Convolo: A Case Study in High-Availability Clustering

Richard Rabbat, Tom McNeal, Tim Burke
Mission Critical Linux, Inc.
{rabbat, mcneal, burke@missioncriticallinux.com)
100 Foot of John Street, Lowell, MA 01852
Phone: 877-625-4689, Fax: 978-606-0390

Abstract

This paper discusses the design and implementation of a high-availability clustering solution. This solution provides data integrity guarantees and a simple and effective way for making services highly available. The implementation of the system design in a Linux environment on commodity hardware is documented. A proof-of-concept to provide a highly available NFS server is explained with caveats and their resolutions. Benchmarks are also provided that show the applicability and effectiveness of such a system.

1 Introduction

Clustering has become a necessity in the enterprise in order to guarantee 24/7 service availability. The high visibility brought about by the Internet puts strains the IT infrastructure by growing transaction volumes and increasing the work hours to full 24 hour days. The new demands on the infrastructure increase the number of servers that are accomplishing the same task on one hand, and the need for redundancy on the other.

Cluster servers that deal with the same task, such as web farms, distribute the load amongst themselves to achieve scalability. Those clusters belong to a class of load balancing clusters dealing largely with static information serving. Their dynamic content is served off of databases in the backend. Since load balancing servers deal with static content, they constitute a poor choice for dynamic content that have to maintain data integrity. A database service that is load balanced will not be able to maintain data consistency across multiple servers unless very sophisticated methods are used, such as in Oracle Parallel Server. The same logic applies to NFS servers that would not be able to maintain a consistent view of the file system contents.

Clusters that guarantee data integrity are called high-availability clusters; their focus is to keep a service running at all times. They achieve data integrity by only allowing one system to own a service and move that service to another server when the other service is down or unavailable due to hardware or software problems. They achieve scalability by making each cluster member responsible for a certain subset of the total services.

This paper discusses the Convolo high-availability cluster. Section 2 discusses previous work in this area. Section 3 discusses the design of the Convolo software. Section 4 is a presentation of NFS clustering, as a proof of concept of the viability of the design. Section 5 concludes this paper by discussing achievements as well as future approaches.

2 Previous Work

In a recent white paper on High Availability cluster technologies, the Aberdeen group notes that HA clusters "usually consist of two to eight or more nodes, with two node clusters being the most frequently used". They note that over 80% of clusters currently in use are using only two nodes. The basic technique cited which acheives high availability is the failover mechanism, where the other node(s) will take over the services provided by a node, in the event of the failure of that node. In the event of a failover, the cluster software will move the required resources to the target node. In general, the target node is the surviving node of the two node cluster, but in some clusters with more than two nodes, the resources may actually be moved to multiple surviving nodes.

Another feature noted in the Aberdeen Group white paper is that cluster products appear in two configurations – the "active-active" configuration, where both nodes are running applications at the same time, and the "active-passive" configuration, where one node runs the applications, while the other node stand by, ready to pick up the application load in the event of a failure on the first node. These two designs offer differences in costs and capabilities, and will also differ in behavior when the cluster consists of more than two nodes.

Finally, the white paper also discusses the organization of storage in the clustered environment, noting that the two choices are "shared data" or "shared nothing" storage. With no data sharing, the access to data is reserved by the particular application, and the application must maintain access to the storage devices regardless of which node is currently executing it. Sharing data among the nodes will not necessarily require concurrent access to the same storage devices from all nodes, but will require distributed lock management in order to maintain data integrity.

As an example of cluster design, the Tru64 Cluster implements a Highly-Available solution. Its scalability and manageability are counterbalanced by its use of proprietary hardware as well as SCSI reservations makes the design highly dependent on proper operation of the hardware in question and the specialization of the hardware drivers to accommodate the needed enhancements.

Another highly-available cluster system, the Windows 2000 Advanced Server, supports a 2-node cluster, and can also run in either "active-active" or an "active-passive" configuration. This type of cluster can detect network failures as well as node failures. It does not, however, deal with hung nodes that may go live again, therefore creating the possibility of both nodes running the same service and writing to the same shared partition.

