MySQL and DRBD High Availability Architectures

An introduction to high availability architectures with MySQL, DRBD and Linux Heartbeat

A MySQL® Technical White Paper

May, 2007

Copyright © 2007, MySQL AB

Table of Contents

Executive Summary

Maintaining the availability of systems and access to data has become more important then ever for businesses big and small. Although high availability has traditionally been the domain of enterprise and mission critical applications, less critical systems in traditional companies are beginning to take advantage of the many open-source, low-cost, high availability solutions currently available. For many modern businesses, their entire revenue models are based on an online web presence which must be available 24 hours a day. For many of these modern companies, open source components have long been key enablers in building scalable, low-cost and highly availability systems. Arguably, the success of these open source solutions has begun to interest many administrators who oversee enterprise and mission critical applications. Because of the widespread dependence on expensive or proprietary software and hardware, administrators are beginning to look at how proven low-cost open source solutions like MySQL and DRBD (Distributed Replicated Block Device) can help lower their costs without sacrificing availability, performance or scalability.

MySQL offers a wide array of options when designing high availability systems. These include MySQL Replication, MySQL Cluster, and MySQL in conjunction with open-source high availability packages or commercial products from the network of MySQL certified partners. In this paper we explore how to design clustered, high availability MySQL systems based on open source software running on commodity-off-the-shelf (COTS) hardware. Because of reduced complexity, these systems are typically easier to design, configure, maintain and upgrade then expensive proprietary systems. All this yields a lower total cost of ownership for your database management system.

MySQL High Availability Solutions Overview

In a nutshell, DRBD is a block device which leverages synchronous replication transparent to the application, database or file system. Which, when used in conjunction with the open-source Linux Heartbeat package, enables solution architects to design a MySQL database which can leverage automated resource fail over after a failure, in a fully transaction-safe “hot standby” configuration.

Before exploring MySQL and DRBD high availability architectures, it is wise to first become familar with other MySQL high availability solutions to see perspective on how they compare to a MySQL, Heartbeat and DRBD cluster.

MySQL Replication

MySQL natively supports one-way, asynchronous replication. MySQL Replication works by simply having one server act as a master, while one or more servers act as slaves. This is in contrast to the synchronous replication which is a characteristic of MySQL Cluster and DRBD.

Asynchronous data replication means data is copied from one machine to another, with a resultant delay in the actual copying. Often this delay is determined by networking bandwidth, resource availability and system load. However, with the correct components and tuning, replication itself can appear to be almost instantaneous to most applications. Synchronous data replication implies that data is committed to one or more machines at the same time, usually via what is commonly known as a “two-phase commit”. Asynchronous vs. synchronous replication is illustrated below in Figure 1.

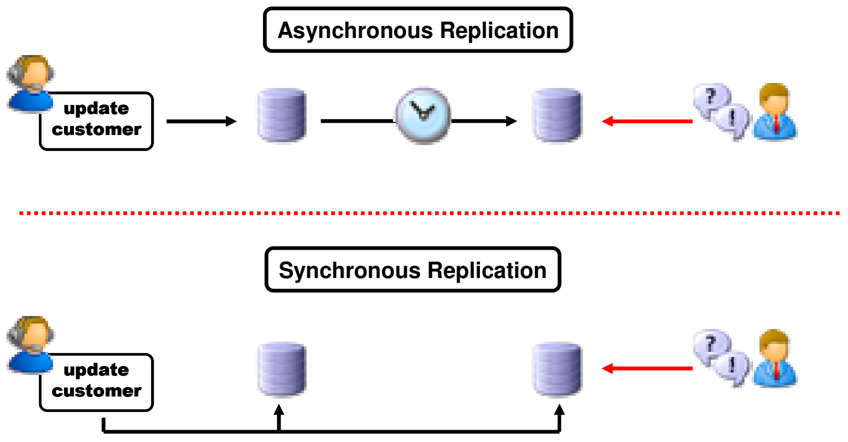


Figure 1

In standard MySQL Replication, the master server writes updates to its binary log files and maintains an index of those files in order to keep track of the log rotation. The binary log files serve as a record of updates to be sent to slave servers. When a slave connects to its master, it determines the last position it has read in the logs on its last successful update. The slave then receives any updates which have taken place since that time. The slave subsequently blocks and waits for the master to notify it of new updates.

