

Transaction Replication

February 2007

Active/active systems depend upon the applications at different nodes having access to distributed copies of the application database. The database copies must all be synchronized so that they present the same application state to their local application instances.

This is typically done by real-time replication of data changes made at one database to the other copies in the application network. In our previous articles, we have discussed software and hardware replication of changes. In this article, we discuss database synchronization by transaction replication – the application of each transaction independently to all database copies.

We first review the other replication technologies before discussing transaction replication.

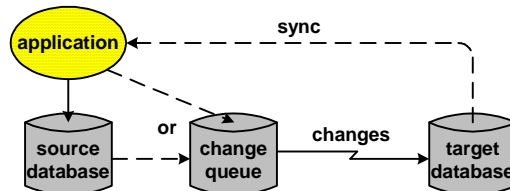
A Review of Replication Techniques

We have previously reviewed data replication techniques that replicated source database changes to a target database. There are two major techniques for doing this – software data replication and hardware data replication.

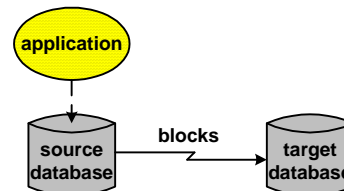
Software data replication¹ depends upon some sort of a Change Queue that contains an ordered sequence of changes that have been made to the source database. This Change Queue could be, for instance, an audit trail maintained by a database manager, or a log of changes created by the application or by database triggers.

A data replication engine reads changes from the Change Queue and applies them to the target database. In active/active applications, the changes made to each database copy are replicated to all of the other database copies to maintain all copies in synchronism.

Software replication can be done asynchronously, in which case it is transparent to the application. However, with asynchronous replication there is the potential for data loss should the



Software Replication



Hardware Replication

¹ See our articles entitled [Asynchronous Replication](#) and [Synchronous Replication](#) in the November and December, 2006, issues of the Availability Digest.

source system fail – transactions in the replication pipeline will be lost. Furthermore, there is the chance that simultaneous updates to the same row at different nodes will result in data collisions.

Data loss and data collisions can be avoided by using synchronous replication. With synchronous replication, the commit of a transaction is delayed until all database copies have informed the source system that they have safe-stored the changes associated with that transaction and are ready to commit. This results in somewhat longer transaction times.

Hardware replication² replicates changed physical blocks as they are flushed from cache to disk rather than replicating logical database changes. Hardware replication can also be asynchronous or synchronous. However, hardware replication suffers from the fact that the target database is not consistent during replication and cannot be used for processing. Therefore, hardware replication is not suitable for active/active systems.

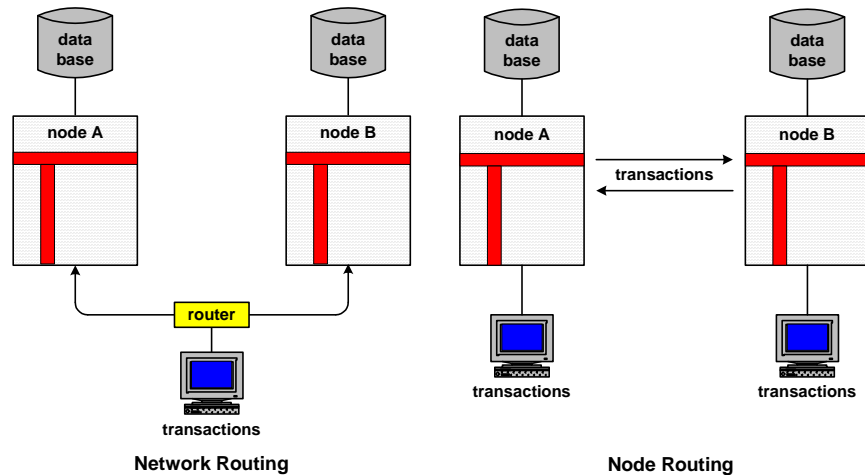
Transaction Replication

Transaction replication³ is yet another technique for maintaining database copy synchronization. Rather than replicating changes, transaction replication replicates entire transactions. Each system node that is directly attached to a database copy independently processes each and every transaction and updates its database copy.

