Using Pacemaker with a Lustre File System

This page describes how to configure and use Pacemaker to implement failover in a Lustre file system. The content was contributed by Dr. Michael Schwartzkopff.

NOTE: This content was created for an earlier version of the Lustre file system.

Setting Up Cluster Communications

Communication between the nodes of the cluster allows all nodes to “see” each other. In modern clusters, OpenAIS, or more specifically, its communication stack *corosync*, is used for this task. All communication paths in the cluster should be redundant so that a failure of a single path is not fatal for the cluster.

An introduction to the setup, configuration and operation of a Pacemaker cluster can be found in:

- *Pacemaker documentation:* www.clusterlabs.org/doc/. The latest docs are in *Pacemaker 1.1 for Corosync 2.x and crmsh*.
- M. Schwartzkopff, *Clusterbau: Hochverfügbarkeit mit Linux*, O'Reilly Vlg. Gmbh & Co., Jun 2012. (German)

Setting Up the *corosync* Communication Stack

The *corosync* communication stack, developed as part of the OpenAIS project, supports all the communication needs of the cluster. The package is included in all recent Linux distributions. If it is not included in your distribution, you can find precompiled binaries at www.clusterlabs.org/rpm. It is also possible to compile OpenAIS from source and install it on all HA nodes by running *./configure*; *make* and *make install*.

Note: If *corosync* is not included in your distribution, your distribution may include the complete OpenAIS package. From the cluster point of view, the only difference is that all files and commands start with *openais* rather than *corosync*. The configuration file is located in */etc/ais/openais.conf*.

Once installed, the software looks for a configuration in the file */etc/corosync/corosync.conf*.

Complete the following steps to set up the *corosync* communication stack:

1. Edit the totem section of the *corosync.conf* (or *openais.conf*) configuration file to designate the IP address and netmask of the interface(s) to be used. The totem section of the configuration file describes the way *corosync* communicates between nodes.

```
totem {
        version: 2
        secauth: off
        threads: 0
        interface {
                ringnumber:    0
                bindnetaddr:   192.168.0.0
                mcastaddr:     226.94.1.1
                mcastport:     5405
        }
}
```

Corosync uses the option bindnetaddr to determine which interface is to be used for cluster communication. The example above assumes one of the node’s interfaces is configured on the network 192.168.0.0/24. The value of the option is calculated from the IP address AND the network mask for the interface (IP & MASK) so the final bits of the address are cleared. Thus the configuration file is independent of any node and can be copied to all nodes. If you have multiple interfaces you can add additional *interface{}* sections with different *ringnumbers*.

2. Edit the *aisexec* section of the configuration file to designate which user can start the service. The user must be *root*:

```
aisexec {
        user: root
        group: root
}
```

3. In the service section of the configuration file, add the services that *corosync* is to administer. In this example, only pacemaker is included:

```
service {
        name: pacemaker
        version: 0
}
```

4. (Optional) To use the Pacemaker GUI, add the *mgmt* daemon to the *service* section:

```
service {
        name: pacemaker
        version: 0
        use_mgmtd: yes
}
```

It is worth noting at this point that RHEL5 includes a basic firewall "out of the box". If you wish to run that firewall, you will need to add a rule to allow the Corosync traffic. I did this by adding a rule to the /etc/sysconfig/iptables file in the appropriate place:

```
-A RH-Firewall-1-INPUT -p udp --dport 5405 -j ACCEPT
```

The *corosync* service starts as part of the normal *init* process. It can also be started manually by entering:

```
/etc/init.d/corosync start
```

After *corosync* has started, the following lines should be visible in the system log file:

```
(...) [MAIN ] Corosync (...) started and ready to provide service.
(...) [TOTEM ] The network interface [...] is now up.
```

You can also check for correct functioning of the network stack by entering:

```
# corosync-cfgtool -s
```

The following should be displayed:

```
Printing ring status.
Local node ID (...)
RING ID 0
        id               = (...)
        status                              = ring 0 active with no faults
```

Setting up Redundant Communication Using Bonding

It is recommended that you set up the cluster communication via two or more redundant paths. One way to achieve this is to use bonding interfaces. Please consult the documentation for your Linux distribution for information about how to configure bonding interfaces.

Setting up Redundant Communication within corosync

The *corosync* package provides a means for redundant communication. If two or more interfaces for the communication exist, an administrator can configure multiple *interface{ }* sections in the configuration file, each with a different ringnumber. The *rrd_mode* option tells the cluster how to use these interfaces. If the value is set to *active*, *corosync* uses all interfaces actively. If the value is set to *passive*, *corosync* uses the second interface only if the first ring fails.

```
rrd_mode: active
totem {
        version: 2
        secauth: off
        threads: 0
        interface {
                ringnumber:  0
                bindnetaddr: 192.168.0.0
                mcastaddr:   226.94.1.0
                mcastport:   5400
        }
        interface {
                ringnumber:  1
                bindnetaddr: 192.168.1.0
                mcastaddr:   226.94.1.1
                mcastport:   5401
        }

}
```

Setting Up Resource Management

All services that the Pacemaker cluster resource manager will manage are called resources. The Pacemaker cluster resource manager uses resource agents to start, stop or monitor resources.

