the Availability Digest

Asynchronous Replication Engines

November 2006

Active/Active Systems

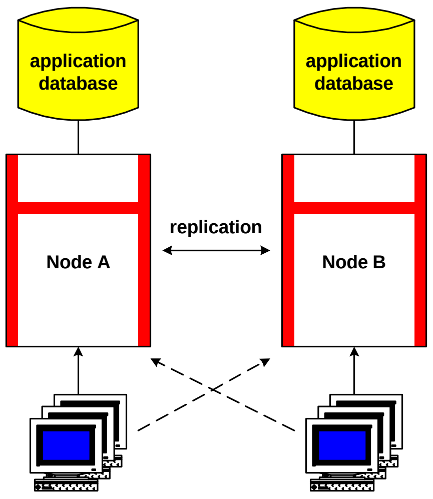
A fundamental tenet of active/active systems is that there are at least two copies of the application database in the network. These copies and the processing nodes which use them should be geographically distributed to ensure that the application will be tolerant to problems affecting a wide area.

The database copies must be kept in synchronism to ensure that any node can use any database copy and end up with the same result. This means that as soon as a change is made to one of the copies, it must be replicated to the other database copies in the application network.

There are many ways to do this, including:

- asynchronous replication
- synchronous replication
- transaction replication
- network transactions

In this article, we will look at the techniques and issues of asynchronous replication. We will talk about the other methods in later articles.



An Active/Active System

The Replication Engine

With asynchronous replication, the source and target systems are loosely coupled. Changes made to the source database are queued for replication to the target database. At some later time (which may be in milliseconds), a replication engine picks up the changes and sends them to the target system where they are applied to the target database. The source applications are in effect unaware of this transfer of data changes and are unaffected by replication.

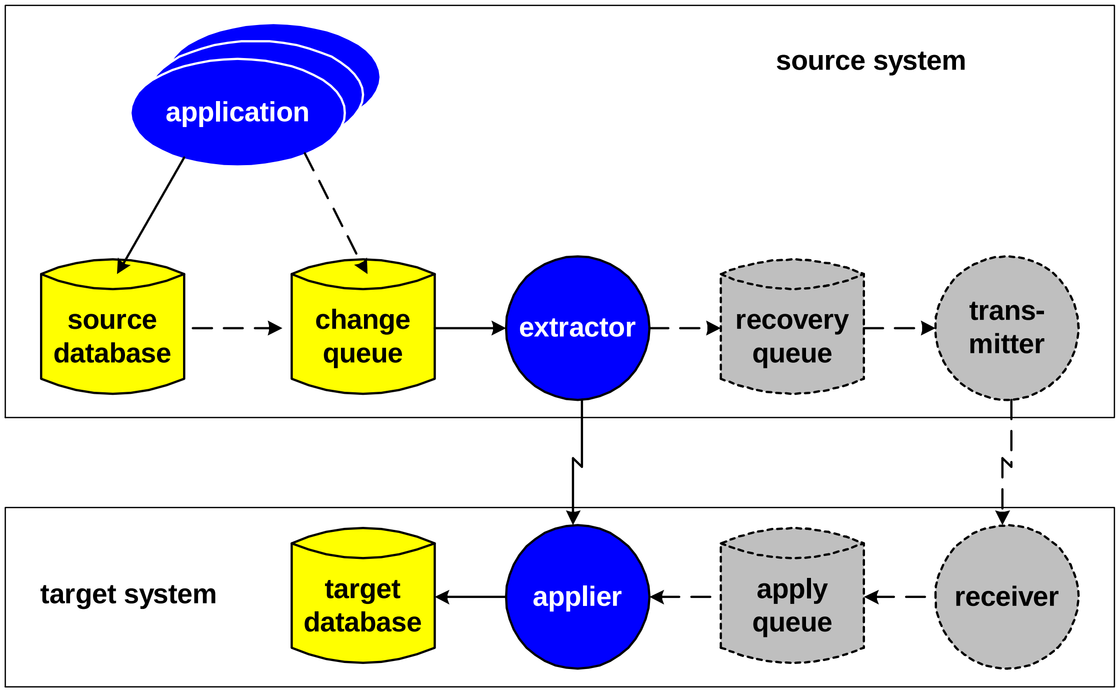
Asynchronous replication is done via an asynchronous replication engine.[1] Although there are many forms of replication engines in the marketplace, by and large they follow the same general architecture in order to replicate data from a source database to a target database.

[1] Asynchronous replication engines are described in great detail in the book entitled *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing,* by Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein.

A replication engine typically depends upon some form of a real-time log of changes. We will call this log the *Change Queue.* Information describing each change made to the source database is entered into the Change Queue either by the database manager, by triggers in the database, or by the application itself. The Change Queue is usually disk-resident so that the replication engine can recover from node or engine failures.

