

MARCH 1985 VOL. 8 NO. 1

a quarterly bulletin
of the IEEE computer society
technical committee
on

Database Engineering

Contents

Letter from the Editor.....	1
Benchmarking Database Systems: Past Efforts and Future Directions <i>D.J. DeWitt</i>	2
Tips on Benchmarking Data Base Systems..... <i>M. Stonebraker</i>	10
Variations on a Benchmark..... <i>P. Hawthorn</i>	19
Benchmarking Database Systems in Multiple Backend Configurations <i>S. Demurjian and D.K. Hsiao</i>	29
Transaction Acceleration..... <i>T. Chou and J. Gray</i>	40
Transaction Oriented Performance Analysis of Database Machines..... <i>M. Eich</i>	53

Special Issue on DBMS Performance

**Chairperson, Technical Committee
on Database Engineering**

Prof. Gio Wiederhold
Medicine and Computer Science
Stanford University
Stanford, CA 94305
(415) 497-0685
ARPANET: Wiederhold@SRI-AI

**Editor-in-Chief,
Database Engineering**

Dr. David Reiner
Computer Corporation of America
Four Cambridge Center
Cambridge, MA 02142
(617) 492-8860
ARPANET: Reiner@CCA
UUCP: decvax!cca!reiner

**Associate Editors,
Database Engineering**

Dr. Haran Boral
Microelectronics and Computer
Technology Corporation (MCC)
9430 Research Blvd.
Austin, TX 78759
(512) 834-3469

Prof. Fred Lochovsky
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S1A1
(416) 978-7441

Dr. C. Mohan
IBM Research Laboratory
K55-281
5600 Cottle Road
San Jose, CA 95193
(408) 256-6251

Prof. Yannis Vassiliou
Graduate School of
Business Administration
New York University
90 Trinity Place
New York, NY
(212) 598-7536

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Database Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Letter from the Editor

The proliferation of Relational Database Management Systems offerings in the marketplace in the past few years has forced users of such systems to seek for means to evaluate their performance. For obvious reasons, and despite its many shortcomings, the technique of benchmarking has become the basis for all such evaluations. A number of such benchmarks have been developed and others are most likely under development (for examples see references in the papers).

For this issue of DBE we asked several of the "known" researchers in the area of performance evaluation of DBMSs to submit papers that summarize their present work in the area. The result is a collection of six papers. Roughly speaking, three general themes are addressed in the papers: a reflection on past work leading to a list of "lessons learned" (the papers by DeWitt and Stonebraker), further use of the existing benchmarks to examine the performance of systems (papers by Hawthorn and Demurjian & Hsiao) and, discussions of new metrics and techniques for evaluating the performance of DBMSs (papers by DeWitt, Chou & Gray, and Eich).

Several other papers were submitted to this issue but were not included because they dealt with the evaluation of the performance of some specific component of a DBMS (such as scheduler in a distributed system or the concurrency control mechanism). I was hoping that we'd be able to have a paper detailing the results of the "Great French Database Machine Competition" (which pits five or six database machines against one another using the Wisconsin benchmark (see DeWitt's paper in this issue)), but unfortunately the results are still not available due to the rescheduling of the "contest" date.

I wish to thank the contributors to this issue for their excellent papers and for keeping to themselves any gripes about the unrealistic deadlines I set.

Haran Boral
February 1985

Upcoming Issues

6/85 Concurrency Control and Recovery in DBMS's (Mohan)
9/85 Natural Languages and Databases (Vassiliou)
12/85 Object Oriented Systems and DBMS's (Lochovsky)

Benchmarking Database Systems: Past Efforts and Future Directions

David J. DeWitt
Computer Sciences Department
University of Wisconsin

1. Introduction

During the past two years we have developed a strategy for benchmarking database management systems and machines [BITT83, BORA84]. At the present time our set of single user benchmarks [BITT83] has been used by over 30 vendors and customers of relational products. For better or worse, it has emerged as the standard set of single user benchmarks. To date, this benchmark has been applied to Unify, Oracle, INGRES, SQL/DS, RDB, to the IDM 500 and DIRECT database machines, and a number of other unannounced software and hardware products.

In Section 2, we review our present single user and multiuser benchmarking methodology. Section 3 contains a number of open research areas that we are currently exploring. Our conclusions are presented in Section 4.

Those of you who were hoping for a new set of numbers will be disappointed with this paper. While we had hoped to do exactly that, we have not yet succeeded in getting a copy of, or access to, each of the key products. While certain vendors have been quite helpful, others have remained reluctant to cooperate. Evidently, bad numbers translate into poor profits. Based on the number of requests we receive each month for updated numbers, there is clearly a market for this information. Perhaps what is needed is an EPA Testing Lab or Consumers Union for benchmark numbers.

.. Overview of the Wisconsin Benchmark Methodology

In this section, we present an overview of the methodology that we have developed for benchmarking relational database systems and machines. First we describe the synthetic database that is used as the basis of our all tests. Second, our strategy for constructing a single user benchmark is presented. Finally, we describe our approach for multiuser tests of a database system. For more details the reader is encouraged to examine [BITT83] and [BORA84].

2.1. Synthetic Database Design

A key component of our benchmarking methodology is a synthetic database. Such databases ([BITT83] and [BODG83]) can be easily generated by programs and have a number of advantages over "real" databases. First, a synthetic database makes it quite simple to specify a wide range of retrieval and/or update queries and to control the sizes of the relations resulting from these queries. With a "real" database, getting a selection query that retrieves precisely 10% or 50% of the tuples in a relation is difficult and sometimes impossible. Furthermore, specifying such a query requires one to either know a good deal about the semantics of the data in the database base or execute a bunch of trial queries. A second advantage of a synthetic database is that the distribution of attribute values is under the control of the program generating the database. With "real" data, one has to deal with very large amounts of data before it can be safely assumed that the data values are randomly distributed. In addition, while we have, to date, only experimented with uniform distributions of attribute values, experimenting with non-uniform distributions would be straightforward.

The benchmark database is designed so that a naive user can quickly understand the structure of the relations and the distribution of each attribute value. The attributes of each relation have distributions of values that can be used for partitioning aggregates, controlling selectivity factors in selections and joins, and varying the number of duplicate tuples created by a projection. It is also straightforward to build an index (primary or secondary) on some of the attributes, and to reorganize a relation so that it is clustered with respect to an index.

There are four "basic" relations in the database. We refer to them as "thoustup", "twohoustup", "fivethoustup", and "tenthoustup" as they contain, respectively, contain 1000, 2000, 5000, and 10000 tuples. A fragment of the thoustup relation is shown in Figure 1. All the tuples are 182 bytes long. Thus, the four relations occupy approximately 4 megabytes of disk storage. However, in order to build queries that operate on more than one operand relation, we often generate two or more relations of the same size. The attributes are either integer numbers (between 0 and 9999), or character strings (of length 52 characters). The first attribute ("unique1") is always an integer number that assumes unique values throughout the relation. We have made the simplest possible choice for the values of "unique1". For example, for the thoustup relation, unique1 assumes the values 0, 1, ... 999. For the relations with 10,000 tuples, the values of "unique1" are 0, 1, ..., 9999. The second attribute "unique2" has the same range of values as "unique1". Thus both "unique1" and "unique2" are key attributes. However, while we have used a random number generator to scramble the values of "unique1" and "unique2", the attribute "unique2" is often used as a sort key. When relations are sorted, they are sorted with respect to this attribute. When we need to build a clustered index, again it is an index on "unique2".

A Fragment of the Thoustup Relation
(some attributes have also been omitted)

unique1	unique2	two	ten	hundred	thousand
378	0	1	3	13	615
816	1	0	4	4	695
673	2	0	6	26	962
910	3	0	2	52	313
180	4	0	0	20	74
879	5	1	9	29	447
557	6	1	7	47	847
916	7	0	4	54	249
73	8	0	6	26	455
101	9	0	2	62	657

Figure 1

As an example of how this database can be used, we may execute the following INGRES query to observe the effect of a primary index on a selection that retrieves 10% of the twohoustup relation:

```
range of t is twohoustup
retrieve (t.all) where t.unique2 < 200
```

After the "unique1" and "unique2" attributes come a set of integer-valued attributes that assume non-unique values. The main purpose of these attributes is to provide a systematic way of modeling a wide range of selectivity factors. Each attribute is named after the range of values the attribute assumes. That is, the "two", "ten", "twenty", "hundred", ..., "tenthous" attributes assume, respectively, values in the ranges (0,1), (0.1,....9), (0.1,....19), (0.1,....99), ... ,(0.1,....9999). For instance, each relation has a "hundred" attribute which has a uniform distribution of the values 0 through 99. Depending on the number of tuples in a relation, the attribute can be used to control the percentage of tuples that will be duplicates in a projection or the percentage of tuples that will be selected in a selection or join query. For example, in the twohoustup relation, the "hundred" attribute can be used for projecting into a single attribute relation where 95% of the tuples are eliminated as duplicates (since only 100 values are distinct among the 2000 attribute values). The INGRES statement for this query would be:

```
range of t is twohoustup
retrieve (t.hundred)
```

The same "hundred" attribute can be used for creating 100 partitions in aggregate function queries. For

example, we may query for the minimum of an attribute that assumes values randomly distributed between 0 and 4999 ("fivethous"), with the relation partitioned into 100 partitions:

```
range of t is twohoustup
retrieve (minvalue = min(t.fivethous by t.hundred ))
```

2.2. Single User Methodology

Once a synthetic database has been constructed and loaded, the next step is to run a set of queries which measure the cost of executing each of the standard relational database operations. Our set of queries includes the following tests:

- (1) Selection queries with different selectivity factors.
- (2) Projection with different percentages of duplicate tuples.
- (3) Queries involving single and multiple joins
- (4) Simple aggregates and aggregate functions.
- (5) Single tuple updates: append, delete, modify.

Three variations of each query are generally run: first, without any applicable index, second, with a primary (clustered) index on the appropriate attribute, and, finally, with a secondary (non-clustered) index.

Some partial results from the single user tests of the IDM 500 database machine (with a database accelerator) are shown in Table 1. It should be clear from these numbers that our set of single user benchmarks is, itself, capable of generating a wide range of loads on a database system.

We consider conducting single user benchmarks to be a crucial first step in any benchmarking effort. First, in a number of cases the single user benchmarks have uncovered various performance anomalies. If a particular system does not provide satisfactory performance for a type of query (e.g. ad-hoc joins on large relations) which constitutes a high percentage of the queries to be executed by the target application, there is no point in performing multiuser benchmarks on such a system. Second, single user benchmarks provide information on the resources required by different queries. As will be described below, we use these results in developing a multi-user benchmark for a particular system.

One might have noticed that, for the most part, our single user methodology evaluates the performance of each operator individually. Only in the case of join queries do we consider more than one operator at a time. There are a couple of good reasons for doing this. First, we can isolate the cost of each operator. If instead we considered only complex queries (e.g. queries with both joins and selections), it becomes difficult, if not impossible, to understand the results. The other motivation for considering each operation in isolation is use the results to predict the performance of a system for a particular application by weighting the response time of each operator according to its frequency of use in the target application. If one just tested complex queries, such an extrapolation would be impossible.

This is not to say that testing complex, single user queries is not important. In conducting benchmark tests it is quite important to insure that the query optimizer works properly. For example, specifying the join operations before the selection operations in a query enables one to test whether the query optimizer has any intelligence at all. We found that several of the systems we tested were not even smart enough to reorder the operations in a query to do selections first.

2.3. Multiuser Methodology

Three key factors affect the performance of a database system in a multiuser environment: the multiprogramming level, the mix of queries running concurrently, and the degree to which these queries access the same portion of the database. This last factor, which we term "degree of data sharing" can have two different effects on performance. If all the concurrently executing queries are retrieval queries, then a high-degree of data sharing should increase throughput due to buffer pool hits. On the other hand, a high degree of data sharing will result in a reduction of throughput when updating transactions are run concurrently with retrieval transactions (as the result of conflicts for access to shared data pages).

Table 1

Query #	Query	Response Time (seconds)	CPU Usage (seconds)	# of Disk Operations
1	Select 1 tuple from 10,000 using a clustered index	0.7	0.18	2-3
2	Select 100 tuples from 10,000 using a clustered index	1.5	0.56	11
3	Select 100 tuples from 10,000 using a non-clustered index	3.3	0.90	91
4	Select 1000 tuples from 10,000 using a clustered index	8.7	5.90	104
5	Select 1000 tuples from 10,000 using a non-clustered index	23.7	8.67	696
6	Min Scalar aggregate operation on 10,000 tuple relation	21.2	9.83	1,011
7	Min Aggregate function on 10,000 tuple relation (100 partitions)	38.2	35.62	1,008
8	Join 10,000 tuples with 1,000 tuples using a clustered index on join attribute of 10,000 tuple relation	27.6	18.96	206
9	Select 1000 tuples from 10,000 using a clustered index followed by a join with a 10,000 tuple relation using a clustered index	23.4	18.88	207
10	Select 1,000 tuples from 10,000 Select 1,000 tuples from 10,000 Join two 1,000 tuple relations to form a 1,000 tuple relation which is then joined with another 1,000 tuple relation	34.8	107.21	306

The hardest part of developing a methodology for multiuser benchmarks is devising a small set of representative queries to test. We found by partitioning the consumption of CPU and I/O resources into "low" and "high" levels, that we were able to reduce the number of queries needed to test a system to four basic query types:

- Type I - low CPU utilization, low disk utilization
- Type II - low CPU utilization, high disk utilization
- Type III - high CPU utilization, low disk utilization
- Type IV - high CPU utilization, high disk utilization

For the Britton-Lee database machine, we selected queries 1, 3, 8, 7 from Table 1 as being representative of Types I, II, III, and IV respectively. In [BORA84], we show that these four query types are sufficient to achieve a throughput difference of three orders of magnitude. Figure 2 provides an illustration of how data sharing and multiprogramming level affect system throughput for Query type II.

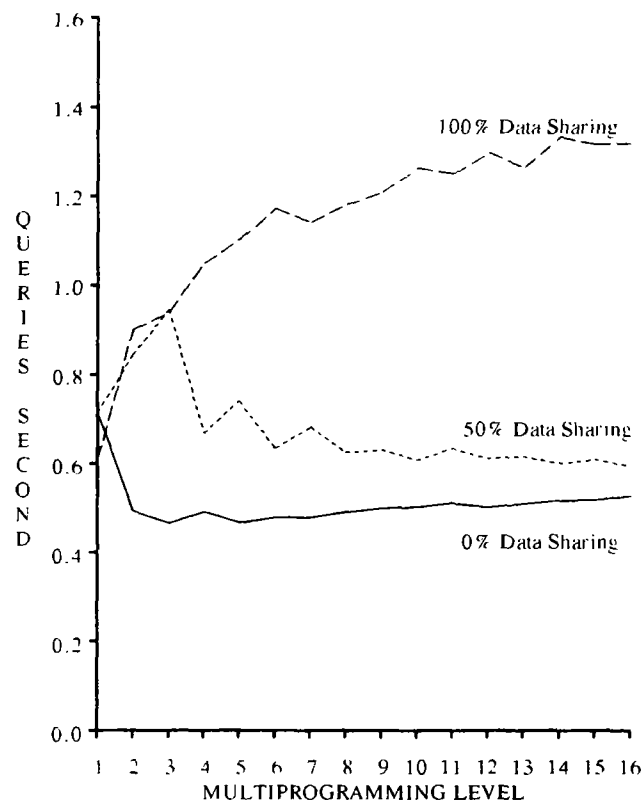


Figure 2

3. Future Research Directions

In this section we outline what we think are some important areas to explore in developing a more complete methodology for benchmarking database systems. We have divided this list into two categories: those that deal with single user tests and those dealing with multiuser issues.

3.1. Single User Research Issues

One pressing issue to explore is to examine under what conditions (if any) can results from single user benchmarks be extrapolated to predict the performance of more complex queries associated with a particular application. The first step in answering this question would be to take application specific queries and transform them into "equivalent" queries on the synthetic database. After being run, one would look to see whether one could have predicted the execution time by combining (in some way) the execution times of the

component operations. Another approach to looking at this problem would be to compare the performance of the "real" application queries on the "real" database with the results obtained by combining numbers from synthetic tests on the "real" database.

A second, unexplored area is the impact of non-uniform distributions of attribute values. That is, if you are retrieving 10% of the tuples in a relation does it make any difference whether the attribute values of those tuples are uniformly distributed or not? It seems fairly obvious that if no indices are involved then it should not make any difference. Differences in performance may, however, occur if an index is being used. Our gut feeling is that there would not be any difference if a self-balancing index mechanism were used¹ (e.g. a B-tree) but that there might be a difference if a poorly organized ISAM structure were being used. One would obviously want to examine the impact on the other relational operations.

A third single user project would be to examine the effect of tuple size on performance. To date, all of our tests have used 182 byte tuples. In [BODG83], a number of tests using varying tuple sizes were conducted on the IDM 500. As one might expect, these results indicate that when one keeps the number of tuples produced by a query constant and increases the tuple width, the response time increases in pretty much a linear fashion. The degree of the increase should depend on the extent to which the system is I/O bound. If I/O can always be overlapped with the CPU, then the degree of the increase should be proportional to the cost (in CPU time) of initiating an I/O operation relative to the cost (in CPU time) of processing the page. While the increase in response time in a single user mode might be relatively small, throughput in a multiuser environment would always be more directly affected. A interesting variation would be to keep the volume of data accessed from disk fixed, while varying the selectivity factor and the tuple size.

Another single user project is to further refine the ability of the benchmark to isolate the "faulty" components of a database system. For example, when testing SQL/DS on a 4341, we once saw a particular query run in a couple of minutes. After we did some vendor-suggested physical database reorganization (ie. how relations and their indices were laid out on disk), the same query took 9 hours. Had we never seen the 2 minute time, we would have tended to suspect either a bug or a poor join algorithm. As it turned out, the database reorganization caused the query optimizer to change its mind about what join algorithm to use. What seems to be needed is a benchmarking methodology that can check out the components of the system independently of one another. This "isolation" strategy would make it possible to evaluate and test the components individually.