HP's MC/ServiceGuard High-Availability cluster solution may also be configured in either the "active-active" or the "active-passive" configuration, and will support up to 16 nodes in the cluster. When the cluster contains more than 2 nodes, and is configured to operate in the "active-passive" mode, one of the cluster members serves as a standby member on a rotating basis, while all other members are active. It also uses the same

package structure as some other clusters, including the Convolo cluster, where IP addresses are associated with a service, and travel with the service when it migrates from one node to another.

Some new application servers provide their own high-availability schema. The most well known of these is the Oracle Parallel Server (OPS), which uses distributed locking techniques to allow concurrent execution on all members of the cluster. In general, the advantage of such a solution is that the failover mechanism is written and optimized specifically for that service. The disadvantage is the need to rebuild the whole failover for other applications, making this approach expensive and duplicating the effort needed to increase the availability of services by that number of services.

3 The Design of Convolo

The design of Convolo tries to build on several goals: high-availability, data integrity and unspecialized hardware. Shows the hardware layout that is supported in Convolo and specifies the different functional pieces needed in its operation.

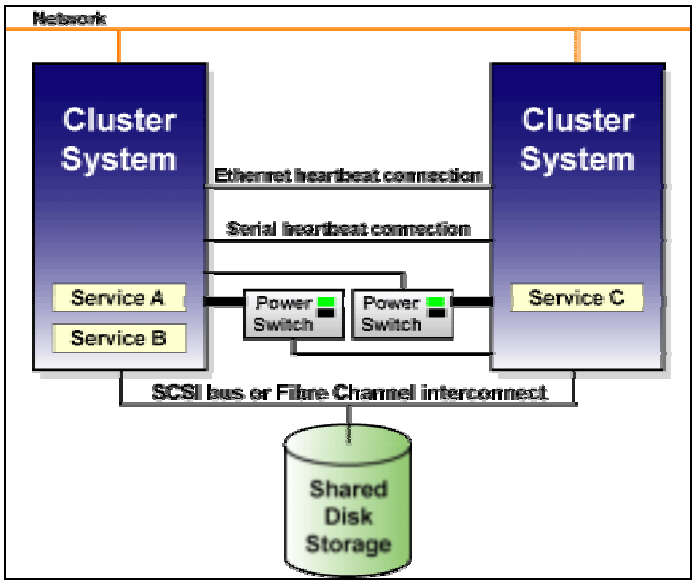

Figure 1 Convolo Cluster Architecture

Several components were built to achieve these goals:

- Host membership: this involves knowing which nodes are considered to be cluster members.
- Shared storage: infrastructure to ensure that the disk devices are accessed by only one server at a time; and that I/O requests cannot be initiated in an unsynchronized manner.
- Services Infrastructure: ability to define high availability services and provide a means of having them stopped and started in response to cluster state transitions.
- System Management: allowing the specification of configuration and tuning parameters, as well as monitoring current operational status.

3.1 Host Membership

The host membership is designed as a peer algorithm, which each cluster member monitoring itself and its peers in the cluster. In addition, there is need for breaking out of inconclusive states. A foundation component is the "Quorum" disk partition (also referred to as shared state disk) used to represent a node's status. On a frequent basis, each node updates its own timestamp on the shared state partition, as well as monitors its peers' state. If a node is unable to access the shared state partition (e.g. SCSI cable pull), that node will take itself out of the cluster by rebooting itself. Inability to access the shared state partition at cluster startup time will cause a node to not commence cluster operation. In the case of an inconclusive state, if a hung node is assumed inoperative and another cluster member takes over services, that hung node may become "un-hung" and the service it hosts would issue I/O requests that would almost certainly compromise data integrity. To prevent that, an I/O barrier is provided through a power switch that may power cycle the hung node after it has stopped responding for a period of time.
As mentioned above, each node monitors the status of other cluster members in order to determine when service state transitions are warranted. Example service state transitions include:

- Cluster node startup
- Cluster node shutdown (planned and unplanned)

Based on a node's state transitions, services will be started up and balanced according to a placement policy.
There are two means of monitoring the state of other members:

3.1.1 Monitoring Status Information in the Shared State Partition

A node whose state on the shared state partition is up and fails to update its timestamp within a specified grace period will be considered failed and forcibly removed from the cluster via a remote power cycle. In this manner, the shared state partition is the cornerstone of the host membership algorithm.

