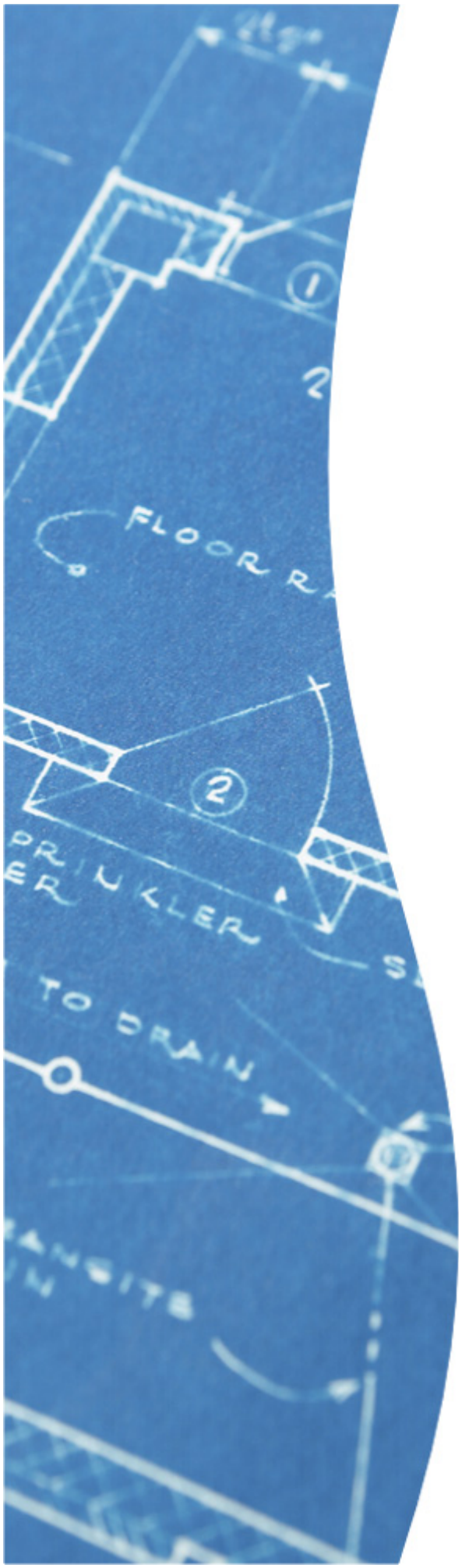

MYSQL™ DATABASE SCALE-OUT AND REPLICATION FOR **HIGH-GROWTH BUSINESSES**

Nick Kloski, Engineered Solutions Group

Sun BluePrints™ Online

Part No 820-6824-10
Revision 1.0, 11/4/08

Table of Contents

MySQL Database Scale-Out and Replication

It is widely recognized that MySQL is the most popular database software in the world. Since its inception in 1995, there have been 11 million product installations around the world in a wide variety of markets. There are more installations of MySQL in use today than any other database architecture. From startup companies hoping to be the next Web2.0 poster child to large global enterprises, the MySQL database architecture has proven to be flexible, extendable, scalable, and more than capable of filling high-capacity database roles in very different venues.

The core MySQL architecture is offered free for anyone to download and use, helping make the MySQL software popular for use in a wide variety of roles. However, the ease by which instances can be deployed often brings challenges: Over time, companies can find themselves dealing with multiple disparate MySQL instances and need to tackle the challenge of how to scale their MySQL installations intelligently both for scale and for resiliency in case of outages.

This paper addresses the primary ways MySQL installations can scale to meet increasing user demands, while still providing the flexibility and ease of use that single installations offer.

Introducing Scale-Out

MySQL costs nothing to use for the core of the database system. Companies of all sizes are encouraged to download and put into production MySQL instances in as wide and as large a configuration as they desire to use. Because of this, many times MySQL is used to do initial proof of concept testing as the back-end datastore for various projects. As those projects prove to be feasible, the MySQL architecture is typically more than able to handle the initial demand generated by associated application.

At some point, however, the demands of the user base require the MySQL database to scale to provide response times that exceed the capabilities of the underlying server hardware. It is at this point that the MySQL architecture needs to extend beyond the initial server to provide more capacity to meet demand. It is at this point that intelligent planning for scaling out will not only solve problems with maintaining response time for end users, but can also provide resiliency against hardware outages that would otherwise bring the database down.

Defining Scale-Out

When Database Administrators (DBAs) talk about *scale-out*, they are referring to being able to serve an ever increasing number of requests against the database. Specific performance of a request once it reaches the database is affected by such factors as application logic and underlying CPU power, memory density, and I/O interconnect.

Note – A discussion of specific database tuning is beyond the scope of this paper. Instead, this paper focuses on Enterprise-level considerations and entire-server scalability.