Replication offers the benefits of reliability, performance, and ease of use:

- In the event the master fails, the application can be designed to switch to the slave. This is typically achieved using some type of virtual IP management software or hardware, such as Linux Heartbeat or a load balancer.
- Better response time for clients can be achieved by splitting the load for processing client queries between the master and slave servers. Queries which simply “read” data, such as SELECTs, may be sent to the slave in order to reduce the query processing load on the master. Statements that modify data should be sent to the master so that the data on the master and slave do not get out of synch. This load-balancing strategy is effective if non-updating queries dominate. An illustration of this type of architecture is depicted below in Figure 2.
- Another benefit of using replication is that database backups can be performed using a slave server without impacting the resources on the master. The master continues to process updates while the backup is being made.

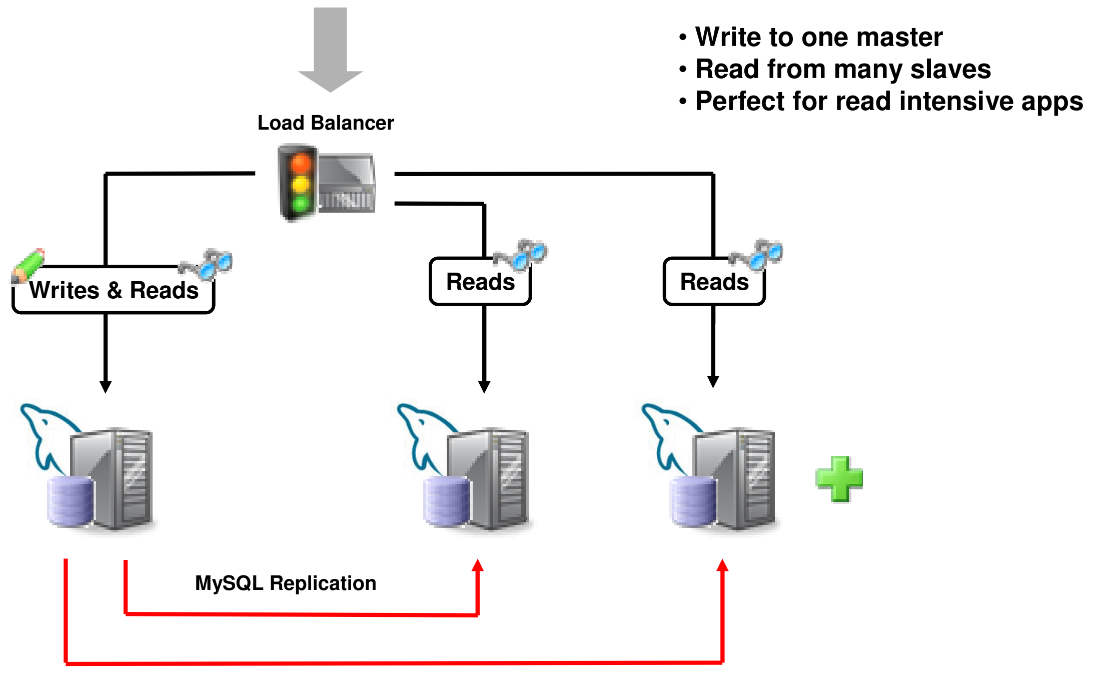


Figure 2

Although MySQL Replication can solve many high availability design issues in a simple and straight forward manner, three problems inherent to the configuration do need to be addressed by developers and/or administrators.

First, in order to take advantage of replication for read scalability, via scale-out, the application interacting with the databases must become “replication aware”. This means that the application must be coded to write only to the master and always read from the slave.

The second issue revolves around creating a strategy for initiating a fail over in the event of a hardware or software problem on the master.

Lastly, there should also be a process (preferably automated) for returning the system to its original configuration before the failure. Often this is referred to as resynchronizing data or systems after a failure.

MySQL Cluster

MySQL Cluster was designed to meet the throughput and response time requirements needed by some of the most demanding enterprise applications in the world. In brief, MySQL Cluster can be described as a shared-nothing, distributed, synchronous database cluster which supports automatic fail over of data nodes, transactions and in-memory data storage without any special networking, hardware or storage requirements. Designing the system in this way allows MySQL Cluster to deliver both high availability and reliability, since single points of failure have been eliminated. Any node can fail without affecting the system as a whole. An application, for example, can continue executing transactions even though a data node has failed. MySQL Cluster has also proven to handle tens of thousands of distributed transactions per second, with changes synchronously replicated across data nodes. A high level architectural illustration can be found below in Figure 3.

