"Achieving Century Uptimes"
An Informational Series on Enterprise Computing

As Seen in *The Connection*, An ITUG Publication
December 2006 – Present

About the Authors:

Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein, have a combined experience of over 90 years in the implementation of fault-tolerant, highly available computing systems. This experience ranges from the early days of custom redundant systems to today's fault-tolerant offerings from HP (NonStop) and Stratus.

***Gravic, Inc*.**
Shadowbase Products Group
17 General Warren Blvd.
Malvern, PA 19355
610-647-6250
www.ShadowbaseSoftware.com

Achieving Century Uptimes

Part 8: Let's Make Availability a Part of Performance Benchmarking

January/February 2007

Dr. Bill Highleyman
Dr. Bruce Holenstein
Paul J. Holenstein

Availability – The Poor Cousin

Performance benchmarks have become an accepted comparison of system costs and capabilities. The commonly used TPC-C benchmark for transaction processing systems[1] measures the capacity of a system in terms of transactions per minute (tpm) and the cost of the system per tpm. This is an objective and invaluable benchmark for sizing, pricing, and comparing systems.

But nowhere is there a defined benchmark for system availability. Isn't system availability as important an attribute as system performance? After all, if a system is down, its performance is zero.

There is some historical merit to the lack of consideration of an availability benchmark. There is also a technical difficulty associated with the measurement of availability. Let us look at these and come up with an answer.

The Historical Importance of Availability

In our previous article, published in the November/December issue of *The Connection* and entitled What is the Availability Barrier, Anyway?, we traced the history of system availability for commercial data processing systems (that is, for online, real-time OLTP systems). From the early vacuum tube systems of the 1950s to the fault-tolerant systems of today, we have improved single-system availability by a factor of 1,000 – from one 9 to four 9s.

But this is as far as we could go at the single-system level. It is not because we cannot build more reliable hardware and software but because other sources of failure have become prominent. These include application software bugs, operator errors, environmental errors, network failures, and so on. In fact, today's hardware and operating system software often achieve five 9s or more of availability; but these faults are seldom seen because of the predominance of other faults. There is not much advantage to making our underlying systems more reliable.

To carry availability to the next step, redundancy has been extended to the system level. Cluster and active/active architectures use multiple systems backing up each other with fast failover to obtain availabilities of five 9s and beyond. In effect, *let it fail, but fix it fast* is the new mantra for what we call *continuous availability*.

[1] Defined by the Transaction Processing Performance Council. See http://www.tpc.org/tpcc/detail.asp for a description.

But few seem to care about availability when they are initially pricing their system options. Systems still seem to be sold based primarily on price/performance ratios. Fault-tolerant systems have been around for over three decades but are just barely making it into the mainstream. Clusters have made some inroads, but active/active architectures are still only for the pioneers.

Years ago, this lack of concern for availability was justified because applications were just not that critical. Today, however, with enterprises becoming almost totally dependent upon their IT resources, the cost of downtime is rapidly escalating. Some companies, especially in the financial arena, report downtime costs exceeding several hundred thousand dollars per hour. 911 system failures in the U.S. could cause loss of life or property. Companies providing services to the general population could experience the "CNN moment" of adverse publicity and even large losses in their market value (this has happened, for instance, to both eBay and AOL[2]).

It is now time to factor in the availability of a system as well as its performance and cost. But management has no tools to do this. We must give them an **availability benchmark**.

The Availability Benchmark Dilemma

This leads to a conundrum – a riddle of sorts. How can we measure accurately in a few weeks an event that might take years to occur? A cluster with five 9s availability will be down an average of about five minutes per year. But that does not mean that it will be down for five minutes each year. Rather, more likely, it will be down for five hours every sixty years.

To get an accurate measurement of this sort of availability, one needs thousands of system hours. This can be achieved either by having a few systems and measuring them over hundreds of years or by having thousands of systems and collecting downtime statistics on all of them. The first certainly is untenable. The second violates the principle that the availability benchmark is to be used as a tool for system selection and thus puts the cart before the horse, so to speak.[3]

However, there is a solution. In our previous article, mentioned above, we noted that there are three common measures of reliability – availability (A), mean time before failure (mtbf), and mean time to restore (mtr). We argued that so far as commercial data processing is concerned, it is mtr that counts. If a system fails daily and takes one second to be restored to service, users will probably not even notice. If the system fails monthly and is down for one minute, this might be a minor aggravation but generally may be quite acceptable. If the system is down one week every 200 years, the users will love it unless that one fateful week happens next week.

Although availability and mtbf in the ranges in which we are interested may not be measurable, fortunately mtr is readily measurable. If you accept the above arguments, the conclusion is obvious – let's include mtr in our performance benchmarks.