Over time, as applications become more popular, the capability of one MySQL instance may not be sufficient to serve those needs. In the MySQL database world, the term scale-out refers to improving application performance and scalability on an incremental, as-needed basis by adding multiple replicated database servers on low-cost commodity hardware (Figure 1).

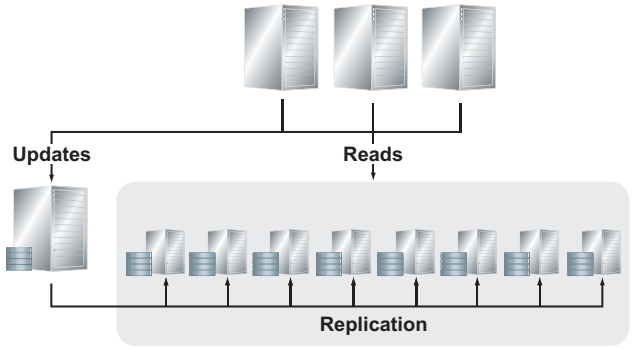

Figure 1. Basic database read scale-out.

This approach is in opposition to another method of scaling: *scale up*. In a scale-up scenario, an existing server's hardware is increased, thus being able to serve more requests by an increase of back-end power.

There are advantages to both methods:

- *Scale-up (also called Vertical Scaling)*
  - Must run on more expensive SMP hardware to allow for scaling.
  - Sometimes runs proprietary software to allow for scaling.
  - Locked in to one hardware/software platform.
  - Once top is reached for a specific hardware platform, entire server must be replaced (*fork lift upgrade*).
- *Scale-out (also called Horizontal Scaling)*
  - Accomplished with commodity Intel/AMD hardware.
  - Runs on open source software/operating systems.
  - Can leverage platform independence to allow scaled MySQL installations to run seamlessly on a variety of servers.

– Adding commodity servers allows for sustained and improved end-user experience.

The MySQL server was not originally designed to run on very expensive and hugely complex servers. Instead, it is designed to easily connect smaller commodity servers together, and includes replication technologies that allow for rapid scale-out of databases on low-cost commodity hardware. The remainder of this paper focuses on MySQL scale-out and replication.

MySQL Database Scale-Out Components

The following components can be used to implement scale-out in a MySQL database architecture:

- *MySQL Replication*
  The definition of MySQL Replication is simple: the duplication of data changes to more than one location. Replication can be either synchronous or asynchronous. With synchronous replication, new data entering a scaled MySQL architecture can be accepted by and written to all database servers at the same time. In contrast, with asynchronous replication the changes and new data go first to one master server, and are sent at a later time to secondary servers.

- *MySQL Cluster*
  The MySQL Cluster product is a *shared nothing* in-memory database that allows an enterprise to increase both redundancy and capacity by adhering to the idea of *more copies in more locations*. By adding data nodes, the underlying database runs on more physical servers and can be accessed correspondingly by more clients. Adding MySQL Replication to Clustering can be used to achieve geographical redundancy (for such purposes as disaster recovery and backups) by taking a running cluster of multiple servers, and replicating that cluster to some other geographical place.

- *Linux Heartbeat*
  The Linux operating system offers a small piece of software that plays an important role in MySQL Clustering. Although it is not required, by adding the heartbeat into the MySQL Cluster, the operating system works hand-in-hand with the database to provide notification to the other side of a cluster when one side goes down, because of hardware failure or other major event. The Linux Heartbeat manages IP takeovers and provides notification to the MySQL framework that a failure has happened and that a failover process needs to be initiated.

- *Load Balancing*
  There are several methods the DBA can utilize to intelligently route the incoming user requests to a scaled MySQL architecture. Through the use of load balancers, incoming user requests get routed to the appropriate MySQL database. In a MySQL scale-out scenario, the load balancer can be either hardware based or

software based. An example of hardware based balancing is an intelligent router that can intercept incoming requests and knows to which MySQL server to send those requests. Software based load balancing is also called *application logic*. The end application, when it receives a read/write/update from a user, can have the intelligence built into the application itself to route the request to the correct MySQL server.

- *Distributed Replicated Block Device*
  Distributed Replicated Block Device (DRBD), as the name suggests, creates block-level replication between different physical servers. The DRBD software is the main component that enables synchronous replication.

In addition, there are a great many shared storage hardware devices and clustering agents for monitoring MySQL replicated clusters, but these are not be covered in depth in this document.

Note – Not all of these technologies are included as part of the MySQL Community download; some services may be fee-based and may require a subscription to MySQL Enterprise. See `http://mysql.com/products/enterprise/features.html` for current information on included features.

The remainder of this paper discusses these components and describes how they can be employed in MySQL scale-out architectures.

Target Uses for MySQL Replication

A combination of these components can be employed to solve problems common to many MySQL implementations. The most simple use case for replication is for pure backup purposes against failure of the Master server. By utilizing a simple Master/Slave setup, data that is sent to the active server gets copied asynchronously to a slave server, and if the active server ever fails, the replication target can take over.

