A Practitioner's Introduction to Database Performance Benchmarks and Measurements

S. W. DIETRICH*,[1] M. BROWN,[2] E. CORTES-RELLO[2] AND S. WUNDERLIN[2]

[1] *Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406, U.S.A.*

[2] *Bull Worldwide Information Systems, Inc., 13430 N. Black Canyon Highway, Phoenix, AZ 85029, U.S.A.*

Database performance benchmarks provide an important measure for the comparison of database management systems. This paper provides an introduction to performance benchmarks for centralised databases. The introduction includes benchmarks for transaction processing, such as DebitCredit, TPC-A and TPC-B, and benchmarks for decision support, such as the Wisconsin benchmark and its extension by Bull to a vendor benchmark, known as the Single-User Decision Support benchmark. An important contribution of this paper is a practitioner's perspective of the issues involved in performance measurement.

Received January 1992, revised March 1992

1. INTRODUCTION

The performance of a database management system (DBMS) plays an integral role in the decision of a company to utilise that database. The price of the database is also an important factor. Such decisions must be based on information that compares the performance and price of database management systems of different vendors. Benchmarks provide a yardstick with which to measure these important factors.

Database performance benchmarks have evolved over time. The performance folklore, as recorded by a group of computer professionals[1], includes several benchmarks. One benchmark for measuring the performance of transaction processing in a database system is the DebitCredit benchmark. The flexibility in this transaction processing benchmark, however, provided incomparable results. The need for industry standard benchmarks led to the development of the Transaction Processing Performance Council (TPC), whose varying membership includes many computer organisations. The council modified the DebitCredit benchmark into an industry standard benchmark TPC-A,[14] which has specific guidelines for the measurement of performance and price.

Although TPC-A provides very specific guidelines, the measurement of the benchmark is not a trivial task. Depending on the tools available on the system to be tested, the measurement may take a matter of weeks or months. Once a measurement is taken, the performance group may tune the system configuration to increase performance and to reduce the cost. This tuning process is iterative and may take much longer than the initial measurement.

This paper provides a practitioner's introduction to performance benchmarks for centralised databases, and reports on the experience of practitioners in the measurement of database benchmarks. This paper is not meant to be a comprehensive survey but an introduction to the benchmarks that defined the database performance area, such as DebitCredit, TPC-A, TPC-B and the Wisconsin benchmark. The paper also describes an extension of the Wisconsin benchmark to a vendor benchmark by Bull, known as the Single-User Decision Support benchmark or SUDS. Other benchmarks exist that target various domain-specific applications and assumptions, such as the AS^{3}AP benchmark,[16] the Set Query benchmark,[12] the Engineering database benchmark,[5] and the Neal Nelson database benchmark®.[11] A description of these benchmarks can be found in a recent book by Gray,[9] which is an invaluable reference in the field of performance measurement.

* Work supported by Bull Worldwide Information Systems under Grant CRP 91228. Correspondence should be addressed to this author.

2. TERMINOLOGY

Many speciality areas are laden with built-in terminology that prevents persons outside that speciality to communicate effectively. This section defines the terminology within the specialty of database performance benchmarks.

2.1 Types of benchmarks

There are three types of benchmarks: industry-standard, vendor, and customer-application. Although benchmarks ultimately measure the performance of the system, each type of benchmark has its own goal.

An industry-standard benchmark provides an external view of the product and therefore, samples the performance of the database system on a specific, usually simple, application. The measurements of an industry-standard benchmark are meant to be published to provide information for comparison across various vendors.

Before the specification of industry-standard benchmarks, however, vendors were running their own benchmarks to identify performance improvements for their product. Today, vendors continue to run their own benchmarks since industry-standard and vendor benchmarks address different goals. A vendor benchmark must be comprehensive, providing an introspective view of the evolving product. As the product evolves, so must the benchmark that tests its performance. The measurements of a vendor benchmark are meant to stay within the company to guide development efforts and to provide sales support.

Customer-application benchmarks are designed by the customer for an important application where performance is critical. After designing and documenting the benchmark, customers provide selected vendors with the benchmark. The vendors then compete for the customer's business by measuring the benchmark to provide the customer with cost and performance figures. The cost of designing and documenting application benchmarks, however, is a considerable factor for the customer. This cost may lead to increased usage of industry-standard measurements in the decision making process for the customer. The Transaction Processing Performance Council, however, warns that while industry-standard benchmarks play an important role in comparing products of different vendors, there are times when specific customer-application benchmarking is critical.

2.2 Transactions and the ACID test

A transaction is informally defined as an atomic (all or nothing) program unit that performs database accesses or updates, taking a consistent (correct) database state into another consistent database state. The atomicity and consistency requirements of a transaction have necessary implications on concurrency control and recovery control. Concurrency control must guarantee that a transaction remains consistent during concurrent execution; this property is known as isolation. Recovery control must guarantee that a transaction is preserved across failures; this property is known as durability. The Atomicity, Consistency, Isolation and Durability properties of a transaction are known as the ACID properties of a transaction.

The Transaction Processing Performance Council[14, 15] defines the ACID properties of a transaction as follows.

Atomicity. 'The system under test must guarantee that transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partially-completed operations leave any effects on the data.'

Consistency. 'Consistency is the property that requires any execution of the transaction to take the database from one consistent state to another.'

Isolation. 'Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.'

Durability. 'The testbed system must guarantee the ability to preserve the effects of committed transactions and insure database consistency after recovery' from single failures, which are clearly specified in the TPC documents.[14, 15]

Typically, database texts (e.g. Ref. 8) characterise a transaction by the properties of Atomicity, Consistency, Isolation, Durability and Serialisability. The additional property of serialisability states that the result of the concurrent execution of transactions is equivalent to some serial execution of the transactions. This definition of serialisability is equivalent to the definition of isolation above. The database texts, however, usually refer to isolation as the property of a transaction such that the transaction does not reveal its uncommitted results to other transactions. Forcing the results to be equivalent to some serial execution order of the transactions would also enforce the textbook definition of isolation, since a transaction would only reveal committed results in a serial execution. Thus, the above ACID properties capture the desired properties of a transaction.

