the Availability Digest

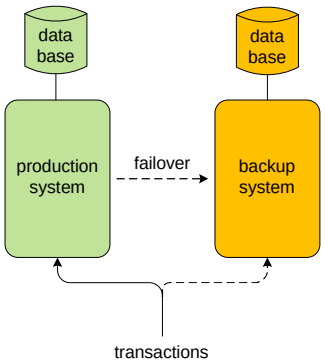
www.availabilitydigest.com
@availabilitydig

The Importance of Failover Testing

July 2017

As we all know, testing an application thoroughly before deployment is an absolute necessity. Bugs caught in the implementation phase of a project are easily corrected. Errors found during system testing can cost a lot more to correct. Faults found following the deployment of the application can not only cause significant hardships to the users of the application but can be much more expensive to fix. At this stage, the corrections to the application may also impact other applications with which it is cooperating.

Now that we have thoroughly tested the application in service, we would like to make it highly available. This is typically accomplished by providing a backup copy of the application. Should the production application fail, the plan is to move all processing to the backup application. A good plan, but will it work? The only way to be sure is to test a real-life failover. Will the backup take over? Will it do it quickly, or will it take minutes or even hours to assume processing? Or will the failover fail? We can only determine this by testing the failover capabilities of our system. But sadly, failover testing is hardly ever done, or is only done partially. There are many reasons for this that we will get to later.



There are several architectures in which a backup system can be deployed. One architecture is to use a cold standby. In this case, the backup system may be processing some other low-priority application. When it is called upon to take over production, applications must be loaded, the database synchronized, and the network switched to route transactions to it. This process may take hours, especially if the database must be loaded from tape.

A hot standby, on the other hand, already has the applications loaded, though it may be doing other processing. A current version of the database exists on the system and has been kept up-to-date via data replication. All that needs to be done it to reroute transactions from the failed production system to the backup system, which is ready to process them. Failover is typically accomplished in minutes.

In an active/active system, there are two or more systems actively processing transactions. The databases of the systems in the application network are kept synchronized via bidirectional replication. Whenever a change is made to one database copy, that change is replicated to all of the database copies in the application network. If a system fails, all that needs to be done is to reroute transactions to a surviving system. Therefore, failover is measured in seconds.

We do not need to test failover in an active/active system, since we know all systems are functioning properly. After all, they are all actively processing transactions.

However, if a cold standby or a hot standby is being used, how do we know that the failover to the standby system will actually work? And there is the rub. We don't know unless we test it. But testing failover in a functioning system is tricky business.

In order to test failover, we must take down the production system so that the backup system can take over the processing function. However, in today's 24x7 global operations, there is no downtime window in which we can do this. We must take down the production system during active processing and wait for the backup system to kick in – if it does. During this time, the users have no access to the application services upon which they depend. This may be a matter of a few minutes if failover is successfully made to a hot standby system, or it could be a matter of an hour or more if failover is made to a cold standby.

As a consequence, failover testing is a disruptive action. No wonder most companies do not perform full failover testing. They may do partial testing that they determine will test most of the failover procedures without impacting users, but the testing does not extend to actually taking the system down. Rather than disrupt their user base for failover testing, companies would rather depend upon faith and hope should there be a production system failure.

Consequently, the probability of a failover fault is significant. Based on random surveys at our presentations on high availability, we find that a company would be happy if 90% of its failovers worked. However, actual experience is far worse than this. The probability of a failover fault is sufficiently high that many companies require a management decision as to whether it is better to fail over or to try to bring the production system back online. By and large, the decision is made to reload the production system and redeploy it rather than trying to bring the backup system online.

There are many reasons for failover faults. One is that the backup system has failed and no one has noticed. Another reason is configuration drift. A typical MTBF for a production system is several months. That is, there may be many months before a failover is executed. This time could extend to years if the common practice is to try to recover the production system without failing over. Updates to the production system often are forgotten to be applied to the backup system. Therefore, the backup system is misconfigured and cannot take over processing of the applications.

A creative way to ensure that the backup system is operating is employed by at least one company with which we are familiar. The company fails over every three months and then runs production on the system to which it failed over. After another three months, it fails back and runs production on the other system for three months. In this way, it is confident that its backup system is configured properly and will perform when it is called upon following a production system failure. An added attraction of this procedure is that it gives the IT staff continued practice in failover procedures so that failover can be accomplished in minutes rather than in hours.

What a difference a little failover testing would make. Let's get creative and figure out how to do it in our respective applications with minimum user disruption. The benefits could be significant if a real disaster hits.