“Achieving Century Uptimes” An Informational Series on Enterprise Computing

As Seen in *The Connection*, A Connect 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 15: Zero-Downtime Migrations: Eliminating Planned Downtime

March/April 2009

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

If your system must be up 24x7, planned downtime is not an option. So how can you possibly upgrade your applications, migrate to new hardware, convert to new versions of operating systems or database managers, or port to a new database structure? All of these maintenance activities and many others like them require that the system[1] that is being upgraded be taken down, often for hours or more.

Even if your system has a window during which it can be shut down for maintenance (over the weekend or at night, for instance), can you be sure that the system will be tested and be ready to be put back into service when the window closes? And what are the penalties if you cannot come up on time?

In this article, we look at a solution to the problem of ensuring that you do not suffer system downtime as a result of planned maintenance activities.

First Requirement: Redundancy

Clearly, if planned downtime is to be eliminated, there must be a backup system available to be put into service while the primary system is undergoing maintenance. With this capability, the backup system is put into operation; and all users are switched to it. The primary system is then taken down, upgraded, and tested.

Once it is assured that the upgraded system is operational, it is returned to service; and all user activity is switched back to it. The backup system reverts to its function of providing user services only if the primary system fails. If desired, the backup system can now be upgraded as well.

An alternate strategy is to upgrade the backup system first and then switch it to the primary role while the former primary system is upgraded. The systems can maintain these roles until the next upgrade (or unplanned outage) requires a role switch.

Of course, performing maintenance in this fashion requires that the environment support the running of different versions of the changes on the primary and backup systems. For some period of time during the update sequence, the primary and backup systems will be on different versions, and this issue must be mitigated by the approach taken. We will discuss this topic more below.

[1] We use "system" here to include the all-important application services to the end users.

Second Requirement: Fault-Free Failover

Many critical systems today function as a redundant pair and meet the first requirement of redundancy. Typically, one system serves as the active system doing all of the work; while the other serves as a backup system ready to take over should the active system fail or be taken down for maintenance. The roles of these systems can be reversed through a process we call *failover*. Failover may be a lengthy process, often requiring hours.

But what if the failover doesn't work? We call this a *failover fault*. Following a failover fault, are both systems down? Are the users without service? Is the maintenance window violated? If the upgrade schedule was mandatory because it fit into a larger upgrade plan involving several other systems or regulatory issues, what are the consequences of this? Failover faults can at best be inconvenient and at worst catastrophic.

Do failover faults happen? Unfortunately, all too regularly.[2] What can cause a failover fault? Many things can, most of them unanticipated. Keep in mind that when running in an active/backup architecture, the backup system is not currently in service. Therefore, it is not really known that it is, or can be made to be, fully operational. Even if the processor, memory, and disk units appear to be operational, there could be faults in the network connections, failover scripts could be out-of-date, wrong versions of applications could be installed, and the list goes on. Consequently, maintenance failovers must be carefully planned and adequately staffed with the appropriate personnel to resolve any problems that may arise.

Furthermore, even after extensive testing, what happens if the new primary system should experience problems? One has to return to the backup system until the primary system is corrected and retested. This creates another pair of failovers – return to the backup system and then switch back to the primary system. The problem has just been seriously compounded.

The elimination of failover faults is extremely important if planned downtime is to be eliminated.

Third Requirement: Fast Failover

While the system is failing over, service to users may be unavailable for minutes or hours. This is not tolerable if planned downtime is to be eliminated. This problem can be minimized or eliminated if failover is fast.

In addition, problems caused by failover faults for maintenance purposes can be lessened, though not eliminated, if failover is fast. For instance, if failover can be accomplished in four seconds instead of four hours, the impact of a failover fault is significantly lessened and perhaps even eliminated. If the failover should fail, within seconds the primary can be back in operation. Only a very small maintenance window is required.

[2] Blackberry Gets Juiced, *Availability Digest*; May, 2007.
Triple Redundancy Failure on the Space Station, *Availability Digest*; November, 2007.
Sydney's M5 Tunnel Closed Again by Computer Glitch, *Availability Digest*; November, 2008.

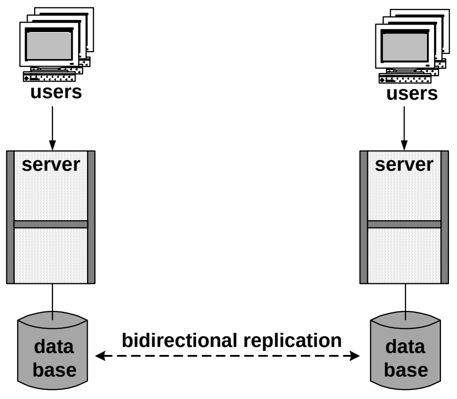
Conversely, if an upgraded primary should begin to exhibit problems when it is returned to service, user support can revert back to the backup system within a very short time.

