the Availability Digest

www.availabilitydigest.com

Active/Active on Commodity Servers

September 2013

In a recent post to our LinkedIn Continuous Availability Forum, I asked the question:

> "**Why is Active/Active stuck in the mainframe world?**
> "It seems that all of my case studies on active/active systems in the Availability Digest use mainframe systems – HP NonStop, OpenVMS Clusters, and IBM Parallel Sysplex. There are plenty of mission-critical systems using Linux and Windows running on large commodity servers that could benefit from active/active, and there are bidirectional replication products that support active/active on commodity servers (Shadowbase and GoldenGate are examples). Why has active/active technology not penetrated the commodity-server marketplace?"

This was one of the most active threads we have had on the Forum, and there were many good points made. All agreed that active/active has found a place in the mainframe world – NonStop servers, OpenVMS clusters, and IBM Parallel Sysplex systems. However, there has not been much active/active penetration in the commodity server marketplace.

One of the few exceptions that I have come across is the use of the Stratus ftServer. It is used by S1 for its financial-transaction switch.[1] Also, payment applications such as ACI's Postilion on Windows and Alaric's Authentic with Stratus Advanced Transaction Router on Linux support an active/active deployment. The latter has provided greater than six 9s of business availability for a customer doing over six billion transactions per year for the past 5 years. True, Stratus is not exactly a commodity server, but it is a fault-tolerant x86 server that runs unmodified Windows and Linux applications. It seems that organizations willing to split for the cost of Stratus fault tolerance are willing to absorb the higher cost of running active/active.

There were two main themes throughout the thread to explain the lack of use of active/active in commodity servers.. One was the effect on system performance imposed by active/active bidirectional data replication. The other was system architecture limitations that have to be observed in order for an application to be active/active ready.

It should be noted that there are active/active applications, but not active/active systems. A system that is supporting active/active applications may also be supporting other applications that are using passive backup or that are not backed up at all.

Architecting an application to be active/active-ready will certainly add complexity to a project implementation and will require additional testing before production can begin. However, if the cost of downtime is high (in some critical applications, the hourly cost of downtime can approach or exceed six figures), the additional cost and schedule required for an active/active implementation may be well worth it.

[1] Maximize Payments Availability with S1 Active/Active Software and Stratus ftServer Systems, *S1 White Paper*: 2011. http://www.availabilitydigest.com/misc/active-active_s1%20project.pdf

Active/Active Bidirectional Replication Performance

Replicator Performance

There was concern by some respondents that contemporary bidirectional replication products could not support high performance. This is, indeed, a valid concern. If the replication latency (the time from when an update is made to the source database to the time that it is made to the target database) is measured in several seconds or even minutes, then the databases will be significantly out of synchronization. A transaction sent to each may be processed differently because the database contents differ.

If the transaction capacity of the replication product cannot support the peak transaction loads, things get even worse. The backup database copy could get hours behind the active copy and take hours to catch up when the load lessens.

This means that one must ensure that the replication product used in the active/active system has sufficiently low replication latency and sufficiently high capacity to meet the application needs. For most applications running on commodity servers, there are replication products that meet these important requirements. They provide sub-second replication latencies and can support thousands of transactions per second. These products include Shadowbase from Gravic, Inc., (http://www.gravic.com/shadowbase/) and GoldenGate from Oracle GoldenGate (http://www.oracle.com/us/products/middleware/data-integration/goldengate/overview/index.html).

Another important parameter is the footprint imposed by the data replicator. If it consumes 50% of the CPU capacity, it is not going to be very effective. Today's replication products such as those mentioned above typically have very small footprints requiring only a few percent of CPU cycles.

System Loading

Another concern was the load imposed upon the target system by the replication of transactions from the source system. Every transaction must update both the source database on the system processing the transaction and must also update the target database on the system backing up the source system. Also, both systems must be configured to handle the entire transaction load should one system fail.

However, this is a fact of replication backup. Even in an active/passive system in which one system is handling all transactions but is keeping a backup database synchronized, the backup system is carrying the database update load; and both systems must be configured to handle the entire load as only one system is in active operation at a time.

Is there a performance difference between an active/active configuration and an active/passive configuration? Consider the following comparison. Let's start with an active/passive system. Let's say that the production system is running 80% loaded at peak, and that half of this is disk activity. That means that the backup system is 40% loaded due to replication disk activity (data replication has a very small CPU footprint - it is mostly disk). Transactions see an 80% loaded host.

