the Availability Digest

www.availabilitydigest.com

Synchronous Replication Recovery Strategies

March 2010

The recovery of a database in an active/active network following a processing node failure is somewhat more complex if synchronous replication is used to keep the database copies in synchronism rather than asynchronous replication. This article explores several methods for resynchronizing a database in a synchronous-replication environment and points out their advantages and disadvantages.

Active/Active Networks

Active/active networks[1] comprise one or more geographically-separated processing nodes all cooperating in a common application. Each node has access to a distributed copy of the application database. Consequently, a transaction may be sent to any processing node in the application network and be properly processed. Should a node fail, all that needs to be done is to reroute its transactions to surviving nodes. As a result, recovery from a node failure can be accomplished in seconds, leading to extremely high availabilities. If recovery is so fast that no one notices the outage, continuous availability has been effectively achieved.

Data Replication

The distributed database copies in the active/active network are kept synchronized via data replication. When a processing node makes a change to its database, that change is immediately replicated to the other database copies in the application network so that each processing node always sees an up-to-date application database.

There are two fundamental methods for data replication in active/active systems – asynchronous replication and synchronous replication. They both support bidirectional replication and concurrent access necessary for active/active systems. Bidirectional replication is necessary since each node must be able to update all other nodes even as those nodes are updating it. Concurrent access is required since an application in a processing node must be able to update its local database in parallel with other nodes.

Asynchronous Replication

Asynchronous replication[2] replicates changes "under the covers." It has no impact on the applications themselves. As updates are made to the local database, the changes are entered into a change queue of some sort. This can be done, for instance, by a transaction monitor, by the application, or via database triggers.

[1] What is Active/Active?, *Availability Digest*; October 2006.
[2] Asynchronous Replication Engines, *Availability Digest*; November 2006.

An asynchronous replication engine follows the change queue, extracting changes and sending them to the target system. There, the replication engine applies them to the target database. There is some delay (typically tens or hundreds of milliseconds) from the time that a change is made to the source database to the time that it is made to the target database. This time delay is called *replication latency*.

Replication latency causes two problems that must be accommodated:

- Should the source node fail, changes still in the replication pipeline may be lost.

- Should applications at two nodes modify the same data object simultaneously (that is, within the replication interval), the databases will be different and will diverge. This is called a *data collision*.

Recovery of a failed node is straightforward. Should a node fail, the surviving node simply continues to queue changes to the failed node. When the failed node is restored, the queue of changes is drained to the recovering node to resynchronize it. Changes that arrive during the recovery process are simply added to the end of the change queue. When the change queue has reached an acceptable length, the recovered node can be returned to service.

Synchronous Replication

Synchronous replication[3] solves the asynchronous-replication problems of data loss and data collisions. With synchronous replication, no change is made to any database copy unless all copies can accept that change. The synchronous replication engine obtains locks on all copies of the data object to be updated before applying the updates.

Therefore, the database copies are kept in exact synchronism. Since all database copies must be updated or none are, there can be no lost data. Also, the locks ensure that only one change at a time will be applied to any given data object copy in the application network, thus eliminating data collisions.

However, synchronous replication comes with its own problems:

- Because the applications must wait for all updates to be made across the application network before proceeding, transaction-response times are extended. This time is called *application* latency. As a consequence, most synchronous replication engines impose limits on how far apart the processing nodes may be – typically several kilometers. This may not allow the degree of separation needed for required disaster tolerance.

- If a processing node fails, transactions can no longer be committed since the failed node cannot vote “yes” on the commit. Therefore, if no corrective action is taken, the application network comes to a halt.

It is this latter problem that we address in this article. How does an active/active system using synchronous replication continue in operation following a node failure, and how is that node subsequently returned to service?

[3] Synchronous Replication, *Availability Digest*; December 2006.

Continuing in the Presence of a Node Failure

Synchronous Replication While All Nodes are Operational

Most synchronous replication is performed using a two-phase commit protocol. When a set of changes within the scope of a transaction is to be made, the synchronous replication engine first obtains locks across the application network on all copies of the data object to be modified. The source node then asks the replication engine to confirm that all nodes are prepared to execute the transaction (the *prepare phase*). If a node is holding all of the requisite locks, it responds with a "yes" vote. If it cannot obtain all locks, it will vote "no."

The second phase is the *commit phase*. If all participants in the transaction voted "yes," the replication engines are instructed to apply the changes and to release their locks. In this case, the transaction has been *committed* across the application network. If any participant votes "no," the replication engines are instructed to release their locks and to apply no changes to the data object copies. The transaction has been *aborted*.

Synchronous Replication in the Presence of a Node Failure

Node Failure

Should a node fail, it can no longer respond to prepare requests. This lack of response will be taken as a "no" vote, and the transaction is aborted. Therefore, all transaction processing is halted.

