CHAPTER 8

Failover

This chapter describes failover in a Lustre system and includes the following sections:

8.1 What is Failover?

A computer system is “highly available” when the services it provides are available with minimal downtime. In a highly-available system, if a failure condition occurs, such as loss of a server or a network or software fault, the services provided remain unaffected. Generally, we measure availability by the percentage of time the system is required to be available.

Availability is accomplished by providing replicated hardware and/or software, so failure of the system will be covered by a paired system. The concept of “failover” is the method of switching an application and its resources to a standby server when the primary system fails or is unavailable. Failover should be automatic and, in most cases, completely application-transparent.

In Lustre, failover means that a client that tries to do I/O to a failed OST continues to try (forever) until it gets an answer. A userspace sees nothing strange, other than that I/O takes (potentially) a very long time to complete.

Lustre failover requires two nodes (a failover pair), which must be connected to a shared storage device. Lustre supports failover for both metadata and object storage servers. MDS failover is achieved most simply by powering off the MDS node in failure (to be absolutely sure of no multi-mounts of the MDT) and mounting the MDT on the partner. When the primary comes back, it MUST NOT mount the MDT while secondary has it mounted. The secondary can then unmount the MDT and the primary mount it.

The Lustre file system only supports failover at the server level. Lustre does not provide the tool set for system-level components that is needed for a complete failover solution (node failure detection, power control, and so on).[1]

Lustre OSS failover is dependent on either a primary or backup OST to recover the file system. You need to set up an external HA mechanism. The recommended choice is the Heartbeat package, available at:

www.linux-ha.org

Heartbeat is responsible to detect failure of the primary server node and control the failover. The HA software controls Lustre using its built-in "file system" mechanism to unmount and mount file systems. Although Heartbeat is recommended, Lustre works with any HA software that supports resource (I/O) fencing.

The hardware setup requires a pair of servers with a connection to a shared physical storage (like SAN, NAS, hardware RAID, SCSI and FC). The method of sharing storage should be essentially transparent at the device level, that is, the same physical LUN should be visible from both nodes. To ensure high availability at the level of physical storage, we encourage the use of RAID arrays to protect against drive-level failures.

To have a fully-automated, highly-available Lustre system, you need power management software and HA software, which must provide the following -

- **Resource fencing** - Physical storage must be protected from simultaneous access by two nodes.
- **Resource control** - Starting and stopping the Lustre processes as a part of failover, maintaining the cluster state, and so on.
- **Health monitoring** - Verifying the availability of hardware and network resources, responding to health indications given by Lustre.

1. This functionality has been available for some time in third-party tools.

For proper resource fencing, the Heartbeat software must be able to completely power off the server or disconnect it from the shared storage device. It is imperative that no two active nodes access the same storage device, at the risk of severely corrupting data. When Heartbeat detects a server failure, it calls a process (STONITH) to power off the failed node; and then starts Lustre on the secondary node using its built-in "file system" resource manager.

Servers providing Lustre resources are configured in primary/secondary pairs for the purpose of failover. When a server `umount` command is issued, the disk device is set read-only. This allows the second node to start service using that same disk, after the command completes. This is known as a *soft* failover, in which case both the servers can be running and connected to the net. Powering off the node is known as a *hard* failover.

8.1.1 The Power Management Software

The Linux-HA package includes a set of power management tools, known as STONITH (Shoot The Other Node In The Head). STONITH has native support for many power control devices, and is extensible. It uses expect scripts to automate control. PowerMan, by the Lawrence Livermore National Laboratory (LLNL), is a tool for manipulating remote power control (RPC) devices from a central location. Several RPC varieties are supported natively by PowerMan.

The latest versions of PowerMan are available at:

http://sourceforge.net/projects/powerman

For more information on PowerMan, go to:

https://computing.llnl.gov/linux/powerman.html

8.1.2 Power Equipment

A multi-port, Ethernet addressable RPC is relatively inexpensive. For recommended products, refer to the list of supported hardware on the PowerMan site. Linux Network Iceboxes are also very good tools. They combine the remote power control and the remote serial console into a single unit.

8.1.3 Heartbeat

The Heartbeat package is one of the core components of the Linux-HA project. Heartbeat is highly-portable, and runs on every known Linux platform, as well as FreeBSD and Solaris. For more information, see:

http://linux-ha.org/HeartbeatProgram

To download Linux-HA, go to:

http://linux-ha.org/download

Lustre supports both Heartbeat V1 and Heartbeat V2. V1 has a simpler configuration and works very well. V2 adds monitoring and supports more complex cluster topologies. For additional information, we recommend that you refer to the Linux-HA website.

