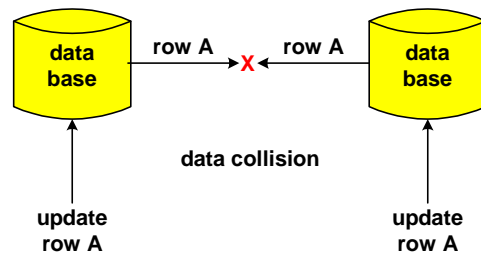


Collision Detection and Resolution

April 2007

In an active/active system, the various database copies in the application network are kept in synchronism through some mechanism. These mechanisms include synchronous replication, asynchronous replication, and transaction replication.¹

Asynchronous replication is the most commonly used technique today for database synchronization. However, it presents a database consistency problem. There is a certain amount of time for a change made to one database to be propagated to another database copy. This time is called *replication latency*. If applications attempt to update the same data item on two different nodes within the replication latency time, these updates will be replicated to the other database copy and will overwrite the changes originally made there. The two database copies will be different, and both will be wrong. This is called a data collision.²



If data collisions cannot be avoided, they must be detected and resolved in order to maintain consistency of the distributed application database. This is the focus of this article.

However, synchronous replication and transaction replication are not without their similar problems. These problems are also discussed.

Probability of Collisions

The probability that a data collision will occur is directly proportional to the replication latency of the asynchronous data replication engine. Knowing this probability is key to determining the extent of the data collision problem.

For a two-node active/active system with updates distributed uniformly over the database, the data collision rate is given by³

$$\text{data collision rate} = 2 \frac{u^2}{D} L$$

where

¹ See the *Availability Digest* articles [Asynchronous Replication Engines](#), [Synchronous Replication](#), and [Transaction Replication](#) in the November and December, 2006, issues and in the January, 2007, issue, respectively.

² Data collisions are discussed in much more detail in Chapter 4, [Active/Active and Related Topologies](#), *Breaking the Availability Barrier: Achieving Century Uptimes with Active/Active Systems*, AuthorHouse; 2007.

³ A more general form of this relationship is derived in our next month's article, [Data Collision Probability](#).

u is the update rate (rows per second)
 D is the database size (rows)
 L is the replication latency (seconds)

For instance, in a system with a replication latency of 100 milliseconds, ten updates per second to a database containing one million rows leads to a collision rate of less than two per day. This may not be deemed to be a problem. But if the update rate grew to 100 updates per second, data collisions grow to almost 200 data collisions per day. This could be a major headache.

Collision Avoidance

There are certain types of applications for which data collisions will not occur. These include applications which are insert-only, such as logging applications, and applications in which there is only one physical entity (such as a patient in a hospital management system), which precludes multiple transactions on that entity from being active simultaneously.

However, in applications that are susceptible to data collisions, there are certain architectures that can be used to avoid collisions.

Synchronous Replication

If synchronous replication can be used, there will be no data collisions. Synchronous replication obtains locks on all required data items in all database copies across the network before making changes to them. Thus, no other application can modify these data items until the locks are released. This guarantees proper serialization of updates.

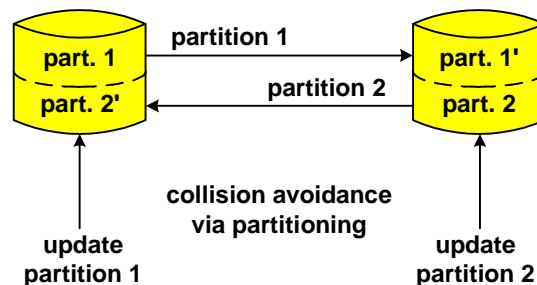
Synchronous replication is subject to deadlocks just as any other locking protocol, and standard deadlock resolution techniques are applicable. This includes using an *intelligent locking protocol*, which specifies the order in which locks are obtained, or deadlock-resolution procedures, such as backing off and trying again.

However, synchronous replication is also subject to a special form of deadlock due to the replication latency of the data replication engine. Since it takes a time equal to the replication latency to propagate a lock from one database copy to another, it is possible for applications in two different nodes to obtain a lock on the same data item. Neither application can proceed because another application is holding a lock that it needs. In this case, the replication latency becomes *lock latency*; and data collisions become deadlocks.

Such deadlocks can be resolved by timing out or backing off. Alternatively, global mutexes can be established with an intelligent locking protocol. In this case, global mutexes reside on a designated master node. Applications must acquire the lock on the remote global mutex before proceeding.

Partitioned Database

Data collisions can be avoided by partitioning the database so that each partition is "owned" by only one node and can only be updated by the owning node. All updates to a partition must be sent to the owning node. As the owning node makes updates to its partition, these updates are replicated to the other partition copies in the application network.



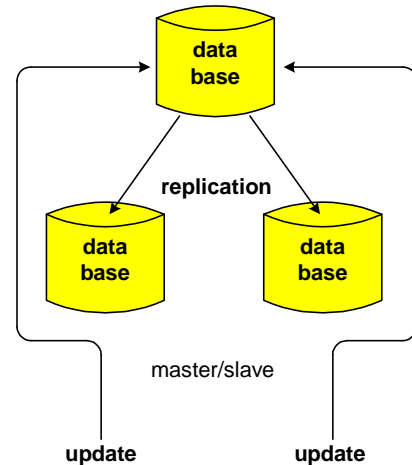
Transactions might be routed by intelligent routers based on data content, or they might be forwarded to the owning node by routing software in each node.

It may be acceptable to back up each partition on only one other database copy. If this is acceptable, there will only be two copies of the database in the application network.

