the Availability Digest

www.availabilitydigest.com

# Defining Active/Active

December 2009

If we ask a dozen people what the term "active/active" means[1], the general consensus will probably be that it is a technique for building extremely reliable computing systems. In fact, this is true. Theory shows that active/active systems can achieve failure intervals measured in centuries. There are many examples of such systems that have been in production for a decade or more without a failure.

But when we probe deeper, we start to find the caveats – the limitations imposed by one technology or another to achieve such high reliability. How far can these caveats reach before we must conclude that a particular approach is really not suitable for our applications because of the approach's limitations?

In this article, we start a LinkedIn group to discuss just this issue. Please contribute to this work-in-progress by sharing your comments with us on our LinkedIn Continuous Availability Forum at http://www.linkedin.com/groupsDirectory?results=&sik=1260921605283&pplSearchOrigin=GLHD&keywords=continuous+availability+forum.
If this link doesn't work for you, search on Continuous Availability Forum under "Groups" in LinkedIn.

As we refine the definition based on your comments, we will be revising this article with the results of your discussions.

## A Reader's Dilemma

Rich Rosales of U.S. Bank, one of the Digest's readers, expressed this concern to us recently in some detail:[2]

> *The value of providing active/active has risen to the point of many software vendors claiming they support it - when in fact they do not. I don't believe that this is a direct attempt to mislead the consumer - rather a symptom of very confusing terminology.*
>
> *In my recent experience, every vendor I spoke with claimed they supported an active/active architecture. When I pressed them for details on how they would handle collisions, they all asked "Why would you let that occur?" - as if the reality of updating the same record (near) simultaneously on two geographically distributed systems was simply unthinkable - "Wouldn't*

[1] What is Active/Active?, *Availability Digest*; October 2006.
http://www.availabilitydigest.com/public_articles/0101/what_is_active-active.pdf

[2] See Rich's description of his experiences with implementing an active/active system, published in U.S. Bank Critiques Active/Active, *Availability Digest*; May 2009.
http://www.availabilitydigest.com/public_articles/0405/usbank.pdf

*that be fraud?" one vendor asked.*

*All of these vendors shared other similar profiles as well:*

- *No changes were made to the application to handle a dual-site design or to facilitate collision handling.*
- *All database synchronization, verification, detection, and conflict resolution was left to the database vendor.*
- *At least in the examples I have looked at, database updates were asynchronous.*

*What we have here is a common dual-site architecture being "used" for active/active - with no attempt at transaction homing or conflict resolution - they're simply letting collisions occur.*

*I think what we have here is an opportunity to help clarify the different levels of active/active. I'd like to offer the following as a starting point for this discussion:*

*Let's first assume that all examples given here describe a dual-site architecture with bidirectional replication - the same database record could be updated on either side of the active/active equation.*

*Level 0 - No collision detection or conflict resolution. If the communications link connecting the two sites is lost or brought down, both sites will store updates for later application when communications is re-established. In this scenario the likely result is that databases at both sites will be out-of-sync. Impact will depend on the nature of the application and how the databases are used.*

*Level 1 - No collision detection or conflict resolution. If the communications link connecting the two sites is lost or brought down, minimally one side of the sites has to be either brought down or put into read-only mode to reduce data collisions. NOTE: I believe Level 1 describes synchronous replication.*

*Level 2 - No collision detection or conflict resolution. To help avoid collisions, transactions are "homed" to one site or the other - the goal being unidirectional updates at a record level. If the communications link connecting the two sites is lost or brought down, routing of transactions can no longer occur. The impact of this design can be mitigated through network design - attempting to keep traffic localized to the respective database. As enormous as this challenge may seem, it has been a staple strategy in many dual-site architectures. Very few of the true advantages of active/active can be realized using this technique. Arguably, this architecture is not sustainable.*

*Level 3 - Customized collision detection and conflict resolution. Although all practical methods of collision avoidance are put into use, collisions are acknowledged as a fact of life in a busy active/active architecture. Collisions are anticipated, and when they do occur they are detected, categorized, and resolved using logic specifically designed for the type of collision situation. Loss of communication between sites causes no disruption in processing, and the increase in synchronization latency is anticipated and handled as a part of normal processing.*

