the Availability Digest

OpenVMS Active/Active Split-Site Clusters

June 2008

HP OpenVMS clusters offer a dramatic improvement over contemporary cluster technology. Nodes in an OpenVMS cluster run in an active/active mode in which multiple nodes across multiple sites cooperate in a common application against a common, distributed file system. The recent "Disaster Proof" video from HP,[1] in which a data center was blown up, showed that OpenVMS had the fastest recovery time of all the clustering technologies used (OpenVMS, HPUX, NonStop, Windows and Linux).

In our earlier article on clusters,[2] it was pointed out that contemporary clusters do not run in an active/active mode in our sense because a disk volume can be mounted only on one node at a time (unless Oracle RAC is used), and only that node can participate in the application. Consequently, when a node fails, the application has to be started on another node, the volume remounted and repaired, and the users switched. This leads to failover times for contemporary clusters measured in minutes or more.

Like active/active systems, OpenVMS clusters recover in seconds because once a failure is detected, all that must be done to continue operation is to switch the subset of users who were connected to the failed node to surviving nodes at any of the sites. Furthermore, no data is lost following a failure (a Recovery Point Objective, or RPO, of zero is achieved) because the application file system copies are updated synchronously.

OpenVMS Cluster Overview

An HP OpenVMS cluster is a shared-everything cluster that can have up to 96 nodes[3] distributed over one or more geographically-separated sites. Redundant data storage is organized as *shadow sets* using HBVS (Host-Based Volume Shadowing). Each disk (or presented storage device in a fibre channel disk array, where each presented device could itself be a RAID 0+1 entity) can be a member of a shadow set, and a shadow set can have up to three members. All disk members of a shadow set are exact copies of each other.

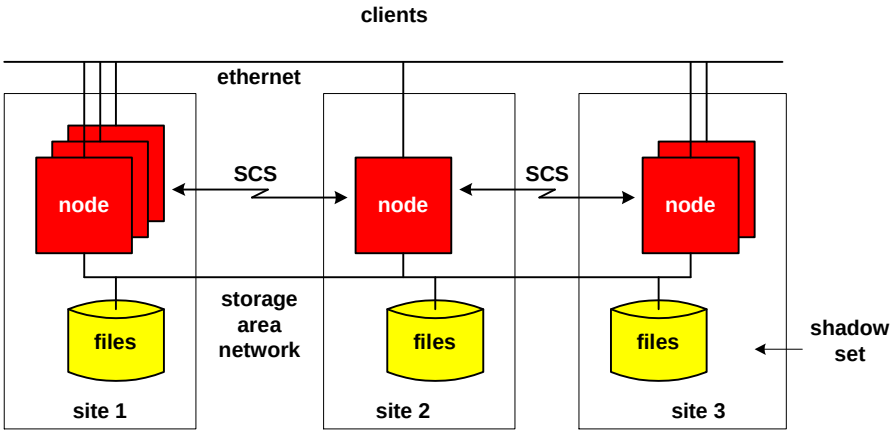
The three members of a shadow set may be distributed across as many as three of the nodes in the cluster. When fibre channel disk arrays in a storage area network are used, all nodes have simultaneous access to all shadow set members. (The three-member limit is currently being increased with an architectural limit of sixteen members and with support for up to six members anticipated). A cluster can support up to 500 disks in multiple-member sets or up to 10,000 disks in single-member sets.

---

[1] [1] http://h20219.www2.hp.com/enterprise/cache/523434-0-0-0-121.html.

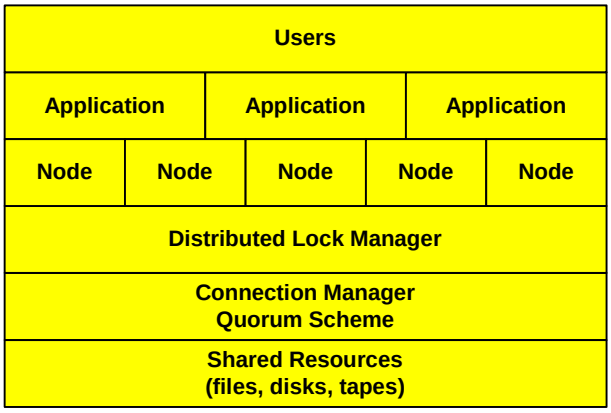
[2] Active/Active Versus Clusters, *Availability Digest*; May 2007.

