A Measure of Transaction Processing Power[1]

Anon Et Al

February 1985

ABSTRACT

Three benchmarks are defined: Sort, Scan and DebitCredit. The first two benchmarks measure a system's input/output performance. DebitCredit is a simple transaction processing application used to define a throughput measure: Transactions Per Second (TPS). These benchmarks measure the performance of diverse transaction processing systems. A standard system cost measure is stated and used to define price/performance metrics.

TABLE OF CONTENTS

[1] A condensed version of this paper appears in Datamation, April 1, 1985. This paper was scanned from the Tandem Technical Report TR 85.2 in 2001 and reformatted by Jim Gray.

Who Needs Performance Metrics?

A measure of transaction processing power is needed -- a standard that can measure and compare the throughput and price/performance of various transaction processing systems.

Vendors of transaction processing systems quote Transaction Per Second (TPS) rates for their systems. But there isn't a standard transaction, so it is difficult to verify or compare these TPS claims. In addition, there is no accepted way to price a system supporting a desired TPS rate. This makes it impossible to compare the price/performance of different systems.

The performance of a transaction processing system depends heavily on the system input/output architecture, data communications architecture and even more importantly on the efficiency of the system software. Traditional computer performance metrics, Whetstones, MIPS, MegaFLOPS, and GigaLIPS, focus on CPU speed. These measures do not capture the features that make one transaction processing system faster or cheaper than another.

This paper is an attempt by two dozen people active in transaction processing to write down the folklore we use to measure system performance. The authors include academics, vendors, and users. A condensation of this paper appears in Datamation (April 1, 1985).

We rate a transaction processing system's performance and price/performance by:

- **Performance** is quantified by measuring the elapsed time for two standard batch transactions and throughput for an interactive transaction.
- **Price** is quantified as the five-year capital cost of the system equipment exclusive of communications lines, terminals, development and operations.
- **Price/Performance** is the ratio Price over Performance.

These measures also gauge the peak performance and performance trends of a system as new hardware and software are introduced. This is a valuable aid to system pricing, sales, and purchase.

We rate a transaction processing system by its performance on three generic operations:

- A simple interactive transaction.
- A mini-batch transaction which updates a small batch of records.
- A utility that does bulk data movement.

We believe this simple benchmark is adequate because:

- The interactive transaction forms the basis for the TPS rating. It is also a litmus test for transaction processing systems –it requires the system have at least minimal presentation services, transaction recovery, and data management.
- The mini-batch transaction tells the IO performance available to the Cobol programmer. It tells us how fast the end-user IO software is.

- The utility program is included to show what a really tricky programmer can squeeze out of the system. It tells us how fast the real IO architecture is. On most systems, the utilities trick the IO software into giving the raw IO device performance with almost no software overhead.

In other words, we believe these three benchmarks indicate the performance of a transaction processing system because the utility benchmark gauges the IO hardware, the mini-batch benchmark gauges the IO software, and the interactive transaction gauges the performance of the online transaction processing system.

The particular programs chosen here have become part of the folklore of computing. Increasingly, they are being used to compare system performance from release to release and in some cases, to compare the price/performance of different vendor's transaction processing systems.

The basic benchmarks are:

DebitCredit: A banking transaction interacts with a block-mode terminal connected via x.25. The system does presentation services to map the input for a Cobol program which in turn uses a database system to debit a bank account, do the standard double-entry bookkeeping and then reply to the terminal. 95% of the transactions must provide one-second response time. Relevant measures are throughput and cost.

Scan: A mini-batch Cobol transaction sequentially scans and updates one thousand records. A duplexed transaction log is automatically maintained for transaction recovery. Relevant measures are elapsed time and cost.

Sort: A disc sort of one million records. The source and target files are sequential. Relevant measures are elapsed time and cost.

A word of caution: these are performance metrics, not function metrics. They make minimal demands on the network (only x. 25 and very minimal presentation services), transaction processing (no distributed data), data management (no complex data structures), and recovery management (no duplexed or distributed data).

Most of us have spent our careers making high-function systems. It is painful to see a metric which rewards simplicity - faster than fancier ones. We really wish this were a function benchmark. It isn't.

Surprisingly, these minimal requirements disqualify many purported transaction processing systems, but there is a very wide range of function and usability among the systems that have these minimal functions.

Our Performance and Price Metrics

What is meant by the terms: elapsed time, cost and throughput? Before getting into any discussion of these issues, you must get the right attitude. These measures are very rough. As the Environmental Protection Agency says about its mileage ratings, “Your actual performance may vary depending on driving habits, road conditions and queue lengths -- use them for comparison purposes only”. This cavalier attitude is required for the rest of this paper and for performance metrics in general --if you don't believe this, reconsider EPA mileage ratings for cars.

