PVFS2 High-Availability Clustering using Heartbeat 2.0

2007

Contents

1 Introduction

This document describes how to configure PVFS2 for high availability using Heartbeat version 2.x from www.linux-ha.org. See pvfs2-ha.tex for documentation on how to configure PVFS2 for high availability using Heartbeat version 1.x.

Heartbeat 2.x offers several improvements. First of all, it allows for an arbitrary cluster size. The servers do not have to be paired up for failover. For example, if you configure 16 servers and one of them fails, then any of the remaining 15 can serve as the failover machine.

Secondly, Heartbeat 2.x supports monitoring of resources. Examples of resources that you may want to actively monitor include the PVFS2 server daemon, the IP interface, and connectivity to storage hardware.

Finally, Heartbeat 2.x includes a configuration mechanism to express dependencies between resources. This can be used to express a preference for where certain servers run within the cluster, or to enforce that resources need to be started or stopped in a specific order.

This document describes how to set up PVFS2 for high availability with an arbitrary number of active servers and an arbitrary number of passive spare nodes. Spare nodes are not required unless you wish to avoid performance degradation upon failure. As configured in this document, PVFS2 will be able to tolerate $\lceil((N/2) - 1)\rceil$ node failures, where N is the number of nodes present in the Heartbeat cluster including spares. Over half of the nodes must be available in order to reach a quorum and decide if another node has failed.

No modifications of PVFS2 are required. Example scripts referenced in this document are available in the `examples/heartbeat` directory of the PVFS2 source tree.

2 Requirements

2.1 Hardware

2.1.1 Nodes

Any number of nodes may be configured, although you need at least 3 in order to tolerate a failure. See the explanation in the introduction of this document. You may also use any number of spare nodes. A spare node is a node that does not run any services until a failover occurs. If you have one or more spares, then they will be selected first to run resources in a failover situation. If you have no spares (or all spares are exhausted), then at least one node will have to run two services simultaneously, which may degrade performance.

The examples in this document will use 4 active nodes and one spare node.

2.1.2 Storage

The specific type of storage hardware is not important, but it must be possible to allocate a separate block device to each server, and all servers must be capable of accessing all block devices.

One way of achieving this is by using a SAN. In the examples used in this document, the SAN has been divided into 4 LUNs. Each of the 5 servers in the cluster is capable of mounting all 4 LUNs. However, the same LUN should never be mounted on two nodes simultaneously. This document assumes that each block device is formatted using ext3. The Heartbeat software will insure that a given LUN is mounted in only one location at a time.

It is also important that the device naming be consistent across all nodes. For example, if node1 mounts /dev/fooa, then it should see the same data as if node2 were to mount /dev/fooa. Likewise for /dev/foob, etc.

2.1.3 Stonith

Heartbeat needs some mechanism to fence or stonith a failed node. One straightforward way to do this is to connect each server node to a network controllable power strip. That will allow any given server to send a command over the network to power off another server.

It is possible to configure PVFS2 and Heartbeat without a power control device. However, if you deploy this configuration for any purpose other than evaluation, then you run a very serious risk of data corruption. Without stonith, there is no way to guarantee that a failed node has completely shutdown and stopped accessing its storage device before failing over.

The example in this document is using an APC switched PDU (which allows commands to be sent via SNMP or ssh) as the power control device.

2.2 Software

This document assumes that you are using Hearbeat version 2.0.8, and PVFS2 version 2.6.x or greater. You may also wish to use example scripts included in the `examples/heartbeat` directory of the PVFS2 source tree.

2.3 Network

There are two special issues regarding the network configuration to be used with Heartbeat. First of all, you must allocate a multicast address to use for communication within the cluster nodes.

Secondly, you need to allocate an extra IP address and hostname for each active PVFS2 server. In the example that this document uses, we must allocate 4 extra IP addresses, along with 4 hostnames in DNS for those IP addresses. In this document, we will refer to these as "virtual addresses". Each active PVFS2 server will be configured to automatically bring up one of these virtual addresses to use for communication. If the node fails, then that IP address is migrated to another node so that clients will appear to communicate with the same server regardless of where it fails over to. It is important that you not use the primary IP address of each node for this purpose.

In the example in this document, we use 225.0.0.1 as the multicast address, node{1-5} as the normal node hostnames, and virtualnode{1-4} as the virtual hostnames.

3 Configuring PVFS2

Download, build, install, and configure PVFS2.