[3] Actually, the system is designed to accommodate up to 255 nodes but has been qualified for only 96 nodes. Although larger clusters are not supported, some customers have run clusters with 150 nodes or more.



An application can be distributed across several nodes, and users can use any instance of an application on any node. To support shared-everything, OpenVMS clusters provide a Distributed Lock Manager that manages lock requests and a Connection Manager that manages the reconfiguration of the cluster should there be a failure. The Connection Manager ensures that there will not be two cluster subsets actively processing in a split-brain mode.

Shadow sets are managed via the Host-Based Volume Shadowing (HBVS) facility of OpenVMS clusters. Within a shadow set, writes are synchronous. That is, a write is applied to all disks in a shadow set simultaneously; and a completion status is returned to the application only when all writes have completed. If a member of a shadow set fails, it is removed from the shadow set until it can be repaired, at which time it will be synchronized with the shadow set. Thus, no data is lost in the event of a failure.



Reads are directed to the disk that can respond the fastest. This is determined by several factors, such as the communication latency of the path to the disk, the number of reads currently queued to the disk, and the speed of the storage array containing the disk volume.

The nodes in the cluster and the disks in a shadow set can be heterogeneous. A cluster can comprise a mix of AlphaServers or Integrity Servers as nodes, or it can contain a mix of VAX and AlphaServer nodes (or VAX and Integrity Server nodes for migration purposes). The supported mixed-version clusters permit “rolling upgrades” so that a cluster can be maintained and have the operating system upgraded while other cluster members continue to provide service.

The disk storage arrays in a shadow set can be a mix of disk technologies, including SCSI disks and fiber-connected arrays organized into a storage area network. For instance, a shadow member can be expanded on-the-fly to a larger disk. The virtual disk represented by the shadow set will have a size equal to the smallest disk in the shadow set. When all disks have been expanded, the shadow set virtual disk can be expanded to provide an increase in storage capacity.

Host-Based Volume Shadowing is not itself transaction-oriented. It is file-oriented, managing files under RMS (the Record Management System). OpenVMS clusters include DECdtm, a distributed transaction manager, to provide transaction semantics. DECdtm can coordinate the actions of resource managers such as Oracle, Rdb (an Oracle relational database for OpenVMS), or OpenVMS RMS Journaling (for RMS files).

Internode and Intersite Communications

Communications within the cluster are managed by SCS, the System Communication Services. The nodes within the cluster are interconnected via high-speed channels such as FDDI or gigabit Ethernet organized as a LAN. Each node can communicate directly with every other node – there is no routing through a node. These channels are used to relay lock information, disk-block contents, and Connection Manager quorum information between nodes. They are also used to allow the nodes to communicate with each other for purposes of heartbeats ("hello" messages) to monitor the health of the nodes.

The SCS protocol is a high-throughput, low-latency layer 2 protocol running directly on the LAN. With recent changes in wide-area network capabilities and the predominance of TCP/IP-managed services, there is work under way to implement the SCS protocol over a TCP/IP network rather than requiring an extended LAN layer 2 network for cluster communications.

Typically, dual extended LANs are provided. SCS will use multiple LANs in a load-sharing arrangement. If multiple interconnects of different types are used, SCS will choose the best channel to use and will fail over to the other channel should the primary channel fail.

The shadow set disk arrays are interconnected by dual fiber channels forming a storage area network (SAN). A shadow set presents a virtual image of its members to the applications running in the nodes. Any node can write to or read from any shadow set over the storage area network and sees that shadow set simply as a single disk.

Should a node fail or communications be lost between a pair of nodes, the cluster is reconfigured by the Connection Manager based on a quorum computation, as described later. The cluster portion that has quorum (a majority of votes) survives, and the rest of the cluster is taken out of service until a repair can be made. As a consequence, there can be no split-brain operation nor can there be a tug-of-war between cluster nodes attempting to gain control of the cluster.