More complex scenarios for replication involve Business Intelligence scenarios where the back-end database can be replicated to another server upon which entirely different analytic data investigation techniques can be applied. Scale-out architectures can provide higher levels of performance and flexible growth. And High Availability (HA) implementations can allow for constant uninterrupted access to database resources, making the implementation resilient to any single hardware outage.

MySQL Database Replication

MySQL 5.1, the latest version of the software as of this writing, provides new support for row-based replication. Previous MySQL versions (5.0 and below) supported statement-based replication. In addition, a combination of statement-based and row-

based replication can be used in MySQL 5.1. The main difference between a *statement* and a *row* for purposes of replication is that a statement is generated through structured query language, while row based replication is based on the raw data in each row.

Before the introduction of MySQL 5.1 support for row-based replication, care had to be taken when utilizing statement-based replication to fully optimize the statement to avoid performance bottlenecks. Such performance problems could be generated through an inefficient query structure that asked for multiple small-result based data, while the overarching query was complex. At times the performance hit for returning such a query was unavoidable. Now in MySQL 5.1, replication can not only be accomplished with statement-based replication, but also with row-based, or even a mixture of the two.

Statement- versus Row-Based Replication

Starting in MySQL 5.1, DBAs have the choice of using statement-based or row-based replication, or a combination of the two. Statement-based replication is best for applications that do not make heavy use of non-deterministic functions or system calls such as SELECT users(). MySQL statement-based replication produces small binary log files, and a binary log can be used to audit the database. Due to the more efficient binary logging, statement-based replication can process more transactions per second in many cases.

With row-based replication, everything (i.e., an entire row) can be replicated. Fewer locks are used for many DML statements on both master and slave servers using row-based replication. In addition, row-based replication can result in faster application of data changes on slave servers, especially for objects with primary keys.

MySQL Replication Internals

Figure 2 illustrates the process of replication in MySQL, assuming a MySQL Master and Slave setup. The `mysqld` daemon on the master server is the process that interacts with the incoming user requests. READS and SELECTS are served out, while UPDATES are written to disk, but also written too what is called a *binary log* or *binlog*. This binlog is indexed and becomes the basis for replicating those UPDATES to the slave server. The binlog from the master server is sent to the `mysqld` daemon on the slave server, which understands that a replication event needs to happen.

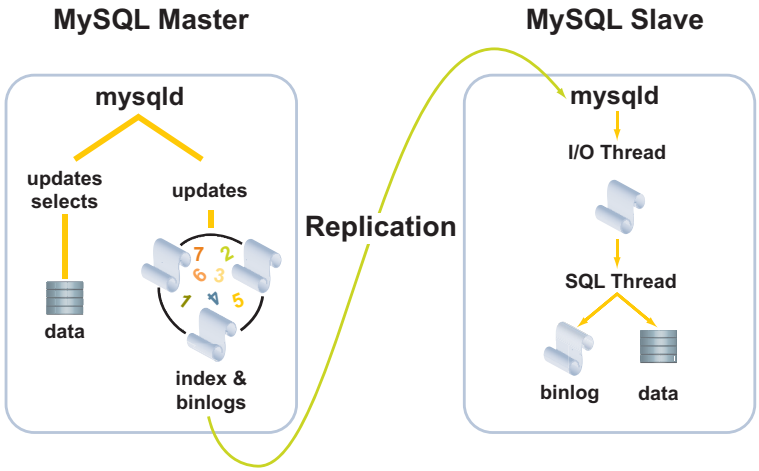

Figure 2. MySQL replication internals.

To handle the incoming binlog stream, MySQL sends the replication binlog to a special process, the *I/O Thread,* which is responsible for emulating an actual write event to that slave server's binlog. On the slave server, this new replication thread (the incoming binlog from the master) is written to a temporary log on the slave server, called the *relay binlog*. The relay binlog is then fed as a normal SQL thread to the database on the slave server, which is responsible for two things. The replication stream, full of changes to the data is written to on-disk storage, and is also funneled at the same time to the slave server's binary log.

This slave server's binlog, if desired, forms the basis for cascading replication down the line to other servers. Once a binlog exists on any MySQL server, whether that server is a master or a slave, that binlog can be used for further replication to other slave servers.

Because the slaves control the replication process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position. Note, however, that in failback scenarios, care should be taken to ensure that the slaves rejoins the replication group successfully, and that any changes made to the online members of the group are also replicated, not just ongoing writes.

MySQL Replication Topologies

Multiple topologies are supported by MySQL replication, as seen in Figure 3. The simplest replication topology is a single master/slave configuration. Configurations with one master and multiple slaves are also supported.

More complex topologies include multi-master configurations, with feature two or more master servers. While these configurations are supported, care must be taken to ensure that dataspaces are not shared between the master servers. If the master is set up incorrectly in a multi-master configuration, overwriting of data can occur.