3.2. Multiuser Experiments

A number of challenging multiuser research projects are also possible. The first is to simplify the present multiuser methodology. The results published in [BORA84] required well over 100 hours of stand-alone time on both an IDM 500 and a host processor. While the results obtained are interesting, it is simply not clear that all four query types are needed to stress a database system adequately. Since the principal goal of the multiuser benchmarks is to explore the behavior of a system under load, it might instead be sufficient to run a very large number of simple queries simultaneously. This is the approach suggested in [ANON85]. While pushing both CPU and I/O utilization levels to 100% is the goal of the multiuser tests, there may be cases when pushing each one as separately as possible to 100% yields more information about the behavior of the system. One potential drawback that we see with this approach is that by using simple debit/credit queries as the basis of such a benchmark, one may miss testing the ability of the buffer manager to properly handle complicated access patterns.

An extension to this effort would be to develop a portable multiuser benchmark. While the methodology described above can itself be applied to any relational database system or machine, the type of a particular query may vary from system to system depending on the algorithms used on each system. One motivation for using a simple/debit credit transaction as the basis for the multiuser benchmark described in [ANON85] was to insure portability across systems. While this is an appropriate benchmark for transaction processing systems, it may not be an acceptable benchmark for more sophisticated relational database systems.

¹ Unless one or more leaf pages had a number of overflow pages that were not physically clustered near the leaf page. In this case, the extra seeks might result in a slight difference in performance.

Techniques for thoroughly testing the concurrency control and recovery mechanisms of a database system also need to be developed. In [BORA84], one experiment was conducted in which transactions that updated a single tuple were run concurrently with transactions that retrieved a single tuple. With a multiprogramming level of 16 and a database consisting of 10,000 tuples, conflicts between transactions were very infrequent even though the updating transactions modified an attribute on which a primary index existed. To thoroughly evaluate a concurrency control mechanism two tests need to be developed. First is a method of generating a range of conflicts between concurrently executing transactions. One possibility would be to run a single bulk-update transaction concurrently with a number of read-only transactions. By varying the percentage of the database updated, one should be able to vary the number of conflicts generated. An alternative approach would be to reduce the size of the relation to just a few tuples and continue to use transactions that do single tuple updates.

For database systems that use locking for concurrency control, a way of testing the deadlock resolution mechanism is also needed. The following approach looks promising. Consider two types of transactions: one which uses a clustered index to access tuples which are then updated by modifying a non-indexed attribute. The second type uses a non-clustered index to access the tuples to be modified. When transactions of the first type are run concurrently, no-deadlocks should occur as data pages will be accessed once and in "key" order. The conflict rate between transactions will depend on the multiprogramming level, the size of the relation, and the number of data pages accessed by each of the queries. When transactions of the second type are run concurrently, deadlocks will occur as data pages will be accessed in random order and, possibly, multiple times. By varying the number of data pages accessed, one will be able to control both the conflict rate and the deadlock rate. By comparing the throughput of the first case with that of the second, one should be able to access how well a system handles deadlocks.

With regard to recovery, measuring two aspects looks appealing. The first project would be to measure the cost of gathering recovery information. One should be able to do this by simply turning the recovery manager off. Determining this cost as a function of the percentage and size of the updating transactions would be interesting. A second project would be to determine the cost of transaction aborts on system throughput by varying the rate at which updating transactions simply exit instead of committing.

3.3. Other Projects

Given the results (ie. response time, cpu utilization, and disk utilization figures) of a set of single user benchmarks, it would be nice to be able to dispense with most of the multiuser benchmarks and instead use an analytical model to predict the multiuser behavior of a database system. Recently we have been examining internal measurements from our multiuser benchmarks on the IDM 500 in an attempt to determine exactly what details need to be captured by such a model. Preliminary results indicate that building such a model may be complicated as the multiuser characteristics of the buffer manager seems to have a significant effect on the throughput of the system. Life becomes even more complicated if one wants to predict multiuser behavior with updating transactions. It may be that models of concurrency control and recovery mechanisms could be adapted to help in this case.

So far we have only addressed single site database systems. Mechanisms for testing distributed database systems will obviously be needed in the future. While some of the single site techniques should be applicable to distributed systems, it seems apparent that a new set of techniques will be needed for these systems.

4. Conclusions

In this paper we have surveyed our earlier work on database system performance evaluation. While our single user and multiuser techniques have become widely used, these tools are just the first of a number of tools that are needed. While the present tools help determine overall system performance, they are not adequate for isolating exactly which components are "faulty". In addition, they do not do an adequate job of stressing the concurrency control and recovery components of a system.

5. Acknowledgments

A number of the ideas in this paper have been stolen from others. Unfortunately, I cannot remember who made exactly what suggestions as some number came during presentations of the various benchmarking papers. Mike Carey and Mike Ubell are certainly two of the contributors. Dina Bitton, Haran Boral, and Carolyn Turbyfill deserve recognition for the key roles they played in developing the current benchmarking tools.

6. References

- [ANON85] Anon Et. Al, "A Measure of Transaction Processing Power," to appear. *Datamation*, Feb 15, 1985.
- [BOGD83] Bogdanowicz, R., Crocker, M., Hsiao, D., Ryder, C., Stone, V., and P. Strawser, "Experiments in Benchmarking Relational Database Machines," **Database Machines**, Springer-Verlag, 1983.
- [BITT83] Bitton, D., DeWitt, D. J., and C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach." Computer Sciences Department Technical Report #526, Computer Sciences Department, University of Wisconsin, December 1983. This is a revised and expanded version of the paper that appeared under the same title in the Proceedings of the 1983 Very Large Database Conference, October, 1983.
- [BORA84] Boral H. and D. J. DeWitt, "A Methodology for Database System Performance Evaluation." Proceedings of the 1984 SIGMOD Conference, Boston, MA., June 1984.
- [IDM500] IDM 500 Reference Manual, Britton-Lee Inc., Los Gatos, California.
- [STRA83] Strawser, Paula, "A Methodology for Benchmarking Relational Database Machines." Ph.D. Dissertation, Ohio State University, December 1983.

TIPS ON BENCHMARKING DATA BASE SYSTEMS

by

Michael Stonebraker
Relational Technology, Inc.
2855 Telegraph Ave.
Berkeley, CA

ABSTRACT

This paper contains a collection of suggestions to persons considering benchmark evaluation of data base systems (DBMS) and summarizes my experience with benchmarking studies over the past several years. These suggestions include comments on published benchmark scripts, issues to consider, and pitfalls to avoid.

1. INTRODUCTION

Many potential DBMS users benchmark the commercial offerings of vendors of data base systems, and base their purchasing decision in part on the results of such benchmarks. Since the available products differ widely in performance and ease of use, this tactic is often a useful one. Benchmarks should measure query/update performance, as well as ease of application development. When planning and conducting a benchmark, one must make decisions regarding the choice of what to benchmark (the benchmark script), the person who does the benchmark, the person who tunes the benchmark for performance, and how to evaluate the result. This paper summarizes my thoughts on these questions.

2. TYPE OF BENCHMARK

The first issue is whether to do a single-user benchmark (in which a collection of commands are timed as if they were submitted sequentially by a single user) or to perform a multi-user benchmark. Of course, a multi-user benchmark is much harder to construct and often much harder to evaluate than a single-user one.

SUGGESTION 1: Perform a multi-user benchmark unless you truly have a single-user environment.

A multi-user benchmark tests two features of data base systems which are not evaluated by a single user benchmark. First, the concurrency control facilities of the various vendors differ widely in function and performance. Locking is the concurrency control mechanism used to guarantee data consistency during reads and updates by multiple users simultaneously. Differences include the locking granularity used by the vendor for read operations (e.g. records, pages, relations, whole data base), the locking granularity for updates, the locking granularity for schema modifications (e.g. adding or dropping an index), how deadlock detection and resolution is accomplished, and whether there is support for multiple lock granularities and lock escalation. (On commands which touch many records, it is more efficient to lock larger objects. Hence when the discovery is made that a data-intensive command is being processed, it is beneficial to

exchange any smaller locks that have already been set for a single larger enclosing lock.)

For example, if a vendor chooses to lock a whole relation when tuples in that relation are modified, then updates are effectively single-threaded because locks must be held to the end of a transaction. As a result of a single-user benchmark, a potential user will not be made aware of important shortcomings such as this one.

Another problem with single-user benchmarks concerns system performance. In many data base systems, there is a considerable difference between performance in single-user and multi-user environments. This results from buffer management issues (such as read-ahead and write-behind tactics) and consumption of system resources.

A data base system which expects to be used in a multi-user environment will often not be concerned with optimizing read-ahead and write-behind for a single user. Such a system would not attempt to read pages from the disk in advance of their being requested by a user (read-ahead), nor would it attempt to queue write requests for a user (write-behind). Rather, it would be more concerned with optimization of concurrency control, crash recovery, and increasing the average number of transactions per second that can be run in a multi-user environment. Such a system, when run in a single-user environment, will either be in page-wait status (waiting for an I/O operation to complete) or it will be executing code on behalf of that user. No CPU activity will be overlapped with I/O activity in a single-user environment. Moreover, it can execute the concurrent commands of two users in about the same amount of time as the command of a single user. This results from overlapping the CPU activity of one user with the I/O activity of the second. Hence, a single user benchmark will often understate the amount of work that can be accomplished in a multi-user environment, and a user will not get an accurate picture of resource consumption.

3. THE BENCHMARK SCRIPT

One next has to face the issue of what commands to put in the benchmark. There are four choices:

- 1) use a canned benchmark such as the one in [BORAL84]
- 2) have a vendor choose the benchmark
- 3) create an artificial benchmark
- 4) use a real application

SUGGESTION 2: Use a real application if possible.

The best choice for a benchmark script is a real application from your environment. We discuss the drawbacks of the other options first, and then comment on the benefits of a real application.

The first option is to use a canned benchmark. The one in [BORAL84] (the Wisconsin benchmark) is widely suggested as a reasonable candidate; however, it suffers from two flaws:

- 1) It has no floating point operations.

Data base systems differ widely in their support for floating point

operations. Some systems simulate floating point operations in software using decimal numbers as a storage mechanism. Others use the available floating-point hardware to support a built-in floating point data type. The latter option is dramatically faster than the former, and one will not be made aware of this difference from the Wisconsin benchmark.

2) It has no copy operations or schema modifications.

Many installations spend a considerable amount of time loading and unloading data sets and building and changing schemas. One will get no information on the performance of such functions from the Wisconsin script. Moreover, one will not be made aware of any concurrency control deficiencies in schema modifications (such as the choice by some vendors to lock the whole data base on schema changes) from the Wisconsin test suite.

A second general benchmark has been suggested by Jim Gray and is in draft state [GRAY84]. This script is appropriate only for production transaction processing systems and includes the well known TPL banking transaction. TPL contains three update commands and one append command each affecting a single record. These commands simulate the action of a bank teller cashing a check for a customer. TPL is largely a test of the concurrency control and crash recovery facilities of a data base system and its overhead on single record interactions. This script is reasonable for some production transaction processing applications, but will not be helpful in any environment which contains decision support functions.

It would be great to have a single (or even a small number) of general purpose scripts (the whetstones of data base management), and I applaud the initial efforts in this direction by the above authors. However, the above comments point out the difficulty of creating a general purpose benchmark that will test all of the aspects of a data base system that many potential clients would want to test. Hence choosing a canned script may not be a suitable option at the current time.

The problem with allowing the vendor to choose the benchmark (option 2) is that almost all data base systems excel at some collection of commands. Most vendors have had enough benchmarking experience to recognize this set of interactions and can easily choose a winning benchmark. Therefore, this option is only desirable if you have already selected a vendor's data base system.

The third way to construct a benchmark script is to choose an artificial benchmark. Unfortunately, this benchmark is arbitrary, and a losing vendor will complain that it is biased and suggest changes. In all probability, the vendor will complain to you, to your boss, and to your boss' boss. Hence, you will have to mediate complaints of unfairness and perhaps dynamically adjust the composition of the script.

A script from a real application is free from any possible criticism concerning arbitrariness; moreover, it will provide a good indication of how one's problems will run on a particular vendor's system. The only consideration is that the benchmark must be relatively simple if the vendors will be required to program it. If the benchmark requires several person-weeks to code, one will guarantee that only the very large vendors can afford to execute the test. But why not do it yourself to test the product's ease of use, documentation completeness, and vendor technical

support?

4. RUNNING THE BENCHMARK

The next issue concerns who will run the benchmark. There are three choices:

- 1) the vendor can run the benchmark on his machine
- 2) the vendor can run the benchmark on your machine
- 3) one can run his own benchmark

SUGESSTION 3: If at all possible, run the benchmark yourself.

There are a multitude of reasons for choosing this option. First, if the benchmark is run on the vendor's machine, the results may not be reproducible in your environment. There are many innocuous reasons for this behavior; disk drives differ in speed, configurations are different, etc. Moreover, a vendor may be tempted to use his latest "about to be Beta-tested" version of his system. This system may not be available to you for several months.

Another disadvantage of this approach is that a vendor can subtly change the benchmark to improve its performance. For example, most systems will execute retrieval operations faster if the output is not sorted and duplicate records are not removed. In addition, all systems go faster if the output is thrown away rather than printed or delivered to an application program. If one's benchmark is not precise on all these points, a vendor is free to choose the option which executes fastest in his environment.

In addition, some vendors have systems which generate a query processing plan for complex queries by examining the clauses in the query qualification from left to right. Hence, performance will differ dramatically depending on the order of the clauses in a complex qualification and query performance will be data dependent. If the vendor runs the benchmark, he is at liberty to rearrange the qualifications to improve performance. A client does not find out about such shortcomings with vendor run benchmarks. Ideally, you want to select a system in which the performance does not depend on the expertise of the person writing the queries.

The second option is to ask each vendor to run the benchmark at your site. Most vendors will respond by having their local technical sales support person do the benchmark, or by sending in a special "swat team".

The problem with a swat team is that such specialists disappear when the benchmark is over, and their tactics are not necessarily ones that you will have the expertise (or desire) to use. The following are tactics which I have seen swat teams use:

- 1) divide a relation into 26 physical data sets, one for each letter of the alphabet. This vendor had a concurrency control scheme which locked whole relations on update. With this scheme multiple concurrent updates could be processed as long as they specified different first letters for an indicated key. Of course, this precludes the possibility of performing aggregates on this relation; unfortunately there were none in the benchmark

to preclude the use of this tactic.

2) rewrite queries to take advantage of formats the optimizer can use. For example, the query language SQL has two ways of expressing joins, as nested queries

```
select sname
from supplier
where s# in
      (select s#
       from supplier_parts
       where p# = 2)
```

and as flat queries

```
select supplier.sname
      where supplier.s# = supplier_parts.s# and
            supplier_parts.p# = 2
```

In some vendor products, the query optimizer can not optimize the nested query format, so vendors will rewrite those queries as flat queries when measuring performance - even though these same vendors emphasize their nested query feature.

Of course, if a vendor chooses to have his local sales support people perform the benchmark, they may well use similar tactics. Hence, a user should always be on the lookout for the use of such programming stunts and disallow them. The one advantage to using local support people rather than a swat team is that a user can evaluate the competency of the people who will assist him after the sale.

The best option is to run one's own benchmark. In following this course of action, the user obtains a great deal more information on a vendor's product than with either of the previous options. In particular, one can test the ease of installation of the system, test the reliability of the software, and discover the quality of the system documentation. Moreover, one discovers how easy it is to write applications on the system. For example, one client elected to test two systems by having two different employees program and run the benchmark on the two data base packages being compared. One product required one-third of the programming time of the second because of subtle restrictions in one programming language interface. Such valuable information results only from internally run benchmarks. Lastly, one can ascertain the responsiveness of the technical support from each organization involved in the benchmark. It appears that various vendors differ widely in how much energy they invest in ensuring that users get helpful, accurate, and prompt answers to their problems.

5. WHO TUNES THE BENCHMARK

Among the tuning options:

- 1) no tuning
- 2) user tunes the benchmark with no assistance
- 3) user tunes the benchmark with vendor assistance
- 4) vendor tunes the benchmark

SUGGESTION 4: Choose option 3 if possible.

One client executed a benchmark between two systems with no tuning of storage structures whatsoever. The reasoning was to simulate the behavior of a naive user who might not consider performance tradeoffs. The problem with this approach is that several orders of magnitude in performance differentiate optimized and unoptimized storage structures. The default storage structures of any particular system (e.g. heap, keyed on the first field, etc.) may or may not work well for any particular script. Hence, performance of any particular system is effectively a random variable. This is no way to evaluate DBMS performance!

The second option is for the user to tune his own benchmark, and this is certainly preferable to no tuning at all. In sophisticated environments where query processing tactics are well understood, this approach will probably lead to an optimized benchmark. Moreover, it will give the user a feel for the optimization parameters of any particular system. However, in shops that are new to relational data base technology, the algorithms used by a query optimizer may not be well understood. In this case, advice from the vendor will help in choosing good storage structures. Even in sophisticated shops, it is probably wise to have the vendor check a schema for performance oversights. A useful way to accomplish this function is to give the vendor a copy of your script and test data. After he has implemented the benchmark on his system, you can ask him for performance suggestions. This option also provides a way of evaluating the quality of training guides, classes, and technical support.

The final option is to have the vendor tune the benchmark for you. This approach is an invitation to the swat team tactics discussed above. Avoid this approach if possible.

6. EVALUATION OF THE RESULTS

There are three considerations to think through when evaluating the results of any benchmark studies.

6.1 Future Versus Present Performance

Every vendor is "about to come out with" his next system which is "2-10 times faster" than his current system. It "fixes all known performance problems" and can be benchmarked "in a little while". In general, one has to decide whether to run a benchmark on:

- 1) a production system
- 2) a Beta-test system
- 3) a system still in development

Moreover, one has to decide whether to delay the benchmark under pressure from a vendor who has a next system "almost ready"

My advise is to simply realize that this issue is bound to arise and to think through in advance how to deal with it. Also, one should realize that all relational systems are becoming progressively faster. A purchase decision will usually result in a commitment to a particular vendor for at least a couple of years. During this time, any system under consideration will get faster. Hence, one should consider both:

- 1) how fast the system is today
- 2) how fast it is likely to be in a year

Of course the future is difficult to predict, and data base salesmen are notorious for optimistic predictions. Two possible tactics to use in addition to asking the vendor for his prediction are:

- 1) Ask the vendor for a benchmark that has been run on his last several releases. One can probably expect approximately the same relative performance improvement in future releases.
- 2) Ask the vendor how many development engineers he has dedicated to performance improvement. Do not expect much improvement if this number is small.

