Doing Your Own Benchmark

Tom Sawyer
15098 Elm Park
Monte Sereno, CA 95030

1. Introduction

When invited to participate in this book I started with the idea that this section would discuss the simple do's and don'ts of running one of the benchmarks described here. Upon some reflection, the scope of the chapter increased to discuss the more fundamental aspects of benchmarking.

2. Why Benchmark?

Vendors often perform benchmark for marketing purposes. These benchmarks are usually the public ones covered in this book, although IBM usually runs its own set of proprietary benchmarks.

Occasionally, vendors run internal benchmarks are run to uncover software bottlenecks that might constrain the product regardless of hardware capacity. Examples of these constraints include looking for a built-in limit on the number of concurrent users or the maximum sixe of a database. More often than not, these seemingly trivial limits prove difficult to remove.

Most customer benchmarks are performed to choose among vendors. Since transaction and batch response times are among the more obvious external measurements of a data processing system, speed is often emphasized in vendor selection. The selection committee may deem the public benchmarks to be inadaquate and propose a benchmark tailored to the installation profile. This chapter will discuss some of the challenges in accomplishing this end.

3. Fundamental Questions

Embarking on a benchmark journey is similar to any learning endeavor. There are three major questions that must be addressed at the start and re-examined many times throughout the adventure. These questions are:

1. What do you want to learn?
2. How much are you prepared to invest?
3. What are you prepared to give up?

Obviously they interact. The answer to the third question is really the *wish list* excised from Q1 by applying Q2. During the exercise, the quantity in the first question diminishes as quantities in the latter two increase. That is, you will always spend more than you think and you will wind up learning less than you desire.

There are several costs involved including:

programmer time
machine resources
management time
lost opportunity.

These costs occur even if you attempt to unload the whole process on the vendors.

3.1. Programmer Time: There will be ample opportunity to spend this resource. The benchmark should have some basis in reality. This will mean

generating the transaction workload
defining transaction logic
defining data access requirements
determining data distributions and generating the database
instrumenting and measuring the system

and more. For example, you may have to supply the database to all vendors to assure consistency.

3.2. Machine Resources: In addition to any machine resources consumed by the programmers, you may have to supply the hardware for the measurement. If you offer an enticing prize (read lots of bucks to the vendor), the vendors may supply the hardware and testing environment. If not, they may insist on using your machine for the measurements and attempt to develop the benchmark on a smaller non-dedicated machine available in their branch. Worst case, they may need to use your machine for benchmark development. This may affect your normal workload more than somewhat.

3.3. Management Time: Management time is divided between reporting, administration and adjudication:

Reporting: The reporting requirement *comes with the territory*. Any effort exceeding a few person-months will require budgeting, progress reporting, and a final report.

Normal Administration: The people assigned to the benchmark will require hand-holding, back-slapping, lower-back slapping, and all else that differentiates us from beast and machine. If vendor personnel are involved, assume two or three vendor folks equal one employee. The long days and normal problems that occur when

products and people are pushed to limits will yield many opportunites for meaningful dialogues. Vendor personnel will participate in these activities.

Adjudication: Adjudication consumes the most managemetn time. The well-defined benchmarks (e.g. TPC-A, TPC-B) have areas that need interpretation. An installation-specified benchmark will present many opportunities for debate.

Both managers and technicians will be involved in rulings that require fundamental tradeoffs between realism, fairness and expense. An example could be the level of performance monitoring that should occur during a measurement. One vendor might have a very complete, albeit CPU consuming, tool that is meant to run occasionly, while another might have efficient online reporting with little or no offline output available.

A complex benchmark will leave managers with the feeling that Soloman had it easy.

3.4 Lost Opportunity: Economists call it opportunity cost; football linebackers call it the paralysis of analysis. Time spent choosing a system is time lost in implementing applications. In the extreme case, a poorly chosen, poorly implemented application will save enough money in a year to pay for a complete re-implementation on a different platform.

The converse is the nightmare of a poorly performing implementation that is over budget and hated by the users.