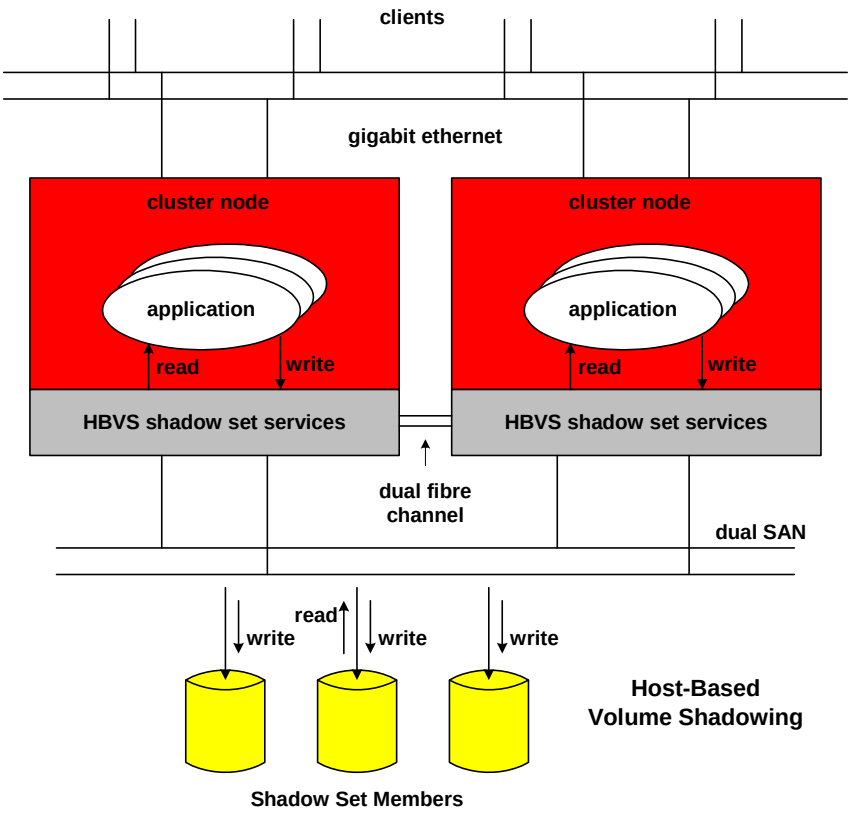
User access to the clusters is typically implemented over a separate network, thus avoiding contention with the cluster interconnects.

Host-Based Volume Shadowing (HBVS)

HBVS is the OpenVMS cluster facility that manages shadow sets. A copy of HBVS runs on every node in the cluster, and the HBVS instances communicate with each other over high-speed links to coordinate locks and cache. All read and write requests are directed to the application's local copy of HBVS, which provides the shadow-set services.

All writes are at the 512-byte block level. When an application updates a record, that record is sent to its local copy of RMS or to the database manager being used. RMS or the database manager will first obtain a cluster-wide lock on the record to be rewritten. If the block containing the record is not currently in the node's local cache, HBVS will read the block into that cache. It will update the block and will write it simultaneously to all members of the shadow set. Only when all writes have completed is the write completion status returned to the application.

When a read request is received, HBVS will determine which shadow set member can respond most quickly. This determination is based upon several factors, such as the length of read queues for the disk, the speed of the disk's storage array, and the communication latency separating the disk from the node making the request.



Both reads and writes can be synchronous or asynchronous. For asynchronous operations, the application is notified of completion by an AST, an asynchronous trap that is a special interrupt to running code.

The database manager or the file system being used shares several responsibilities with HBVS:

- To synchronize writes to all members of a shadow set, the database manager or the file system uses distributed locks and distributed local cache provided by OpenVMS.
- HBVS writes synchronously to all of the disk members of a shadow set.
- HBVS optimizes reads by reading from the shadow-set member that will respond the fastest.
- HBVS resynchronizes a disk when that disk is being returned to service so that it can rejoin its shadow set.
- HBVS synchronizes new disks that are being added to a shadow set.

