the Availability Digest

Adding Availability to Performance Benchmarks

September 2007

In almost six decades of computer development, availability has been the poor cousin of performance. While performance has increased by a factor of over 10,000, from the megahertz processors of the late 1950s to the multi-gigahertz multicore processors of today, availability of single processor systems has improved only by a factor of 100, from one 9 to three 9s.

Over the years, there have been many formalized performance benchmarks that have been used to measure and compare the performance of different systems (and to spawn the new technology of benchmarketing). In the transaction processing world, the most used benchmarks are those of the TPC, the Transaction Processing Performance Council.

The History of Transaction Processing Benchmarks

Emerging out of the batch computing models of the 1960s and 1970s was the advent of online data processing. Relatively unsophisticated users would execute rather simple transactions against an online database. Thus was born OLTP, the online transaction processing model of computing that lives today.[1]

As this technology took hold, competition for the marketplace become intense. Clearly, a need existed for a measurement that could be used to compare systems.

In the mid-1980s, vendors began to make claims based on the TP1 benchmark originally developed by IBM. This benchmark measured the performance of a system handling ATM transactions in batch mode without user interaction or network connections. Since there was no policing body, vendors took great liberties in promoting their TP1 results, which ultimately led to great skepticism of these results.

In 1985, a group of two dozen researchers from academia and industry, led by Jim Gray, published an anonymous article[2] suggesting a debit-credit benchmark that was based on a true OLTP configuration with users connected to the system by a network. In addition to the capacity of the system, this proposed benchmark added some additional measures, most importantly response time and cost. This "Debit-Credit" benchmark was immediately adopted by the industry but suffered the same fate as the TP1 benchmark. With no standard body oversight, vendors took such liberties with this benchmark that the waters were even further muddied.

In 1988, Omri Serlin, an industry analyst, realized that a governing performance benchmark standards agency was needed to bring sanity to benchmarking. With eight other companies, he founded the TPC to perform this function.

[1] Kim Shanley, History and Overview of the TPC, www.tpc.org/information/about/history.asp; February, 1998.
[2] Anon, A Measure of Transaction Processing Power, Datamation; April 1, 1985.

By November, 1989, the TPC published its first benchmark, the TPC-A, which was modeled after the Debit-Credit benchmark. TPC-A specified that all benchmark procedures be publicly disclosed in a formal report.

The TPC also formalized the old TP1 benchmark as the TPC-B benchmark for measuring the batch performance of systems.

In 1992, the TPC-C benchmark was released by the TPC. This benchmark is based on an order entry system and greatly expands the processing activities of the system. While other benchmarks have since been promulgated, TPC-C is the benchmark in most common use today for OLTP systems, As with the TPC-A benchmark, the primary system attributes that are measured by TCP-C are the transaction capacity of the system while achieving a specified response time and the cost of the system measured in terms of the cost of a transaction/second of capacity.

What Happened to Availability?

TPC-C and the other transaction processing benchmarks focus on transaction throughput and transaction cost. But it seems that they miss a very important attribute – availability. After all, a system that is down has zero performance.

Which is the better system for your business – (a) a system that costs $1,000 per tps (transactions per second) and is down for 10 hours per year or (b) a system that costs twice as much but is down for only 12 minutes per year? Of course, the answer is, “That depends.” It depends upon the transaction rate that is required for the system, and it depends upon the cost of downtime.

Let us say that the system must handle 1,000 transactions per second and has a downtime cost of $100,000 per hour. The example system (a) would have an initial cost of $1,000,000 and a downtime cost per year of $1,000,000. Over a five-year period, the system cost would amount to $5,000,000.

On the other hand, system (b) would have an initial cost of $2,000,000 and a five-year downtime cost of $100,000 (it would be down for a total of one hour over five years). Thus, its total cost over a five-year period would be $2,100,000. Clearly, system (b) is the proper choice.

On the other hand, if the required transaction rate was 100 tps, and if the cost of downtime was $1,000 per hour, then by the same reasoning, system (a) would have a five-year cost of $150,000 and system (b) would have a five-year cost of $201,000. Clearly, in this case, system (a) is the correct choice.

But how can this choice be made with the current benchmarks? We know what the capacity of a system is and how much an increment of capacity costs. But we have no idea as to how much downtime to expect. To do this, a measurement of availability must be a reported attribute of the benchmark. Unfortunately, this is not the case in today’s benchmarks.