One topology that is not supported is a multi-source configuration, with multiple masters updating a single slave server.

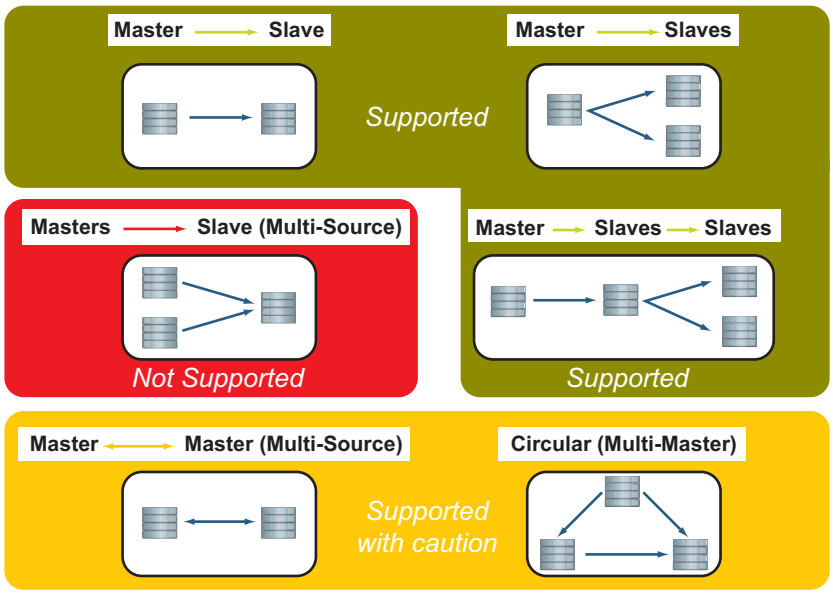

Figure 3. MySQL replication topologies.

Basic Scale-Out Architectures

Over time, many databases need to expand in their ability to respond to increases in user requests. Others may need to be able to withstand hardware failures without downtime, and look for high availability solutions. MySQL replication is a means to help solve these common problems.

Read Scale-Out

The most common use of databases is to serve out information stored in the database to users requesting that data. A common problem in growing databases is the need to be able to serve more reads than writes. This situation occurs when the application accessing the database becomes more popular and overwhelms the originally provisioned hardware. The problem, therefore, is one of being able to server more and more read operations while correspondingly being able to serve slightly more operations that modify the database. This approach is called *read scaling*, and is the most common form of replication.

A basic read scale-out architecture is shown in Figure 4. This architecture adds multiple servers that are designed to handle READ operations, and is well-suited for read-intensive applications. All database writes are served by one master server. The database is replicated on multiple slave servers, and these slaves handle the read requests.

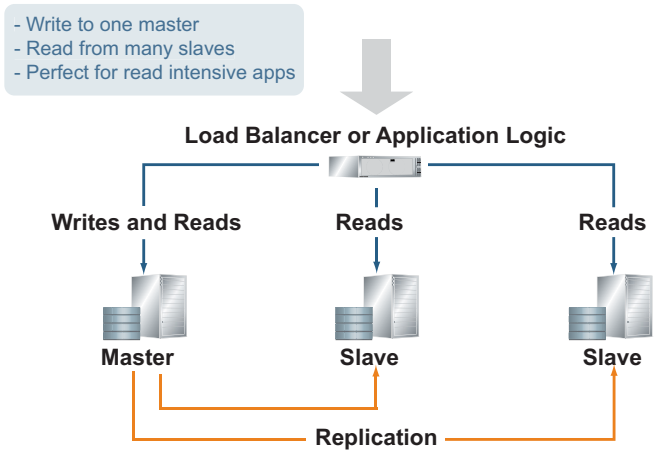

Figure 4. Basic read scale-out architecture.

Asynchronous Replication

It is actually quite expensive, in database processing terms, to make an update or addition to a database. It is much easier to provision for multiple READ operations on multiple servers and allow the main updating to happen on the Master server.

Unlike clustering, MySQL replication sends all updates to the database on the master server, while read operations hit any number of *read-scaled* slave servers. The updates to the master server are made and committed to the database like they always have been. A*synchronous replication* (hereafter called replication unless specified otherwise) enables those updates to be sent to any number of slave servers at a later time than the updates are made and committed to the master server. Thus, updates made to the database may not be immediately available to reads made against the slave servers. This effect can be seen on many large social media Web sites where updates that a user makes are seen by that user, but it takes some time for those updates to show up for other users using the same site.

Load Balancing

The MySQL server includes replication technologies that allow for rapid scale-out of databases on low-cost commodity hardware. The ability to easily connect many smaller servers together provides flexibility, as an organization can use any sort of replication targets to solve a certain problem. Because of this flexibility, however, there are no unifying technologies in the line that act as general MySQL proxy servers for replication. At the current time, while several MySQL proxy applications are in development, none has risen to the level of a fully developed product. As such, the

main issue an organization needs to consider when moving to a scaled-out architecture is the actual separation of incoming user SELECT/READ/WRITE requests to separate MySQL servers.