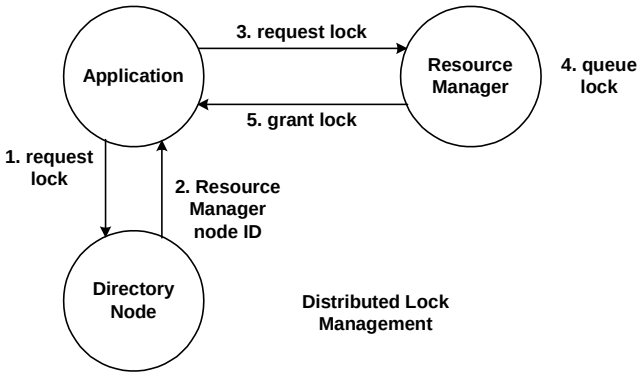
Distributed Lock Management

OpenVMS synchronizes write activities among the nodes in the cluster via distributed lock management. OpenVMS supports several levels of locks, including read locks, write locks, and exclusive locks.[4]

[4] W. E. Snaman, D. W. Thiel, The VAX/VMS Distributed Lock Manager, *Digital Technical Journal, No. 5*; September, 1987.

A lock request may be non-queued, queued synchronous, or queued asynchronous. If the lock is requested without queuing, and if the lock is unavailable, the request is simply denied. If queuing is requested, the lock request is queued behind other earlier lock requests until it can be granted in turn. If synchronous lock queuing is requested, the application is paused until the lock can be granted. If asynchronous lock queuing is requested, the application continues processing and is notified by an asynchronous trap (AST) when the lock has been granted. The AST can invoke an application routine specified by the lock request.

To handle distributed lock management, OpenVMS first defines Directory Nodes. A Directory Node contains the node IDs of the nodes that are currently acting as Resource Managers, if any, for a tree of resource objects. When a node wishes to request a lock, it hashes the name of the resource that it wishes to lock to obtain the Directory Node for that resource. It then sends its lock request to the appropriate Directory Node.



If there is a node that is currently acting as a Resource Manager, the Directory Node will return the node ID of the Resource Manager to the requesting node. The requesting node will then resend its lock request to the Resource Manager, which will queue the lock request. When the lock can be granted, the Resource Manager so informs the requesting node.

If the Directory Node is the Resource Manager for the resource, it will manage the lock request directly.

If the Directory Node receives a lock request for which there is no currently assigned Resource Manager, the requesting node is told to become the Resource Manager for this resource tree. The Directory Node creates a directory entry noting this association. Thereafter, the requesting node need not make any further requests to any node. It handles lock requests for the resources which it is managing locally.

In this way, if an application is locking a resource frequently, there is no messaging traffic. To extend this concept, if another node becomes more active on a resource, the role of Resource Manager is passed to that node.

When a Resource Manager has no more lock requests for the resource that it is managing, it notifies the Directory Node, which will remove this entry from its directory. An application can acquire a null lock on a resource, a lock that has no impact except to prevent the Resource Manager for that resource from disappearing.

Using these algorithms, most lock requests are handled locally. Some require one internode request to a remote Resource Manager. At most, two internode requests are required, one to the Directory Node and one to the remote Resource Manager. As a rule of thumb, a local lock request takes about three to five microseconds, depending upon the speed of the server hardware; and a remote lock request takes about 180 microseconds over a Fast Ethernet LAN.

The Distributed Lock Manager allows an application to acquire a lock and hold it if it is going to use the resource for an extended period of time. However, if another lock request is received for that resource, the application holding the lock is notified via an AST so that it can release the lock.

The Distributed Lock Manager provides cluster-wide deadlock detection and will reject a lock request that will lead to a deadlock.