6.2 Future Functionality

All vendors are making their systems more functional in areas such as:

- number of data types
- number of operators
- support for business graphics
- support for business forms
- application generators
- report writers
- support for spread sheets

Moreover, many vendors have plans to release their system on new computers. Ask each vendor about his future plans in areas of concern to you. Remember that one will, in all probability, be using several releases of the vendor's software. Again, one must assess the credibility of vendors who say that everything one could conceivably want will be available "in the very near future". Ask each vendor for the dates and functional enhancements in his last few systems. One can probably assume that he will make about the same rate of progress in the future as he has made in the past. Another good reality check is to ask the vendor for the size of his development staff and how they are allocated (e.g. how many are working on conversions, how many on new functionality, and how many on performance). While you are asking such questions, also inquire how many clients per technical support engineer the vendor has and how many quality assurance engineers. These will be good indicators of the level of technical support and the reliability of the software.

6.3 Performance Versus Coding Difficulty

Many users evaluate a DBMS only on the expected level of performance and do not consider the suitability of vendor provided application development tools for end users and application developers. I feel that most users are very shortsighted and should consider the cost of writing an application in addition to the cost of executing it. During the remainder of this century, software costs will play an increasingly important role in overall application cost. This section gives a hypothetical example to illustrate this point.

Suppose a client is purchasing a DBMS to run a single application and has performed enough benchmarking to obtain the following table for three

hypothetical systems.

	Lines of code	Running time of the benchmark
System A	20,000	6 minutes
System B	12,000	8 minutes
System C	6,000	10 minutes

For example, System C might have a sophisticated application generator which dramatically cuts the effort involved in coding an application while System A might have only a subroutine call interface from a general purpose programming language. A user would have to write a lot more code to get his application to run in such an environment. Presumably (but not necessarily) the added function which allows a user to bring up his application with less code in System C will result in slower performance. The above table suggests three systems with increasing performance and difficulty of use.

Suppose all three systems are "fast enough" to meet whatever response time requirements exist. Hence, one should choose between the systems based on how cost effective each is at coding and executing your application for its lifetime. Assume that your application will be run for 5 years and then discarded or rewritten. During that time it will be used 8 hours per day, 250 days per year, and must support an average of 5 interactions per minute. Suppose the benchmark noted above is constructed of 100 such interactions. Lastly, suppose that computer time on your system costs \$3.00 per minute and that your shop can write a line of code and maintain it for its 5 year lifetime for \$25.00 per line.

With these numbers one can compute the following table for the 2,400,000 interactions which will be run during the lifetime of this application:

	cost to write and maintain the application	running cost per interaction	hardware cost during the lifetime of the application	total cost of the application
A	\$500,000	\$0.18	\$432,000	\$932,000
B	\$300,000	\$0.24	\$576,000	\$876,000
C	\$150,000	\$0.30	\$720,000	\$870,000

Notice that the fastest system (A) is the least cost effective solution. Moreover, Systems B and C are about equally cost effective, even though System C is 20% slower. In fact the following is the general strategy for this application:

If your lifetime number of interactions is:

less than 2,500,000 then choose System C
more than 3,333,333 then choose System A
else choose System B

Hence, our hypothetical lifetime number of transactions is in the range where System C is the winner for the assumed parameters. Of course, any real shop should perform the above calculation using actual numbers for their application. However, one should carefully note that for any

technology parameters (e.g. \$25.00 per line, \$3.00 per minute) there will be an interaction volume below which C is the correct choice and another volume above which A is the best choice.

Over the remainder of this century the cost to write and maintain a line of code will probably remain constant or increase while the cost of computer time should decrease rapidly. These technology trends will increase the lifetime number of transactions for which System C is the best choice. Hence one can say in general:

For any application with any given lifetime number of interactions, sooner or later the easier to use a system with slower performance will be the most cost effective solution.

One should simply keep these technological trends in mind when conducting a benchmark, and include application generation and maintenance costs in one's overall benchmark evaluation, whenever possible.

7. SUMMARY

The conclusions to be drawn from this paper are that one should use a simple multi-user application from one's own environment for a benchmark. One should control the running and tuning of the benchmark and should carefully avoid swat team tactics. Set a firm date for the conclusion of the study to avoid procrastination by a vendor with a next system "almost ready". Set clear criteria for determining the winner. Lastly, include application development costs, predicted future performance, and future functionality, as well as current performance in your evaluation.

REFERENCES

- [BORAL84] Boral, H., DeWitt, D.J., "A Methodology for Data Base Systems Performance Evaluation", Proc. 1984 SIGMOD, Boston, Massachusetts, June 1984.
- [GRAY84] Gray, J., "A Transaction Processing Benchmark" unpublished working paper.

Variations on a Benchmark

Paula Hawthorn
Britton Lee Inc.

ABSTRACT

The Wisconsin benchmark is a general program that has been used to benchmark the Britton Lee Intelligent Database Machine (IDM). This paper presents an augmentation to the benchmark that more correctly represents the performance effect of Britton Lee's use of special-purpose hardware. This augmentation is necessary because the Wisconsin Benchmark does not include "amount of data returned" as a factor in the performance of a DBMS.

1. Introduction

The Wisconsin benchmark was developed by Bitton, Boral, DeWitt and Turbyfill at the University of Wisconsin. The benchmark is readily available, uses standard relational queries, and is well documented. In a previous paper, [BORA84], DeWitt and Boral use the Wisconsin benchmark to show the effect of the Britton Lee special-purpose processor, the Database Accelerator, on the throughput of the system. In this paper we show that the result in [BORA84] is lower than can be expected in many applications. We then show why the Wisconsin benchmark failed to fully expose the capabilities of the DAC, and finally offer suggestions for the augmentation of the benchmark based on a more general model of DBMS processing.

Section 2 is a discussion of the Wisconsin benchmark, and of the queries we added. Section 3 is the conclusion.

2. Benchmark

The Wisconsin Benchmark is discussed in [BITT83] and [BORA84]. One of several experiments described in [BORA84] was to determine the effect of Britton Lee's custom designed 10 MIPS processor, the Database Accelerator (DAC) on the performance of the system. The Britton Lee Intelligent Database Machine uses a DAC as a callable co-processor, called by code executing on a general-purpose Z8000. If the DAC is not present, or is turned off, the Z8000 executes the code itself.

In [BORA84] an experiment was described where several queries were run with the DAC alternately on, then off. The result, that the DAC increased performance by, at most, a factor of 1.71, was counter to what we at Britton Lee typically see in user applications, where a factor of 2 - 7 is more usual. Therefore, we reran the benchmark, analyzed the results, and added a class of queries to the benchmark that show where the DAC more significantly increases performance. The following is a brief description of the Wisconsin Benchmark, and of the benchmark runs that we performed.

2.1. Description of Wisconsin Benchmark

The following section briefly summarizes the description of the Wisconsin benchmark found in [BORA84].

The Wisconsin Benchmark consists of:

Benchmark Variations

1) database creation scripts that create thirty-two relations: 16 of type tenKtup and 16 of type oneKtup. These relations contain, respectively, 10,000 tuples and 1000 tuples. The tuples in both relations consist of 13 2-byte integer attributes and 3 52-byte compressed character fields. 16 of each type are created so that in the multi-user benchmarks a multiprogramming level of 16 with no data sharing can be assured.

2) database loading scripts that load the relations with synthetically generated data. The resulting relations then have precisely known characteristics: the first two attributes ("unique1" and "unique2") are unique and random; the next 11 integer values are generated according to precise selectivity factors, so that, for instance, attribute "two" has two distinct values, attribute "hundred" has one hundred distinct values, etc.

3) index creation scripts that create clustered indices on "unique2" in all relations, and non-clustered indices on "unique1" in all relations.

4) a UNIX script to call the "multibench" program, control the level of multiprogramming, and direct timing results to the proper files.

5) The "multibench" program itself, which contains four queries (described below) and executes the queries on the database according to parameters furnished it: the parameters specify the percentage of datasharing, and based on that percentage a random number is generated to randomly select which relations shall be accessed; parameters also specify the number of queries to be run, and the percent of that total that should be Query 1, Query 2, Query 3 or Query 4. During execution, a random number is drawn to determine which of the queries should be executed next, according to the percentage specified. Multibench calls the Unix 1-second granularity clock before and after each query is executed, and writes the time to a file.

6) The "stats" program that analyzes the output files to determine the total throughput of the benchmark run.

The queries are:¹

```
Query 1: /* select one tuple using clustered index */
        int a, b, value; long randomnumber;
```

```
        /* select a random key value between 0 and 9999 */
        randomnumner = r0random(&seed);
        value = randomnumber % 10000);
```

```
        range of x is tenKtup
        retrieve (a = unique1D, b = unique2D)
                where x.unique2D = value
```

```
Query 2: /* select 100 tuples out of 10000 */
        /* using a non-clustered index */
```

```
        int lowervalue, uppervalue, a, b;
```

```
        /* select a lower range value between 0 and 9900 */
        randomnumber = r0random(&seed);
        lowervalue = (randomnumber % 9901);
```

¹ [BORA84]

Benchmark Variations

```

uppervalue = lowervalue + 100;

range of x is tenKtup
retrieve (a = x.unique1D, b = x.unique2D) where
(x.unique1D >= lowervalue) and x.unique1D < uppervalue)

```

```

Query 3: /* Join using clustered index on unique2D */
/* Query produces 1000 tuples */

int a, b, c, d;

range of t is tenKtup
range of w is oneKtup

retrieve (a = t.unique1D, b = t.unique2D, c = w.unique1A
d = w.unique2A) where t.unique2D = w.unique1A

```

```

Query 4: /* Aggregate function min with 100 partitions */

int xmin;

range of x is tenKtup
retrieve (min = min(x.twothousD by x.hundredD))

```

The selection process that led the Wisconsin researchers to use the above four queries as the complete multiuser benchmark set was that they determined that there are basically four types of queries, and chose the above queries as representative of the types. The types are: low CPU utilization, low disk utilization (Query 1); low CPU, high disk (Query 2); high CPU, low disk (Query 3, with adequate buffering); and high CPU, high disk (Query 4).

2.2. Benchmark Results

We reran the Wisconsin Benchmark on a Britton Lee Intelligent Database Machine (IDM) exactly as the benchmark came from Wisconsin. In Figure 1, the bars Query 1, Query 2, Query 3 and Query 4 represent the IDM performance as reported by the "stats" program for queries 1-4 of the Wisconsin benchmark.

The benchmark was run on a 3-Mbyte, 4 disk IDM, front-ended by an 11/70 and connected via a (IEEE 488) parallel channel. The benchmarks were run with levels of multiprogramming varied from 1-10, and with 100% data sharing.

maximum throughput (queries/sec)		DAC		no DAC		DAC		no DAC	
	(26)								
26	xx								
	xx								
24	xx								
	xx								
22	xx								
	xx	(20)							
20	xx	xx							
	--	--							
	xx	xx							
6	xx	xx							
	xx	xx							
4	xx	xx	(2.45)	(2.45)					
	xx	xx	xx	xx	(.13)	(.07)	(.03)	(.02)	
2	xx	xx	xx	xx	xx				
	xx	xx	xx	xx	xx	xx	xx	xx	xx

	Query 1	Query 2	Query 3	Query 4					

100% data sharing

FIGURE 1. Wisconsin Benchmark Queries

We express the results in "queries/sec", as is done in [BORA84], for comparison purposes. The above results agree with the results reported in [BORA84]: that the maximum percentage improvement provided by the DAC for these queries was in Query 3, the clustered index join. To keep the experiments the same as in [BORA84], we used 100% datasharing (that is, all queries go to the same two relations) for the DAC tests.

For this particular data set, we would not expect that the DAC would improve the performance greatly. That is because the DAC is designed primarily for fast string comparisons, and none of the above queries perform such comparisons. Since we were trying to understand why the Wisconsin benchmark did not represent our actual user experience with the IDM, we reviewed the customer benchmarks that have been previously run on the IDM, and noted that there are two striking dissimilarities between those benchmarks and the Wisconsin benchmark: first, the data in the user benchmarks is nearly all character data, and the key attributes almost always uncompressed character data. Second, the queries tend to return little data.

It turns out that *both* of these differences are the cause of the failure of the Wisconsin benchmark to reflect our experience with the DAC. To show this, we changed one factor at a time. First, we created new relations (see Appendix A for the scripts to create the new relations) substituting character strings for each of the integer strings, and ran the benchmark set again. The result was that the extra length of the fields returned caused more character handling by the IDM channels and the 11/70, which caused this particular query set to still show results similar to the first set: no more than double improvement between the non-DAC and DAC systems. This is shown in Figure 2, below, in the case of Query 3C: this is query 3, with character attributes in the place of numeric attributes.

Query 3 returns 1000 tuples to the user process. To change the second factor - that is, to return less data, we ran Query 3 with a qualification; the query then became Query 3Cq:

```
Query 3Cq: /* Join using clustered index on unique2D */
          /* Query produces 1000 tuples */
          /* further qualified to produce one tuple */

char     a[20];

range of t is tenKtup
range of w is oneKtup

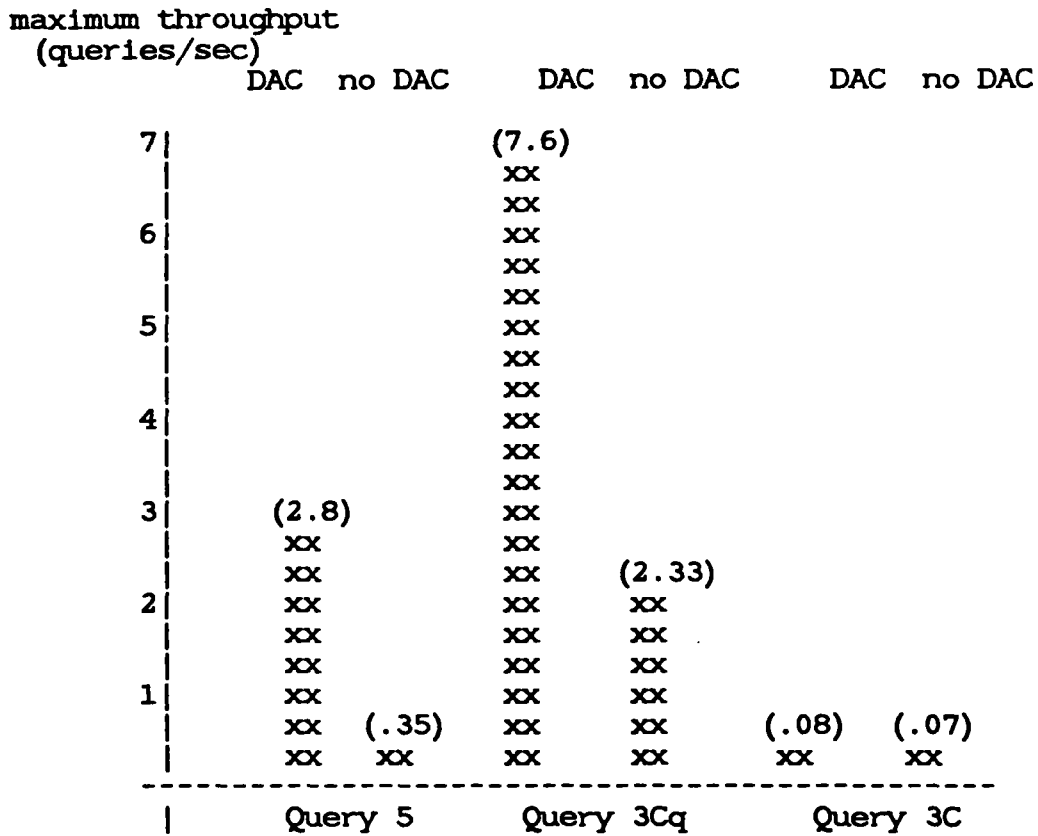
retrieve (a = w.unique1A) where
         t.unique2D = w.unique1A and w.thousandD = "10000"
```

The qualification results in one tuple returned. Figure 2 shows that use of the DAC improves Query 3Cq performance by a factor of 3.2.

As a final test, we changed the database itself. The DAC requires a set-up time per page, and per compressed attribute involved in the query. Therefore, maximal DAC improvement will result when there are many uncompressed attributes per page included in the qualification clause. So we created a relation made up of 10,000 tuples, each having 2 uncompressed 20-character attributes, and ran the following query:

```
Query 5:
range of p is packed
retrieve (cnt = count(p.thousandD where p.thousandD = "10000"))
```

This resulted in the largest DAC speedup in this benchmark - a factor of 8.



100% data sharing

FIGURE 2. Augmented Queries

2.3. Analysis

All of the above queries are artificial in the sense that they re-reference the same data. That is, the experiment is to, for instance, run Query 3 fifteen times - since "100% data sharing" is specified, that means that, with adequate buffering (and there was adequate buffering) all the data referenced the second time will be in memory. This was done purposefully in [BORA84] in order to make the queries as CPU bound as possible, and thus show the maximum effect of the DAC. The augmented queries above show that the results of [BORA84] do not actually represent the effect of the DAC because there are more than two factors that effect the per-process response time of any DBMS. Other factors are:

- 1) Input/Output processing speed: for the IDM, this is a function of the number of bytes transferred to the host, of the speed of the channel, and the speed of the host. For single-machine DBMS, this is the time to move data from the user application to the DBMS and back again.
- 2) Disk utilization: this is a function of the number of disks, the spread of accessed data across the disks, the use of algorithms that minimize disk accesses, and the amount of memory available.
- 3) General purpose CPU utilization: this is a function of the algorithm used in the query, the amount of output processing that must be done (formatting the data for display), the amount of memory (less memory means more CPU time spent managing memory).
- 4) Special-purpose hardware utilization: this depends on the set-up time to use the special-purpose hardware, and its functionality. For the DAC, Where the set-up time is large, and the time spent in the DAC small (e.g., whole-page tuples with only a single small attribute that requires character comparisons) the DAC will not be heavily utilized; where the set-up time is low (e.g., creating indices on character attributes), the DAC utilization will be high.
- 5) Memory utilization: This is a function of the size of memory, the buffer management scheme used, and the query mix.
- 6) Locking strategy

A benchmark that attempts to represent DBMS performance inherently is affected by all the above factors, whether or not the user is aware of the factors.

2.4. The Real World

Since the above queries are clearly artificial, what is the real world like? In our experience, most people are using mainly character data for their "business" databases; most buy large-memory systems, so there is adequate buffering, and the DAC is used effectively; many connect a single IDM via RS232 serial lines to several hosts, and are more concerned about total IDM throughput than per-process response time. However, the most important thing we have found is what we have all known to be true: the overall performance of the IDM is very application-dependent.