There is, however, a major hurdle to measuring availability. Today’s systems are so reliable that it will take years of system time to measure availability to a reasonable degree of accuracy. By the time we can reasonably report that attribute, the system will have been retired long ago.

But there is a solution to this dilemma, and that is to not use availability as the metric but rather to use recovery time.[3] The argument for this becomes clear as we consider the improvements in availability over the last decades.

[3] Interestingly, an early TPC proposed benchmark in the 1990s included a measure of system recovery time; but this benchmark was never approved by the Council.

The History of Availability

Starting with One 9

In the 1950s, the early commercial computers from IBM, Univac, and RCA were all vacuum tube computers (they followed the mechanical relay computers developed for early computer research). They exhibited availabilities of 90%.

For instance, the Whirlwind computer at MIT, operational in the late 1950s, had an average uptime of about eight hours. Since many programs took much longer than that to run, the concept of frequently checkpointing application state to persistent memory was born of necessity.

Transistors Bring Two 9s

In the 1960s, a new device – the solid state transistor – replaced vacuum tubes. With the elimination of the vacuum tube filament that would periodically burn out like a light bulb, the transistor brought with it an order of magnitude more reliability as compared to vacuum tubes. New systems such as the IBM 7070 and 7090 exhibited availabilities in the order of 99%.

The transistor went on to spawn integrated circuits and today's chip technology. Solid state technology has been the prime driver in the tremendous improvements in system performance since the time of its introduction.

Three 9s with Redundancy and QA

As data processing became mainstream with online applications, two 9s was no longer acceptable in many applications. The industry responded to this need with hardened computers. These systems provided redundant components for those most likely to fail. RAID disk arrays continued in operation even in the presence of a drive failure. Multiprocessor architectures using symmetric multiprocessing (SMP) survived the failure of a processor. Dual power supplies, dual fans, and dual network connections were provided.

Improved software QA reduced the incidence of software faults. Improved operational procedures reduced the frequency of operator errors. Together, these improvements brought an order of magnitude improvement in system availability. Commonly known today as industry-standard servers, these systems today provide availabilities on the order of 99.9%.

Enter Fault Tolerance and Four 9s

There were many single faults that could still take down a system. In the 1970s, fault-tolerant systems were introduced and are marketed today by HP (its NonStop servers) and Stratus Technologies (its ftServers). These systems provide full redundancy for all components and can withstand any single component failure. They often can continue in operation even in the presence of multiple hardware failures and many software failures.

Fault-tolerant systems improved system availability by another order of magnitude. Today's fault tolerant systems can exceed 99.99% uptime when all sources of failure are considered – hardware, software, operator, and environmental faults.

The Availability Barrier

All of these system improvements were focused on one thing – increasing the time before failure of the system (the mean time before failure, or MTBF). But systems had become so reliable that the advantage of further improvements in system MTBF had begun to lose meaning.

This is because further improvements in hardware and operating system reliability became overshadowed by the seemingly persistent failure rate of application software and the equally persistent system failure rate induced by operator errors. Many of these systems were exhibiting basic availabilities of five 9s at the hardware and operating system level, but application software bugs and operator errors were keeping availabilities stuck at the three or four 9s level. Clearly, continued improvements in system MTBF were not going to get us to the next level of availability.

If we look at the equation for availability, *A*:

$$A = \text{MTBF}/(\text{MTBF}+\text{MTR})$$

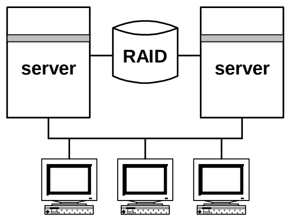
where MTR is the mean time to recovery, we see that we can improve the availability by increasing MTBF or by decreasing MTR. Since MTBF seems to be stuck by software and operator errors, we must now focus our attention on MTR, or recovery time.

Basically, we are saying “Let it fail, but fix it fast.” In the extreme, if we can fix any failure so fast that the users are unaware of the failure, we have, in effect, achieved 100% availability.

There have been two major advances that attack the availability problem by reducing the recovery time, MTR.

Clusters Bring Five 9s

About the time that fault-tolerant systems were being introduced, cluster technology was also being introduced. A cluster consists of two or more processors, each with a physical connection to a common database (typically a RAID array). Each cluster is running a different application; but should it fail, another processor in the cluster can take over its functions by connecting to the application database of the failed processor, bringing up that application, and seizing the IP address used by external users to access that application.



