the Availability Digest

What is Active/Active?

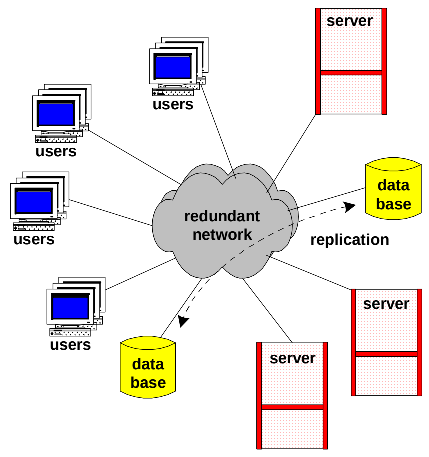
October 2006

It is a fundamental fact that any system can and will fail at some point. The secret to achieving extreme availabilities is to *let it fail, but fix it fast*. This is the premise behind active/active. If a service outage is too short for users to notice, it will not be perceived as a service outage. In effect, service availability has been maintained.

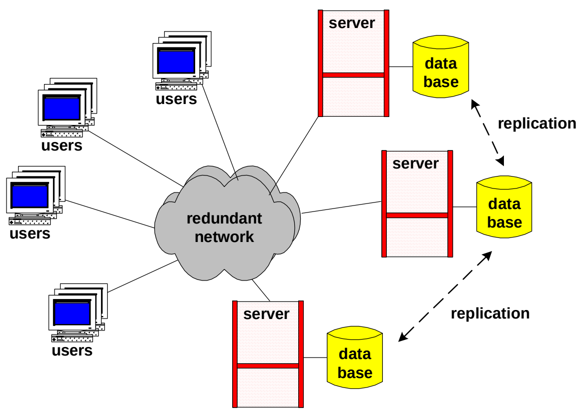
From a high-level perspective, active/active architectures accomplish fast recovery by distributing the user base over multiple independent and geographically distributed processing nodes. Should a node fail, the users of that node are switched to one or more surviving nodes in the application network. This switchover can often be done in subseconds – taking no more time than the resubmission of a failed transaction, an otherwise common occurrence.[1]

Active/Active Architectures

An active/active system is a network of independent processing nodes, each having access to a common replicated database such that all nodes can participate in a common application.



An Active/Active System with Network-Attached Storage



An Active/Active System with Direct-Attached Storage

The application network comprises two or more processing nodes connected to a redundant communications network. Any user can be connected to any node over this network. The nodes all have access to two or more copies of the database. The database copies may be connected directly to the network, or they may each be directly attached to one of the nodes.

[1] Much more information concerning active/active architectures may be found in the book entitled *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, by Dr. Bill Highleyman, Paul J. Holenstein, and Dr. Bruce Holenstein, published by AuthorHouse; 2004.

The database copies are kept in synchronism by ensuring that any change to one copy is immediately propagated to the other copies. There are many techniques for doing this, such as network transactions or data replication. These techniques are described later in this article.

Should a node or its attached database fail, the users connected to that node have lost their services. In this case, they are switched to another node or are distributed across multiple surviving nodes to immediately restore service. Users at other nodes are unaffected. (See *Do You Know Where Your Train Is?* for a description of one effective way to switch users.)

Providing that the nodes and database copies are geographically distributed, active/active systems provide disaster tolerance. Should a disaster take out a node or a database site, there are others in the network to take its place.

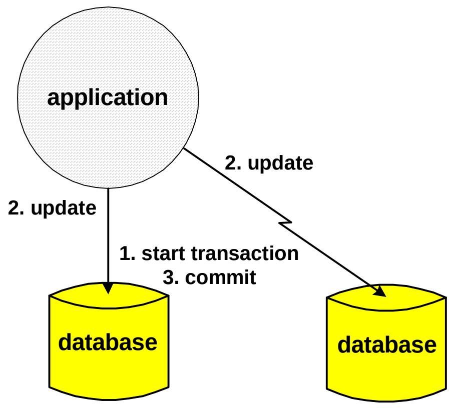
There are many ways to keep two or more database copies in synchronization. These include network transactions, asynchronous data replication, and synchronous replication.