2.3 Transaction processing versus decision support

Transaction processing environments typically consist of update-intensive database services characterised by 'significant disk input/output, moderate system and application execution time, and transaction integrity'[15]. The term 'transaction integrity' refers to the ACID properties of a transaction.

Performance metrics for transaction processing are typically a measure of throughput, which is the rate at which transactions are completed. Thus, the metric unit is transactions per second, which is abbreviated tps. Another important measure is the response time of the transaction, which is the elapsed time for the execution of the transaction. In practice, a combination of these measures is used. The transaction processing benchmarks, discussed in the next section, optimise throughput with constraints on response time. These benchmarks also include a cost metric. Since there is a relationship between the performance of a system and its cost, the price of a system is normalised with respect to tps. Thus, a system is benchmarked with respect to its performance in tps and its price in $/tps.

Decision support differs from transaction processing. A decision support environment is typically not update-intensive and is characterised by 'a wide range of functions, provided over small to large databases'.[4] The use of ad hoc queries for decision support is facilitated by the flexibility of query specification offered by relational database query languages, such as SQL.

Performance metrics for decision support are typically a measure of response time, which is also called query elapsed time. Other performance data, however, is typically collected on the utilisation of the CPU and I/O so that the performance of the system can be analysed. Although decision support benchmarks do not typically include a throughput or cost component, the Set Query benchmark[12] includes both the computation of the average query throughput in units of query per minute (QPM) and a calculation for pricing.

3. BENCHMARKS

This section describes various performance benchmarks with respect to the type of processing that the benchmark is designed to test. The two categories of benchmarks described include update-intensive transaction processing and the ad hoc query processing of decision support. Most benchmarks, including those discussed in this section, measure the use of the system in transaction processing and querying rather than the performance of the utilities of the DBMS, such as bulk loading and organisation of the database. An example of a benchmark that includes tests to measure database loading and structuring, in addition to transaction processing and querying, is the AS^{3}AP benchmark.[16]

The transaction processing benchmarks, DebitCredit, TPC-A and TPC-B, are designed to test update-intensive operations in a particular banking enterprise. The decision support benchmarks, Wisconsin and SUDS, are

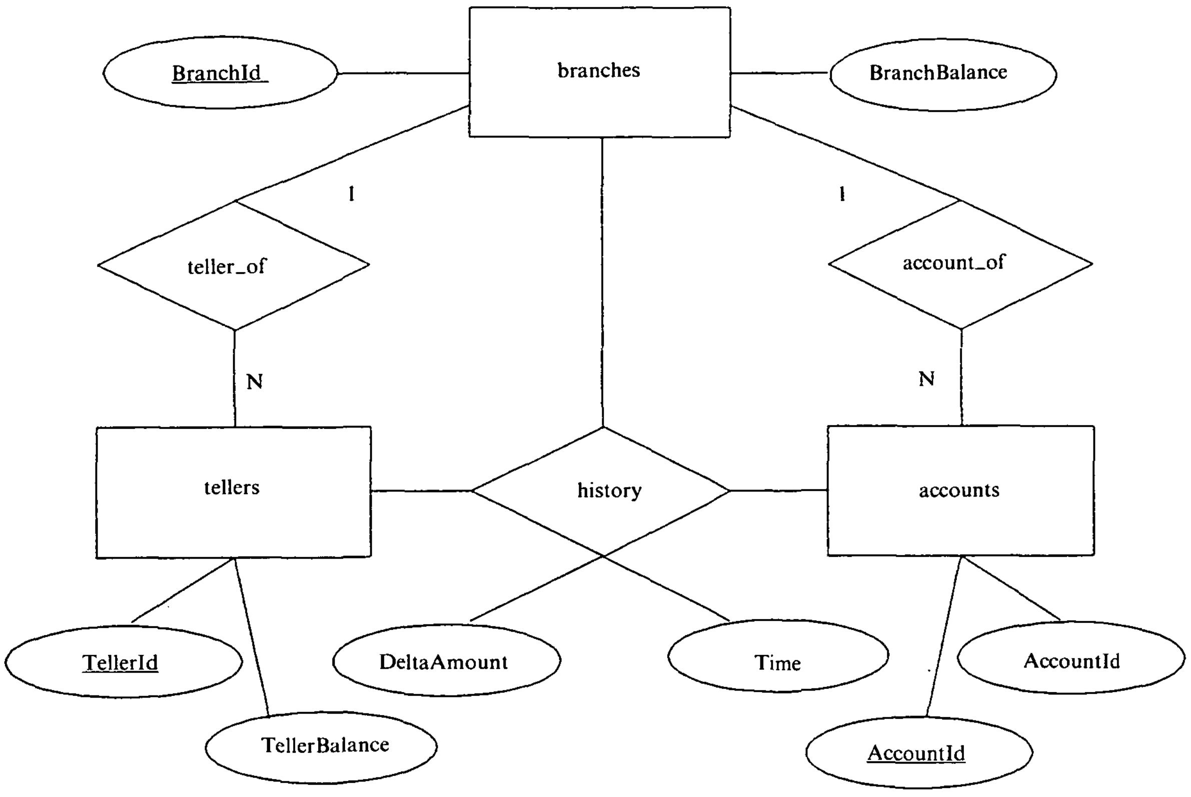

Figure 1. Entity relationship diagram for bank enterprise.

designed to test ad hoc querying on a carefully constructed database. The semantics of the enterprises are implicit in the database and the benchmark programs. Database technology is evolving toward making the semantics of an enterprise explicit through the specification of semantic constraints. The next generation of benchmarks, with the standardisation of the extension to SQL, known as SQL2,[6] will probably address the performance impact of the added responsibility of the DBMS to manage these constraints.

3.1 Transaction processing

The transaction processing benchmarks are designed to test update-intensive operations. We describe the original DebitCredit benchmark, followed by its evolution to the industry standard benchmarks: TPC-A and TPC-B. The TPC is currently working on another benchmark for order entry transaction processing, called TPC-C.

