the Availability Digest

Active/Active Systems – A Taxonomy

September 2008

Defining an Active/Active System

In our publications, we have defined active/active systems in terms of their architecture:

> Active/Active Architectural Rule: *An active/active system is a network of independent processing nodes, each having access to a common replicated database. All nodes can cooperate in a common application, and users can be serviced by multiple nodes.*

As we have seen in the Digest's articles and many product reviews, several ways exist to architect active/active systems. But focusing on their architecture obfuscates what active/active systems really are about, and that is the benefits they provide. An alternative definition of an active/active system is therefore a list of the attributes required of such systems.

Taking this focus, we submit that an active/active system has the following four attributes:

- ***Availability*:** An active/active system has a mean time between failure (mtbf) measured in centuries. In practical terms, this means an availability of at least six 9s - an average of 30 seconds per year of downtime. Given an average repair time of one hour, six 9s translates to an mtbf of 120 years. Six 9s also implies that there is no planned downtime required for maintenance and upgrade activities.

- ***Survivability*:** An active/active system can withstand a disaster that takes down a processing node. Survivability implies that redundant nodes be far enough apart so that the probability of losing two nodes to a common disaster is extremely unlikely.

- ***Scalability*:** An active/active system is easily scalable by simply adding additional computing resources. These resources can be put into service without affecting ongoing operations.

- ***Consistency*:** Two identical operations issued simultaneously anywhere in the application network will provide the same result.

Like the ACID properties of transactions, we will call the above attributes the ASSC properties of active/active systems.

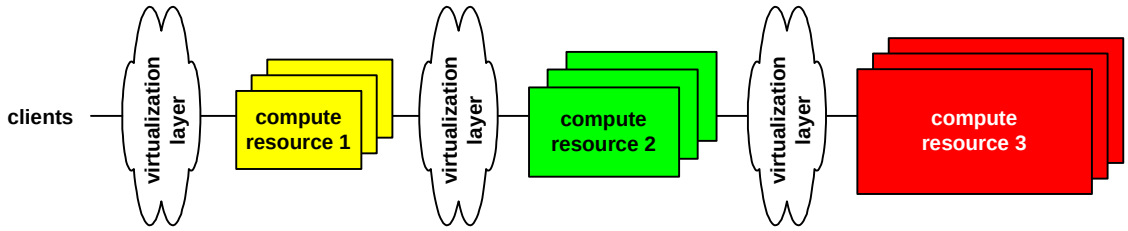
We submit that any system architecture having these four properties qualifies to be called an active/active system:

> Active/Active Attribute Rule: *An active/active system must be available, survivable, scalable, and consistent. It is available if it has an mtbf measured in centuries. It is*

survivable if it can withstand an entire processing-node outage. It is scalable if computing resources can be added or deleted with no impact on ongoing operations. It is consistent if it always behaves the same.

It is possible to build active/active systems in many ways. However, the systems which we have studied all fall within a particular class of virtualized architectures. These architectures are described by the following Active/Active Virtualization Rule, which is a dependent extension of the Active/Active Architectural Rule stated above. The Active/Active Virtualization Rule will form the basis of our taxonomy:

Active/Active Virtualized Rule: A*n active/active system is a collection of redundant, geographically-distributed, context-free compute resources with no single point of failure. Each redundant compute resource is a virtualized pool of computing elements that appears externally as a single resource. Within a virtualized pool, computing elements can be added or removed transparently to the users.*[1]



An Active/Active System

This definition is consistent with our earlier definition given above, but represents a specific architecture that satisfies both the Active/Active Architectural Rule and the Active/Active Attribute Rule. There are several points to note about the Active/Active Architecture Rule:

- *Redundant compute resources:* Each compute resource must be redundant. Should one compute resource fail, there must be another that can take over its operation. The time required to fail over to an alternate compute resource must be such as to meet the availability requirement.

- *Virtualized pool:* One way to achieve rapid failover is for each redundant compute resource to be a pool of compute resources all actively participating in the application. The pool of compute resources is accessed via a *virtualization layer* that provides a single compute-resource view of the compute-resource pool. Via the virtualization layer, any request or transaction that was being processed by a failed or removed compute resource is resubmitted to a surviving compute resource. Thus, failover time becomes resubmission time.

