the Availability Digest

www.availabilitydigest.com

Ring-of-Fire Bank Beats Earthquakes with Active/Active

July 2012

Not many banks avoided exposure to the recent subprime crisis and speculative real-estate mortgage meltdown. One bank that did, due to its rational credit policies, remained the number one lender in its area while other financial institutions severely restricted credit to their customers. The bank continues to steer clear of many of the credit challenges facing the financial industry.

However, it was not as well prepared for Mother Nature. The bank is located on the Pacific Rim *Ring of Fire* and lost its data center for several hours during an earthquake. When the failover to the backup system did not go as planned, the bank said, “Never again!” It started on the path to move its data center operations to a continuously available active/active environment.

The Bank’s Background

History

The bank was established over 150 years ago and has grown to the point where it now has over $15 billion USD in assets. It provides all of the traditional services to its customers, including checking and savings accounts, credit cards and debit cards, mortgage financing, consumer and commercial loans, and online banking. Its conservative credit philosophy is derived from its parent bank, one of the top five banks in the world, according to Forbes magazine. Standard & Poor’s ranks the parent bank sixth in the world in terms of credit strength.

Data Centers

The bank operates two HP NonStop data centers that are located about 50 miles apart. Prior to the earthquake, one data center served as the production site and the other as a passive backup site. The backup database was kept synchronized with the production database via a replication product that supported only active/passive architectures in which the target environment had to be an exact mirror of the source. Additionally, the replication product limited the target application to read-only operations. Unfortunately, the third-party banking application used by the bank could not run in read-only mode. Hence, testing of the target system required a complete outage of the source environment and the consequent loss of application services to users during the test period.

As many companies have discovered, testing failover to a backup data center is a risky, inconvenient, and costly proposition. In order to properly test its backup system, the bank had to bring down its production system, denying application services to its customers. Applications had to be loaded and started on the backup system, and the user network had to be switched. Furthermore, the replication product used by the bank did not maintain a transactionally-consistent copy of a production database on the backup system during replication. Therefore, the bank’s backup database had to be brought into consistency before the backup systems could be tested and placed into use.

The failover process is complex. Knowledgeable staff must be assembled, typically in the middle of the night or over a weekend, to handle any problems that may arise. Failover, if successful, can take more than an hour, during which time the applications are down. Even worse, it is entirely possible that the failover itself will fail and that the backup system may be unable to be brought into service. This *failover fault* delays the overall failover test even further.

Therefore, the bank followed what is unfortunately a common practice. Although it periodically performed failover testing, its tests often ended up incomplete. The process to switch to the backup site and then to switch back to the production site typically ran into many small problems that collectively added up to a painful and slow sequence. Many failover tests simply did not complete or did not complete successfully in the allowed testing timeframe.

In practice, many companies that rely on active/passive architectures confront the same issues. In the end, they rely on faith and hope that their backup systems will come up in a reasonable amount of time should the primary systems fail. Even if a company's data center is well-designed for redundancy and its staff well-trained in the proper failover procedures, and the staff practices these procedures successfully within a failover window that is acceptable to the business, other external factors such as the management decision time to fail over often significantly extend the failover process. The result can be a failover that even if successful severely violates the disaster-recovery service level agreement (SLA).

Mother Nature Strikes

One fateful day, the bank's backup preparedness faced a crucial test. An earthquake struck and caused the production systems and their networks to fail. The bank initiated its disaster-recovery plan. As might have been predicted, the bank suffered a *failover fault*. It could not bring its backup systems into operation. The most critical outages were those of its online banking services and its ATM/POS network. With no ATMs or POS devices working, much of the area's retail activity came to a halt at a terribly critical time since people needed to buy supplies and to take other actions to survive the damage caused by the earthquake.

The problem was further aggravated by the facts that the production system had lost power and that the IT staff had been evacuated from the primary data center due to concerns about structural damage. Hours passed before the bank's staff could reenter the production data center and bring up the production system in order to restore ATM and POS services to the community.

The Search for Continuous Availability

This disastrous experience led the bank to realize that its approach to disaster recovery was unacceptable. Failover was slow and unreliable and was difficult to test, and failover testing always caused an application outage. Therefore, the bank initiated an in-depth study of where it currently was and where it wanted to be.

Where it wanted to be was obvious. The bank needed to eliminate unplanned and planned downtime for its critical applications. It did not want its online banking services, its ATM and POS networks, and its other critical services ever to be down again. It wanted them to be *continuously available*.

The bank concluded that it had to eliminate the problem of failing over to a system whose operational state might be questionable. It had to have a backup system that was known to be working and that could be tested frequently without impacting the application users.