3. Conclusion

There are two general methods of benchmarking data management systems: application benchmarks and system benchmarks. An application benchmark is an attempt to capture the essence of an application by modelling the application queries and data, then running the benchmark on the data management system using this model. The purpose of the system benchmark is to characterize the performance of the data management system in an application-independent environment. The challenge in benchmark development is to partition the

query/data/algorithm/hardware space into a finite set of queries which reflect fully the factors of performance of the system.

Application benchmarks are commonly developed as part of a potential customer's evaluation of competitive systems. They represent a great cost to both the customer, who must spend time and effort characterizing the application, and the vendor, who must help the customer develop and run the benchmark. It would be wonderful if a single, believable system benchmark could be developed and used which would characterize the performance of the DBMS for all applications, so that customized application benchmarks would no longer be necessary.

The Wisconsin benchmark was developed as an application-independent systematic benchmark of relational data management systems. The relations contain precisely quantified attributes so that the selectivity and randomness of the queries is known. In [BORA84] it is proposed that there are 4 query types, representing two factors of DBMS performance: CPU utilization and disk utilization. We have shown that there is at least a third factor: amount of data returned to user; and that without consideration of this third factor the benchmark cannot expose the performance enhancement furnished by the DAC. The new queries that we propose adding to the Wisconsin benchmark are queries that include this factor.

The Wisconsin benchmark is an important contribution to the development of the architecture of data management systems - both software and hardware. As new systems are developed, the Wisconsin benchmark can be run on them to determine whether the new systems are improvements over other, existing systems. However, as we have shown, this benchmark (or any general benchmark) must be used with care. In analyzing the results of the benchmark, the analyst should be sure that the benchmark is actually measuring what the analyst is looking for - that is, in measuring one of the six factors in section 2.3, to be sure the benchmark is not hung up on one of the other six factors. For instance, [BORA84] was specifically attempting to define what the increase in performance due to the DAC was (factor 4), but in fact was measuring input/output processing speed (factor 1). It is especially important that developers of new systems, who don't have the advantage of user experience that Britton Lee has and who therefore would tend to place more emphasis on one particular benchmark, understand clearly the system performance factors and assure themselves that the benchmark measures the factors they are interested in.

In general it appears that developing an application-independent benchmark is very hard, because such a benchmark would need to be representative of all the query/data/algorithm/hardware dependent performance factors of a DBMS, and that is difficult. Application-independent benchmarks, when carefully used, are useful for obtaining a relative comparison of certain system properties, but, because they are necessarily limited, give little information about throughput for specific applications.

Bibliography

[BITT83] Bitton, D., DeWitt, D.J., and C. Turbyfil, "Benchmarking Database Systems: A Systematic Approach," Computer Sciences Department Technical Report #526, Computer Sciences Department, University of Wisconsin, December 1983.

[BORA84] Boral, H., DeWitt, D., "A Methodology for Database System Performance Evaluation," Proceedings, SIGMOD, Boston, 1984.

Appendix A.

Query streams to create character databases:

```
open wiscbench
range of at is AtenKtup
range of ab is ABprime
create ABstring(
  unique1A  =uc20,
  unique2A  =uc20,
  twoA      =uc20,
  fourA     =uc20,
  tenA      =uc20,
  twentyA   =uc20,
  hundredA  =uc20,
  thousandA =uc20,
  twothousA =uc20)
go
```

```
append to ABstring(
  unique1A  =string(20,ab.unique1A + 10000),
  unique2A  =string(20,ab.unique2A + 10000),
  twoA      =string(20,ab.twoA + 10000),
  fourA     =string(20,ab.fourA + 10000),
  tenA      =string(20,ab.tenA + 10000),
  twentyA   =string(20,ab.twentyA + 10000),
  hundredA  =string(20,ab.hundredA + 10000),
  thousandA =string(20,ab.thousandA + 10000),
  twothousA =string(20,ab.twothousA + 10000))
go
```

```
create Atenstring(
  unique1D  =uc20,
  unique2D  =uc20,
  twoD      =uc20,
  fourD     =uc20,
  tenD      =uc20,
  twentyD   =uc20,
  hundredD  =uc20,
  thousandD =uc20,
  twothousD =uc20)
go
```

```
append to Atenstring(
  unique1D  =string(20,at.unique1D + 10000),
  unique2D  =string(20,at.unique2D + 10000),
  twoD      =string(20,at.twoD + 10000),
  fourD     =string(20,at.fourD + 10000),
  tenD      =string(20,at.tenD + 10000),
  twentyD   =string(20,at.twentyD + 10000),
  hundredD  =string(20,at.hundredD + 10000),
  thousandD =string(20,at.thousandD + 10000),
  twothousD =string(20,at.twothousD + 10000))
go
```

```
create unique clustered index on Atenstring (unique2D) with fillfactor=98,skip=1
```

```

go
create nonclustered index on Atenstring (unique1D) with fillfactor=98,skip=1
go
create unique clustered index on ABstring (unique2A) with fillfactor=98,skip=1
go
create nonclustered index on ABstring (unique1A) with fillfactor=98,skip=1
go

range of at is Atenstring
range of ab is ABstring

retrieve into Btenstring (at.all) go
retrieve into BBstring (ab.all) go
create unique clustered index on Btenstring (unique2D) with fillfactor=98,skip=1
go
create nonclustered index on Btenstring (unique1D) with fillfactor=98,skip=1
go
create unique clustered index on BBstring (unique2A) with fillfactor=98,skip=1
go
create nonclustered index on BBstring (unique1A) with fillfactor=98,skip=1
go
retrieve into CBstring (ab.all) go
retrieve into Ctenstring (at.all) go
create unique clustered index on Ctenstring (unique2F) with fillfactor=98,skip=1
go
create nonclustered index on Ctenstring (unique1F) with fillfactor=98,skip=1
go
create unique clustered index on CBstring (unique2A) with fillfactor=98,skip=1
go
create nonclustered index on CBstring (unique1A) with fillfactor=98,skip=1
go

- and so on, until -

create nonclustered index on PBstring (unique1A) with fillfactor=98,skip=1
go

```

BENCHMARKING DATABASE SYSTEMS IN MULTIPLE BACKEND CONFIGURATIONS *

Steven A. Demurjian and David K. Hsiao
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
(408)-646-2449

ABSTRACT

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

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

1. INTRODUCTION

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

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

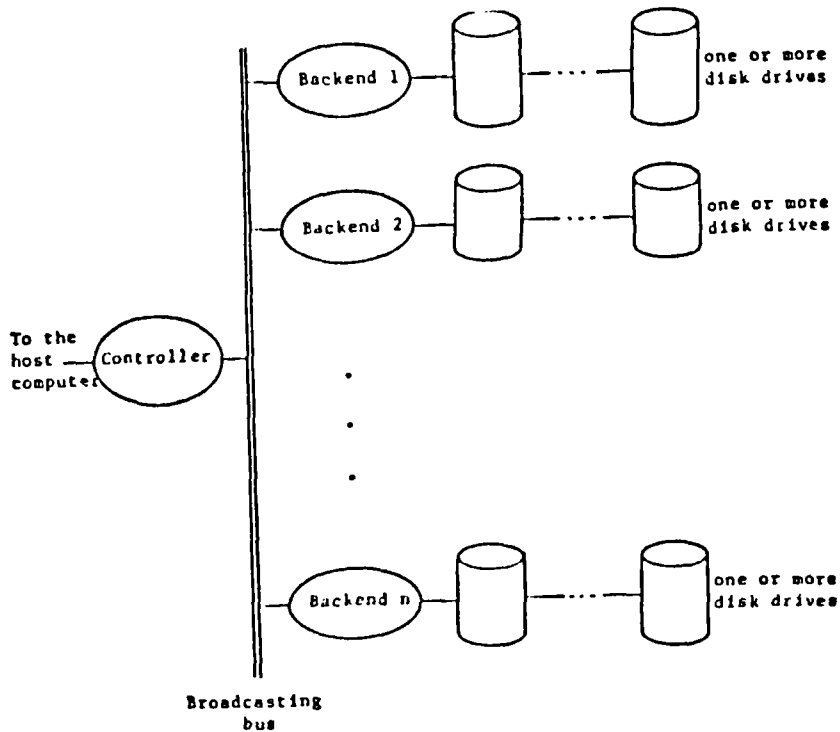


Figure 1. The MBDS Hardware Organization

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

In this paper we provide a preliminary evaluation of the validity of the MBDS performance claims. The main focus of this paper is on the external performance measurement of MBDS. The external performance measurement evaluates a system by collecting the response times of requests. External performance measurement is a macroscopic evaluation of the system. Ingres, Oracle, and the Britton-Lee IDM/500, have all been evaluated using external performance measurement techniques [Stra84, Schi84, Bitt83].

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

2. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

The current hardware configuration of MBDS consists of a VAX-11/780 (VMS OS) running as the controller and two PDP-11/44s (RSX-11M OS) and their disk systems running as backends. The disk system on each backend is an DEC RM02 disk drive, which has a 67MB formatted capacity, a peak transfer rate of 806KB/s and an average access time of 42.5ms (30ms average seek time + 12.5ms average latency time). Inter-computer communication is supported by three parallel communication links (PCL-11Bs), which is a time-divisioned-multiplexed bus. An overview of MBDS can be found in [Kerr82]. The implementation efforts are documented in [He82, Boyn83b, Demu84]. MBDS is a message-oriented system (see [Boyn83a]). In a message-oriented system, each process corresponds to one system function. These processes, then, communicate among themselves by passing messages. User requests are passed between processes as messages. The message paths between processes are fixed for the system. The MBDS processes are created at the start-up time and exist throughout the entire running time of the system.

MBDS provides a centralized database system where the database itself is evenly distributed across the backend processors. Only a single copy of the database is stored. The underlying data model for MBDS is the attribute-based data model [Hsia70]. The attribute-based data model stores data in files of records. MBDS stores records of a file in clusters. A *cluster* is a group of records such that every record in the cluster satisfies the same set of attribute-value pairs or ranges. Thus, a file is divided into one or more clusters. The distribution of the database is accomplished using a cluster-based data placement algorithm.

The cluster-based data placement algorithm is arbitrated and managed by the controller. When a new cluster is defined, the backend processor notifies the controller. The controller then decides which backend will insert the new record. Under the direction of the controller, the chosen backend will continue to insert records of the new cluster, until the backend processor fills a block of secondary memory storage. When this occurs, the backend processor notifies the controller that the block is full. The controller then directs another backend for the insertion of new records of the cluster. In a multiple-backend configuration, the controller attempts to achieve a block-parallel-and-record-serial operation for any subsequent access to the records of the cluster.

Let's trace through an example. Suppose that our system has four backend processors, the average size of a record is 200 bytes, and the size of a block of secondary storage memory is 4K (so, each block contains approximately 20 records). A new cluster of 100 records, say C, is defined. The controller picks say, Backend 3, for inserting records of cluster C. Backend 3 will insert 20 records into a block for the cluster C under the direction of the controller. Then the controller will have Backend 4 insert records of cluster C. After Backend 4 has inserted 20 records, the controller will cycle to Backend 1, and continue the round-robin process until all 100 records are placed on the secondary storage blocks. For the next new cluster, say, C', the controller will then pick Backend 4, since Backend 3 is the last backend used by the previous cluster in the algorithm.

3. THE BENCHMARK STRATEGY

In this section we analyze the basic benchmark strategy for the preliminary performance evaluation of the multi-backend database system. The benchmark strategy focuses on collecting macroscopic measurements on the systems performance. Macroscopic measurements correspond to the external performance measurement of the system, which collects the response time of requests that are processed by the system. To adequately conduct the external performance measurement of the system, software was developed to collect timing information and data. The performance software was bracketed in conditional compilation statements to facilitate an easy transition between a testing system and a running system.

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

3.1. The Test Database Organization and Testing Configurations

The test database was constructed using a record size of 200 bytes. A total of 24 clusters are defined for the test database. The virtual and physical memory limitations of each backend restricted the database size to a maximum of 1000 records per backend. This limitation, coupled with the need to verify the two performance claims, led us to the specification of three different system configurations for the MBDS performance measurements. Table 1 displays the configurations.

Test A configures MBDS with one backend and one thousand records in the test database. Test B configures MBDS with two backends and one thousand records split evenly between the backends. The transition from Test A to Test B is used to verify the first performance claim (see Section 1). Tests A and B have 23 clusters that contain 40 records and one cluster that contains 80 records. In Test A, all of the records are stored on the single backend. In Test B, each backend stores 20 records for the first 23 clusters and 40 records for the last cluster.

Test C also configures MBDS with two backends, but, the size of the database is doubled to two thousand records. The transition from Test A to Test C is used to verify the second performance claim (see Section 1). Test C has 23 clusters that contain 80 records each and one cluster that contains 160 records. In Test C, each backend stores 40 records for each of the first 23 clusters and 80 records for the last cluster. Notice that the record totals per cluster per backend are the same for Test A and Test C.

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

Table 1. The Measurement Configurations

3.2. The Request Set

In this section we review the retrieve requests that are used to benchmark MBDS. The retrievals, shown in Table 2, are a mix of single and double predicates. There are two directory attributes and thirty-one non-directory attributes in each record. The directory attributes, INTE1 and INTE2, are integer-valued, and are used for the cluster definition and formation. INTE1 is defined using 5 attribute-value ranges, while INTE2 is defined using 24 attribute-value ranges. The non-directory attributes are used as fillers for the 200-byte record. The retrieve requests given in Table 2 are specified using equality and inequality predicates, to control the search space when accessing the database records.

In Table 3 we present a high-level analysis of the request set given in Table 2. We focus on specifying two characteristics for each retrieve request in the request set; the number of clusters examined by the particular retrieve request and the volume of the database information that is retrieved. The values in Table 3 apply to the three testing configurations, A, B, and C, with one exception. The numbers in parenthesis in the third column represent the number of records retrieved for Test C.

3.3. The Measurement Strategy, Statistics and Limitations

The basic measurement statistics used in the performance evaluation of MBDS is the response time of request(s) that are processed by the database system. The *response time* of a request is the time between the initial issuance of the request by the user and the final receipt of the entire request set for the request. The response times are collected for the request set (see Table 2) for each of the three configurations (see Table 1). Each request is sent a total of ten times per database configuration. The response time of each request is recorded. We determine that ten repetitions of each request produce an acceptable standard deviation. Upon completion of the ten repetitions for a request, we calculate the mean and the standard deviation of the ten response times. There are two main statistics that we calculate to evaluate the MBDS performance claims, the response-time improvement and the response-time reduction.

Request Number	Retrieval Request
1	(INTE1 = 10) or (INTE1 = 230)
2	(INTE2 =< 250)
3	(INTE2 =< 500)
4	(INTE1 =< 1000)
5	(INTE1 =< 200) or (INTE1 >= 801)
6	(INTE1 =< 400) or (INTE1 >= 601)
7	(INTE1 <= 201)
8	(INTE1 <= 401)
9	(INTE1 <= 201) or (INTE1 >= 800)

Table 2. The Retrieval Requests

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

Table 3. The Number of Clusters Examined and the Percent of the Database Retrieved

The *response-time improvement* is defined to be the percentage improvement in the response time of a request, when the request is executed in n backends as opposed to one backend and the number of records in the database remains the same. Equation 1 provides the formula used to calculate the response-time improvement for a particular request, where Configuration Y represents n backends and Configuration X represents one backend. The response-time improvement is calculated for the configuration pair (A, B). The configuration pair (A, B) is evaluated for the retrieve requests (1) through (9) (see Table 2).

$$\text{The Response Time Improvement} = 100\% * \left[1 - \left(\frac{\text{The Response Time of Configuration Y}}{\text{The Response Time of Configuration X}} \right) \right]$$

Equation 1. The Response-Time-Improvement Calculation

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

$$\text{The Response Time Reduction} = 100\% * \left[1 - \frac{\left(\frac{\text{The Response Time of Configuration Z}}{\text{The Response Time of Configuration X}} \right)}{\right]$$

Equation 2. The Response-Time-Reduction Calculation

Finally, we examine the limitations of the testing strategy. The last two versions of MBDS differ in the implementation of the directory tables. The newest version of the system, called Version F, implements the directory tables on the secondary storage. The previous version, called Version E, stored the directory tables in the primary memory. The major roadblock that we have encountered in the performance measurement of MBDS has been the hardware limitations of the backend processors (PDP-11/44). With only 64K of virtual memory per process and a total of 256K physical memory, we found that we could not increase the MBDS system parameters to permit an extensive test of the system on a large database. These restrictions have forced us to benchmark the primary-memory-based directory management version of the system which, excluding the directory table management routines, is nevertheless equivalent in functionality to Version F.

4. THE BENCHMARKING RESULTS

In this section, we present the results obtained from the performance measurement of MBDS. In particular, we review the results of external performance measurement, in the hope of verifying the MBDS performance and capacity claims. One final note, the units of measurement presented in the tables of this section are expressed in seconds.

Table 4 provides the results of the external performance measurement of MBDS. There are three parts to Table 4. Each part contains the mean and the standard deviation of the response times for requests (1) through (9), which are outlined in Section 3.2. The three parts of Table 4 represent three different configurations of the MBDS hardware and the database capacity. The first part has configured MBDS with one backend and the database with 1000 records on its disk. The second part has configured MBDS with two backends, with the database of 1000 records, split evenly between the disks of the backends. The third part has configured MBDS with two backends and with a database doubled of 2000 records, where the disk of each backend has 1000 records.

Given the data presented in Table 4, we can now attempt to verify or disprove the two MBDS performance claims. We begin by calculating the response-time improvement for the nine requests. In Table 5 we present the response-time improvement for the data given in Table 4. Notice that the response-time improvement is lowest for request (1), which represents a retrieval of two records of the database. On the other hand, the response-time improvement of request (4), which retrieves all of the database information is highest, approaching the upper bound of fifty percent. In general, we find that the response-time improvement increases as the number of records retrieved increases. This seems to support a hypothesis that even if the response set (therefore the database) is larger, the response-time improvement will remain at a relatively high level (between

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

Table 4. The Response Time Results

40 an 50 percent).