*I think this is a start - it certainly has helped us to understand the challenges of a fully active/active architecture - and the decisions and architectural considerations that must be made to achieve more than simply a dual-site architecture.*

*We need to come up with new terminology - to prevent vendors from flooding the market with*

*active/active (level 0) architectures, and provide a means by which those vendors that can and do provide higher levels of availability can distinguish themselves.*

## The Gold Standard – the "Perfect" Active/Active System

We agree whole-heartedly with Rich. Those of you who have been involved intimately with active/active systems understand the many shades of the term.

To get a handle on categorizing active/active systems, let us first define the "perfect" active/active system. If we can agree on such a reference definition, then we can measure any architecture against that definition and can decide where the architecture complies and where if falls short. This will give us a level playing field when comparing active/active technologies.

We submit that a "perfect" active/active system will have the following attribute:

*An active/active system comprises multiple geographically-distributed processing nodes using geographically-distributed consistent copies of the application database, such that the application network survives any single point of failure with no data loss. Furthermore, a transaction can be directed to any processing node in the application network, and the recovery time from the failure of a processing node or database copy is short enough that users are not aware of the fault.*

A "perfect" active/active system meeting this attribute has many desirable characteristics:

(1) Disaster Tolerance - The system is disaster tolerant since it comprises fully redundant, geographically-distributed components (processing nodes and database copies) and will survive the failure of any one component (or more if properly architected).

(2) No Data Loss - Should the system suffer a fault, no data is lost. All completed transactions or updates survive in all database copies.

(3) No Idle Nodes - All processing nodes and all database copies are available to equally share the application load, since reads and updates can be routed to any processing node.

(4) Locality – Database requests can be directed to the closest processing node to maximize performance. Processing nodes can use the closest database copy.

(5) Load Balancing -The system can be easily load balanced by redirecting read and update activity.

(6) Scalability - The system can be easily scaled since processing nodes and database copies can be added and the load rebalanced.

(7) No Unplanned Downtime - Unplanned downtime can be eliminated since transactions can be routed around a node failure with no impact on users.

(8) No Planned Downtime - Planned downtime can be eliminated by redirecting traffic prior to the scheduled outage, downing a node, upgrading it, and returning it to service while the other nodes handle the application workload.

We understand that in today's world, no technology achieves this goal.[3] However, different active/active technologies achieve different elements of this attribute, and users can select a technology that best suits their needs.

## Categorizing Active/Active Architectures

The first thing to recognize is that there is a natural dichotomy in active/active architectures – asynchronous systems and synchronous systems.

- In an *asynchronous active/active system*, the database copies in the application network are updated asynchronously. There are two primary methods for implementing an asynchronous active/active system:

    - with *data replication*, in which changes made to one node are subsequently replicated to the other nodes, and

    - with *transaction replication*, in which changes are routed to all nodes, which execute them independently.

    In either event, database updates are made at different times to different database copies. This means that nearly-simultaneous (typically within a second) updates to a data item by different processing nodes might be made to different data item copies in different order, leaving them in different states. This is called a *data collision* and can lead to distributed database corruption.

    Furthermore, if a data replication engine is used to keep the databases in synchronism, there can be data loss if a node fails, as some of the data committed at the failed node may not have been replicated to the other nodes yet. This is not true of transaction replication if transactions are routed by the network or client and not by forwarding from the receiving node.

- In a *synchronous active/active system*, updates are made to the database copies synchronously. An update must be successfully applied to all database copies in the application network before the update is complete. Therefore, data collisions and data loss are eliminated, but the distance separating the nodes is typically limited (generally to a few kilometers) by performance considerations. Consequently, disaster tolerance may be compromised.[4]

### *Split-Brain Mode*

A common problem with active/active systems occurs if connectivity between the systems is lost. In this case, changes made by one node cannot be replicated to the other node. If the system is asynchronous and both nodes are allowed to continue in operation, then their databases will diverge over time and must be resynchronized when the network is restored. Data collisions will have to be detected and resolved. This is called *split-brain mode*.