8.1.4 Connection Handling During Failover

A connection is alive when it is active and in operation. When a connection request is sent, a connection is not established until either a reply arrives or a connection disconnects or fails. If there is no traffic on a given connection, periodically check the connection to ensure its status.

If an active connection disconnects, it leads to at least one timeout request. New and old requests are in sleep until:

- The reply arrives (in case of re-activation of the connection and during the re-send request asynchronously).
- The application gets a signal (such as TERM or KILL).
- The server evicts the client, which gives an I/O error (EIO) for these requests or the connection becomes "failed."

A timeout is effectively infinite, and Lustre waits as long as it needs to avoid giving the application an EIO. A client process waits until the OST is back alive, unless either the process is killed (which should be possible after the Lustre recovery timeout is exceeded, 100s by default) or the OST is explicitly marked "inactive" on the clients.

Note – If an OST becomes unavailable, and you want clients to return -EIO if they access files located on the OST, then deactivate the OSC on the client:

```
lctl --device <failed OSC device on the client> deactivate
```

After the OSC is marked inactive, all I/O to this OST should immediately return with -EIO, and not hang.

Note – Under heavy load, clients may have to wait a long time for requests sent to the server to complete (100s of seconds in some cases). It is difficult for clients to distinguish between heavy server load (common) and server death (unlikely).

In the case where a server dies and fails over, the clients have to wait for their requests to time out, then they re-send and wait again (in the common case the server is just overloaded), then they try to contact another server listed as a failover server for that node.

If a connection goes to the "failed" condition, which happens immediately in "failout" OST mode, new and old requests receive EIOs. In non-failout mode, a connection can only get into this state by using `lctl deactivate`, which is the only option for the client in the event of an OST failure.

Failout means that if an OST becomes unreachable (because it has failed, been taken off the network, unmounted, turned off, etc.), then I/O to get objects from that OST cause a Lustre client to get an EIO.

8.1.5 Roles of Nodes in a Failover

A failover pair of nodes can be configured in two ways – **active / active** and **active / passive**. An active node actively serves data while a passive node is idle, standing by to take over in the event of a failure. In the following example, using two OSTs (both of which are attached to the same shared disk device), the following failover configurations are possible:

- **active / passive** - This configuration has two nodes out of which only one is actively serving data all the time.

  In case of a failure, the other node takes over.If the active node fails, the OST in use by the active node will be taken over by the passive node, which now becomes active. This node serves most services that were on the failed node.

- **active / active** - This configuration has two nodes actively serving data all the time. In case of a failure, one node takes over for the other.

  To configure this for the shared disk, the shared disk must provide multiple partitions; each OST is the primary server for one partition and the secondary server for the other partition. The **active / passive** configuration doubles the hardware cost without improving performance, and is seldom used for OST servers.

8.2 OST Failover

The OST has two operating modes: **failover** and **failout**. The default mode is failover.

- **Failover** - Clients attempt to connect to each OSS node configured to serve the OST, until one of them responds with it active. Data on the OST is written synchronously, and the clients replay transactions which were in progress and uncommitted to disk before the OST failure. In the typical OST failover scenario, an OSS node fails and the other node mounts the OST (typically done by Linux HA/Heartbeat). When this happens, no applications see any errors.
- **Failout** - When the underlying hardware has failed or the connection to storage has failed (one reason to use multipath IO), Lustre returns IO errors to the application.

8.3 MDS Failover

The MDS has only one failover mode: active/passive, as only one MDS may (can?) be active at a given time. In a failover configuration, there are two MDSs, each of which have access to the same MDT. Either MDS can mount the MDT, but not both at the same time.

8.4 Configuring Lustre for Failover

For OST failover, multiple OSS nodes are configured to be able to serve the same OST. Only one OSS node can serve the OST at a time. An OST can be moved back and forth between OSS nodes (using the umount/mount commands), as long as the OSSs can access the same disk.

Note – Defining an OST for failover does not require that more than OSS be defined for it. You can provide failover service (i.e., no I/O errors to clients) using a single OSS. In this configuration, if the OST fails, clients are blocked until the OST becomes active again.

For MDT failover, two MDSs are configured to serve the same MDT. Only one MDS node can serve the MDT at a time.

