# PVFS High Availability Clustering using Heartbeat 2.0

2008

## Contents

# 1 Introduction

This document describes how to configure PVFS for high availability using Heartbeat version 2.x from www.linux-ha.org.

The combination of PVFS and Heartbeat can support an arbitrary number of active server nodes 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, PVFS 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.

Heartbeat can be configured to monitor IP connectivity, storage hardware connectivity, and responsiveness of the PVFS daemons. Failure of any of these components will trigger a node level failover event. PVFS clients will transparently continue operation following a failover.

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

# 2 Requirements

## 2.1 Hardware

### 2.1.1 Nodes

Any number of nodes may be configured, although you need at least three total in order to tolerate a failure. 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 1 spare node, for a total of 5 nodes. Heartbeat has been tested with up to 16 nodes in configurations similar to the one outlined in this document.

### 2.1.2 Storage

A shared storage device is required. The storage must be configured to allocate a separate block device to each PVFS daemon, and all nodes (including spares) 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 nodes in the cluster is capable of mounting all 4 LUNs. The Heartbeat software will insure that a given LUN is mounted in only one location at a time.

Each block device should be formatted with a local file system. This document assumes the use of ext3. Unique labels should be set on each file system (for example, using the `-L` argument to `mke2fs` or `tune2fs`). This will allow the block devices to be mounted by consistent labels regardless of how Linux assigned device file names to the devices.

### 2.1.3 Stonith

Heartbeat needs some mechanism to fence or stonith a failed node. Two popular ways to do this are either to use IPMI or a network controllable power strip. Each node needs to have a mechanism available to reset any other node in the cluster. The example configuration in this document uses IPMI.

It is possible to configure PVFS 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.

### 2.2 Software

This document assumes that you are using Heartbeat version 2.1.3, and PVFS version 2.7.x or greater. You may also wish to use example scripts included in the `examples/heartbeat` directory of the PVFS 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 PVFS daemon. 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" or "virtual hostnames". Each active PVFS 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, virtual{1-4} as the virtual hostnames, and 192.168.0.{1-4} as the virtual addresses.

Note that the virtual addresses must be on the same subnet as the true IP addresses for the nodes.

## 3 Configuring PVFS

Download, build, and install PVFS on all server nodes. Configure PVFS for use on each of the active nodes.

There are a few points to consider when configuring PVFS:

- Use the virtual hostnames 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). Coalescing is allowed.
- Use the `TCPBindSpecific` option (this allows multiple daemons to run on the same node using different virtual addresses)
- 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.

An example PVFS configuration is shown below including the sections relevant to Heartbeat:

```
<Defaults>
        ...
        ServerJobBMITimeoutSecs 30
        ServerJobFlowTimeoutSecs 30
        ClientJobBMITimeoutSecs 30
```

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

<Aliases>
        Alias virtual1_tcp3334 tcp://virtual1:3334
        Alias virtual2_tcp3334 tcp://virtual2:3334
        Alias virtual3_tcp3334 tcp://virtual3:3334
        Alias virtual4_tcp3334 tcp://virtual4:3334
</Aliases>

<Filesystem>
        ...
        <MetaHandleRanges>
                Range virtual1_tcp3334 4-536870914
                Range virtual2_tcp3334 536870915-1073741825
                Range virtual3_tcp3334 1073741826-1610612736
                Range virtual4_tcp3334 1610612737-2147483647
        </MetaHandleRanges>
        <DataHandleRanges>
                Range virtual1_tcp3334 2147483648-2684354558
                Range virtual2_tcp3334 2684354559-3221225469
                Range virtual3_tcp3334 3221225470-3758096380
                Range virtual4_tcp3334 3758096381-4294967291
        </DataHandleRanges>
        <StorageHints>
                TroveSyncMeta yes
                TroveSyncData yes
        </StorageHints>
```

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 PVFS 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 using the file system label. Do this one node at a time. Never mount the same block device concurrently on two nodes.

## 5 Configuring stonith

Make sure that your stonith device is accessible and responding from each node in the cluster. For the IPMI stonith example used in this document, this means confirming that `ipmitool` is capable of monitoring each node. Each node will have its own IPMI IP address, username, and password.