- *Context-free:* In order for a request to be handled by any one of the compute resources in a compute-resource pool, the compute resources must not have to store context from previous requests. Request context, if any, must be contained within each request message; or it must be stored in a common database accessible by each of the compute resources in the pool.

- *No single point of failure:* Since virtualized compute-resource pools inherently have no single point of failure, this requirement generally applies to the virtualization layer.

[1] The term "virtualization" has multiple uses. For one, it is the logical representation of an underlying structure, such as in storage virtualization. For another, it is providing multiple personalities to an underlying structure, such as multiple virtual machines on a physical server. We use the first meaning of virtualization in this article.

Incoming requests must be routed to a compute resource via redundant means, such as redundant LANs, redundant routers, or redundant networks.

The Active/Active Attribute Rule is a necessary and sufficient rule. It is necessary that the attributes of availability, survivability, scalability, and consistency must be met if the system is to be considered an active/active system. If these attributes are met, then that is sufficient for a system to be an active/active system.

The Active/Active Virtualization Rule is a sufficient rule. If a system is implemented according to this architecture, then the system can be considered an active/active system. However, it is not necessary that the system be implemented according to the Active/Active Virtualization Rule. It may be that other implementations will also satisfy the Active/Active Attribute Rule. For purposes of our taxonomy, we will use the Active/Active Virtualization Rule.

Context versus State

We have noted that compute resources should be *context-free*. They need not have knowledge of previous results for a given transaction.

However, this does not eliminate the requirement that a compute resource may have to remember global *application state*. The current state of the application may have to be available to members of one or more compute-resource pools.

The recording and dissemination of application state is typically the responsibility of a special class of compute-resource pool, the database pool. The database pool is *stateful* in that it remembers the state of the application. Typically, the other virtualized pools are *stateless* since they remember no application state. Rather, they obtain application state when needed from the database pool.

Though the compute-resources in a database pool are stateful, they must nevertheless comply with the context-free requirement. That is, a database request can be routed to any member of the database pool for execution. No matter which member receives the request, its actions on the database must be the same as any other member that might have received that request.

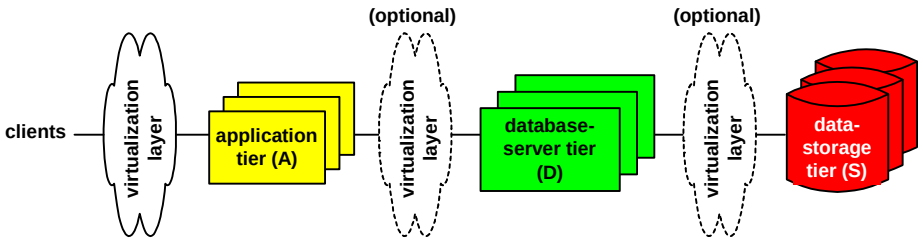
This means that the contents of each database held by a database pool member must be identical to the databases held by the other members of that pool. The synchronization of these database copies is one of the challenges of active/active systems, for which there are several solutions, as we shall see in the taxonomy.

The Three-Tier Active/Active Architecture

For purposes of an active/active taxonomy, we view an active/active system as comprising three tiers – an application tier, a database-server tier, and a storage tier.[2]

- The *application tier* provides all of the application-specific functions of the system.
- The *database-server tier* manages the database that contains the application state. This may be a relational database or a file system.
- The *storage tier* comprises the data storage for the application state. This tier consists typically of direct-attached storage (DAS), network-attached storage (NAS), or storage area networks (SAN).

[2] This is at variance with the usual definition of a three-tier architecture, which comprises a presentation tier, an application tier, and a database tier. We fold the presentation tier into the application tier and separate the database tier into two tiers – the database server and the storage devices.



Active/Active Three-Tier Architecture

One or more of these tiers can be virtualized to insulate them from the other tiers. However, the application tier is always virtualized. The application-virtualization layer ensures that a client request can be routed to any application server.