The Limitations of the Bank's Architecture

The bank came to understand that its backup approach was constrained by the data replication technology that it had adopted and the network infrastructure it had deployed. Data replication is fundamental to disaster recovery and high availability, as it is the mechanism that provides an up-to-date copy of the production database at a remote site. Without a synchronized remote copy of the database, there can be no recovery from a lost production system since there is no data for the application to use.

Many replication products are available, and they offer different advantages. The problem the bank faced was that it had chosen a replication product and network infrastructure that did not support the functionality needed to ensure rapid and reliable failover. The limitations of the bank's architecture included:

- The replication product was very restrictive. It required the backup system to be configured exactly the same as the production system. Any changes made to one environment could cause replication outages if these changes were not also made to the other environment. Many of the bank's failover testing problems were caused by a change that had been made to the production system but not to the backup system.

- The target database provided by the replication product was transactionally inconsistent, and the replication product prevented applications from opening the database in update mode. Therefore, applications on the backup system could not be up and running with the database mounted for fast failover.

- The replication product provided only unidirectional replication. The bank could never move to a configuration in the future in which both systems were actively processing transactions, informing each other as to the database changes that they were making. Active/active architectures or their lesser brethren *sizzling-hot stand-by* configurations, in which all processing nodes are available to process transactions, are required to achieve continuous availability.

The fact that applications could not be running on the backup system limited the bank to an active/passive configuration, in which the backup system was idle except for being a replication target. Therefore, failover testing was a lengthy process. The production system had to be stopped, the backup database had to be brought into a consistent state, and reverse replication had to be configured. (This optional but useful step allows the newly promoted node to send all of its changes back to the original production node to simplify resynchronizing it.) The backup applications then had to be started, the network switched, and the system tested before it could be put into service. This process typically required several hours, usually in the middle of the night, during which time all application services were unavailable.

Compounding this challenge was the fact that configuration errors could not be detected until the backup applications were up and running. If found, these errors had to be corrected; otherwise, the test had to be terminated. Between the complexity of the failover process and the problem of configuration errors, failover not only was difficult and time-consuming, but it also was unreliable.

These problems had an additional effect on recovery from an unplanned outage such as the earthquake. Because failover was so lengthy and unreliable, the *management decision* to fail over rather than to try to recover the failed production system was a difficult call and added additional time to the failover sequence.

The Road to Availability Improvement

The bank decided that it had to move from *disaster recovery* to *disaster tolerance*. Disaster recovery means that the IT systems *recover* from a disastrous event and continue operating, even if that requires

hours or days of downtime. Disaster tolerance means that recovery is so fast that no one notices the outage or at least is not inconvenienced by it. In essence, the application is available during the failure.

The bank was not anxious to immediately and totally reconfigure its systems into a complex, continuously available architecture. Rather, it wanted to take small, controlled steps towards improved availability with the eventual goal of continuous availability.

Implementing a disaster-tolerant architecture can be a daunting task, as there are many considerations beyond those needed for a disaster-recovery active/passive architecture. This task can be accomplished via a controlled process that can achieve incremental improvements. At the same time, experience and trust in the process and products used can build as the application's overall availability profile is improved. The key, then, is to understand the end-state goals and to define an incremental process to achieve those goals.

The Availability Improvement Process

The bank's availability improvement process proceeded as follows:

Step 1: Define Requirements

The bank began by reviewing its options for a new replication product. It decided that the only way it was going to eliminate the failover problem was to have a backup system that was ready to take over instantly. Therefore, the replication engine that it needed had to possess the following characteristics:

- Backup applications had to be up and running with the databases mounted, ready to take over processing on an instant's notice, meaning that the replication engine had to allow applications to open the database in read/write mode even as replication was in progress.

- The target database always had to be in a consistent state during replication so that it could be used immediately following a failover.

- The systems had to be decoupled so that they did not have to be configured identically, thus eliminating failover faults due to configuration errors.

- The delay in replicating database changes (the replication latency) had to be small to minimize data loss following a source-system failure.

- The replication product had to support bidirectional replication so that reverse replication could be configured for the backup system. In this way, after a failover, the backup system would queue changes that it made to the database while the production system was down. The change queue would be used to resynchronize the failed system upon its return to service. In addition, upon system restoration, a 'new' backup system would immediately be available without reconfiguration.

Step 2: Choose a Data Replication Engine

With its specifications in hand, the bank next evaluated the various replication alternatives that were available for NonStop systems. It chose the Shadowbase replication engine from Gravic, Inc. (www.gravic.com/shadowbase) as the one that best satisfied its requirements.

Shadowbase software supports bidirectional replication. Applications can be actively running on both systems and can be simultaneously updating the application database. Replication is process-to-process with no intermediate disk storage, leading to small replication-latency times. Shadowbase solutions can

replicate between heterogeneous systems, so maintaining identical system configurations is not a requirement.