To add a failover partner to a Lustre configuration, use the `--failnode` option. This may be done at creation time with `mkfs.lustre` or at a later time with `tunefs.lustre`. For a failover example, see More Complicated Configurations. For an explanation of the `mkfs.lustre` and `tunefs.lustre` utilities, see mkfs.lustre and tunefs.lustre.

Caution – Lustre's OST failover functionality does not protect against corruption caused by a disk failure. If the storage media (i.e., physical disk) used for an OST fails, Lustre cannot recover it. This is why we strongly recommended that some form of RAID be used for OSTs. Lustre assumes that the storage is reliable and it adds no redundancy to/for OSTs or the MDT.

8.4.1 Starting/Stopping a Resource

You can start a resource with the `mount` command and stop it with the `umount` command. For details, see Mounting a Server and Unmounting a Server.

8.4.2 Active/Active Failover Configuration

With OSSs it is possible to have a load-balanced active/active configuration, which means that out of all of the OSTs that both machines see/use, you mount 50% of them on one OSS and the other 50% on the other OSS, with the capability of one machine taking 100% of them should the other node die.

An OSS is the primary node for a group of OSTs, and the failover node for another group of OSTs. To expand the simple two-node example, we add ost2 which is primary on nodeB, and it's device path is /dev/sdc1 on nodeB and /dev/sdd1 on nodeA. This demonstrates that the /dev/ identity can differ between nodes, but both devices must map to the same physical LUN. In this type of failover configuration, you can mount two OSTs on two different nodes. With failover, two OSSs provide the same service to the Lustre network in parallel. In case of a failure in one of the nodes, the other OSS can provide uninterrupted file system services.

For an active/active configuration, each OSS provides a subset of the connected OSTs. In case one of the OSS fails, the other OSS takes over the other OSTs.

Note – The two OSS nodes must have shared disks.

8.4.3 Hardware Requirements for Failover

This section describes hardware requirements that must be met to configure Lustre for failover.

8.4.3.1 Hardware Preconditions

- The setup must consist of a failover pair where each node of the pair has access to shared storage. If possible, the storage paths should be identical (`nodeA:/dev/sda == nodeB:/dev/sda`).

Note – A failover pair is a combination of two separate nodes. Each node has access to the same shared disk(s).

- Shared storage can be arranged in an active/passive (MDS, OSS) or active/active (OSS only) configuration. Each shared resource has a primary (default) node. Heartbeat assumes that the non-primary node is secondary for that resource.

- The two nodes must have one or more communication paths for Heartbeat traffic. A communication path can be:
  - Dedicated Ethernet
  - Serial live (serial crossover cable)
  - Non-dedicated network connection

All Heartbeat communications failing at the same time will result in a so-called “split-brain” situation. Heartbeat software resolves this situation by powering down one node.

- The two nodes must have a method to control one another's state; RPC hardware is the best choice. The Heartbeat STONITH package provides a number of control methods to shut nodes down. It is recommended to have OSS/MDS equipped with service processors to power down these nodes in a failure situation.
- Heartbeat provides a remote ping service that is used to monitor the health of the external network. If you wish to use the ipfail service, then you must have a very reliable external address to use as the ping target. Typically, this is a firewall route or another very reliable network endpoint external to the cluster.

In Lustre, a disk failure is an unrecoverable error. For this reason, you must have reliable back-end storage with RAID.

Note – If a disk fails, requiring you to change the disk or resync the RAID, you can deactivate the affected OST, using `lctl` on the clients and MDT. This allows access functions to complete without errors (files on the affected OST will be of 0-length, however, you can save rest of your files).

8.5 Setting Up Failover with Heartbeat V1

This section describes how to set up failover with Heartbeat V1.

8.5.1 Installing the Software

1. **Install Lustre (see** Installing Lustre from RPMs**).**
2. **Install the RPMs that are required to configure Heartbeat.**

   The following packages are needed for Heartbeat V1. We used the 1.2.3-1 version. RedHat supplies v1.2.3-2. Heartbeat is available as an RPM or source.

   The required Heartbeat packages are:

   - **heartbeat-stonith** -> `heartbeat-stonith-1.2.3-1.i586.rpm`
   - **heartbeat-pils** -> `heartbeat-pils-1.2.3-1.i586.rpm`
   - **heartbeat itself** -> `heartbeat-1.2.3-1.i586.rpm`

   You can find the above RPMs at:

   http://linux-ha.org/download/index.html#1.2.3
3. **Satisfy the installation prerequisites.**

   Heartbeat 1.2.3 installation requires a number of additional packages. It is recommended to use a package management tool (like yum, yast or aptitude) to install packages.