3.1.2 Heartbeat Pinging over Ethernet and/or Serial Ports

Cluster nodes also monitor each other through a set of "heartbeat channels", which can include any number of Ethernet connections (both LAN-based as well as point-to-point). Additionally, Convolo supports point-to-point serial connections as heartbeat channels for 2-node clusters.

3.2 Shared Storage

Shared storage is needed when all of servers are serving dynamic content (such as web content) and need to interact with a common back end data store.
Shared physical storage is required for the cluster shared state partition and is also used for the highly available cluster services. The cluster implementation provides primitives that are utilized by services requiring access to shared storage. Highlights include:

- Service start/stop infrastructure guaranteeing that a service is only running on one node at a time.

- Filesystem mount and unmount mechanisms associated with service start/stop. This includes escalating levels of "forced unmount".
- A low-level lock synchronization mechanism layered on top of the shared state disk primitives. This is used by the service start/stop infrastructure to protect against inherent race conditions.

As the design assumption involves Off-The-Shelf (OTS) hardware, Convolo does not rely on SCSI reservations or any other low-level storage substrate mechanism to reserve devices, or otherwise try to block I/O operations from failed nodes. Instead, the design only requires support for single-initiator SCSI buses or single-initiator Fibre Channel interconnects. To use single-initiator buses, a RAID controller must have multiple host ports and provide simultaneous access to all the logical units on the host ports. Convolo cannot use host-based, adapter-based, or software RAID products in a cluster, because these products usually do not properly coordinate multi-system access to shared storage. Not relying on SCSI reservations also allows Convolo to have a single physical disk associated with more than one highly available cluster service. This obviously improves hardware utilization. In the case of single-initiator SCSI, for hot plugging support, Convolo requires an external LVD active terminator to a host bus adapter that has disabled internal termination. This enables the system to disconnect the terminator from the adapter without affecting bus operation.

3.3 Services Infrastructure: The Service Manager

This is a daemon that runs on all nodes of the cluster to manage availability of cluster services. It ensures that each service is running only once in the cluster and chooses a cluster node if more than one are available to run a service. The service manager takes care of the different service states and applies action appropriately according to its state machine to take a service from a certain state to another or to copy with an error status. Table 1 shows the different states that a service could be in. A description of each state is also presented. This allows us to design the state-machine that shows all possible states and transitions between states.

Table 1 Service States

Name	Description
Service Stopped SVC_STOPPED	A service that is in the stopped state is a service that is not running on any cluster node and is a candidate to run. A service in the stopped state does not have an owner assigned to it and all of its resources are not configured on any cluster node.
Service Starting SVC_STARTING	A service in the starting state is a service that has been requested to start on a cluster node. It is in this state until either the start of the service is successful or it fails. This is a transient state and it is owned by the cluster node starting the service
Service Running SVC_RUNNING	A service in the running state is a service that has all of its resources configured on a cluster node. This is a persistent state in which the cluster node that owns the resources owns the service.
Service Stopping SVC_STOPPING	A service in the stopping state is a service that has been requested to stop on a cluster node. It is in this state until either the stop of the service is successful or it fails. This is a transient state owned by the cluster node stopping the service.
Service Disabling SVC_DISABLING	A service in the disabling state is the same as a service in the stopping state except the resulting state will be **SVC_DISABLED**.

	This is a transient state requested by a cluster system administrator. Disabling a service means to stop the service and not start it unless requested by the cluster system administrator
Service Disabled SVC_DISABLED	A service in the disabled state is similar to a service in the stopped state except the service will not be started unless requested by the cluster system administrator. In normal operation the Service Manager will skip over any service in the disabled state and not change its state.
Service Error SVC_ERROR	A service in the error state is a service in which its state cannot be determined. Moving the service out of the error state requires cluster system administrative intervention. It is unclear what resources associated with the service might still be configured on a cluster node.

Figure 2 is the state machine for the service manager that was used in the implementation of the service manager.