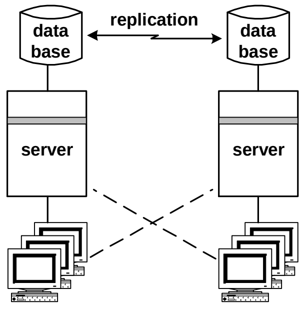
A Cluster

The advantage of a cluster is that its failover time can be measured in minutes, whereas the recovery time for industry-standard servers and fault-tolerant systems is often measured in hours as they are repaired and their databases recovered. This is because in a cluster there is always another processor that is immediately available to take over the functions of the failed processor. The starting of applications, the checking of the database for contamination as the result of the failure, and the switching of users is usually measured in minutes (unless significant database contamination has occurred).

If an industry-standard server in a cluster fails about once every six months and is recovered in three minutes, then its applications are experiencing an availability of almost five 9s.

Six 9s and Beyond with Active/Active Systems

In the late 1990s, active/active systems were introduced. These systems comprise a geographically distributed set of processing nodes and databases. Each of the processing nodes is cooperating in a common application using its assigned database (often, though not necessarily, directly attached). Each database contains a copy of the application database. Therefore, a transaction may be routed to any node in the application network.



An Active/Active System

The database copies are kept in synchronism via data replication. Whenever one system makes an update to its database copy, that update is replicated to all of the other database copies in the application network so that all copies contain an updated application database.

Should a processing node or its database fail, all that is required is to move the users that were being served by the failed node to a surviving node. This can often be done in seconds, thus providing an availability that is one or two orders of magnitude better than clusters.

In addition, since the processing nodes in an active/active system can be geographically distributed, recovery from natural or manmade disasters that take down an entire site is inherent in the architecture.

Thus, active/active systems today are providing availabilities in excess of six 9s.

Recovery Time as a Benchmark Attribute

As pointed out earlier, an accurate measure of availability may be impossible to obtain in a reasonable period of time unless there are hundreds of systems available from which to obtain failure statistics. This is clearly not feasible.

However, with the level of MTBF achieved by today's systems, a measure of availability itself is perhaps less meaningful than the time that the systems take to recover. That is, let it fail, but fix it fast. Very short recovery times are achievable today, even with single systems. A single system can be backed up by replicating its database to a backup system, thus allowing the backup system to quickly restore operations to the users, perhaps in fractions of hours. Clusters provide recovery times measured in minutes, and active/active systems provide recovery times measured in seconds.

The beauty of recovery time as a performance metric is that it is easily measurable. Certainly, the failure conditions under which recovery time is to be measured must be specified, but this is an exercise better left to the standards organizations such as the TPC. The downside is that the average downtime per year - i.e., the system availability – is not known because the system failure interval (its MTBF) cannot be accurately measured over a reasonable period of time. But the cost of a failure can be determined.

The result is that the cost analysis focus is shifted from total system downtime to the number of failures required to balance the cost between two systems.

To get a further insight into this, let us consider our previous example in which system (a) costs $1,000 per 1,000 tps and system (b) costs $2,000 per 1,000 tps. Furthermore, system (a) has a recovery time of one hour and system (b) has a recovery time of three minutes (1/20 of an hour).

If 100 transactions per second must by supported, and the cost to the enterprise of downtime is $100,000 per hour, system (a) will cost $100,000 plus $100,000 per failure. The cost of system (b) will be $200,000 plus $5,000 per failure. After the second failure, system (b) has a significantly lower cost than system (a). System (b) is therefore the preferred choice according to these criteria providing multiple system failures are expected over the life of the system.

Summary

Current performance benchmarks tell only part of the comparative story between systems. The cost of downtime in many applications is quite significant. The cost of a system as determined by the current TPC-C benchmark may be quite impressive, but its recovery time could well make the system more expensive than other systems to which it is being compared when the cost of downtime is considered.

For instance, a more expensive fault-tolerant system might be shown to be significantly less expensive over the life of the system than a standard server when recovery time and the cost of downtime are considered. Similarly, an active/active complex of fault-tolerant systems might be less expensive than a single fault-tolerant system.

Recovery time is not an expensive measurement to make in the context of the cost of the TPC-C benchmark. Isn’t it time to add this attribute to our performance benchmarks so that users can make a truly informed decision?