   Some of the required packages include:

   - **python**
   - **openssl**
   - **libnet**-> `libnet-1.1.2.1-19.i586.rpm`
   - **libpopt** -> `popt-1.7-274.i586.rpm`
   - **librpm** -> `rpm-4.1.1-222.i586.rpm`
   - **glib** -> `glib-2.6.1-2.i586.rpm`
   - **glib-devel** -> `glib-devel-2.6.1-2.i586.rpm`

8.5.1.1 Configuring Heartbeat

This section describes basic configuration of Heartbeat with and without STONITH.

Note – LNET does not support virtual IP addresses. The IP address specified in the haresources file should be a 'dummy' address (valid, but unused). With later releases of Heartbeat, you may avoid the use of virtual IPs, but it is required in earlier releases.

Basic Configuration - Without STONITH

The http://linux-ha.org website has several guides covering basic setup and initial testing of Heartbeat; We suggest that you read them.

1. **Configure and test the Heartbeat setup before adding STONITH.**

   Let us assume there are two nodes, nodeA and nodeB. nodeA owns ost1 and nodeB owns ost2. Both the nodes are with dedicated Ethernet – eth0 having serial crossover link – /dev/ttySO. Consider that both nodes are pinging to a remote host – 192.168.0.3 for health.

2. **Create /etc/ha.d/ha.cf**

   - This file must be identical on both the nodes.
   - Follow the specific order of the directives.
   - Sample `ha.cf` file

   # Suggested fields - logging

   ```
   debugfile /var/log/ha-debug
   logfile /var/log/ha-log
   logfacility local0
   ```

   # Required fields - Timing

   ```
   keepalive 2
   deadtime 30
   initdead 120
   ```

   # If using serial Heartbeat

   ```
   baud 19200
   serial /dev/ttyS0
   ```

   # For Ethernet broadcast

   ```
   udpport 694
   bcast eth0
   ```

   # Use manual failback

   ```
   auto_failback off
   ```

 # Cluster members - name must match `hostname`

```
node oss161.clusterfs.com oss162. clusterfs.com
```

 # remote health ping

```
ping 192.168.16.1
respawn hacluster /usr/lib/heartbeat/ipfail
```

3. **Create /etc/ha.d/haresources**
   - This file must be identical on both the nodes.
   - It specifies a virtual IP address and a service.
   - Sample haresources

```
oss161.clusterfs.com 192.168.16.35 \
Filesystem::/dev/sda::/ost1::lustre

oss162.clusterfs.com 192.168.16.36 \
Filesystem::/dev/sda::/ost1::lustre
```

4. **Create /etc/ha.d/authkeys**
   - Copy the example from `/usr/share/doc/heartbeat-<version>`.
   - chmod the file '0600' – Heartbeat does not start if the permissions on this file are incorrect.
   - Sample authkeys files

```
auth 1
1 sha1 PutYourSuperSecretKeyHere
```

   a. **Start Heartbeat.**

```
[root@oss161 ha.d]# service heartbeat start
Starting High-Availability services:
[ OK ]
```

b. Monitor the syslog on both nodes. After the initial deadtime interval, you should see the nodes discovering each other's state, and then they start the Lustre resources they own. You should see the startup command in the log:

```
Aug 9 09:50:44 oss161 crmd: [4733]: info: update_dc: Set DC to
<null> (<null>)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_election_count_vote:
Election check: vote from oss162.clusterfs.com
Aug 9 09:50:44 oss161 crmd: [4733]: info: update_dc: Set DC to
<null> (<null>)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_election_check:
Still waiting on 2 non-votes (2 total)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_election_count_vote:
Updated voted hash for oss161.clusterfs.com to vote
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_election_count_vote:
Election ignore: our vote (oss161.clusterfs.com)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_election_check:
Still waiting on 1 non-votes (2 total)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_state_transition:
State transition S_ELECTION -> S_PENDING [ input=I_PENDING cause=
C_FSA_INTERNAL origin=do_election_count_vote ]
Aug 9 09:50:44 oss161 crmd: [4733]: info: update_dc: Set DC to
<null> (<null>)
Aug 9 09:50:44 oss161 crmd: [4733]: info: do_dc_release: DC role
released
Aug 9 09:50:45 oss161 crmd: [4733]: info: do_election_count_vote:
Election check: vote from oss162.clusterfs.com
Aug 9 09:50:45 oss161 crmd: [4733]: info: update_dc: Set DC to
<null> (<null>)
Aug 9 09:50:46 oss161 crmd: [4733]: info: update_dc: Set DC to
oss162.clusterfs.com (1.0.9)
Aug 9 09:50:47 oss161 crmd: [4733]: info: update_dc: Set DC to
oss161.clusterfs.com (1.0.9)
Aug 9 09:50:47 oss161 cib: [4729]: info: cib_replace_notify:
Local-only Replace: 0.0.1 from <null>
Aug 9 09:50:47 oss161 crmd: [4733]: info: do_state_transition:
State transition S_PENDING -> S_NOT_DC [ input=I_NOT_DC cause=
C_HA_MESSAGE origin=do_cl_join_finalize_respond ]
Aug 9 09:50:47 oss161 crmd: [4733]: info: populate_cib_nodes:
Requesting the list of configured nodes
Aug 9 09:50:48 oss161 crmd: [4733]: notice: populate_cib_nodes:
Node: oss162.clusterfs.com (uuid: 00e8c292-2a28-4492-bcfc-
fb2625ab1c61)
Sep 7 10:42:40 d1_q_0 heartbeat: info: Running \
/etc/ha.d/resource.d/ost1 start
```