If the system is synchronous, then it must first convert to asynchronous mode or else no transaction will be able to complete – the system will be down. It is now operating in split-brain mode as described above.

[3] Though a technology called "coordinated commits," to be introduced perhaps in the next year, may substantially close this gap. See HP's NonStop Synchronous Gateway, *Availability Digest*; June 2009. http://www.availabilitydigest.com/public_articles/0406/hp_srg.pdf

[4] New synchronous algorithms being developed may lessen this problem. See Footnote 3.

If split-brain is allowed and if collisions can occur, collisions must be resolved when the nodes are reconnected. If split-brain mode is not acceptable, then separated nodes must be shut down until connectivity is restored.

Let us look independently at asynchronous and synchronous architectures.

***Asynchronous Active/Active Architectures***

An asynchronous active/active system may be implemented in at least one of two ways. One way is to allow a transaction to be processed by any node and to replicate the updates created by the transaction to the other nodes via an asynchronous replication engine. We call this *data replication.* The other way is to send the entire transaction to all nodes, which will independently process the transaction. We call this *transaction* replication. Both have their advantages and disadvantages.

Data Replication

Asynchronous active/active architectures are generally implemented with asynchronous replication engines such as those from HP, IBM, Goldengate, Gravic, Network Technologies, Oracle, Double-Take, and Continuent. In all cases, changes made to one database copy (which we will call the *source database*) are placed in a change queue of some sort and then sent after-the-fact over the replication network to the other (*target*) database copies in the application network.

The result is that all database copies are kept in synchronism in "near-real time." By near-real time, we mean that changes made to one database copy are reflected in the other database copies after a short (typically subsecond) delay. We call this delay the *replication latency*. Replication latency includes not only the processing and intermediate storage delays in the replication engine but also the communication channel delay as changes propagate across the replication network (the *communication latency*).

Replication latency has two undesirable consequences:

(1) If changes are being replicated from one node to another, and should a processing node fail, all the changes that are currently in its replication pipeline will be lost.

(2) It is possible for two nodes to update the same data item at approximately the same time (within the replication latency interval). Each change will be replicated to the other system and will overwrite the original change there. As a consequence, both database copies are different, and both are wrong.

To handle the problem of data collisions, many asynchronous replication engines incorporate data collision and resolution facilities. Once detected, a collision is resolved by business rules either built into the replication engine or added via user exits.

Data loss and uncorrected data collisions result in a corrupted distributed database.

Thus, there are two important aspects to understand with respect to an asynchronous data replication engine:

(1) What is its replication latency? This will determine the expected data loss following a node failure and the data collision rate.

(2) What are its capabilities to identify and resolve data collisions?

The first aspect is simply a number (though it is system-dependent and may not be easily determined). It is the second aspect that leads to a variety of active/active architectures that must be clearly understood. It is this aspect that leads to the active/active levels to which Rich Rosales refers as he suggests different levels of active/active.

Transaction Replication

With transaction replication, there is no data loss if transactions are routed to the different nodes by the client or by the network. If transactions are routed by the nodes, then data loss may occur if the transaction is processed by a node before it is routed.

There is also a replication latency interval with transaction replication, but it is in the time interval during which different nodes process the transaction. The longer this interval, the more likely it is that there will be data collisions.