To cure this situation, the failed node must be taken out of service. All further transaction processing must exclude that node's database copy from the scope of transactions, and all user requests must be directed to the surviving nodes. In this way, synchronous replication can continue with the remaining nodes.

Replication Network Failure and Split-Brain Mode

One serious problem occurs if the failure is not the processing node but rather is the network link connecting the processing node to the rest of the application network. In this case, a node cannot tell if a remote node is not voting because it has failed or because communication with it has been lost. An important and complex decision must be made. Should processing continue or not?

If processing continues in the presence of a network failure, the nodes on either side of the failure will assume that the nodes with which they cannot communicate are down and will remove them from the scope of their transactions. The two isolated sets of nodes will proceed in what is called *split-brain mode*. Both node subsets will continue to process transactions, and their databases will diverge. In addition, if the database is not partitioned so that updates are limited to a master node for that partition, data collisions will occur. This is exactly what synchronous replication is supposed to avoid.

Therefore, a decision must be made as to whether the failure is a node failure and to remove that node from the scope of further transactions or whether the failure is a replication-link failure. In the latter case, a further decision must be made as to which subset of nodes to take out of service before continuing on with the remaining nodes. This is a complex decision, during which time transaction processing is halted.

There are at least two solutions to this dilemma:

- Pause processing until the problem can be determined and a course of action can be taken.

- Use an independent quorum system to monitor the health of all nodes and to take corrective action immediately.

The first solution adds downtime. The second solution adds complexity. Neither is attractive. Therefore, it is of utmost importance that the replication link be highly reliable. It should be redundant with automatic failover and with no single point of failure. The probability of a dual (or even triple) link failure should be so small that it is almost impossible to happen during the life of the system.

Recovering a Failed Node

Removing a failed node from the scope of transactions is one problem. Returning it to service is quite a different problem.

The first step is to recover the failed node's database copy and to bring it into exact synchronism with the application network's database. This has to be done while the application database is being actively updated to avoid having to stop processing. The failed node's database must then be kept synchronized with the application database as the failed node is reentered into the scope of application transactions.

There are several ways that synchronous node recovery is being done in today's products. We review these methods below.

Synchronous Online Copy

One technique is to copy the operational application database to the recovering database and to keep synchronized that part of the recovering database that has been copied. Once the copy has been completed, the recovered database can be reintroduced into the scope of transactions without any impact on transaction processing.

This technique is used by OpenVMS Active/Active Split-Site Clusters,[4] and we will use that implementation as a model for the following description.

Full Copy

If a full copy of the database is required, it proceeds table by table. Tables may be copied in parallel if desired. We assume that the node being recovered has an old database copy that is current as of the time of failure (if not, a bulk load of the database may be made from one of the operational copies).

For a given table, the copy utility reads the first row from a designated active database copy (which we will call the "master") and compares that row with the corresponding row on the table being recovered. If they are the same, the copy utility advances to the next row. If both rows exist but are different, the master row replaces the target row. If the master row does not exist in the target table, it is inserted into the target table. If the target row does not exist in the master table, it is deleted from the target table.

One concern is that the source database is being actively updated during this process, and the copy utility has not locked the master row for performance reasons. Therefore, if a modification has occurred, the rows are reread and compared again. If they are different, that means that the master row changed during the comparison; and the above process is repeated. If several

[4] OpenVMS Active/Active Split-Site Clusters, *Availability Digest*; June 2008. http://www.availabilitydigest.com/public_articles/0306/openvms.pdf.

attempts fail to get a match (due to a highly active master row), the master row is locked and the procedure is once again repeated, this time with a guaranteed successful outcome.

A "fence" on the recovering table separates the part that has been copied from the part that has not been copied. Target applications can use the part of the recovering table before the fence for read-only operations. The part following the fence is off-limits.

Replicated changes are applied synchronously to the table rows that are before the fence. The target system will respond to a commit request with a "yes" vote if it is capable of applying all changes to its rows before the fence. It may or may not apply replicated changes to rows after the fence because these will be corrected by the copy utility when it gets to those rows.

When all tables have been successfully copied, the recovered database is in synchronization with the operational database. The surviving processing nodes can now be notified to include the recovered node in the scope of their transactions.

The downside of this technique is performance. Though the applications are never paused during the recovery process, each row in the database must be read at least once. This imposes an additional load on the master database and may affect performance during the copy. The copy facility can be throttled to control its copy rate so as to minimize the performance impact.

Partial Copy

If a processing node is to be removed from service for only a short time, a partial copy may be used. In this case, the designated master node keeps a list of the primary keys and tables of all rows that have been modified. When the node is returned to service, only those rows need to undergo the copy process described above.

Asynchronous Online Copy

Another technique for synchronous database recovery is to use asynchronous replication to catch up, and then switch over to synchronous replication. Using this technique, the surviving nodes queue changes to the failed node while it is down. If upon recovery the failed node does not have its copy of the database at the time of failure, an online load of the operational database is first made to it from a surviving node's database copy.[5]

The changes queued by the surviving nodes are then replicated asynchronously to the recovering database. As new changes come in, they are added to the surviving nodes' change queues and are replicated.