Distributed Cache Management

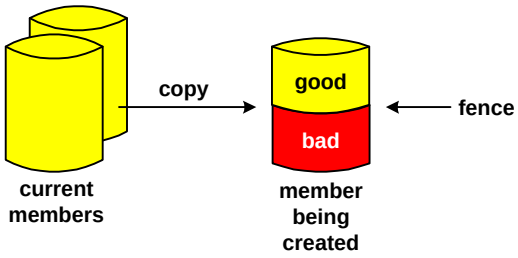
Each node maintains its own local block cache. Associated with each block is a sequence number that is carried with the cached blocks. Whenever a block is written to disk, its sequence number is incremented. At this time, only the writing node and the storage arrays in the SAN have the updated block in their local caches.

When a node wants to read a block, a lock request is generated. The returned message granting the lock also contains the current sequence number for the block. If the node's local cache copy matches the current sequence number, it can use its copy of the block. If the sequence number has changed, the node must fetch the current block from disk.

In a later implementation, the Extended File Cache (XFC) was introduced. In this system, if a node has a copy of a block in its local cache, it has registered an interest in that block. Whenever a block is modified, the modifying node, via a bit map, notifies each node that has registered an interest in the block of the block modification via an AST. Each node then marks its copy of the block in its local cache as being invalid.

Shadow Member Copy

When a new disk (or one that has been out of service for a long time) is to be added to a shadow set, the entire contents of a current member of the shadow set must be copied onto the new member. The current members are designated as the source members, and their contents will be copied to the member-to-be. One of the source members is chosen to be the master.



The copy begins by reading a 127-block segment (the default value) from the master and from the new member being created. If the segment of the new member matches the contents of the segment held by the master, the next segment is tested. If a segment found on the new member is different from that held by the master, the current segment is transferred from the master to the new member. The comparison is repeated following the segment write to ensure that the segment has not been modified during the copy. If so, it is recopied. If this occurs several times (i.e., the segment is being heavily updated), HBVS will acquire a lock temporarily pausing I/Os to the shadow set so that it can copy the segment accurately.

A "fence" on the new member separates the part that has been copied from the part that has not been copied. The part that has been copied is available to participate in read requests from the cluster applications. The part following the fence is not available.

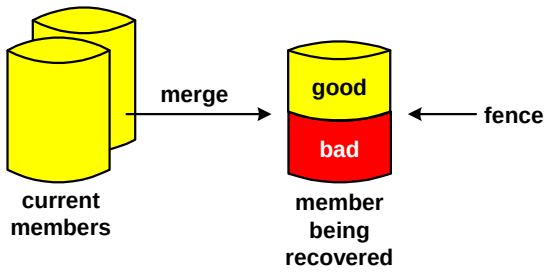
OpenVMS also provides a mini-copy for faster recovery should a member be removed from the shadow set for operational purposes, such as providing a snapshot to be written to tape for backup. In this case, a bit map of all segments on the disk is cleared before the disk is taken offline. Then, as processing continues, each segment that is modified is noted in the bit map. When the member is brought back online, only those segments so noted in the bit map need to be written to the returning member. The initial comparison is not done in this case.

The copy process to an initialized disk can be accelerated by writing a backup copy of the current database to the member to be created. This is relatively fast compared to the copy operation. Next, the copy as described above is performed. Only those segments that have changed since the backup need be written, thus speeding up the copy significantly.

Shadow Member Merge

When a node that is accessing a shadow set leaves the cluster unexpectedly, there is some chance that some of the writes it was performing to the shadow set may have completed on some members but not on others, causing a discrepancy in their contents. To ensure that the data on all shadow-set members is identical, the shadow-set is resynchronized via a merge operation. One of the current shadow-set members is designated the master, and it is used to bring the other member(s) into currency.

The merge operation is similar to the copy operation except that the entire contents of the disk, even those segments after the fence, are available for reading. However, if a segment after the fence is to be read, it must be merged first by comparing it to the master and by refreshing it if necessary. This is required to assure read consistency – that is, the same record contents will be returned on subsequent reads no matter which shadow set member is used.