In this example, `ost1` is the shared resource. Common things to watch out for:

- If you configure two nodes as primary for one resource, then you will see both nodes attempt to start it. This is very bad. Shut down immediately and correct your HA resources files.
- If the commutation between nodes is not correct, both nodes may also attempt to mount the same resource, or will attempt to STONITH each other. There should be many error messages in syslog indicating a communication fault.
- When in doubt, you can set a Heartbeat debug level in `ha.cf`—levels above 5 produce huge volumes of data.

c. Try some manual failover/ failback. Heartbeat provides two tools for this purpose (by default they are installed in /usr/lib/heartbeat) –

- `hb_standby [local|foreign]` - Causes a node to yield resources to another node—if a resource is running on its primary node it is local, otherwise it is foreign.
- `hb_takeover [local|foreign]` - Causes a node to grab resources from another node.

Basic Configuration - With STONITH

STONITH automates the process of power control with the expect package. Expect scripts are very dependent on the exact set of commands provided by each hardware vendor, and as a result any change made in the power control hardware/firmware requires tweaking STONITH.

Much must be deduced by running the STONITH package by hand. STONITH has some supplied packages, but can also run with an external script. There are two STONITH modes:

- Single STONITH command for all nodes found in `ha.cf`:

```
-------/etc/ha.d/ha.cf-------------------
stonith <type> <config file>
```

- STONITH command per-node:

```
-------/etc/ha.d/ha.cf--------------------
stonith_host <hostfrom> <stonith_type> <params...>
```

You can use an external script to kill each node:

```
stonith_host nodeA external foo /etc/ha.d/reset-nodeB
stonith_host nodeB external foo /etc/ha.d/reset-nodeA
```

Here, foo is a placeholder for an unused parameter.

To get the proper syntax, run:

```
$ stonith -L
```

The above command lists supported models.

To list required parameters and specify the config file name, run:

```
$ stonith -l -t <model>
```

To attempt a test, run:

```
$ stonith -l -t <mode1> <fake host name>
```

This command also gives data on what is required. To test, use a real hostname. The external STONITH scripts should take the parameters `{start|stop|status}` and return 0 or 1.

STONITH _only happens when the cluster cannot do things in an orderly manner. If two cluster nodes can communicate, they usually shut down properly. This means many tests do not produce a STONITH, for example:

- Calling init 0 or **shutdown** or **reboot** on a node, orderly halt, no STONITH
- Stopping the heartbeat service on a node, again, orderly halt, no STONITH

You have to do something drastic (for example, killall -9 heartbeat) like pulling cables, or so on before you trigger STONITH.

Also, the alert script does a software failover, which halts Lustre but does not halt or STONITH the system. To use STONITH, edit the `fail_lustre.alert` script and add your preferred shutdown command after the line:

```
`/usr/lib/heartbeat/hb_standby local &`;
```

A simple method to halt the system is the sysrq method. Run:

```
$ !/bin/bash
```

This script forces a boot. Run:

```
$ 'echo s' = sync
$ 'echo u' = remount read-only
$ 'echo b' = reboot
$

SYST="/proc/sysrq-trigger"
if [ ! -f $SYST ]; then
          echo "$SYST not found!"
          exit 1
fi

$ sync, unmount, sync, reboot
echo s > $SYST
echo u > $SYST
echo s > $SYST
echo b > $SYST

exit 0
```

8.6 Using MMP

The multiple mount protection (MMP) feature protects the file system from being mounted more than one time simultaneously. If the file system is mounted, MMP also protects changes by e2fsprogs to the file system. This feature is very important in a shared storage environment (for example, when an OST and a failover OST share a partition).