Note: The simplest way to configure the cluster is by using a crm subshell. All examples will be given in this notation. If you understood the syntax of the cluster configuration, you also can use the GUI or XML notation.

Completing a Basic Setup of the Cluster

To test that your cluster manager is running and set global options, complete the steps below.

1. Display the cluster status. Enter:

```
 # crm_mon -1
```

The output should look similar to:

```
============
Last updated: Fri Dec 25 17:31:54 2009
Stack: openais
Current DC: node1 - partition with quorum
Version: 1.0.6-cebe2b6ff49b36b29a3bd7ada1c4701c7470febe
2 Nodes configured, 2 expected votes
0 Resources configured.
============

Online: [ node1 node2 ]
```

This output indicates that *corosync* started the cluster resource manager and it is ready to manage resources.

Several global options must be set in the cluster. The two described in the next two steps are especially important to consider.

2. If your cluster consists of just two nodes, switch the quorum feature off. On the command line, enter:

```
 # crm configure property no-quorum-policy=ignore
```

If your lustre setup comprises more than two nodes, you can leave the no-quorum option as it is.

3. In a Lustre setup, fencing is normally used and is enabled by default. If you have a good reason not to use it, disable it by entering:

```
# crm configure property stonith-enabled=false
```

After the global options of the cluster are set up correctly, continue to the following sections to configure resources and constraints.

Configuring Resources

OSTs are represented as Filesystem resources. A Lustre cluster consists of several Filesystem resources along with constraints that determine on which nodes of the cluster the resources can run.

By default, the start, stop, and monitor operations in a Filesystem resource time out after 20 sec. Since some mounts in Lustre require up to 5 minutes or more, the default timeouts for these operations must be modified. Also, a monitor operation must be added to the resource so that Pacemaker can check if the resource is still alive and react in case of any problems.

1. Create a definition of the Filesystem resource and save it in a file such as *MyOST.res*.

If you have multiple OSTs, you will need to define additional resources.

The example below shows a complete definition of the Filesystem resource. You will need to change the *device* and *directory* to correspond to your setup.

```
primitive resMyOST ocf:heartbeat:Filesystem \
        meta target-role="stopped" \
        operations $id="resMyOST-operations" \
        op monitor interval="120" timeout="60" \
        op start interval="0" timeout="300" \
        op stop interval="0" timeout="300" \
        params device="device" directory="directory" fstype="lustre"
```

In this example, the resource is initially stopped (*target-role="stopped"*) because the constraints specifying where the resource is to be run have not yet been defined.

The *start* and *stop* operations have each been set to a timeout of 300 sec. The resource is monitored at intervals of 120 seconds. The parameters "*device*", "*directory*" and "lustre" are passed to the *mount* command.

For the *device* parameter it is highly recommended to avoid using /dev/sd* device names but use something that is immune to device reordering such as the /dev/disk/by-*/* links. /dev/disk/by-uuid/* for example should be coherent for the lifetime of a formatting of the target as those names are constructed with the ldiskfs (i.e. ext3/ext4) UUID. This value should only ever change if you reformat the target.

If you do reformat the target you can discover it's new UUID by running blkid and udevtrigger after the format and either looking at it's output or looking in /dev/disk/by-uuid/ for the new name.

Once the new name has been determined you can update your resource with:

```
crm resource param resMyOST set device /dev/disk/by-uuid/new-uuid
```

2. Read the definition into your cluster configuration by entering:

```
# crm configure < MyOST.res
```

You can define as many OST resources as you want.

If a server fails or the monitoring of a OST results in the detection of a failure, the cluster first tries to restart the resource on the failed node. If the node fails to restart it, the resource is migrated to another node.

More sophosticated ways of failure management (such as trying to restart a node three times before migrating to another node) are possible using the cluster resource manager. See the Pacemaker documentation for details.

If mounting the file system depends on another resource like the start of a RAID or multipath driver, you can include this resource in the cluster configuration. This resource is then monitored by the cluster, enabling Pacemaker to react to failures.

Configuring Constraints

In a simple Lustre cluster setup, constraints are not required. However, in a larger cluster setup, you may want to use constraints to establish relationships between resources. For example, to keep the load distributed equally across nodes in your cluster, you may want to control how many OSTs can run on a particular node.