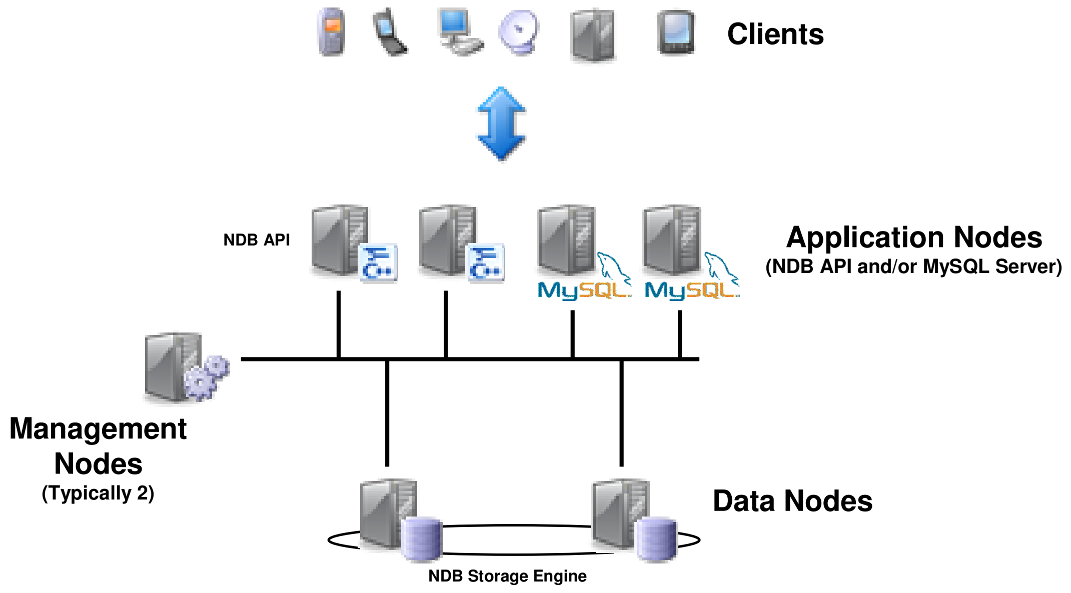


Figure 3

MySQL Cluster delivers an extremely fast fail over time with sub-second responses so your applications can recover quickly in the event of a software, network or hardware failure. MySQL Cluster uses synchronous replication to propagate transaction information to all the appropriate data nodes. This also eliminates the time consuming operation of recreating and replaying log files as is typically required by clusters employing shared-disk architectures. MySQL Cluster data nodes are also able to automatically restart, recover, and dynamically reconfigure themselves in the event of failures, without developers having to program any fail over logic into their applications.

MySQL Cluster implements automatic data node recovery that ensures any fail over to another data node will contain a consistent set of data. Should all the data nodes fail due to hardware faults, MySQL Cluster ensures an entire system can be safely recovered in a consistent state by using a combination of checkpoints and log execution. Furthermore, MySQL Cluster ensures systems are available and consistent across geographies by enabling entire clusters to be replicated across regions using MySQL Replication

The information above showcases how MySQL Cluster addresses several problems MySQL Replication leaves unresolved when used in a high availability architecture. MySQL Cluster provides a mechanism for ensuring there is consistent view of data in the database at all times by employing synchronous replication. It also provides a mechanism for resynchronizing data nodes which have failed so their data is consistent and up to date since the failure.

It's worth pointing out that applications do not need to become “replication-aware”, in order to gain read scalability. Because of the distributed nature of MySQL Cluster, all data nodes are able to participate in read operations without the need for modifications to the application.

One issue which must be addressed outside of MySQL Cluster is what happens in the event the MySQL Server an application is using to access the cluster fails. The application needs some mechanism for failing over to another MySQL Server in order to gain access to the data residing in the MySQL Cluster. This is typically achieved using some type of virtual IP management software or hardware, such as Linux Heartbeat or a load balancer to reconnect the application to a viable MySQL Server.

MySQL Replication with Linux Heartbeat

The Linux-HA project 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 fail over 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.