So, what is meant by the terms: elapsed time, cost and throughput?

Elapsed Time is the wall-clock time required to do the operation on an otherwise empty system. It is a very crude performance measure but it is both intuitive and indicative. It gives an optimistic performance measure. In a real system, things never go that fast, but someone got it to go that fast once.

Cost is a much more complex measure. Anyone involved with an accounting system appreciates this. What should be included? Should it include the cost of communications lines, terminals, application development, personnel, facilities, maintenance, etc.? Ideally, cost would capture the entire "cost-of-ownership". It is very hard to measure cost-of-ownership. We take a myopic vendor's view: cost is the 5-year capital cost of vendor supplied hardware and software in the machine room. It does not include terminal costs, application development costs, or operations costs. It does include hardware and software purchase, installation, and maintenance charges.

This cost measure is typically one fifth of the total cost-of -ownership. We take this narrow view of cost because it is simple. One can count the hardware boxes and software packages. Each has a price in the price book. Computing this cost is a matter of inventory and arithmetic.

A benchmark is charged for the resources it uses rather than the entire system cost. For example, if the benchmark runs for an hour, we charge it for an hour. This in turn requires a way to measure system cost/hour rather than just system cost. Rather than get into discussions of the cost of money, we normalize the discussion by ignoring interest and imagine that the system is straight-line depreciated over 5 years. Hence an hour costs about 2E-5 of the five-year cost and a second costs about 5E-9 of the five year cost.

Utilization is another tough issue. Who pays for overhead? The answer we adopt is a simple one: the benchmark is charged for all operating system activity. Similarly, the disc is charged for all disc activity, either direct (e. g. application input/output) or indirect (e.g. paging).

To make this specific, let's compute the cost of a sort benchmark which runs for an hour, uses 2 megabytes of memory and two, discs and their controllers.

Package	Package cost	Per hour cost	Benchmark cost
Processor	8OK$	1.8$	1.8$
Memory	15K$	.3$	.3$
Disc	50K$	l. l$	l. l$
Software	50K$	1.1$	1.1$
			4.3$

So the cost is 4.3$ per sort.

The people who run the benchmark are free to configure it for minimum cost or minimum time. They may pick a fast processor, add or drop memory, channels, or other accelerators. In general the minimum-elapsed-time system is not the minimum-cost system. For example, the minimum cost Tandem system for Sort is a one processor two disc system. Sort takes about 30 minutes at a cost of 1.5$. On the other hand, we believe a 16 processor two disc Tandem system with 8 Mbytes per processor could do Sort within ten minutes for about 15$ - six times faster and 10 times as expensive. In the IBM world, minimum cost generally comes with model 4300 processors, minimum time generally comes with 308x processors.

The macho performance measure is throughput --how much work the system can do per second. MIPS, GigaLIPS, and MegaFLOPS are all throughput measures. For transaction processing, transactions per second (TPS) is the throughput measure.

A standard definition of the unit transaction is required to make the TPS metric concrete. We use the DebitCredit transaction as such a unit transaction.

To normalize the TPS measure, most of the transactions must have less than a specified response time. To eliminate the issue of communication line speed and delay, response time is defined as the time interval between the arrival of the last bit from the communications line and the sending of the first bit to the communications line. This is the metric used by most teleprocessing stress testers.

Hence the Transactions Per Second (TPS) unit is defined as:

TPS: Peak DebitCredit transactions per second with 95%of the transactions having less than one second response time.

Having defined the terms: elapsed time, cost and throughput, we can define the various benchmarks.

The Sort Benchmark

The Sort benchmark measures the performance possible with the best programmers using all the mean tricks in the system. It is an excellent test of the input-output architecture of a computer and its operating system.

The definition of the sort benchmark is simple. The input is one- million hundred-byte records stored in a sequential disc file. The first ten bytes of each record are the key. The keys of the input file are in random order. The sort program creates an output file and fills it with the input file sorted in key order. The sort may use as many scratch discs and as much memory as it likes.

Implementers of sort care about seeks, disc IO, compares, and such. Users only care how long it takes and how much it costs. From the user's viewpoint, relevant metrics are:

Elapsed time: the time from the start to the end of the sort program.

Cost: the time weighted cost of the sort software, hardware packages it uses.

In theory, a fast machine with a 100 MB memory could do it in a minute at a cost of 20$. In practice, elapsed times range from 10 minutes to 10 hours and costs vary between 1$ and 100$. A one hour 10$ sort is typical of good commercial systems.

