SUN BLUEPRINTS APPROVED

[illegible]

Availability - What It Means, Why It's Important, and How to Improve It

By Richard Mc Dougall

Sun BluePrints™ OnLine - October 1999

Sun microsystems
ENTERPRISE ENGINEERING

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-3755-10
Revision 01, October 1999

Availability - What It Means, Why It's Important, and How to Improve It

Availability has long been one of the key criteria of online systems because when systems are down users, productivity, and critical business processes come to a standstill. The real costs of being unable to deliver service and product to customers can be frighteningly expensive. I recently helped a large customer who claimed that the cost of downtime on their point of sale verification systems was in the order of $5,000,000 per minute of downtime, driving home the point of the real cost of downtime, and the importance for the highest realistic level of system availability.

Impact of Downtime

Not all systems have the same level of dependency on availability. Downtime in some systems may be painful, but the impact may be localized so that only a small group of users are affected. And those users may be completing work that is not time critical. For example, downtime on a small office file server may only delay simple word-processing tasks, which can be completed in a few hours, with a minimal cost for the delay.

Now more than ever, availability has become a critical design criteria -- not to say that availability hasn't been important, but the impact of downtime and exposure has become much greater. The reason for this is that we now provide systems that interact directly with customers, and there is no insulation between the system problems and those customers. Consider an online sales order business, which funnels 100% of its business though a web-based sales order application. When that system is down the company cannot sell product to its customers. More importantly, the frustration experienced by customers from the unavailability of the web application can lead to instant loss of those customers, making the follow-on lost opportunity cost huge. The more successful the business, the more visible the web-based applications are, and the more vulnerable you are to exposure caused by system outages. Today’s E-business applications can go from world-wide leadership and success to newspaper headline disasters in a matter of hours.

Obviously, there is a wide range of the cost of downtime, so it is useful to categorize the impact of downtime into different categories. Many applications can be classified into the following groups:

- Mission critical--If the application is down, then critical business processes and/ or customers are affected in a way that has massive impact on the company’s profitability
- Business Critical--Downtime that is often not visible to customers, but does have a significant cost associated with it
- Task Critical--The outage affects only a few users, or the impact is limited and the cost is insignificant.

The more mission critical oriented your application, the more the focus on availability should be. Unfortunately, increases in availability do not come for free. It is often tempting to try to increase application availability by first spending money on the system platform (e.g., installing a clustered system). But most of the time, it is far more effective to focus your efforts in other areas before you concentrate on the system platform. I’m certainly not saying that investing time and money on the platform is not productive, but often larger gains can be made in other areas first.

The 3 P’s--People, Process, and Product

Let’s look further into what this means -- what areas have the greatest effect on availability? Industry analysts report that the system platform on average accounts for 20% of system availability, while people and practices account for 80%.&nbsp; For example, a simple change to a system parameter can easily cause an outage that could otherwise be avoided with proper testing and a change in control processes.

Process Issues (40% of System Downtime)

Process issues are key, and many lessons can be learned from the processes that have been developed over the last 20 years by the mainframe and UNIX(R) system managers. Processes that increase availability are wide ranged, and careful attention should be paid to analyzing their potential impact at all levels. The most obvious are the practices and processes used for managing the system hosting the applications in question. But remember that the service delivered by an application is often much wider than a single system, instead it’s the interconnection of a number of systems and networks. Any processes applied must encompass every component in the system, from the platform and storage to the network infrastructure that connects the application to your users or customers. We can categorize the process responsibilities into different groups:

- Systems Management Principles
- Standards for Systems Installation &amp; Configuration
- Change Control Practices
- Maintenance and Patch Strategy
- Release Upgrade Processes
- Problem Escalation &amp; Support Agreements
- System Staging and Test
- Support Agreement &amp; Associated Response Times
- Backup and Recovery testing
- System Recovery (reinstallation) Procedures<
- Disaster Recovery Planning

People (40% of Downtime)

It is often said that availability is most effectively implemented when it is considered an attitude, rather than just a business priority. A customer recently said to me that they regularly remind all of their operations staff that they have no room for cowboys, but they encourage and recognize innovation and creativity. The key here is that they have separated the long-term creative potential of their staff from the temptation of short-term quick fixes. Another customer introduced a new policy that requires a peer review of every change that is made to a system. This policy was first taken as a “lack of trust” of the staff. But after a while, everyone could see a visible change in quality. With every change visible, the focus quickly shifted to minimizing the number of public errors in the changes.

Training operations and systems management staff has a definite cumulative effect and is a worthwhile investment. Planning ahead with clear processes and training results in higher quality changes and tasks that complete more quickly. This is most visible at the time it’s most needed--during a system failure, when the time taken to recover a system to an operational state is critical.

Product (20% of downtime)

The system platform is responsible for on average 20% of total downtime, and hence importance should be focussed on designing a system platform architecture for maximum availability. Note that the system platform is more than the hardware: It is the combination of a network, the application(s), the operating system, the supporting system software, and the hardware platform. The choices you make at this level often provide the last few percentage points of availability, taking you from 97 to 99% availability.