When Linux Heartbeat is initially configured, a master node is selected. When the heartbeat starts up on the master node, it will assign a virtual IP address to the master's network interface. This interface will be the manner in which external processes, applications and users will access the node. If the master node fails, then another node within the cluster will start up an interface for this virtual IP address and use "gratuitous ARP" to ensure that all traffic bound for this address is received by this machine. (Gratuitous ARP (Address Resolution Protocol) is when a host sends an ARP request to resolve its own IP address.) This method of fail-over is often referred to as "IP Address Takeover".

Each virtual IP address is considered to be a resource. Resources are encapsulated as programs that work similarly to Unix "init" scripts. This means that the resource can be started and stopped, and it can be queried to see if it is running. In this manner, Linux Heartbeat is able to start and stop these resources (virtual IPs) depending on the status of the other node that it is communicating with using this heartbeat-protocol.

The integration of MySQL into a Linux Heartbeat system will typically exhibit the following qualities:

- Easy to configure with low architectural complexity
- Open source
- Low cost
- No special hardware or networking components required
- Virtual IP management is automatically handled
- Data replication will still remain asynchronous, so there could be potential issues with confirming that all applicable data has been transferred to the slave server in the event of a fail over
- The above issue may be further complicated by additional failures in the middle of a fail over
- Repairing a fail over can also become complex

Figure 4 illustrates a simulated fail over in which the master MySQL Server has suffered a failure. In turn, Linux Heartbeat automatically redirects the application to the slave server. Notice however that the database administrator will have to employ some mechanism (manual or automated) to ensure that all the appropriate data has been replicated from the master to the slave in the event of a fail over.

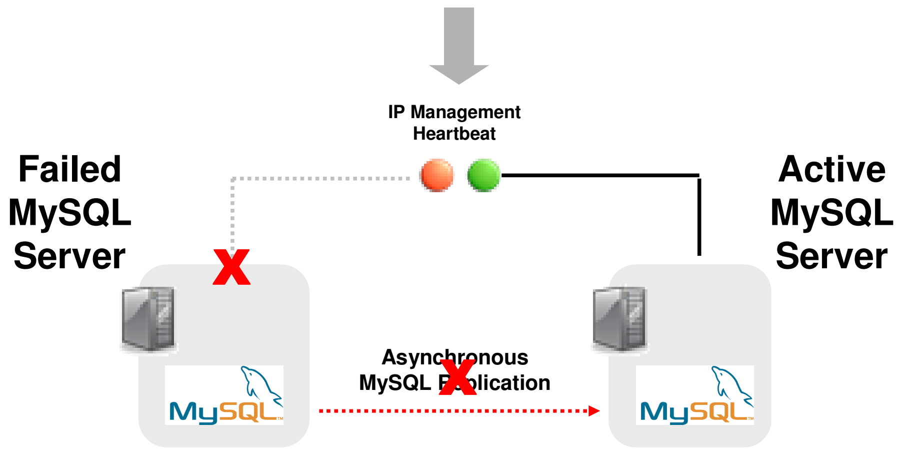


Figure 4

Introduction to DRBD

As previously mentioned, DRBD is an acronym for Distributed Replicated Block Device. It is a Linux kernel module which when combined with some additional components like Linux Heartbeat, offers a distributed, synchronized storage system. DRBD bears superficial similarities to RAID-1, albeit run over a network.

Block Devices

Block devices constitute the lowest-level disk input/output (I/O) abstraction layer present in the Linux kernel and other UNIX-style operating systems. They are the operating system's interface to any random-access persistent data storage. The operating system writes to and reads from block devices using chunks of data referred to as blocks.

Block devices in Linux may correspond to “real” hardware devices, such as disks or disk partitions. This is the most common case. But block devices may also be stacked on top of one another, or be completely virtual and detached from actual disk hardware. Regardless, the kernel and applications address all block devices in an equivalent manner, namely, by reading and writing blocks.

Other I/O abstraction facilities are frequently layered on top of block devices. The most common of these are file systems, an interface which masks block access and allows interaction with files and directories, but others are available as well. What should be noted, however, is that all these additional layers are always able to make full use of any features the underlying block device may have, and that they do so in a fully transparent fashion.

Block devices use buffered input and output routines. The operating system allocates a data buffer to hold a single block each for input and output. When a program sends a request for data to be read from or written to the device, each character of that data is stored in the appropriate buffer. When the buffer is full and a complete block is achieved, the appropriate operation is performed and the buffer is cleared.