3.1.1 DebitCredit

As its name implies, the DebitCredit benchmark[1] is in the domain of banking, where the operation of interest is a debit or credit to an account performed by a teller at a particular branch. Historically, the origin of the benchmark is based upon the on-line requirements of a retail bank for 1000 branches with 10000 tellers and 10000000 accounts. The additional requirement for the system was a peak load of 100 transactions per second (tps).

The DebitCredit database maintains information on `accounts`, `tellers`, and `branches` and a `history` file of bank transactions. Figure 1 is an Entity-Relationship (ER) diagram for the bank enterprise. ER diagrams are a common tool used for the conceptual modelling phase of a database design.[8] The entities, denoted by rectangles, are `accounts`, `tellers` and `branches`. The relationships are denoted by diamonds. The relationships `account_of` and `teller_of` indicate the branch associated with the account and teller, respectively. The relationship `history` is a ternary relationship, relating the account, teller and branch involved in the transaction. The attributes, denoted by ovals, indicate properties of the entities or relationships to which they are connected. The underlined attributes are key attributes, which are uniquely identifying attributes for the entities.

The schema for the relational data model representation of the bank enterprise is shown in Figure 2. The relations for `branches`, `tellers` and `accounts` include the attributes associated with the entities from the ER diagram. In addition, the `accounts` and `tellers` relations include the attribute `BranchId`, which indicates the associated branch as given by the relationships `account_of` and `teller_of`. The primary keys of the `accounts`, `tellers` and `branches` relations are given by the corresponding key attributes from the ER diagram. The records for the relations `accounts`, `tellers` and `branches` must be 100 bytes in length, which can be achieved through the use of an additional filler field to construct a record of the required size. The `history` relation contains the attributes corresponding to the key attributes of the `branches`, `tellers` and `accounts` entities and the relationship's descriptive attributes `DeltaAmount` and `Time`. The `history` record consists of 50 bytes, using filler if needed, and the benchmark assumes that there is

```
branches(BranchId, BranchBalance)
tellers(TellerId, BranchId, TellerBalance)
accounts(AccountId, AccountBalance, BranchId)
history(AccountId, TellerId, BranchId, DeltaAmount, Time)
```

Figure 2. Relational schema for bank enterprise.

one history file that must be able to store 90 days worth of history data.

The DebitCredit transaction represents a change to an account, either a debit or a credit, that is performed by a teller at a particular branch. The transaction includes input from the terminal, the updates to the account, teller, and branch information, the writing of the history record for the transactions and an output message to the terminal. The message processing assumes the use of an X.25 wide area network protocol on block mode terminals, such as IBM 3270.

The input and output processing to the terminal in the DebitCredit transaction tests the tasks associated with *on-line transaction processing* (OLTP), which adds an additional requirement of multiple on-line terminal sessions to the requirements of transaction processing previously introduced. An important component of OLTP is the transaction arrival distribution. After receiving a response, each emulated terminal must wait before sending its next request to update the database. This waiting is known as 'Think Time'. The think time in the DebitCredit benchmark is 100 seconds, indicating that each terminal waits, on the average, 100 seconds before submitting the next transaction. Therefore, to meet the requirement of a peak load of 100 tps, the benchmark specifies that there are 100 terminals per tps.

The statements used to process the updates of the transaction in the industry-standard query language SQL are shown in Figure 3. (Note that the terminal I/O statements that are part of the transaction are not shown.) The transaction updates the balance of `accounts`, `tellers`, and `branches` and inserts a `history` record, including a timestamp, of the transaction. The identifiers prefixed by: represent the values for the particular transaction under consideration. Identifiers that are not prefixed represent attribute names, which are the same as in Figure 2.

The DebitCredit transaction has to satisfy additional requirements, including response time, concurrency control and recovery control. The benchmark requires that 95% of the transactions provide less than one second response time, as measured at the system under test (SUT). Concurrency control requirements necessitate the protection of all data files by locking and logging. For recovery control, the log file is assumed to be replicated to handle single failures.

The DebitCredit benchmark provided guidelines but the implementation of the benchmark was vulnerable to interpretations by the implementer. This degree of flexibility and the lack of full disclosure of how the performance measurements were obtained, resulted in measurements that were not necessarily comparable. These incomparable results led to the development of the first industry-standard benchmark TPC-A by the Transaction Processing Performance Council.

3.1.2 TPC-A

The TPC-A benchmark specification,[14] consisting of 42 pages, provides substantial guidelines. TPC-A requires strong ACID properties of the system with specific tests for checking these properties. TPC-A also requires scaling rules for the database to maintain a fixed relationship between the size of the relations and the transaction load on the system. In addition, a full disclosure of the reporting procedure is required for TPC-A with auditing strongly recommended but not required. However, before any TPC-A results can be announced, the disclosure report must be given to the TPC.

There is also a difference between DebitCredit and TPC-A with respect to the requirements on the history file. In TPC-A, the history file may be horizontally partitioned, whereas DebitCredit required a unified history file. The DebitCredit benchmark also required storage for the history file for 90 days. TPC-A uses a more realistic constraint that requires storage for the history file for eight hours of operation of the SUT, with pricing requirements for 90 days of history data.

The TPC-A transaction strongly resembles the DebitCredit transaction.[13] The main difference is that the terminal I/O is outside the boundaries of the transaction. There is a driver system that emulates the terminal requirements. Recall that after receiving a response, each emulated terminal must wait before sending its next

```
update accounts
set AccountBalance=AccountBalance+ :DeltaAmount
where AccountId= :AccountId

update tellers
set TellerBalance=TellerBalance+ :DeltaAmount
where TellerId= :TellerId

update branches
set BranchBalance=BranchBalance+ :DeltaAmount
where BranchId= :BranchId

insert into history(AccountId, TellerId, BranchId, DeltaAmount, Time)
values(:AccountId, :TellerId, :BranchId, :DeltaAmount, :CurrentTime
```

Figure 3. SQL statements for updates in transaction.

request to update the database. In TPC-A, the average of the think time added to the response time must be ten seconds. The think time is approximated by a delay, which provides an essentially random arrival distribution. In DebitCredit, the average think time is 100 seconds. Due to the differences in the average think time between the two benchmarks, the 100 terminals per tps in DebitCredit was updated to 10 terminals per tps in TPC-A.