A Solution – Active/Active Systems

As we have discussed above, the elimination of planned downtime requires three characteristics of the system:

1) It must be redundant so that there is a backup system that can take over the role of the primary system while the primary system is being upgraded.
2) The ability to fail over from one system to the other without a failover fault must be assured.
3) Failover must not only be reliable, it must also be fast.

Active/active system architectures fulfill all of these requirements. As shown in Figure 1, an active/active system[3] comprises two or more geographically-dispersed nodes that are actively participating in a common application. That is, each node is actively processing and sharing the application load with the other nodes. Should a node fail or need to be brought down for maintenance purposes, all that needs to be done is to switch transactions (or users) from the failed or downed node to the surviving nodes, a switch that can be done in seconds.



An Active/Active System
Figure 1

In order for a node to participate in an application, it must have access to an up-to-date copy of the application database. In an active/active system, each node has its own local copy of the database. The database copies in the application network are kept synchronized via data replication. That is, when one node makes a change to its copy of the application database, that change is immediately replicated to all of the other database copies so that they are all in the same state.

With an active/active system, in the event of a node failure, there is, in effect, no failover. No idle node need be brought into service. There is only the rerouting of transactions or the switching of users that were connected to the failed node.

Thus, with respect to our requirements for eliminating planned downtime:

1) Redundancy: There is always one or more processing nodes that can take over the load of a node to be taken out of service.

[3] What is Active/Active?, *Availability Digest*; October, 2006.
Breaking the Availability Barrier: Survivable Systems for Enterprise Computing, AuthorHouse; 2004.
Breaking the Availability Barrier Volume 2: Achieving Century Uptimes with Active/Active Systems, Authorhouse; 2007.
Breaking the Availability Barrier Volume 3: Active/Active Systems in Practice; Authorhouse, 2007.

2) Fault-Free Failover: Since all nodes are actively processing transactions, it is known that all are working properly. Therefore, users or transactions can be rerouted from the node to be taken down to one or more of the other nodes with little if any risk.

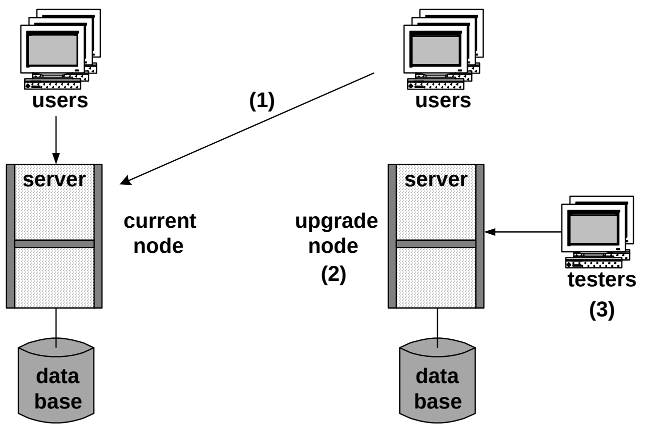
3) Fast Failover: Since failover is really only the rerouting of transactions or the switching of users, it can be accomplished in seconds or even in subseconds. Though care should be taken in this process, the process can often be automated and managed by intelligent networking software (for example, some external network routers periodically poll the systems to determine which are currently available for transaction routing).

The upgrade process comprises seven steps. We call this process *Zero-Downtime Migration*, or *ZDM*. The first three steps are shown in Figure 2.

- Step 1: Take down the node to be upgraded. Move the users from the node to be upgraded to one or more of the other nodes. If dynamic transaction routing is used, reroute all transactions to the other nodes. Stop data replication from and to the node being removed from service, and take down the node. Changes made by other nodes in the application network will be queued for later replication when the node is returned to service.

- Step 2: Upgrade the downed node. Perform whatever maintenance is to be done on the downed node. This might include an application upgrade, the installation of a new operating-system version or a new database management system version, or even the migration to new hardware.

- Step 3: Test the upgraded node. The upgraded node can now be thoroughly tested before returning it to service. If the test procedure should last for an extended period of time, for example days or even weeks, there is no problem because the surviving nodes in the active/active application network continue to provide full user services.