Whatever the technique, one result is mandatory. The database copies must always maintain referential integrity so that each can be used actively by any application copy. Referential integrity typically means that transactions initiated by a node be committed in the same order at any of the database copies. In some cases, this requirement may extend to the updates within a transaction. This is known as preserving the “natural flow” of all updates.

Network Transactions

Using network transactions, the scope of a transaction includes all database copies. This results in each copy of a data item across the network being locked before any copy is updated. As a result, all database copies are kept in exact synchronism.

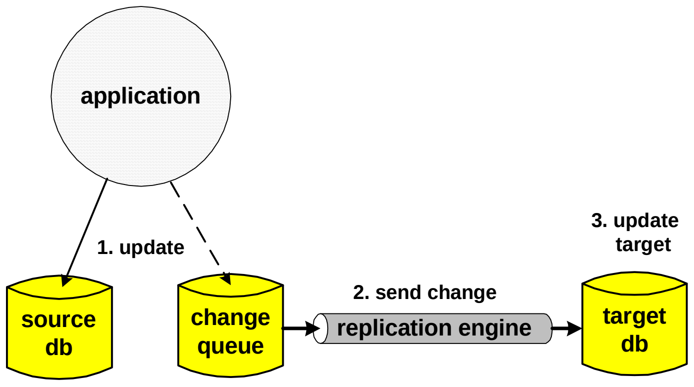
One problem with network transactions is that each lock request and each update must individually flow across the network and a completion response received. In widely dispersed systems, such a round trip could take tens of milliseconds; and application performance can be seriously affected. This is a clear example of the compromise between availability and performance.



Network Transaction

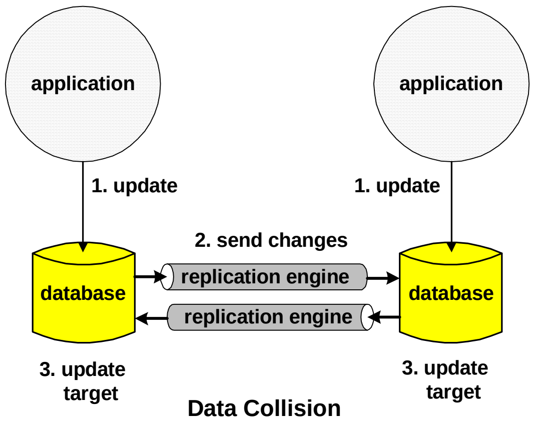
Asynchronous Replication

An asynchronous replication engine extracts changes made to its source database from some sort of a change queue (such as a change log or an audit trail) and sends these changes to a target database. This replication is done “under the covers” with respect to the application, which is therefore unaware of the replication activity. Consequently, there is no performance impact on the application.



Asynchronous Replication

However, there is a delay between the time that a change is applied to the source database and the time that it is subsequently applied to the target database. This delay is known as *replication latency* and typically ranges from hundreds of milliseconds to a few seconds. As a consequence, should a node fail, there will likely be transactions that, having been applied to the source database, are still in the replication pipeline at the time of the failure. These transactions never make it to the target database and, in effect, are lost.

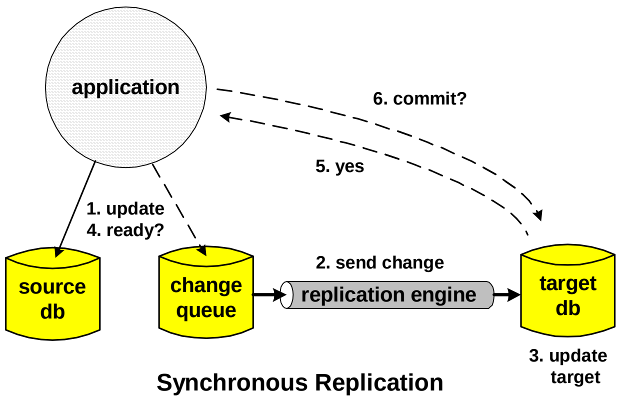


Data Collision

Another issue with asynchronous replication is data collisions. If a data item is updated on each of two database copies within the replication latency time, each will be replicated to the other database copy and will overwrite the original update to that copy. The two database copies are now different, and both are wrong. There are several techniques for avoiding data collisions or for detecting and resolving them. These techniques will be discussed in later articles.