The response time requirements of TPC-A specify that 90% of the transactions must respond within two seconds, with the response time being measured at the remote terminal emulator (RTE) rather than at the SUT. Note that the response time measured at the RTE includes the delay across the communication network, which in TPC-A may include local area networks. Due to the differences in communication delay between local area networks and wide area networks, the reporting metrics for tps in TPC-A include the specification of tpsA-Wide or tpsA-Local.

3.1.3 TPC-B

The Transaction Processing Performance Council designed another benchmark, TPC-B,[15] that tests the database aspects of the transaction. TPC-B does not generate transactions through terminal emulation but through driver programs, which generate transactions as quickly as possible without allowing for any think time. This relaxation is expected to lead to increased throughput, which is measured in transactions per second (tps). The reporting metric used for TPC-B is tpsB, which denotes the tps metric following the specification of TPC-B, and is not comparable to results from TPC-A.

TPC-B requires the throughput to be subject to a residence time constraint specifying that 90% of all transactions must have a residence time of less than 2 seconds. Since TPC-B does not include terminal emulation, residence time is measured by the elapsed time at the driver between supplying the inputs to the transaction and receiving a corresponding response. This residence time constraint is similar to the response time constraint of TPC-A, where response time is measured by the elapsed time between sending the first byte of the input message and receiving the last byte of the output message.

The TPC-B benchmark is viewed by performance practitioners as a precursor to the TPC-A benchmark with respect to performance measurement. Typically, TPC-B is installed and measured to test the database installation and to tune the system. TPC-A, which has added requirements for on-line processing, is then installed and measured.

3.2 Decision support

Relational databases provide the capability of ad hoc query specification, typically through the use of the industry-standard query language SQL. A decision support environment takes advantage of this flexibility to specify ad hoc queries to the database. This flexible environment requires a more comprehensive approach to benchmarking the performance of the database across a wide range of functions. The Wisconsin benchmark,[2] which consists of a carefully constructed database and a comprehensive set of queries, provides a systematic approach to benchmarking relational database systems. This paper reports on the database and queries for the original Wisconsin benchmark as they appeared in a later paper[7] and discusses the extension of the benchmark for parallel database systems.[7] The Wisconsin benchmark is also extensible for use as a vendor benchmark, as illustrated by the development of the Bull Single-User Decision Support (SUDS) benchmark.[4] The TPC is currently working on a decision support benchmark, called TPC-DSS.

3.2.1 Wisconsin

The database for the Wisconsin benchmark does not correspond to a particular application, such as the TPC benchmarks, but is carefully designed to produce predictable results for decision support benchmarking. The names of the relations and the attributes provide a self-description of its contents. This systematic naming convention allows for the design of an extensive database for systematic benchmarking.

A relation is named by the number of tuples that the relation contains, which is called its cardinality. For example, the relation `ONEKTUP` denotes a relation that contains one thousand tuples. More than one relation of the same cardinality can exist in the database. For example, `TENKTUP1` and `TENKTUP2` denote two relations both having ten thousand tuples. The Wisconsin database consists of `ONEKTUP`, `TENKTUP1` and `TENKTUP2` relations. The specification of the attributes in a relation, e.g. `TENKTUP`, is given in Figure 4.[3]

Attribute name	Attribute domain	Attribute value
unique1	0..9999	unique, random
unique2	0..9999	unique, random
two	0..1	cyclic: 0, 1
four	0..3	cyclic: 0, 1, 2, 3
ten	0..9	cyclic: 0, 1, ..., 9
twenty	0..19	cyclic: 0, 1, ..., 19
hundred	0..99	cyclic: 0, 1, ..., 99
thousand	0..999	cyclic: 0, 1, ..., 999
twothous	0..1999	cyclic: 0, 1, ..., 1999
fivethous	0..4999	cyclic: 0, 1, ..., 4999
tenthous	0..9999	cyclic: 0, 1, ..., 9999
odd100	1, 3, 5, ..., 99	cyclic: 1, 3, ..., 99
even100	2, 4, 6, ..., 100	cyclic: 2, 4, ..., 100
stringu1	per template	derived from unique1
stringu2	per template	derived from unique2
string4	per template	cyclic: A, H, O, V

Figure 4. Original Wisconsin benchmark: attribute specification.

A relation contains two unique (integer-valued) attributes: `unique1` and `unique2`. The values of the `unique1` and `unique2` attributes in the relation instance are determined randomly in the range between 0 and one less than the cardinality of the relation. Thus, both attributes are candidate keys although the `unique2` attribute serves as a designated sort key, when required.

An integer-valued attribute, in general, is named by the number of distinct values that the attribute contains in the relation. For example, the attribute `ten` has ten distinct values (0..9). The range of values that the attribute assumes appears with a uniform distribution in the relation by cycling through its possible values. Obviously, an attribute named in this way is typically a non-unique attribute, since the value of the attribute appears in the relation more than once (assuming that the cardinality of the relation exceeds the number of distinct values of the attribute). For example, the TENKTUP1 relation of the Wisconsin database contains the following non-unique integer-valued attributes with cyclic order: two, four, ten, twenty, hundred, thousand, twothous, fivethous, tenthous, odd100 and even100. The attributes, odd100 and even100, represent the odd and even numbers, respectively, in the range of 1 to 100. These non-unique integer-valued attributes are used to model various selectivity factors.

The database also has string attributes to test string operations. Each string consists of 52 letters, and must obey the following template:

$xxxxxxxxxxxxxxxxxxxxxxxxx$xxxxxxxxxxxxxxxxxxxxxxxx$

where $ designates a letter in the set {A..V}, which consists of 22 letters. There is a substring consisting of 25 x's between the first and second $, and a substring consisting of 24 x's between the second and third $. This basic string pattern allows 10648 (22*22*22) unique string values, and may be easily modified to allow for additional values.