4. What can be learned

Most think of benchmarks as measures of speed (e.g. transactions per second). This is a narrow definition of performance. Performance encompasses function as well as speed. Indeed a benchmark may be used to measure system functionality, ease of use, operability, ease of development, as well as other aspects of data processing. Most of the benchmarks discussed in this book are performance measurements; although the AS^{3}AP benchmark begins to assess operational issues by including loading and recovery time. The State of California, Department of Motor Vehicles Operational Assessment is a good example of a measurement of both performance and operability.

Functional benchmarks are not highly publicised. Far too often, customers infer more capability in the vendor checklist of features than is actually present - e.g. the capabilities of DBMS stored procedures. Given the wide gaps between claimed function and usability, installations would be

wise to specify a functional level of capability and test for compliance. Discussions on this aspect of benchmarking are beyond the scope of this chapter.

We will focus on the decisions necessary when implementing a performance benchmark. Fortunately, the implementation of a benchmark limited to performance aspects will also bring many development and operation issues to light.

Performance benchmarks are used to measure response time, batch throughput and current hardware capacities.

4.1.What you won't learn

Future performance: Performance benchmarks are reasonable measures of today's capabilities. They are not good measures of future capability. Measured performance is the result of the interplay of many hardware and software components. An improvement in one area may or may not affect your measured performance; improvements may interact, not necessarily to your benefit.

The bottom line is that benchmarks are no better for future predictions than any of the other *tea-leaf* devices currently used.

Vendor support: The support you receive during an acquisition benchmark has little bearing on the type of support you will receive after you buy the system - unless the benchmark support was poor.

5. Benchmark Specification -- An Extended Example

A benchmark has two major components:
the workload specification
the measurement specification.

The workload specification requires some analysis of what we hope to implement. To the extent that the new system has roots in current systems, the specification will have some real numbers. If the system is completely new, some effort must go into database and application design before any work specification can be performed.

Imagine we are attempting to benchmark a newsystem containing both online and batch components. For simplicity, assume the online component consists of several online transactions and several ad-hoc queries. Assume the batch component has and update and a report job. Assume that the application system being benchmarked will be a complete re-write and re-design of an existing system. Also assume that the system recieves some input from other application

systems that are not part of the rewrite and cannot be modified. The benchmark will be used to select a hardware and DBMS platform for the new application system.

These assumptions imply that we have some idea of reasonable response times for the new and re-written transactions. Current data exists although it will be much augmented when the new system is complete. Reasonable projections of the database and workload can be made.

A first step in the specification is to describe the database. To simplify even further, assume that the bidders will supply relational DBMSs (perhaps at our request) and that tables have been designed with some idea of what columns go with them.

5.1. The Online Transactions

The new system will support the current transactions (possibly enhanced) and will also contain new transactions. The first order of business is defining the transactions to be benchmarked.

We have both current transactions and some of the major projected transactions. This is the first of many opportunities to trade off realism for simplicity. Online transactions have the following components:

data transmission (input and response)
teleprocessing
application logic
database access
operating system overhead.

Every component we either eliminate or modify will remove some element of reality and hopefully simplify the benchmark. The decisions on how much to slice may also affect the fairness of the benchmark. You will get lots of vendor assistence on this topic. Lets examine them in easy-to-difficult order.

5.1.1. Application logic

Unless you have compute-intensive applications, this should be the first thing eliminated from a benchmark. Application logic rarely accounts for more than 10 - 15% of the CPU load. Specifying application logic then verifying it across all bidders is time consuming and error prone.

The rationale for retaining application logic is that a single transaction module may have several different paths from top to bottom, each with a different data access type. Retaining data access patterns is important as we will see in the next section.

One simplifying approach is to treat each path as a separate transaction type and weight it accordingly. For instance, transaction type X has separate paths for retrieval, update, insert and

delete, and if the percentage for each path and the overall rate for the transaction is known, then the transaction can be decomposed as follows:

Path Name	Path Percentage	tps Rate	Path Rate
X-Retrieve	70	10	7
X-Update	15	10	1.5
X-Insert	10	10	1
X-Delete	5	10	.5

Do not round up the X-Update number; as we will see in the next section, there will be ample opportunity to make adjustments.

5.1.2. Database access

This component should not be omitted. Most online applications spend the majority of the time in DBMS code. The trick is to get coverage of most database access patterns with as few transactions and as little data as possible.

