High-availability mechanisms in MySQL

Ari J. Flinkman
ari@flinkman.com,

University of Helsinki

Abstract. With MySQL, the popular commercial open source relational database engine, been implemented several mechanisms to provide high availability and resistance to hardware failures. This paper looks at ways to build a fault-tolerant MySQL installation by creating a active/passive setup using either MySQL's standard replication, shared storage or DRBD, the Distributed Replicated Block Device. The paper also takes a look at MySQL Cluster and NDB Storage Engine, which can be used to create a setup with multiple active server instances with redundant, distributed storage for data.

1 Introduction

A computer database is a structured collection of records of data that is stored in a computer system so that a computer program or person using a query language can consult it to answer queries. The records retrieved in answer to queries are information that can be used to make decisions. The computer program used to manage and query database is known as a database management systems (DBMS)

As databases are increasingly deployed in environments which need to be reliable, the need to have high-availability databases has increased as well. A system which can be characterized as highly available can be trusted to work properly in cases of hardware or software failures. This availability is often expressed as a percentage of proper functioning over a period of time, typically a year.

High availability on "five nines" magnitude requires the system to be unavailable for less than six minutes a year. To achieve this, the DBMS must not rely on single pieces of hardware. It must be spread on group of servers, where any server can fail at any time, without impact. As redundant setups can become vulnerable when normal functioning is disrupted, it is important that the system can regain its redundancy as fast as possible.

MySQL is a multithreaded, multi-user SQL DBMS, which enjoys huge popularity. The server is available as free software under the GNU General Public License and as a commercial product from MySQL AB. With the growing number of MySQL installations the need for high availability has grown too. Replication was the first feature to be implemented in MySQL 3.23, and the cluster features were first released in MySQL 4.1. As the cluster features in MySQL 5.1 will be distinctively different to MySQL 5.0, we will focus solely on 5.1.

2 Concepts of Highly Available Databases

The key to high availability is the ability to withstand failures in environment used for producing the service. Hardware and software failures, including the operating system, are examples of failures from which the high availability system must be prepared to survive.

When a DBMS instance fails, the service it provided must be available through other route. This requires process redundancy. With redundant processes, after a DBMS process fails alternate process can still provide the required service.

Process redundancy can be achieved through two different approaches. There can be either only one active process which handles all the transactions, or there can be several. With single active process there must be standby processes ready to become active. This active/standby setup also requires mechanisms to make sure the standby correctly recognizes the situations requiring it to take over the duties of active process [2].

Data redundancy is also required for high availability. Without it the data couldn't be stored reliably, as a loss of single copy could happen because of hardware failure, e.g. with a hard drive failure. Data redundancy can be achieved either by having a storage subsystem which maintains redundancy while presenting the data to DBMS processes as a single copy, or having the DBMS explicitly maintain multiple copies of the data.

Combining the requirement of data redundancy and process requirement has lead to several approaches to produce a proper HA-DBMS. Arguably the most common approach is to have a redundant processes in active/standby setup, accessing a shared storage on a redundant disk subsystem. With a single authoritative set of data the DBMS processes won't end up with conflicting data. If active process writes transactions to a log before committing, standby process can wake up to a clean state regardless of the timing of active process failure.

Having a active/standby setup with per-process set of data is another common way to implement HA-DBMS. This way, the requirement of fault-tolerant physical storage subsystem is eliminated. However, now the standby processes must be kept updated to the most current data. This can be achieved by replicating all the updates active process has to do. When the active server fails and standby has to take over, its own data set will be current.

Replication requires that the initial set of data on all DBMS processes is identical when the replication is started. When standby process receives every update the active process has to do, the data on standby process will stay identical. Replication can also be used to increase the performance of DBMS, as the standby processes can also be used for data retrieval. However, updates must be done through the active process and on every standby process, so the performance benefits only manifest itself on a read-oriented application.