The backing file system for Lustre, `ldiskfs`, supports the MMP mechanism. A block in the file system is updated by a `kmmpd` daemon at one second intervals, and a monotonically increasing sequence number is written in this block. If the file system is cleanly unmounted, then a special "clean" sequence is written in this block. When mounting a file system, ldiskfs checks if the MMP block has a clean sequence or not.

Even if the MMP block holds a clean sequence, ldiskfs waits for some interval to guard against the following situations:

- Under heavy I/O, it may take longer for the MMP block to be updated
- If another node is also trying to mount the same file system, there may be a 'race'

With MMP enabled, mounting a clean file system takes at least 10 seconds. If the file system was not cleanly unmounted, then mounting the file system may require additional time.

Note – The MMP feature is only supported on Linux kernel versions >= 2.6.9.

Note – The MMP feature is automatically enabled by mkfs.lustre for new file systems at format time if failover is being used and the kernel and e2fsprogs support it. Otherwise, the Lustre administrator has to manually enable this feature when the file system is unmounted. If failover is being used, the MMP feature is automatically enabled by `mkfs.lustre`.

- To determine if MMP is enabled: `dumpe2fs -h <device>|grep features`

Example:

```
dumpe2fs -h /dev/sdc | grep features
Filesystem features: has_journal ext_attr resize_inode dir_index
filetype extent mmp sparse_super large_file uninit_bg
```

- To manually disable MMP: `tune2fs -O ^mmp <device>`
- To manually enable MMP: `tune2fs -O mmp <device>`

If `ldiskfs` detects that a file system is being mounted multiple times, it reports the time when the MMP block was last updated, the node name and the device name.

8.7 Setting Up Failover with Heartbeat V2

This section describes how to set up failover with Heartbeat V2.

8.7.1 Installing the Software

1. **Install Lustre (see** Installing Lustre from RPMs**).**
2. **Install RPMs required for configuring Heartbeat.**

   The following packages are needed for Heartbeat (v2). We used the 2.0.4 version of Heartbeat.

   The required Heartbeat packages are:

   - **heartbeat-stonith** -> heartbeat-stonith-2.0.4-1.i586.rpm
   - **heartbeat-pils** -> heartbeat-pils-2.0.4-1.i586.rpm
   - **heartbeat itself** -> heartbeat-2.0.4-1.i586.rpm

   You can find all the RPMs at the following location:

   http://linux-ha.org/download/index.html#2.0.4
3. **Satisfy the installation prerequisites.**

   Heartbeat 1.2.3 installation requires a number of additional packages. It is recommended to use a package management tool (like yum, yast or aptitude) to install packages.

   Some of the required packages include:

   - **Python**
   - **openssl**
   - **libnet**-> `libnet-1.1.2.1-19.i586.rpm`
   - **libpopt** -> `popt-1.7-274.i586.rpm`
   - **librpm** -> `rpm-4.1.1-222.i586.rpm`
   - **libtld**- > `libtool-ltdl-1.5.16.multilib2-3.i386.rpm`
   - **lingnutls** -> `gnutls-1.2.10-1.i386.rpm`
   - **Libzo** -> `lzo2-2.02-1.1.fc3.rf.i386.rpm`
   - **glib** -> `glib-2.6.1-2.i586.rpm`
   - **glib-devel** -> `glib-devel-2.6.1-2.i586.rpm`

8.7.2 Configuring the Hardware

Heartbeat v2 runs well with an un-altered v1 configuration. This makes upgrading simple. You can test the basic function and quickly roll back if issues appear. Heartbeat v2 does not require a virtual IP address to be associated with a resource. This is good since we do not use virtual IPs.

Heartbeat v2 supports multi-node clusters (of more than two nodes), though it has not been tested for a multi-node cluster. This section describes only the two-node case. The multi-node setup adds a score value to the resource configuration. This value is used to decide the proper node for a resource when failover occurs.

Heartbeat v2 adds a resource manager (crm). The resource configuration is maintained as an XML file. This file is re-written by the cluster frequently. Any alterations to the configuration should be made with the HA tools or when the cluster is stopped.