Virtualization techniques for the application layer can be implemented in many ways. For instance, the client may be intelligent enough to detect a failure and resubmit its request to another application server. Alternatively, intelligent routers may redirect requests. In any event, as noted above, this virtualization layer must be redundant so as to be fault-tolerant.

The application-tier virtualization layer may also perform other functions, such as load balancing or the partitioning of requests among the application servers according to data content.

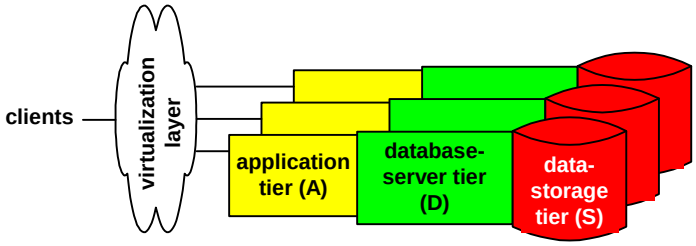
An Active/Active Architectural Taxonomy

The above definition of the active/active architecture results in four configurations. Let us designate these four configurations by identifying each tier by a letter:

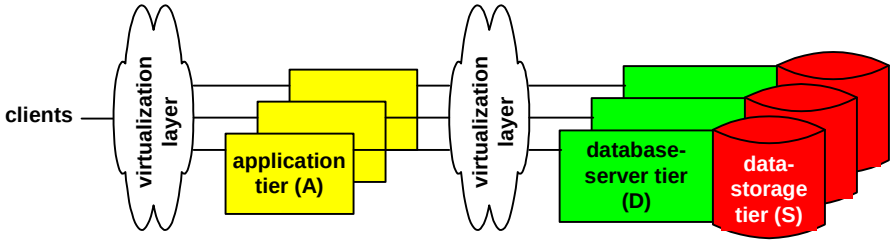
- *A* is the application tier.
- *D i*s the database-server tier.
- S is the storage tier.

We designate each configuration by listing the letters of the tiers that are virtualized:

- ***Configuration A***: Only the application tier is virtualized. Each compute element in the pool is a complete server comprising the application logic, a database server, and a private database. A client request can be passed to any compute element in the pool.

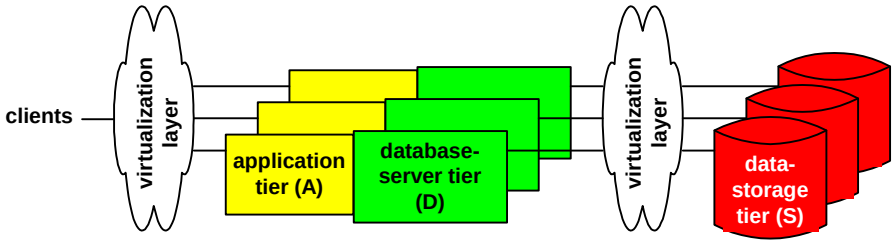


- ***Configuration AD***: In addition to the application tier, the database-server tier is virtualized. Any data request can be passed from the application tier to any database server, which will execute it against its private copy of the application database.



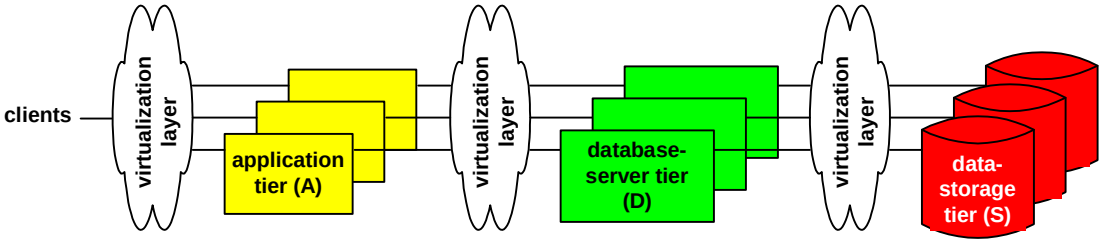
Active/Active AD Architecture

- ***Configuration AS***: In addition to the application tier, the data-storage tier is virtualized. A client request is sent to an application server that has its own database server. The database server can execute its database requests on any of the virtualized data-storage compute elements.