If we moved to an active/active architecture, then each system is handling half of the transaction load (40%) and half of the replication load (20%). Each system is 60% loaded.

Which would you rather have - an 80% loaded production system with an hour failover to its backup or two 60% loaded production systems with instant failover? Of course, if one of the active/active servers fails, then the remaining one is handling all 80% of the transaction load (there is no replication load), which is the same as the production system in an active/passive configuration.

I think that assuming that the replication load is 50% of the transaction load is too high based on my knowledge of current installations, but it serves to illustrate the argument that active/active beats active/passive every time if active/active can be implemented.

Is the Application Active/Active Ready?

There are several architectural challenges to deploying an application in an active/active environment. For current applications, one must determine if there are any impediments that must be corrected before deployment. The modified application must be thoroughly tested to make sure that all potential problems have been corrected.

For new applications, the problem is simpler. One must be aware of the architectural limitations and ensure that the application design results in an active/active-ready product.

Structural deficiencies in the application architecture to be avoided include the following:

Data Collisions

Most active/active systems today use asynchronous replication. Replication is "under the covers" and is transparent to the application. This is typically done by replicating changes from the transaction log file on the source system and applying these changes to the target database. Therefore, there is a lag from when an update is made to the source system and when it is applied to the target system. This lag is known as replication latency.

Because of replication latency, it is possible for two transactions, one at each system, to update the same data item. Each system will replicate its change to the other system. Both systems will have a different value for the data item, and both are wrong. This is a data collision.

If the application can be designed to avoid data collisions, this is the optimum solution. However, if data collisions can occur, they must be detected and resolved. These capabilities are built into most bidirectional data replication engines.

Exclusive Opens

If the application opens certain tables for exclusive access, these tables cannot be synchronized via data replication. The replication engine will not be able to open the tables for writes. Great care must be taken if the application is to be modified to open these tables on a non-exclusive basis as there may be undesirable side effects. It must be understood why such tables are dedicated to one application. Are there functional reasons for this, or is it simply conservative coding?

Memory-Resident Data

Some data structures may be memory-resident and will not be replicated by a database replication engine. There are several solutions to this problem. One solution is to move memory-resident data to disk. However, performance considerations may preclude this. Another solution is to build logic into the application to replicate this data from memory to memory.

In some cases, the application might be able to be changed to accommodate memory-resident structures. For instance, if invoice numbers are assigned by incrementing an in-memory counter, multiple active/active nodes will assign duplicate invoice numbers. This problem can be corrected by prepending a node ID to the invoice number or by assigning blocks of invoice numbers to different nodes.

Hard-Coded Parameters

Though it represents poor coding, many applications have hard-coded parameters in their source codes. Often, this may preclude distributed operation. For instance, if a node ID or IP addresses are hard-coded, they cannot be changed across the nodes in an active/active application network. These parameters will have to be replaced with parameters that can be specified for each node.

Sequential Event Processing

In some applications, inputs must be processed in the same order as they were received. An example of this sort of application is a power-control system. Whether the tripping of a circuit breaker caused the transformer to be overloaded or the overloaded transformer caused the circuit breaker to trip is of paramount importance to the resolution of the power outage.

These sorts of events cannot be processed by independent nodes unless there is intelligent event routing or unless some synchronization mechanism can be implemented.

Networking

The active/active network must be able to reroute transactions from a failed node to one or more surviving nodes in the event of a node failure. Rerouting requires two capabilities – detecting a node failure and network rerouting.

With respect to node-failure detection, unlike a single system in which the node fails and simply must be restored, a node failure in an active/active system requires immediate action – namely, rerouting users to surviving nodes. Therefore, there must be an effective mechanism for one node to know that another node has failed. This is often done via heartbeats between the nodes. However, care must be taken to avoid a *tug-of-war* in which each node thinks that the other node is down; and both try to take over the other node's users.

Once a node failure is determined, there must be a means to transfer transactions from the failed node to surviving nodes. There are many ways to do this.[2] One is for the clients to be intelligent enough to detect the failure and to switch to backup IP addresses to which to send further transactions. Another technique is for the network to be intelligent enough to detect link failures and to reroute further traffic over an alternate route. Still another is to use virtual IP addresses and to have a surviving node advertise that it is now the owner of the virtual address of a failed node.