8.7.2.1 Hardware Preconditions

- The basic cluster assumptions are the same as those for Heartbeat v1. For the sake of clarity, here are the preconditions:
- The setup must consist of a failover pair where each node of the pair has access to shared storage. If possible, the storage paths should be identical (`d1_q_0:/dev/sda == d2_q_0:/dev/sda`).
- Shared storage can be arranged in an active/passive (MDS,OSS) or active/active (OSS only) configuration. Each shared resource will have a primary (default) node. The secondary node is assumed.
- The two nodes must have one or more communication paths for heartbeat traffic. A communication path can be:
  - Dedicated Ethernet
  - Serial live (serial crossover cable)

  Failure of all heartbeat communication is not good. This condition is called “split-brain” and the heartbeat software will resolve this situation by powering down one node.
- The two nodes must have a method to control each other's state. The Remote Power Control hardware is the best. There must be a script to start and stop a given node from the other node. STONITH provides soft power control methods (ssh, meatware) but these cannot be used in a production situation.
- Heartbeat provides a remote ping service that is used to monitor the health of the external network. If you wish to use the ipfail service, you must have a very reliable external address to use as the ping target.

8.7.2.2 Configuring Lustre

Configuring Lustre for Heartbeat V2 is identical to the V1 case.

8.7.2.3 Configuring Heartbeat

For details on all configuration options, refer to the Linux HA website:

http://linux-ha.org/ha.cf

As mentioned earlier, you can run Heartbeat V2 using the V1 configuration. To convert from the V1 configuration to V2, use the `haresources2cib.py` script (typically found in `/usr/lib/heartbeat`).

If you are starting with V2, we recommend that you create a V1-style configuration and converting it, as the V1 style is human-readable. The heartbeat XML configuration is located at `/var/lib/heartbeat/cib.xml` and the new resource manager is enabled with the crm yes directive in `/etc/ha.d/ha.cf`. For additional information on CiB, refer to:

http://linux-ha.org/ClusterInformationBase/UserGuide

Heartbeat log daemon

Heartbeat V2 adds a logging daemon, which manages logging on behalf of cluster clients. The UNIX syslog API makes calls that can block, Heartbeat requires log writes to complete as a sign of health. This daemon prevents a busy syslog from triggering a false failover. The logging configuration has been moved to `/etc/logd.cf`, while the directives are essentially unchanged.

Basic configuration (No STONITH or monitor)

Assuming two nodes, d1_q_0 and d21_q_0:

- d1_q_0 owns ost-alpha
- d2_q_0 owns ost-beta
- dedicated Ethernet - eth0
- serial crossover link - /dev/ttySO
- remote host for health ping - 192.168.0.3

Use this procedure:

1. **Create the basic ha.cf and haresources files.**

   haresources no longer requires the dummy virtual IP address.

   This is an example of `/etc/ha.d/haresouces`

   ```
   oss161.clusterfs.com 192.168.16.35 \ Filesystem::/dev/sda::/ost1::lustre
   oss162.clusterfs.com 192.168.16.36 \ Filesystem::/dev/sda::/ost1::lustre
   ```

   Once you have these files created, you can run the conversion tool:

   ```
   $ /usr/lib/heartbeat/haresources2cib.py -c basic.ha.cf \
   basic.haresources > basic.cib.xml
   ```

2. **Examine the cib.xml file**

   The first section in the XML file is <attributes>. The default values should be fine for most installations.

   The actual resources are defined in the <primitive> section. The default behavior of Heartbeat is an automatic failback of resources when a server is restored. To avoid this, you must add a parameter to the <primitive> definition. You may also like to reduce the timeouts. In addition, the current version of the script does not correctly name the parameters.

   ```
   <cib generated="true" admin_epoch="0" epoch="0" num_updates="0" \
   have_quorum="true" ignore_dtd="false" num_peers="2"
   ccm_transition="1" cib-last- \ written="Thu Aug 9 09:50:12 2007">
              <configuration>
                 <crm_config/>
                    <nodes>
   <node id="00e8c292-2a28-4492-bcfs-fb2625ab1c61" \
   uname="oss162.spsoftware.com" type="normal" />
   <node id="e370be9a-24f4-46a5-99ac-41a88c5fa344" \
   uname="oss161.spsoftware.com" type="normal"/>
                    </nodes>
                    <resources/>
                    <constraints/>
              </configuration>
   </cib>
   ```

   a. **Copy the modified resource file to /var/lib/heartbeat/crm/cib.xml**

   b. **Start the Heartbeat software.**

   c. **After startup, Heartbeat re-writes the cib.xml, adding a <node> section and status information. Do not alter those fields.**

Basic Configuration – Adding STONITH

As per Basic configuration (No STONITH or monitor), the best way to do this is to add the STONITH options to ha.cf and run the conversion script. For more information, see:

http://linux-ha.org/ExternalStonithPlugins

8.7.3 Operation

In normal operation, Lustre should be controlled by the Heartbeat software. Start Heartbeat at the boot time. It starts Lustre after the initial dead time.