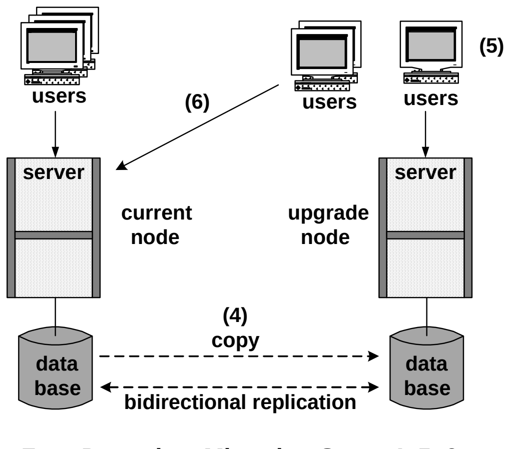


Zero-Downtime Migration Steps 1, 2, 3
Figure 2

Steps four through six are shown in Figure 3.

- Step 4: Synchronize the new database. Once the upgraded node has passed its test, copy the current database to the database on the upgraded node (or drain the changes that have accumulated from an active database to the database on the upgraded node). Optionally, run a verification and validation utility to ensure that the database on the node being returned to service is correct. Then start bidirectional replication between the node being returned to service and the other nodes in the application network.

- Step 5: Put the upgraded node into trial use. Move a few users to the upgraded node for a live trial. If this is successful, move more users to the upgraded node. In this way, the upgraded node can be returned to service gradually and in a controlled manner.



Zero-Downtime Migration Steps 4, 5, 6
Figure 3

- Step 6: Revert if a problem occurs. If a problem occurs during the gradual migration of users to the upgraded node, be prepared to revert back to Step 1 to fix the problem. Move the users off of the newly upgraded node, stop replication, take down the node, and fix the problem. Follow Steps 3 through 5 to reattempt returning the upgraded node to service.

- Step 7: Put into full service. When all of the upgraded node's users have been returned to it, and when operation is satisfactory, the upgrade of this node is complete. It has been returned to full service, as shown in Figure 1.

Another node can now be selected to receive the upgrade. In this way, the upgrade can be rolled node-by-node through the rest of the application network.

Sizzling-Hot Standby

The zero-downtime migration described above applies to systems running in an active/active configuration. If your system is not running active/active, it may seem a simple step to provide bidirectional replication between your current primary and backup systems and to put the backup system to work as a cooperating member of an active/active pair.

However, things are not so simple. There are many application structures that may have to be modified in order that the application can be active/active ready.[4] For instance, data collisions may occur if two nodes try to update the same data item at the same time. Unique numbers such as invoice numbers may no longer be unique across the nodes. Memory-resident context may have to be made visible to other nodes.

If it is deemed too expensive to move to active/active, a partial step is to move to a "sizzling-hot standby" system. This system configuration is the same as that of an active/active system, except only one node is processing *update* transactions. The applications on the other node are up and running, and may optionally be processing read-only requests. Its database is kept synchronized with the active node via data replication. To insure full end-to-end processing is operational, test or verification transactions can periodically be sent to the backup node's applications.

[4] B. D. Holenstein, P. J. Holenstein, W H. Highleyman, Appendix 4 – A Consultant's Critique, *Breaking the Availability Barrier III: Active/Active Systems in Practice*, AuthorHouse; 2007.

In this way, applications do not have to be modified to run in a multinode environment. However, all of the failover properties of active/active systems so necessary to zero-downtime migrations are maintained. It is known that the standby node is operational – it can be easily tested by continuously sending it test transactions. Should the primary node fail, users can be quickly reconnected to the standby system (or transactions rerouted to it) within seconds. If data replication has been configured to be bidirectional, the downed node will be resynchronized with the operational node when the downed node is returned to service.

The active/backup configurations so common in today's IT environments can often be easily extended to a sizzling-hot standby configuration by simply implementing bidirectional data replication to keep the active and backup databases in synchronization. In this way, with no application changes and perhaps with little or no additional hardware expense, the benefits of faster recovery in the event of failover and zero-downtime migration can be achieved.

Summary

System upgrades in conventional IT environments are expensive and can lead to extensive system downtime. For stand-alone systems, the system must be taken down, upgraded, and returned to service during a maintenance window. Upgrading can take a long time, and an upgrade gone bad can prevent a system from being returned to service in the required time.

If a backup system is available, it is only necessary to switch operations from the primary system to the backup system within the maintenance window. However, failing over to a backup system can still take a long time during which both systems are down; and the failover is subject to failover faults, following which the backup may not be functional.

With active/active systems and their close cousins, sizzling-hot standby systems, failover is virtually instantaneous and fault-free. With these systems, planned downtime (as well as unplanned downtime) can be eliminated, leading to exceptionally high availabilities measured as six 9s and beyond.