Constraints on resources are established by Pacemaker through a point system. Resources accumulate or lose points according to the constraints you define. If a resource has negative points with respect to a certain node, it cannot run on that node.

For example, to constrain the co-location of two resources, complete the steps below.

1. Add co-location constraints between resources. Enter commands similar to the following:

```
# crm configure colocation colresOST1resOST2 -100: resOST1 resOST2
```

This constraint assigns -100 points to resOST2 if an attempt is made to run resOST2 on the same node as resOST1. If the resulting total number of points assigned to reOST2 is negative, it will not be able run on that node.

2. After defining all necessary constraints, start the resources. Enter:

```
# crm resource start resMyOST
```

Execute this command for each OST (Filesystem resource) in the cluster.

Note: Use care when setting up your point system. You can use the point system if your cluster has at least three nodes or if the resource can acquire points from other constraints. However, in a system with only two nodes and no way to acquire points, the constraint in the example above will result in an inability to migrate a resource from a failed node.

For example, if resOST1 is running on *node1* and resOST2 on *node2* and *node2* fails, an attempt will be made to run resOST2 on *node1*. However, the constraint will assign resOST2 -100 points since resOST1 is already running on *node1*. Consequently resOST2 will be unable to run on *node1* and, since it is a two-node system, no other node is available.

To find out more about how the cluster resource manager calculates points, see the Pacemaker documentation.

Internal Monitoring of the System

In addition to monitoring of the resource itself, the nodes of the cluster must also be monitored. An important parameter to monitor is whether the node is connected to the network. Each node pings one or more hosts and counts the answers it receives. The number of responses determines how “good” its connection is to the network.

Pacemaker provides a simple way to configure this task.

1. Define a ping resource. In the command below, the *host_list* contains a list of hosts that the nodes should ping.

```
# crm configure resPing ocf:pacemaker:pingd \
  params host_list=“host1 ...“ multiplier=“10“ dampen=”5s“
# crm configure clone clonePing resPing
```

For every accessible host detected, any resource on that node gets 10 points (set by the *multiplier*= parameter). The clone configuration makes the ping resource run on every available node.

2. Set up constraints to run a resource on the node with the best connectivity. The score from the *ping* resource can be used in other constraints to allow a resource to run only on those nodes that have a sufficient ping score. For example, enter:

```
# crm configure location locMyOST resMyOST rule $id="locMyOST" pingd: defined pingd
```

This location constraint adds the *ping* score to the total score assigned to a resource for a particular node. The resource will tend to run on the node with the best connectivity.

Other system checks, such as CPU usage or free RAM, are measured by the Sysinfo resource. The capabilities of the Sysinfo resource are somewhat limited, so it will be replaced by the SystemHealth strategy in future releases of Pacemaker. For more information about the SystemHealth feature, see: www.clusterlabs.org/wiki/SystemHealth

Administering the Cluster

Careful system administration is required to support high availability in a cluster. A primary task of an administrator is to check the cluster for errors or failures of any resources. When a failure occurs, the administrator must search for the cause of the problem, solve it and then reset the corresponding failcounter. This section describes some basic commands useful to an administrator. For more detailed information, see the Pacemaker documentation.

Displaying a Status Overview

The command *crm_mon* displays an overview of the status of the cluster. It functions similarly to the Linux top command by updating the output each time a cluster event occurs. To generate a one-time output, add the
option *-1*.

To include a display of all failcounters for all resources on the nodes, add the *-f* option to the command. The output of the command crm_mon -1f looks similar to:

```
============
Last updated: Fri Dec 25 17:31:54 2009
Stack: openais
Current DC: node1 - partition with quorum
Version: 1.0.6-cebe2b6ff49b36b29a3bd7ada1c4701c7470febe
2 Nodes configured, 2 expected votes
2 Resources configured.
============

Online: [ node1 node2 ]

Clone Set: clonePing
     Started: [ node1 node2 ]
resMyOST       (ocf::heartbeat:filesys): Started node1

Migration summary:
* Node node1:  pingd=20
   resMyOST: migration-threshold=1000000 fail-count=1
* Node node2:  pingd=20
```

Switching a Node to Standby

You can switch a node to standby to, for example, perform maintenance on the node. In standby, the node is still a full member of the cluster but cannot run any resources. All resources that were running on that node are forced away.

To switch the node called *node01* to standby, enter:

```
# crm node standby node01
```

To switch the node online again enter:

```
# crm node online node01
```

Migrating a Resource to Another Node

The cluster resource manager can migrate a resource from one node to another while the resource is running. To migrate a resource away from the node it is running on, enter:

```
# crm resource migrate resMyOST
```

This command adds a location constraint to the configuration that specifies that the resource *resMyOST* can no longer run on the original node.