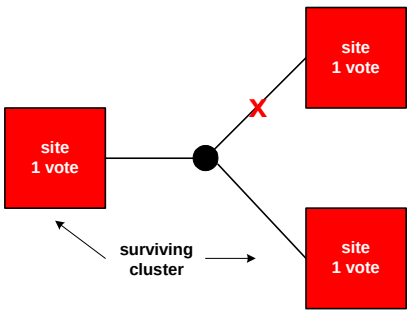
Mini-merges can also be used to recover after node failures. To do this, bit-map recording is started during normal operation. Areas of the shadow set that are written to by nodes in the cluster are recorded by setting bits in the bit map. After a while, a new bit map is started; and the old bit map is discarded. Using this method, HBVS knows what specific areas of the disk have been written to recently; and when a node fails, only those areas written to recently need to be merged instead of the entire contents of the shadow set.

Shadow copies and merges can be sped up by partitioning the data and by performing parallel copies or merges on the partitions.

Quorum

Should a cluster failure occur that isolates part of the cluster from the rest of the cluster, the Connection Manager must reorganize the cluster components into a single cluster by discarding one of the separated parts. It accomplishes this through a quorum procedure. Otherwise, the system could function in “split-brain” mode. Each partial cluster would be processing its own transactions against that part of the disk subsystem to which it had access. The disk contents would diverge, leading to database corruption.

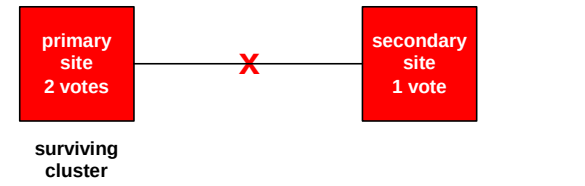
Basically, each processor (and a quorum disk if desired) can be given one or more votes. A quorum requires a majority of votes. For instance, in a three-site system, each site might have one vote. In this case, quorum is two votes. Should one site be disconnected from the other two sites, the disconnected site will have a vote of one. The two connected sites, as a cluster, will have a vote of two. Thus, the connected sites will have quorum and will continue processing. The single site will not have quorum and will halt processing. The quorum decision and subsequent reconfiguration are the responsibility of the cluster’s Connection Manager.



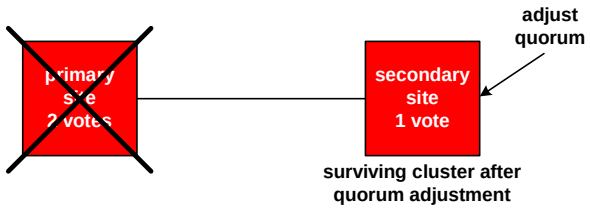
This configuration is often used in a two-site system with a third small quorum site. Alternatively, a quorum disk, not a member of a shadow set, can be connected to the SAN and can provide the

same function. The two operational sites maintain their status on the quorum site or quorum disk for use by the Connection Manager.

If only two sites are used, one site can be declared the primary site and the other site the secondary site by giving the primary site two votes and the secondary site one vote. Again, in this configuration, quorum is two votes.



Should the secondary site fail, or should the communication link between the sites fail, the primary site will continue in operation since it has quorum. However, should the primary site fail, the secondary site will pause processing since it does not have quorum. Manual intervention is needed to change the quorum value to that of the votes held by the surviving site. At this point, the secondary system will take over processing. In a lights-out operation, the lights-out site should be the primary site since it may be difficult to get to it to manually adjust quorum.



There are more complex configurations that are handled. For instance, consider two sites with two nodes each. Site 1's processors are given two votes each, and Site 2's processors are given one vote each. Clearly, Site 1 is the primary site. However, should a Site 1 processor fail, then both sites have two votes; and at this point there is no longer a distinction between the sites in terms of the number of votes. In this case, the site with the most operating nodes – Site 2 in this example – will be considered the new primary site.

In a two-site system in which each site has an equal number of votes, the failure of one site or of the communication link requires manual intervention to give one of the sites quorum so that it can take over processing. Alternatively, the cluster's Connection Manager can be configured to automatically make a choice; or a third site with an additional node to provide a tie-breaking vote can be added.

When a node fails and quorum is maintained, it takes about three seconds to fail over while the lock tables are rebuilt.

Channel Latency

A concern with any synchronous replication method is channel latency. Synchronous operations must wait until all systems involved have reported completion. If some of these systems are remote, significant delays can be introduced to the operations.