A colleague, Brian Wong, is renowned for saying “Fast, Cheap, Safe -- Pick two!”. This is the paradigm we exist in with availability, and the choices we make lead to one corner of the fast/cheap/safe triangle. For example, a single disk drive is cheap, but not reliable or fast, a stripe of disks optimizes for performance yet suffers from a lack of reliability, and mirror of two disks is fast and reliable yet expensive. As you get closer to 100% availability, the amount of money you have to spend increases, so your decisions here must be carefully matched with business requirements.

- The opportunity for system level availability improvement include:
- Network Resilience, including redundant circuits and multi-pathing
- Hardware Platform Hardening - using features such as DR (Dynamic Reconfiguration)
- Reliable Storage (Redundancy using RAID)
- Duplexing of systems (Clustering)

Measuring Availability

Simple engineering methodology states that before you can begin to control something, we must first measure it. The same applies to availability, even more so given the cost of implementing highly available systems can double for just a fraction of percentage of availability.

Availability is typically measured as a percentage of time the system is up, so 100% availability is a system that is always up, and availability is defined as:

$$\text{availability} = 100 \times \text{uptime} \;/\; (\text{uptime} - \text{downtime})$$

The key is obviously to minimize downtime, since as downtime approaches zero availability approaches 100%. Not all downtime results from unexpected system outages, since it also includes scheduled maintenance. Downtime consists of two categories: *planned* and *unplanned*, where unplanned downtime is the result of an unexpected system failure and planned downtime is that from planned system maintenance such as upgrades and patch installs. In some cases, it is possible to eliminate planned downtime from the availability calculation, since the hours of operation of the business process may be restricted (e.g. 9-5), so that scheduled downtime has no affect on any users of the system.

Measuring downtime is a complex problem because it is not simply the time the hardware platform is down. The question to ask is how far up the layers of the system to count in your availability measurement (network, application, operating system, hardware etc.). The answer is almost always everything, from top to bottom, which is often referred to as "end to end availability measurement".

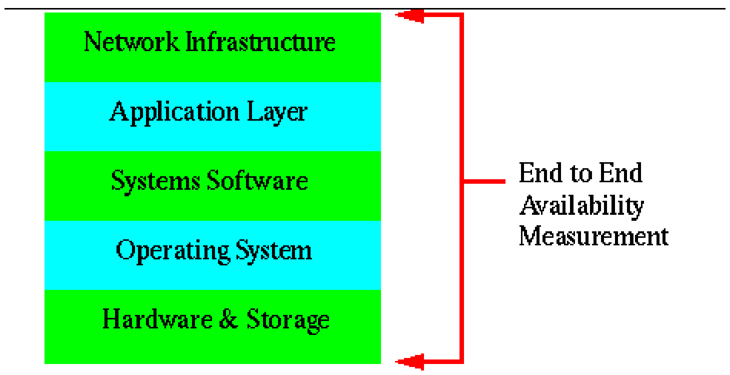

FIGURE 1 The Multiple Levels of Availability Measurement

Operating System and Hardware Platform Measurement

There are several different approaches to measuring application availability, depending on how many layers of the stack are measured. At the low end are approaches that measure when the hardware platform is up and running, such as an SNMP agent and a network monitoring tool. This is the most common form of measurement and shows when the operating system is running and the hardware platform is functioning.&nbsp; Simple monitoring can be done by using a network "ping", and more advanced monitoring can be done with agents such as those bundled with the Sun™ Enterprise SyMON™ system management suite.

The danger of this method is that it often misses failures that affect your real users. For example, a hardware platform can be up and running, with part of the operating system crippled (a corrupted disk and file system), and as a result the application's database is down and the application is unavailable. For this reason, I strongly recommended that you use a more robust method of measurement.

Application Level Measurement

Moving up the stack, take a look at application level measurement. This form of measurement is quite popular, and is often used in clustered environments as the mechanism for determining if a system is functioning correctly. Measuring from the application level provides a greatly improved level of availability measurement because any failure in the storage, hardware, operating system and database will most likely be detected.

Measurement at this level is typically performed by some sort of test transaction, such as a simple application update SQL transaction that updates a dummy record in a database table, or an dummy read invocation of an application level object. The complexity of the test transaction affects the accuracy of the application level availability measurement because not all test transactions will detect all database or application faults, so you must consider how these test transactions are implemented.

Another growing method for application measurement is application instrumentation. This is a technique used to insert probes into the target application at key points, e.g. the start and end of key transactions. These probes can be used to record the frequency and response time of transactions, and can be used for both performance and availability measurements. More applications are being developed with built-in probes, and good examples are SAP R3 and Baan, both deliver instrumented applications. An application level instrumental known as ARM (Application Response Measurement) toolkit can be found at http://www.cmg.org.

Again, there is a small downside to application level measurement, in that it does not always measure the network infrastructure around the systems that host the application.