Next, we calculate the response-time reduction for each of the nine requests. In Table 6 we present the response-time reductions for the data given in Table 4. Notice that the response-time reduction is worst for request (1), which represents a retrieval of two records of the database. On the other hand, the response-time reductions for the requests which access larger portions of the database, requests (4) and (6), have only a small response-time reduction. In general, we found that the response-time reduction decreases as the number of records retrieved increases, i.e., the response time remains virtually constant. Again we seem to have evidence to support the hypothesis that, as the size of the response set increases for the same request, the response-time

Request Number	Response-Time Improvement (A,B)
1	36.07
2	45.14
3	46.53
4	48.94
5	47.27
6	48.81
7	45.93
8	47.79
9	47.21

Table 5. The Response-Time Improvement Between Configurations A and B.

reduction will decrease to a relatively low level (0.1% or less).

Request Number	Response-Time Reduction (A,C)
1	4.49
2	4.03
3	0.92
4	0.32
5	0.47
6	0.12
7	0.50
8	0.23
9	1.07

Table 6. The Response-Time Reduction Between Configurations A and C

5. CONCLUSIONS AND FUTURE WORK

We have shown that the two basic performance claims of the multi-backend database system are valid. While these results are preliminary, they are encouraging. Overall, the response-time improvement ranged from 36.07 percent to 48.94 percent, when the number of backends and their disks is doubled for the same database. The low end of the scale represented a request which involved the actual retrieval of only two records. The high end represents a request which has to access all of the database information. The response-time reductions were also impressive, ranging from a 4.49 percent change to a 0.12 change. In other words, when we double the number of backends and their disks, the response time of a request is nearly invariant despite the fact that the response set for the request is doubled. Another crucial discovery that we made was that the results were consistent and reproducible. The tests were conducted at least twice for most of the request set, with the testing done on different days by different people. The resulting data was consistent and reproducible. The data presented in this paper represents the last set of tests for the request set.

The next logical step in the performance evaluation of the multi-backend database system is to extend the testing to include the other request types, update, insert and delete. Additionally, there are still some more tests to run on the retrieval request. We also seek to provide some insight into the internal performance of MBDS. Internal performance measurement provides a microscopic view of the system, by collecting the times of the work distributed and performed by the system components, i.e., in our case, individual processes.

Because MBDS is intended for microprocessor-based backends, winchester-type disks and an Ethernet-like broadcast bus, we will not continue our benchmark work on the present VAX-PDPs configuration. Instead, we plan to download MBDS to either MicroVaxs or Sun Workstations. With either choice, we can utilize a broadcast bus, which was not available when the work began in 1981. We may also eliminate all the physical and virtual memory problems. In the new environment we can perhaps obtain

a more thorough benchmarking of MBDS, and study various benchmarking strategies.

ACKNOWLEDGEMENTS

We would like to thank all of those who have helped us with the performance evaluation of the multi-backend database system. Robert C. Tekampe and Robert J. Watson were involved with the development and implementation of the testing software for our system [Teka84]. Prof. Douglas S. Kerr and Dr. Paula R. Strawser provided valuable advice and assistance on the benchmark strategy. Finally, Albert Wong and the technical staff at the Naval Postgraduate School provided assistance with configuring the computer systems for testing.

REFERENCES

- [Bitt83] Bitton, D., DeWitt, D. and Turbytil, C., "Benchmarking Database Systems: A Systematic Approach," *Proceedings on Very Large Data Bases*, 1983.
- [Boyn83a] Boyne, R., et al., "A Message-Oriented Implementation of a Multi-Backend Database System (MBDS)," in *Database Machines*, Leilich and Missikoff (eds.), Springer-Verlag, 1983.
- [Boyn83b] Boyne, R., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management," Technical Report, NPS-52-83-003, Naval Postgraduate School, Monterey, California, March 1983.
- [Demu84] Demurjian, S. A., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part IV - The Revised Concurrency Control and Directory Management Processes and the Revised Definitions of Inter-Process and Inter-Computer Messages" Technical Report, NPS-52-84-005, Naval Postgraduate School, Monterey, California, March 1984.
- [He82] He, X., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part II - The First Prototype MBDS and the Software Engineering Experience," Technical Report, NPS-52-82-008, Naval Postgraduate School, Monterey, California, July 1982; also appeared in *Advanced Database Machine Architecture*, Hsiao (ed.), Prentice Hall, 1983.
- [Hsia70] Hsiao, D. K., and Harary, F., "A Formal System for Information Retrieval from Files," *Communications of the ACM*, Vol. 13, No. 2, February 1970, Corrigenda, Vol 13., No. 4, April 1970.
- [Hsia81a] Hsiao, D.K. and Menon, M.J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

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

[Kerr82] Kerr, D.S., et al., "The Implementation of a Multi-Backend Database System (MBDS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MBDS," Technical Report, OSU-CISRC-TR-82-1, The Ohio State University, Columbus, Ohio, January 1982; also appeared in *Advanced Database Machine Architecture*, Hsiao (ed.), Prentice Hall, 1983.

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

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

[Teka84] Tekampe, R. C., and Watson, R. J., "Internal and External Performance Measurement Methodologies for Database Systems," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

TRANSACTION ACCELERATION

Timothy Chou
Jim Gray

Tandem Computers, Inc.
19333 Vallco Parkway
Cupertino, CA 95014

TRANSACTION ACCELERATION

1.0 INTRODUCTION

Today corporations are faced with a myriad of vendors providing transaction processing systems. When a corporation is willing to commit considerable resources to a large application it is important to have confidence that the application will perform adequately on that vendor's system. A traditional method to insure that performance goals are met is to specify a representative benchmark, run it on each vendor's system and compare the results [GLES81]. While this is certainly an accurate method, it both time consuming and expensive. Initially, a customer may only be interested in the relative capabilities of the various vendor's systems. With a single standard metric for transaction processing performance, systems which are totally incapable of supporting the desired workload could be removed from consideration early in the procurement cycle.

The scientific marketplace has had the same requirement. In the 1970's the Whetstone benchmark [CURN74,WICH75] was developed by the Central Computer Agency (CCA) of the British Government as a standard scientific benchmark. Today practically every manufacturer of systems for the scientific processing marketplace advertizes, or at least knows, the number of Whetstones/sec the machine is capable of.

In the transaction processing marketplace one standard performance metric is transactions per second (TPS), the throughput of a standard transaction at a given response time. The transaction definition is typically the DebitCredit transaction. This is also sometimes referred to as the TP1 or ET1 transaction [TAND85].

While TPS is a valid metric of system performance, it doesn't tell the whole story. This paper introduces a new metric, transaction acceleration, which gives the customer a new, different and useful characterization of transaction processing system performance.

2.0 RESPONSE TIME

Response time is the time the system takes to process a transaction. One may speak of a minimum, average or maximum response time. Response time curves usually have long tails because of anomalies such as lock waits, system checkpoints, operator tasks, etc., performance metrics usually use the 90% or 95% response time -- 95% of the transactions have response time less than this.

To eliminate the issue of communication line speed and delay, response time is typically defined as the time interval between the arrival of the last bit from the communication line and the sending of the first bit to the communication line. This is the metric used by most transaction processing stress testers [ENCO83].

In evaluating a system's performance it is often interesting to plot the response time curves. This curve easily illustrates that the minimum response time occurs when the system is running a very low load. As the load grows, response time increases, slowly at first but eventually the system saturates and response times approach infinity.

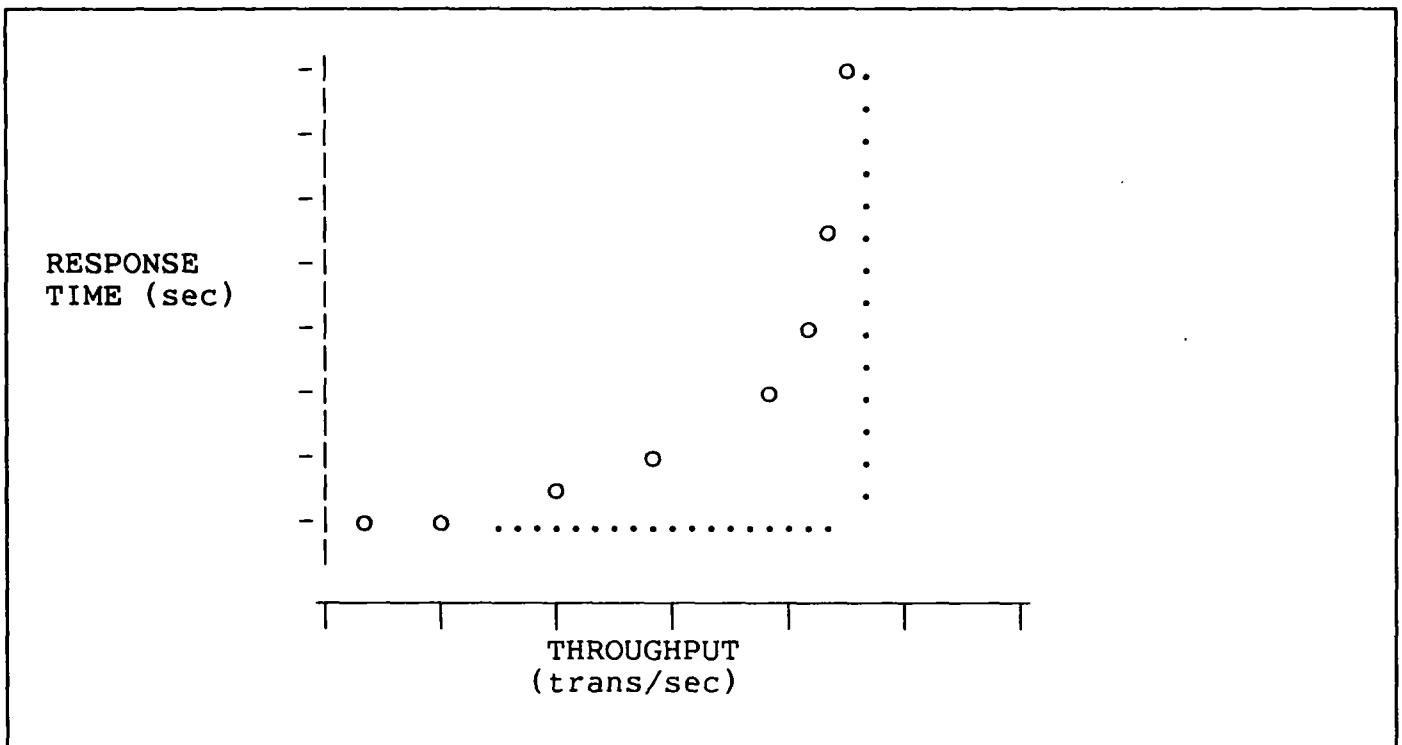


FIGURE 1. TRANSACTION RESPONSE TIME

In Figure 1, the horizontal dotted line is the minimum response time. The vertical dotted line is the maximum throughput of the system measured in transactions per second.

The traditional TPS rating is obtained by defining the maximum throughput when 95% of the transactions have a response time of less than 1 sec. The computation of the TPS rating is shown in Figure 2.

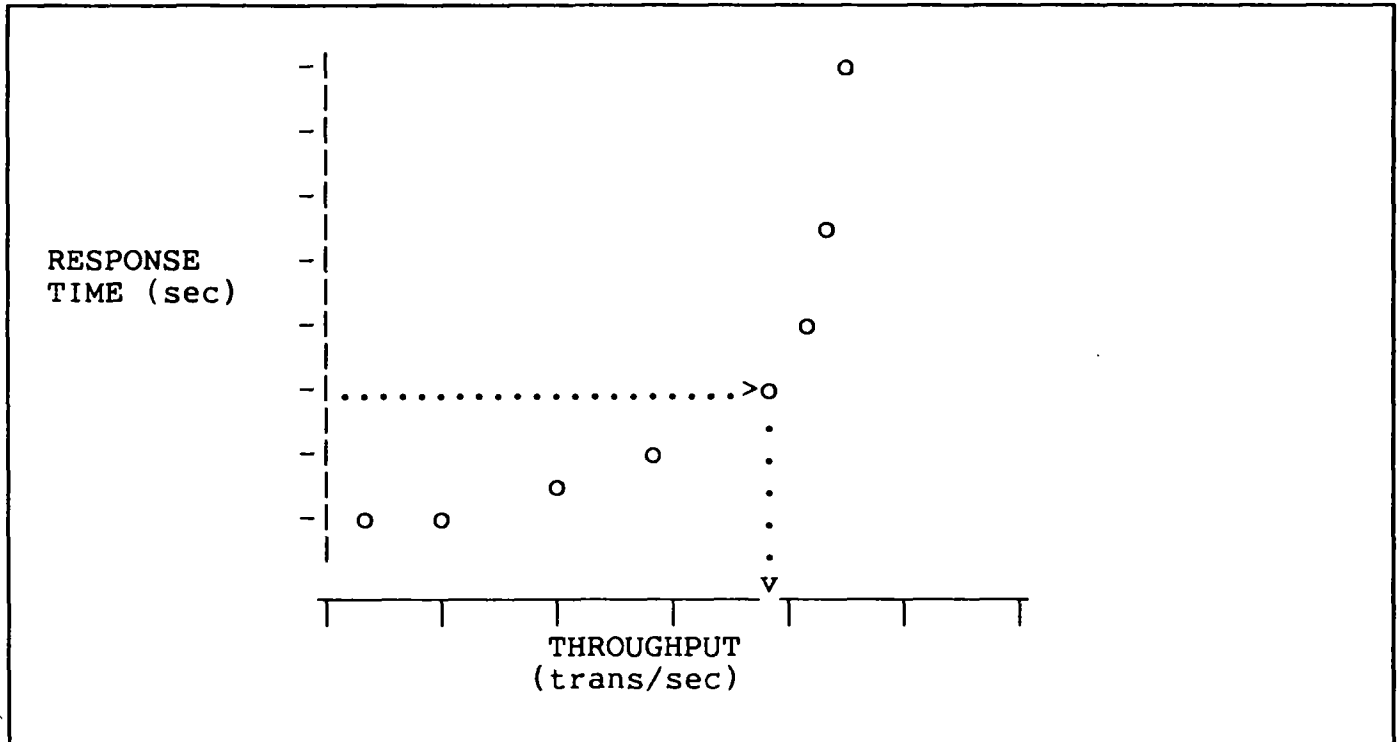


FIGURE 2. COMPUTING TRADITIONAL TPS SYSTEM RATING

3.0 TRANSACTION ACCELERATION

Transaction acceleration is defined as the slope of the curve which plots transaction throughput on the Y-axis and response time on the X-axis. It is called transaction acceleration because the units are transactions/sec/sec. Transaction acceleration can be computed from the following steps.

STEP 1: Compute the response time curve for the DebitCredit transaction and database on the system.

STEP 2: Transpose the curve to get throughput vs. response time rather than response time vs. throughput.

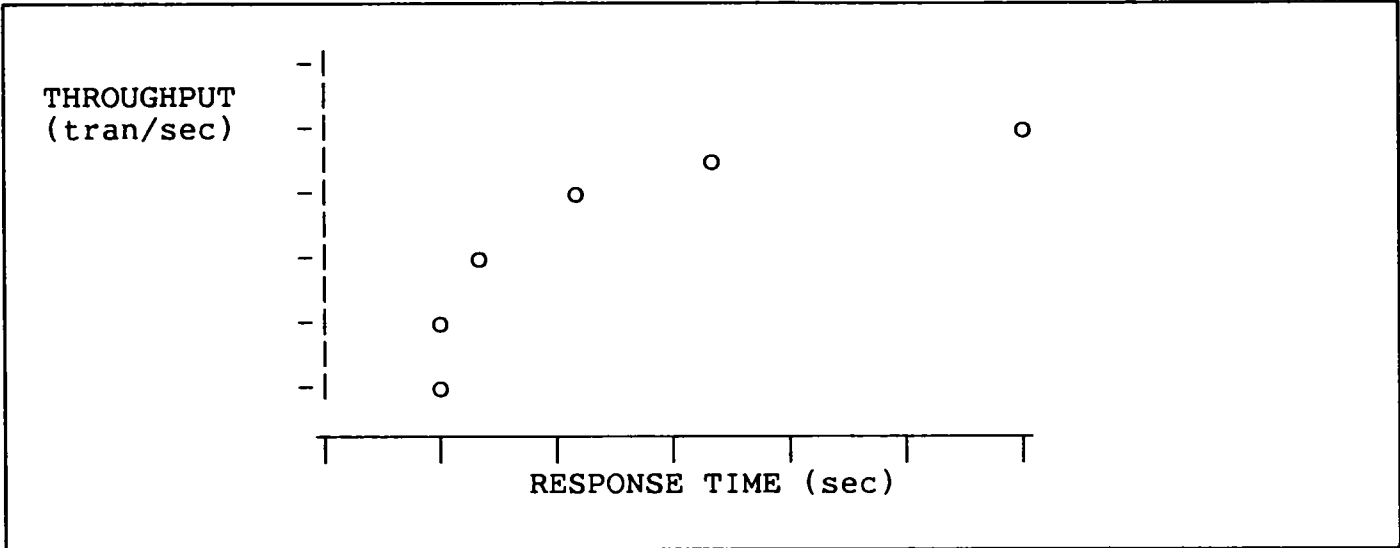


FIGURE 3. THROUGHPUT VS. RESPONSE TIME

STEP 3: Plot the derivative of the resulting curve to give transaction acceleration.