What is DRBD?

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 distributions with user space management tools. The DRBD project itself is maintained by LINBIT with support available from MySQL as an add-on to a MySQL Enterprise subscription.

DRBD Fundamentals

Each device configured in a DRBD cluster has a state, which can be either primary or secondary. DRBD creates, on both nodes, a link between the virtual device and a local partition. All writes are done on the primary node, which in turn transfers data to the lower-level block device and then propagates it to the secondary node. The secondary node transfers data to the lower-level block device. All reads are performed locally which is a key advantage over SAN or NAS based storage.

Should the primary node fail, a cluster management process will promote the secondary node to a primary state. When the failed node attempts to rejoin the cluster, the system may (or may not) promote it back to primary after data synchronization. This is a configuration option which can be set by the administrator.

Linux Heartbeat is typically used to handle primary/secondary state transitions. Together with these state transitions, Heartbeat will also mount the file system it uses on top of the virtual device created by DRBD.

It is worth pointing out that the DRBD cluster is fully available during the resynchronization process. Only the parts of the device which have changed since the failure are resynchronized. This allows for a total resynchronization in just several minutes, regardless of device size even after the hard crash of an active node. Resynchronization rates may be configured per device with device sizes currently limited to 4 TB.

Another important aspect of DRBD the fact that it is much more resilient against “split-brain” situations than conventional cluster architectures involving shared storage. A split-brain situation occurs when all network connectivity between cluster nodes is lost, but the nodes are still both alive and responsive. In such a situation, each node assumes that it is the only surviving node in the cluster, and therefore attempts to activate all cluster resources locally.

In shared-storage environments, the results are potentially disastrous as both nodes will concurrently attempt to mount and write to the same file systems, causing data corruption. Administrators must therefore carefully implement “fencing” strategies to work around this scenario. In a DRBD-based system, in contrast, the split-brain situation creates diverging sets of data, with policies available for manual or fully-automatic resolution. Fencing is typically not required (although it is supported).

Finally, because DRBD operates at the disk I/O layer, it allows for a high availability architecture which is MySQL storage engine independent. This means that both the InnoDB and Falcon transactional storage engines can be leveraged with DRBD.

Sample Architectures

The following sections highlight will look at several ways to design high availability architectures with MySQL and DRBD, but also achieve scale-out by combining them with MySQL Replication. Figure 5 we illustrates some of the differences between the architectures already mentioned.

Requirements	***MySQL Replication***	***MySQL Replication + Heartbeat***	***MySQL, Heartbeat + DRBD***	***MySQL Cluster***
Availability				
Automated IP Fail Over	No	Yes	Yes	No
Automated DB Fail Over	No	No	Yes	Yes
Typical Fail Over Time	Varies	Varies	< 30 secs	< 3 secs
Auto Resynch of Data	No	No	Yes	Yes
Geographic Redundancy	Yes	Yes	MySQL Replication	MySQL Replication
Scalability				
Built-in Load Balancing	MySQL Replication	MySQL Replication	MySQL Replication	Yes
Read Intensive	Yes	Yes	MySQL Replication	Yes
Write Intensive	No	No	Possible	Yes
# of Nodes per Cluster	Master/Slave(s)	Master/Slave(s)	Active/Passive	255
# of Slaves	Dozens for Reads	Dozens for Reads	Dozens for Reads	Dozens for Reads

Figure 5

Active/Passive DRBD Cluster

Figure 6 illustrates a basic configuration with two systems comprised of an active/passive DRBD cluster with Heartbeat providing failover services.