Scan Benchmark

The Sort benchmark indicates what sequential performance a wizard can get out of the system. The Scan benchmark indicates the comparable performance available to end-users: Cobol programmers. The difference is frequently a factor of five or ten.

The Scan benchmark is based on a Cobol program that sequentially scans a sequential file, reading and updating each record. Such scans are typical of end-of-day processing in online transaction processing systems. The total scan is broken into mini-batch transactions each of which scans a thousand records. Each mini-batch transaction is a Scan transaction.

The input is a sequential file of 100 byte records stored on one disc. Because the data is online, Scan cannot get exclusive access to the file and cannot use old-master new-master recovery techniques. Scan must use fine granularity locking so that concurrent access to other parts of the file is possible while Scan is running. Updates to the file must be protected by a system maintained duplexed log which can be used to reconstruct the file in case of failure.

Scan must be written in Cobol, PLI, or some other end-user application interface. It must use the standard IO library of the system and otherwise behave as a good citizen with portable and maintainable code. Scan cannot use features not directly supported by the language.

The transaction flow is:

```
OPEN file SHARED, RECORD LOCKING
PERFORM SCAN 1000 TIMES
BEGIN --Start of Scan Transaction
      BEGIN_TRANSACTION
      PERFORM 1000 TIMES
            READ file NEXT RECORD record WITH LOCK
            REWRITE record
      COMMIT_TRANSACTION
      END --End of Scan Transaction
CLOSE FILE
```

The relevant measures of Scan are:

Elapsed time: The average time between successive BeginTransaction steps. If the data is buffered in main memory, the time to flush to disc must be included.

Cost: the time-weighted system cost of Scan.

In theory, a fast machine with a conventional disk and flawless software could do a Scan in .1 seconds. In practice, elapsed times range from 1 to 100 seconds while costs range from .001$ to .1$. Commercial systems execute Scan for a penny and take about ten seconds.

DebitCredit Benchmark

The Sort and Scan benchmarks have the virtue of simplicity. They can be ported to a system in a few hours if it has a reasonable software base - a sort utility, a Cobol compiler, and a transactional file system. Without this base, there is not much sense considering the system for transaction processing.

The DebitCredit transaction is a more difficult benchmark to describe or port - it can take a day or several months to install depending on the available tools. On the other hand, it is the simplest application we can imagine.

A little history explains how DebitCredit became a de facto standard. In 1973 a large retail bank wanted to put its 1,000 branches, 10,000 tellers and 10,000,000 accounts online. They wanted to run a peak load of 100 transactions per second against the system. They also wanted high availability (central system availability of 99.5%) with two data centers.

The bank got two bids, one for 5M$ from a minicomputer vendor and another for 25M$ from a major-computer vendor. The mini solution was picked and built [Good]. It had a 50K$/TPS cost whereas the other system had a 250K$/TPS cost. This event crystallized the concept of cost/TPS. A generalization (and elaboration) of the bread-and-butter transaction to support those 10,000 tellers has come to be variously known as the TPl, ET1, or DebitCredit transaction [Gray].

In order to make the transaction definition portable and explicit, we define some extra details, namely the communication protocol (x. 25) and presentation services.

The DebitCredit application has a database consisting of four record types. History records are 50 bytes, others are 100 bytes.

- 1,000 branches .1MB random access
- 10,000 tellers 1 MB random access
- 10,000,000 accounts 1 GB random access
- a 90 day history 10 GB sequential access

The transaction has the flow:

```
DebitCredit:
      BEGIN-TRANSACTION
      READ MESSAGE FROM TERMINAL (100 bytes)
      REWRITE ACCOUNT (random)
      WRITE HISTORY (sequential)
      REWRITE TELLER (random)
      REWRITE BRANCH (random)
      WRITE MESSAGE TO TERMINAL (200 bytes)
      COMMIT-TRANSACTION
```

A few more things need to be said about the transaction. Branch keys are generated randomly. Then a teller within the branch is picked at random. Then a random account at

the branch is picked 85%of the time and a random account at a different branch is picked 15% of the time. Account keys are 10 bytes; the other keys can be short. All data files must be protected by fine granularity locking and logging. The log file for transaction recovery must be duplexed to tolerate single failures. Data files need not be duplexed. 95%of the transactions must give less than one second response time. Message handling should deal with a block-mode terminal (e.g. IBM 3270) with a base screen of 20 fields. Ten of these fields are read, mapped by presentation services and then remapped and written as part of the reply. The line protocol is X. 25.