An *Extractor* process follows the tail of the Change Queue and extracts the description of a data change as it is logged. It sends the data change information to the target system so that it can be applied to the target database.

Data changes may be sent directly to the target system by the Extractor, or they may first pass through another disk queue used for recovery purposes. In the latter case, the Extractor will write the change to the disk-resident *Recovery Queue*. A *Transmitter* process then reads the recovery queue and sends the data to the target system.



Asynchronous Replication Engine

At the target system, change information may be received directly by an *Applier* process, which updates the target database with the changes. Alternatively, the change information may be written to another intermediate disk queue which is used to control the sequence of changes applied to the target database. The Applier can use this *Apply Queue* to filter out aborted transactions and to apply transactions in their original order to guarantee referential integrity.

Replication Latency

One of the most important characteristics of a data replication engine is its *replication latency*. Replication latency is the time that it takes for a change to propagate from the source database to the target database. As we shall see later, replication latency creates some of the undesirable characteristics of data replication. These characteristics include data loss following a node failure, data collisions, and application performance if synchronous replication is being used.[2]

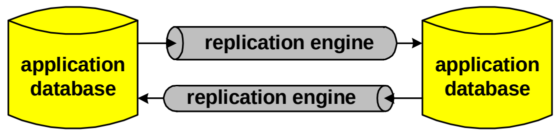
Many replication engines have replication latency measured as subseconds. Others require many seconds or more to propagate changes. There are many causes of replication latency, including:

- The number of disk queuing points in the replication path.
- Buffering delays at the communication channel as changes are batched to improve communication efficiency.
- The speed of the communication channel.
- Whether the processes that follow the tail of a disk queue are event-driven or use polling.
- If disk-reading processes are poll-driven, what is their polling interval.
- The delay required to reserialize commits to guarantee referential integrity.

It is important to choose a replication engine whose replication latency is satisfactory for the application.

Bidirectional Replication

The replication engine which we described above is a one-way replication engine. Of course, for active/active systems, replication has to be bidirectional. This is accomplished simply by configuring two replication engines, one replicating in each direction.



Bidirectional Replication

If there are more than two nodes in an active/active network, a bidirectional pair of replication engines is required between enough of the nodes to create a fully connected network. Note that not all nodes in an active/active network need have database copies, and some nodes might be configured to forward changes to other nodes.

Advantages of Asynchronous Replication

Since an asynchronous replication engine feeds off a Change Queue that may be created by the application anyway (such as a change log created by a database manager), the replication process is totally transparent to the applications. It happens under the covers, and except for a small additional processor load required to support the replication engine, it has no impact on the application.

In addition, asynchronous replication is totally noninvasive. No changes to the application are required. Replication engines are off-the-shelf products that simply plug in.

One exception to the above is if there is no inherent Change Queue created by the application. In this case, the application must be modified to create a change log of some sort. This will also increase the processing load somewhat.

[2] A detailed performance analysis for replication engines is provided in the forthcoming book entitled *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems,* by Paul J. Holenstein, Dr. Bill Highleyman, and Dr. Bruce Holenstein.

Of course, the most obvious advantage of an active/active system is the ability to switch users rapidly to surviving nodes should a node fail. However, this is a characteristic of the active/active architecture. Each of the data replication methods mentioned above provides this capability.

Asynchronous Replication Issues

There are several considerations that must be taken into account when contemplating asynchronous replication for active/active systems. These include referential integrity, ping-ponging, data loss following a node failure, and data collisions.

Referential Integrity

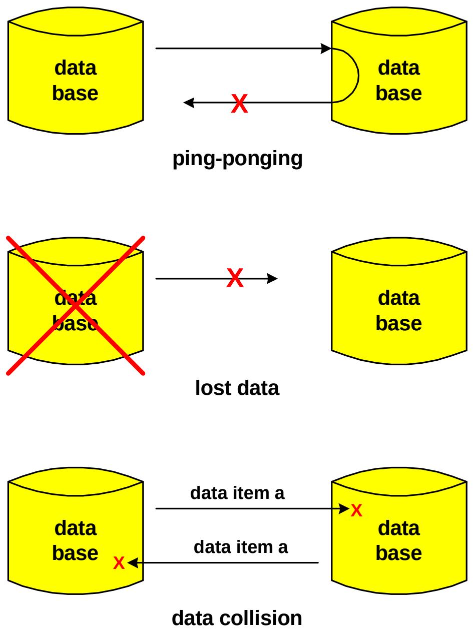
The data replication engine must guarantee referential integrity. Only then can all database copies be used for application processing.