Load balancing is needed in a MySQL read scale-out architecture to separate user actions before they reach the scaled back-end database servers. When moving to a replicated environment, there is no intermediary that makes this possible. Either a hardware or software-based load balancer is required.

For the hardware-based solution, the load balancer is a piece of specialized hardware that is designed to intelligently read the incoming user requests and includes logic to send those requests to specific MySQL servers on the back end. The same end result can be accomplished with a software-based approach. This requires modification of the application itself, so that the application knows about the replication topologies existent on the back end and routes user READ requests accordingly to the newly added read-only servers.

Application Partitioning

Application partitioning, or *sharding*, is another technique that can be considered to enable MySQL scale-out. The term *shard* means "a piece of," and is used in this context to refer to the distribution of write operations, not reads.

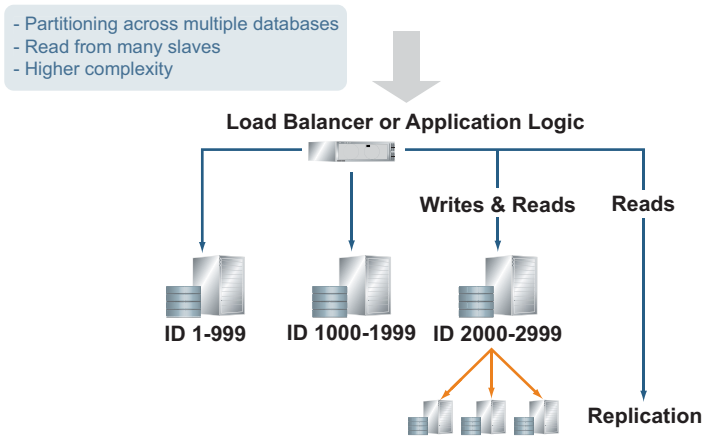

Figure 5. Application partitioning, or sharding.

Whereas the read replication has multiple read-only copies of the database, sharding or application partitioning still has multiple physical servers, but sections of the main database are spread out between those servers (Figure 5). As an example, assume the end users of the database are categorized by userid. In this case, the deployment can be scaled to three servers, with one-third of the userid range residing on each of the

three servers. Assuming the load of all of the users at any given time is equally distributed, then those servers should therefore be only 1/3 as busy as a single server handling all the database functions.

In a pure sharding configuration, as shown in this example, a loss of any server will cause an outage to the users on that particular server. In a pure read-replication setup, the loss of a slave server will only cause a temporary outage for the end user, until the load balancing mechanism sends the displaced users to one of the remaining servers.

Layering Techniques

While the techniques introduced in this paper can be applied as a single technique, they can also applied together to give the benefits of both/all strategies. For example:

- Read-replication through replication slaves helps to scale reads but is vulnerable to the master server having an outage (in which case only reads will be possible).
- Application partitioning/sharding helps to scale writes but is even more vulnerable to server outage since there are physically more servers (in which case no reads or writes will be possible on the failed server).
- Sharding plus replication slaves offers scalability for both reads and writes (where a master server outage will still prevent updates to that server but READs will still be available).

When thinking about layering these techniques, do not consider the servers as the basic building block for replication. Even for small installations, there is often a need for an organization to create another copy of the entire database setup in another location for Disaster Recovery (DR) purposes. If something happens to the main datacenter, then the alternate datacenter can take over.

Replication can be used for geographic redundancy. For any entire MySQL architecture, that architecture can, through replication, be re-created to another entire datacenter. This process is asynchronous, as it is with normal replication, but the failover datacenter will have a reasonably up-to-date copy of your data when the failure happened.

Caution – Geographic Replication from one datacenter to another does not have automated failover, nor automatic resynchronization of data once your main site has come back online!

Linux Heartbeat

The Linux operating system contains a heartbeat utility that allows for two servers (or groups of servers) to use a connection between them to determine if either of the servers has encountered an error and has gone down. This allows one group of servers to act as the main MySQL group and the other to act as stand by, ready to pick up if the main one goes down.

An example configuration employing the Linux heartbeat mechanism is shown in Figure 6. Messages are sent between the master and slave server to determine if the other server is running. A virtual IP address is used to direct incoming requests to the active server. In this example, the system is operating normally and all requests sent to the virtual IP address 192.168.0.50 are directed to the master server's private IP address of 192.168.0.30.

In this configuration, the alternate (slave) server group is updated by asynchronous replication, and therefore has a reasonably updated set of data in case of a failover. However, it is still asynchronous replication: when a failure of the main server group initiates a failover, the standby server group might have slightly out-of-date writes as writes made to the master server may not have been replicated to the slave server.