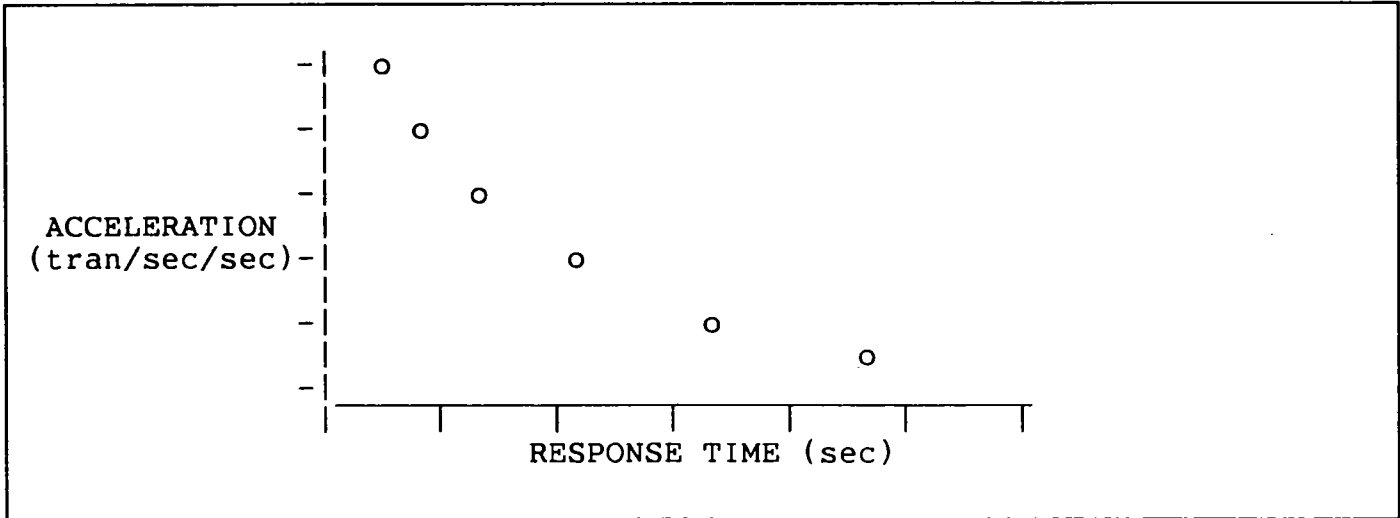


FIGURE 4. TRANSACTION ACCELERATION VS. RESPONSE TIME

In Figure 4, the initial transaction acceleration of this system is infinite (because response time stays steady at first). But

eventually, the acceleration declines to zero. If throughput declined when the system saturated, then acceleration might become negative.

So why is transaction acceleration a good system performance metric? Consider the following graph of transaction throughput for three vendor's systems.

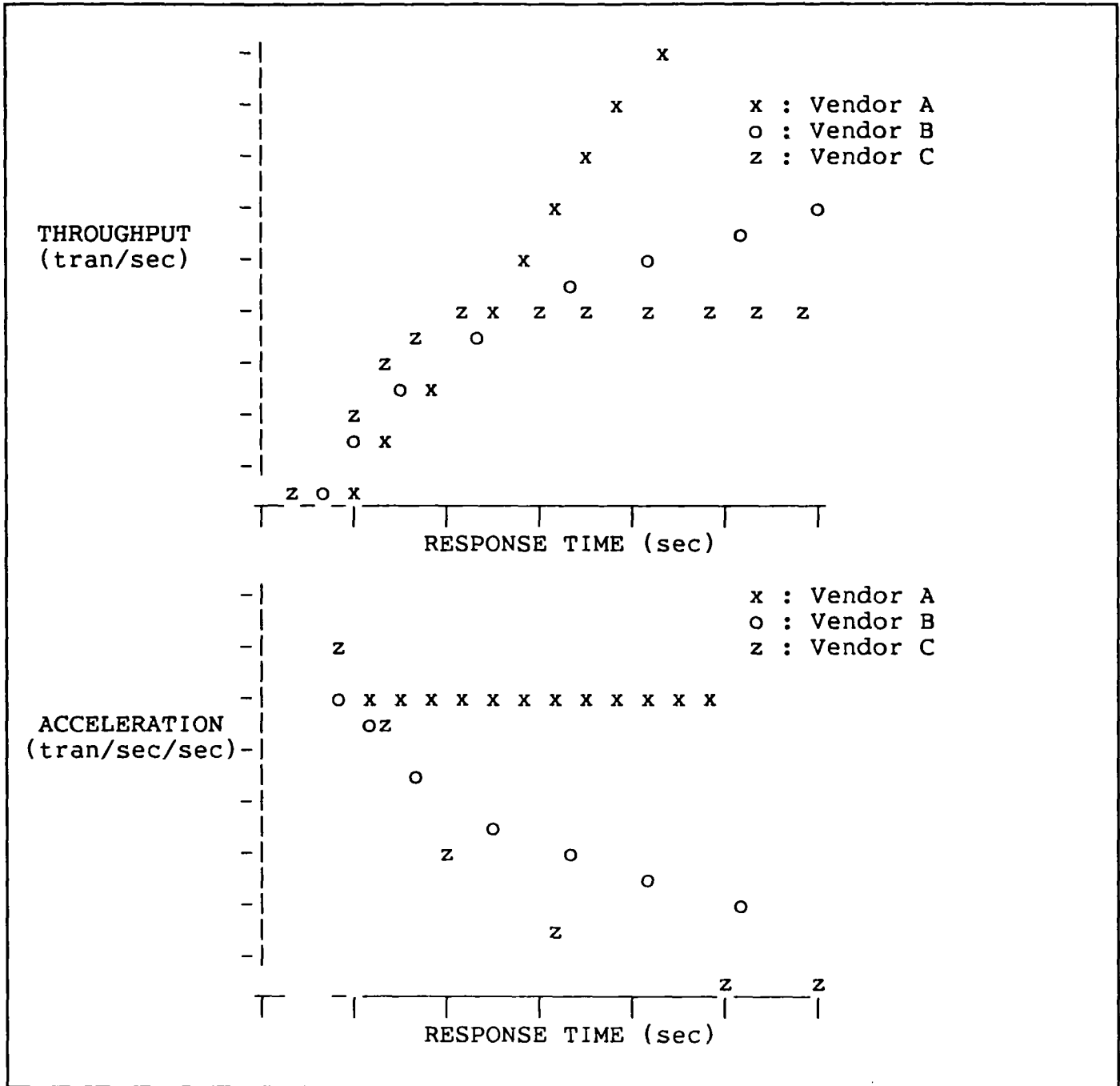


FIGURE 5. TRANSACTION ACCELERATION OF THREE SYSTEMS

If we assume that the intersection of the three curves is at a response time of 1 second then all three systems have the same transaction throughput, and therefore the same TPS rating. Using the TPS metric there is no difference between the three systems. However,

that is clearly not the case. Vendor A's system is clearly a better one when we look at the transaction acceleration curves -- as the transaction workload increases system A will suffer the least degradation in response time. In other words, if you have to "step on the gas" system A is going to be "more responsive". Since transaction processing workloads have a large dynamic range, it is clear that knowing how much "headroom" you have is important. Only the transaction acceleration metric captures this aspect of system performance.

4.0 THROUGHPUT vs. RESPONSE TIME CURVES

While transaction acceleration is a metric which characterizes transaction processing performance the throughput vs. response time curves are also interesting in their own right. They provide another way of looking at the implication of many other transaction processing design issues. For example, a vendor's profile of a system could also include the following curves:

- o Single point of failure curves
- o Fixed cost curves

A single point of failure curve could easily show to what degree performance would suffer if there were such a failure. A hypothetical example is shown below in Figure 6. Note that Vendors B and C do not have fault-tolerant configurations.

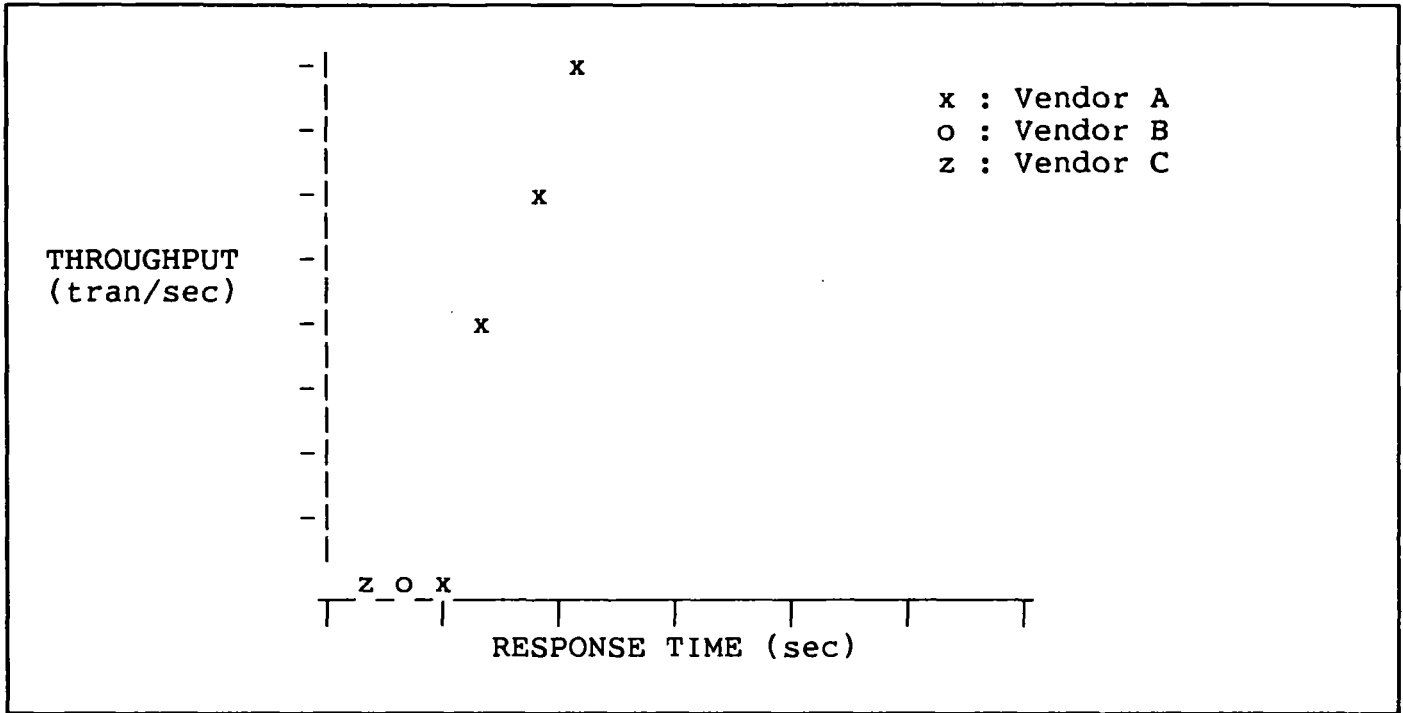


FIGURE 6. SINGLE PROCESSOR FAILURE

The capability to incrementally expand a transaction processing system is important. A fixed cost transaction acceleration curve can illustrate what the units of expandability are. Figure 7 and 8 show two representative cost curves.

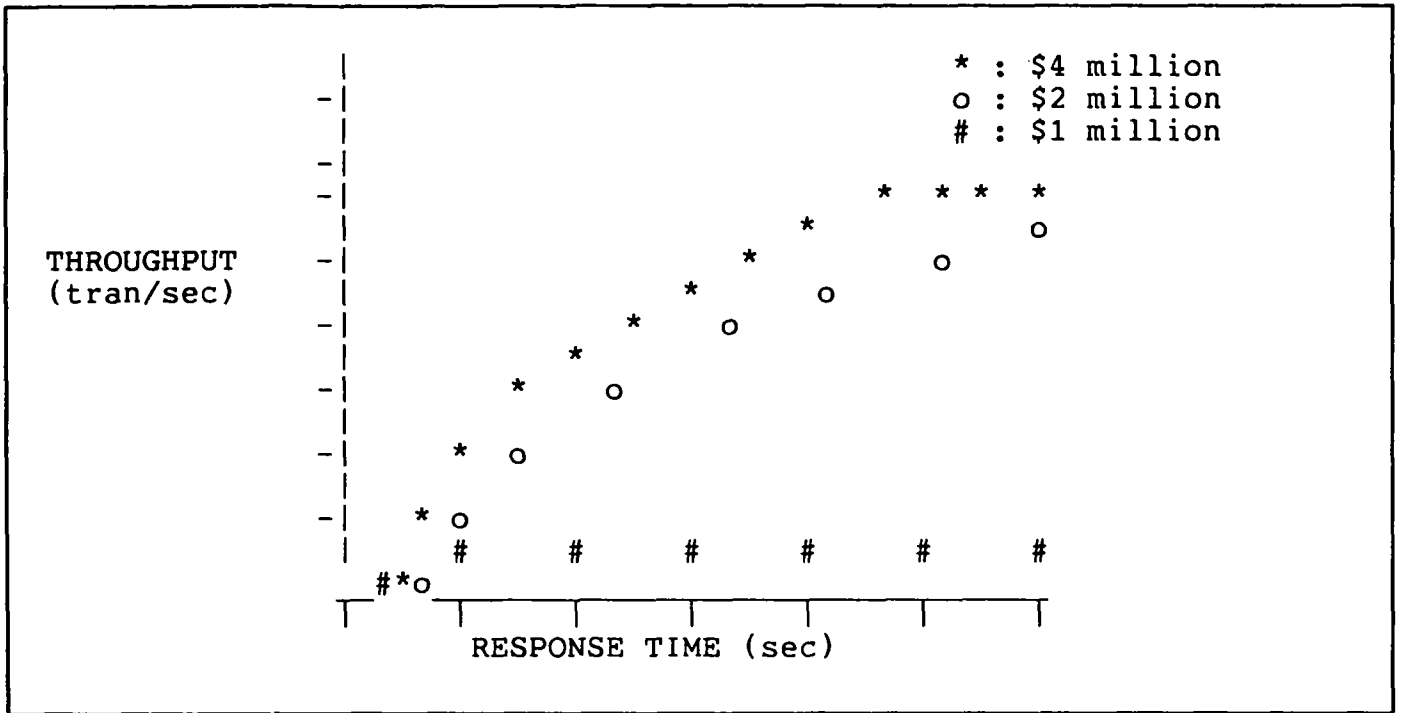


FIGURE 7. VENDOR A

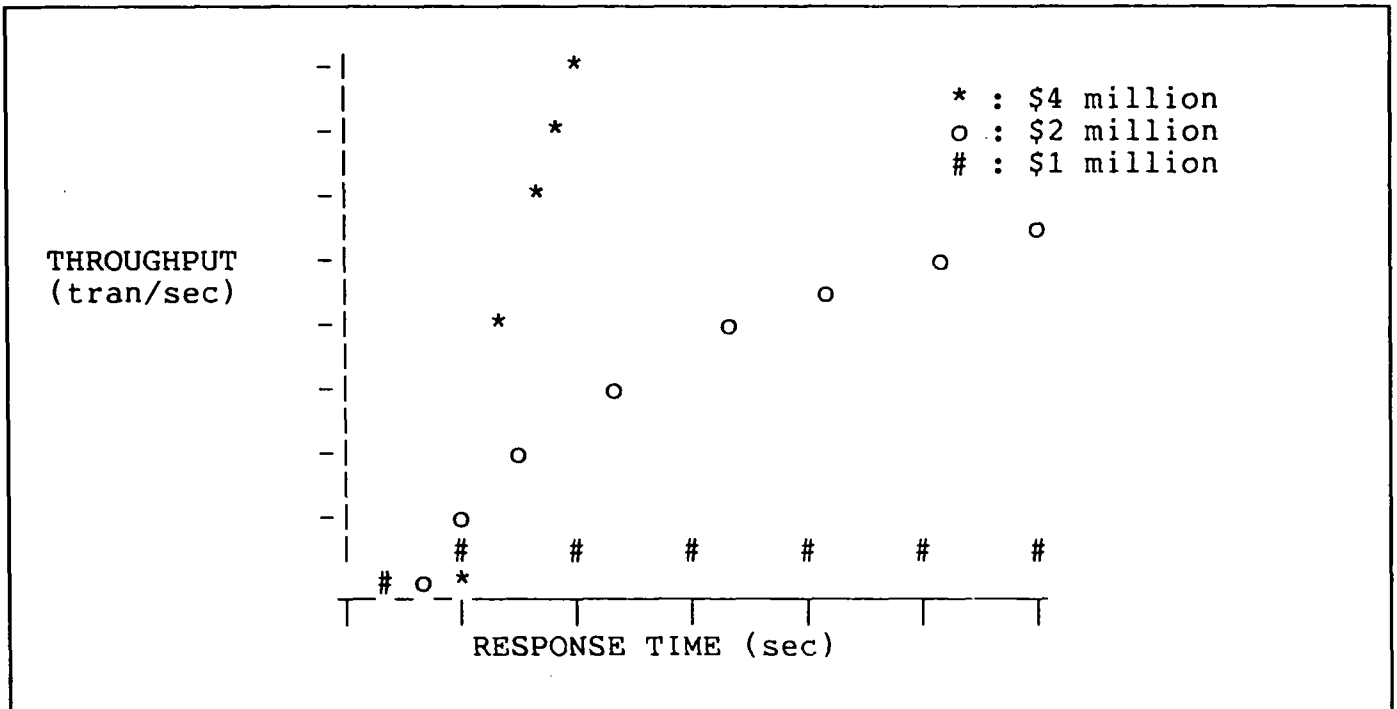


FIGURE 8. VENDOR B

Another proposal [REUT84] has suggested plotting K\$/TPS against response time. A curve like this could tell how much you have to pay for decreasing response time below a given level, or conversely how much you can gain by letting it increase.

4.0 SUMMARY

There have been and continue to be a number of methods of characterizing computer system performance. While benchmarks, simulations and analytic models can produce performance numbers there is still a need for standard metrics to be able to compare systems from various vendors. Transaction acceleration provides a new and useful metric of system performance.

REFERENCES

- [CURN74] Curnow, H.J. and Wichmann, B.A., "A Synthetic Benchmark," The Computer Journal, Volume 19, No. 1, ppgs 43-48.
- [ENCO83] "ENCORE User's Guide", Tandem Computers Part 82350, April 1983.
- [GLES81] Gleser, M.A., Bayard, J., Lang, D.D., "Benchmarking for the Best," Datamation, May 1981.
- [GRAY78] Gray, J. , "Notes on Database Operating Systems", Lecture Notes in Computer Science, vol. 60, Bayer-Seegmuller editors, Springer Verlag, 1978.
- [GRAY85] Gray, J. et al., "One Thousand Transactions Per Second", To appear in the Proceedings of the IEEE COMPCON-85, San Francisco.
- [REUT84] Reuter, A., private communication.
- [TAND85] "A Measure of Transaction Processing Power," to appear in Datamation.
- [WICH75] Wichmann, Brian A., "The Design of Synthetic Programs-1" Computer Evaluation and Measurement, John Wiley and Son, New York, ppgs 89-98.

TRANSACTION ORIENTED PERFORMANCE ANALYSIS OF DATABASE MACHINES

Margaret H. Eich

Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275
(214) 692-3087 eich%smu@csnet-relay

Abstract. A new technique for evaluating the performance of database machines (DBM) is proposed. In previous studies, performance analysis has been based on a single transaction, a reference string for transactions, or read only transactions. These approaches are not satisfactory in a DBM environment when subtransactions may be executed on different processors, and the interrelationships between operations within and across transactions can greatly effect the DBM performance. An approach for DBM performance analysis based upon a coordinated view of transactions, hardware, software, and databases is described.

