MySQL and DRBD Guide

MySQL and DRBD Guide

Abstract

This is the MySQL and DRBD extract from the MySQL Reference Manual.

Document generated on: 2009-06-02 (revision: 15161)

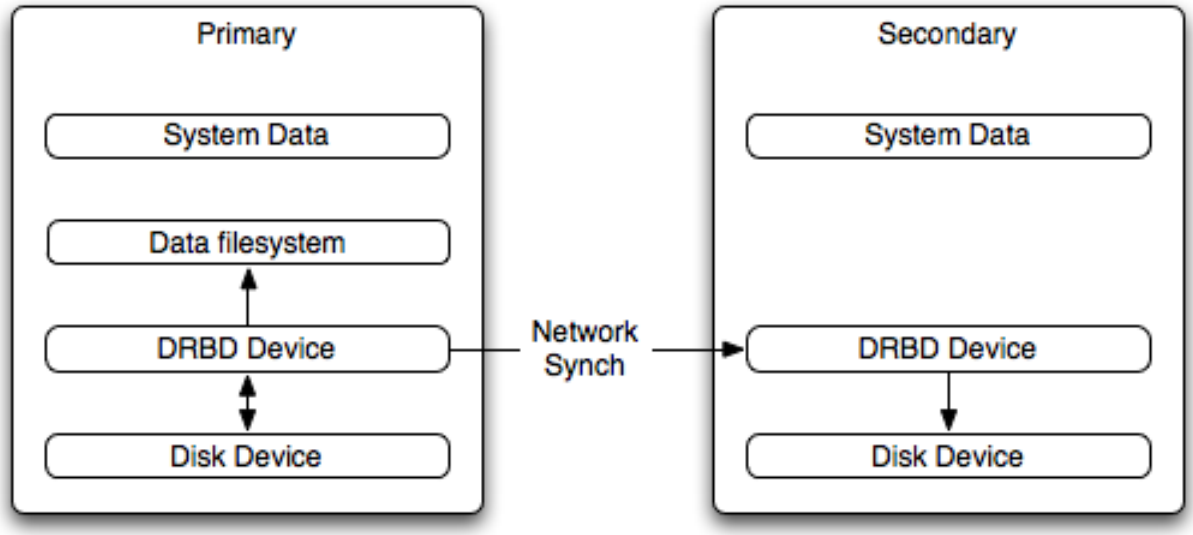
Using MySQL with DRBD

The Distributed Replicated Block Device (DRBD) is a Linux Kernel module that constitutes a distributed storage system. You can use DRBD to share block devices between Linux servers and, in turn, share file systems and data.

DRBD implements a block device which can be used for storage and which is replicated from a primary server to one or more secondary servers. The distributed block device is handled by the DRBD service. Writes to the DRBD block device are distributed among the servers. Each DRBD service writes the information from the DRBD block device to a local physical block device (hard disk).

On the primary data writes are written both to the underlying physical block device and distributed to the secondary DRBD services. On the secondary, the writes received through DRBD and written to the local physical block device. On both the primary and the secondary, reads from the DRBD block device are handled by the underlying physical block device. The information is shared between the primary DRBD server and the secondary DRBD server synchronously and at a block level, and this means that DRBD can be used in high-availability solutions where you need failover support.

Figure 1. DRBD Architecture Overview



When used with MySQL, DRBD can be used to ensure availability in the event of a failure. MySQL is configured to store information on the DRBD block device, with one server acting as the primary and a second machine available to operate as an immediate replacement in the event of a failure.

For automatic failover support you can combine DRBD with the Linux Heartbeat project, which will manage the interfaces on the two servers and automatically configure the secondary (passive) server to replace the primary (active) server in the event of a failure. You can also combine DRBD with MySQL Replication to provide both failover and scalability within your MySQL environment.

For information on how to configure DRBD and MySQL, including Heartbeat support, see Chapter 1, *Configuring the DRBD Environment*.

An FAQ for using DRBD and MySQL is available. See Chapter 5, *MySQL 5.1 FAQ — MySQL, DRBD, and Heartbeat*.

> **Note**
>
> Because DRBD is a Linux Kernel module it is currently not supported on platforms other than Linux.

Chapter 1. Configuring the DRBD Environment

To set up DRBD, MySQL and Heartbeat you need to follow a number of steps that affect the operating system, DRBD and your MySQL installation.

Before starting the installation process, you should be aware of the following information, terms and requirements on using DRBD:

- DRBD is a solution for enabling high-availability, and therefore you need to ensure that the two machines within your DRBD setup are as identically configured as possible so that the secondary machine can act as a direct replacement for the primary machine in the event of system failure.
- DRBD works through two (or more) servers, each called a *node*
- The node that contains the primary data, has read/write access to the data, and in an HA environment is the currently active node is called the *primary*.
- The server to which the data is replicated is referred as *secondary*.
- A collection of nodes that are sharing information are referred to as a *DRBD cluster*.
- For DRBD to operate you must have a block device on which the information can be stored on *each* DRBD node. The *lower level* block device can be a physical disk partition, a partition from a volume group or RAID device or any other block device.

  Typically you use a spare partition on which the physical data will be stored . On the primary node, this disk will hold the raw data that you want replicated. On the secondary nodes, the disk will hold the data replicated to the secondary server by the DRBD service. Ideally, the size of the partition on the two DRBD servers should be identical, but this is not necessary as long as there is enough space to hold the data that you want distributed between the two servers.
- For the distribution of data to work, DRBD is used to create a logical block device that uses the lower level block device for the actual storage of information. To store information on the distributed device, a file system is created on the DRBD logical block device.
- When used with MySQL, once the file system has been created, you move the MySQL data directory (including InnoDB data files and binary logs) to the new file system.
- When you set up the secondary DRBD server, you set up the physical block device and the DRBD logical block device that will store the data. The block device data is then copied from the primary to the secondary server.

The overview for the installation and configuration sequence is as follows:

1. First you need to set up your operating system and environment. This includes setting the correct host name, updating the system and preparing the available packages and software required by DRBD, and configuring a physical block device to be used with the DRBD block device. See Section 1.1, “Setting Up Your Operating System for DRBD”.
2. Installing DRBD requires installing or compiling the DRBD source code and then configuring the DRBD service to set up the block devices that will be shared. See Section 1.2, “Installing and Configuring DRBD”.
3. Once DRBD has been configured, you must alter the configuration and storage location of the MySQL data. See Chapter 2, *Configuring MySQL for DRBD*.

You may optionally want to configure high availability using the Linux Heartbeat service. See Chapter 4, *Using Linux HA Heartbeat*, for more information.

1.1. Setting Up Your Operating System for DRBD

To set your Linux environment for using DRBD there are a number of system configuration steps that you must follow.

- Make sure that the primary and secondary DRBD servers have the correct host name, and that the host names are unique. You can verify this by using the `uname` command:

  ```
  shell> uname -n
  drbd-one
  ```

  If the host name is not set correctly, edit the appropriate file (usually `/etc/sysconfig/network`, `/etc/hostname`, or

`/etc/conf.d/hostname`) and set the name correctly.

- Each DRBD node must have a unique IP address. Make sure that the IP address information is set correctly within the network configuration and that the host name and IP address has been set correctly within the `/etc/hosts` file.
- Although you can rely on the DNS or NIS system for host resolving, in the event of a major network failure these services may not be available. If possible, add the IP address and host name of each DRBD node into the /etc/hosts file for each machine. This will ensure that the node information can always be determined even if the DNS/NIS servers are unavailable.
- As a general rule, the faster your network connection the better. Because the block device data is exchanged over the network, everything that will be written to the local disk on the DRBD primary will also be written to the network for distribution to the DRBD secondary.

  For tips on configuring a faster network connection see Chapter 3, *Optimizing Performance and Reliability*.
- You must have a spare disk or disk partition that you can use as the physical storage location for the DRBD data that will be replicated. You do not have to have a complete disk available, a partition on an existing disk is acceptable.

  If the disk is unpartitioned, partition the disk using `fdisk`, `cfdisk` or other partitioning solution. Do not create a file system on the new partition.

  Remember that you must have a physical disk available for the storage of the replicated information on each DRBD node. Ideally the partitions that will be used on each node should be of an identical size, although this is not strictly necessary. Do, however, ensure that the physical partition on the DRBD secondary is at least as big as the partitions on the DRBD primary node.
- If possible, upgrade your system to the latest available Linux kernel for your distribution. Once the kernel has been installed, you must reboot to make the kernel active. To use DRBD you will also need to install the relevant kernel development and header files that are required for building kernel modules. Platform specification information for this is available later in this section.