```
$ ipmitool -I lan -U Administrator -P password -H 192.168.0.10 power status
Chassis Power is on
```

## 6 Distributing Heartbeat scripts

The PVFS2 resource script must be installed and set as runnable on every cluster node as follows:

```
$ mkdir -p /usr/lib/ocf/resource.d
$ cp examples/heartbeat/PVFS2 /usr/lib/ocf/resource.d/external/
$ chmod a+x /usr/lib/ocf/resource.d/external/PVFS2
```

## 7 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 on every node (both active and spare) as shown below. Make sure to use the multicast address, sha1 key, and list of nodes (including spares) appropriate for your environment.

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

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

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.

## 8 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 PVFS servers. Relevant portions of the XML file are outlined below.

This file should be modified to reflect your configuration. You can test the validity of the XML with the following commands before installing it. The former checks generic XML syntax, while the latter performs Heartbeat specific checking:

```
$ xmllint --noout cib.xml
$ crm_verify -x cib.xml
```

Once your XML is filled in correctly, it must be copied into the correct location (with correct ownership) on each node in the cluster:

```
$ mkdir -p /var/lib/heartbeat/crm
$ cp cib.xml /var/lib/heartbeat/crm
$ chown -R hacluster:haclient /var/lib/heartbeat/crm
```

Please note that once Heartbeat has been started, it is no longer legal to modify cib.xml by hand. See the `cibadmin` command line tool and Heartbeat information on making modifications to existing or online CIB configurations.

### 8.1 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.

### 8.2 nodes

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

### 8.3 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 PVFS servers that will run on the cluster.

### 8.4 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.

## 8.5 Filesystem

The `Filesystem` resources, such as `server0_fs`, are used to describe the shared storage block devices that serve as back end storage for PVFS. This is where the PVFS storage space for each server will be created. In this example, the device names are labeled as `label0` through `label3`. They are each mounted on directories such as `/san_mount0` through `/san_mount3`. 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. The file system type can be changed to reflect the use of alternative underlying file systems.

## 8.6 PVFS

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 PVFS fs 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
- `alias`: alias to identify this PVFS daemon instance

Also notice that there is a monitor operation associated with the PVFS 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 PVFS 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 on startup for each server if it is not already present.

## 8.7 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, for example. Otherwise the placement will be left up to Heartbeat.

## 8.8 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 until its IP address and storage are available.

### 8.9 stonith

The `external/ipmi` stonith device is used in this example. Please see the Heartbeat documentation for instructions on configuring other types of devices.

There is one IPMI stonith device for each node. The attributes for that resources specify which node is being controlled, and the username, password, and IP address of corresponding IPMI device.

## 9 Starting Heartbeat

Once the CIB file is completed and installed in the correct location, then the Heartbeat services can be started on every node. 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.

```
$ /etc/init.d/heartbeat start
$ # wait a few minutes for heartbeat services to start
$ crm_mon -r
```

## 10 Mounting the file system

Mounting PVFS with high availability is no different than mounting a normal PVFS file system, except that you must use the virtual hostname for the PVFS server rather than the primary hostname of the node:

```
$ mount -t pvfs2 tcp://virtual1:3334/pvfs2-fs /mnt/pvfs2
```

## 11 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 virtual2 IP address) suffers a failure
2. Client node begins timeout/retry cycle
3. Heartbeat services running on remaining nodes notice that node2 is not responding
4. After a timeout has elapsed, remaining nodes 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 virtual2 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

## 12 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

## 13 Additional examples

The `examples/heartbeat/hardware-specific` directory contains additional example scripts that may be helpful in some scenarios:

- `pvfs2-stonith-plugin`: An example stonith plugin that can use an arbitrary script to power off nodes. May be used (for example) with the `apc*` and `baytech*` scripts to control remote controlled power strips if the scripts provided by Heartbeat are not sufficient.
- `Filesystem-qla-monitor`: A modified version of the standard FileSystem resource that uses the `qla-monitor.pl` script to provide additional monitoring capability for QLogic fibre channel cards.
- `PVFS2-notify`: An example of a dummy resource that could be added to the configuration to perform additional logging or notification steps on startup.