the **Availability Digest**

www.availabilitydigest.com

Maximizing Availability in Everyday Systems
July 2010

Not everyone runs active/active systems for continuous availability. Many of you don't even run a backup system. Downtime is simply a cost of business with which to be dealt.

But downtime is still costly. Even if you don't run redundant systems and are willing to accept some amount of downtime, how can you reduce it through tuning up your operating procedures and purchasing decisions? At the last month's HP Technology Forum held in Las Vegas, Keith Parris, a Systems/Software Engineer at HP, presented an excellent set of insights and tips for maximizing your availability no matter what kind of system you run. In this article, we review his observations and suggestions.

Know Your Most Failure-Prone Components

Hardware

The hardware components that fail most share the following characteristics:

- They move. Fans and spinning disks are examples of moving parts.
- They are heat generators. Power supplies lead this category.
- They are very new. Components that have not yet been "burned in" suffer "infant mortality."
- They are very old. They are worn out. This gives ordinary servers and disk farms a typical life span of three to five years.

Software

Newly written software is prone to bugs that can take a system down. Avoid version 1.0 of anything. Wait until it has proven itself with a modicum of field experience by others. The same is true for new features and fixes in existing software.

Seldom-used code paths such as those for error handling and recovery are not field-proven by extensive use. Therefore, as you load your system more and more, these code paths become more frequently used; and a latent bug may finally bite you.

Should your environment change, old code that has worked for years may suddenly have problems. Be aware of network upgrades and changes in other applications or partner services with which your application must interface. Hardware technology upgrades and operating system version updates may cause problems. Even the speed at which disks, CPUs, and networks operate can reveal hidden race conditions.

Facilities and Infrastructure

Even if your data center is in a closet back by the lunchroom, there are steps that you should take to ensure its survivability. For one thing, you must be prepared for the almost certainty of a power failure and must provide a UPS (uninterruptable power supply) and a generator to ride through extended power outages. Larger data centers should be powered by two different substations and have redundant UPS systems and generators. Servers and other equipment should have dual power supplies with dual power connectors to the redundant power sources.

Your air conditioning system must be included in your power needs calculation. It should have excess capacity to handle extremely hot days and the inevitable growth of your data-processing needs. Just like power backup, air conditioning should be redundant.[1]

The Uptime Institute (www.uptimeinstitute.org) certifies data centers into four tiers - Tier 1 (Basic), Tier 2 (Redundant Capacity Components), Tier 3 (Concurrently Maintainable), and Tier 4 (Fault Tolerant and Concurrently Maintainable). The Institute makes the very important statement that even a Tier 4 data center will not provide five 9s (99.999% uptime) because of data center outages caused by anything from the accidental operation of the EPO (Emergency Power Off) switch to a fire event. To achieve this level of reliability, multiple data centers backing each other up with fast failover are required.

Eliminate Single Points of Failure

Have at least two of everything. This seems like an obvious statement, but single points of failure (SPOFs) are not always obvious. For instance, is your DNS server redundant? Is your RAID storage system located under a water sprinkler?

Though your attention should be focused first on those components that can fail the most often, consider the consequences of any component failure. Even reliable components fail. Design to accommodate a failure of any component. And what good is a redundant component if it proves to be nonoperational when it is needed? Be sure to periodically test any standby components to ensure that they continue to be operational.

Components are not the only things which can represent SPOFs. Capacity overloads of servers, storage, networks, air conditioning, or power feeds will also cause user services to be lost.

Another SPOF can be your operations staff. Cross-train members of your staff so that no individual becomes a single point of failure.

Even an entire geographically-dispersed disaster-tolerant cluster can be an SPOF. Some customers have implemented dual, independent disaster-tolerant clusters with provisions for moving customers from one cluster to another if needed. A facility such as HP's heterogeneous Reliable Transaction Router (RTR) can be used to route transactions to the two clusters to keep them synchronized.

Detect and Repair Failures Rapidly

This goes without saying if the component that failed is not redundant. While it is down, your system is down.

---

[1] Editor's note: Fan-driven cooling using outside air can provide an economic backup in many instances. Several years ago, system operators knocked out windows in an eighth-floor data center in lower Manhattan, New York City, to cool servers with outside night air when air conditioning was lost. It worked.