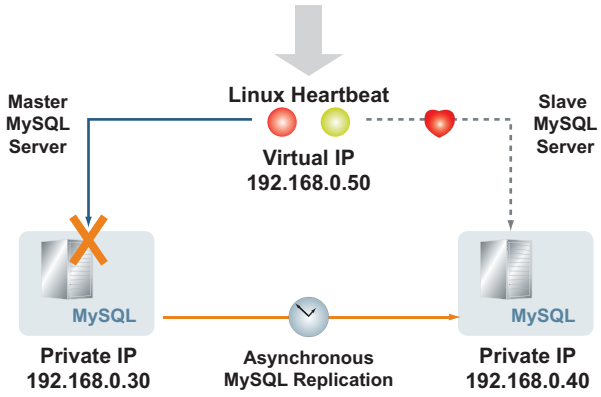

Figure 6. Linux heartbeat and MySQL replication.

If the master server fails, the Linux heartbeat mechanism detects this and automatically remaps the virtual IP address used by the clients to the private IP address of the slave server, as shown in Figure 7. Thus, user requests are automatically and transparently re-routed to the slave server in the event of a master server failure.

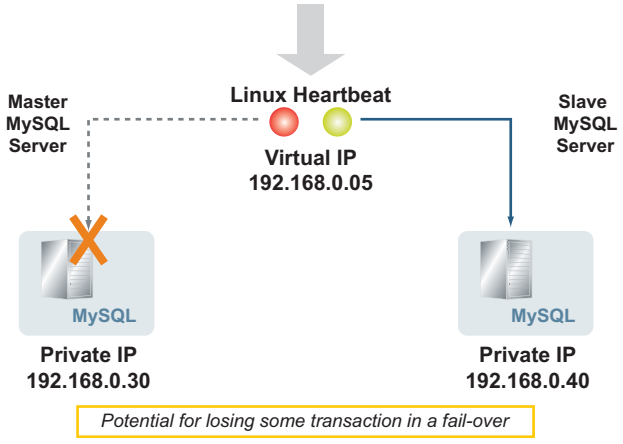

Figure 7. Linux heartbeat and MySQL replication, after a failure occurs.

After a failure occurs and the service has failed over to the slave server, procedures are needed to fail back to the original master MySQL server after the error condition has been resolved. Many times, the failover mechanism from the main servers to the backup/replicated servers is well known and tested, but the recovery process can be

longer and more complex than anticipated. There is no “perform reverse replication” command in MySQL and recovering essentially in a backwards direction can sometimes be complex.

The Linux heartbeat facility typically requires a serial connection between servers to implement. This solution is open source and easy to configure, making it inexpensive to implement and maintain. Once implemented, the virtual IP management is automatic.

Configuring the Linux heartbeat functionality can be useful when deploying groups of servers in separate datacenters. While replication between these groups of servers does not have any facility in the MySQL architecture for automatic failover, the Linux heartbeat can be used to implement auto-failover.

Distributed Replicated Block Device (DRBD)

MySQL replication mechanisms can be used to scale both READ and write (UPDATE, INSERT, DELETE) functions. Geographic replication can also be implemented to provide some security in case of a failure of an entire group of servers. Two main drawbacks, however, still exist with the approaches mentioned up to this point in the paper:

- Asynchronous replication raises the possibility of having a less-than-updated version of the MySQL database serving customers in case of a failover.
- Recovery times for to move back to the *original* database topology is somewhat complex.

Unlike asynchronous replication, the Distributed Replicated Block Device (DRBD) feature in MySQL helps solve these problems by having all copies of all data on all servers be fully updated. Then, if a master server failure occurs, all slave servers have updated copies of the database available.

The DRBD acronym can be defined as:

- *Distributed (D)* — Runs across multiple servers.
- *Replicated (R)* — Performs database replication duties.
- *Block (B)* — Replication happens at a block/hardware level, not at the database level.
- *Device (D)* — Implements a device driver that performs the synchronization, not at the database level.

The DRBD feature of MySQL allows for synchronous replication of the blocks of data underlying the database itself. DRBD runs over standard IP networks between the servers (without the need for special hardware), with failover and virtual IP addresses managed by the Linux heartbeat program (Figure 8). DRBD can offer great performance because the complexity of any database is hidden from DRBD — DRBD cares only about the underlying block data and not about database complexity at all. DRBD does introduce a bit more complexity during the setup phase, but the payoffs later on for replication and recovery more than make up for the initial setup steps.

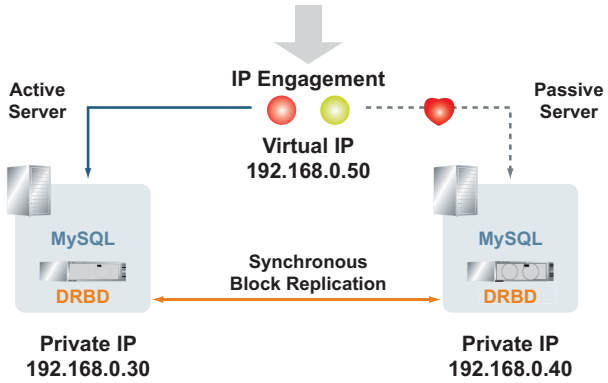

Figure 8. Linux heartbeat and distributed replicated block device (DRBD).