I recommend using a two dimensional grid to record the tps rate (transactions per second) and the access patterns of the major applications. Once this is done transactions may be combined. We can use the following table as an example:

Transaction	tps	Table 1	Table 2	Table 3	Table 4	Total
Tran 1	6	2R 1U	6F	-	1R	3R 1U 6F
Tran 2	12	-	3F	1R	1R	2R 3F
Tran 3	4	2R	3F	-	1R	3R 3F
Tran 4	1	1R	1U	9F	1R	2R 1U 9F
	R = Random read		U = Update		F = Sequential Fetch	

Note that Transactions 2 and 3 are similar in pattern although different tables get accessed. If the tables are large, there is little chance that any caching will occur. We could combine the transactions by dropping Tran 3 and raising Tran 2's rate to 16 tps.

Similarly, Tran 4 could either be dropped (1/23 = 4.35% loss) or combined with Tran 1 by raising Tran 1's tps rate to 7.

The combined matrix follows:

Transaction	tps	Table 1	Table 2	Table 3	Table 4	Total
Tran 1-4	7	2R 1U	6F	-	1R	3R 1U 6F
Tran 2-3	16	-	3F	1R	1R	2R 3F

Combining transactions may also allow elimination of tables, Table 3 is a candidate for elimination. If the Tran 2-3 Table 3 accesses are redirected to Table 1 the following matrix results:

Transaction	tps	Table 1	Table 2	Table 3	Table 4	Total
Tran 1-4	7	2R 1U	6F	-	1R	3R 1U 6F
Tran 2-3	16	1R	3F	-	1R	2R 3F

Retaining reality during these transformations is a challenge. If the affected tables are large, then little is lost in combining them. If each table has a single index, caching requirements will be slightly understated by eliminating the space needed to cache the index of the deleted table.

However, simplicity has been gained, there is one less table to deal with. This reduces the data generation task. It also reduces the load on the vendors. They have one less table to load, build indexes on, back up, find physical disk space for, etc.

Such simplifications cost little and shorten the benchmark process.

5.1.3. Teleprocessing

Teleprocessing includes the hardware terminal configuration possibly with concentrators and the system software components. The benchmark should include a usable configuration. The trap to avoid consists of benchmarking a *high-performance* environment and discovering later that making it operational involves added cost.

A example of this trap consists of emulating ASCII terminals via a LAN-based driver which submits complete messages in each packet. It later turns out that the terminals actually send packets of one character each, unless some intelligent, perhaps specially programmed concentrator is inserted between terminal and host. This character-at-a-time load multiplies the teleprocessing load on the host by a factor of 10 or 100. So the benchmarked system does not correspond to the real system.

Even more important, the actual number of users that will be simultanously using the system should be simulated. A system that uses ten processes, each submitting one tps, may behave much differently than a system with one thousand users, each submitting a .01 tps. Yet the tps loads of the two systems are identical.

If transaction monitors (e.g. CICS, /T) are included in the configuration, the environment should be examined for operability as well as speed.

5.1.4. Operating system overhead

This component pretty much comes with the territory. If this were a software only benchmark then little consideration need be given to this component as long as all vendors run with the same settings. In our example, the hardware and therefore the operating system are part of the procurement. The major decisions involve the following items:

security
integrity
charge-back accounting
system monitoring.

They all present interesting challenges. You need to decide some base level of function for each vendor that keeps things even. You will also need a mechanism to keep track of *goodies* each system supplies that are not measured but are of sufficient interest to be considered tie-breakers. In the event of a run-off, you may have to at least functionally test these features, when you do, measure performance impact on the benchmark also.

These debates get particularly interesting when one of the systems is more thoroughly understood than the others. There is a tendency for the well-known system to get nailed to the wall while the others have their vendor claims accepted at face value.

Integrity capabilities have large affects on performance. Panicked vendors may state that none of their customers use l*evel-x*, but use *level-y* with no dire consequences - vendor user groups are good sources of sanity.