Before you compile or install DRBD, you must make sure the following tools and files are in place:

- Kernel header files
- Kernel source files
- GCC Compiler
- `glib 2`
- `flex`

Here are some operating system specific tips for setting up your installation:

- **Tips for Red Hat (including CentOS and Fedora)**:

  Use `up2date` or `yum` to update and install the latest kernel and kernel header files:

  ```
  root-shell> up2date kernel-smp-devel kernel-smp
  ```

  Reboot. If you are going to build DRBD from source, then update your system with the required development packages:

  ```
  root-shell> up2date glib-devel openssl-devel libgcrypt-devel glib2-devel \
  pkgconfig ncurses-devel rpm-build rpm-devel redhat-rpm-config gcc \
  gcc-c++ bison flex gnutls-devel lm_sensors-devel net-snmp-devel \
  python-devel bzip2-devel libselinux-devel perl-DBI
  ```

  If you are going to use the pre-built DRBD RPMs:

  ```
  root-shell> up2date gnutls lm_sensors net-snmp ncurses libgcrypt glib2 openssl glib
  ```

- **Tips for Debian, Ubuntu, Kubuntu**:

  Use `apt-get` to install the kernel packages

  ```
  root-shell> apt-get install linux-headers linux-image-server
  ```

If you are going to use the pre-built Debian packages for DRBD then you should not need any additional packages.

If you want to build DRBD from source, you will need to use the following command to install the required components:

```
root-shell> apt-get install devscripts flex bison build-essential \
dpkg-dev kernel-package debconf-utils dpatch debhelper \
libnet1-dev e2fslibs-dev libglib2.0-dev automake1.9 \
libgnutls-dev libtool libltdl3 libltdl3-dev
```

- **Tips for Gentoo**:

  Gentoo is a source based Linux distribution and therefore many of the source files and components that you will need are either already installed or will be installed automatically by `emerge`.

  To install DRBD 0.8.x, you must unmask the `sys-cluster/drbd` build by adding the following line to `/etc/portage/package.keywords`:

  ```
  sys-cluster/drbd ~x86
  sys-cluster/drbd-kernel ~x86
  ```

  If your kernel does not already have the userspace to kernelspace linker enabled, then you will need to rebuild the kernel with this option. The best way to do this is to use `genkernel` with the `--menuconfig` option to select the option and then re-build the kernel. For example, at the command line as `root`:

  ```
  root-shell> genkernel --menuconfig all
  ```

  Then through the menu options, select DEVICE DRIVERS, CONNECTOR - UNIFIED USERSPACE <-> KERNELSPACE LINKER and finally press 'y' or 'space' to select the CONNECTOR - UNIFIED USERSPACE <-> KERNELSPACE LINKER option. Then exit the menu configuration. The kernel will be rebuilt and installed. If this is a new kernel, make sure you update your bootloader accordingly. Now reboot to enable the new kernel.

1.2. Installing and Configuring DRBD

To install DRBD you can choose either the pre-built binary installation packages or you can use the source packages and build from source. If you want to build from source you must have installed the source and development packages.

If you are installing using a binary distribution then you must ensure that the kernel version number of the binary package matches your currently active kernel. You can use `uname` to find out this information:

```
shell> uname -r
2.6.20-gentoo-r6
```

Once DRBD has been built and installed, you need to edit the `/etc/drbd.conf` file and then run a number of commands to build the block device and set up the replication.

Although the steps below are split into those for the primary node and the secondary node, it should be noted that the configuration files for all nodes should be identical, and many of the same steps have to be repeated on each node to enable the DRBD block device.

Building from source:

To download and install from the source code:

1. Download the source code.
2. Unpack the package:

   ```
   shell> tar zxf drbd-8.3.0.tar.gz
   ```

3. Change to the extracted directory, and then run `make` to build the DRBD driver:

   ```
   shell> cd drbd-8.3.0
   shell> make
   ```

4. Install the kernel driver and commands:

   ```
   shell> make install
   ```

Binary Installation:

- **SUSE Linux Enterprise Server (SLES)**

  For SUSE, use `yast`:

  ```
  shell> yast -i drbd
  ```

  Alternatively:

  ```
  shell> rug install drbd
  ```

- **Debian**

  Use `apt-get` to install the modules. You do not need to install any other components.

  ```
  shell> apt-get install drbd8-utils drbd8-module
  ```

- **Debian 3.1 and 4.0**

  You must install the `module-assistant` to build the DRBD kernel module, in addition to the DRBD components.

  ```
  shell> apt-get install drbd0.7-utils drbd0.7-module-source \
    build-essential module-assistant
  shell> module-assistant auto-install drbd0.7
  ```

- **CentOS**

  DRBD can be installed using yum:

  ```
  shell> yum install drbd kmod-drbd
  ```

- **Ubuntu**

  You must enable the universe component for your preferred Ubuntu mirror in `/etc/apt/sources.list`, and then issue these commands:

  ```
  shell> apt-get update
  shell> apt-get install drbd8-utils drbd8-module-source \
    build-essential module-assistant
  shell> module-assistant auto-install drbd8
  ```

- **Gentoo**

  You can now `emerge` DRBD 0.8.x into your Gentoo installation:

  ```
  root-shell> emerge drbd
  ```

  Once `drbd` has been downloaded and installed, you need to decompress and copy the default configuration file from `/usr/share/doc/drbd-8.0.7/drbd.conf.bz2` into `/etc/drbd.conf`.

1.3. Setting Up a DRBD Primary Node

To set up a DRBD primary node you need to configure the DRBD service, create the first DRBD block device and then create a file system on the device so that you can store files and data.

The DRBD configuration file `/etc/drbd.conf` defines a number of parameters for your DRBD configuration, including the frequency of updates and block sizes, security information and the definition of the DRBD devices that you want to create.

The key elements to configure are the `on` sections which specify the configuration of each node.

To follow the configuration, the sequence below shows only the changes from the default `drbd.conf` file. Configurations within the file can be both global or tied to specific resource.

1. Set the synchronization rate between the two nodes. This is the rate at which devices are synchronized in the background after a disk failure, device replacement or during the initial setup. You should keep this in check compared to the speed of your net-

work connection. Gigabit Ethernet can support up to 125 MB/second, 100Mbps Ethernet slightly less than a tenth of that (12MBps). If you are using a shared network connection, rather than a dedicated, then you should gauge accordingly.

To set the synchronization rate, edit the `rate` setting within the `syncer` block:

```
syncer {
    rate 10M;
}
```

You may additionally want to set the `al-extents` parameter. The default for this parameter is 257.

For more detailed information on synchronization, the effects of the synchronization rate and the effects on network performance, see Section 3.2, “Optimizing the Synchronization Rate”.

2. Set up some basic authentication. DRBD supports a simple password hash exchange mechanism. This helps to ensure that only those hosts with the same shared secret are able to join the DRBD node group.

```
cram-hmac-alg â##sha1â##;
shared-secret "shared-string";
```

3. Now you must configure the host information. Remember that you must have the node information for the primary and secondary nodes in the `drbd.conf` file on each host. You need to configure the following information for each node:

   - `device` — the path of the logical block device that will be created by DRBD.
   - `disk` — the block device that will be used to store the data.
   - `address` — the IP address and port number of the host that will hold this DRBD device.
   - `meta-disk` — the location where the metadata about the DRBD device will be stored. You can set this to `internal` and DRBD will use the physical block device to store the information, by recording the metadata within the last sections of the disk. The exact size will depend on the size of the logical block device you have created, but it may involve up to 128MB.

   A sample configuration for our primary server might look like this:

```
on drbd-one {
device /dev/drbd0;
disk /dev/hdd1;
address 192.168.0.240:8888;
meta-disk internal;
}
```

   The `on` configuration block should be repeated for the secondary node (and any further) nodes:

```
on drbd-two {
device /dev/drbd0;
disk /dev/hdd1;
address 192.168.0.241:8888;
meta-disk internal;
}
```

   The IP address of each `on` block must match the IP address of the corresponding host. Do not set this value to the IP address of the corresponding primary or secondary in each case.

4. Before starting the primary node, you should create the metadata for the devices:

```
root-shell> drbdadm create-md all
```

5. You are now ready to start DRBD:

```
root-shell> /etc/init.d/drbd start
```

   DRBD should now start and initialize, creating the DRBD devices that you have configured.