[2] Joseph Williams, *Avoiding the CNN Moment*, IEEE Computer Society Library; March/April,2001.

[3] It is this technique that has allowed us to ascertain the availability of high availability (HA) and cluster systems in hindsight. The availability of active/active systems still is supported primarily only by availability theory.

The History of Performance Benchmarks

Before we proceed with what availability benchmarking might look like, it is useful to review the history of performance benchmarking for commercial data processing systems.[4]

In the early 1980s, online applications began to appear. Vendors began to make claims based on IBM's TP1 benchmark, which only measured batch performance for ATM transactions.

Then in 1985, Jim Gray of Tandem Computers and others anonymously published a description of a new benchmark[5] that truly measured online performance. It was based on a simple banking application and became known as the "DebitCredit" benchmark.

Both benchmarks suffered a common problem. Since there was no control, exaggerated claims of system performance proliferated. One industry analyst had enough. In 1988, Omri Serlin convinced eight companies to form a performance standards organization named the Transaction Processing Performance Council (TPC) to define and audit benchmarks.

The TPC's first benchmark was the TPC-A benchmark, which was based on the DebitCredit benchmark. Today's common benchmark for commercial transaction processing systems is the TPC-C benchmark, which simulates a warehouse environment. It is significantly more complex than the TPC-A benchmark, but it gives a more balanced view of the system.

The first published result for a TPC-A benchmark showed a capacity of almost 2,000 transactions per minute (tpm) at a cost of $425 per tpm (that is, a system price of almost $850,000). Today, with the more complex TPC-C benchmark, high-performance systems are showing capacities of 4,000,000 tpm at a price of $3 per tpm (a $12 million system). Small systems are showing capacities of 100,000 tpm at a price of about $0.75 per tpm (a $75,000 system). Performance has come a long way in two decades!

Using the Availability Benchmark

We would like to include in the cost of a system the cost of downtime. However, this means that we need to know how much downtime to expect over the life of the system, which by definition is its availability. Unfortunately, as we discussed earlier, it is not feasible to measure the availability (or the mtbf) of today's very reliable systems. All we can accurately measure is the time required to return a system to service (its restore time).

However, the very high reliability of today's hardware and operating systems gives us a reasonable approximation for the amount of downtime that a system will suffer. This is because the bulk of system failures today are not caused by hardware or operating system faults. They are caused by operator errors, application program errors, and environmental faults. We know from experience that industry-standard servers seem to fail about once or twice per year, and fault-tolerant servers seem to fail about once every four or five years. We might want to tune these

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

mtbf numbers; but as we shall see, these numbers will give us a basis to compare system costs with downtime included, if not the actual downtime costs.

Forgive a little algebra, but we'll keep it simple. For our benchmark purposes, the cost of a system is its initial cost plus its downtime cost over the life of the system (this ignores the costs of maintenance, operators, sites, etc.) . If we know the hourly cost of downtime and the system restore time, we can calculate the downtime cost of a single failure. For instance, if downtime cost per hour is $100,000, and if the system restore time is two hours, then the cost of a single failure is $200,000.

Thus, if there are *N* failures over the life of the system, the system cost is

$$\text{(system cost)} = \text{(initial cost)} + N^*\text{(failure cost)}$$

Let us consider two different systems, Systems A and B, with different initial costs and different costs of failure. System A is relatively inexpensive but has a relatively long restore time. Compared to System A, System B is more expensive but has a shorter restore time.

Over the life of the system, there will be some number of failures for which the costs of the two systems will be equal. We call this number of failures the *failure index*. Let us see how to calculate this number.

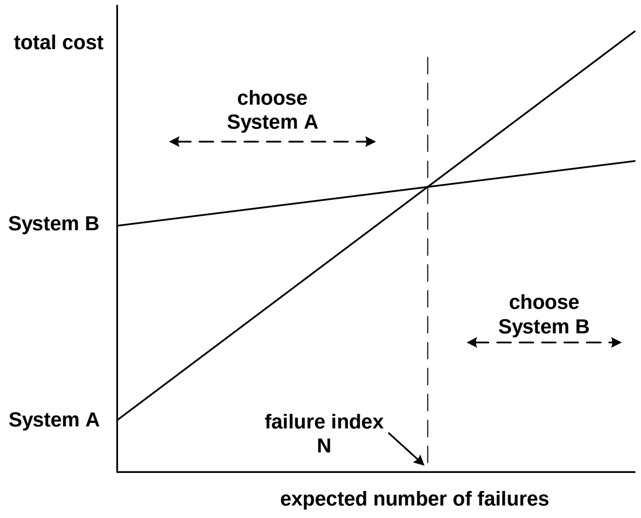
The failure index is that number of failures that make the system costs equal. Setting the system costs for systems A and B equal, we have

$$\text{(initial cost A)} + N^*\text{(failure cost A)} = \text{(initial cost B)} + N^*\text{(failure cost B)}$$