Active/Active AS Architecture

- ***Configuration ADS***: All tiers are virtualized. Any request can be sent to any application compute element. An application compute element can send a database request to any database server. A database server can request reads and updates from any data-storage compute element.



Active/Active ADS Architecture

An Active/Active Database Synchronization Taxonomy

The architectural taxonomy suggested above does not consider how the databases are synchronized. The only way that the data-storage tier can be virtualized is if all of the data storage devices are maintained in the same state.

In use today are two primary techniques for achieving database synchronization. They include:

- ***Asynchronous Replication (A)*:** As changes are made to a data-storage element, those changes are replicated in the background asynchronously to the other data-storage elements, independently of the application.[3]

- ***Synchronous Replication (S)*:** A change to the database is executed simultaneously on all data-storage elements. Either the change is applied successfully to all data-storage elements, or it is not applied to any data-storage element.[4]

Each of these techniques can replicate either individual operations or transactions:

- ***Operation Replication (O)*:** The replication scope is a single change command (insert, update, delete). Several rows or fields may be affected by a single operation - for instance, a single SQL statement.

- ***Transaction Replication (T)*:** The replication scope is all of the change operations contained within the scope of a single transaction.

Thus, there are four replication techniques in the taxonomy:

- ***A/O*:** Asynchronous operation replication.
- ***A/T*:** Asynchronous transaction replication.
- ***S/O*:** Synchronous operation replication.
- ***S/T*:** Synchronous transaction replication.

Interestingly, none of these replication techniques are totally compliant with the Active/Active Attribute Rule. They each suffer some deficiency that must be taken into account when an active/active architecture is chosen.

- Except for the AD architecture, consistency is not guaranteed by any of the replication techniques. That is, there can be race conditions between reads and replicated updates so that, at any given instant in time, a read processed by one data-storage element may give a different result than a simultaneous equivalent read processed by another data-storage element. Whether a read retrieves data just before a change is made or just after a change is made depends upon the temporal relation of the read and change operations.

  This problem is aggravated by asynchronous replication because there can be a significant delay between the time that an update is applied to a source data-storage element and the time that it is applied to the target data-storage elements.

  The AD architecture can provide consistency because the virtualized database server behaves as a single database server and can properly order all database operations.

  The lack of consistency does not seem to be a practical problem in practice today because the window of confusion is small (milliseconds, typically).

- Asynchronous replication does not quite meet the availability attribute because data changes in the replication pipeline may be lost upon the failure of a database server.

- Synchronous replication may not meet the survivability attribute because it imposes distance limitations. Since the application must wait for an operation or a transaction to complete on all data-storage elements, the distance separating the elements of the