As a result, each database copy contains an up-to-date copy of the application. All nodes are actively processing transactions, and the various copies can be used by applications at different nodes cooperating in an active/active environment.

Transaction Routing

There are two ways in which transactions can be distributed to the nodes in the application network. One is for the network to route each transaction to all nodes. Each node then processes the transaction independently of the other nodes and nearly simultaneously with those nodes.



The other technique is for each node to receive transactions from its own sources. As part of its processing, a node transmits each transaction to the other nodes, which also process the transaction. Thus, all nodes process all transactions. A potential advantage of this technique is

² See our article entitled [Hardware Replication](#) in the January, 2007, issue of the Availability Digest.

³ See Chapter 4, [Active/Active Topologies](#), *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems*, to be published in 2007.

that partially processed transactions may be sent to the other nodes, thus reducing their processing time for “second-hand” transactions. The downside of this method is that additional communication facilities may be needed.

Data Collisions

With asynchronous change replication as described earlier, there is the problem of data collisions. A data collision occurs when two users at different nodes attempt to update the same row almost simultaneously – within the replication latency time of the data replication engine (replication latency is the time that it takes for a transaction to be posted to the target system after it has been posted to the source system). The result is that each update is replicated to the other system and overwrites the update that already had been made there. The database copies are now different, and each is wrong.

Transaction replication does not suffer from data collisions. However, it brings with it its own form of potential database corruption. It is possible for each system to process the transaction differently. This will generally be caused by the nodes receiving transactions in a different order.

For instance, consider an order entry system comprising a two-node active/active system. Node A receives an order from Customer 1 for six items. Node B receives an order from Customer 2 for three of the same item. There are seven such items in stock. Node A processes its order and sends it to Node B. At the same time, Node B processes its order and sends it to Node A. Node A then processes Node B’s order, and Node B processes Node A’s order.

The result is that Node A has satisfied Customer 1’s order for six items and Customer 2’s order for one item and has backordered two items for Customer 2. However, Node B has satisfied Customer 2’s order for three items and Customer 1’s order for four items. It has back ordered two items for Customer 1.

This particular case can be disastrous if both systems have taken subsequent action independently and have attempted to send six items to Customer 1 (Node A) and three items to Customer 2 (Node B). The error would be noted by the shipping system, and one customer would have to be reprocessed and notified of the changed status of his order (to his consternation, no doubt).

The possibility for and type of data corruption that can be caused by transaction replication is very application-sensitive. The above example shows that data corruption can indeed happen. The only correction is to periodically compare the databases via an online compare and repair facility⁴ and denote one of the databases as the database of record. Any discrepancies are resolved by synchronizing the divergent databases with the database of record. This, however, corrects the problem after the fact and may not be acceptable.

In some cases, after-the-fact database correction may be acceptable, especially if no immediate action is taken based on the outcome of the transactions. There are also applications for which this type of database corruption will not occur. Applications which only insert rows into a database are such an example. For instance, a telephony application which logs call detail records for later billing is such an example.

⁴ See Chapter 12, SOLV, in *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems*, to be published in 2007.

System Capacity

With transaction replication, all systems are processing all transactions; so the highly scalable attribute of active/active systems is lost. Though the system will provide the rapid failover inherent with active/active systems, its capacity is limited to that of a single node. It cannot be scaled by simply adding nodes.

This problem can be somewhat softened if node routing is used and if the “second-hand” transactions are partially processed so that their subsequent processing time in the other nodes is reduced. In this case, each node fully processes the transactions directly routed to it but has reduced processing requirements for transactions that have been forwarded to it by other nodes. Partial scalability has been achieved.

This capacity issue may not be of concern in a two-node active/active system if each node must be configured anyway to be able to handle the full load should the other node fail.

Applicability to Active/Active Configurations

In spite of the data corruption and scalability problems, transaction replication has found applicability to active/active systems.