Replication, just as active/standby setup with shared storage, requires a mechanism which will promote a standby process to a active status. If this mechanism fails and standby server mistakenly becomes active while the active process is still functioning, there's significant risk that both active processes will commit to a subset of transactions, thus creating two different sets of data which could be impossible to combine. With shared storage this split-brain situation can be avoided by using a lock file as an additional security mechanism.

Another and completely different approach is to have multiple active processes at once. With shared storage it is challenging to produce a locking scheme with good performance characteristics. Without shared storage it is difficult to keep the set of data synchronous across all the active processes without sacrificing the performance. These problems can be made easier if some features assumed from a DBMS can bargained against performance. These features hard to implement include transactional isolation and atomicity, and guaranteeing referential integrity across multiple tables.

One aspect with HA-DBMSs is the need for client connections to be directed to the active process. This can be done with a load balancer or making the clients aware of several processes, and building the ability select active process in clients' database connection handler. We will assume that some such scheme is at use, and focus on the problems of process and data synchronization.

3 Active/Standby MySQL Configurations

MySQL's storage architecture is designed to be used exclusively from a single active process, therefore clustering solutions based on shared storage and familiar from commercial databases such as Oracle are not possible with MySQL. However, MySQL offers ways to build Active/Standby configurations.

With this limitation in mind, there are two approaches left. Either the data can be replicated from active server to standby server, which is possible on block device level or transaction level, or the storage can be made accessible from standby server as well.

3.1 Data Redundancy with Standard MySQL Replication

MySQL's architecture represented in figure 1 shows how the storage engines are independent components, while the other components are common. Replication feature is implemented in Management Services and Utilities, and it's usable with all storage engines despite their differences.

Active process, the master in replication, writes updates to a binary log. A standby process, the slave, connects to the active process to retrieve the update log. The slave then applies the updates to its own set of data. This process is pictured in figure 2. As replication doesn't require any specific features for pluggable storage engine used, it can be used with any of them, even those which doesn't write the data to disk.

The standby process must be promoted to a active state after the active process dies. MySQL in itself doesn't provide tools for this monitoring and activation, in essence making process redundancy unsupported. However, the process of activating a standby process is fairly simple.

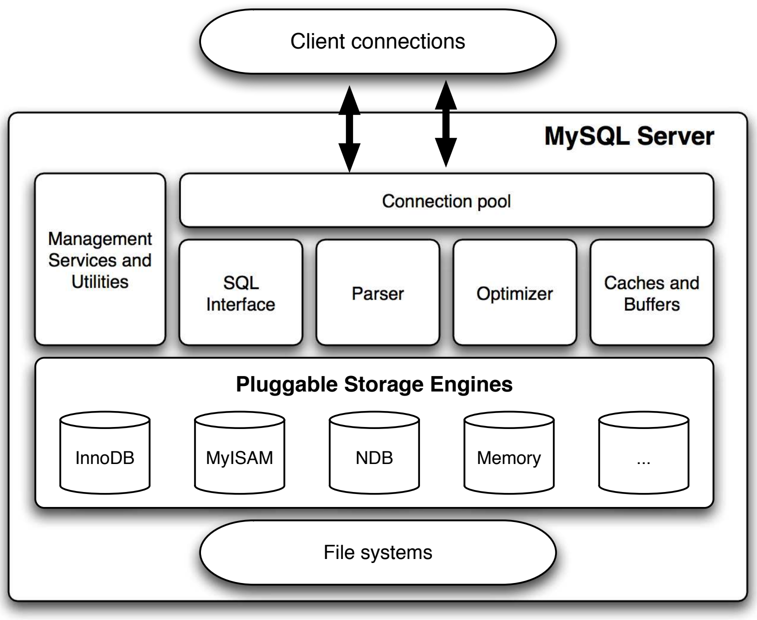

Fig. 1. MySQL Architecture. Image from [5]