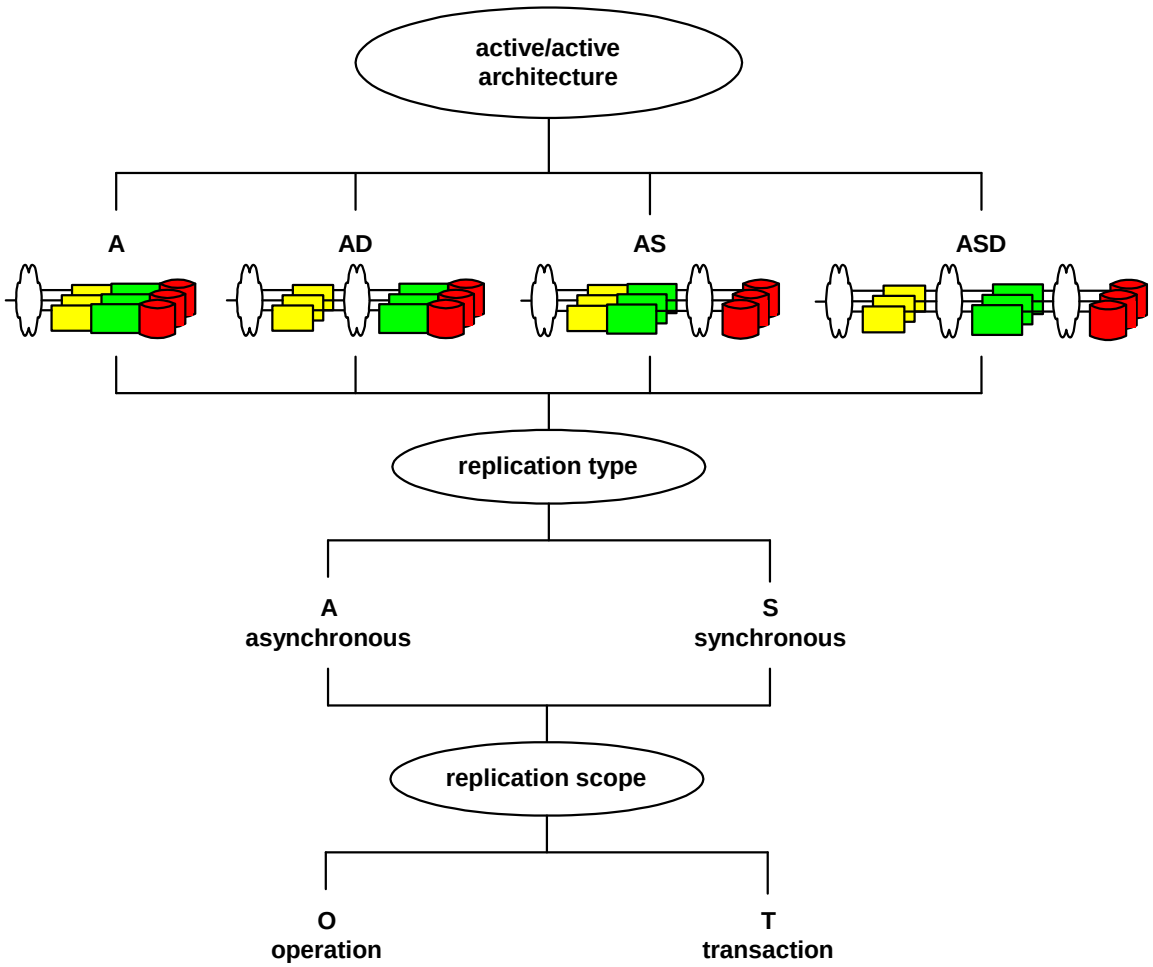
[3] Asynchronous Replication Engines, *Availability Digest*; November, 2006.
[4] Synchronous Replication, *Availability Digest*; December, 2006.

system is constrained due to performance considerations. A distance limitation may affect the disaster tolerance of the system.

An Active/Active Taxonomy

The above definitions lead to a three-level taxonomy. At one level is the system architecture. At the other levels are the replication mechanism and the replication scope.

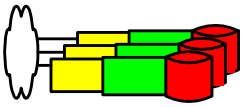
For instance, an active/active system in which the database-server tier is virtualized and in which operations are synchronously replicated is an AD/S/O system.



Active/Active Taxonomy

Examples of Active/Active Systems

Virtualized Processing Nodes (Configuration A)

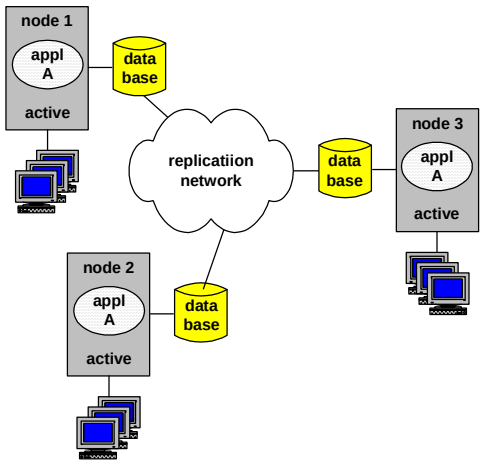


In most of our *Availability Digest* articles, we have focused on active/active systems that comprise two or more self-contained processing nodes that all cooperate in a common application. A transaction can be routed to any node. It will be processed by that node, and changes made to its application database copy will be replicated to the other databases in the application network.