Network Level Measurement

One of the more accurate ways of measuring application availability is to implement simulated test transactions over the network. This helps ensure that all of the network infrastructure around the application host(s) is included in the measurement. Where possible, try to include the Internet if the application is web based, so that an outside view of the application's availability is captured. If the T-1 line feeding your site goes down or is saturated, then the application is down as far as the users are concerned. There are web sites that offer external network-level measurement as a service, and some of these provide a summarized report of the measured availability--including performance levels.

Performance Implications

It is important to include performance in the availability measurement. A system that is running slow may show up as a system that is available, but there is a threshold that when crossed the system should be deemed unavailable. Performance thresholds are dependent on the type of application being measured, but most of the time any response time greater than 10 or 20 seconds is considered down to an end user.

Modeling Availability

Mathematical modelling of a system to derive availability is complex because the system comprises many components. The problem would be much simpler if each component in the system was not related--but they are. For example, a system of disks, each with a mean time between failure (MTBF) of 500,000 hours could have an MTBF of 500,000 hours (a single disk), or 250,000 hours (2 disks), but what is the availability of a mirrored pair of disks? The answer is not 100% availability because there are scenarios where multiple failures can cause downtime, such as both disks failing at similar times. In a mirrored disk scenario such as this, availability can only be calculated by considering the MTBF of the disks, the time taken to replace failing disks, the disk re-synchronization times, and the list goes on. Because such a simple configuration can lead to a complex set of interdependent calculations, we need to use mathematical chained modeling to simplify the problem.

The most common techniques are Monte Carlo principles and Markov techniques. These techniques may be used to represent the entire system as a series of known states because they break down the problem into state transitions, the time taken, and the probability of moving between states.

The typical parameters used to simulate availability measurements are

- MTBF - the mean time between failure in hours
- MTTR - the mean time to repair a specific failure
- FIT - Faults in time, measured as failures in one billion (10^9) hours<
- LFTR - Chance of a Latent Fault

A simple Markov model describes a system with two nodes.

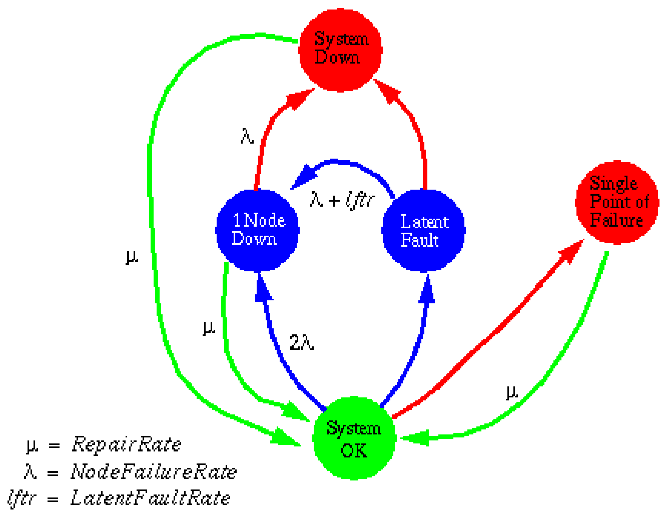

FIGURE 2 Markov Modeling

It's not as complex as it first seems, the diagram simply represents the states that a system of two nodes can be in. The major states are:

- The system is functioning normally
- One node is down
- A single point of failure
- Latent Faults

The parameters used in Markov modeling are rates, which are derivations of the typical MTBF numbers, plus some rates of failures found during testing. Consider the example shown in Figure.1, a system of two nodes. The system starts in the "OK"&nbsp; state. There are probabilities of getting from the "OK" state into on of the other states such as when a single node failure occurs, and the system enters the "1 Node Down" state. From there, there is a very small probability that a second node failure will occur, moving to the "System Down" state. A higher probability is that the node will be repaired and the system will re-enter the "System OK state".

The idea of this quick modeling introduction was to show that system availability consists of much more than just the availability of the components. A true availability model would consider the hardware MTBF and MTTR, system design, service personnel response times and other factors that affect overall availability. Further details on availability modeling will be covered in later BluePrint articles, or as an alternative more information can be found in Markov and Monte Carlo mathematical texts.

Improving Availability

This article is targeted as an introduction to some of the concepts and principles for developing a strategy to improve overall application availability. As we move further into best practices on Availability, we will detail some of the processes and architectures required to achieve higher availability. Stay Tuned!

The Sun BluePrints™ Program

The Sun BluePrints program, articles, and books are designated to provide Sun's recommended practice in many of these areas. Throughout the life of the Sun BluePrints program we expect to see more and more topics covered, some with step-by-step guides to practices, and others that are methodologies that help set direction and strategy. We also encourage you to engage with the Sun BluePrints program and share your practices with others though the online magazine with online articles such as this. See `http://www.sun.com/blueprints` for further information.

Author's Bio: Richard Mc Dougall

Richard has over 11 years of UNIX experience including application design, kernel development and performance analysis, and specializes in operating system tools and architecture.