A relation contains three string attributes: stringu1, stringu2 and string4. The attributes stringu1 and stringu2 are string versions of unique1 and unique2, respectively. Either attribute may be used as a key attribute. The stringu2 attribute is typically used for sorting and indexing. The attribute string4 has four distinct values. The unique values are constructed by forcing the $ positions of the string to have the same value and to be chosen from a set of four letters: {A, H, O, V}. The string4 attribute plays a similar role as the non-unique integer-valued attributes.

The Wisconsin Benchmark contains a set of queries that test various operations: selection, projection, join, aggregation, append, delete, and modify. There are variations on each type of query for selectivity factors and the availability of primary/secondary indexes. The benchmark is considered an industry-standard benchmark since it has a fixed set of queries and was used to compare the performance of products across several vendors.[2]

The Wisconsin benchmark provides a very useful tool for performance comparison. A retrospective view of the benchmark by (some of) its authors[3] indicated suggestions for improvement. A revised Wisconsin benchmark[7] addresses scalability issues for benchmarking of parallel database systems.

The schema of the relations in the scalable Wisconsin benchmark have been updated, resulting in the attribute specification shown in Figure 5.[7] The unique1 attribute remains unchanged, representing a candidate key of the relation with its 0 to (cardinality − 1) values being randomly distributed. All other attributes have been changed either with respect to its domain of values or its computation of values. The updated unique2 attribute is a declared key and is ordered sequentially. The attributes two, four, ten and twenty now have a random rather than a cyclic ordering of values, which is derived by an appropriate mod of the unique1 values. For the TENKTUP1 relation in the original benchmark, the attributes hundred, thousand, twothous and fivethous were used to provide access to a known percentage of values in the relation, respectively, 1%, 10%, 20%, and 50%. In the revised benchmark, these attributes have been replaced by new attributes onePercent, tenPercent, twentyPercent, and fiftyPercent. The order of the values of these 'percentage' attributes are random and are based on a mod of the unique1 value. The tenthous attribute of the TENKTUP1 relation referred to a single tuple. This attribute has been replaced by the attribute unique3, which takes on the value of unique1. The even100 and odd100 attributes have been updated to evenOnePercent and oddOnePercent, deriving attribute values from an appropriate function defined over the value of the new attribute onePercent.

Attribute name	Attribute domain	Attribute value
unique1	0..(MAX-1)	unique, random
unique2	0..(MAX-1)	unique, sequential
two	0..1	unique1 mod 2
four	0..3	unique1 mod 4
ten	0..9	unique1 mod 10
twenty	0..19	unique1 mod 20
onePercent	0..99	unique1 mod 100
tenPercent	0..9	unique1 mod 10
twentyPercent	0..4	unique1 mod 5
fiftyPercent	0..1	unique1 mod 2
unique3	0..(MAX-1)	unique1
evenOnePercent	0, 2, 4, ..., 198	onePercent*2
oddOnePercent	1, 3, 5, ..., 199	(onePercent*2)+1
stringu1	per template	derived from unique1
stringu2	per template	derived from unique2
string4	per template	cyclic: A, H, O, V

Figure 5. Scalable Wisconsin benchmark: attribute specification.

The string attributes in the revised benchmark have also changed. Although the length of the strings are the same, the template for the strings has been modified. The stringu1 and stringu2 attributes derive their values from the unique1 and unique2 values, and must obey the following template:

$$$$$$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

where the first seven characters in the string, indicated by $, are a letter in the set {A..Z} and there is a concluding substring consisting of 45 x's. This template allows 26^7 possible values of the string, and is easily modifiable. This change to the string template addresses the concern that most strings are differentiated in the early portion of the strings. The original benchmark had differentiating positions in the middle and at the end of the string. The string4 attribute still takes on four unique values in a cyclic fashion. The unique values are constructed by forcing the first four positions of the following string template

$$$$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

to have the same value and to be chosen from a set of four letters: {A, H, O, V}.

The Wisconsin benchmark provides for the systematic benchmarking of database systems. Its extensibility is evident both from its revision into a parallel database system benchmark and from its extension into a comprehensive vendor benchmark.

3.2.2 Single-user decision support

The Bull Single-User Decision Support (SUDS) benchmark[4] developed as an extension of the Wisconsin benchmark to measure the performance of INTEREL, the Bull decision support product for Bull's proprietary operating system GCOS 8. INTEREL is a relational system that uses the industry-standard query language SQL. INTEREL also has the capability to access data contained in several file types. For example, INTEREL provides decision support on data in an IDS/II (network) database through the use of a utility that provides a relational view of the network schema. INTEREL can then access the data in the network database using SQL.

The SUDS benchmark analyses the performance of INTEREL, with the goal of identifying areas within the system that require performance enhancement. The benchmark assumes a single-user execution environment. For each statement executed, an internal performance monitor measures and reports on response time, processor time, input/output and memory usage.

The benchmark contains approximately 230 SQL statements, whereas the Wisconsin benchmark contains 32 statements. SUDS measures the performance of various retrievals, such as unique versus non-unique and indexed versus non-indexed retrievals, variations of the where clause, scalar aggregation, the order by clause, the group by clause, the unique qualification, updates, insertions, deletions, set operations, simple joins, views and complex joins.

The SUDS benchmark results provide a detailed and comprehensive view of the INTEREL product, leading to a substantial performance improvement.[4] Also, the knowledge learned from the benchmark provides useful information for technical customer support personnel, who can share this knowledge with the developers of customer applications.

4. PERFORMANCE MEASUREMENT

The task of measuring performance is a complex process, and many iterations and adjustments are needed to obtain the best performance possible in a given computer system. The following sections describe the tasks involved in a 'serious' measurement of database performance, using the TPC-B benchmark as an example. Although we describe the measurement process for industry-standard benchmarks, the measurement process for vendor-specific and customer-application benchmarks is similar but differs slightly due to the dissimilarities in constraints and objectives of the various types of benchmarks.

4.1 Hardware configuration

As a first step, the hardware configuration to be measured must be specified. The specification includes the model and some other characteristics, for example, a Bull DPX-2/340 with four processors, 32 MBytes of memory in each processor, three disk controllers and 16 disk drives. Also a specification of the basic software that will run on top of the hardware platform is needed, for example, BOS 2.0 – the Bull version of the UNIX operating system.