In this case, replication is done via independent replication engines that read source database changes from a Change Queue and then send them to the remote target databases. Both asynchronous and synchronous replication engines are in use and are available from several vendors.

When asynchronous replication is used, replication is typically done at the transaction level. Synchronous replication may be done either at the operation level or at the transaction level. Synchronous replication at the operation level is often called *dual writes* or *network transactions*. Each operation is replicated individually, and the application waits for each operation to finish before it proceeds to the next operation.

At the transaction level, operations are replicated asynchronously. It is only at commit time that the commits are applied synchronously. Thus, the application is delayed only by the time that it takes for the transaction to be committed across the network rather than being delayed by each individual operation. This technique is called *coordinated commits*.

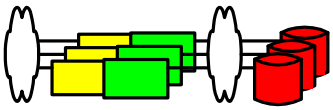


Network transactions may be more efficient over short distances and for small transactions. Coordinated commits are more efficient for long distances as well as for large transactions.

Thus, the architecture of virtualized processing nodes may be classified as A/A/O, A/A/T, A/S/O, and A/S/T.

We have published many examples of such systems as Case Studies in the *Availability Digest.* Most of them use HP NonStop servers for the nodes in the application network.[5]

OpenVMS Clusters (Configuration AS)



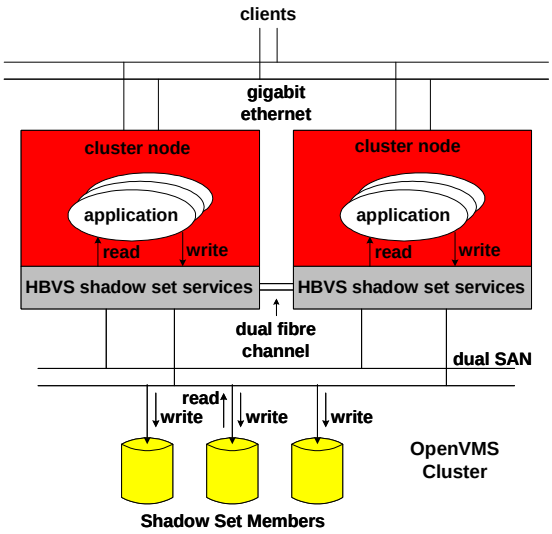
HP's OpenVMS Clusters are arguably the original active/active system.[6] Introduced in 1984 by Digital Equipment Corporation, an OpenVMS Cluster supports up to 96 processing nodes and up to 500 data-storage shadow sets. A processing node is a stand-alone server containing the application programs and a file server. A data-storage shadow set comprises two or more disk systems (up to six) and is virtualized to look like a single disk system to the processing nodes. The processing nodes can be geographically distributed as can the members of a shadow set.

The data-storage virtualization layer uses Host-Based Volume Shadowing (HBVS) to provide distributed lock management and a distributed cache mechanism for virtualization of the members of a shadow set. File-update operations are sent synchronously to all members of a

[5] Bank Verlag – the Active/Active Pioneer, *Availability Digest*; December, 2006.
Telecom Italia's Active/Active Mobile Services, *Availability Digest*; March, 2007.
BANKSERV Goes Active/Active, *Availability Digest*; April, 2007.
HP's OpenCall INS Goes Active/Active, *Availability Digest*; June, 2007.
Major Bank Uses Active/Active to Avoid Hurricanes, *Availability Digest*; October, 2007.
Asymmetric Active/Active at Banco de Credito, *Availability Digest*; November, 2007.
Payment Authorization – A Journey from DR to Active/Active, *Availability Digest*; December, 2007.
Active/Active Payment Processing at Swedbank, *Availability Digest*; January, 2008.

[6] OpenVMS Active/Active Split-Site Clusters, *Availability Digest*; June, 2008.

shadow set and are completed when all shadow-set members have reported completion. Reads are sent to the shadow-set member that can respond the fastest.