There are a few points to consider when configuring PVFS2:

- Use the virtual addresses when specifying meta servers and I/O servers
- Synchronize file data on every operation (necessary for consistency on failover)
- Synchronize meta data on every operation (necessary for consistency on failover)
- Use the `TCPBindSpecific` option (this allows multiple daemons to run on the same node if needed)
- Tune retry and timeout values appropriately for your system. This may depend on how long it takes for your power control device to safely shutdown a node.

```
<Defaults>
        ...
        ServerJobBMITimeoutSecs 30
        ServerJobFlowTimeoutSecs 30
        ClientJobBMITimeoutSecs 30
        ClientJobFlowTimeoutSecs 30
        ClientRetryLimit 5
        ClientRetryDelayMilliSecs 33000
        TCPBindSpecific yes
        ...
</Defaults>

<Aliases>
        Alias virtualnode1_tcp3334 tcp://virtualnode1:3334
        Alias virtualnode2_tcp3334 tcp://virtualnode2:3334
        Alias virtualnode3_tcp3334 tcp://virtualnode3:3334
        Alias virtualnode4_tcp3334 tcp://virtualnode4:3334
</Aliases>

<Filesystem>
        ...
        <MetaHandleRanges>
                Range virtualnode1_tcp3334 4-536870914
                Range virtualnode2_tcp3334 536870915-1073741825
                Range virtualnode3_tcp3334 1073741826-1610612736
                Range virtualnode4_tcp3334 1610612737-2147483647
        </MetaHandleRanges>
        <DataHandleRanges>
                Range virtualnode1_tcp3334 2147483648-2684354558
                Range virtualnode2_tcp3334 2684354559-3221225469
                Range virtualnode3_tcp3334 3221225470-3758096380
                Range virtualnode4_tcp3334 3758096381-4294967291
        </DataHandleRanges>
        <StorageHints>
                TroveSyncMeta yes
                TroveSyncData yes
                CoalescingHighWatermark 1
                CoalescingLowWatermark 1
        </StorageHints>
</Filesystem>
```

Figure 1: Example `pvfs2-fs.conf` file

Figure 1 shows one example of how to configure PVFS2. Only the parameters relevant to the Heartbeat scenario are shown.

Download, build, and install Heartbeat following the instructions on their web site. No special parameters or options are required. Do not start the Heartbeat service.

4 Configuring storage

Make sure that there is a block device allocated for each active server in the file system. Format each one with ext3. Do not create a PVFS2 storage space yet, but you can create subdirectories within each file system if you wish.

Confirm that each block device can be mounted from every node, and that the device names are consistent. Do this one node at a time. Never mount the same block device concurrently on two or more nodes.

```
$ dd if=/dev/urandom count=4 2>/dev/null | openssl dgst -sha1
dcdebc13c41977eac8cca0023266a8b16d234262

$ pvfs2-ha-heartbeat-configure.sh /etc/ha.d 225.0.0.1 \
  dcdebc13c41977eac8cca0023266a8b16d234262 \
  node1 node2 node3 node4 node5
```

Figure 2: Example `pvfs2-ha-heartbeat-configure.sh` commands

5 Distributing Heartbeat scripts

The scripts that are in the `examples/heartbeat` subdirectory may be installed to the following suggested locations on each server node:

- pvfs2-ha-heartbeat-configure.sh: /usr/bin
- apc*: /usr/bin
- baytech*: /usr/bin
- qla*: /usr/bin
- PVFS2: /usr/lib/ocf/resource.d/external/
- PVFS2-notify: /usr/lib/ocf/resource.d/external
- Filesystem-qla-monitor: /usr/lib/ocf/resource.d/external
- pvfs2-stonith-plugin: /usr/lib/stonith/plugins/external

6 Base Heartbeat configuration

This section describes how to configure the basic Heartbeat daemon parameters, which include an authentication key and a list of nodes that will participate in the cluster.

Begin by generating a random sha1 key, which is used to secure communication between the cluster nodes. Then run the pvfs2-ha-heartbeat-configure.sh script as shown in figure 2 on every node (both active and spare). You should use your multicast address as described in the network requirements, your own sha1 key, and a list of nodes (including spares) that will participate.

You can view the configuration file that this generates in /etc/ha.d/ha.cf. An example ha.cf file (with comments) is provided with the Heartbeat package if you wish to investigate how to add or change any settings.

7 CIB configuration

`Cluster Information Base` (CIB) is the the mechanism that Heartbeat 2.x uses to store information about the resources that are configured for high availability. The configuration is stored in an XML format and automatically synchronized across all of the cluster nodes.