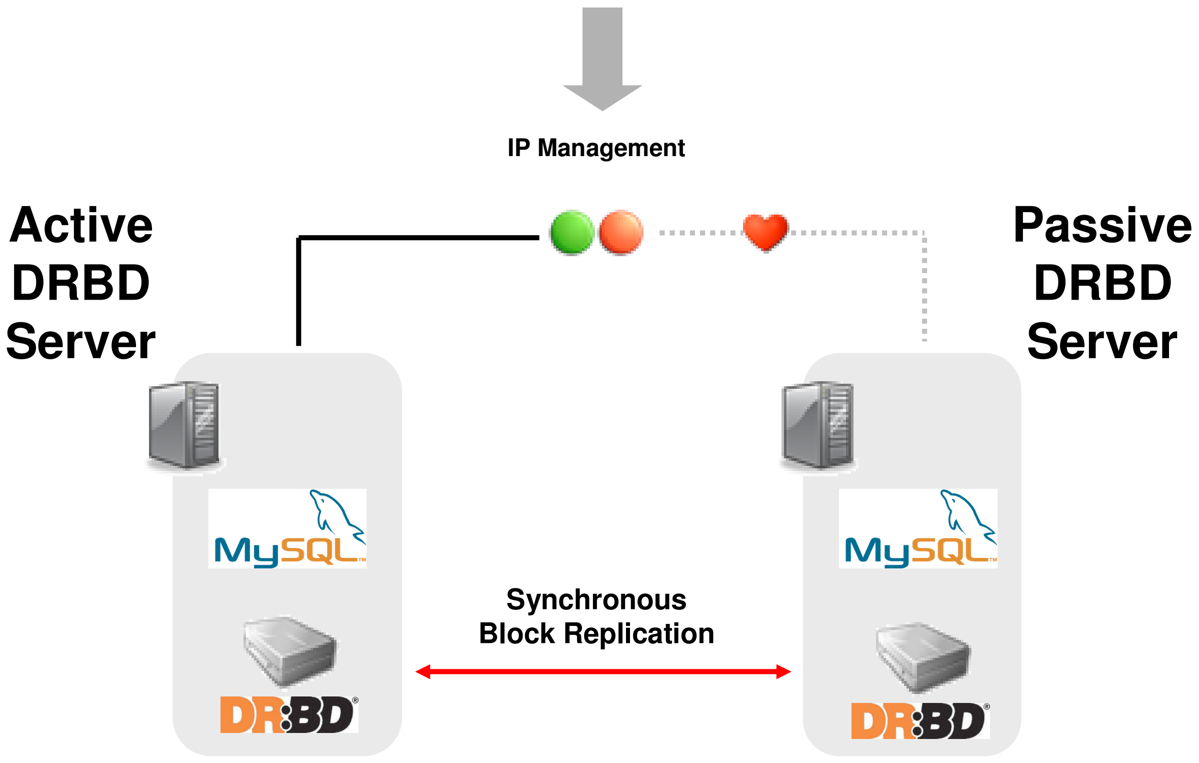
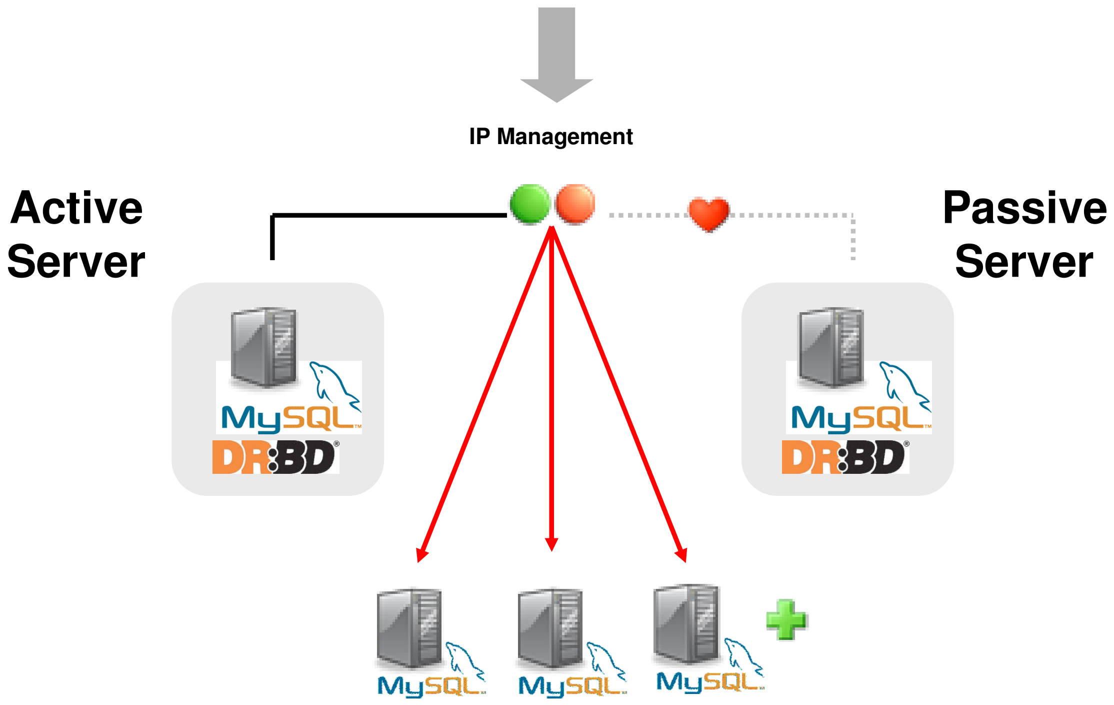