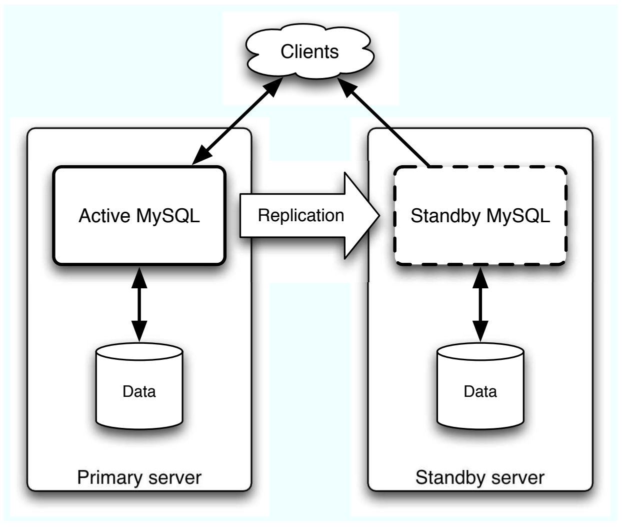

Fig. 2. Standard Replication

Activating the standby process is two-step process. First, the event of failure must be correctly detected, and when it occurs, the standby process must be made active. The monitoring can be quite easily done with readily available tools, such as Linux HA project's Heartbeat. For this purpose, Heartbeat is also the toolkit recommended by the vendor. The actual activation of standby process can also be easily implemented with simple shell scripts.

The actual replication is initiated by the slave process, which will connect to the active process and request updates since last update it received. The active process is not aware of replication status. Replication is neither a consideration when the active process commits to transactions, therefore there's no guarantees to what is the state of replication slave.

This asynchronous approach to replication is a probable cause for lost transactions. If the node active process runs on fails terminally, there might be transactions the active node was committed to, but which weren't replicated to the standby server. In such case, these transactions will be permanently lost.

Asynchronous replication has also the benefit of making it possible to run the replication over very long distances if required. This makes it suitable for uses such as creating off-site data redundancy for a disaster recovery purposes. Another benefit, which is also depicted in figure 2, is the ability perform read-only queries on standby server, which can be used to increase total performance of the system. However, switchover time in case of failure might be unexpectedly long [6].

3.2 Shared Storage

In a shared disk setup several processes share the same physical storage media, although in active/passive setup only the active process may write to the storage. As MySQL creates its data files and metadata on top of the filesystem, the filesystems might cause additional restrictions to this. This approach is illustrated in figure 3.

The basic assumption is that underneath the data partition is a redundant storage subsystem which can be visible to all DBMS nodes. This can be achieved by attaching the storage subsystem to the node by a bus which supports multiple host devices, such as SCSI, or a storage area network.

The safe approach is to have only the node, on which the active process runs, to mount the shared storage as read/write device. This way, when the active process fails, a standby node will remount the shared storage as read/write, and activate the DBMS process. If a file system allowing multiple concurrent read/write hosts is used, the mount manipulation is no longer necessary. Making the switchover to happen automatically requires the same approach as with replication.

With shared storage, the shared storage will have exactly the same data set for active and standby processes. When active process fails, the standby process can continue without risk of losing some recent transactions, which might happen with asynchronous replication.