To delete this constraint, enter:

```
# crm resource unmigrate resMyOST
```

A target node can be specified in the migration command as follows:

```
# crm configure migrate resMyOST node02
```

This command causes the resource *resMyOST* to move to node *node02*, while adding a location constraint to the configuration. To remove the location constraint, enter the *unmigrate* command again.

Resetting the failcounter

If Pacemaker monitors a resource and finds that it isn't running, by default it restarts the resource on the node. If the resource cannot be restarted on the node, it then migrates the resource to another node.

It is the administrator's task to find out the cause of the error and to reset the failcounter of the resource. This can be achieved by entering:

```
# crm resource failcount <resource> delete <node>
```

This command deletes (resets) the failcounter for the resource on the specified node.

"Cleaning up" a Resource

Sometimes it is necessary to "clean up" a resource. Internally, this command removes any information about a resource from the Local Resource Manager on every node and thus forces a complete re-read of the status of that resource. The command syntax is:

```
# crm resource cleanup resMyOST
```

This command removes information about the resource called *resMyOST* on all nodes.

Setting up Fencing

Fencing is a technique used to isolate a node from the cluster when it is malfunctioning to prevent data corruption. For example, if a "split-brain" condition occurs in which two nodes can no longer communicate and both attempt to mount the same filesystem resource, data corruption can result. (The Multiple Mount Protection (MMP) mechanism in Lustre is designed to protect a file system from being mounted simultaneously by more than one node.)

Pacemaker uses the STONITH (Shoot The Other Node In The Head) approach to fencing malfunctioning nodes, in which a malfunctioning node is simply switched off. A good discussion about fencing can be found here. This article provides information useful for deciding which devices to purchase or how to set up STONITH resources for your cluster and also provides a detailed setup procedure.

A basic setup includes the following steps:

1. Test your fencing system manually before configuring the corresponding resources in the cluster. Manual testing is done by calling the STONITH command directly from each node. If this works in all tests, it will work in the cluster.

2. After configuring of the according resources, check that the system works as expected. To cause an artificial “split-brain” situation, you could use a host-based firewall to prohibit communication from other nodes on the heartbeat interface(s) by entering:

```
# iptables -I INPUT -i <heartbeat-IF> -p 5405 -s <other node> -j DROP
```

When the other nodes are not able to see the node isolated by the firewall, the isolated node should be shut down or rebooted.

Setting Up Monitoring

Any cluster must to be monitored to provide the high availability it was designed for. Consider the following scenario demonstrating the importance of monitoring:

Pacemaker offers several options for making information available to a monitoring system. These include:

- Utilizing the *crm_mon* program to send out information about changes in cluster status.
- Using scripts to check resource failcounters.

These options are described in the following sections.

Using *crm_mon* to Send Email Messages

In the most simple setup, the *crm_mon* program can be used to send out an email each time the status of the cluster changes. This approach requires a fully working mail environment and *mail* command.

Before configuring the *crm_mon* daemon, check that emails sent from the command line are delivered correctly by entering:

```
# crm_mon --daemonize –-mail-to <user@example.com> [--mail-host mail.example.com]
```

The resource monitor in the cluster can be configured to ensure the mail alerting service resource is running, as shown below:

```
primitive resMON ocf:pacemaker:ClusterMon \
        operations $id="resMON-operations" \
        op monitor interval="180" timeout="20" \
        params extra_options="--mail-to <your@mail.address>"
```

If a node fails, which could prevent the email from being sent, the resource is started on another node and an email about the successful start of the resource is sent out from the new node. The administrator's task is to search for the cause of the failover.

Using *crm_mon* to Send SNMP Traps

The *crm_mon* daemon can be used to send SNMP traps to a network management server. The configuration from the command line is:

```
# crm_mon –-daemonize –-snmp-traps nms.example.com
```

This daemon can also be configured as a cluster resource as shown below:

```
primitive resMON ocf:pacemaker:ClusterMon \
        operations $id="resMON-operations" \
        op monitor interval="180" timeout="20" \
        params extra_options="--snmp-trap nms.example.com
```

The MIB of the traps is defined in the *PCMKR.txt* file.

Polling the Failcounters

If all the nodes of a cluster have problems, pushing information about events may be not be sufficient. An alternative is to check the failcounters of all resources periodically from the network management station (NMS). A simple script that checks for the presence of any failcounters in the output of *crm_mon -1f* is shown below:

```
# crm_mon -1f | grep fail-count
```

This script can be called by the NMS via SSH, or by the SNMP agent on the nodes by adding the following line to the Net-SNMP configuration in *snmpd.conf*:

```
extend failcounter crm_mon -1f | grep -q fail-count
```

The code returned by the script can be checked by the NMS using:

```
snmpget <node> nsExtend.\”failcounter\”
```

A result of *0* indicates a failure.