Even at this early stage, an estimation of the expected performance is required. Based upon previous measurements, the expected number of transactions per second can be inferred. The database instance needed for the performance measurement is then scaled to the expected number of transactions per second. In TPC-B, for each tps configured, the database instance must have a minimum of 100000 accounts, 10 tellers, and 1 branch. The ratios between these numbers must also be maintained. A change in any value must be reflected by a proportional change in the other values. (For TPC-A, additional scaling rules dictate that a minimum of 10 terminals is included for each tps configured.) Given the capacity and I/O rate of available disks and controllers and the average number of I/Os per transaction, it is possible to anticipate the number of disks and controllers needed. It is always better to have some excess storage and I/O rate capacity. Once the machine is available, it has to be physically installed and possibly hooked up into a network.

4.2 System software configuration and tuning

Once the machine and the basic software are available, they have to be configured and tuned. The system has several layers that need configuration: the operating system, communications software and the database management system. Initially, most of the configuration parameters will be set to default values. Some operating system parameters may have to be adjusted in order to be able to start running the database management system.

In the process of tuning, tools must be used to monitor the overall behaviour of the system. Examples of such tools are UNIX's sar or easytune, which is a special performance monitoring tool of Bull BOS. The tools provide real time information on important parameters such as memory utilisation, swapping activity, processor utilisation, buffer and caching activity, and physical and logical I/O activity. Other tools provide real time information on DBMS critical parameters such as buffers, and the status of locking and logging. The information is used to detect bottlenecks and tune the operating system (OS) and the database management system (DBMS). An example of such a tool is INGRES' Interactive Performance Monitor.[10]

4.3 Application tuning

The next layer to tune is the benchmark software. In a normal case, such as TPC-B, the benchmark is a series of programs and scripts. Some programs perform setup and statistical functions, while others execute transactions. Some examples of the statistics that must be collected are: tps, response time per transaction, and logging and locking statistics. Some examples of the setup functions are: setting up database tables, setting up temporary tables to collect statistics and setting up timing plans for the execution of the benchmark. The database is scaled with respect to the expected throughput to avoid a completely in-core database. This scaling process involves estimating the expected throughput. If the measured throughput is larger than the estimated throughput, the database has to be re-sized, which is a time consuming process.

In a more complex case, such as TPC-A, a terminal driver is needed to emulate terminals that issue transactions through a network. The benchmark programs must be optimised to minimise the consumption of processor time and memory. An objective of the optimisation is to minimise the overhead of execution of the statistical functions compared to the load to be measured.

Particular attention must be given to problems with locking and concurrency control. While the ultimate goal of a performance measurement is to obtain a metric such as tps or $K/tps, the measurement process has some important constraints. The measurement must preserve the ACID properties of the database. The TPC-B document[15] clearly describes tests to run to verify the ACID properties.

4.3.1 Performance

The objective of this phase is to maximise the transactions per second (tps) metric. At this time, the real work begins. Each one of the relevant configuration parameters from the OS and the DBMS should be analysed, and a 'reasonable' value must be assigned.

On the operating system side, the parameters related to file system, system buffers and allocation of memory are key to performance. On the database side the data buffers, the concurrency control parameters, the logging/archiving and recovery parameters are the most influential.

Some database management systems have special features that may increase performance with the constraints of the benchmark. For example, some databases, such as Ingres and Oracle, provide 'fast-commit' facilities, or faster ways to execute the transaction, such as stored (precompiled) database procedures in Ingres and macroexecution in Teradata RDBC.

Many operating system parameters are directly related to the database management system parameters; in some other cases the relationship is indirect. Some apparently independent parameters may affect each other in an indirect manner. For example, the block size of the database management system is related to the block size of the operating system.

Once the parameters have been set up, the physical structure of data must be considered. Some databases provide multiple storage structures. For example, in Ingres the database designer has the choice of btree, heap, isam and hash storage structures. In many cases, the tables and indexes can be spread over multiple devices. The choice of storage structure depends on what is best for the benchmark, the database management system and the operating system.

Once an initial setup has been configured, the tuning is done via experimentation. The process is painful, and a lot of patience and discipline are needed. Good records must be kept of the step-by-step process.

The objective of the measurement process is not only to generate a number but to analyse whether the number is reasonable and repeatable. For example, a measurement that indicates a 100% processor utilisation may not indicate that the maximum transaction rate has been achieved. An analysis of the processor utilisation by the operating system, database management system and application is needed to understand how the processor is being used by the different components. Once the tps metric has been obtained, other runs of the benchmark under the same conditions (hardware, software and configuration) should yield the same results; that is, the measurements must be repeatable.

By this process, diminishing returns are obtained per unit of effort. At first the most obvious flaws are discovered; and later, given the same *unit of effort* less improvement will be observed in the transactions per second (tps) metric.

It is difficult to know when to stop. There are no good guidelines. The measurement process stops when you run out of time, resources, or simply when the probability of improvement to the tps metric is negligible.

4.3.2 Cost

Up to this point the effort has been targeted toward maximising of the transactions per second (tps) metric. Once the maximum tps has been obtained from the basic hardware/software platform, the next consideration is cost; that is, the metric thousands of dollars per tps ($K/tps). To minimise $K/tps, the benchmark must be run in the least expensive configuration of hardware and software in terms of dollars.

Technical factors. At this stage, a lot will be known about the utilisation of resources such as memory, processor and I/O. For example, the number of physical I/O operations per transaction will be known; and this information, along with the knowledge about the I/O rates of disks and controllers will be enough to decide the optimal combination of controllers and disks, and the optimal allocation of the tables.

Also, the database may be re-scaled to reflect, in a more precise manner, the expected performance of the DBMS. The implication is that less disks may be needed to store the database, and the total cost of the system may decrease.

Marketing factors. The problem of minimising the $K/tps is not only a matter of increasing the tps and decreasing the hardware and software resources; it is also a marketing problem, and specifically it is a pricing problem.

Technical people tend to work in an isolated environment were the decisions are based on technical arguments; but in the case of minimising $K/tps marketing people can play a major role. Certain combinations of software and hardware can be priced in special ways. Of course, the configuration being priced must be available to the public – at the price quoted.