However, rapid fault detection, isolation, and repair are equally important even if the component is redundant and if no outage is experienced by users as a result of a component failure. As long as the failed component is not operational, you are running with a single point of failure – the currently functioning component. The failed component must be returned to service as quickly as possible to minimize the probability that you will lose your system due to a dual component failure.

This requires rapid fault detection as well as rapid repair. Proactive monitoring of all system components is a must for high availability.

Increased Redundancy Levels

For highly critical applications, take advantage of increased levels of redundancy. If the technology supports it, consider going beyond dual redundancy to three, four, or even higher levels. Web farms are a good example of this. Page queries can be routed to any one of many web servers. If one fails, it is simply removed from the pool.

At the furthest extreme are multiple-site data centers. They can protect against the loss of one or even more data centers.

Solve Problems Once and for All

Record data about failures so they can be solved and not recur. Log errors and events for later analysis. Put a Console Management system in place to log console output and actions taken at the consoles. Take crash dumps whenever possible when an application or operating system fails. Synchronize the clocks of all systems with NTP (Network Time Protocol) so that logs and events are in synchronism. One of the early indications of a problem is the service call from a frustrated user. Log these, and insist that problems get fixed.[2]

When a failure of a vendor product (hardware, software, or service) occurs, log a service call under your support contract, promptly provide all the data requested, and insist that a fix be provided. Track the resolution process to completion.

Remember that performance problems are failures, too. Record these failures, and give them as much attention as you give to component failures.

Avoid Problems

There are preemptive steps that you can take to avoid problems in the future. For instance, make sure that all cables are properly labeled and that the labels are updated during system reconfigurations to protect against cabling errors.

An extremely important task is maintaining proper documentation of current operator procedures. This documentation should be tested thoroughly before making it generally available, just like any application. Google recently suffered a multihour failure of its popular Google Apps Engine when it had to fail over to a backup data center with incomplete documentation.[3]

---

[2] An extreme example of ignoring a problem caused an Amtrak train crash. See Software Bug Causes Train Wreck, *Availability Digest*; October 2006.
http://www.availabilitydigest.com/public_articles/0101/software_bug_causes_train_wreck.pdf.
[3] Poor Documentation Snags Google, *Availability Digest*; April 2010.
http://www.availabilitydigest.com/public_articles/0504/google_power_out.pdf.

Consider Component Maturity

The reliability of products tends to get better over time as improvements are made. This is true of both hardware and software products through engineering change orders (ECOs) and software patches.

However, don't be the first to use a new product or technology – there may not have been enough accumulated field experience to find and correct latent problems. By the same token, don't throw out a product that is working well just because it is "old" or is no longer fashionable.

On the other hand, don't be the last to continue using an old product or technology past the end of its market life. Support for the product will eventually go away, and support quality degrades over time as the memory of support personnel fades.

Implement a Test Environment

Test any change before deploying it. To do this, you need a test environment that is independent of your production environment. The test environment should duplicate the scale of your production environment as closely as possible, including the same hardware, the same software, the same firmware, and the same applications.

If you have difficulty getting funding to set up a test environment, you may be able to leverage the test system for development, quality assurance, and as a backup system for disaster recovery to help justify its cost. But be careful, as this dual-purpose use of equipment in itself can introduce additional risks into the environment.

Any change to your production environment should be tested in your test environment first. This includes even those updates to applications not under your control (operating system patches and virus updates, for example). McAfee recently released an anti-virus update that took down computers all over the world.[4] Those that did not test this update before deploying it paid the consequences.

Software and Firmware Revisions

Stay on supported versions of software and firmware. Troubleshooting support is available, and problems will get fixed. If you are on an unsupported version, you are on your own.

Many of the things that we have said before apply to software and firmware revisions. Don't be the first to use one – let someone else find the bugs. Don't be the last to upgrade to a version - support personnel tend to forget details of old releases. And be sure to exercise any revision in your test environment before deploying it.

Patches and ECO Kits

The caveats are much the same as for other upgrades. Monitor patches and ECO kit releases to determine what potential problems they purport to fix. Consider the likelihood that you will encounter those problems before installing a fix for a problem that you don't ever expect to encounter.