5.1.2. Ad-hoc queries

In this hypothetical benchmark, the intent of the ad-hoc queries is to simulate some decision support activity occuring in conjunction with operational transactions. Since the queries must be fair to all vendors, they must not be truely ad-hoc, i.e. they must be defined in advance. This fore-knowledge presents the vendors with some interesting tuning opportunities.

The art is specifying the rules surrounding the execution of these queries. These rules will have heavy bearing on the fairness of the tests. For example, one vendor's DBMS might use the order of the tables in the SQL `FROM` clause to determine internal order of query processing. This vendor will be most interested in being allowed to specify the table order. One argument in favor is that experienced query users will soon learn the proper order to list tables, or a view using the proper order could be provided to less experienced users. A competitor with an optimizer that makes its own assumptions about internal query processing might claim that ad-hoc means exactly that and users will often use the wrong table order. The second vendor's idea of fair might encompass random order of tables or all orders averaged, etc.

There are other opportunities to exercise fairness.

The hypothetical benchmark assumes two queries - AH1 and AH2. Their makeup is not particularly important, we need them as examples for measurement purposes.

5.2. The Batch Jobs

Unless your projected shop is 100% online - end-user oriented, its necessary to include batch work in your benchmark. We will assume that the goals of the proposed system include the ability to run batch jobs without degrading the performance of the online work.

You may also want to consider benchmarking a few key batch jobs which must be run frequently and can be run when the online environment need not be up.

The realism issue involves the degree of preplanned access allowed. Questions to be answered include:

Can the tables be ordered in a favorable sequence?
How much data may be duplicated in indexes.
If input is involved, what sequences may be used?
Can the job be manually divided into mini-jobs that run in parallel?

Here are two examples involving the issue of the third question - batch input sequence. In one case, the input provided to the benchmark was prepared in the same organization running the benchmark. There were many steps performed prior to the batch run being benchmarked. The organization allowed the vendors to specify any sequence and any additional information needed to improve performance as long as it could be reasonably accomodated in the prior batch jobs.

All vendors elected to sequence the input on the same key as the major master file. The vendors also specified multiple input tapes which were then run as a series of batch jobs. This approach took advantage of the parallelism available from multiple processors in their configurations.

One vendor further specified an input file describing the number of records in each batch file and the deadline for the total job. This information was used by a *governor* process that scheduled the batch processes. This governor sampled the tps rate and available CPU capacity and scheduled transactions accordingly.

At the other extreme, the batch input was provided by another department over which there was no control and little flexibility. Thus, no changes or additional information were allowed.

5.4. Error Checking

The batch jobs and online transactions should have minimal error checking. This checking is necessary to detect errors in the database and possible access pattern errors in the application. Without this checking, you may be measuring different patterns between the vendors.

One error reporting approach is to insert a descriptive record into an error log table; this table may be checked at any time to monitor errors.

5.5. Omissions

Have we left anything out? Oh sure, astute readers may think of several areas that are ignored. For example the affect on system performance of infrequent transactions. The simplifying assumptions make it easy to cache most programs needed by the measured transactions. However, most installations have some portion of the workload that is never resident.

For example, if 500 transaction types make up 20% of the workload it would not be cost-effective to dedicate resources (main memory) to each of the transaction types, yet the execution of this mix will place a larger burden on the system than a 20% increase in the resident workload. This missing workload could be simulated by creating one transaction which had a rough approximation to the access pattern of the 500 transaction types. This access pattern could then be mapped onto the existing table specifications. The code for this transaction would then be installed 500 times with the appropriate transaction names. The vendors would be required to generate 20% of the transaction rate with transaction names randomly spread over the 500 names. Such a process woudl be more realistic, but would also complicate the benchmark -- it’s a tradeoff.

The simplified benchmark succesfully simulates the system portion of the transactions and to some degree simulates the database portion of the workload. However, by using the smaller benchmark set of tables, it does not simulated the system cost of managing the database portion of the workload - e.g. opening and closing OS files. Ah well, another tradeoff.

5.6. A Caution

Make certain you are measuring the hardware/software environment and not the implementation team's degree of cleverness. One team may come up with a processing scheme that would also work on the contenders' implementations. If this improvement is neither disallowed nor spread about, you have allowed a *confounding factor* to muddy any comparisons.