DRBD synchronously updates both servers so that in case of a database server failover there are no out-of-date copies of the database. Also, when the cause of the server failure has been resolved, DRBD will copy, at a block level, the running database back over to the main server, thus making all databases current. This function allows for recovery to be far less complex and time consuming than otherwise would have taken place in a traditional replication scenario. Because the database is current at all times, database administrators do not have to concern themselves with the intricacies of merging two databases back together.

DRBD and Replication

While DRBD can be used as a stand-alone function of MySQL Replication, the real power in this technology is seen in combination with other MySQL features. Using DRBD in conjunction with read replication (Figure 9) allows for the database servers (which are typically more expensively full featured from a hardware perspective) to synchronously record write requests, while read replication slaves serve the bulk of the incoming READ operations. Managed by the Linux heartbeat function, the single active database server can suffer a failure, the passive server will become active, and there is no loss in any user data, nor do any users experience an outage.

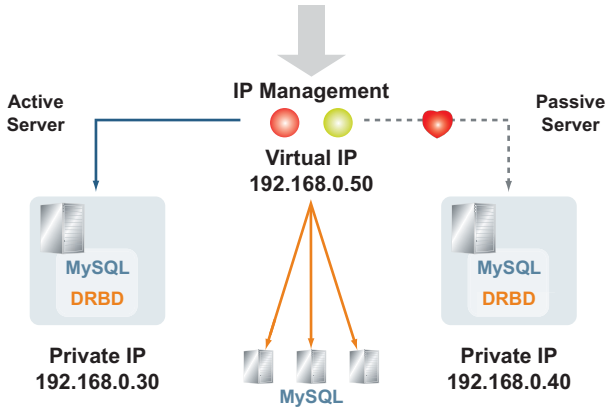

Figure 9. DRBD with MySQL replication.

DRBD Clusters, Application Partitioning, and Replication

One final example brings all of the features and uses cases discussed previously into one topology. Figure 10 shows a topology featuring DRBD, application partitioning, and replication. By using DRBD for server coherence, application partitioning for write scalability, and multiple read replication slaves for read scalability a database can be made fully redundant as well as handle increasing numbers of user requests. If needed, this architecture can also be geographically replicated asynchronously to a failover datacenter for disaster recovery purposes.

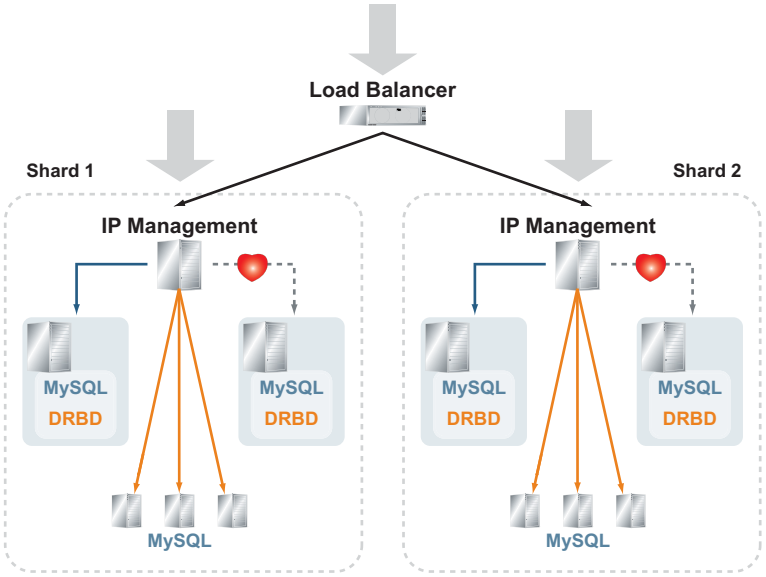

Figure 10. DRBD clusters, application partitioning, and replication.

Guidelines for Implementing Scale-Out

The following guidelines are relevant when planning for MySQL scale-out and replication:

- Don't think synchronously
  Not every access to the database needs to happen at the same time.
- Don't think vertically
  Often times the best performance can be achieved by adding commodity servers to serve specific segments of the architecture.
- Don't mix transactions and business intelligence
  Since it is easy to set up replication slaves, run business intelligence applications against those slaves, saving the main server to serve write requests.
- Avoid mixing hot and cold data
  When setting up replication slaves, make sure to put more static tables onto the read slaves, and keep frequently updated tablespaces on the main MySQL server.
- Don't forget the power of memory
  Sufficient memory is needed to successfully handle incoming transactions. Configuring adequate memory can help avoid many of the problems that would otherwise force an enterprise to adopt a complicated replication strategy.