Connections and Sessions

If a client has to be rerouted, it is ideal that it does not lose its connection or session. One solution is to provide intelligence in the client so that it can reconnect automatically to a surviving node and log in to create a new session. Another solution is to have all clients open two connections and to establish two sessions, one on the primary node and one on a backup node. In this case, if a client finds that it has lost the link to its primary node, it can simply begin sending transactions to its backup node.

If connections or sessions are lost, a human user may have to log onto the system again. This impacts the fault-transparency benefit afforded by active/active applications.

Local Networking Context

Typically, communication context is stored in memory. In this case, it is not globally accessible even though it is needed by the distributed applications.

An example of such context is the description of a connection. If a message requiring a response is sent to a remote system asynchronously so that the response is returned on a separate connection, it may not be possible for the responding system to know to which node the response should be sent.

---

[2] Achieving Fast Failover in Active/Active Systems – Parts 1 and 2, *Availability Digest*; August, September 2009.
http://www.availabilitydigest.com/public_articles/0408/user_redirection.pdf
http://www.availabilitydigest.com/public_articles/0409/user_redirection_2.pdf

This is a particular problem if the remote system is that of another organization. The remote system cannot be modified to correct this situation, and some other solution must be found.

Batch Processing

Batch processes are expected to run only in one node. In a distributed environment, provision must be made to designate a node in which batch processing will be performed. In some cases, batches are scheduled to run during a batch window. In these cases, it is easy to assign batch jobs to a specific node. However, in other cases, batches are run when some application status is reached. The application should be modified to allow this to occur in only one node.

Sometimes, application decisions are made based on the status of batch jobs. For instance, the application may not provide certain functions while a batch job is running in order to control processor loading. This creates no problem. However, in other cases, global decisions must be made which depend upon batch status. In these cases, batch status must be made available globally.

Recovery

When a failed node returns to service, it must be resynchronized with one of the active nodes. This can be accomplished via an online copy that updates the recovered node's database from an active node while the active node is still processing transactions. Alternatively, the changes that occurred during the failure can be queued on an active node and flushed to the recovering system. It may be desirable to implement both procedures. Normally, the change queue will be used to recover a failed system. But if the system has been down for so long that it will be faster to copy the database rather than to drain a large queue of changes, then the online copy will be used.

Application Management

In an active/active system, applications are running in a distributed environment. The application management tool currently being used to monitor and configure the system may not be extendable to a networked environment. In this case, a new distributed-management tool will have to be selected.

Furthermore, there must be a means to distribute configuration changes to the nodes in the application network without taking down the system. Configuration changes can range from parameter changes to application upgrades. Configuration parameters may have to include parameters whose values are node-specific.

Sizzling-Hot Standby

If an application cannot be architected to be active/active, it may still be possible to obtain the availability of an active/active architecture without having to modify the application. This is accomplished by employing two systems that are up and running with all applications active and with bidirectional replication configured, but all transactions are sent to only one system. The backup system can, of course, be used for queries and reports.

In this case, the application is not running in a distributed environment. It is processing all transactions just as if it were an isolated system. The only difference is that it is replicating its database to the backup system.

Because the backup system has all of its applications up and running, it is available to take over processing immediately. Should the primary system fail, all that needs to be done is to reroute transactions to the standby system.

Even more important, the standby system is known to be working (it can be easily tested by sending it periodic test transactions) so that failover will be reliable. Failover faults are eliminated.

There are still some application concerns as listed above, but most are irrelevant. Data collisions are eliminated. However, data maintained in memory will not be replicated, and a distributed application management framework is still required.

Summary

Converting an existing application or designing a new application to run in a distributed active/active environment is not a trivial task. It seems that the complexity of this task has dissuaded organizations from benefiting from the extreme availability of active/active architectures. Only if there is an investment in mainframe-type systems such as HP NonStop, OpenVMS clusters, or IBM Parallel Sysplex do organizations make this investment.

However, there are many applications using commodity Windows and Linux servers that suffer major costs due to downtime. The common method today to protect these applications is to run them in a cluster. However, clusters are difficult to configure and manage and typically take several minutes to fail over – much longer if the common database has been corrupted by the failure of the active server. Furthermore, clusters cannot be distributed geographically – all of the servers in the cluster must be collocated.

Though it may be difficult (and perhaps impossible) to modify existing applications to run in a distributed environment, organizations should consider implementing new mission-critical applications to run active/active. The savings in downtime costs may well overwhelm the cost of some architectural complexities.