While cleverness should be rewarded, the clever people may disappear after you own the equipment.

6. The Measurement Environment

The second major hurdle in the benchmark specification is defining the measurement environment. Once again, the trade-off between reality and simplicity governs all. The major decision is

How much of the environment will be measured?

In our case, we have both a batch and an online environment. Furthermore, our online environment has both production transactions and ad-hoc queries. We can resolve the batch environment fairly quickly.

6.1. The Batch Environment

We have two batch jobs, a report and an update. The key decision on these jobs is the extent of application process to be simulated. Again, the major effort should be spent capturing the database access pattern.

6.1.1. DBMS Processing

When benchmarking relational systems, some effort must be expended to insure capturing the DBMS processing time to return information.

For example, the retrieval of information from a secondary file might be accomplished via a *Join* with its master file counterpart. However, if there are master file records which must be examined even if no corresponding records exist in the secondary file, the Join must be disallowed since it would skip the master file records with no secondary counterpart. This means that the data for both files must be accessed separately in the benchmark; this type of processing may be more CPU-intensive than Join processing.

The batch jobs should report some data that can be verified. This might include counts of records accessed, totals of columns known to have certain values, etc.

6.2. The Report

There's little point in measuring printer time, hence, we will consider the report job complete when the report output file is closed.

The report job should be more extensive than the ad hoc queries; e.g. it should contain multiple SQL statements.

6.3. The Update

The challenge here is to push as much update activity into the SQL statements as makes sense. For instance, if the data to be updated does not need to be examined, the update may be

performed directly in SQL. For example, an update that changes an account balance by some transaction amount could be coded:

```
Update charge_account
set balance = balance + :delta
where   account = :account_nbr
  and  (((balance - :delta) <= - account.credit_limit) or
        (:delta             >=  0))
```

This statement will accept charges (negative delta) which leave the balance positive or within the credit limit. It will accept payments (positive delta) in any case - i.e. as long as a negative balance gets less negative, we'll take the money.

Where data must be summarized and/or reported, insure that the appropriate retrieval operations occur as well as the requisite updates.

We can consider this job complete when it writes a summary file. This file could contain totals and possibly a count of records updated by table. Often this latter information is available from SQL.

6.4. The Online Environment

There are three components of online transaction measurement:

response time
transaction rate
hardware configuration

In the hypothetical example, the first two are specified; it is up to the vendor to minimize the third. The transaction rate by specifying by transaction type was specified above, now it remains to spefify the response time. There are two components to a response time measurement: the response time, and the percentile where measured.

6.4.1. Transaction Response Time Percentile

Naive folks will use the average response time (which is close to but not equal to the 50th percentile); more sophisticated specifyiers will opt for the 90th or 95th percentile. The following response time graph illustrates the danger of using the average.

