlag, 1982) and *Principles of Data Base Design* (Prentice-Hall, 1982). He is founder of Software Systems Technology, a computer firm specializing in database systems and machines. His international consulting clients include Bell Laboratories, IBM, and the Swedish Technical Development Board.

Dr. Yao is an Editor of the *ACM Transactions on Office Information Systems*, is on the Editorial Board of the *ACM Transactions on Database Systems*, was Program Chairman of the Fourth International Conference on Very Large Data Bases and the NYU Symposium on Database Design, and Conference Chairman of the Ninth International Conference on Very Large Data Bases.



Alan R. Hevner (M'80) received the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, in 1979.

He is an Associate Professor of Information Systems in the College of Business and Management at the University of Maryland at College Park. His research interests include distributed database systems, database system performance evaluation, information systems analysis and design, and office information systems. He is coauthor, with Harlan Mills and Rick Linger, of the book *Principles of Information Systems Analysis and Design* (Academic Press, 1986).

Dr. Hevner is a member of the Association for Computing Machinery and the IEEE Computer Society.

Hélène Young-Myers received the M.S. degree in computer science from Virginia Polytechnic Institute, Blacksburg.

She is currently a Ph.D. candidate in information systems at the University of Maryland, College Park. She has worked for Bell Aerospace, InterNorth, and Northern Natural Gas. Her current research interests are in database security and operations reseach applications in database design.

Ms. Young-Myers is a member of the Association for Computing Machinery, the IEEE Computer Society, the Operations Research Society of America, and The Institute of Management Sciences.