8.7.3.1 Initial startup

1. **Stop the Heartbeat software (if running).**

   If this is a new Lustre file system:

   ```
   $ mkfs.lustre --fsname=spfs --ost --failnode=oss162 \
   --mgsnode=mds16@tcp0 /dev/sdb (one)
   ```

2. **mount -t lustre /dev/sdb /mnt/spfs/ost/**
3. **/etc/init.d/heartbeat start on one node.**
4. **tail -f /var/log/ha-log to see progress.**
5. **After initdead, this node should start all Lustre objects.**
6. **/etc/init.d/heartbeat start on second node.**
7. **After heartbeat is up on both the nodes, failback the resources to the second node. On the second node, run:**

   ```
   $ /usr/lib/heartbeart/hb_takeover local
   ```

   You should see the resources stop on the first node, and start up on the second node

8.7.3.2 Testing

1. **Pull power from one node.**
2. **Pull networking from one node.**
3. **After Mon is setup, pull the connection between the OST and the backend storage.**

8.7.3.3 Failback

Normally, do the failback manually after determining that the failed node is now good. Lustre clients can work during a failback, but they are momentarily blocked.

Note – When formatting the MGS, the `--failnode` option is not available. This is because MGSs do not need to be told about a failover MGS; they do not communicate with other MGSs at any time. However, OSSs, MDSs and Lustre clients need to know about failover MGSs. MDSs and OSSs are told about failover MGSs with the `--mgsnode` parameter and/or using multi-NID mgsspec specifications. At mount time, clients are told about all MGSs with a multi-NID mgsspec specification. For more details on the multi-NID mgsspec specification and how to tell clients about failover MGSs, see the mount.lustre man page.

8.8 Considerations with Failover Software and Solutions

The failover mechanisms used by Lustre and tools such as Heartbeat are soft failover mechanisms. They check system and/or application health at a regular interval, typically measured in seconds. This, combined with the data protection mechanisms of Lustre, is usually sufficient for most user applications.

However, these *soft* mechanisms are not perfect. The Heartbeat poll interval is typically 30 seconds. To avoid a false failover, Heartbeat waits for a *deadtime* interval before triggering a failover. In normal case, a user I/O request should block and recover after the failover completes. But this may not always be the case, given the delay imposed by Heartbeat.

Likewise, the Lustre health_check mechanism does not provide perfect protection against any or all failures. It is a sample taken at a time interval, not something that brackets each and every I/O request.[2] There are a few places where health_check could generate a bad status:

- On a device basis if there are requests that have not been processed in a very long time (more than the maximum allowed timeout), a CERROR is printed:

  ```
  {service}: unhealthy - request has been waiting Ns
  ```

  Ns is the number of seconds. The CERROR displays a true value for Ns, for example "... request has been waiting 100s"

- If the backing file system has gone read-only due to file system errors
- On a per-device basis if any of the above failed, it is reported in the `/proc/fs/lustre/health_check` file:

  ```
  device {device} reported unhealthy
  ```

- If ANY device or service on the node is unhealthy, it also prints:

  ```
  NOT HEALTHY
  ```

- If ALL devices and services on the node are healthy, it prints:

  ```
  healthy
  ```

There will be cases where a user job will die prior to the HA software triggering a failover. You can certainly shorten timeouts, add monitoring, and take other steps to decrease this probability. But there is a serious trade-off – shortening timeouts increases the probability of false-triggering a busy system. Increasing monitoring takes the system resources, and can likewise cause a false trigger.

Unfortunately, *hard* failover solutions capable of catching failures in the sub-second range generally require special hardware. As a result, they are quite expensive.

Tip – Failover of the Lustre client is dependent on the `obd_timeout` parameter. The Lustre client does not attempt failover until the request times out. Then, the client tries to re-send the request to the original server (if again, an `obd_timeout` occurs). After that, the Lustre client refers to the import list for that target and tries to connect (in a round-robin manner) until one of the nodes replies. The timeouts for the connection are much lower (obd_timeout / 20, 5).

---

2. This is true for every HA monitor, not just the Lustre health_check.