6. DRBD creates a standard block device - to make it usable, you must create a file system on the block device just as you would with any standard disk partition. Before you can create the file system, you must mark the new device as the primary device (i.e. where the data will be written and stored), and initialize the device. Because this is a destructive operation, you must specify the command line option to overwrite the raw data:

```
root-shell> drbdadm -- --overwrite-data-of-peer primary all
```

If you are using a version of DRBD 0.7.x or earlier, then you need to use a different command-line option:

```
root-shell> drbdadm -- --do-what-I-say primary all
```

Now create a file system using your chosen file system type:

```
root-shell> mkfs.ext3 /dev/drbd0
```

7. You can now mount the file system and if necessary copy files to the mount point:

```
root-shell> mkdir /mnt/drbd
root-shell> mount /dev/drbd0 /mnt/drbd
root-shell> echo "DRBD Device" >/mnt/drbd/samplefile
```

Your primary node is now ready to use. You should now configure your secondary node or nodes.

1.4. Setting Up a DRBD Secondary Node

The configuration process for setting up a secondary node is the same as for the primary node, except that you do not have to create the file system on the secondary node device, as this information will automatically be transferred from the primary node.

To set up a secondary node:

1. Copy the `/etc/drbd.conf` file from your primary node to your secondary node. It should already contain all the information and configuration that you need, since you had to specify the secondary node IP address and other information for the primary node configuration.
2. Create the DRBD metadata on the underlying disk device:

```
root-shell> drbdadm create-md all
```

3. Start DRBD:

```
root-shell> /etc/init.d/drbd start
```

Once DRBD has started, it will start the copy the data from the primary node to the secondary node. Even with an empty file system this will take some time, since DRBD is copying the block information from a block device, not simply copying the file system data.

You can monitor the progress of the copy between the primary and secondary nodes by viewing the output of `/proc/drbd`:

```
root-shell> cat /proc/drbd
version: 8.0.4 (api:86/proto:86)
SVN Revision: 2947 build by root@drbd-one, 2007-07-30 16:43:05
 0: cs:SyncSource st:Primary/Secondary ds:UpToDate/Inconsistent C r---
    ns:252284 nr:0 dw:0 dr:257280 al:0 bm:15 lo:0 pe:7 ua:157 ap:0
        [==>.................] sync'ed: 12.3% (1845088/2097152)K
        finish: 0:06:06 speed: 4,972 (4,580) K/sec
        resync: used:1/31 hits:15901 misses:16 starving:0 dirty:0 changed:16
        act_log: used:0/257 hits:0 misses:0 starving:0 dirty:0 changed:0
```

You can monitor the synchronization process by using the `watch` command to run the command at specific intervals:

```
root-shell> watch -n 10 'cat /proc/drbd'
```

1.5. Monitoring DRBD Device

Once the primary and secondary machines are configured and synchronized, you can get the status information about your DRBD device by viewing the output from `/proc/drbd`:

```
root-shell> cat /proc/drbd
version: 8.0.4 (api:86/proto:86)
SVN Revision: 2947 build by root@drbd-one, 2007-07-30 16:43:05
 0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:2175704 nr:0 dw:99192 dr:2076641 al:33 bm:128 lo:0 pe:0 ua:0 ap:0
        resync: used:0/31 hits:134841 misses:135 starving:0 dirty:0 changed:135
        act_log: used:0/257 hits:24765 misses:33 starving:0 dirty:0 changed:33
```

The first line provides the version/revision and build information.

The second line starts the detailed status information for an individual resource. The individual field headings are as follows:

- cs — connection state
- st — node state (local/remote)
- ld — local data consistency
- ds — data consistency
- ns — network send
- nr — network receive
- dw — disk write
- dr — disk read
- pe — pending (waiting for ack)
- ua — unack'd (still need to send ack)
- al — access log write count

In the previous example, the information shown indicates that the nodes are connected, the local node is the primary (because it is listed first), and the local and remote data is up to date with each other. The remainder of the information is statistical data about the device, and the data exchanged that kept the information up to date.

You can also get the status information for DRBD by using the startup script with the `status` option:

```
root-shell> /etc/init.d/drbd status
 * status:  started
 * drbd driver loaded OK; device status: ...
version: 8.3.0 (api:88/proto:86-89)
GIT-hash: 9ba8b93e24d842f0dd3fb1f9b90e8348ddb95829 build by root@gentoo1.vmbear, 2009-03-14 23:00:06
 0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r---
    ns:0 nr:0 dw:0 dr:8385604 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

The information and statistics are the same.

1.6. Managing your DRBD Installation

For administration, the main command is `drbdadm`. There are a number of commands supported by this tool the control the connectivity and status of the DRBD devices.

> **Note**
>
> For convenience, a bash completion script is available. This will provide tab completion for options to `drbdadm`. The file `drbdadm.bash_completion` can be found within the standard DRBD source package within the `scripts` directory. To enable, copy the file to `/etc/bash_completion.d/drbdadm`. You can load it manually by using:
>
> ```
> shell> source /etc/bash_completion.d/drbdadm
> ```

The most common commands are those to set the primary/secondary status of the local device. You can manually set this information for a number of reasons, including when you want to check the physical status of the secondary device (since you cannot mount a DRBD device in primary mode), or when you are temporarily moving the responsibility of keeping the data in check to a different machine (for example, during an upgrade or physical move of the normal primary node). You can set state of all local device to be the primary using this command:

```
root-shell> drbdadm primary all
```

Or switch the local device to be the secondary using:

```
root-shell> drbdadm secondary all
```

To change only a single DRBD resource, specify the resource name instead of `all`.

You can temporarily disconnect the DRBD nodes:

```
root-shell> drbdadm disconnect all
```

Reconnect them using `connect`:

```
root-shell> drbdadm connect all
```

For other commands and help with `drbdadm` see the DRBD documentation.

1.7. Additional DRBD Configuration Options

Additional options you may want to configure:

- `protocol` — specifies the level of consistency to be used when information is written to the block device. The option is similar in principle to the `innodb_flush_log_at_trx_commit` option within MySQL. Three levels are supported:
  - `A` — data is considered written when the information reaches the TCP send buffer and the local physical disk. There is no guarantee that the data has been written to the remote server or the remote physical disk.
  - `B` — data is considered written when the data has reached the local disk and the remote node's network buffer. The data has reached the remote server, but there is no guarantee it has reached the remote server's physical disk.
  - `C` — data is considered written when the data has reached the local disk and the remote node's physical disk.

  The preferred and recommended protocol is C, as it is the only protocol which ensures the consistency of the local and remote physical storage.
- `size` — if you do not want to use the entire partition space with your DRBD block device then you can specify the size of the DRBD device to be created. The size specification can include a quantifier. For example, to set the maximum size of the DRBD partition to 1GB you would use:

  ```
  size 1G;
  ```

With the configuration file suitably configured and ready to use, you now need to populate the lower-level device with the metadata information, and then start the DRBD service.

Chapter 2. Configuring MySQL for DRBD

Once you have configured DRBD and have an active DRBD device and file system, you can configure MySQL to use the chosen device to store the MySQL data.

When performing a new installation of MySQL, you can either select to install MySQL entirely onto the DRBD device, or just configure the data directory to be located on the new file system.

In either case, the files and installation must take place on the primary node, because that is the only DRBD node on which you can mount the DRBD device file system as read/write.

You should store the following files and information on your DRBD device:

- MySQL data files, including the binary log, and InnoDB data files.
- MySQL configuration file (`my.cnf`).

To set up MySQL to use your new DRBD device and file system:

1. If you are migrating an existing MySQL installation, stop MySQL:

   ```
   shell> mysqladmin shutdown
   ```

2. Copy the `my.cnf` onto the DRBD device. If you are not already using a configuration file, copy one of the sample configuration files from the MySQL distribution.

   ```
   root-shell> mkdir /mnt/drbd/mysql
   root-shell> cp /etc/my.cnf /mnt/drbd/mysql
   ```

3. Copy your MySQL data directory to the DRBD device and mounted file system.

   ```
   root-shell> cp -R /var/lib/mysql /drbd/mysql/data
   ```

4. Edit the configuration file to reflect the change of directory by setting the value of the `datadir` option. If you have not already enabled the binary log, also set the value of the `log-bin` option.

   ```
   datadir = /drbd/mysql/data
     log-bin = mysql-bin
   ```

5. Create a symbolic link from `/etc/my.cnf` to the new configuration file on the DRBD device file system.

   ```
   root-shell> ln -s /drbd/mysql/my.cnf /etc/my.cnf
   ```

6. Now start MySQL and check that the data that you copied to the DRBD device file system is present.

   ```
   root-shell> /etc/init.d/mysql start
   ```

Your MySQL data should now be located on the file system running on your DRBD device. The data will be physically stored on the underlying device that you configured for the DRBD device. Meanwhile, the content of your MySQL databases will be copied to the secondary DRBD node.

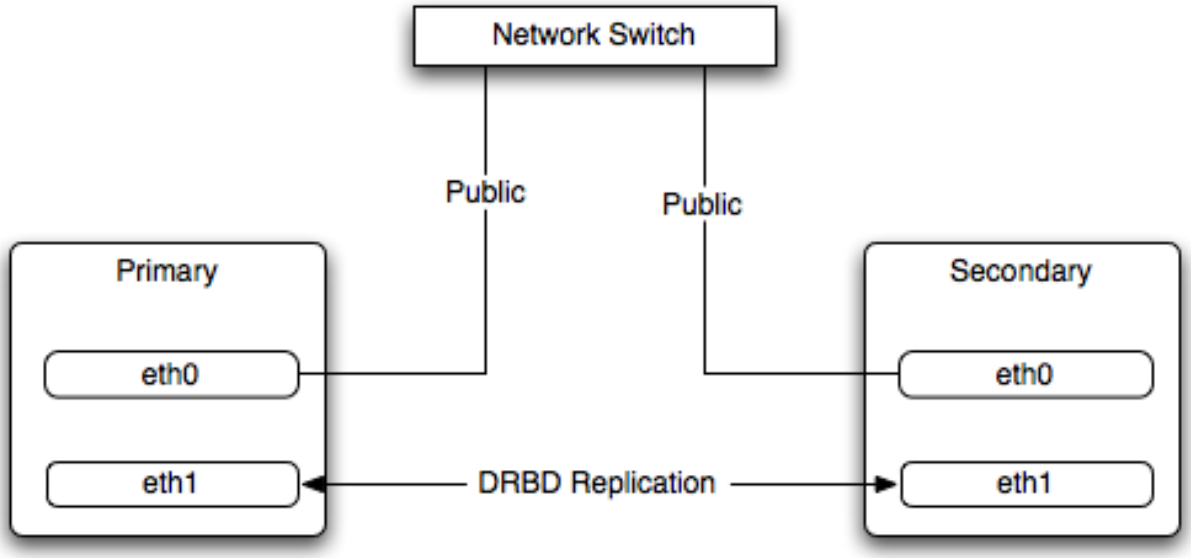
Note that you cannot access the information on your secondary node, as a DRBD device working in secondary mode is not available for use.

Chapter 3. Optimizing Performance and Reliability

Because of the nature of the DRBD system, the critical requirements are for a very fast exchange of the information between the two hosts. To ensure that your DRBD setup is available to switch over in the event of a failure as quickly as possible, you must transfer the information between the two hosts using the fastest method available.

Typically, a dedicated network circuit should be used for exchanging DRBD data between the two hosts. You should then use a separate, additional, network interface for your standard network connection. For an example of this layout, see Figure 3.1, “DRBD Architecture Using Separate Network Interfaces”.

Figure 3.1. DRBD Architecture Using Separate Network Interfaces



The dedicated DRBD network interfaces should be configured to use a non-routed TCP/IP network configuration. For example, you might want to set the primary to use 192.168.0.1 and the secondary 192.168.0.2. These networks and IP addresses should not be part of normal network subnet.

> **Note**
>
> The preferred setup, whenever possible, is to use a direct cable connection (using a crossover cable with Ethernet, for example) between the two machines. This eliminates the risk of loss of connectivity due to switch failures.

3.1. Using Bonded Ethernet Network Interfaces

For a set-up where there is a high-throughput of information being written, you may want to use bonded network interfaces. This is where you combine the connectivity of more than one network port, increasing the throughput linearly according to the number of bonded connections.

Bonding also provides an additional benefit in that with multiple network interfaces effectively supporting the same communications channel, a fault within a single network interface in a bonded group does not stop communication. For example, imagine you have a bonded setup with four network interfaces providing a single interface channel between two DRBD servers. If one network interface fails, communication can continue on the other three without interruption, although it will be at a lower speed

To enable bonded connections you must enable bonding within the kernel. You then need to configure the module to specify the bonded devices and then configure each new bonded device just as you would a standard network device:

- To configure the bonded devices, you need to edit the `/etc/modprobe.conf` file (RedHat) or add a file to the `/etc/modprobe.d` directory.. In each case you will define the parameters for the kernel module. First, you need to specify each bonding device:

```
alias bond0 bonding
```

You can then configure additional parameters for the kernel module. Typical parameters are the `mode` option and the `miimon` option.

The `mode` option specifies how the network interfaces are used. The default setting is 0, which means that each network interface is used in a round-robin fashion (this supports aggregation and fault tolerance). Using setting 1 sets the bonding mode to active-backup. This means that only one network interface is used as a time, but that the link will automatically failover to a new interface if the primary interface fails. This settings only supports fault-tolerance.

The `miimon` option enables the MII link monitoring. A positive value greater than zero indicates the monitoring frequency in milliseconds for checking each slave network interface that is configured as part of the bonded interface. A typical value is 100.

You set th options within the module parameter file, and you must set the options for each bonded device individually:

```
options bond0 miimon=100 mode=1
```

- Reboot your server to enable the bonded devices.
- Configure the network device parameters. There are two parts to this, you need to setup the bonded device configuration, and then configure the original network interfaces as 'slaves' of the new bonded interface.
  - For RedHat Linux:

    Edit the configuration file for the bonded device. For device `bond0` this would be `/etc/sysconfig/network-scripts/ifcfg-bond0`:

    ```
    DEVICE=bond0
    BOOTPROTO=none
    ONBOOT=yes
    GATEWAY=192.168.0.254
    NETWORK=192.168.0.0
    NETMASK=255.255.255.0
    IPADDR=192.168.0.1
    USERCTL=no
    ```

    Then for each network interface that you want to be part of the bonded device, configure the interface as a slave to the 'master' bond. For example, the configuration of `eth0` in `/etc/sysconfig/network-scripts/ifcfg-eth0` might look like this::

    ```
    DEVICE=eth0
    BOOTPROTO=none
    HWADDR=00:11:22:33:44:55
    ONBOOT=yes
    TYPE=Ethernet
    MASTER=bond0
    SLAVE=yes
    ```

  - For Debian Linux:

    Edit the `/etc/iftab` file and configure the logical name and MAC address for each devices. For example:

    ```
    eth0 mac 00:11:22:33:44:55
    ```

    Now you need to set the configuration of the devices in `/etc/network/interfaces`:

    ```
    auto bond0
        iface bond0 inet static
        address 192.168.0.1
        netmask 255.255.255.0
        network 192.168.0.0
        gateway 192.168.0.254
        up /sbin/ifenslave bond0 eth0
        up /sbin/ifenslave bond0 eth1
    ```

  - For Gentoo:

    Use `emerge` to add the `net-misc/ifenslave` package to your system.

    Edit the `/etc.conf.d/net` file and specify the network interface slaves in a bond, the dependencies and then the configuration for the bond itself. A sample configuration might look like this:

    ```
    slaves_bond0="eth0 eth1 eth2"
    config_bond0=( "192.168.0.1 netmask 255.255.255.0"  )
    depend_bond0() {
    need net.eth0 net.eth1 net.eth2
    }
    ```

```
```

Then make sure that you add the new network interface to list of interfaces configured during boot:

```
root-shell> rc-update add default net.bond0
```

Once the bonded devices are configured you should reboot your systems.

You can monitor the status of a bonded connection using the `/proc` file system:

```
root-shell> cat /proc/net/bonding/bond0
Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eth1
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 200
Down Delay (ms): 200
Slave Interface: eth1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:11:22:33:44:55
Slave Interface: eth2
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:11:22:33:44:56
```

3.2. Optimizing the Synchronization Rate

The `syncer rate` configuration parameter should be configured with care as the synchronization rate can have a significant effect on the performance of the DRBD setup in the event of a node or disk failure where the information is being synchronized from the Primary to the Secondary node.

In DRBD, there are two distinct ways of data being transferred between peer nodes:

- *Replication* refers to the transfer of modified blocks being transferred from the primary to the secondary node. This happens automatically when the block is modified on the primary node, and the replication process uses whatever bandwidth is available over the replication link. The replication process cannot be throttled, because you want to transfer of the block information to happen as quickly as possible during normal operation.
- *Synchronization* refers to the process of bringing peers back in sync after some sort of outage, due to manual intervention, node failure, disk swap, or the initial setup. Synchronization is limited to the `syncer rate` configured for the DRBD device.

Both replication and synchronization can take place at the same time. For example, the block devices can be being synchronized while they are actively being used by the primary node. Any I/O that updates on the primary node will automatically trigger replication of the modified block. In the event of a failure within an HA environment, it is highly likely that synchronization and replication will take place at the same time.

Unfortunately, if the synchronization rate is set too high, then the synchronization process will use up all the available network bandwidth between the primary and secondary nodes. In turn, the bandwidth available for replication of changed blocks is zero, which means replication will stall and I/O will block, and ultimately the application will fail or degrade.

To avoid enabling the `syncer rate` to consume the available network bandwidth and prevent the replication of changed blocks you should set the `syncer rate` to less than the maximum network bandwidth.

You should avoid setting the sync rate to more than 30% of the maximum bandwidth available to your device and network bandwidth. For example, if your network bandwidth is based on Gigabit ethernet, you should achieve 110MB/s. Assuming your disk interface is capable of handling data at 110MB/s or more, then the sync rate should be configered as `33M` (33MB/s). If your disk system works at a rate lower than your network interface, use 30% of your disk interface speed.

Depending on the application, you may wish to limit the synchronization rate. For example, on a busy server you may wish to configure a significantly slower synchronization rate to ensure the replication rate is not affected.

The `al-extents` parameter controls the number of 4MB blocks of the underlying disk that can be written to at the same time. Increasing this parameter lowers the frequency of the meta data transactions required to log the changes to the DRBD device, which in turn lowers the number of interruptions in your I/O stream when synchronizing changes. This can lower the latency of changes to the DRBD device. However, if a crash occurs on your primary, then all of the blocks in the activity log (i.e. the number of `al-extents` blocks) will need to be completely resynchronized before replication can continue.

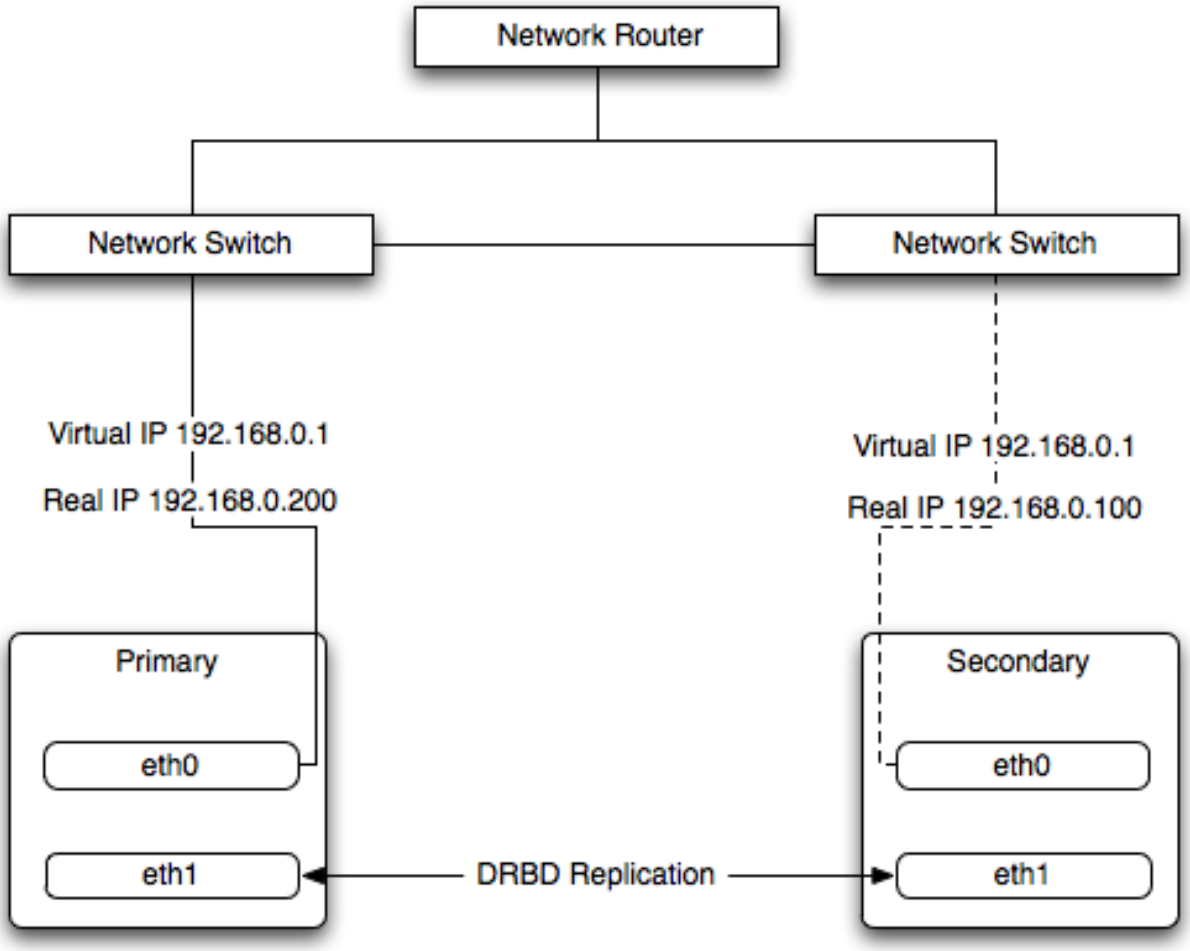
Chapter 4. Using Linux HA Heartbeat

The Heartbeat program provides a basis for verifying the availability of resources on one or more systems within a cluster. In this context a resource includes MySQL, the file systems on which the MySQL data is being stored and, if you are using DRBD, the DRBD device being used for the file system. Heartbeat also manages a virtual IP address, and the virtual IP address should be used for all communication to the MySQL instance.

A cluster within the context of Heartbeat is defined as two computers notionally providing the same service. By definition, each computer in the cluster is physically capable of providing the same services as all the others in the cluster. However, because the cluster is designed for high-availability, only one of the servers is actively providing the service at any one time. Each additional server within the cluster is a “hot-spare” that can be brought into service in the event of a failure of the master, its next connectivity or the connectivity of the network in general.

The basics of Heartbeat are very simple. Within the Heartbeat cluster (see Figure 4.1, “Heartbeat Architecture”, each machine sends a 'heartbeat' signal to the other hosts in the cluster. The other cluster nodes monitor this heartbeat. The heartbeat can be transmitted over many different systems, including shared network devices, dedicated network interfaces and serial connections. Failure to get a heartbeat from a node is treated as failure of the node. Although we do not know the reason for the failure (it could be an OS failure, a hardware failure in the server, or a failure in the network switch), it is safe to assume that if no heartbeat is produced there is a fault.

Figure 4.1. Heartbeat Architecture



In addition to checking the heartbeat from the server, the system can also check the connectivity (using `ping`) to another host on the network, such as the network router. This allows Heartbeat to detect a failure of communication between a server and the router (and therefore failure of the server, since it is no longer capable of providing the necessary service), even if the heartbeat between the servers in the clusters is working fine.

In the event of a failure, the resources on the failed host are disabled, and the resources on one of the replacement hosts is enabled instead. In addition, the Virtual IP address for the cluster is redirected to the new host in place of the failed device.

When used with MySQL and DRBD, the MySQL data is replicated from the master to the slave using the DRBD device, but MySQL is only running on the master. When the master fails, the slave switches the DRBD devices to be primary, the file systems on those devices are mounted, and MySQL is started. The original master (if still available) has its resources disabled, which means shutting down MySQL and unmounting the file systems and switching the DRBD device to secondary.

4.1. Heartbeat Configuration

Heartbeat configuration requires three files located in `/etc/ha.d`. The `ha.cf` contains the main heartbeat configuration, including the list of the nodes and times for identifying failures. `haresources` contains the list of resources to be managed within the cluster. The `authkeys` file contains the security information for the cluster.

The contents of these files should be identical on each host within the Heartbeat cluster. It is important that you keep these files in sync across all the hosts. Any changes in the information on one host should be copied to the all the others.

For these examples n example of the `ha.cf` file is shown below:

```
logfacility local0
keepalive 500ms
deadtime 10
warntime 5
initdead 30
mcast bond0 225.0.0.1 694 2 0
mcast bond1 225.0.0.2 694 1 0
auto_failback off
node drbd1
node drbd2
```

The individual lines in the file can be identified as follows:

- `logfacility` — sets the logging, in this case setting the logging to use `syslog`.
- `keepalive` — defines how frequently the heartbeat signal is sent to the other hosts.
- `deadtime`— the delay in seconds before other hosts in the cluster are considered 'dead' (failed).
- `warntime` — the delay in seconds before a warning is written to the log that a node cannot be contacted.
- `initdead` — the period in seconds to wait during system startup before the other host is considered to be down.
- `mcast` — defines a method for sending a heartbeat signal. In the above example, a multicast network address is being used over a bonded network device. If you have multiple clusters then the multicast address for each cluster should be unique on your network. Other choices for the heartbeat exchange exist, including a serial connection.

  If you are using multiple network interfaces (for example, one interface for your server connectivity and a secondary and/or bonded interface for your DRBD data exchange) then you should use both interfaces for your heartbeat connection. This decreases the chance of a transient failure causing a invalid failure event.
- `auto_failback` — sets whether the original (preferred) server should be enabled again if it becomes available. Switching this to `on` may cause problems if the preferred went offline and then comes back on line again. If the DRBD device has not been synced properly, or if the problem with the original server happens again you may end up with two different datasets on the two servers, or with a continually changing environment where the two servers flip-flop as the preferred server reboots and then starts again.
- `node` — sets the nodes within the Heartbeat cluster group. There should be one `node` for each server.

An optional additional set of information provides the configuration for a ping test that will check the connectivity to another host. You should use this to ensure that you have connectivity on the public interface for your servers, so the ping test should be to a reliable host such as a router or switch. The additional lines specify the destination machine for the `ping`, which should be specified as an IP address, rather than a host name; the command to run when a failure occurs, the authority for the failure and the timeout before an non-response triggers a failure. A sample configure is shown below:

```
ping 10.0.0.1
respawn hacluster /usr/lib64/heartbeat/ipfail
apiauth ipfail gid=haclient uid=hacluster
deadping 5
```

In the above example, the `ipfail` command, which is part of the Heartbeat solution, is called on a failure and 'fakes' a fault on the currently active server. You need to configure the user and group ID under which the command should be executed (using the

`apiauth`). The failure will be triggered after 5 seconds.

> **Note**
>
> The `deadping` value must be less than the `deadtime` value.

The `authkeys` file holds the authorization information for the Heartbeat cluster. The authorization relies on a single unique 'key' that is used to verify the two machines in the Heartbeat cluster. The file is used only to confirm that the two machines are in the same cluster and is used to ensure that the multiple clusters can co-exist within the same network.

4.2. Using Heartbeat with MySQL and DRBD

To use Heartbeat in combination with MySQL you should be using DRBD (see Using MySQL with DRBD) or another solution that allows for sharing of the MySQL database files in event of a system failure. In these examples, DRBD is used as the data sharing solution.

Heartbeat manages the configuration of different resources to manage the switching between two servers in the event of a failure. The resource configuration defines the individual services that should be brought up (or taken down) in the event of a failure.

The `haresources` file within `/etc/ha.d` defines the resources that should be managed, and the individual resource mentioned in this file in turn relates to scripts located within `/etc/ha.d/resource.d`. The resource definition is defined all on one line:

```
drbd1 drbddisk Filesystem::/dev/drbd0::/drbd::ext3 mysql 10.0.0.100
```

The line is notionally split by whitespace. The first entry (`drbd1`) is the name of the preferred host, i.e. the server that is normally responsible for handling the service. The last field is virtual IP address or name that should be used to share the service. This is the IP address that should be used to connect to the MySQL server. It will automatically be allocated to the server that is active when Heartbeat starts.

The remaining fields between these two fields define the resources that should be managed. Each Field should contain the name of the resource (and each name should refer to a script within `/etc/ha.d/resource.d`). In the event of a failure, these resources are started on the backup server by calling the corresponding script (with a single argument, `start`), in order from left to right. If there are additional arguments to the script, you can use a double colon to separate each additional argument.

In the above example, we manage the following resources:

- `drbddisk` — the DRBD resource script, this will switch the DRBD disk on the secondary host into primary mode, making the device read/write.
- `Filesystem` — manages the Filesystem resource. In this case we have supplied additional arguments to specify the DRBD device, mount point and file system type. When executed this should mount the specified file system.
- `mysql` — manages the MySQL instances and starts the MySQL server. You should copy the `mysql.resource` file from the `support-files` directory from any MySQL release into the `/etc/ha.d/resources.d` directory.

  If this file is not available in your distribution, you can use the following as the contents of the `/etc/ha.d/resource.d/mysql.resource` file:

```
#!/bin/bash
#
# This script is inteded to be used as resource script by heartbeat
#
# Mar 2006 by Monty Taylor
#
###
. /etc/ha.d/shellfuncs
case "$1" in
    start)
        res=`/etc/init.d/mysql start`
        ret=$?
        ha_log $res
        exit $ret
        ;;
    stop)
        res=`/etc/init.d/mysql stop`
        ret=$?
        ha_log $res
        exit $ret
        ;;
    status)
        if [ `ps -ef | grep '[m]ysqld'` ] ; then
           echo "running"
        else
           echo "stopped"
        fi
        ;;
    *)
        echo "Usage: mysql {start|stop|status}"
```

```
        exit 1
        ;;
esac
exit 0
```

If you want to be notified of the failure by email, you can add another line to the `haresources` file with the address for warnings and the warning text:

```
MailTo::youremail@address.com::DRBDFailure
```

With the Heartbeat configuration in place, copy the `haresources`, `authkeys` and `ha.cf` files from your primary and secondary servers to make sure that the configuration is identical. Then start the Heartbeat service, either by calling `/etc/init.d/heartbeat start` or by rebooting both primary and secondary servers.

You can test the configuration by running a manual failover, connect to the primary node and run:

```
root-shell> /usr/lib64/heartbeat/hb_standby
```

This will cause the current node to relinquish its resources cleanly to the other node.

4.3. Using Heartbeat with DRBD and dopd

As a further extension to using DRBD and Heartbeat together, you can enable `dopd`. The `dopd` daemon handles the situation where a DRBD node is out of date compared to the master and prevents the slave from being promoted to master in the event of a failure. This stops a situation where you have two machines that have been masters ending up different data on the underlying device.

For example, imagine that you have a two server DRBD setup, master and slave. If the DRBD connectivity between master and slave fails then the slave would be out of the sync with the master. If Heartbeat identifies a connectivity issue for master and then switches over to the slave, the slave DRBD device will be promoted to the primary device, even though the data on the slave and the master is not in synchronization.

In this situation, with `dopd` enabled, the connectivity failure between the master and slave would be identified and the metadata on the slave would be set to `Outdated`. Heartbeat will then refuse to switch over to the slave even if the master failed. In a dual-host solution this would effectively render the cluster out of action, as there is no additional fail over server. In an HA cluster with three or more servers, control would be passed to the slave that has an up to date version of the DRBD device data.

To enable `dopd`, you need to modify the Heartbeat configuration and specify `dopd` as part of the commands executed during the monitoring process. Add the following lines to your `ha.cf` file:

```
respawn hacluster /usr/lib/heartbeat/dopd
apiauth dopd gid=haclient uid=hacluster
```

Make sure you make the same modification on both your primary and secondary nodes.

You will need to reload the Heartbeat configuration:

```
root-shell> /etc/init.d/heartbeat reload
```

You will also need to modify your DRBD configuration by configuration the `outdate-peer` option. You will need to add the configuration line into the `common` section of `/etc/drbd.conf` on both hosts. An example of the full block is shown below:

```
common {
  handlers {
    outdate-peer "/usr/lib/heartbeat/drbd-peer-outdater";
  }
}
```

Finally, set the `fencing` option on your DRBD configured resources:

```
resource my-resource {
  disk {
    fencing    resource-only;
  }
}
```

Now reload your DRBD configuration:

```
root-shell> drbdadmin adjust all
```

You can test the system by unplugging your DRBD link and monitoring the output from `/proc/drbd`.

4.4. Dealing with System Level Errors

Because a kernel panic or oops may indicate potential problem with your server, you should configure your server to remove itself from the cluster in the event of a problem. Typically on a kernel panic your system will automatically trigger a hard reboot. For a kernel oops a reboot may not happen automatically, but the issue that caused that oops may still lead to potential problems.

You can force a reboot by setting the `kernel.panic` and `kernel.panic_on_oops` parameters of the kernel control file `/etc/sysctl.conf`. For example:

```
kernel.panic_on_oops = 1
kernel.panic = 1
```

You can also set these parameters during runtime by using the `sysctl` command. You can either specify the parameters on the command line:

```
shell> sysctl -w kernel.panic=1
```

Or you can edit your `sysctl.conf` file and then reload the configuration information:

```
shell> sysctl -p
```

By setting both these parameters to a positive value (actually the number of seconds to wait before triggering the reboot), the system will reboot. Your second heartbeat node should then detect that the server is down and then switch over to the failover host.

Chapter 5. MySQL 5.1 FAQ — MySQL, DRBD, and Heartbeat

5.1. Distributed Replicated Block Device (DRBD)

In the following section, we provide answers to questions that are most frequently asked about Distributed Replicated Block Device (DRBD).

Questions

- 5.1.1: What is DRBD?
- 5.1.2: What are “Block Devices”?
- 5.1.3: How is DRBD licensed?
- 5.1.4: Where can I download DRBD?
- 5.1.5: If I find a bug in DRBD, to whom do I submit the issue?
- 5.1.6: Where can I get more technical and business information concerning MySQL and DRBD?

Questions and Answers

5.1.1: What is DRBD?

DRBD is an acronym for Distributed Replicated Block Device. DRBD is an open source Linux kernel block device which leverages synchronous replication to achieve a consistent view of data between two systems, typically an Active and Passive system. DRBD currently supports all the major flavors of Linux and comes bundled in several major Linux distributions. The DRBD project is maintained by LINBIT.

5.1.2: What are “Block Devices”?

A *block device* is the type of device used to represent storage in the Linux Kernel. All physical disk devices present a block device interface. Additionally, virtual disk systems like LVM or DRBD present a block device interface. In this way, the file system or other software that might want to access a disk device can be used with any number of real or virtual devices without having to know anything about their underlying implementation details.

5.1.3: How is DRBD licensed?

DRBD is licensed under the GPL.

5.1.4: Where can I download DRBD?

Please see http://www.drbd.org/download/packages/.

5.1.5: If I find a bug in DRBD, to whom do I submit the issue?

Bug reports should be submitted to the DRBD mailing list. Please see http://lists.linbit.com/.

5.1.6: Where can I get more technical and business information concerning MySQL and DRBD?

Please visit http://mysql.com/drbd/.

5.2. Linux Heartbeat

In the following section, we provide answers to questions that are most frequently asked about Linux Heartbeat.

Questions

- 5.2.1: What is Linux Heartbeat?
- 5.2.2: How is Linux Heartbeat licensed?
- 5.2.3: Where can I download Linux Heartbeat?
- 5.2.4: If I find a bug with Linux Heartbeat, to whom do I submit the issue?

Questions and Answers

5.2.1: What is Linux Heartbeat?

The Linux-HA project (http://www.linux-ha.org/) offers a high availability solution commonly referred to as Linux Heartbeat. Linux Heartbeat ships as part of several Linux distributions, as well as within several embedded high availability systems. This solution can also be used for other applications besides databases servers, such as mail servers, web servers, file servers, and DNS servers.

Linux Heartbeat implements a heartbeat-protocol. A heartbeat-protocol means that messages are sent at regular intervals between two or more nodes. If a message is not received from a node within a given interval, then it is assumed the node has failed and some type of failover or recovery action is required. Linux Heartbeat is typically configured to send these heartbeat messages over standard Ethernet interfaces, but it does also support other methods, such as serial-line links.

5.2.2: How is Linux Heartbeat licensed?

Linux Heartbeat is licensed under the GPL.

5.2.3: Where can I download Linux Heartbeat?

Please see http://linux-ha.org/download/index.html.

5.2.4: If I find a bug with Linux Heartbeat, to whom do I submit the issue?

Bug reports should be submitted to http://www.linux-ha.org/ClusterResourceManager/BugReports.

5.3. DRBD Architecture

In the following section, we provide answers to questions that are most frequently asked about DRBD Architecture.

Questions

- 5.3.1: Is an Active/Active option available for MySQL with DRBD?
- 5.3.2: What MySQL storage engines are supported with DRBD?
- 5.3.3: How long does a failover take?
- 5.3.4: How long does it take to resynchronize data after a failure?
- 5.3.5: Are there any situations where you shouldn't use DRBD?
- 5.3.6: Are there any limitations to DRBD?
- 5.3.7: Where can I find more information on sample architectures?

Questions and Answers

5.3.1: Is an Active/Active option available for MySQL with DRBD?

Currently, MySQL does not support Active/Active configurations using DRBD “out of the box”.

5.3.2: What MySQL storage engines are supported with DRBD?

All of the MySQL transactional storage engines are supported by DRBD, including InnoDB and Falcon. For archived or read-only data, MyISAM or Archive can also be used.

5.3.3: How long does a failover take?

Failover time is dependent on many things, some of which are configurable. After activating the passive host, MySQL will have to start and run a normal recovery process. If the InnoDB log files have been configured to a large size and there was heavy write traffic, this may take a reasonably long period of time. However, under normal circumstances, failover tends to take less than a minute.

5.3.4: How long does it take to resynchronize data after a failure?

Resynchronization time depends on how long the two machines are out of communication and how much data was written during that period of time. Resynchronization time is a function of data to be synced, network speed and disk speed. DRBD maintains a bitmap of changed blocks on the primary machine, so only those blocks that have changed will need to be transferred.

5.3.5: Are there any situations where you shouldn't use DRBD?

See When Not To Use DRBD.

5.3.6: Are there any limitations to DRBD?

See DRBD limitations (or are they?).

5.3.7: Where can I find more information on sample architectures?

For an example of a Heartbeat R1-compatible resource configuration involving a MySQL database backed by DRBD, see DRBD User's Guide.

For an example of the same DRBD-backed configuration for a MySQL database in a Heartbeat CRM cluster, see DRBD User's Guide.

5.4. DRBD and MySQL Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication Scale-out.

Questions

- 5.4.1: What is the difference between MySQL Cluster and DRBD?
- 5.4.2: What is the difference between MySQL Replication and DRBD?
- 5.4.3: How can I combine MySQL Replication scale-out with DRBD?

Questions and Answers

5.4.1: What is the difference between MySQL Cluster and DRBD?

Both MySQL Cluster and DRBD replicate data synchronously. MySQL Cluster leverages a shared-nothing storage architecture in which the cluster can be architected beyond an Active/Passive configuration. DRBD operates at a much lower level within the "stack", at the disk I/O level. For a comparison of various high availability features between these two options, please refer to High Availability and Scalability.

5.4.2: What is the difference between MySQL Replication and DRBD?

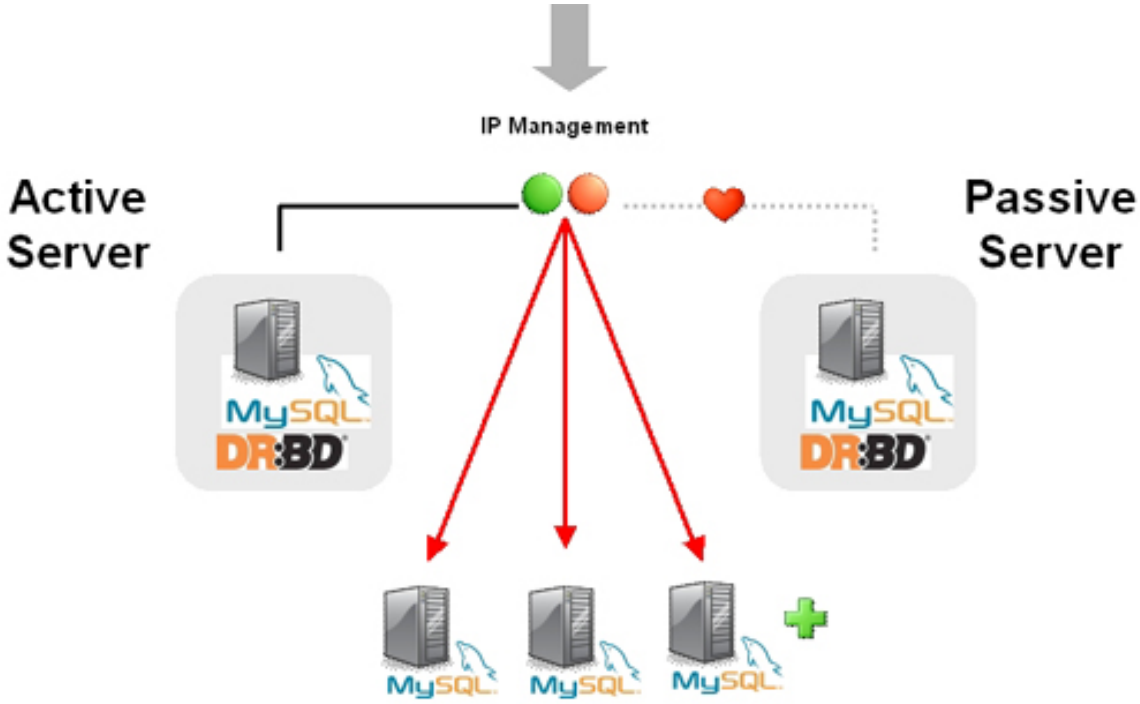
MySQL Replication replicates data asynchronously while DRBD replicates data synchronously. Also, MySQL Replication replicates MySQL statements, while DRBD replicates the underlying block device that stores the MySQL data files. For a comparison of various high availability features between these two options, please refer to the high availability comparison grid, High Availability and Scalability.