Figure 6

Scale Out: Active/Passive DRBD Cluster with MySQL Replication

Figure 7 extends the read scalability of the previous DRBD cluster by leveraging MySQL Replication. MySQL slaves replicate from which ever DRBD system is in a primary or active state. This allows the directing of direct reads to MySQL slaves and capitalizes on the benefits of scale out.



MySQL Replication Slaves – Read Scale Out – Asynchronous Replication

Figure 7

Application Partitioning: DRBD Cluster with MySQL Replication

In order to gain even more scalability, application-level partitioning can be leveraged. This technique we separates data into distinct, autonomous silos or "shards" within individual DRBD clusters. In turn, applications access "shards" directly or through load balancing. When used in conjunction with MySQL Replication, an increase in read & write scalability is realized. The potential to increase the availability characteristics of the system is also possible depending on partitioning scheme. Ultimately, through the use of application partitioning, read vs. write query partitioning and possibly even MySQL's data partitioning, a system can be designed that can make the most efficient and effective use of network and hardware resources. A DRBD cluster which makes use of application partitioning is illustrated below in Figure 8.

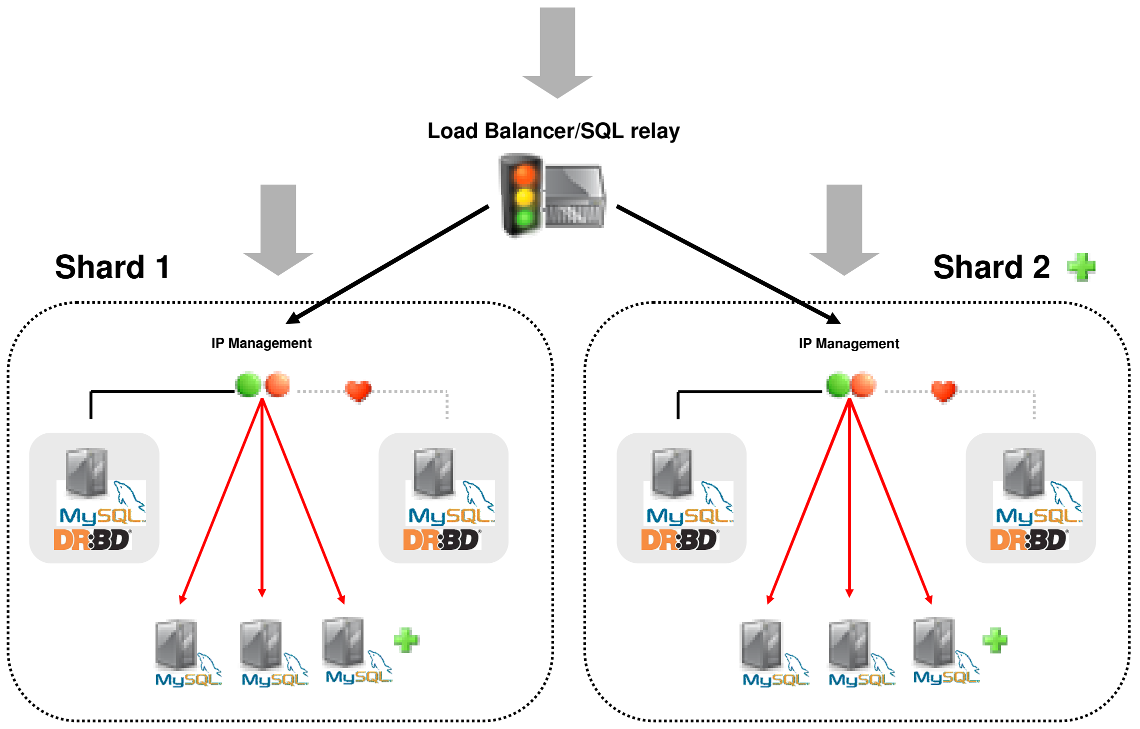


Figure 8

Setup and Configuration Overview

Although the steps listed below are not exhaustive, they are intended to give the user a general idea of the complexity and requirements required to build a MySQL and DRBD cluster. For complete information including a detailed "How-To" please visit http://www.mysql.com/products/enterprise/drbd.html