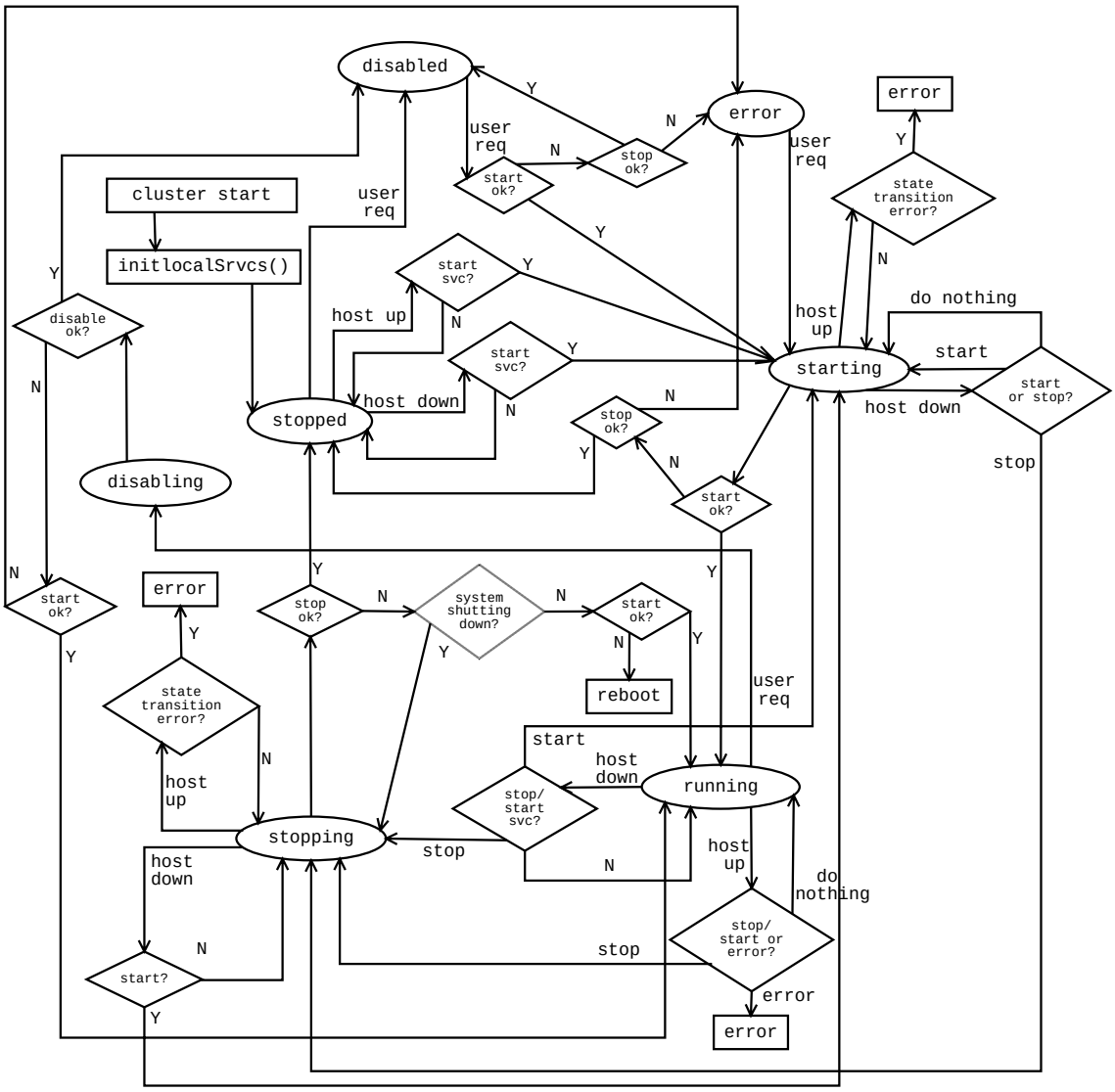

Figure 2 State diagram used by Service Manager

The Service Manager allows the cluster to use the standard start/stop scripts of the daemon/service in question. This enables support for a range of services without the need to develop customized solutions for every service needed. The caveat is to make sure that the cluster node does not try to start the script itself as is common at bootup. The only process that will be allowed to start and stop the service in question is the cluster daemon. If no scripts are provided for a certain service, then it is sufficient to write a minimal script that can start or stop that service depending on the command-line option passed to that script.