Synchronous Replication

With synchronous replication, changes are replicated to the target database via asynchronous replication but are held there and used only to lock the affected data items. When the source node is ready to commit the transaction, it checks with all database copies to ensure that they have been able to obtain locks on all of the affected data items. It does this by sending a query behind the last update over the replication channel. If all database copies are ready, the source node instructs them to apply the updates and to release their locks. Otherwise, all copies are instructed to abort the transaction.[2]



Synchronous Replication

Synchronous replication, like network transactions, guarantees that all database copies will be in exact synchronism (as opposed to asynchronous replication, which keeps the database copies in near-synchronism because of replication latency). Thus, no transactions are lost as the result of a failure; and data collisions cannot occur.

In this case, the application is delayed as it waits for the commit to complete across the network. This is called *application latency*. However, this delay compares to network transaction delays, which must wait for each update as well as the commit to complete across the network. As a consequence, synchronous replication is generally more efficient if database copies are widely distributed or if transactions are large. Network transactions may be more efficient for collocated database copies and short transactions.

[2] See Holenstein, B. D., Holenstein, P. J., Strickler, G. E., *Collision Avoidance in Bidirectional Database Replication*, United States Patent 6,662,196; December 9, 2003.

Other Advantages of Active/Active

There are many other advantages that an active/active architecture brings:

- **Elimination of planned downtime:** A node can be upgraded by simply switching its users to other nodes. The node then can be brought down and its hardware, operating system, database, or applications upgraded and tested. At this point, the node can be returned to service by returning its users to it. This technique effectively eliminates planned downtime.

- **Data locality:** As compared to an active/backup configuration in which the backup system is not processing transactions, an active/active configuration can be distributed to provide data locality. Users can be connected to their nearest respective node, thus improving performance.

- **Use of all purchased capacity:** There is no idle backup system sitting around in an active/active system. Therefore, in a multinode active/active architecture, less capacity may need to be purchased. For instance, in a five-node configuration, if each node can carry 25% of the load, full capacity is provided even in the event of a node failure. However, only 125% of required capacity must be purchased rather than 200% for an active/backup system.

- **Online capacity expansion:** Capacity easily can be added by installing a new node and then switching some users to the new node.

- **Load balancing:** The load across the application network can be rebalanced by moving users from a heavily loaded node to lightly loaded nodes.

- **Risk-free failover testing:** Failover testing can be risk-free and not require any user downtime. In an active/backup system, it may take hours to fail over, during which time the users are denied service. The same downtime impact occurs when users are switched back to the primary node following failover testing. Furthermore, what if the backup node turns out to be nonoperational? In an active/active system, it is known that the other nodes are working; and failover takes seconds at most.

- **Lights-out operation:** In a multinode active/active application network, it may not be necessary to have every site staffed since should a node fail, the system continues operating anyway. Time to recover the failed node is less time-critical, especially if service can still be provided even after a second node failure.

RPO and RTO

When deciding upon which active/active architecture to use, one has to consider two important corporate goals, RPO and RTO.

- **RPO** is the *recovery point objective*. It specifies the amount of data that can be lost in the event of a failure. Tape backups can lose hours of data (as well as having hours of recovery time). Asynchronous replication can lose seconds of data, depending upon how fast the replication engine is (some have replication latencies measured in subsecond times). Synchronous replication and network transactions provide a zero RPO.

- **RTO** is the *recovery time objective*. It states the tolerance of the operation to an outage. Some applications can be down for minutes or hours without disastrous consequences.

For other applications, downtime of minutes or even seconds can be unacceptable. The primary advantage of active/active systems is that they provide essentially a zero RTO.

Active/Active Issues

There are several important active/active system issues that must be understood and resolved.

- How will user switching be handled? Can it be done automatically?

- Is there a chance of data collisions? If so, how will they be handled?

- Are lost transactions acceptable following a node failure? To what extent? Can they be recovered?

- Can the applications be run in an active/active environment? Do they require modification?

- What performance impact can be tolerated when implementing an active/active architecture?

- What additional cost can be tolerated? This may include hardware, software licenses, networking, people, and sites.

In Summary

There is always a fundamental compromise between availability, cost, and performance. Active/active technology can be tailored to a particular application to optimize this compromise.