Prerequisites

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. As of this writing, the current DRBD version is 8, available at:

http://www.drbd.org/download.html

Some of the key features introduced in version 8 include:

- Better, fine tune recovery mechanisms
- Ability to run DRBD on top of LVM (Logical Volume Management), and support for online device resizing
- Performance optimizations including support for Jumbo frames

Installing DRBD

The basic steps for installing DRBD include:

1. Install DRBD package
2. Edit and copy configuration file (drbd.conf) to nodes
3. Choose a primary node
4. Synchronize the underlying devices
5. Device should now be ready
6. Create a file system if one does not exist
7. Mount DRBD on the primary
8. Heartbeat handles changing the DRBD primary or secondary status as well as the mounting and unmounting of volumes

Installing MySQL

The basic steps for installing MySQL and preparing it to work with DRBD and Heartbeat include:

1. Ensure all MySQL files are installed on the DRBD volume
2. Create 'mysql' group and user
3. Create MySQL directory
4. Install MySQL
5. Shutdown MySQL
6. Unmount the DRBD volume

Installing Linux Heartbeat

The basic steps for installing Linux Heartbeat include:

1. Install Heartbeat V2 package, available at http://www.linux-ha.org/
    a. Ships with many Linux distributions so may already be installed
2. Important configuration files

    a. ha.cf: describes the participating machines and their configuration
    b. haresources: specifies virtual IP and managed services
    c. ipfiles: detects when one of their network links has become unusable, and compensate accordingly
    d. authkeys: three types - crc, md5, and sha1
    e. resource.d/mysql: start and stop script for MySQL

Initialization of the MySQL, Heartbeat and DRBD Cluster

1. Start DRBD
2. Start Heartbeat
3. Node becomes DRBD primary
4. DRBD volume is mounted
5. MySQL starts
6. IP address taken over
7. Next steps
    a. Fail over testing
    b. Benchmarking
    c. Stress testing
    d. Create monitoring framework
    e. Attach MySQL Replication slaves for read scalability

Conclusion

This paper has explored how open source components including MySQL, DRBD and Linux Heartbeat can be combined to create highly available clustered systems. When combined with MySQL Replication, the power of scale-out is realized, which results in a gain of scalability for read intensive applications at a fraction of the cost. This architecture can be extended to include application level partitioning to increase the scalability of the system without compromising its availability. Because all the components are open source, run on commodity-off-the-shelf hardware and are fully supported, the cost savings of designing, configuring and maintaining scalable high availability databases can now be fully realized.

Additional Resources

MySQL and DRBD How-Tos

DRBD How-to

For more a detailed technical guide to help you design, install and configure a MySQL, DRBD and Linux Heartbeat cluster please see:

http://www.mysql.com/products/enterprise/drbd.html

MySQL Support for DRBD

DRBD for MySQL High Availability

MySQL 24x7 Production Support available as an add-on to a MySQL Enterprise Subscription. To learn more, please visit:

http://www.mysql.com/products/enterprise/drbd.html

Copyright © 2007, MySQL AB

MySQL Professional Services

MySQL High Availability Jumpstart

MySQL Professional Services offers design, installation and configuration services for MySQL, DRBD and Linux Heartbeat clusters. For more information about the MySQL High Availability Jumpstart please see:

http://www.mysql.com/consulting/packaged/scaleout.html

About MySQL, AB

MySQL AB develops, markets, and supports a family of high performance, affordable database servers and tools. The company's flagship product is MySQL, the world's most popular open source database, with more than six million active installations. Many of the world's largest organizations, including Google, Sabre Holdings, The Associated Press, Suzuki and NASA, are realizing significant cost savings by using MySQL to power web sites, business-critical enterprise applications and packaged software. MySQL AB is a second generation open source company, and supports both open source values and corporate customers' needs in a profitable, sustainable business. For more information about MySQL, please go to http://www.mysql.com/