1. Introduction

The need for performance evaluation measures and techniques applicable in a *Database Machine* DBM environment has been previously expressed [VEMU80], and there have been several studies evaluating the performance of various DBM designs ([BANE78], [DEWI81], [HAWT82], and [OZKA77]). These previous studies have had a very simplistic view of database transactions with little concern for the actual transactions to be executed in the target DBM. Analysis is based on a single transaction, a reference string for transactions, or read only transactions. These approaches are not satisfactory in a DBM environment where subtransactions may be executed on different processors, and the interrelationships between operations within and across transactions can greatly effect the DBM performance. As stated by Vemuri, the overall goal of DBM research is to "improve the performance and capability of the total system." Performance studies should attempt to measure total system performance.

This paper introduces a technique which can be used in DBM performance analysis research. Of concern is the technique to be used for simulation, not the measures gathered. The next section discusses some of the problems with previous approaches to DBM performance analysis. The third section describes the proposed technique, while the fourth section discusses the application of this approach in the simulation of a new DBM concurrency control technique.

2. Previous Studies

Based upon transaction representation, previous DBM performance studies can be classified as single query ([BANE78], [BANE79], [DEWI81], [HAWT82], [OZKA75], and [OZKA77]), reference string ([AGRA83] and [WILK81]), or read only [BORA81]. Single query studies determine performance analysis based on the execution of a single query or operation. These studies ignore the interrelationship between transactions. Analysis techniques usually fall into this category. Performance studies in the second class view

transactions as a sequence of page (record, tuple) references. This view overlooks variations in the time at which different operations in the same transaction actually make page references. The third view of transactions only sees transactions as performing retrieval operations, ignoring the impact of update operations. This view is sometimes taken because updates take a relatively short time to execute, and are often implemented very similar to retrieval operations ([DEWI81] and [HAWT82]). Even if these are true, the impact of update operations on overall system performance is not unimportant.

There are five major reasons why these simplistic views of transactions are unacceptable for DBM performance analysis:

1. Total Performance
2. Subtransaction Execution
3. Protocol Evaluation
4. Cross Transaction Effect
5. Bottleneck Analysis

To satisfactorily determine overall DBM system performance, the ability to model portions of transactions as well as to determine the effect that transactions have on each other must be examined.

Good performance of the total DBM system is often listed as the overall goal for the development of DBMs [VEMU80]. To determine total DBM performance, any performance analysis study should examine the execution of multiple concurrent transactions. Obtaining performance statistics based upon a single transaction, or read only transactions does not necessarily indicate the overall system performance with multiple concurrent update and retrieval transactions. It has been previously observed that input to DBM performance analysis should "reflect a wide range of transactions to determine the suitability of the various machines to different transaction types" [BORA81]. Previous studies have provided valuable insight into the performance of various DBM architectures for certain types of transactions, but they have not provided any indication of total DBM performance for concurrently executing potentially conflicting transactions.

Some DBM architectures require the processing of portions of transactions on different processors ([CESA83] and [DEWI79]). To accurately determine the impact of protocol overhead on the performance, it needs to be simulated at the point in time that it will occur in actual implementation. A more detailed view of transactions would facilitate the placement of protocol overhead within a transaction execution as well as providing the capability for simulating execution of portions of transactions on different processors. Thus a better estimate of performance could be obtained.

When estimating DBM performance, the impact of transactions on each other is crucial. One of the performance drawbacks of conventional sequential von Neumann machines is their lack of appropriateness for the "parallel process of data manipulation" [SU80]. DBMs should be designed to provide efficient storage, retrieval, management, and update of large databases with concurrent access [HSIA79]. The application of database machines will primarily be in large multiuser environments where the issues of integrity and synchronization are crucial. Estimates of overall DBM performance can not be obtained with any accuracy if the impact of multiple transactions are not examined.

The last reason that previous views of transactions have been inadequate when estimating DBM performance analysis, is that the identification of potential bottlenecks can not be accurately determined. Timing of different events is crucial in identifying

system bottlenecks. Overly simplified views of transactions do not provide realistic estimates of the time when different events occur. More detailed descriptions of transactions give a better estimate of these times and can thus be used to more accurately predict potential system bottlenecks.

3. Proposed Technique

This section describes a technique for conducting a DBM performance analysis simulation using a more realistic view of transactions than is found in previous DBM performance studies. This method provides the ability to execute subtransactions while still allowing the ability to determine the impact of total transactions on system performance. The relational data model using relational algebra operations is assumed. This transaction oriented method requires a detailed representation for transactions, the definition of various mixes of transactions, as well as the description of other system and workload parameters needed for simulation. When using a more realistic view of transactions, other components of the DBM must still be appropriately described. Thus, this performance analysis approach is based upon a coordinated view of transactions, hardware, software, and databases in the target DBM being analyzed. TABLE 1 lists functions and parameters needed to describe these four DBM components during a simulation. These lists are not intended to be all inclusive, but to indicate the types of parameters needed.

TABLE 1
SIMULATION PARAMETERS

Software	Hardware	Transaction	Database
Processor Allocation Method	Number of Processors	Number	Number
Operation Processing Times	Number of I/O Devices	Description	Access Method
Recovery Technique and Times	Type of Network	Select Probability	Size
Concurrency Control Times	Network Transmission Times	Update Probability	Location
Preprocessing Time	I/O Times	Relations Accessed	
Commit Processing Time			

The software functions described should include such things as the processor allocation method in an MIMD architecture, description of concurrency control and recovery techniques used, and processing times for various CPU operations. The hardware parameters include descriptions and quantity of the different hardware equipment, description of network used, and times for I/O and network processing. The structure of all relations considered needs to be described. This should include the number and access methods to be used for the various relations, as well as their size and location. Transaction structures need to be explicitly described including all database operations and relations accessed. During simulation runs, the probability of actually selecting and updating the examined pages (records, tuples) needs to be included. While we feel that the software, hardware, and database components are often adequately described in DBM performance studies, transactions are not. Therefore, the remainder of this section discusses a possible method to be used for representing transactions.

During transaction execution in a database system, a transaction request is often compiled into a *query parse tree* (QPT) [DEWI79]. A query parse tree is a directed graph where the nodes define the database operations to be performed and the arcs show the precedence relationships between the operations. During optimization, the trees may be converted into query dags by combining common subexpressions. It is assumed that each node in the dag completely describes the operation to be performed, including the databases acted on and the selection criteria for the operation. Figure 1 shows a query parse tree for a multirelation query used in a previous performance study [HAWT82]. Input databases have been shown as source nodes labelled by the relation name, and output data is shown by a sink node labelled OUTPUT. Query dags will be the basis for our detailed representation of transactions.

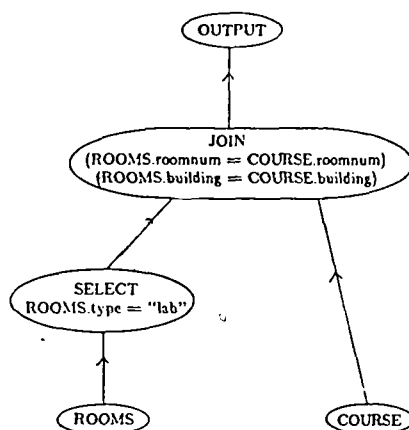


Fig. 1. - Sample Query Parse Tree

Both retrieval and update operations can be classified according to the number of input relations (and thus input arcs). A *simple* operation has one, while a *complex* operation has two. Simple operations include select, project, modify, insert, and delete, while complex operations include union, difference, product, and join. A complex version of the three types of update operations (modify, insert, delete) may also be considered. Here the target tuples (records) to be updated are identified by the tuples on the second input arc. For simulation purposes, the precise definition of the operations is not usually required. All that is really needed is an indication of the type of operations to be executed. Therefore, instead of specifically describing a transaction by a query dag, we use a *transaction skeleton*. The transaction skeleton for a transaction contains the same graphical structure as its associated query dag, but does not describe the operations in any detail, nor does it identify precise input relations. Nodes simply identify the type of operation (simple retrieval, simple update, complex retrieval, complex update) and input relation nodes are assigned unique integers. During simulation, the integers will be replaced with one of the relations assumed in existence. An example of the transaction skeleton for the query parse tree in Figure 1 is shown in Figure 2. The source nodes are labelled 1 and 2 to indicate two potentially different input relations.

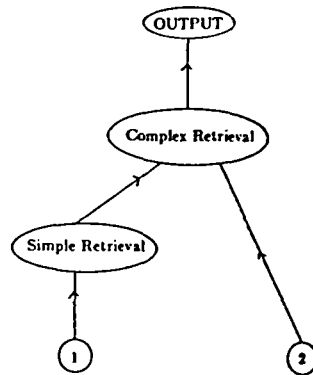


Fig. 2. - Transaction Skeleton for QPT in Figure 1

During simulation, a transaction skeleton is used to direct the simulation processing functions. The graph identifies precedence relationships between the various operations performed during the simulated execution of the transaction, and the node types identify time to be simulated for the operation. In actual use, a transaction skeleton may contain more nodes than in the associated query dag. Additional nodes would be added for any required processing activities needed during transaction execution such as concurrency control, data transmission between sites in a distributed system, or scheduling overhead. The placement of the nodes in the graph indicates the time during execution when the associated function would be performed. Addition of these nodes to the skeleton allows a more precise representation of processing overhead requirements than simply showing the database operations and assuming a fixed overhead for each.

To describe all transactions to be executed during simulation, a mix of transaction skeletons is defined by identifying the number of occurrences of each transaction skeleton to be included. Determining the exact mix of transactions to be examined in a DBM performance analysis study is not an easy task due to the fact that there is no accepted set of representative transactions. Previous investigations of the usage of different database operations have resulted in identification of the types of transactions most often used in the specific environment involved, but they have not identified a standard set of transactions which could be used for performance analysis ([EAST75] and [JOYC83]). The exact mix of transactions to be examined should be determined based upon the intended environment. If the target environment is not well understood, then the transactions to be included should be representative of many different types.

To perform a simulation experiment, each transaction to be executed must be defined by creating its transaction skeleton. Various mixes should then be described which use these skeletons. During simulation, the precise definition and order of transactions within a mix is determined. The order of transactions can be determined by assuming any distribution of the total number of transactions as long as the correct number of each type is included. The precise definition of each skeleton occurrence is obtained by replacing the label of each source node in the skeleton with the name of one of the relations being considered. To obtain different degrees of conflict across transactions, different distributions for assigning the relation names could be used. For the same transaction mix, different runs could yield extremely different results based upon the order of transactions and relations involved.

4. Example of Use

A new database concurrency control mechanism utilizing specialized data flow graphs, *database flow graphs* (DBFG), has been introduced and shown to perform as well or better than locking in an MIMD DBM environment ([EICH84a] and [EICH84b]). A database flow graph is a special type of data flow graph used to show dependencies between database operations (both intra-transaction and inter-transaction). The dependencies shown in a DBFG can be used to define a multiple transaction schedule of database operations which is serializable. With DBFG scheduling, concurrency control is implicitly obtained by ensuring that all schedules of database operations are valid data flow schedules.

The performance of DBFG scheduling was evaluated both analytically and through simulation [EICH84a]. The simulation utilized the transaction oriented approach to gather performance statistics. Software parameters used represented CPU processing times for transaction preprocessing, concurrency control, database operation execution, and commit processing. Hardware parameters described the number of secondary storage devices and query processors, as well as the I/O times needed for page access. The database structure consisted of fifteen relations, each with a predefined number of pages. Any given simulation run could uniformly change the number of pages for all relations by providing a page multiple parameter. The actual number of pages per relation was then calculated by multiplying the base number of pages by the page multiple.

Due to the fact that this was a general simulation with no knowledge of typical transaction mixes, a general set of transactions was created representing different types and levels of complexity. The twelve transaction skeletons used are shown in TABLE 2. Twenty mixes of these twelve transactions were defined by identifying the percentage of each transaction included. Depending on the experiment, some or all mixes were used. Some mixes were constructed based upon input to previous simulation experiments ([BORA81] and [HAWT82]), some were based upon reasonable and extreme combinations of transactions, and still others were based on the actual usage patterns of different operations in specific database systems ([EAST75] and [JOYC83]).

TABLE 2
TRANSACTION SKELETONS

Simple Retrievals	(SRet 1) (SRet (SRet 1))
Simple Updates	(SUpd 1) (SRet (SUpd 1))
Complex Retrievals	(CRet (SRet (SRet 1)) (SRet 2)) (CRet (SRet (SRet 1)) (SRet (SRet 2))) (CRet (SRet 1) (SRet 2)) (CRet (CRet (SRet 1) (SRet 2)) (SRet 3)) (CRet (CRet (SRet 1) (SRet 2)) (CRet (SRet 3) (SRet 4)))
Complex Updates	(CUpd (SRet (SUpd 1)) (2)) (CUpd (SRet 1) (2)) (CRet (SRet (SRet (SUpd 1))) (SRet (SRet (SUpd 2))))

At run time, the total number of each transaction type was determined based upon the associated percentage and actual number of transactions desired for that run. The order of the transactions was determined assuming a uniform distribution of the transactions across the total number of transactions in the run. The number in the transaction skeleton indicating the relation, was replaced at run time with one of the actual relations based upon an input parameter giving the mean of an exponential distribution whose range is the set of relation numbers. An exponential distribution was used to mimic the behavior which is found when a small number of relations is accessed more than the remaining ones. During execution, selection of a page for retrieval or update was made based upon an input parameter stating the mean of an exponential distribution from 0 to 1.

Six experiments were performed. The objectives of the first experiment were to determine the effect of the number of transactions on performance measures and to determine the number of transactions required to reach a steady state. The next three experiments determined the impact on performance of differences in various workload parameters: number of secondary storage devices, number of pages per relation, and probability of conflict among transactions. Experiment five examined the effect of different transaction mixes on the performance measures. The last experiment simulated the use of data flow processor allocation [BORA81] techniques across transactions.

The transaction oriented approach used in this simulation made the representation of different concurrency control processing locations within a transaction possible. A more simplistic view of transactions would not have allowed this definition of concurrency overhead during transaction processing. A simple yet flexible definition of transactions and transaction mixes provided the capability for an unlimited number of simulated transactions and mixes. While it is not implied that all aspects relating to DBM performance were included in this simulation, a more complex view of transactions was used than has been found in previous simulation experiments.

5. Summary

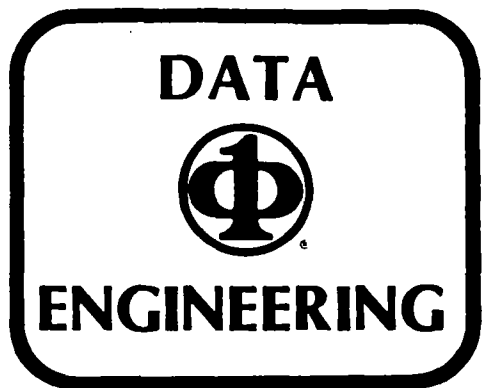
A transaction oriented approach to DBM performance analysis has been proposed. This technique requires a more precise representation of transactions than has been used by previous DBM performance studies. Used in conjunction with descriptions for the hardware, software, and database components of the DBM, this technique defines a coordinated approach to DBM performance analysis. The transaction orientation provides a more realistic predictor of DBM performance than would be possible using a simple reference string, a single transaction, or read only transactions.

6. References

- [AGRA83] Rakesh Agrawal, "Concurrency Control and Recovery in Multiprocessor Database Machines: Design and Performance Evaluation," Ph.D. Dissertation, University of Wisconsin-Madison, September 1983.
- [BANE78] Jayanta Banerjee and David K. Hsiao, "Performance Study of a Database Machine in Supporting Relational Databases," *Proceedings 1978 Very Large Data Bases Conference*, 1978, pp. 319-329.
- [BANE79] Jayanta Banerjee, David K. Hsiao, and Krishnamurthi Kannan, "DBC - A Database Computer for Very Large Databases," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June 1979, pp. 414-429.

- [BORA81] Haran Boral, "On the Use of Data-Flow Techniques in Database Machines," Ph.D. Dissertation, University of Wisconsin-Madison, April 1981.
- [CESA83] F. Cesarini, D. DeLuca Cardillo, and G. Soda, "An Assessment of the Query-Processing Capability of DBMAC," *Advanced Database Machine Architecture*, 1983, Printice-Hall Inc., pp. 109-129.
- [DEWI79] David J. DeWitt, "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," *IEEE Transactions on Computers*, Vol. C-28, No. 6, June 1979, pp. 395-406.
- [DEWI81] David J. DeWitt and Paula B. Hawthorn, "A Performance Evaluation of Database Machine Architectures," *Proceedings of the 1981 Very Large Databases Conference*, pp. 199-213.
- [EAST75] M. C. Easton, "Model for Interactive Data Base Reference String," *IBM Journal of Research and Development*, November 1975, pp 550-555.
- [EICH84a] Margaret H. Eich, *Concurrency in a Data Flow Database Machine*, PhD Dissertation, Department of Computer Science and Engineering, Southern Methodist University, August 1984.
- [EICH84b] Margaret H. Eich and David L. Wells, "Database Flow Graphs," *Proceedings of the 1984 International Conference on Parallel Processing*, August 21-24 1984, pp. 266-268.
- [HAWT82] Paula B. Hawthorn and David J. DeWitt, "Performance Analysis of Alternative Database Machine Architectures," *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 1, January 1982, pp. 61-75.
- [HSIA79] David K. Hsiao, "Data Base Machines Are Coming, Data Base Machines Are Coming!," *Computer*, March 1979, pp. 7-9.
- [JOYC83] John D. Joyce and David R. Warn, "Command Use in a Relational Database System," *AFIPS Proceedings National Computer Conference, 1983*, pp 247-253.
- [OZKA75] E. A. Ozkarahan, S. A. Schuster, and K. C. Smith, "RAP - An Associative Processor for Data Base Management," *National Computer Conference, 1975*, pp. 379-387.
- [OZKA77] E. A. Ozkarahan, S. A. Schuster, and K. C. Sevcik, "Performance Evaluation of a Relational Associative Processor," *ACM Transaction on Database Systems*, Vol. 2, No. 2, June 1977, pp. 175-195.
- [SU80] Stanley Y. W. Su, Hsu Chang, George Copeland, Paul Fisher, Eugene Lowenthal, and Stewart Schuster, "Database Machines and Some Issues on DBMS Standards," *AFIPS Proceedings National Computer Conference, 1980*, pp. 191-208.
- [VEMU80] V. Vemuri, R. A. Liuzzi, J. P. Cavano, and P. B. Berra, "Evaluation of Alternate Data Base Machine Designs," *Proceedings of the Fifthe Workshop on Computer Architecture for Non-Numeric Processing*, March 11-14 1980, pp. 29-38.
- [WILK81] William Kevin Wilkinson, "Database Concurrency Control and Recovery in Local Broadcast Networks," Ph.D. Dissertation, The University of Wisconsin-Madison, August 1981.