An overview of the basic setup for MySQL replication is included in “Appendix: Overview of MySQL Replication Setup” on page 18.

MySQL Monitoring and Professional Services

The MySQL organization offers many ways to help evaluate and create replication architectures. On the software side, the MySQL Enterprise Monitor application helps a database administrator see the status of replication targets, determine what problems might cause a replication slave to not receive data, and check the current status of replication activities on a running server. Access to the Enterprise Dashboard comes with a subscription to the MySQL Enterprise Silver product support level, while more advanced features such as the Replication Advisor and Memory Advisor come as part of the MySQL Enterprise Gold support level.

While the core MySQL database is completely free to download and use, additional support is available to help enterprises maximize their MySQL experience. For example, MySQL Enterprise product includes the following components at varying levels of subscription:

- MySQL Enterprise Server software — the most reliable, secure and up-to-date version of the MySQL database software
- MySQL Software Update Service — provides proactive alerts regarding new product releases, security issues and bugs
- MySQL Knowledge Base — a valuable self-help resource allows users to search a comprehensive library of technical articles to resolve difficult problems on the most popular database topics

MySQL Enterprise Unlimited is another option that includes production support across the entire enterprise for one flat fee, and includes support for an unlimited number of MySQL servers.

- MySQL Production Support — provides support from experts inside the MySQL organization who can provide advice on database performance, design, and implementation strategies and can help solve problems quickly and completely
- MySQL Monitoring and Advisory Services — work as a *virtual DBA assistant* to help enforce MySQL-recommended best practices across all MySQL servers

---

Note – See `http://www.mysql.com/products/enterprise/features.html` for more information on MySQL Enterprise features.

Summary

The MySQL database, the most popular database in use today, includes features that businesses rely on for their day-to-day operations of Web serving and enterprise deployments. Though large amounts of money are charged for other databases on the market, the MySQL database offers something those other databases do not: the ability to deploy an enterprise-quality database on virtually any hardware, at no-cost.

As application popularity increases over time and the demands on back-end databases grow, it is necessary to consider how to intelligently grow and scale MySQL deployments. Improperly scaling MySQL installations can adversely affect performance and cause undue complexity in database infrastructure. This paper provided information on approaches to MySQL database scale-out to address these problems. Careful planning using this document as a guide is a starting point. Additional support is available through the vibrant and responsive MySQL user base and via professional support services, such as the MySQL Production Support offerings. With planning, MySQL database deployments can be managed to scale to meet the demands of database growth patterns now and into the future.

About the Author

Nick Kloski is a Web2.0 Solutions Architect in the Web/HPC group in the Systems Technical Marketing Group at Sun. In his 10 years at Sun, Nick has had a wide exposure to both Sun and competitive systems, including systems administration work, over six years of technical Field Service, internal QA testing, and as a member of the Technical Marketing Department. In the role of Web2.0 Solutions Architect, Nick is responsible for being aware of market trends and discovering ways Sun technology can help uniquely solve customer problems.

Acknowledgements

The author would like to recognize Jimmy Guerrero of the MySQL Marketing Team for his contributions to this article.

References

MySQL Database: `http://www.sun.com/software/products/mysql/index.jsp`.

MySQL Documentation: `http://dev.mysql.com/doc/index.html`

MySQL Enterprise:
`http://www.mysql.com/products/enterprise/features.html`

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is
`http://docs.sun.com/`

To reference Sun BluePrints Online articles, visit the Sun BluePrints Online Web site at:
`http://www.sun.com/blueprints/online.html`

Appendix: Overview of MySQL Replication Setup

The following high-level steps provide an overview of the basic setup of MySQL replication. These steps do not offer specific commands needed to create a MySQL replication slave, but are included as a conceptual tool to provide an overview of the replication process.

Note – For more detailed information on replication setup, refer to the MySQL documentation for the specific MySQL version being used. For example, the *MySQL 5.1 Reference Manual*, chapter 15 “Replication” and the *MySQL 5.0 Reference Manual*, chapter 15 “Replication” both describe how to set up complete replication of a MySQL server on releases 5.1 and 5.0 respectively.

---

1. Ensure master and slave servers are configured with unique IDs.
2. Grant replication privileges to the slave
3. Edit master/slave `my.cnf` files (binary logging, etc.)
4. Restart master server
5. Copy initial master data to slave(s):
   - Cold Backup/Restore
   - Mysqldump — master-data option
   - LOCK tables and file copy
6. Issue MySQL `show master status` command to record the index file and its position
7. Issue MySQL `stop slave` command
8. Issue MySQL `change master` command
   - master_host=, master_user=, master_password,master_log_file=, master_log_pos
9. Issue MySQL `start slave` command

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** sun.com

Printed in USA 11/08