It is possible to start the Heartbeat services and then configure the CIB, but it is simpler to begin with a populated XML file on all nodes.

`cib.xml.example` provides an example of a fully populated Heartbeat configuration with 5 nodes and 4 active PVFS2 servers. It also includes some optional components for completeness. Relevant portions of the XML file are outlined below.

This file should be modified to reflect your configuration, and then copied into /var/lib/crm/cib.xml on every node in the cluster (including spares).

7.1 crm_config

The `crm_config` portion of the CIB is used to set global parameters for Heartbeat. This includes behavioral settings (such as how to respond if quorum is lost) as well as tunable parameters (such as timeout values).

The options selected in this section should work well as a starting point, but you may refer to the Heartbeat documentation for more details.

7.2 nodes

The `nodes` section is empty on purpose. This will be filled in dynamically by the Heartbeat daemons.

7.3 resources and groups

The `resources` section describes all resources that the Heartbeat software needs to manage for failover purposes. This includes IP addresses, SAN mount points, and `pvfs2-server` processes. The resources are organized into groups, such as `server0`, to indicate that certain groups of resources should be treated as a single unit. For example, if a node were to fail, you cannot just migrate its `pvfs2-server` process. You must also migrate the associated IP address and SAN mount point at the same time. Groups also make it easier to start or stop all associated resources for a node with one unified command.

In the example `cib.xml`, there are 4 groups (server0 through server3). These represent the 4 active PVFS2 servers that will run on the cluster.

7.4 PVFS2-notify

The `PVFS2-notify` resources, such as `server0_notify`, are used as a mechanism to send alerts when a server process fails over to another node. This is provided by the `PVFS2-notify` script in the examples directory.

The use of a notify resource is entirely optional and may be omitted. This particular script is designed to take four parameters:

- `firsthost`: name of the node that the server group should normally run on
- `fsname`: arbitrary name for the PVFS2 file system
- `conf_dir`: location of notification configuration files
- `title`: component of the title for the notification

The `PVFS2-notify` script serves as an example for how one might implement a notification mechanism. However, it is incomplete on its own. This example relies on a secondary script called `fs-instance-alarm.pl` to send the actual notification. For example, one could implement a script that sends an email when a failure

occurs. The `conf_dir` parameter could be passed along to provide a location to read a configurable list of email addresses from.

`fs-instance-alarm.pl` is not provided with this example or documentation.

7.5 IPaddr

The `IPaddr` resources, such as `server0_address`, are used to indicate what virtual IP address should be used with each group. In this example, all IP addresses are allocated from a private range, but these should be replaced with IP addresses that are appropriate for use on your network. See the network requirements section for more details.

7.6 Filesystem

The `Filesystem` resources, such as `server0_fs`, are used to describe the shared storage block devices that serve as back end storage for PVFS2. This is where the PVFS2 storage space for each server will be created. In this example, the device names are `/dev/fooa1` through `/dev/food1`. They are each mounted on directories such as `/san_mounta1` through `/san_mountd1`. Please note that each device should be mounted on a different mount point to allow multiple `pvfs2-server` processes to operate on the same node without collision.

7.7 PVFS2

The `PVFS2` resources, such as `server0_daemon`, are used to describe each `pvfs2-server` process. This resource is provided by the PVFS2 script in the examples directory. The parameters to this resource are listed below:

- `fsconfig`: location of PVFS2 fs configuration file
- `serverconfig`: location of PVFS2 server configuration file
- `port`: TCP/IP port that the server will listen on (must match server configuration file)
- `ip`: IP address that the server will listen on (must match both the file system configuration file and the IPAddr resource)
- `pidfile`: Location where a pid file can be written

Also notice that there is a monitor operation associated with the PVFS2 resource. This will cause the `pvfs2-check-server` utility to be triggered periodically to make sure that the `pvfs2-server` process is not only running, but is correctly responding to PVFS2 protocol requests. This allows problems such as hung `pvfs2-server` processes to be treated as failure conditions.

Please note that the PVFS2 script provided in the examples will attempt to create a storage space for each server if it is not already present.

7.8 rsc_location

The `rsc_location` constraints, such as `run_server0`, are used to express a preference for where each resource group should run (if possible). It may be useful for administrative purposes to have the first server group default to run on the first node of your cluster, etc.