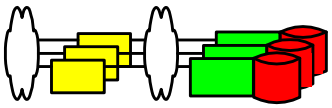


Therefore, an OpenVMS Cluster can be classified as an AS/S/O active/active system. The active/active availability attribute is met since the failure of any processing node means that outstanding requests are resubmitted. The failure of any shadow-set member simply results in that member being removed from the data-storage pool.

The system is survivable since processing nodes and shadow-set members can be geographically distributed. However, survivability is compromised by the synchronous replication of updates. To avoid severe performance impacts on applications, the distance of separation is limited. HP supports distance separations of up to 500 miles.

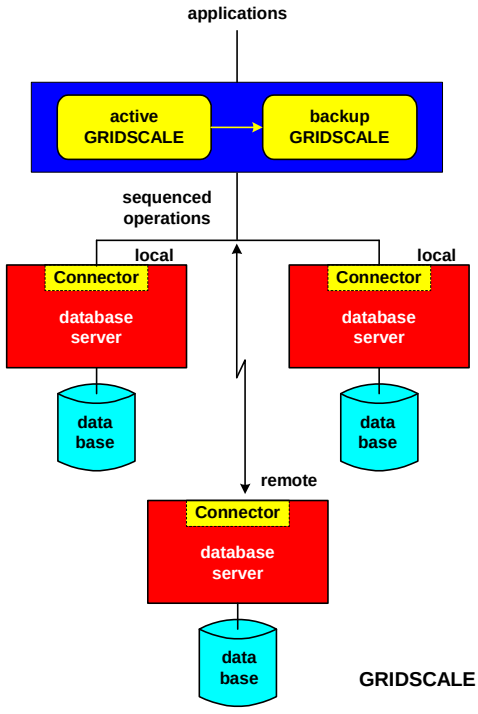
In *write seldom, read often* applications, an OpenVMS Cluster is scalable up its supported limits of 96 processing nodes and 500 shadow sets. It provides consistency for updates because of the distributed locking mechanism. Read consistency is provided except for reads that reach the shadow-set members within the confusion window when an update to the records to be read is about to be made.

GRIDSCALE (Configuration AD)



GRIDSCALE, from xkoto,[7] is a virtualized database. It does not provide an application tier, but any virtualized application tier can be used.

Database requests are received from the application tier by the GRIDSCALE database virtualization layer. This virtualization layer is implemented as a stand-alone server that assigns a sequence number to each database operation. It then sends each operation to the database servers making up the GRIDSCALE system. The database servers are guaranteed to execute each operation in exactly the same sequence based on the sequence number assigned by the GRIDSCALE server. This applies not only to updates, but also to read operations and transaction semantics.



The database servers may be geographically distributed with no distance limitation.

GRIDSCALE's replication algorithm is interesting because it combines the best of both asynchronous and synchronous replication. It responds to the application with a completion status as soon as it receives the first successful completion response from any of the database servers (Reads are only sent to

[7] GRIDSCALE – A Virtualized Distributed Database, *Availability Digest*; July, 2008.

one database server – the one that will provide the fastest response.) Since at least one of the database servers is bound to be local to the GRIDSCALE server, performance is equivalent to that of a standalone transaction-processing system except for a small overhead imposed by the GRIDSCALE server. Applications do not have to wait for update operations to complete on all database servers in the pool.

However, replication is synchronous in that every database server is guaranteed to apply all updates in exactly the same sequence.

The GRIDSCALE server is a single point of failure. It can be run in an active/standby configuration with failover measured in seconds; but should the data center be taken down by a disaster, the GRIDSCALE server is lost. However, in this case, operation can continue with a remote database server until the GRIDSCALE configuration can be reestablished.

A GRIDSCALE system may be characterized as an AD/S/T active/active system. GRIDSCALE meets all of the active/active attributes. It is available since the loss of any database server simply means that the failed database server is evicted from the database server pool. It is survivable because it can continue in the face of the loss of a data center. In a *write seldom, read often* application, it is scalable by adding additional database servers.

Finally, it uses the one active/active configuration that guarantees consistency since the sequence of execution of all reads and all writes is enforced. (GRIDSCALE does allow query applications to be run on each database server independently of GRIDSCALE. In this case, there is no enforcement of consistency.)