Step 3: Switch Replication Engines

Before taking any further steps, the bank wanted to make sure that it was comfortable with its new data replication engine. To start the process, it replaced its original replication engine with the new engine doing the same job. The replication engine was installed in unidirectional mode to keep the backup database synchronized with the production database in an active/passive configuration.

As of this writing, the bank is successfully running in active/passive mode with the new replication engine. It is gaining experience with the replication engine, is confident in the engine's handling of fault and error conditions, and has learned to tune it properly.

Step 4: Test Bidirectional Replication

The bank is now preparing for its next step, and that is extending to bidirectional replication. Initially, there will be no applications running on the backup system. However, failover testing will now be simpler: start the target applications, switch the network, and test the backup system.

Equally important, since replication will be bidirectional, any changes made by the backup system during the test will optionally be replicated back to the production system. This change will make the fallback process much easier since the paused production system will now have an up-to-date database. There will be no need for a special step to bring the production system into synchronization before returning it to service.

In addition, the applications on the production system will not have to be shut down during the test since these applications will continue to have the database mounted. When the backup test is complete, the network will be switched back to the production system so that it will receive further transactions to process.

Step 5: Configure the Fast Failover System

Once it has become comfortable with bidirectional replication, the bank will be in a position to reconfigure its existing systems to provide fast failover. With bidirectional replication functional on both systems, it will be able to put both systems into operation with all applications up and running. Both systems will be fully functional, with transactions sent to either system for proper processing. However, the bank will direct all transaction activity to only one system; the other system will serve as a sizzling-hot standby.

Since the application is already running on the standby node, the standby system can be frequently tested by simply sending it verification (or test) transactions. This testing is risk-free and will be totally transparent to the users. It can be performed at any time of the day or night with no source-application outage needed. Therefore, in the event that the standby system should be needed, it will be known to be operating properly and will take over with no failover faults. In fact, should the active system fail, all that will be necessary will be to reroute further transactions to the standby system, which will immediately continue transaction processing. This failover will require only network rerouting, which can be accomplished in seconds.

With only the acquisition of a proper replication engine and some system reconfiguration, the bank will have moved from multi-hour unreliable failover to multi-second reliable failover. It will have achieved its goal of continuous availability with no change in its hardware configuration.

Step 6: Move to Active/Active (a future option)

At this point, the bank will have achieved its availability goal by following a very structured and straightforward process; and it will be in a position to take this process one step further if it wishes. It can put both of its systems into active production, called an *active/active system*. Since both systems can process transactions, the transaction workload can be split between the two systems. Should one system fail, it will only be necessary to reroute all further transactions to the surviving system.

When using an active/active configuration, the capacity of both systems is utilized. Though each must be configured to handle the entire transaction load in case one system fails, performance is improved during normal operation since each system is less heavily loaded. Production load spikes are much more easily tolerated across two nodes rather than just one.

The bank realizes that going active/active is not as simple as moving to a sizzling-hot standby. Several problems must be faced when running an application in a distributed environment. For instance, the applications may need to be modified to correctly handle memory-resident data structures that are not replicated. Applications cannot open files for exclusive access since the replication engine cannot then also open them to apply replicated changes.

Another problem is data collisions. A data collision occurs if applications at two nodes modify the same data object within the replication latency of the replication engine. Neither will know of the other's update until that update is replicated, thus overwriting the original update. Shadowbase replication provides facilities for detecting and resolving data collisions.

Fortunately for this bank, its third party banking application software provider is already running Shadowbase active/active implementations at other sites. The eventual move to anactive/active architecture in order to attain continuous availability is therefore well within the bank's reach.

By following a controlled process, the bank can take its time to study active/active configurations and to decide whether the complexity of this move is worth it. It can migrate applications to active/active one at a time if it desires. This approach was implemented by many other banks, including such North American banks as Wells Fargo, U.S. Bank, Royal Bank of Canada, and Fifth Third Bank, as well as banks worldwide, including Lloyd's, Bank-Verlag, Swedbank, and Bank of Chile.

Summary

The success that this bank has achieved in moving in a controlled fashion towards continuous availability teaches an important lesson – do not give up if you think moving to higher availability architectures is too hard. Each architectural step, from magnetic-tape backup to virtual-tape backup to active/passive to sizzling-hot standby to active/active, moves you closer to continuous availability. The migration is a process that can be managed and controlled to ensure success on your schedule.

After all, if you already are running a backup site, you already have accepted the cost of redundancy, which is the first requirement for improved availability. Why, then, accept outages measured in hours with the possibility of catastrophic failover faults when you can have continuous availability for the cost of a data replication engine and some reconfiguration? This is what active/active architectures provide.