Solving for *N*, we have

$$N = \text{ failure index} = \frac{\text{(initial cost A) - (initial cost B)}}{\text{(failure cost B) - (failure cost A)}}$$

N is the failure index that we are looking for. If there are *N* failures over the life of the system, the costs for System A and System B are the same. If the actual number of failures is greater than the failure index, then System B, with its lower cost of failure, will be less expensive. If the actual number of failures is less than the failure index, then System A, with its lower initial cost, will be less expensive.



Consider two systems as follows over a five-year life span. System A is an industry-standard server (ISS), and System B is a two-node cluster of industry-standard servers. For this example, let us assume the following initial costs and restore times from a single node failure:

	Initial Cost	Restore Time
System A - ISS	\$100,000	4 hours
System B – ISS Cluster	\$500,000	3 minutes

Assuming a failure rate of one failure per year, we would expect System A to fail about five times over its five-year life. System B has two nodes, so it would be expected that one node would fail ten times over the system's five-year life. Let us consider two cases for the cost of downtime - \$5,000 per hour and \$100,000 per hour. The per-failure cost for each system for each of these cases is

Downtime Cost	\$5,000/hour	\$100,000/hour
Per-failure cost – System A	\$20,000	\$400,000
Per-failure cost – System B	\$250	\$5,000

For the \$5,000 per hour downtime cost, the failure index is

$$N = \frac{500 - 100}{20 - 0.25} \approx 20$$

The number of expected failures for both systems over the five-year system life is significantly less than the failure index. Therefore, System A is the system with the least expected cost. At five failures, System A would cost (100,000 + 5 * 20,000) = \$200,000 versus (500,000 + 10 * 250) = \$502,500 for System B with its ten failures.

For the \$100,000 per hour downtime cost case, the failure index is

$$N = \frac{500 - 100}{400 - 5} \approx 1$$

Both systems will fail significantly more times than just once over their five-year life. Therefore, System B is the preferred system. At five failures, System A would cost (100,000 + 5 * 400,000) = \$2,100,000. At ten failures, System B would cost (500,000 + 10 * 5,000) = \$550,000. In summary,

Downtime cost	\$5,000/hour	\$100,000/hour
Total cost – System A	\$200,000	\$2,100,000
Total cost – System B	\$502,500	\$550,000

We have seen here that we can make good comparative judgments about the cost of downtime without knowing exactly what the system availability is. Restore time is a good metric for system reliability. Even though System B costs five times as much as System A to purchase, it can be much less expensive in the long run if downtime costs are high.

Too often, a manager is judged by his short-term performance. He may be held, for instance, to quarterly budgets. Consequently, in the second case, he may choose System A because of its lower initial cost and ignore the long-term consequences of the cost of downtime. He is now a hero and leaves the long-term damage to his successor. An availability benchmark would help to eliminate this problem.

How Do We Measure System Restore Time?

So how do we measure system restore time? This is a task better left to the benchmark standards people. There are many kinds of faults that can take a system down, such as processor faults, software faults, database faults, network faults, operator errors, and others. To subject a system to a complete mix of faults and measure its restore time might well be overkill. After all, most faults simply cause the system to crash; and any crash recovery might be a reasonable test.

From a restore time viewpoint, there are two types of faults – those that require hardware repair followed by a system recovery (loading applications, recovering the database, testing, etc.) and those that only require recovery (such as an application fault or operator error). One might design a set of tests to create a simple mix of these two types of faults. For instance, the restore times might be measured for the cases of pulling and replacing a processor, pulling and replacing one or more disks (as required to create a disk crash), killing an application, and halting the system via an operator command. All faults would be induced while the system is under the full load of the performance benchmark. The resulting restore times would be averaged according to an agreed upon weighting. The result would be the published restore time.

Summary

Current performance benchmarks give us two numbers – the transaction per minute (tpm) capacity of the system and the cost per tpm. Adding an availability test would strengthen this data with the expected restore time and would allow a much more informed choice of systems.

There is no doubt that the definition of the restore time test would require a great deal of work by those knowledgeable in the field. But so did the current performance benchmarks. Once done, this procedure could be invaluable to the system decision-making process and result in better system choices for the long term.

It is time for the vendors of high availability systems such as HP, Stratus, and IBM to work with the Transaction Processing Performance Council or other appropriate standards organizations to meet this critical need.