the Availability Digest

Using an Availability Benchmark

October 2008

In several previous articles, we have talked about availability being the poor cousin to performance when it comes to benchmarks. In fact, for over two decades, the IT industry has depended upon performance benchmarks like TPC-C from the Transaction Processing Council to make informed decisions concerning system purchases. However, in this era of 24/7 requirements for our mission-critical systems, there is still no way to compare the availabilities of diverse systems.

Consulting firms such as Gartner Group and Standish Group have published the results of extensive surveys of the availability experience for broad classes of systems, but there is not yet a recognized means to compare the availabilities of specific systems.

In today's world of gigabit/second processors and petabyte storage systems, is not availability equally important? After all, a system that is down has zero availability. And downtime in many applications has a price tag measured as hundreds of thousands of dollars per hour. Over the life span of a system, its downtime cost can often be multiples of its original cost.

In our most recent article on this subject, entitled Adding Availability to Performance Benchmarks (September, 2007), we suggested that recovery time is a useful and measurable metric appropriate as an availability benchmark. But how does management use this measure to aid them in a purchase decision? We suggest a formal method to do so in this article.

A Review

The solution is to add an availability measure to the current TPC-C transaction-processing benchmark. But how do we measure availability when many current systems are showing failure intervals of years?

In many of our previous Digest articles, we have addressed this issue. In Let's Get an Availability Benchmark (June, 2007), we reviewed the current transaction-processing benchmarks and discussed the problems of adding availability to these benchmarks. We discussed many possible tradeoffs between availability and performance in Availability versus Performance (August, 2007). Yes – availability not only costs money but also costs performance.

In Adding Availability to Performance Benchmarks (September, 2007), we attempted to break the availability-benchmark barrier by pointing out that there were several metrics for availability – availability itself (the proportion of time that a system is up), its failure interval (MTBF – the mean time before failure), and system recovery time (MTR – the mean time to recover). These are related by the well-known equation

$$\text{Availbility} = A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTR}}$$

As we think about it, the measure of reliability is in the eye of the beholder. Any one of these parameters may be chosen as the measure of reliability. For instance:[1]

- In the Hubble satellite, mean time before failure, MTBF, is of paramount importance. It is critical for a system contained in a satellite to be up as long as possible (hopefully, years) because once it fails, it is generally not repairable. (Most satellites have backup systems, but will they be operational when the primary system fails?)

- In a system measuring oil flow in a pipeline, availability, *A*, is the most important measure because when the system is down, oil flow is unknown. Any downtime results in revenue loss for oil delivered.

- In a commercial transaction-processing system, we submit that mean time to recover, MTR, is the most important measure. After all, if the recovery from a fault is so fast that no one notices that there has been a fault, in effect, there has been no fault.

We expand on the reasoning behind recovery time as the availability metric and how it would be used as a benchmark deliverable in the next sections.

The Importance of Recovery Time in Transaction Processing Systems

The importance of recovery time in transaction-processing systems can best be shown by example. Assume that we have a system with an availability of four 9s (it is up 99.99% of the time). There are any number of combinations of recovery time, MTR, and failure interval, MTBF, that lead to an availability of four 9s. Let us look at some cases:

- Case 1: The system is down one second every three hours. The users probably will not even notice this outage as it is within the response time that they expect from the system.

- Case 2: The system is down one minute every week. Users will probably be affected by the outage but will grudgingly accept it. After all, “Computers do fail.” (How often do you reboot your PC?)

- Case 3: The system is down one week every 200 years. The users will be delighted with the system until they all lose their jobs because the company goes out of business. If you say “So what? The system will be gone long before then,” you are missing the point. That bad week could be next week.

Based on these examples, we submit that availability and failure intervals are not nearly as important as recovery time in these sorts of transaction-processing systems. It does not matter how much of the time a system is up or how often a failure occurs so long as the failure interval is so short that the users don’t notice it or at least are not inconvenienced. Therefore, it is recovery time that counts in transaction-processing systems.

Recovery Time as an Availability Benchmark

So how do we measure system recovery time? This is a task better left to the benchmark standards people. There are many kinds of faults that can take down a system, 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 recovery time might well be overkill.

[1] Achieving Century Uptimes Part 7: What is the Availability Barrier, Anyway?, *Connection*; November/December, 2007.

After all, most faults simply cause the system to crash; and any crash recovery might be a reasonable test.