***Categorization of Asynchronous Active/Active Systems***

Following Rich's thoughts, we suggest the following categorizations of active/active architectures. The category names use the following nomenclature:

| Nomenclature | | | Description |
|---|---|---|---|
| A | | | asynchronous replication |
| S | | | synchronous replication |
| | | | |
| | A | | amorphous database (update anywhere) |
| | U | | partitioned database (update specific) |
| | T | | transaction replication (update all) |
| | | | |
| | | + | data collision and resolution supported |
| | | - | data collisions not supported |

This leads to six asynchronous active/active architectures:

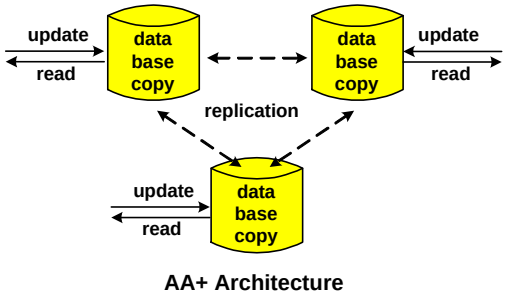
- AA+: Amorphous database with collision support.
- AA-: Amorphous database without collision support.
- AP+: Partitioned database with collision support.
- AP-: Partitioned database without collision support.
- AT+: Transaction replication with collision support.
- AT-: Transaction replication without collision support.

In the descriptions of the following architectures, the active/active attributes of the approach are described followed by the characteristics of the replication engine required to support the architecture.

## Architecture AA+: Amorphous Database with Collision Support

*Architecture*

The database is fully distributed and appears amorphous to the application. Any transaction, update, or read can be applied to any database in the application network and is typically directed to the closest processing node. Data collisions are anticipated, and they are either avoided[5] or are detected and resolved by facilities built into the replication engine.[6] Should a node fail, transactions are automatically routed to surviving nodes.



**AA+ Architecture**

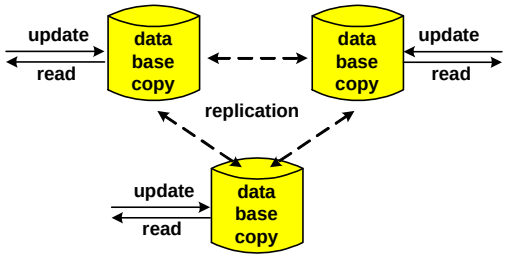
*Replication Engine*

This architecture requires bidirectional replication.[7] All database copies must be open for full read/write access. The architecture also requires that full data collision detection and resolution be supported by the replication engine.

## Architecture AA-: Amorphous Database without Collision Support

*Architecture*

This architecture assumes that the application is such that collisions will not occur or that they may be ignored. Any transaction, update, or read can be applied to any database in the application network and is typically directed to the closest node. Data collisions are not anticipated, nor are they handled. Should a node fail, transactions are automatically routed to surviving nodes.



**AA- Architecture**

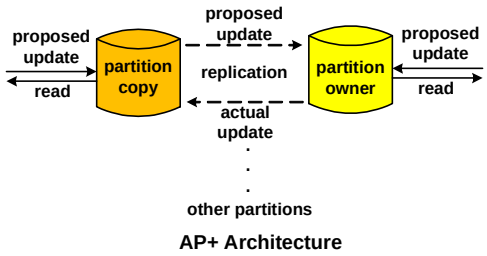
*Replication Engine*

This architecture requires bidirectional replication. All database copies must be open for full read/write access. Data collision detection and resolution need not be supported by the replication engine.

[5] Such as by relative replication, in which operations are replicated rather than row images.
[6] Chapter 4, Active/Active Topologies, *Breaking the Availability Barrier II: Achieving Century Uptimes with Active/Active Systems*
[7] Bidirectional replication implies that the replication engine prevents ping-ponging, or the return of a replicated data item to its source.

## Architecture AP+: Partitioned Database with Collision Support

*Architecture*



AP+ Architecture

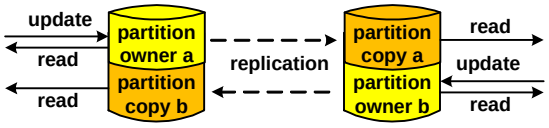
The database is partitioned, and each partition is owned by one node. Only the owning node can update its partition, and it resolves any data collisions. Reads and updates may be directed to any node, thus preserving locality. Updates made by non-owning nodes to their partitions are replicated to the owning node, which will resolve any data collisions by accepting one update and rejecting the other. The final updates made to the owning node's partition are then replicated back to the other nodes, including the nodes that originated the updates, thus maintaining proper synchronization among the database copies. Provision must be made to detect the failure of a node. Should an owning node fail, another node must be promoted to become the owning node for that partition.