The benchmark scales as follows. Tellers have 100 second think times on average. So at 10 TPS, store only a tenth of the database. At 1TPS, store one hundredth of the database. At one teller, store only one ten thousandth of the database and run .0l TPS.

Typical costs for DebitCredit appear below. These numbers come from real systems, hence the anomaly that the lean-and-mean system does too many disc IOs. Identifying these systems makes an interesting parlor game.

	K-inst	IO	TPS	K$/TPS	¢/T	Packets
Lean and Mean	20	6	400	40	.02	2
Fast	50	4	100	60	.03	2
Good	100	10	50	80	.04	2
Common	300	20	15	150	.75	4
Funny	1,000	20	1	400	2.0	8

The units in the table are:

K-inst: The number of thousands of instructions to run the transaction. You might think that adding 10$ to your bank account is a single instruction (add). Not so, one system needs a million instructions to do that add. Instructions are expressed in 370 instructions or their equivalent and are fuzzy numbers for non-370 systems.

DiscIO: The number of disc IOs required to run the transaction. The fast system does two database IOs and two log writes.

TPS: Maximum Transactions Per Second you can run before the largest system saturates (response time exceeds one second). This is a throughput measure. The good system peaks at 50 transactions per second.

K$/TPS: Cost per transaction per second. This is just system cost divided by TPS. It is a simple measure to compute. The funny system costs 400K$ per transaction per second. That is, it costs 400K$ over 5 years and can barely run one transaction per second with one second response time. The cost/transaction for these systems is .5E-8 times the K$/TPS.

¢/T: Cost per transaction (measured in pennies per transaction). This may be computed by multiplying the system $/TPS by 5E-9.

Packets: The number of X. 25 packets exchanged per transaction. This charges for network traffic. A good system will send two x. 25 packets per transaction. A bad one will send four times that many. This translates into larger demands for communications bandwidth, longer response times at the higher costs. X. 25 was chosen both because is standard and because it allows one to count packets.

Observations On The DebitCredit Benchmark

The numbers in the table on page 9 are ones achieved by vendors benchmarking their own systems. Strangely, customers rarely achieve these numbers - typical customers report three to five times these costs and small fractions of the TPS ratings. We suspect this is because vendor benchmarks are perfectly tuned while customers focus more on getting it to work at all and dealing with constant change and growth. If this explanation is correct, real systems are seriously out of tune and automatic system tuning will reap enormous cost savings.

The relatively small variation in costs is surprising - the TPS range is 400x but the K$/TPS range is 10x. In part the narrow cost range stems from the small systems being priced on the minicomputer curve and hence being much cheaper than the mainframe systems. Another factor is that disc capacity and access are a major part of the system cost. The disc storage scales with TPS and disc accesses only vary by a factor of 5. Perhaps the real determinant is that few people will pay 400 times more for one system over a competing system.

There are definite economies of scale in transaction processing - high performance systems have very good price/performance.

It is also surprising to note that a personal computer with appropriate hardware and data management software supports one teller, scales to .01 TPS, and costs 8K$ - about 800K$/TPS! Yes, that's an unfair comparison. Performance comparisons are unfair.

There are many pitfalls for the data management system running DebitCredit. These pitfalls are typical of other applications. For example, the branch database is a high-traffic small database, the end of the history file is a hotspot, the log may grow rapidly at 100 TPS unless it is compressed, the account file is large but it must be spread across many discs because of the high disc traffic to it, and so on. Most data management systems bottleneck on software performance bugs long before hardware limits are reached [Gawlick], [Gray, et. al.]

The system must be able to run the periodic reporting – sort-merge the history file with the other account activity each night to produce 1/20th the monthly statements. This can be done as a collection of background batch jobs that run after the end-of-day and must complete before the next end-of-day. This accounts for the interest in the Scan and Sort benchmarks.

Criticism

Twenty four people wrote this paper. Each feels it fails to capture the performance bugs in his system. Each knows that systems have already evolved to make some of the assumptions irrelevant (e. g. intelligent terminals now do distributed presentation services). But these benchmarks have been with us for a long time and provide a static yardstick for our systems.

There is particular concern that we ignore the performance of system startup (after a crash or installation of new software), and transaction startup (the first time it is called). These are serious performance bugs in some systems. A system should restart in a minute, and should NEVER lose a 10,000 terminal network because restart would be unacceptably long. With the advent of the 64 Kbit memory chip (not to mention the l Mbit memory chip), program loading should be instantaneous.

The second major concern is that this is a performance benchmark. Most of us have spent our careers making high-function systems. It is painful to see a metric which rewards simplicity – simple systems are faster than fancy ones. We really wish this were a functionality benchmark. It isn't.