Hierarchical Architecture

A hierarchical structure comprising one master node and multiple slave nodes can be used to avoid collisions. In this architecture, all updates are sent to the master node. The master node makes all updates to its database copy and replicates these updates to the slave copies. Since all updates are being made to only the master database copy, there will be no data collisions.

However, all transactions at the slave nodes will be slowed because the transactions must be sent across the network to the master node.



Collision Detection

If collisions cannot be avoided, they must be detected and resolved either automatically or manually. Data collisions can be detected by ensuring that the version of the row or record on the target system is the same as that which was updated on the source system. If the target row version is different from that of the source row, a collision has occurred.

Version checking is accomplished by sending some sort of row version indicator with the update so that the source and target row versions can be compared and verified. There are several ways in which this can be accomplished, including:

- sending the before image of the source row.
- time stamping each row with the time of its last update and sending the timestamp of the source row.
- adding a version number to each row and sending the version number of the source row.
- sending a checksum of the source row.

If a collision is detected, the participating systems must make a determination as to how to resolve it, as described next.

Collision Resolution

There are many techniques for resolving data collisions. Some of the more common ones are discussed below.

Resolution by Data Content

The collision can be resolved based on data contained in the row. For instance, the latest (or earliest) update could be accepted.

Node Precedence

The nodes in the active/active network could be given precedence. If a collision occurs, the update from the node with the highest precedence would be accepted by both systems.

Relative Replication

Most replication engines send the entire image (or a compressed version thereof) of the modified row to the target system. This image overwrites the current contents of the row on the target system.

With relative replication, only the operation on the field is replicated. For instance, if Node A added 10 to a field, and if at the same time Node B subtracted 4 from that field, then Node A would tell Node B to add 10 and Node B would tell Node A to subtract 4. The result is that the field would be increased by 6 in both nodes.

When using relative replication, one must be careful not to combine operations that are not commutative. As we have seen above, addition and subtraction can be replicated. Likewise, multiplication and division can be replicated. For instance, $(10 \times 2) / 5 = (10 / 5) \times 2 = 4$.

However, addition and subtraction cannot be intermixed with multiplication and division. $(10 + 2) \times 3 = 36$ is not the same as $(10 \times 3) + 2 = 32$.

Ignore the Collision

In some cases, it may be decided to ignore collisions. For instance, a statistical application may be little affected if a handful of numbers in a very large set are inaccurate.

It also may be felt that collisions will be self-correcting over time as later noncolliding updates overwrite earlier erroneous results caused by collisions.

Business Rules

There may be cases in which collisions can be resolved via specialized business rules. Many replication engines allow such business rules to be bound into the engine and invoked as user exits.

Manual Resolution

Should all else fail, the collision should be logged for later manual resolution. It is, of course, the goal of the automatic collision resolution algorithms described above to minimize the need for manual resolution.

Logging Data Collisions

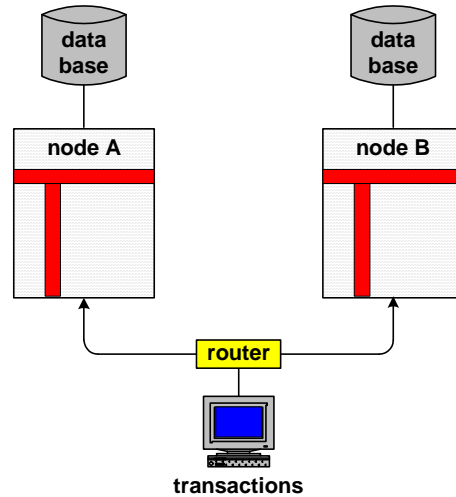
No matter the outcome, all collisions and their resolutions should be logged for later review to ensure that the collision resolutions are appropriate.

Transaction Replication

Transaction replication is another architecture for active/active systems. With transaction replication, a transaction is sent to two or more systems, where it is independently executed. If all goes well, the databases at the various systems will remain identical without the need for data replication.

Aside from the loss of the scalability of active/active architectures (all nodes must process all transactions), transaction replication systems bring their own form of data collisions. Unless the systems are run in lockstep,⁴ there is no guarantee that all systems will process the transactions in the same way. For instance, transactions may be received in different orders. Functions depending upon the current time may execute differently. A disk error may cause a transaction to abort in one system but not in the others.

There is no way to detect such collisions as they occur. Rather, the databases should be compared periodically to find any differences and then resolved.⁵ In fact, such a comparison should also be run against databases synchronized via asynchronous replication to ensure that all data collisions have been successfully resolved.



Summary

Since each node in an active/active system is processing transactions independently of each other, there is the possibility that nodes may process transactions in different orders. Depending upon the method used to synchronize the database copies, this can lead to various problems:

- If synchronous replication is used, deadlocks may occur. Deadlocks are immediately observable to the applications and can be resolved with well-understood techniques.
- If asynchronous replication is used, data collisions may occur. Data collisions can sometimes be avoided by proper application structure. Otherwise, they can be detected and in some cases resolved automatically.
- If transaction replication is used, the results of the transaction may lead to diverging database copies. Diverging database copies can only be detected by periodic database comparisons and require manual correction.

In the design of active/active systems, it is important that these problems be understood and that procedures be in place to resolve them should they occur.

⁴ See [CPA at Aqueduct, Belmont, and Saratoga](#) in the November issue of the *Availability Digest* for an example of a lock-stepped transaction replication system.

⁵ See [Flexible Availability Options with GoldenGate's TDM](#) and [Shadowbase – The Active/Active Solution](#), the *Availability Digest*; February, 2007, and March, 2007, respectively.