the Availability Digest

Availability Best Practices

January 2007

Every year, ITUG, the Global HP User Community (formerly the International Tandem Users Group), recognizes with the ITUG NonStop Availability Award the company who has demonstrated the best availability practices during the prior year. Wendy Bartlett, an HP Distinguished Technologist, has been documenting the practices of those companies who have submitted entries for this award and has given many presentations on these practices. This article summarizes her findings[1] with some additional suggestions of our own.

Availability best practices cover the entire gamut of a data processing operation. They start with the development and use of robust software and proceed with good operator training and documented procedures. Planned downtime is minimized or eliminated, and repair strategies that lead to minimal repair time are adopted to minimize unplanned downtime. We examine availability best practices in some detail below.

Fault-Tolerant Hardware

The system hardware should be configured to be fault-tolerant. It should be able to survive at least any single failure. Failure points such as processors, disks, power supplies, fans and communication controllers should be monitored; and the system should allow component repair or replacement without the system having to be taken down.

Redundant disk subsystems should be used for all critical data. These could be mirrored disk pairs or RAID arrays. Network attached storage (NAS) or storage area networks (SAN) often can offer more reliability and effective administration than can be afforded in a system using its own direct-attached storage.

System and application resources such as swap space and file extents should be configured for growth and thereafter monitored so that action can be taken if a resource in nearing exhaustion.

Robust Software

Applications and data should be isolated so that a fault in one application will not corrupt other applications or their data sets. No application should be able to reach in and modify another application or its data set.

[1] Wendy Bartlett, Availability Best Practices, paper MA-03-HP; ITUG Europe 2006.

This is commonly accomplished by having applications interact through a messaging system. In this way, an application can be defensive and can determine whether or not to honor a request from another application. It has total control over its private data set. Object-oriented languages such as Java are designed to do just this.

Applications should be designed to have as high a degree of fault tolerance as possible. They should be able to determine that a problem has occurred and to report that problem. If possible, an application process that has failed should be able to recover from a problem as quickly as possible.

Fast recovery from a problem is best accomplished by having a backup process immediately available to replace a failed process. This can be done in several ways, such as

- having a pool of like processes so that if one fails, it can be removed from the pool, with further transactions processed by the surviving members of the pool.
- using an application monitor that can detect a failed process and then restart it automatically.
- providing a hot backup process whose state is kept in synchronism with its active companion and which is ready to take over processing should the active process fail.

Applications should be designed to handle any unanticipated event without crashing. They should also be designed to minimize or eliminate any disruption due to making normal configuration changes, such as adding users, adding terminals, or modifying global parameters.

Redundant Networks

The system must be able to survive network faults. This can be done by

- using redundant private networks (WANs or LANs).
- using self-repairing networks (such as the public telephone system, the public Internet, or a properly configured private Intranet).
- providing a backup communication facility such as dialed lines.

If redundant long-haul networks are used, they should be obtained from separate carriers that do not use a common carrier in their backbone, as this could cause a single point of failure.

All connections to the network should be redundant. Dual communication controllers should be used.

If a disaster recovery system is used, the communication facilities serving it should be independent of those serving the primary site.

The Environment

Both power and cooling should be redundant. Two independent power sources should be used to provide power for the computer systems. Each power source should come from a different provider if possible, or at least they should be routed through different conduits.

Backup power in the form of an uninterruptible power supply (UPS) should be available. Sufficient short-term battery power should be provided to maintain system operation until the UPS power is stable.

The computer room cooling facilities should be redundant and should be powered by separate power sources. Temperature and humidity monitoring should be provided.

Problem Monitoring and Management

More important than environmental monitoring is system monitoring. There are an abundance of monitoring tools in the marketplace for most systems. Not only should system resources be monitored, but applications should be monitored as well.

Scripts should be created to perform tests to detect problems that the system monitoring tools may not detect. For instance, routers should be pinged, the spooler status should be checked, status commands should be issued on critical processes, and communication lines should be monitored.

A single monitoring environment such as HP's OpenView or Unicenter from Computer Associates should be provided for all systems. Responses to problems from the monitoring system should be automated to avoid operator error. Problems can range in complexity from adding extents to a file that is filling up to switching over to a backup system.

The appropriate staff should be notified by email or paging of any problem, even if it is automatically corrected (automatic corrections don't always work).

A problem management system should be used to track problems, to ensure that they are corrected, to record the correction, and to maintain a history to detect failure trends.

People

Operations and application support teams should be available on a 24x7 basis. Critical staff who are not currently on site should be contactable by phone or pager in the event of a failure. Within this group of people should be a 24x7 crisis team that can be rapidly assembled to handle any critical failure.

There should be a full-time person assigned to monitor the performance of the system. That person should be responsible for tuning performance parameters, for balancing the load on the system, and for predicting future processing requirements so that there will be no overload failures.

There should be a daily hand-off between shifts to review open problem tickets. Weekly meetings should be scheduled to review all of the system problems which have been encountered since the last meeting.

Most importantly, a root-cause analysis of each fault should be performed. The root cause of a failure is that event which, if it had been corrected, would have precluded the failure. Root-cause analyses are a powerful tool to tune up operational procedures as well as the automated procedures for failure recovery.

Change Management

Changes are the life blood of a system. Without changes, the system will become functionally and technologically obsolete. However, a preponderance of system failures are caused by changes.[2]

Proper change management involves change tracking, documentation, review, testing and monitoring.

Change tracking can best be done by using an existing change management system, such as IBM's Tivoli or Scopus. Ensure that all changes are entered into the tracking system. This includes not only hardware and software changes but also environmental, network, facilities, and procedure changes.