Convolo does not provide more data integrity than a server can allow. For example, in the case of the MySQL database server, since the application does not support transactions, if it failed during the processing of server SQL commands, the data may become inconsistent since some in-memory computations may not have been saved to disk. On the other hand, if the application guarantees data integrity, such as the case for PostgreSQL and Oracle through the use of transactions, then Convolo will be able to failover the application from the failed node and run the service on another node consistently, with no loss of data.

The implementation of the Convolo design was done for the Linux operating system. Several tools were put together to ease installation and allow remote administration through a web interface. This interface shows snapshots of the clustered solution, as well as the services running in the cluster. Adding/deleting/disabling/enabling services can all be done remotely through that management tool.

4 Proof-of-Concept: NFS Fail-over

Several features of NFS need to be considered when supporting failover of NFS services in a High-Availability cluster. Some of the considerations may be due to the cluster design itself, and some are unavoidable consequences of NFS designs in general. The general class of problems encountered during our implementation of a cluster using dual connected shared-nothing storage, along with service-specific IP aliases, were related to the following issues:

- *Local File System Usage* - NFS uses files in the Server's local file system to maintain the Server's current state.
- *Migrating the Service IP Address* - The migration of NFS services involves migration of the IP address associated with the service
- *Server Request Queue Management* - Graceful failover may leave the Server which originally supported the NFS service in an inconsistent state
- *Lock Management for an NFS Service* - Starting and stopping the NFS service on the same Server is not well supported by the NFS locking daemons.

In addition, some general lock management bugs not associated with cluster issues were uncovered during testing, and were solved and submitted back into the open source community. One interesting feature we encountered was that supporting graceful

failovers - that is, shutting down and migrating a particular NFS service while leaving the original server running - presented a much more complex environment than the standard catistrophic failover, where the server supporting the particular service has died, either by natural causes, or by being shot by the other server, due to heartbeat or partition failures. This is due partially to the fact that an NFS service is defined as a particular file system export (or exports), and does not include the entire NFS subsystem, and partly to the fact that the server executing a graceful failover does not go through all the initialization stages encountered during a reboot. Making sure that the service in question migrates cleanly, and that the services remaining on the original server (including other exported file systems covered by another NFS service) are unaffected, was a challenging task for the cluster implementation team.

4.1 Local File System Usage

Some clusters associate specific disk partitions and file systems on these partitions with the service being supported by the cluster Servers. In the case of the Convolo cluster, the service partitions are on a shared storage device, separate from the device supporting the Server's local file system. This presents an issue for NFS Servers, since the Server maintains its state by continuously updating several files on the local file system. Migration of the NFS service from one Server to another must assure that these files – or the information they contain – are migrated as well.

The first instance of this issue is found with the Server's mount daemon, rpc.mountd, which keeps the current mount state in the */var/lib/nfs/xtab* file. When clients request mounts from a Server, the Server's mount daemon will add an entry to this file, once the mount has been approved. This file is consulted when requests come in from clients who have already mounted the exported file system, and if the file is not up to date on the Server to which a service has migrated, the Server may deny requests which should be approved.

The second, and more obvious, instance, involves the protocol used by the Server's lock daemon and status monitor daemon, where the status monitor registers all locks granted by the Server in files contained in the directories */var/lib/nfs/sm* and */var/lib/nfs/sm.bak*. These directories allow a recovering Server to notify all clients which own locks in the Server's exported file system, allowing those clients to recover ownership of those locks. If the contents of these directories are not migrated to the Server which is taking ownership of the NFS service, the new Server will be unaware that these locks are held, and will not notify such clients that their locks are no longer valid. If this occurs, the Server may grant a lock to another client, allowing more than one client to own – or think that they own – the lock, resulting in possible data corruption

The solution taken by the Convolo cluster software is to associate files with the particular NFS service (remembering that there may be more that one, each of which is associated with a specific IP address), using the shared storage partition owned by the NFS service.

This assures that all updates to the relevant files and/or directories are seen by both Servers, and will be consistent after a service migration.

4.2 Migrating the Service IP Address

Another issue related to locking, and recovery of locks, is the differentiation between the Server's generic IP address, and the IP address associated with a particular service, whether it is NFS, or any other application. Since the IP address in question is associated with the service, this differentiation from the Server's underlying IP address is important. In the case of lock recovery, it must be clearly understood by the client that the lock should be reacquired from the IP address representing the service.