*Replication Engine*

This architecture requires bidirectional. All database partitions must be open for read/write access on all nodes to support fast failover when a non-owning node must be promoted to an owning node. Furthermore, full data collision detection and resolution must be supported by the replication engine.

## Architecture AP-: Partitioned Database without Collision Support

*Architecture*



AP- Architecture

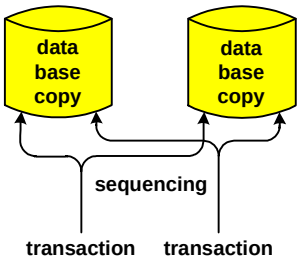
The database is partitioned, and each partition is owned by one node. Only the owning node can update its partition, and all updates for a partition must be routed to the owning node. However, all nodes can read all partitions. Data collisions cannot occur. Provision must be made to detect the failure of a node. Should a node fail, its partition ownership must be transferred to another node.

*Replication Engine*

This architecture requires unidirectional replication from each partition owner to all of the other nodes. All database partitions must be open for read/write access on all nodes to support fast failover. Data collision detection and resolution are not required.

## Architecture AT+: Transaction Replication with Collision Support

*Architecture*



AT+ Architecture

All transactions are sent simultaneously to every node in the application network. This may be done by the client, by network routing, or by node forwarding. Each node executes the transactions or updates independently of the other nodes. However, there is a sequencing mechanism to ensure that all nodes execute transactions in the same order. Therefore, data collisions cannot occur and all databases will be identical.

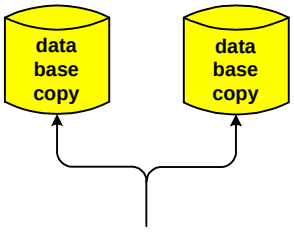
*Replication Engine*

The replication engine required by this architecture submits transactions to all nodes in the application network. It supplies the coordination necessary to control transaction ordering among the nodes.

Architecture AT-: Transaction Replication without Collision Support

*Architecture*

This architecture assumes that the application database is insensitive to transaction execution order. All transactions are sent simultaneously to every node in the application network by each client application. Each node executes the transactions or updates independently of the other nodes. Updates submitted by different clients may be executed in different order by the nodes. Therefore, the databases may diverge over time due to data collisions unless the application is insensitive to transaction order.




**AT- Architecture**

*Replication Engine*

The replication engine required by this architecture submits transactions simultaneously to all nodes in the application network. There is no coordination to control transaction ordering among the nodes.

Master/Slave Architectures

There are other active/active architectures known as "master/slave." In these systems, one node is the master. It performs all updates and replicates these updates to the slave systems. These architectures are, in effect, a partitioned architecture with a single partition and are covered by the AP+ and AP- architectures.

Asynchronous Summary

These levels are summarized in the following table, which shows the support for the desirable active/active characteristics.

| Archi-tecture | Fast Failover | Disaster Tolerance | Data Loss Avoided | Data Collision Support | Balance Updates | Balance Reads | Scalable | Locality (route tx anywhere) | Split-Brain Avoided |
|---|---|---|---|---|---|---|---|---|---|
| AA+ | Y | Y | N | Y | Y | Y | Y | Y | N |
| AA- | Y | Y | N | N | Y | Y | Y | Y | N |
| AP+ | Y | Y | N | Y | Y* | Y | Y* | Y | Y |
| AP- | Y | Y | N | N | Y* | Y | Y* | N | Y |
| AT+ | Y | Y | Y | Y | N | Y | N | N | Y** |
| AT- | Y | Y | Y | N | N | Y | N | N | N*** |

* only by repartitioning the database.
** only if transaction routing is done by the client or network, not by node forwarding.
*** AT- is always in split-brain mode

**Asynchronous Active/Active Characteristics**

The architectures have different characteristics:

- AA-, AP-, and AT- cannot be used if data collisions are a concern.
- AA+, AA-, and AP+ allow a transaction to be routed to any node.
- AP- requires that transactions be routed to a specific node. This may compromise the performance advantage of local processing.
- AT+ and AT- require that transactions be routed to all nodes.
- All levels support read load balancing.
- AA+ and AA- support update load balancing.
- AP+ and AP- require that the database be repartitioned to balance update load.
- AT+ and AT- cannot be balanced for update load.
- AA+ and AA- databases will diverge if replication is lost (split-brain mode).
- AP+, AP-, and AT+ prevent split-brain database divergence.
- AT- is always in split-brain mode.

In general, an asynchronous replication engine can be categorized by the active/active architecture that it supports (its level) and its inherent replication latency (exclusive of communication latency). For instance, a particular replication engine might be a 100 millisecond AP+ asynchronous engine.

Note that active/active architecture AA+ has all of the characteristics of our definition of an ideal active/active system except for the potential of data loss following a node failure. An AA- architecture also meets this test if data collisions cannot happen or can be ignored.

***Synchronous Active/Active Architectures***

Synchronous active/active architectures follow a different path from asynchronous architectures. Using today's available technology, most synchronous active/active systems are entire systems available from a system vendor. OpenVMS split-site clusters, IBM Parallel Sysplex, and Stratus Avance are examples of these.[8]

This may change when coordinated commit synchronous replication engines become available.[3] These engines use asynchronous replication to propagate updates and synchronize transactions only at commit time.

Synchronous active/active architectures generally meet all of the characteristics of our ideal active/active architecture except for limits on geographical separation. Since a transaction or an update must be completed on all copies of the database across the application network, transaction response times are increased. Most vendors of synchronous replication systems limit their support of geographical dispersion to some specified distance.

Synchronous replication faces a unique problem should the nodes lose connectivity or if one node becomes so slow that it slows down all transactions. In this case, the system must switch to asynchronous replication so that transaction processing can continue. This leads to split-brain mode, and if this is not acceptable, the disconnected or malfunctioning node must be shut down.

Recovery from a node failure is far more complex since the failed node's database must be resynchronized with the active database. During this process, the failed node will be processing a

[8] OpenVMS Active/Active Split-Site Clusters, *Availability Digest*; June 2008.
http://www.availabilitydigest.com/public_articles/0306/openvms.pdf
Parallel Sysplex – Fault Tolerance from IBM, *Availability Digest*; April 2008.
http://www.availabilitydigest.com/public_articles/0304/ibm_sysplex.pdf
Stratus' Avance Brings Availability to the Edge, *Availability Digest*; February 2009.
http://www.availabilitydigest.com/public_articles/0402/avance.pdf

mix of asynchronous and synchronous transactions as it transitions to a fully synchronous state. This transition must be carefully managed.

Each synchronous technique varies in how it maintains database copies in synchronism and how it resynchronizes following a node or network fault. However, the result is the same. The active/active attribute is met in full except for the limits on disaster tolerance. There is no data loss. There are no data collisions. Failover is fast, and system upgrades can be rolled through the application network one node at a time to eliminate planned downtime. Read and update activity can be uniformly distributed across the nodes (of course, every database copy must receive all resulting changes), and the system is scalable by adding nodes.

Therefore, a convenient means of classification of synchronous replication systems is simply to state the distance limitation. For instance, a system may be a 20-kilometer synchronous replication system.

## Summary

This categorization of active/active technologies should be considered a work-in-progress. To those looking to implement active/active systems, a categorization that achieves substantial consensus is important to be able to weed out vendor offerings (or in-house implementation strategies) that will not meet the application's needs.

Please contribute to this important subject by sharing your comments with us on our LinkedIn Continuous Availability Forum at
http://www.linkedin.com/groupsDirectory?results=&sik=1260921605283&pplSearchOrigin=GLHD&keywords=continuous+availability+forum.
If this link doesn't work for you, search on Continuous Availability Forum under "Groups" in LinkedIn.