This process continues until the change queues have been drained to some specified minimum size.[6] At this time, transaction processing is paused to allow the change queues to completely drain. In-process transactions are allowed to complete, but no new transactions are started.

When the change queues have been completely drained, the recovering database is in complete synchronization with the operational database; and transaction processing can be resumed with the recovered node included in the scope of further transactions.

[5] This load must be performed without affecting the operation of the surviving nodes. A load facility such as SOLV from Gravic, Inc. will not only load the recovering database while the operational database is active but will keep it current during the load process, See www.gravic.com/solv.

[6] That a minimum size exists is shown by queuing theory. If the load on the replication channel is *L*, the probability that the queue length will be zero is (1 – *L*). Of course, this is complicated by change-queue polling and by communication blocking; but the result is the same. See W. H. Highleyman, pg. 121, Equation (4-79), *Performance Analysis of Transaction Processing Systems*, Prentice-Hall; 1989.

Using the online asynchronous copy method for database recovery eliminates the performance impact imposed by the online synchronous method. The performance impact is no greater than that imposed by asynchronous replication. In fact, in a two-node system, performance will probably be better when one node is down since there will be no application-latency delay.

However, with this method, the application must be paused at the end of the copy to let the recovering database catch up with the operational database. In many applications, this pause can be configured to be less than a second (about the time of a transaction). However, if the application executes long transactions (such as a batch process), application pause time may be as long as the longest transaction; and this method may not be appropriate.

Mixed Online Copy

The synchronous and asynchronous online copies described above each carry a penalty:

- The synchronous online copy imposes an additional load on one of the operational nodes during the copy. This is the node that is designated as the master node from which the copy is made.

- The asynchronous online copy requires that the application be paused for a short time while the change queue is finally drained. The duration of this pause depends upon the transaction-processing times required by the application.

Both of these problems can be solved by using a synchronous-replication method known as *coordinated commits*.[7] With coordinated commits, all changes are replicated via an asynchronous replication engine. As with normal asynchronous replication, changes are entered into a change queue. Changes are read from the change queue with no impact on the application and are sent to the target system, where they are applied to the target database.

However, in this method, there is an extra step. The asynchronous replication engine joins the source transaction and is therefore a voting member for that transaction. When the source transaction is ready to commit, the replication engine is asked to vote. If it has acquired locks on all data objects to be modified by the transaction, it votes "yes." If it cannot apply the transaction updates, it votes "no;" and the transaction is aborted. Application latency is minimized with coordinated commits since the application must only wait for the commit to complete across the network rather than having to wait for the network completion of each database operation plus the commit.

Database recovery is seamless using coordinated commits.[8] As with the asynchronous online-copy method, when a node fails, the surviving node queues changes to the failed node. When the node is to be recovered, the queue-draining process begins. Changes in the queue are sent to the target system, where they are applied to the target database. Synchronous changes are entered into the queue behind the asynchronous changes just as they would be entered during normal operation.

When all asynchronous changes have been applied, and when the remaining queue of synchronous changes has reached an acceptably short length, the replication engine can once again become a party to further transactions. However, there are still some queued changes that were part of earlier transactions to which the failed node was not a voting party. When these

[7] Achieving Century Uptimes Part 17: HP Unveils Its Synchronous-Replication API for TMF, *The Connection*; July/August 2009.
[8] Achieving Century Uptimes Part 18: Recovering from Synchronous-Replication Failures, *The Connection*; September/October 2009.

changes have been applied to the target database, the recovering node is now fully synchronized and can be returned to service.

With coordinated commits, there is no additional load placed on the surviving node during recovery. It continues to operate in its normal asynchronous-replication mode. Also, there is no need to pause the application to complete resynchronization. If a very long transaction was in process at the time that the replication engine began to rejoin transactions, all that happens is that the recovery of the failed node is delayed until that transaction completes.

Summary

Recovering the database of a failed node in an active/active system using synchronous replication is somewhat more involved than in such a system using asynchronous replication. Recovery must be accomplished in both cases without taking the applications down.

Recovery first requires that the node to be recovered have a copy of the application database that is current as of the outage or at some time later. Changes that have accumulated during the outage must then be applied while at the same time keeping the recovering database up-to-date with new changes coming in. Only when the recovering database is fully synchronized with the operational database can it be returned to service and begin processing transactions.

Recovery methods include synchronous online copying that imposes a load on a surviving system during recovery, asynchronous online copying that requires the application to be briefly paused before returning the recovered node to service, and a mixed online copy that avoids these problems but that depends upon a specific synchronous-replication architecture.

In any event, the impact of a node failure on a synchronously-replicated active/active system is no different than a node failure on an asynchronously-replicated active/active system. The failed node is removed from the active/active network, which continues to carry the application load until the node is returned to service.