The propagation times over networks vary widely depending upon the communication medium and the intervening equipment. However, a useful rule of thumb is that signal propagation will be about one-half the speed of light. This means a round-trip delay of about 2 milliseconds per 100 miles.

OpenVMS split-site clusters are typically configured over campus clusters using fibre-channel links that are up to 100 kilometers in length. This gives a round-trip time of a little over a millisecond that is added to each write operation. For typical applications that are write-once, read-many, experience has shown that this amount of delay is quite tolerable.

There is no fundamental limit to the distance between two OpenVMS cluster sites. In fact, by using SAN channel extenders that extend fibre channels over IP, intersite distances of hundreds

of miles can be achieved. The OpenVMS Cluster Software SPD (Software Product Description) specifies a maximum supported distance of 500 miles based on potential concerns about application performance. However, OpenVMS engineering recently demonstrated a cluster using SCS over IP that operates over sites separated by 8,000 miles – from Nashua, New Hampshire, USA, to Bangalore, India.

Shadow-set member separation of 60,000 miles has been tested by simulation and found to work (60,000 miles was the limit of the test equipment). Why go more than halfway around the world? Maybe to use satellite channels. Also, some circuitous routing, especially after a network path failure, could extend the path length indeterminately.[5]

However, performance rapidly degrades as intersite distances increase. One set of tests showed the following performance over varying distances:[6]

Simulated Distance (mi/km)	One-Way Network Latency (msec.)	Transactions Per Second	Degradation
0/0	0	6,500	0
372/600	3	2,000	3.3
621/1,000	5	1,250	5.2

The performance problems of clusters separated by great distances are further compounded by the amount of internode traffic required to support HBVS. Though a minimum 10 megabit/sec. channel is recommended, heavy block traffic may require significantly greater bandwidth. The bandwidth of even the fastest feasible long-distance channels may throttle performance, adding even more to the performance degradation caused by channel latency. Moreover, the intersite bandwidth of channels required to support greater distances is often determined by the shadow-set copy rather than by the application write load.

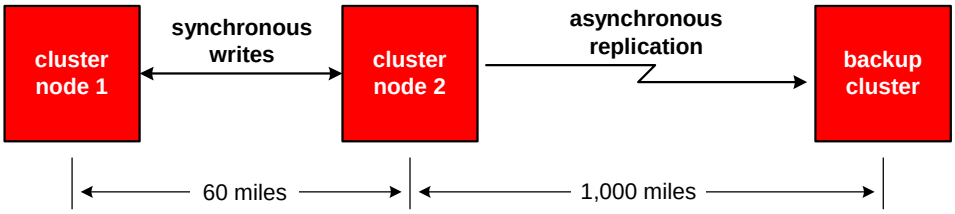
Clearly, one should not run with OpenVMS cluster sites over long distances unless the application can tolerate this sort of performance degradation. That being said, there is one customer that is running a cluster with a 3,000 mile intersite distance.

The effects of channel latency also bring up another issue. It is common practice when negotiating an SLA (Service Level Agreement) with a communications carrier to include bandwidth and error-rate guarantees. When using synchronous replication, it is also important to negotiate channel-latency guarantees. There have been cases where a link failure caused traffic to be rerouted over a long, circuitous route. While the bandwidth and error-rate guarantees were still met, the increased channel latencies brought the applications to their knees.

This leaves the problem of metro clusters and continental clusters - providing a disaster recovery site that is far from the operational sites. OpenVMS clusters solve this problem with asynchronous replication. Using any one of a number of asynchronous replication engines, one site can be chosen to replicate its database to a backup database any distance away. The replicated database cannot be used actively with the production databases, and some data in transit may be lost following a production-site failure. However, this use of asynchronous replication provides the standard disaster-recovery protection common in the industry today.

[5] A recent break in an undersea cable between Egypt and Italy caused traffic to be routed around the world. See What? No Internet?, *Availability Digest*; February, 2008.