- It may be easier to implement an active/active system with transaction replication since no special facilities need be installed and managed for data replication.
- In a two-node system, the dual processing of each transaction may not be seen as a problem since each node has to handle the full processing load anyway should the other node fail.
- Sending transactions to the nearest node can take advantage of locality to improve performance by minimizing network latency.
- Spreading transaction processing across multiple nodes gives one the confidence that the backup systems will be working when needed since each is actively processing transactions.
- Transaction replication is a better use of a backup system than letting it sit idle as a cold standby. By letting the backup system also process all transactions, it is immediately available as a “sizzling-hot standby.” It can take over processing in the event of a failure of the primary system by simply switching users to it. Its applications are already up and running, and its database is current. If transactions are sent to it by the primary system rather than by the network, transaction order is guaranteed; and the possibility of data corruption as described above is virtually eliminated.

A caveat is appropriate here for transaction replication and is applicable to all active/active systems. Special care must be taken to ensure that the application is active/active-ready.⁵ There are many application constructs that are not compatible with active/active processing. For instance, if unique number generators are used (for instance, to create customer or part numbers), node uniqueness must be added to prevent the duplication of numbers. Locking protocols may have to be modified. Memory-resident context may have to be made available to

⁵ See Chapter 8, *Eliminating Planned Outages with Zero Downtime Migration*, *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems*, to be published in 2007.

other nodes. Care must be taken to ensure that the application can, in fact, run in an active/active environment.

Advantages of Transaction Replication

In summary, the advantages of transaction replication include the following:

- It provides the availability advantages of active/active systems. Extreme availabilities with near-instantaneous recovery can be achieved.
- It is nonintrusive. Applications do not need to be changed (providing that they are active/active-ready).
- It can provide locality for transaction processing.
- It can be used easily for insert-only applications.
- It is a better use of a cold standby system and provides the capability for immediate and assured recovery.

Disadvantages of Transaction Replication

Transaction replication comes with some significant disadvantages:

- It is not scalable. If some pre-processing is done on second-hand transactions, it may be somewhat scalable. However, scalability may not be a problem if each node must handle the entire transaction load in a failure situation.
- It can create inconsistent database copies for applications other than insert-only.
- Recovery of a failed node requires an online database copy. There is no Change Queue available from which to recover by simply draining the queue of changes that had accumulated during its downtime.

Examples

There are many examples of the successful use of transaction replication.

- Bank-Verlag uses transaction replication to operate a pair of NonStop nodes in an active/active configuration.⁶ This system provides debit card maintenance and authorization for 300 German banks. Because this system is fundamentally subject to data corruption due to different processing paths for a transaction, Bank-Verlag verifies the database periodically by comparing the two databases. Interestingly, they report that they have not seen a case of data corruption in seventeen years of operation.
- The New York Racing Association's first totalizator system used transaction replication to provide a sizzling-hot standby configuration.⁷ The totalizator system accepts wagers and posts odds and payoffs on the large infield boards and on monitors around its Aqueduct, Belmont, and Saratoga race tracks. In its case, means are provided to synchronize the order of transaction processing between the active and backup systems to avoid database corruption.
- A major retailer uses transaction replication to run a gift-card system. This is an insert-only application so that data corruption is not a problem.

⁶ See our article entitled, [Bank Verlag – The Active/Active Pioneer](#), in the December, 2006, issue of the Availability Digest.

⁷ See our article entitled, [CPA at Aqueduct, Belmont, and Saratoga Race Track](#), in the January, 2007, issue of the Availability Digest.

Summary

Transaction replication is another technique that is being used today for synchronizing database copies in an active/active system. It comes with its own set of advantages and issues.

The primary issue with transaction replication is database corruption should a transaction be processed differently in two different nodes. There are certain types of applications that are corruption-safe, such as insert-only applications.

Scalability is another concern since all nodes must process all transactions. However, in two-node active/active applications, this may not be a problem if the nodes have to be sized anyway to carry the entire transaction load. This is necessary if a node is to be able to provide the required capacity in the event of the failure of the other node.

Replicated transactions may provide an easier implementation path than data replication. It is an ideal technique to support a sizzling-hot standby system for immediate and assured failover.