If a data replication engine does not guarantee that transactions are applied to the target database in the same order that they were applied to the source database, the database copy will not be consistent. Child rows may exist without parents. Indices may point to nothing. New data may be overwritten by old data.

This is particularly a problem with hardware replication schemes in which a physical data block is replicated without regard to transaction boundaries. It is also a characteristic of some software-based replication engines.

Ping-Ponging

In order to run in a bidirectional configuration, the replication engine must prevent ping-ponging. This is the return to the source system of a change just received from the source system.



Asynchronous Replication Issues

Data Loss Following a Node Failure

Should a node containing a database copy fail, it is likely that there will be changes still in the replication pipeline at the time of failure. These changes will never be propagated to the target system and will be lost. There is no way to recover them unless the failed node can be restored quickly.[3]

This is one case in which replication latency is important. The longer the replication latency, the more likely it is that data will be lost following a node failure.

Data Collisions

A data collision occurs if two users should update the same data item at two different database copies within the replication latency interval. In this case,

[3] HP NonStop servers support a remote mirror of their audit trail. The remote mirror guarantees that no transactions will be lost following a node failure. HP calls this configuration ZLT for *Zero Lost Transactions*.

each new value of the data item will be replicated to the other system and will overwrite the original change made at that system. As a result, the database copies are different and both are wrong.

This is another case in which replication latency is important. The shorter the latency time, the less likely it is that there will be a data collision.[4]

There are several ways to attack the problem of data collisions:

Avoidance: Data collisions can be avoided by:

- Partitioning the database so that a particular data item is always updated on a designated database copy. Those changes are then replicated from that copy to the other database copies in the application network.
- Creating a master node to which all updates are directed. The master node then replicates changes to the other nodes in the network.
- Using relative replication to replicate operations on the data rather than replicating the final value of the data item itself (for instance, add 10 or subtract 5).
- Using synchronous replication as described in our next article.

Detection and Resolution: Collisions can be detected by comparing the version of the data item to be updated to the version of the update. If they differ, a collision has occurred. There are several options for resolving a detected collision:[5]

- Establish a node precedence. The node with the highest precedence wins, and its data value is accepted.
- Use data content to resolve a collision. For instance, the update with the most recent timestamp may win.
- Ignore collisions if the database will be self-correcting over time due to other noncolliding updates.
- Ignore collisions and periodically resynchronize the databases to one designated as the database of record.
- If all else fails, collisions must be resolved manually.

Products

There are several replication engines off the shelf that purport to support active/active architectures. Following is only a partial list of the many available products.[6] Be aware that some

[4] The probabilities of data collisions for different circumstances are derived in Chapter 9, Data Conflict Rates, in the book entitled *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing,* referred to earlier.

[5] The handling of data collisions is extensively covered in the *Breaking the Availability Barrier* series of books referenced earlier. See especially Chapter 4, Active/Active and Related Technologies in the forthcoming book, *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems.*

[6] Others are also listed in Appendix 4, Implementing a Data Replication Project, in the book entitled *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing,* referred to earlier.

vendors may redefine the term active/active to fit their product's capabilities, so be sure to analyze the characteristics of a data replication engine before committing to its use.

- Streams from Oracle (www.oracle.com)
- Times Ten from Oracle (memory-to-memory replication) (www.oracle.com)
- DRNet from Network Technologies (www.network-tech.com)
- Shadowbase from Gravic (www.gravic.com)
- GoldenGate for Active/Active from GoldenGate (www.goldengate.com)
- Sun Cluster Geographic Edition (Availability Suite) for Sun (www.sun.com)
- Metro Mirror (for Parallel Sysplex) from IBM (www.ibm.com)
- Global Mirror (for Parallel Sysplex) from IBM (www.ibm.com)
- RepliStor from EMC Legato (software.emc.com)
- Cluster Replica SQL for MSSQL from XLink (www.xlink.com)
- NSI Double-Take from Double-Take (www.doubletake.com)
- DataXtend RE from Progress (www.progress.com)
- MetiLinx Database Suite (for MySQL) from MetiLinx (www.metilinx.com)
- Colada from MARSYS (www.marsys.com)

Summary

Asynchronous replication is by far the most popular replication tool used in today's active/active systems. It is fast, non-intrusive, and under-the-covers.

It does have its issues, which must be understood before moving to an asynchronous active/active environment. These issues include the assurance of referential integrity of the replicated data, lost data following a node failover, data collisions, and minimizing replication latency.

Available today are many products that support asynchronous replication. There is virtually a product for any platform, and many products are heterogeneous in that they can replicate between disparate platforms.

Active/active is here today, as are the products that support it.