In focusing on DebitCredit, we have ignored system features that pay off in more complex applications: e. g. clustering of detail records on the same page with the master record, sophisticated use of alternate access paths, support for distributed data and distributed execution, and so on. Each of these features has major performance benefits. However, benchmarks to demonstrate them are too complex to be portable.

Lastly, we have grave reservations about our cost model.

First, our “cost” ignores communications costs and terminal costs. An ATM costs 50 K$ over 5 years, the machine room hardware to support it costs 5 K$. The communications costs are somewhere in between. Typically, the machine room cost is 10%of the system cost. But we can find no reasonable way to capture this "other 90%" of the cost. In defense of our cost metric, the other costs are fixed, while the central system cost does vary by an order of magnitude,

Second, our “cost” ignores the cost of development and maintenance. One can implement the DebitCredit transaction in a day or two on some systems. On others it takes months to get started. There are huge differences in productivity between different systems. Implementing these benchmarks is a good test of a system's productivity tools. We have brought it up (from scratch) in a week, complete with test database and scripts for the network driver. We estimate the leanest-meanest system would require six months of expert time to get DebitCredit operational. What's more, it has no Sort utility or transaction logging.

Third, our “cost” ignores the cost of outages. People comprise 60% of most DP budgets. People costs do not enter into our calculations at all. We can argue that a system with 10,000 active users and a 30 minute outage each week costs 100 K$/TPS just in lost labor over five years. Needless to say, this calculation is very controversial.

In defense of our myopic cost model, it is the vendor's model and the customer's model when money changes hands. Systems are sold (or not sold) based on the vendor's bid which is our cost number.

Summary

Computer performance is difficult to quantify. Different measures are appropriate to different application areas. None of the benchmarks described here use any floating point operations or logical inferences. Hence MegaFLOPS and GigaLIPS are not helpful on these applications. Even the MIPS measure is a poor metric - one software system may use ten times the resources of another on the same hardware.

Cpu power measures miss an important trend in computer architecture: the emergence of parallel processing systems built out of modest processors which deliver impressive performance by using a large number of them. Cost and throughput are the only reasonable metrics for such computer architectures.

In addition, input-output architecture largely dominates the performance of most applications. Conventional measures ignore input-output completely.

We defined three benchmarks, Sort, Scan and DebitCredit. The first two benchmarks measure the system's input/output performance. DebitCredit is a very simple transaction processing application.

Based on the definition of DebitCredit we defined the Transaction Per Second (TPS) measure:

TPS: Peak DebitCredit transactions per second with 95%of the transactions having less than one second response time.

TPS is a good metric because it measures software and hardware performance including input-output.

These three benchmarks combined allow performance and price/performance comparisons of systems.

In closing, we restate our cavalier attitude about all this: “Actual performance may vary depending on driving habits, road conditions, and queue lengths. Use these numbers for comparison purposes only”. Put more bluntly, there are lies, damn lies and then there are performance measures.

References

[Gibson] Gibson, J. C., "The Gibson Mix", IBM TR00.2043, June 1970.

[Gawlick] Gawlick, D., "Processing of Hot Spots in Database Systems", Proceedings of IEEE COMPCON, San Francisco, IEEE Press, Feb.1985

[Gray] Gray, J., "Notes on Database Operating Systems", pp. 395-396. In *Lecture Notes in Computer Science*, Vol. 60, Bayer-Seegmuller eds., Springer Verlag, 1978

[Gray2] Gray, J., Gawlick, D., Good, J. R., Homan, P., Sammer, H.P., "One Thousand Transactions Per Second", Proceedings of IEEE COMPCON, San Francisco, IEEE Press, Feb. 1985. Also, Tandem TR 85.1.

[Good]Good, J. R., "Experience With a Large Distributed Banking System", IEEE Computer Society on Database Engineering, Vol. 6, No. 2, June 1983.

[Anon Et Al] Dina Bitton of Cornell, Mark Brown of DEC, Rick Catell of Sun, Stefano Ceri of Milan, Tim Chou of Tandem, Dave DeWitt of Wisconsin, Dieter Gawlick of Amdahl, Hector Garcia-Molina of Princeton, Bob Good of BofA, Jim Gray of Tandem, Pete Homan of Tandem, Bob Jolles of Tandem, Tony Lukes of HP, Ed Lazowska of U. Washington, John Nauman of 3Com, Mike Pong of Tandem, Alfred Spector of CMU, Kent Trieber of IBM, Harald Sammer of Tandem, Omri Serlin of FT News, Mike Stonebraker of Berkeley, Andreas Reuter of U. Kaiserslutern, Peter Weinberger of ATT.