CALL FOR PAPERS



The Second International Conference on Data Engineering

Bonaventure Hotel
Los Angeles, California, USA
February 4-6, 1986

Sponsored by the  IEEE Computer Society

Committee

Honorary Chairman:

C. V. Ramamoorthy
University of California, Berkeley, CA

General Chairman:

P. Bruce Berra
Syracuse University, Syracuse, NY
(315) 423-4445

Program Chairman:

Gio Wiederhold
Dept. of Computer Science
Stanford University, Stanford, CA 94305
(415) 497-0685

Program Co-Chairpersons:

Iris Kameny, SDC, Santa Monica, CA 90406
Ming T. (Mike) Liu, Ohio State Univ., Columbus, OH 43210
Richard L. Shuey, Schenectady, NY 12309
Joseph Urban, Univ. S.W. Louisiana, Lafayette, LA 70504

Tutorials:

Benjamin Wah, Purdue
Peter Ng, Univ. of Missouri, Columbia, MO

Treasurers:

Lily Chow, IEEE, Silver Spring, MD 20910
Aldo Castillo, TRW, Redondo Park, CA 90278

Local Arrangements:

Walter Bond, Cal State Univ., Dominguez Hills, CA 90747
1000 East Victoria Street: (213) 516-3580/3398

Publicity:

Mas Tsuchiya, TRW Colorado Springs, CO 80916
1555 North Newport Rd: (303) 570-8376

SCOPE

Data Engineering is concerned with the role of data and knowledge about data in the design, development, management, and utilization of information systems. As such, it encompasses traditional aspects of databases, knowledge bases, and data management in general. The purpose of this conference is to continue to provide a forum for the sharing of experience, practice, and theory of automated data and knowledge management from an engineering point-of-view. The effectiveness and productivity of future information systems will depend critically on improvements in their design, organization, and management.

We are actively soliciting industrial contributions. We believe that it is critically important to share practical experience. We look forward to reports of experiments, evaluation, and problems in achieving the objectives of information systems. Papers which are identified as such will be processed, scheduled, and published in a distinct track.

TOPICS OF INTEREST

- Logical and physical database design
- Data management methodologies
- Distribution of data and information
- Performance Evaluation
- Design of knowledge-based systems
- Architectures for data- and knowledge-based systems
- Data engineering tools

We also are planning a special workshop track:

- Performance models and measurement of relational database systems

and solicit papers which report or evaluate such findings.

Awards, Student Papers, and Subsequent Publication:

An award will be given for the best paper at the conference.

Up to three awards of \$500 each to help defray travel costs will be given for outstanding papers authored by students.

Outstanding papers will be considered for publication in the IEEE Computer Magazine, the Transactions on Computers, and the Transactions on Software Engineering. For more information, contact the General Chairman.

Paper submission:


Four copies of papers should be mailed before July 1, 1985 to:

Second Data Engineering Conference
IEEE Computer Society
1109 Spring Street, Suite 300
Silver Spring, MD 20910
(301) 598-8142

Conference Timetable:

Manuscripts due: July 1, 1985
Acceptance letters sent: October 1, 1985
Camera-ready copy due: November 15, 1985
Tutorials: February 3, 1986
Conference: February 4-6, 1986


See you in Los Angeles!

For further information write to:
Second Data Engineering Conference 
 c/o IEEE Computer Society
 P.O. Box 639
 Silver Spring, MD 20901 USA
 (301) 589-8142
 TWX: 7108250437 IEEE COMPSO

Name _____

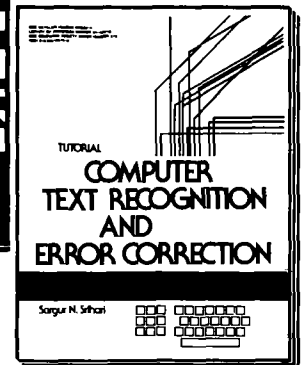
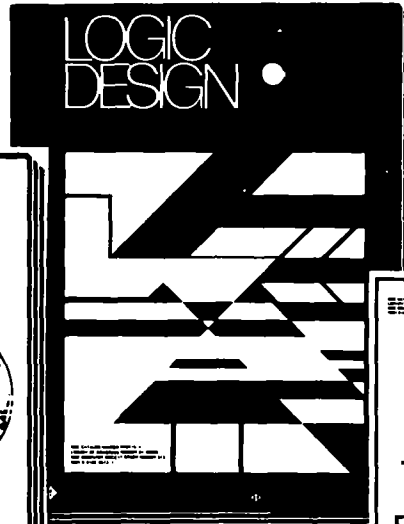
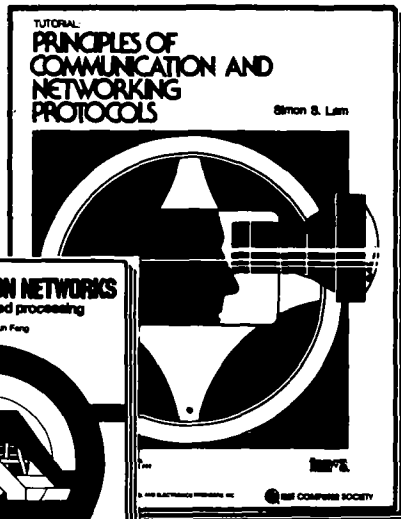
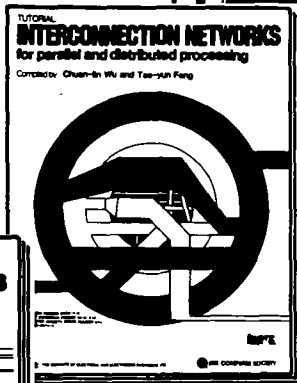
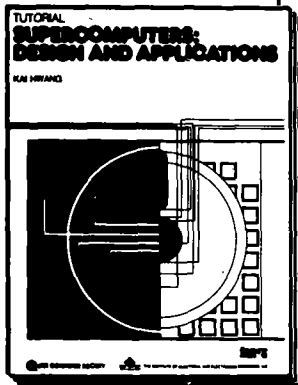
Affiliation _____

Address _____




THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

NEW Tutorials



THE
COMPUTER
SOCIETY
PRESS

Computer Society Press tutorial texts are collections of original materials and reprinted articles assembled as a coherent package designed to address a well-defined area

Tutorial: Supercomputers: Design and Applications by Kai Hwang

This tutorial introduces state-of-the-art supercomputers and presents major design issues and typical applications requirements of high-performance computer systems. The text is designed for scientists, systems designers, programmers, and educators, as well as for people who are involved in the research, development, and application of high-performance computers. It is intended for novices as well as for a major reference for computer professionals. Divided into five main parts, it covers the following aspects of supercomputers: systems architecture, technology buses, large-scale computations, vector processing, language extensions, compiling techniques, commercial and exploratory systems, parallel algorithms, resource allocation, important applications, data flow and very large-scale integration computing, and future trends.

BJ581 (ISBN 0-8186-0581-2): August 1984, 648 pp.,
NM, \$36.00; M, \$24.00

Tutorial: Interconnection Networks for Parallel and Distributed Processing

by Chuan-lin Wu and Tse-yun Feng

This tutorial serves as a useful guide for beginners and as a major reference for all computer professionals. It is hoped that readers, after going through the text, will be able to design interconnection networks that fit their computer architecture needs, design better algorithms, write better programs, and trigger a revolution on the system control concept. It presents fundamentals in interconnection networks, a crucial topic in the field of parallel/distributed processing.

BJ574 (ISBN 0-8186-0574-X): August 1984, 656 pp.,
NM, \$36.00; M, \$24.00

Tutorial: Principles of Communication Network Protocols by Simon S. Lam

This tutorial is a valuable reference for systems engineers and analysts who desire an understanding of the technical principles that underlie the design of communication network software and the performance tradeoffs involved in its design. It is also an excellent source of supplemental readings for graduate-level courses on computer communication networks. Divided into seven chapters, it covers: the fundamentals of computer communication networks, data link control protocols, multiple access protocols, local area networks, resource allocation problems and solution techniques in store-and-forward networks with point-to-point links, communication protocols for wide area networks and internetworks, and models and methods for protocol verification and construction.

BJ582 (ISBN 0-8186-0582-0): October 1984, 528 pp.,
NM, \$36.00; M, \$25.00

Selected Reprints on Logic Design for Testability by Constantin C. Timoc

Interest in designing testable digital logic has grown rapidly in the past decade, therefore, this collection of reprints was carefully compiled to discuss the current status and growing trends toward higher levels of integration and to illustrate the considerable effort required to test the integrated circuits that perform complex logic functions. Topics include: testability problems, logic and switch models of physical failures, test generation and fault simulation, serial and random scan, and built-in self-testing.

BJ573 (ISBN 0-8186-0573-1): August 1984, 324 pp.,
NM, \$25.00; M, \$18.75

Tutorial: Computer Text Recognition and Error Correction by Sargur N. Srihari

Designed for computer researchers interested in developing flexible techniques for text processing and computer vision, this tutorial is concerned with transferring a degree of intelligence to text processing systems. In particular, the ability to automatically detect and correct spelling and typographical errors and to interpret digitized video images of print and script whose iconic representations (visual symbols) are ambiguous. Organized into four parts: an introduction to text error correction, the interpretation of print/script and their postprocessing, spelling and typographical error correction, and dictionary organization, the papers included have appeared in a relatively wide context of computing literature and encompass a fairly wide span of time.

BJ579 (ISBN 0-8186-0579-0): December 1984, 363 pp., NM,
\$36.00; M, \$24.00

VLSI

Selected Reprints on VLSI Technologies and Computer Graphics

by Henry Fuchs

This compilation of reprints is intended for professionals interested in the intersection of, and the relationship between, computer graphics and VLSI. Two major areas are represented: the graphical aspects of VLSI design and the impact of VLSI computing structures on graphics hardware. This book contains 56 printed articles that are divided into eight sections covering the following topics: mask level layout; symbolic layout; floorplanning, placement, and routing; artwork analysis; algorithms for layout synthesis and analysis; CAD systems and related graphics issues; and image analysis.

BJ491 (ISBN 0-8186-0491-3): July 1983, 500 pp.,
NM, \$36.00; M, \$20.00

Tutorial: VLSI—The Coming Revolution in Applications and Design

by Rex Rice

Providing a broad interdisciplinary perspective on present and potential uses of VLSI, this tutorial emphasizes economic considerations rather than covering details of processes. It includes both historical background and examples of current VLSI programs. The main portion of the tutorial traces a VLSI design through the complete processes from system design through a tested computer, giving particular emphasis to the hazards to be avoided and discussing available alternatives and economic considerations for each step.

BJ288 (ISBN 0-8186-0288-0): February 1980, 316 pp.,
NM, \$30.00; M, \$20.00

Tutorial: VLSI Technologies—Through the 80's and Beyond

by Denis J. McGreivy and Kenneth A. Pickar

The semiconductor industry, now in its fourth decade of growth, is experiencing unprecedented demands from all facets of government, industry, and science. This has created highly competitive R&D efforts to reduce physical size of these chips while improving performance-to-cost characteristics. In this tutorial, an attempt is made to chart the most probable path of technology evolution in the integrated circuit industry through the remainder of this decade. Parameters with which to identify and describe past and future trends are discussed, and market demands and projections of anticipated supply are also presented. A detailed discussion of the trends in various VLSI technologies in which such factors as size, complexity, and costs, are also examined.

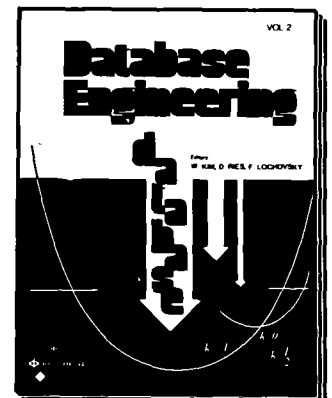
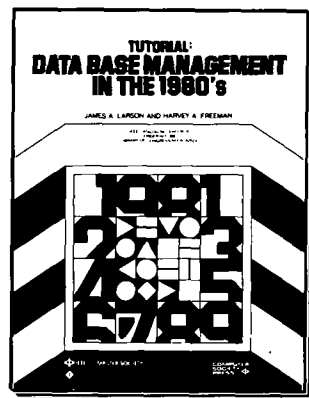
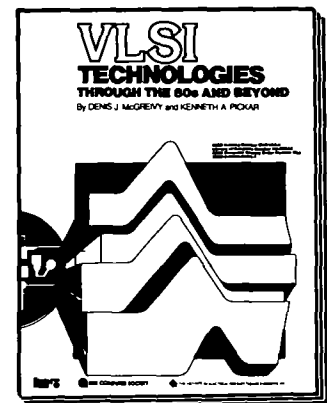
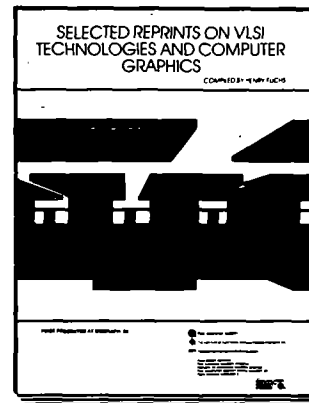
BJ424 (ISBN 0-8186-0424-7): April 1982, 450 pp.,
NM, \$30.00; M, \$20.00

Tutorial: VLSI Support Technologies (Computer-Aided Design, Testing, and Packaging)

by Rex Rice

Designed to answer the question of what technologies one needs to design and to use custom and "semi-custom" VLSI arrays, this tutorial attempts to provide a single source of information on the three major concerns of VLSI designers today. For the least expensive and most expeditious development of VLSI, the author provides bridging materials to enable designers, implementers, and users to realize the effects of their activities on other aspects of VLSI design.

BJ386 (ISBN 0-8186-0386-0): February 1982, 480 pp.,
NM, \$30.00; M, \$18.75



Tutorial: Data Base Management in the 80's

by James A. Larson and Harvey A. Freeman

This tutorial addresses the kinds of data base management systems (DBMS) that will be available through this decade. Interfaces available to various classes of users are described, including self-contained query languages and graphical displays. Techniques available to data base administrators to design both logical and practical DBMS architectures are reviewed, as are data base computers and other hardware specifically designed to accelerate database management functions.

BJ369 (ISBN 0-8186-0369-0): September 1981, 472 pp.,
NM, \$27.00; M, \$20.00

Database Engineering, Volume 2

Binding the four 1983 issues of the quarterly newsletter of the Technical Committee on Database Engineering, this book featured articles covering such topics as: database systems being marketed by major vendors in Japan, commercial transaction-processing systems, various approaches to automating office systems, and expert systems. Includes 37 papers.

BJ553 (ISBN 0-8186-0553-7): February 1984, 274 pp.,
NM, \$20.00; M, \$15.00

PUBLICATIONS ORDER FORM

Return with remittance to:

IEEE Computer Society Order Department
 P.O. Box 80452
 Worldway Postal Center
 Los Angeles, CA 90080 U.S.A.



Discounts, Orders, and Shipping Policies:

Member discounts apply on the **FIRST COPY OF A MULTIPLE-COPY ORDER** (for the same title) **ONLY!** Additional copies are sold at list price.

Priority shipping in U.S. or Canada, **ADD \$5.00 PER BOOK ORDERED.** Airmail service to Mexico and Foreign countries, **ADD \$15.00 PER BOOK ORDERED.**

Requests for refunds/returns honored for **60 days** from date of shipment (90 days for overseas).

ALL PRICES ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL BOOKS SUBJECT TO AVAILABILITY ON DATE OF PAYMENT.

ALL FOREIGN/OVERSEAS ORDERS MUST BE PREPAID.

Minimum credit card charges (excluding postage and handling), **\$15.00.**

Service charge for checks returned or expired credit cards, **\$10.00.**

PAYMENTS MUST BE MADE IN U.S. FUNDS ONLY, DRAWN ON A U.S. BANK. UNESCO coupons, International money orders, travelers checks are accepted. **PLEASE DO NOT SEND CASH.**

ORDER HANDLING CHARGES (based on the \$ value of your order—not including sales tax and postage)	
For orders totaling:	Add:
\$ 1.00 to \$ 10.00	\$ 3.00 handling charge
\$ 10.01 to \$ 25.00	\$ 4.00 handling charge
\$ 25.01 to \$ 50.00	\$ 5.00 handling charge
\$ 50.01 to \$100.00	\$ 7.00 handling charge
\$100.01 to \$200.00	\$10.00 handling charge
over \$200.00	\$15.00 handling charge



PLEASE SHIP TO:

NAME

AFFILIATION (company or attention of)

ADDRESS (Line - 1)

ADDRESS (Line - 2)

CITY/STATE/ZIP-CODE

COUNTRY

IEEE/COMPUTER SOCIETY MEMBER NUMBER (required for discount)

PHONE/TELEX NUMBER

PURCHASE ORDER NUMBER

QTY	ORDER NO.	TITLE/DESCRIPTION	M/NM PRICE	AMOUNT

If your selection is no longer in print, will you accept microfiche at the same price? Yes No

CALIFORNIA RESIDENTS ADD 6% SALES TAX

HANDLING CHARGE (BASED ON SUB-TOTAL) \$ _____

OPTIONAL PRIORITY SHIPPING CHARGE \$ _____

TOTAL \$ _____

METHOD OF PAYMENT (CHECK ONE)

CHECK ENCL. VISA MASTERCARD AMERICAN EXPRESS

CHARGE CARD NUMBER

EXPIRATION DATE

SIGNATURE _____ **BJ**



IEEE COMPUTER SOCIETY

Administrative Office

P.O. Box 639
Silver Spring, Maryland
20901

Non-profit
Organization
U.S. Postage
Paid
Silver Spring, MD
Permit No. 1398