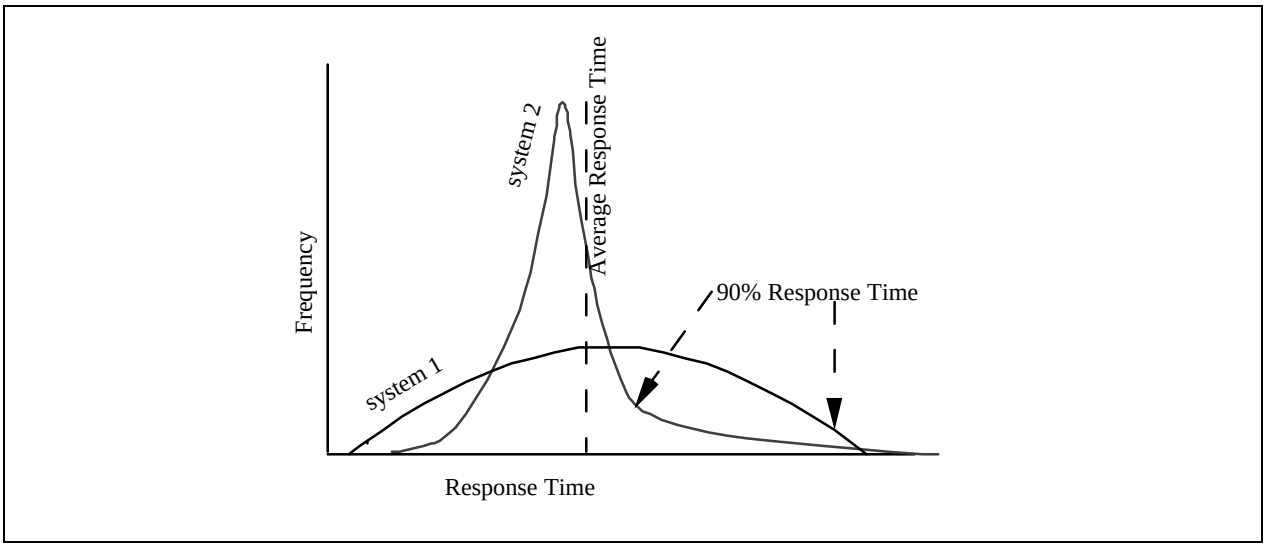

Figure 1. Two response time frequency curves with the same average response time but with very different 90% response times.

There will be more smily faces in the System-1 crowd.

I recommend using the 90th percentile. The difference in effort required to maintain 95 percentile performance over the 90th is something most installations will not pay for.

You will have to decide on the appropriate response time. Your friendly vendors will help you keep it realistic. One rule of thumb is that each random disk access takes about 30 milliseconds; thus, the sum of a transaction's random disk accesses, times .030, will yield a near optimal response time in seconds.

6.4.2. Response Time Measurement Point

Before the response time can be fixed, we must decide where it is to be measured. Response time is the time difference between two complimentary points - e.g. from the time the user presses the Return key until the first character appears on the screen. There are several points at which response time could be taken, including

after the application commits the transaction
after the teleprocessing code
after transmission to the terminal
after display on the terminal.

These points are illustrated in Figure 2.

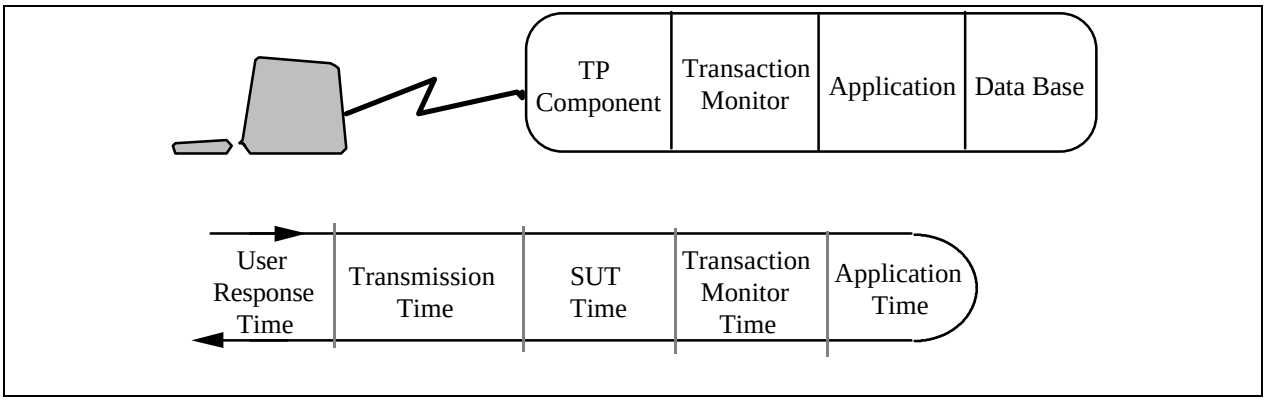

Figure 2. The various places response time can be measured.

The most difficult point to instrument is after the teleprocessing code and before the physical transmission component. The easiest response time calculation is to measure from receipt of transaction to commit of the transaction in the application.

The piece of realism lost will depend to some degree on whether the transactions are submitted through the teleprocessing component or read from disk. The latter implementation is easier to construct and does not require a separate machine to generate and time the transactions (often called a driver). This approach works well when the teleprocessing overhead is well known and the measured machine is not heavily loaded. If the measured machine is to be pushed to its limits, then the transactions must be submitted and timed from a driver machine (see next section).