7.9 rsc_order

The `rsc_order` constraints, such as `server0_order_start_fs` can be used to dictate the order in which resources must be started or stopped. The resources are already organized into groups, but without ordering constraints, the resources within a group may be started in any order relative to each other. These constraints are necessary because a `pvfs2-server` process will not start properly if the IP address that it should listen on and the shared storage that it should use are not available yet.

7.10 pvfs2-stonith-plugin

The `pvfs2-stonith-plugin` resource is an example of how to configure a stonith device for use in Heartbeat. See the Heartbeat documentation for a list of officially supported devices.

In this example, the stonith device is setup as a clone, which means that there are N identical copies of the resource (one per node). This allows any node in the cluster to quickly send a stonith command if needed.

The `pvfs2-stonith-plugin` is provided by a script in the examples directories. It requires a parameter to specify the file system name, and a parameter to specify a configuration directory. This plugin is not complete by itself, however. It relies on three scripts to actually perform the stonith commands:

- `fs-power-control.pl`: used to send commands to control power to a node
- `fs-power-gethosts.pl`: used to print a list of nodes that can be controlled with this device
- `fs-power-monitor.pl`: used to monitor the stonith device and confirm that is available

These three stonith scripts are not provided with these examples. They may need to be specifically implemented for your environment. As an alternative, you can simply use one of the standard stonith devices that are supported by Heartbeat (see Heartbeat documentation for details).

The following scripts provide lower level examples of how to control an APC power strip (via SNMP or SSH) or a Baytech power strip (via SSH):

- `apc-switched-pdu-hybrid-control.pl`
- `apc-switched-pdu-hybrid-monitor.pl`
- `baytech-mgmt-control.pl`
- `baytech-mgmt-monitor.pl`

One approach to implementing power control would be to use the pvfs2-stonith-plugin device script and write `fs-power{control/monitor/gethosts}` scripts that can parse configuration files describing your cluster and send appropriate commands to the above provided APC and Baytech control scripts.

7.11 SAN monitoring

The example CIB configuration does not use this feature, but an additional resource script has been included that modifies the `Filesystem` resource to allow it to monitor SAN connectivity. This script is called `Filesystem-qla-monitor`. It requires that the nodes use QLogic fibre channel adapters and EMC PowerPath software for SAN connectivity. If this configuration is available, then this script can issue appropriate PowerPath commands periodically to confirm that there is connectivity between each node and its block device.

```
$ /etc/init.d/heartbeat start
$ # wait until all Heartbeat services started
$ crm_mon -r
```

Figure 3: Starting Heartbeat services

```
$ mount -t pvfs2 tcp://virtualnode1:3334/pvfs2-fs /mnt/pvfs2
```

Figure 4: Mounting PVFS2 file system

8 Starting Heartbeat

Once the CIB file is completed and installed in the correct location, then the Heartbeat services can be started on every node with the command in figure 3. The `crm_mon` command, when run with the arguments shown, will provide a periodically updated view of the state of each resource configured within Heartbeat. Check `/var/log/messages` if any of the groups fail to start.

9 Mounting the file system

Mounting PVFS2 with high availability is no different than mounting a normal PVFS2 file system, except that you must use the virtual hostname for the PVFS2 server rather than the primary hostname of the node. Figure 4 provides an example.

10 What happens during failover

The following example illustrates the steps that occur when a node fails:

1. Node2 (which is running a `pvfs2-server` on the virtualnode2 IP address) suffers a failure
2. Client node begins timeout/retry cycle
3. Heartbeat services running on remaining servers notice that node2 is not responding
4. After a timeout has elapsed, remaining servers reach a quorum and vote to treat node2 as a failed node
5. Node1 sends a stonith command to reset node2
6. Node2 either reboots or remains powered off (depending on nature of failure)
7. Once stonith command succeeds, node5 is selected to replace it
8. The virtualnode2 IP address, mount point, and `pvfs2-server` service are started on node5
9. Client node retry eventually succeeds, but now the network traffic is routed to node5

11 Controlling Heartbeat

The Heartbeat software comes with a wide variety of tools for managing resources. The following are a few useful examples:

- `cibadmin -Q`: Display the current CIB information
- `crm_mon -r -1`: Display the current resource status
- `crm_standby`: Used to manually take a node in an out of standby mode. This can be used to take a node offline for maintenance without a true failure event.
- `crm_resource`: Modify resource information. For example, `crm_resource -r server0 -p target_role -v stopped` will stop a particular resource group.
- `crm_verify`: can be used to confirm if the CIB information is valid and consistent