Change documentation evaluates a change and describes the procedure for implementing it. It should include

- a risk assessment describing the benefits of the change as well as the potential problems which it may cause.
- a production impact assessment.
- a detailed plan for the installation.
- procedures for monitoring the installation.
- a detailed change test plan.
- the change success criteria.
- a change back-out plan in case the change causes unacceptable problems.
- contact information in case help is needed.

There should be daily change review meetings to keep all technical staff updated on what changes are being made. There should be a sign-off process for the various stages of the change. The sign-off process can be aided by change-checking tools, especially for configuration changes. Regular change reports should be generated for review by the technical groups and by management.

A detailed test plan must be generated to ensure that the change is proper and has no undesirable side effects before putting it into service. This test plan should itself be tested early in the process by having a non-expert run through the documented test procedures. The change should first be tested on a development system, then on a QA system prior to moving it to the production system and the backup system, if any. Performance tests should be run to compare the system operation before and after the change. Finally, end users should be involved in the final testing of the change.

The effect of changes should then be monitored until the success criteria have been met.

Software Upgrades

A major class of changes is operating system and application upgrades. Operating system upgrade information should be tracked to understand what corrections, enhancements, and new features are included in each new version. Using this information, the proper time to install an upgrade can be intelligently determined.

---

[2] See Rule 23 in *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, by Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holensten.

Try not to upgrade to a new operating system version until it has been in service for a while at other sites. This will minimize the chance of running into bugs in the new version.

A multiple-system architecture should be used to avoid planned downtime. The new operating system or application version should first be installed on a non-production system and should be thoroughly tested before putting it into production. Switchover from the old production system to the new version must be carefully planned to minimize any application downtime. The best case is the use of an active/active system which will allow nodes to be taken out of service, upgraded, and put back into service with little if any impact on the users.

Minimize Planned Downtime

The requirement for planned downtime should be minimized. For instance, batch runs should be able to run concurrently with normal operations. This can be done by either designing batch processing to be compatible with operations, or by providing an online copy facility that can copy a consistent version of the database to an offline system for batch processing. The results of the batch processing run may then have to be moved back to the online system.

Other sources of planned outages include database backups, database reorganizations, and database replication for shared data purposes.

To support major upgrades to hardware, software, and the database, switchover times between the active and backup systems must be minimized. Switchover times are preferably measured in minutes, not hours or days.[3]

Repair Strategies

Just because there is a spare component to take over the functions of a failed component does not mean that repair time can be casual. The longer the system is running with a single point of failure, the longer it is exposed to a catastrophic system failure should the spare component fail.

System availability is directly proportional to the repair time of any failed component. If average repair time can be improved by a factor of ten (say from one day to one hour), an additional nine is added to the system availability.

Repair time can be minimized by having onsite spares, onsite service personnel, rapid response service contracts, and many other techniques.

Repair time can be more casual if triple redundancy (or greater) is provided. Multinode active/active systems are just such an example.[4] This can help greatly in lights-out operations.

Disaster Recovery Site

Any data processing site is subject to a variety of disasters that could take it out of service. Among these causes are fire, flood, earthquake, social unrest (riots), malicious acts, and radioactive or disease quarantines.

[3] See Tackling Switchover Times, the *Availability Digest*; October, 2006.

[4] Another example is the HP NonStop server's triple modular redundancy (TMR) configuration.

Therefore, if continuous availability is to be achieved, there must be a backup site separated from the primary site by an appropriate distance. This site could contain another corporate-owned system, or it could be a system provided as a backup service to several disparate customers. In the latter case, it should be ensured that access to the backup system is reasonably available as these systems are often provided on a first-come, first-served basis.

Recovery time of the backup system is a critical parameter. There are several types of backup configurations, including a cold standby, a hot standby with a database synchronized to the active system, and an active/active system. Recovery time for a cold standby could be hours or days. Recovery time for a hot standby could be minutes to hours. Recovery time to another node in an active/active system is, in effect, instantaneous.

Periodic testing of the switchover plan is paramount. Otherwise, should the backup be needed, it may be found that the recovery procedures don't work. An important exception to this rule is backup provided in an active/active system. In these systems, all processing nodes are actively processing transactions. Should a node fail, all transactions are routed to surviving nodes. Therefore, it is always known that the "backup" system is operational.

Availability Management

A comprehensive application monitoring and failure management system should be provided. Transaction volumes and response times should be measured to anticipate growing problems. Audit trails should be created to improve the recovery of data.

Every outage, no matter how minor, should be recorded. Information recorded should include the nature of the failure, the root cause of the failure, the duration of the failure, the manner in which the failure was corrected (hardware repair, software recovery, switchover to the backup system), the number of users affected, and any other information that might be useful in later failure analyses and procedural reviews.

Summary

We can attempt to minimize failures by proper operator training, considered selection of hardware and software components, redundant networks, and so on. But failures will occur.

The secret to high availability is not to eliminate failures but to recover from them quickly. Only if recovery is so rapid that the user does not view it as a denial of service is availability maintained.

There are many ways to recover from hardware, software, network, and environmental failures; and they all require redundancy. There must be a backup component to immediately take over the functions of the failed component in a time so short that it is not visible to the users.

The ultimate in fast failover are active/active systems. In this architecture, multiple nodes in an active/active network cooperate in a common application, each node providing all of the functions of the application. Should any one node fail, all further work can be routed to the surviving nodes. Recovery from any kind of fault can be achieved in seconds.

It is for this reason that active/active systems can achieve availability at the user level of six 9s or more, with failure intervals measured in centuries.