Let a patch age awhile before installing it. This can avoid your having to remove it if it should subsequently be recalled. On the other hand, don't wait too long to install a fix as you may encounter the problem meant to be fixed by the patch.

---

[4] Anti-Virus – A Single Point of Failure?, *Availability Digest*; May 2010.
http://www.availabilitydigest.com/public_articles/0505/mcafee.pdf.

As always, be sure to exercise the patch in your test system before deploying it into production.

Managing Change

Change causes outages.[5] Change may be one of the most predominate causes of system outages. We have addressed many examples of changes in our previous sections.

Changes can be modifications to the application or to the system configuration. Changes to the external environment, such as other applications, networks, or partner systems, may or not require changes to an application but may cause the application to suddenly perform improperly. Any change introduces risk. However, change is unavoidable and must be managed.

There is a balance between keeping up-to-date and minimizing unnecessary changes. Remember that new features often introduce new bugs. Fixes to problems can cause new problems. Always test new changes exhaustively in your test system before deploying them.

Try to introduce only one change at a time. This makes it easier to determine the cause of new problems (even after the change has been tested and deployed) and to back out the change if necessary.

The reason and procedure for each change should be thoroughly documented and reviewed by your technical staff. Above all, the change procedure should include a backout plan to remove the change should it prove faulty. If you do not have a viable backout plan, you may well be down for an extended period of time if the change causes your application to crash.

There may be some cases in which a change simply cannot be backed out and the system returned to an earlier state. In this case, this additional risk should be thoroughly documented and plans and personnel should be in place to fix any problems that may occur.

Should a new problem be detected, it is common to scrutinize recent changes to see if any have contributed to the problem. Therefore, change tracking is a valuable part of change management so that all recent changes can be reviewed. Remember, however, that factors outside of recent changes can cause problems as well.

Unless you are running an active/active system, a cluster, or a virtualized environment in which changes can be rolled through node-by-node without denying services to users, you will have to take downtime in order to make changes. Schedule periodic maintenance windows during which the application can be taken down for upgrades. Your choice is whether you want downtime to be mostly scheduled or unscheduled.

Computer Application of Biodiversity

Using a mix of different technologies can improve reliability and survivability. For instance, backing up a transatlantic cable communication channel with a satellite channel virtually eliminates the probably of a network failure due to any cause so long as the channels are accessed by your system via independent paths.

Minimize Complexity

In your design, try to minimize the number of things that could fail or cause an interruption. For instance, use the simplest hardware necessary for the task. Use the hardware, operating system,

---

[5] Niehaus' Law, Chapter 5, <u>The Facts of Life</u>, *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, AuthorHouse; 2004.

and management facilities that your staff is most familiar with, if appropriate. Minimize the number of tiers for a service.

Reduce the Impact of a Failure

Don't put all your eggs in one basket. Preserve the independence of different resources. For instance, don't use a simple service provided by another server if that service can be easily replicated on the same server that is running your application. Otherwise, the application can be taken down by the failure of either server.

Divide services into portions so that the failure of one portion does not affect all portions. This is a compromise between the span of the impact and the complexity of the services. For instance, a product-ordering application should not depend upon a customer relationship management (CRM) application if the two can be cleanly separated. Otherwise, the failure of the CRM system will take down order entry.

Be cautious about sharing resources among unrelated applications despite the lure of cost savings. For instance, if you want to significantly reduce your server count, you may determine that you can run all of your servers as virtual machines in a virtualized environment. If you go this route, be sure to configure at least two servers with the facilities to move virtual machines from one physical server to the other. Otherwise, a physical server failure will take down all of your applications instead of just one.

The same philosophy applies to virtualized storage. If all applications are using the same SAN for their storage requirements, a second SAN kept in synchronization with the primary SAN via mirroring should be available in case the primary SAN fails.

Do not connect redundant components. They should be totally independent. Otherwise, a failure in one component could take down the other. A failed component must be able to be isolated and removed for repair. It is also helpful to be able to isolate any suspected component of a set of redundant components to identify whether or not it is contributing to a problem.

Summary

Many of the availability tips suggested by Keith may be obvious to you, but I am sure that there are nuggets of good ideas buried in this discussion. Improving availability is not just a matter of hardware and software choice and configuration. It is a process that must be planned and followed throughout the deployment of your application.