5.4.3: How can I combine MySQL Replication scale-out with DRBD?

MySQL Replication is typically deployed in a Master to many Slaves configuration. In this configuration, having many Slaves provides read scalability. DRBD is used to provide high-availability for the Master MySQL Server in an Active/Passive configuration. This provides for automatic failover, safeguards against data loss, and automatically synchronizes the failed MySQL Master after a failover.

The most likely scenario in which MySQL Replication scale-out can be leveraged with DRBD is in the form of attaching replicated MySQL "read-slaves" off of the Active-Master MySQL Server, shown in Figure 5.1, "Active-Master MySQL server". Since DRBD replicates an entire block device, master information such as the binary logs are also replicated. In this way, all of the slaves can attach to the Virtual IP Address managed by Linux Heartbeat. In the event of a failure, the asynchronous nature of MySQL Replication allows the slaves to continue with the new Active machine as their master with no intervention needed.

Figure 5.1. Active-Master MySQL server



5.5. DRBD and File Systems

In the following section, we provide answers to questions that are most frequently asked about DRBD and file systems.

Questions

- 5.5.1: Can XFS be used with DRBD?

Questions and Answers

5.5.1: Can XFS be used with DRBD?

Yes. XFS uses dynamic block size, thus DRBD 0.7 or later is needed.

5.6. DRBD and LVM

In the following section, we provide answers to questions that are most frequently asked about DRBD and LVM.

Questions

- 5.6.1: Can I use DRBD on top of LVM?
- 5.6.2: Can I use LVM on top of DRBD?
- 5.6.3: Can I use DRBD on top of LVM while at the same time running LVM on top of that DRBD?

Questions and Answers

5.6.1: Can I use DRBD on top of LVM?

Yes, DRBD supports on-line resizing. If you enlarge your logical volume that acts as a backing device for DRBD, you can enlarge DRBD itself too, and of course your file system if it supports resizing.

5.6.2: Can I use LVM on top of DRBD?

Yes, you can use DRBD as a Physical Volume (PV) for LVM. Depending on the default LVM configuration shipped with your dis-

tribution, you may need to add the `/dev/drbd*` device files to the `filter` option in your `lvm.conf` so LVM scans your DRBDs for PV signatures.

5.6.3: Can I use DRBD on top of LVM while at the same time running LVM on top of that DRBD?

This requires careful tuning of your LVM configuration to avoid duplicate PV scans, but yes, it is possible.

5.7. DRBD and Virtualization

In the following section, we provide answers to questions that are most frequently asked about DRBD and virtualization.

Questions

- 5.7.1: Can I use DRBD with OpenVZ?
- 5.7.2: Can I use DRBD with Xen and/or KVM?

Questions and Answers

5.7.1: Can I use DRBD with OpenVZ?

See http://wiki.openvz.org/HA_cluster_with_DRBD_and_Heartbeat.

5.7.2: Can I use DRBD with Xen and/or KVM?

Yes. If you are looking for professional consultancy or expert commercial support for Xen- or KVM-based virtualization clusters with DRBD, contact LINBIT (http://www.linbit.com).

5.8. DRBD and Security

In the following section, we provide answers to questions that are most frequently asked about DRBD and security.

Questions

- 5.8.1: Can I encrypt/compress the exchanged data?
- 5.8.2: Does DRBD do mutual node authentication?

Questions and Answers

5.8.1: Can I encrypt/compress the exchanged data?

Yes. But there is no option within DRBD to allow for this. Youâ##ll need to leverage a VPN and the network layer should do the rest.

5.8.2: Does DRBD do mutual node authentication?

Yes, starting with DRBD 8 shared-secret mutual node authentication is supported.

5.9. DRBD and System Requirements

In the following section, we provide answers to questions that are most frequently asked about DRBD and System Requirements.

Questions

- 5.9.1: What other packages besides DRBD are required?
- 5.9.2: How many machines are required to set up DRBD?
- 5.9.3: Does DRBD only run on Linux?

Questions and Answers

5.9.1: What other packages besides DRBD are required?

When using pre-built binary packages, none except a matching kernel, plus packages for `glibc` and your favorite shell. When compiling DRBD from source additional prerequisite packages may be required. They include but are not limited to:

- glib-devel
- openssl
- devel
- libgcrypt-devel
- glib2-devel
- pkgconfig
- ncurses-devel
- rpm-build
- rpm-devel
- redhat-rpm-config
- gcc
- gcc-c++
- bison
- flex
- gnutls-devel
- lm_sensors-devel
- net-snmp-devel
- python-devel
- bzip2-devel
- libselinux-devel
- perl-DBI
- libnet

Pre-built x86 and x86_64 packages for specific kernel versions are available with a support subscription from LINBIT. Please note that if the kernel is upgraded, DRBD must be as well.

5.9.2: How many machines are required to set up DRBD?

Two machines are required to achieve the minimum degree of high availability. Although at any one given point in time one will be primary and one will be secondary, it is better to consider the machines as part of a mirrored pair without a “natural” primary machine.

5.9.3: Does DRBD only run on Linux?

DRBD is a Linux Kernel Module, and can work with many popular Linux distributions. DRBD is currently not available for non-Linux operating systems.

5.10. DBRD and Support and Consulting

In the following section, we provide answers to questions that are most frequently asked about DRBD and resources.

Questions

- 5.10.1: Does MySQL offer professional consulting to help with designing a DRBD system?

- 5.10.2: Does MySQL offer support for DRBD and Linux Heartbeat from MySQL?
- 5.10.3: Are pre-built binaries or RPMs available?
- 5.10.4: Does MySQL have documentation to help me with the installation and configuration of DRBD and Linux Heartbeat?
- 5.10.5: Is there a dedicated discussion forum for MySQL High-Availability?
- 5.10.6: Where can I get more information about MySQL for DRBD?

Questions and Answers

5.10.1: Does MySQL offer professional consulting to help with designing a DRBD system?

Yes. MySQL offers consulting for the design, installation, configuration, and monitoring of high availability DRBD. For more information concerning a High Availability Jumpstart, please see: http://www.mysql.com/consulting/packaged/scaleout.html.

5.10.2: Does MySQL offer support for DRBD and Linux Heartbeat from MySQL?

Yes. Support for DRBD is available with an add-on subscription to MySQL Enterprise called â##DRBD for MySQLâ##. For more information about support options for DRBD see: http://mysql.com/products/enterprise/features.html.

For the list of supported Linux distributions, please see: http://www.mysql.com/support/supportedplatforms/enterprise.html.

> **Note**
>
> DRBD is only available on Linux. DRBD is not available on Windows, MacOS, Solaris, HPUX, AIX, FreeBSD, or other non-Linux platforms.

5.10.3: Are pre-built binaries or RPMs available?

Yes. “DRBD for MySQL” is an add-on subscription to MySQL Enterprise, which provides pre-built binaries for DRBD. For more information, see: http://mysql.com/products/enterprise/features.html.

5.10.4: Does MySQL have documentation to help me with the installation and configuration of DRBD and Linux Heartbeat?

For MySQL-specific DRBD documentation, see Using MySQL with DRBD.

For general DRBD documentation, see DRBD User's Guide.

5.10.5: Is there a dedicated discussion forum for MySQL High-Availability?

Yes, http://forums.mysql.com/list.php?144.

5.10.6: Where can I get more information about MySQL for DRBD?

For more information about MySQL for DRBD, including a technical white paper please see: DRBD for MySQL High Availability.