[6] Disaster Tolerance Proof of Concept: OpenVMS Host-Based Volume Shadowing and Oracle9i RAC over Extended Distances, *HP White Paper.*



Reliable Transaction Router (RTR)

Another alternative for providing longer distance protection without the performance penalties of synchronous disk replication is to use HP Reliable Transaction Router software (RTR) to do transaction replication. With this software, a transaction is replicated to two different backend servers, each of which could be a full OpenVMS disaster-tolerant cluster in its own right. In effect, the two servers form a server shadow set rather than a disk shadow set. One site is the primary site, and one is the secondary site. With RTR, transaction replication is synchronous. The transaction does not commit until it has been successfully applied to the database in the primary system and has been safe-stored in the secondary system.

Using RTR shadowing, if one of the backend server cluster sites fails, the application continues to operate uninterrupted and without data loss. If one site suffers an outage, the remaining site records the transactions missed by the absent site and can replay them to resynchronize the backend server cluster after the second site returns to service At this point, redundancy has been restored; and processing continues with full protection.

By replicating at the transaction level instead of at the disk-block level, experience at RTR customer sites has shown that higher intersite distances can be tolerated with less adverse performance impact. This approach also avoids any risk of data collisions.

Though this technique consumes twice the system capacity during normal operation, that capacity is generally needed anyway to continue processing in the event of a site failure.

Summary

OpenVMS clusters are the “founding fathers” of cluster technology, having been introduced in 1984. Analyst firm Illuminata has called OpenVMS clusters the “gold standard” for commercial clusters, noting their capability to protect against disasters that affect an entire site.[7]

In a TCO (Total Cost of Ownership) study, research firm TechWise Research compared OpenVMS clusters with cluster technology from IBM AIX System p5 clusters and Sun Solaris Sun-Fire clusters.[8] They found that OpenVMS clusters averaged 2 hours of downtime per year as compared to over six hours per year for IBM clusters and over 9 hours per year for Sun clusters. A major factor in this comparative reliability was the operating systems. OpenVMS and its cluster software experienced less than 0.2 hours per year of downtime as compared to over four hours per year of downtime for IBM’s AIX and Sun’s Solaris, including their cluster software. OpenVMS Integrity clusters offered the lowest TCO for any of the configurations considered. An earlier study by TechWise Research found a similar disparity between OpenVMS clusters and HPUX clusters.[9]

OpenVMS file synchronization is synchronous, whereas most active/active installations today use bidirectional asynchronous replication. Consequently, OpenVMS clusters trade off performance

[7] Illuminata, Inc., Disaster Tolerant Unix: Removing the Last Single Point of Failure, http://h71000.www7.hp.com/openvms/whitepapers/Illuminata.pdf; August 9, 2002.
[8] TechWise Research Inc., Quantifying the Total Cost of Ownership for Entry-Level and Mid-Range Server Clusters; June, 2007.
[9] Techwise Research, Inc., Quantifying the Value of Availability; June, 2000.

over long distances to eliminate the problems of asynchronous replication - data collisions and loss of data following a site failure.

The flip side of this coin is that bidirectional asynchronous replication supports active/active nodes that can be thousands of miles apart provided that the problems of data loss following a node failure and of data collisions can be tolerated.

To achieve disaster tolerance when sites must be separated by great distances, one can build an OpenVMS multisite cluster in which the operational sites are separated by distances commensurate with synchronous replication and then use unidirectional asynchronous replication to a remote site. The remote site cannot actively participate in the application unless it takes over following the failure of the operational sites.

Alternatively, Reliable Transaction Router software may be utilized in conjunction with OpenVMS clusters to do synchronous replication at the transaction level to two geographically-separated servers. This is less sensitive to distance.

OpenVMS clusters are an interesting mix of cluster and active/active technologies. Although they are structured logically as multinode clusters accessing a local file system, they can be split into geographically-separated multinode sites; and the nodes can all be executing common applications as an active/active network. OpenVMS clusters can be put to many additional uses, such as acting as continuously available file servers with clients accessing data through mechanisms such as FTP and NFS, thus protecting data on behalf of less capable systems. With these as significant advantages, the author can think of no disadvantages of OpenVMS clusters over today's contemporary cluster technology.