4.4 Integration

After the tps metric has been maximised and the $K/tps has been minimised, a disclosure report must be written. It is recommended to audit the measurement, in order to check compliance with the benchmark definition.

From the description above, the reader may get the misleading impression that the process of measuring performance, although complicated and lengthy, is sequential. Figure 6 shows a more realistic picture of the performance measurement process.

Problems may occur at any of the phases. For example, sometimes the hardware configuration process can be

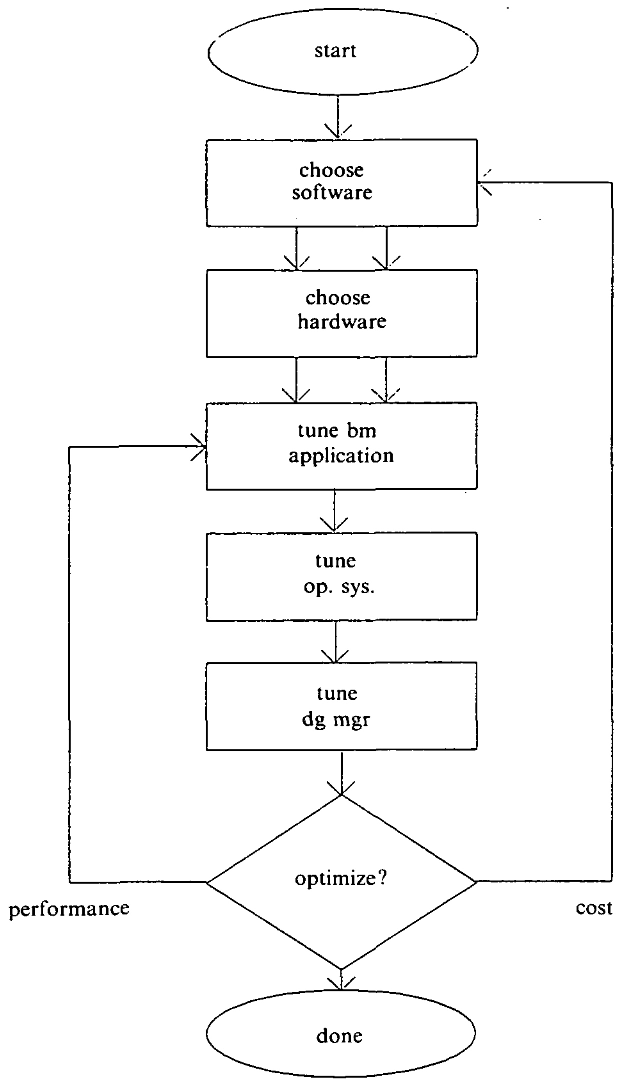

Figure 6. Integration of performance and cost tuning.

slowed because some trivial connectors or cables were not ordered or were lost. Also, as measurements are done as early as possible in the development cycle, some of the layers involved may not be stable enough, and the measurement process will be an opportunity to discover certain hard-to-detect problems.

The system software configuration and tuning is an iterative process. Initially, the default values of the system parameters are used, and sometimes they are inadequate. Tuning is an experimental process; but the experiments are not random: they are guided by the knowledge of the operating system and the database management system. During the process, the hardware resources may prove to be inadequate and the entire process may start all over again.

In the application tuning phase, the knowledge about the database management system, the operating system and their relationships is again crucial. During this phase, the system software tuning continues as a parallel activity.

Tuning for performance (tps maximisation) can uncover problems in any of the three previous phases. For example, we may discover that the benchmark application is using too much memory, that a crucial parameter of the operating system must be changed, or that there are problems with the DBMS configuration.

Finally, tuning the cost ($K/tps minimisation) involves a very difficult dialogue between people with technical and marketing backgrounds. The cost minimisation process may affect the choice of software components or may involve changes in the hardware configuration.

5. SUMMARY

This paper provided an introduction to performance benchmarks that defined the database performance area, including benchmarks for transaction processing, such as DebitCredit, TPC-A, TPC-B, and benchmarks for decision support, such as the Wisconsin benchmark and its extension by Bull to a vendor benchmark, known as the Single-User Decision Support benchmark. We also gave a practitioner's perspective of the issues involved in performance measurement. The task of measuring performance is a time consuming and complex process. The measurement of benchmarks typically requires an optimisation phase. For industry-standard benchmarks, the goal is to maximise the performance metrics and to minimise the cost metrics. This optimisation phase is an iterative process. Each layer of the system must be tuned according to its own parameters but the tuning of one layer probably has an effect on another layer. The optimisation is also not limited to pure technical factors; marketing factors play an important role in minimising the cost metric.

Acknowledgements

The authors would like to thank the anonymous referee for thoughtful comments that improved the paper. Thanks are also due to Forouzan Golshani, Aime Bayle and the performance Group at Bull Worldwide Information Systems in Phoenix for feedback on earlier versions of this paper.

REFERENCES

1. Anon, A measure of transaction processing power. *Datamation*, (1 April 1985) pp. 113–116. Also appears in *Readings in Database Systems*, edited M. Stonebraker, pp. 300–312. Morgan Kaufmann, San Mateo, CA (1988).
2. D. Bitton, D. J. DeWitt and C. Turbyfill, Benchmarking database systems – a systematic approach. In *Proceedings of Ninth International Conference on Very Large Data Bases*, pp. 8–19 (1983).
3. D. Bitton and C. Turbyfill, A retrospective on the Wisconsin benchmark. In *Readings in Database Systems*, edited M. Stonebraker, pp. 280–299. Morgan Kaufmann, San Mateo, CA (1988).
4. M. Brown, Vendor benchmarking for relational decision support. *BullHN Technical Update*, pp. 8–13 (April 1990).
5. R. G. G. Cattell, An engineering database benchmark. In Ref. 9, pp. 247–281.
6. C. J. Date, An overview of SQL2. In *Relational Database Writings 1989–1991*, edited C. J. Date, pp. 413–423. Addison-Wesley, Reading, MA (1992).