From a recovery-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 recovery 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 was under the full load of the performance benchmark. The resulting recovery times would be averaged according to an agreed upon weighting. The result would be the published recovery time.

However, there is still a leap from knowing the recovery times of competing systems to making a purchase decision.

Using Recovery Time as an Availability Benchmark

To get a further insight into this, let us consider a comparison between two systems. System A costs $1,000 per tps (transactions per second), and System B costs $2,000 per 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 be supported, and if 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. We don't know what the failure intervals are for the two systems, but we do know that 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.

The very high reliability of today's hardware and operating systems gives us a reasonable approximation for the number of failures that a system will suffer over its lifetime. 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 that fault-tolerant servers seem to fail about once every four or five years. We might want to tune these MTBF values; but as we shall see, these numbers will give us a basis to compare system costs with downtime included.

For our benchmark purposes, the total cost of a system is its initial cost and its operating costs plus its downtime cost over the life of the system. Let us call the initial cost and operating costs the "system cost." System cost includes the initial cost of the system plus the costs of maintenance, operators, sites, energy, insurance, etc., over the life of the system.

If we know the hourly cost of downtime and the system recovery 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 recovery time is two hours, then the cost of a single failure is $200,000. We call this the "failure cost."

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

(total cost) = (system cost) + N*(failure cost)

Let us consider two different systems, Systems A and B, with different system costs and different costs of failure. System A is relatively inexpensive but has a relatively long recovery time. Compared to System A, System B is more expensive but has a shorter recovery 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's calculate this number.

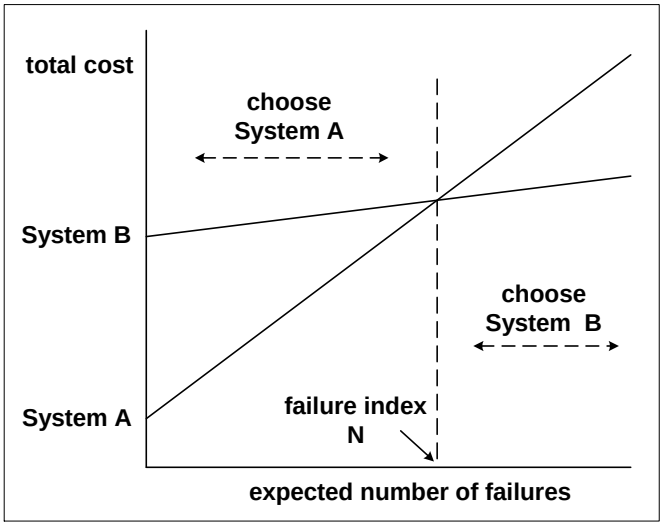
The failure index is that number of failures that makes the total costs equal. Setting the total costs for systems A and B equal, we have

(system cost A) + N*(failure cost A) = (system cost B) + N*(failure cost B)

Solving for *N*, we have

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

N is the failure index for which we are looking. If there are *N* failures over the life of the system, the total 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.



Calculating the Failure Index

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 system costs and recovery times from a single-node failure:

	System Cost	Recovery Time
System A - ISS	$100,000	4 hours (repair time)
System B – ISS Cluster	$500,000	3 minutes (failover time)

Assuming a nodal failure rate of one failure per year, we would expect both System A and System B to fail about five times over its five-year life. (Though System B has two nodes, it is only the failure of the primary node that will take the system down.) 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 either system 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 + 5 * 250) = $501,250 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. System B would cost (500,000 + 5 * 5,000) = $525,000. In summary,

Downtime cost	$5,000/hour	$100,000/hour
Total cost – System A	$200,000	$2,100,000
Total cost – System B	$501,250	$525,000

We have seen here that we can make good comparative judgments about the cost of downtime without knowing exactly what the system availability is. Recovery 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.

Summary

Too often, a manager is judged by his short-term performance. He may be held, for instance, to quarterly budgets. Consequently, he may choose a system 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.

System reliability as measured by recovery time is only one of the parameters involved in a system choice. Performance (which dictates a system's size) and cost are other important benchmarks. We submit that the cost of downtime is an equally important metric. The use of recovery-time measurements incorporates the cost of downtime in the system choice. After all, in many applications, it is the cost of downtime that is the predominant cost over the life of the system. Perhaps equally important, it is recovery time upon which the users of a transaction-processing system judge its effectiveness.