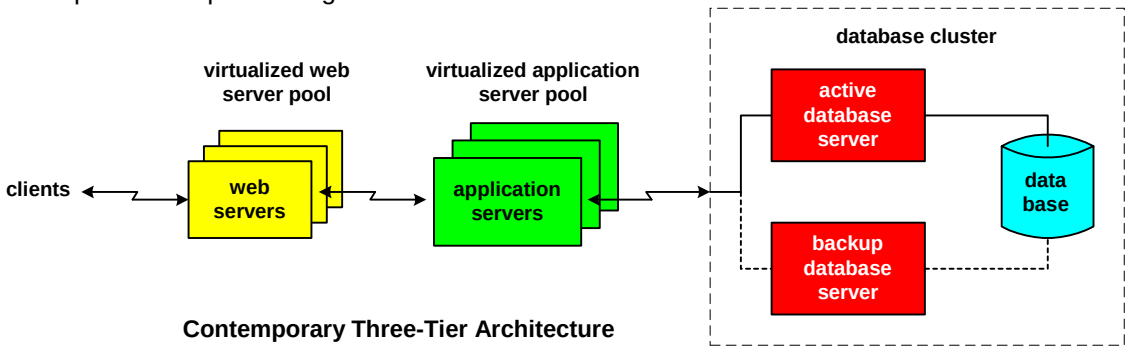
Why Contemporary Clusters Are Not Active/Active

An interesting exercise is to apply the Active/Active Attribute Definition to the currently popular three-tier architecture comprising a presentation tier, an application tier, and a database tier.

- The *Presentation Tier* manages data received from and sent to the clients of the system. Perhaps the most widespread example of a presentation tier is that comprising web servers.

- The *Application Tier* processes requests from clients received from the presentation tier and returns results to the clients through the presentation tier.

- The *Database Tier* provides the application database needed by the application tier to perform its processing functions.



Contemporary Three-Tier Architecture

In typical systems today, the presentation tier and the application tier are virtualized. That is, they comprise a pool of similar servers, any of which can provide the required processing. Should a

server fail, it can be removed from the pool; and the rest of the pool will continue to carry the system load with no interruption to the users. If more capacity is needed in a tier, additional servers can be added to help carry the load.

However, the database tier is typically a single system – often a cluster. Although it may be configured as an active/backup system with redundant data storage such as RAID, it is not scalable without installing larger servers since only one server is active at a time. Furthermore, failover is not transparent to the user. Clusters typically take minutes to failover, and that is only if there has been no database corruption caused by the failure of the primary server.

Consequently, this architecture does not qualify as an active/active system because of the database tier. For one, cluster failover times preclude the possibility of achieving six 9s of availability – typical cluster availabilities are a little less than five 9s. A cluster cannot easily be geographically distributed for survivability. Furthermore, in order to increase the capacity of the cluster, the database-server systems must be replaced with larger database-server systems, necessarily requiring the downing of the cluster (clusters typically require homogeneity). Consistency is not an issue since there is only one active database server.

In short, clusters are not active/active according to our definition.

Summary

Active/active systems are characterized by the attributes of availability, survivability, scalability, and consistency. They eliminate planned downtime and have expected mean time between failures that is measured in centuries. They can survive the failure of an entire data center. They are scalable by simply adding compute resources with no impact to the users. They will consistently execute operations in a predictable manner across the application network.

A common way to implement active/active systems is via context-free virtualized pools of compute resources. Each pool is virtualized to appear as a single resource to external users. If a member of a pool fails, work that it had in progress is simply resubmitted to another member.

At least one of the resource pools must maintain application state, usually by managing the application database. Since the database is a virtualized pool, this means that there will be two or more copies of the application database in the database pool. These databases must be kept in synchronism. This is typically done by asynchronous or synchronous replication at the operation or transaction level.

There are many implementations of active/active systems in production today. Some are complete products. Others are custom-developed systems using data-replication products. For those applications that require continuous uptime, active/active systems are a proven approach to meet this need.