7. D. DeWitt, The Wisconsin benchmark: past, present, and future. In Ref. 9, pp. 119–165.
8. R. Elmasri and S. Navathe, *Fundamentals of Database Systems*. Benjamin–Cummings, Menlo Park, CA (1989).
9. J. Gray (ed.), *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann, San Mateo, CA (1991).
10. *INGRES Interactive Performance Monitor User's Guide*, 64–9 (9) 47913.
11. N. Nelson, The Neal Nelson benchmark®: A benchmark based on the realities of business. In Ref. 9, pp. 283–300.
12. P. E. O'Neil, The set query benchmark. In Ref. 9, pp. 209–245.
13. O. Serlin, Measuring OLTP with a better yardstick. *Datamation*, pp. 62–64 (15 July 1990).
14. Transaction Processing Performance Council, *TPC Benchmark™ A, Standard Specification* (10 November 1989). Also appears in Ref. 9, pp. 39–78.
15. Transaction Processing Performance Council, TPC benchmark™ B, Standard Specification (23 August 1990). Also appears in Ref. 9, pp. 79–117.
16. C. Turbyfill, C. Orji and D. Bitton, AS^{3}AP: an ANSI SQL standard scaleable and portable benchmark for relational database systems. In Ref. 9, pp. 167–207.

Book Review

F. MADDIX
Human–Computer Interaction: Theory and Practice Ellis Horwood, 1990. £18.95 ISBN 0-13-446220-3

For a reluctant book reviewer, the page of errata that fell out of my copy was less than encouraging. Secondly, glancing at the references, a mere 71 of them, it was obvious to me that this book is seriously under-referenced, particular as my name did not appear in appendix A, where the references are buried. In fact, one of my papers is referenced, but is so at a chapter end. Indeed, there is a problem with the whole style of referencing in the book which is inconsistent (sometimes names and dates, sometimes name and numbers, and some references in appendix A and some not).

If the references are one problem with this book, the content index is a more important second problem. At five pages, in double-column format, it at first looks adequate. However, the naturally suspicious will note that each index item has few page numbers associated with it, and at most only six, and this is a very rare exception. Taking what, for me, is becoming a standard test of HCI books, I looked up a couple of vital and difficult index entries: (i) task analysis; (ii) user modelling. Apart from my expertise in these two areas of HCI, they are informative because task analysis and the whole concept of tasks is central to HCI, as is user modelling, and both are technically difficult. In a word, the index is inadequate. 'Task analysis', 'Task model' and 'Tasks' yields but four index entries and 'User model' a mere two. Within the book there are numerous references to these two topics, some of them in bold (e.g. p. 226, 'The task') and some even as section headings (e.g. p. 44, 'Other User models'; p.88, 'Task Analysis'), which just do not occur in the content index at all. I estimate that these two topics between them require two to three times the little indexing they enjoy. Indeed, one of my major criticisms of the book is that user modelling is not dealt with adequately as a topic. In fact, there is a great deal on users in the book, as is essential for any HCI book, but it is dispersed and it appears that the author is often not aware of the need to deal with the whole issue in a coherent manner. To take one example for illustration, there is a whole section on 'Operator psychology' (pp. 247–8) which describes users in the context of their personality attributes, which is clearly an aspect of user modelling, yet, of course, it is not indexed either under 'User model' or even under 'Operator psychology'.

I wish I had had a chance to be the editor of this book, as overall it makes a very valuable contribution to the HCI textbook literature. Unfortunately I think it is structured in a manner that is odd and less than helpful. Like many HCI textbooks it has a few chapters of irrelevant material, most particularly chapter 3 on 'Information processing'. The focus of the chapter is wrong simply because information theory's definition of the bit is inappropriate for describing the information content of stimuli perceived by people because the actual content is determined by the state (knowledge, goals, task constraints) of the perceiving information processor (the mind). Furthermore, while I would have replaced this chapter with one on user modelling, I would still question it utility for practical HCI. To me it seems that there is a tremendous gap between a lot of psychological waffle (and I do have a doctoral degree in experimental psychology) about people and the impact (little) all this psychology has on the design of computer systems. Furthermore, I question how even some relevant user psychology can be transmitted to system designers, who generally do not have a background in 'difficult' subjects such as psychology, sociology, philosophy, economics, etc.

The division of chapters 7 and 8, 'Interfaces' and 'Visual interfaces' respectively, seems to me arbitrary. I also question the utility of chapter 12's 'Experimental studies', which describe four very brief student experiments and the results of which I would not trust for the design of real computer systems, which have to support particular types of user performing specific types of task. Certainly there is a need to balance these small examples with real software engineering examples, and it is here that chapter 13 ('HCI and design') fails. The author's heart is in the right place with respect to the need to go beyond the direct end-user or operator, and that a social perspective is also essential, but this view, expressed in the early chapters, is not carried through the rest of the book. Similarly, I would have appreciated a section, if not a chapter, on ethical issues associated with computers since HCI, it has been argued, by being inherently interdisciplinary and user-centred, is particularly well positioned to address this topic.

Whatever it says in the preface, this is a student textbook and really one for computer science or information technology students. It has sufficient computer technology to maintain some interest for such an audience, but too much for those learning about HCI from other disciplines such as psychology. In this context, its overall weakness in psychology may be seen as an advantage. Since I do teach computer science undergraduates and master-level students about HCI, I intend to recommend this book next year as a primary reference source for my students. The book is not suitable for those in industry because of its weakness at providing advice and practical example of HCI in real system design and software engineering.

Finally, this is a book with a layout of which the publishers should be thoroughly ashamed. I would guess that this book has been prepared on the author's computer system and that the publishers have simply taken the author's version and printed it. The pictures, tables, etc. are not floated so that there are frequent, large blank sections at the bottom of pages. These blanks are not just unaesthetic but actually interfere with the book's semantics, since they do not denote the meaningful end of a section or topic. I know that Ellis Horwood have been publicly criticised in reviews before over such matters, and this is yet another example of what is wrong with far too much technology publishing today. I think the author got a raw deal, and what is really quite a good textbook has been ruined by an absence of the services that I demand from the publishers I deal with.

DAN DIAPER
Liverpool