As noted above, assuring accurate reacquisition of locks requires that the Server keep local lock status, which is accomplished through zero length file entries in the status monitor directories listed above, and that the clients named in these directories are notified by the Server that their previously held locks are no longer held, and must be reacquired. The reacquisition request must be made to the IP representing the NFS service, and not to the IP representing the specific Server itself, since the service address is designed to follow the service, and is not specifically related to an individual Server.

4.3 Server Request Queue Management

When a service migrates from one Server to another, either due to a request for a graceful shutdown, or due to a migration back to the preferred Server after a catastrophic failover and the subsequent Server reboot, the Server which releases ownership of the NFS service for that particular IP address will modify its own privately held export table, in order to eliminate the export of the file system associated with the service.

If the migration is graceful, there may be outstanding requests left on the original Server's NFS daemon queues at the time that the Server deletes the export details from its export table. When these requests are subsequently executed, they will encounter errors since the Server no longer exports this particular file system to the client. In this case, an error (ESTALE) will be generated and sent back to the client. Since the service has actually migrated over to the other Server, the request generating this error should simply be dropped, forcing the client to reissue the request. When the request is reissued, it will be sent to the correct Server, which is now exporting the file system in question, and no ESTALE error will be generated.

4.4 Lock Management for an NFS service

There are two lock management issues encountered during service migration, when the original Server is remains in operation – allowing the service to be released on the original Server (requiring the deletion of the file system in question from the original Server's export list), and allowing the necessary time on the new Server for the clients to reacquire the locks released by the original Server.

In the first case, the locks must be released on the original Server, or else the export command which removes the exported file system from the Server's export tables will encounter an error (EBUSY) due to the locks held on files in this file system. Additionally, the Server which now owns the NFS service must now give the clients which originally held locks the time to reacquire those locks. This is generally accomplished at startup time by giving the lock manager daemon a "grace period" of a given, configurable length, when the lock manager and status monitor are first executed.

Since the NFS service migrates to an already running Server, this grace period must be imposed on the running lock manager, in order to prevent the it from granting locks to other clients, which may conflict with locks currently being reacquired.

4.5 Overall NFS Lock Management

During the implementation and testing of the cluster, the engineers at Mission Critical Linux discovered several generic problems related to locking and lock recovery which were unrelated to the cluster, or to management of cluster services. These were solved, and submitted back to the open source community prior to the release of 2.2.18, and are included in that release. The specific problems encountered were:

- An infinite loop in nlm_traverse_locks if a lock is held by a client during shutdown
- A memory leak in the status monitor daemon
- Unreliable server notification of all clients holding locks after a server crash
- Client recovery of locks after notification from the server that the server had crashed and lost all locks

4.6 Summary

The Linux environment, the structure and behavior of an NFS Server, and the requirements of a High Availability cluster solution presented challenges in implementing NFS failover capability. The behavior of an NFS client, particularly when using the request and reply mechanisms of UDP, assures that the migration of the NFS service from one server to another in a dual-connected data storage environment is fast, seamless and straightforward, with no effect upon data integrity. The client and server behavior in a connection-oriented network may present further challenges for clustered servers, and will be addressed when NFS over TCP has been successfully implemented in the Linux operating system.

5 Conclusion

High-Availability cluster implementations are currently available in a range of Operating System environments, offering different configurations, behavior, manageability, and reliability options. This paper discusses one implementation, running on the Linux operating system, using design assumptions weighted toward execution on "off-the-shelf" hardware, and towards supporting any software service. In general, service support may

require a minimal set of requirements, but in some cases, such as NFS, the cluster design as well as the service design will require service modification as well as cluster software changes. The overall cluster design, implementation, and behavior will impact the supportability of generic services, and the service behavior must handle the clustered environment, as well as the cluster failover mechanism.

6 References

Analysis of High Availability Cluster Technologies for Linux: Who are the Leaders?, Aberdeen Group, Inc., January, 2001

Clusters for High Availability, Peter S. Weygant, Prentice Hall, 1996

NFS Illustrated, Brent Callaghan, Addison Wesley, December, 1999

Private Communication, Gregory Myrdal, Mission Critical Linux, August, 2000