If we assume block mode terminals or work stations as the submitting devices, we can simulate the environment with the configuration shown in Figure 3.

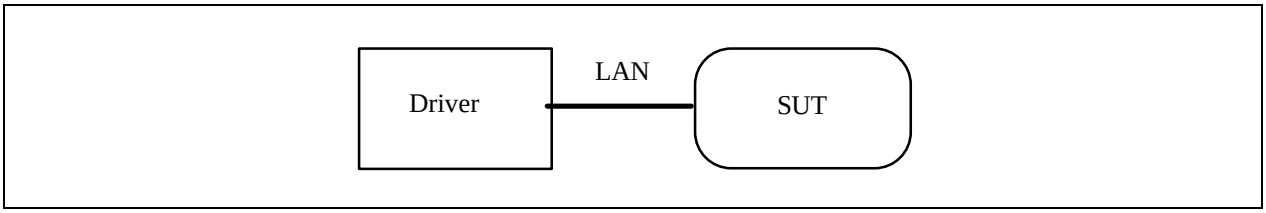

Figure 3. test system consists of a driver and a system under test connected via a LAN.

Some realism has been sacrificed. The submitting device is assumed to place no keying overhead on the SUT. Also the transaction is submitted in the minimum number of LAN packets.

Since the online portion will be submitted and timed externally, it would simplify our lives if we converted the batch jobs to single transactions and used their respective response times as the batch completion times. The one message needed to start or mark the end of a batch job will add little to the completion time and will allow us to collect all results in one place.

The ad hoc queries can be submitted by using very long inter-arrival (or think) times. Again we have simplified the reporting.

6.4.3. Environment Calibration

One of the most important aspects of benchmarking is to be sure you are measuring the right bottleneck. All systems will have a bottleneck. The trick is to force the one you want to measure to be the constraining resource. Stated in converse, it does little good to measure different systems via the same LAN and later discover that the LAN was the bottleneck.

Sufficient headroom is needed in each ancillary component to allow the measured components to be saturated. A simple way to test this hypothesis is to replace the SUT transactions with a ones that echo only and submit transactions with no think time. Since the echo transaction provides near instantaneous response time, either the network or the driver system will saturate at some TPS rate - TPS-Network.

You must insure that the benchmark does not come close to this rate or the results will not be meaningful. If this rate is not sufficient to saturate your projected environment, you may be able to either add drivers or additional networks to increase your headroom. If you suspect the network rather than the driver, and the driver scripts allow it, nullify the call to the network. The driver will saturate at some TPS rate - TPS-Driver. If this rate is close to TPS-Network, then the driver must be replicated. If not, you will want to consider multiple networks or perhaps a connection with more bandwidth - e.g. channel-to-channel, FDDI, etc.

This saturation calibration is also necessary when the transactions are being timed on the measured machine. If the measured machine saturates, the response times will not be meaningful. This occurs because there is insufficient CPU capacity to process tasks that are on the ready queue. Thus they sit on the ready queue waiting to post their response times while the CPU is off fighting the Klingons.

7. Beware the War of Wizards

We will also tell our vendors that this benchmark is a single trial; no excuses, no reruns. This prevents the escalating *war of wizards*. Lets assume vendor A wins the benchmark. Vendor B then says

> "Well, we didn't realize Vendor A was going to use one-star wizards, whereas we used our branch office folks. We would like to rerun the benchmark with the additional tuning and insight equivilent to that used by Vendor A."

Vendor A will in turn want to rerun with two-star wizards claiming that's who Vendor B used, etc. This process can escalate past the four-star wizard ranking with lots of time lost.

A varient of this tactic is the new-and-improved release.

8. Summary

The late 1980's have shown that the *vaporware* announcements of the PC days have been raised to new levels of professionalism. Some vendors have claimed competitor's products only work on one hardware platform - the foil projector. Sometimes they have been correct.

Often functional benchmarks may yield more meaningful results than performance comparisons. Unfortunately, functional benchmarks are more difficult to specify than performance ones and harder to judge.

A benchmark should be treated in the same manner as a developement project with a budget, goals and time estimates. And as with any project, well thought out goals reviewed throughout the benchmark process will produce better results than a cavalry charge.