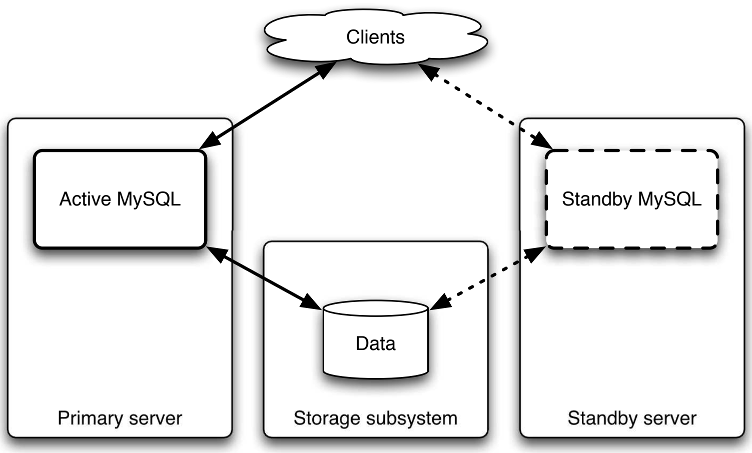

Fig. 3. Shared storage

The problem with shared physical storage is the susceptibility to mishaps caused by software. If the active process' nodes memory gets corrupted in a way which causes it to overwrite the data set with garbage, the data will be lost despite its redundancy on the physical level. The standard replication would keep the standby process's data intact, although the overall need to switch over to the standby process might go unnoticed.

Unlike replication, shared storage is unsuitable for use with pluggable storage modules which doesn't use local disk, such as memory and NDB.

3.3 Shared Data with Distributed Replicated Block Device

Distributed Replicated Block Device (DRBD) is an open source Linux kernel block device which creates consistent view of data between multiple systems through synchronous replication over network. This setup, in relation to MySQL server, is illustrated in figure 4.

In normal operation, the writes to the DRBD device are replicated to all DRBD nodes over the network. Reads are conducted locally as long as the local disk is available: when it fails, the read is done from backing DRBD instances [3]. Once more, the required fail-over mechanics must be separately implemented.

Overall, this is combination of the behaviors of standard replication and shared storage. Normally, only active process writes to the storage on DRBD device, and when active process fails, the standby node must first mount the DRBD in read/write mode and after that, activate its DBMS process.

Compared to shared storage, the biggest benefit is to have no requirement of special expensive hardware. When compared to standard replication, DRBD decouples data redundancy from process redundancy. With standard replication, the DBMS process is tied to its set of data – active process must also fail when

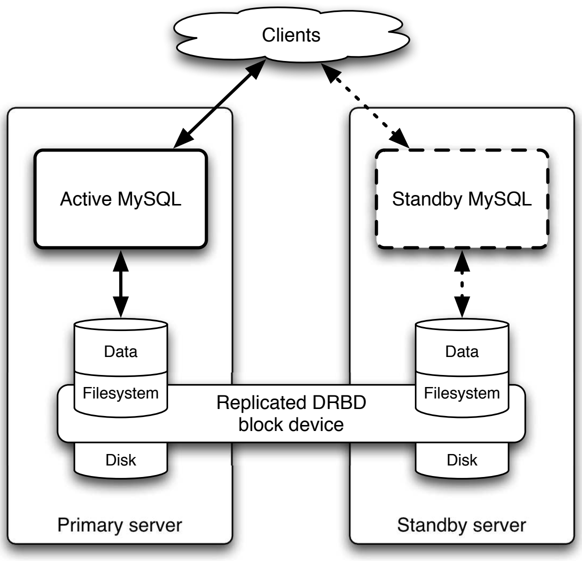

Fig. 4. Virtual shared storage using DRBD

data on its local storage fails. If the local disk on active process node fails, DRBD will continue with the data retrieved from standby node.

Unlike with standard replication, DRBD does require reasonably close proximity between the replication peers. When the local disk subsystem fails, DRBD's read operations from standby systems disk subsystem are constricted by latency between the systems, which might cause significant performance loss. Vendor claims this approach can result in switchover times under 30 seconds [6].

4 MySQL Cluster

MySQL cluster has several concurrently active processes on at least three separate nodes. It is implemented by creating distributed and redundant storage engine to work as standard MySQL's pluggable storage engine. It is designed to be able to survive node failure without the need to reconfigure nodes to regain operational capability.

4.1 Overview

MySQL cluster is the regular MySQL server as depicted in figure 5, just using the NDB Storage Engine as it's storage. As from the DBMS standpoint the NDB contains a single, internally redundant set of data, it can be considered as a shared storage solution. In a way it's very similar to the DRBD described in

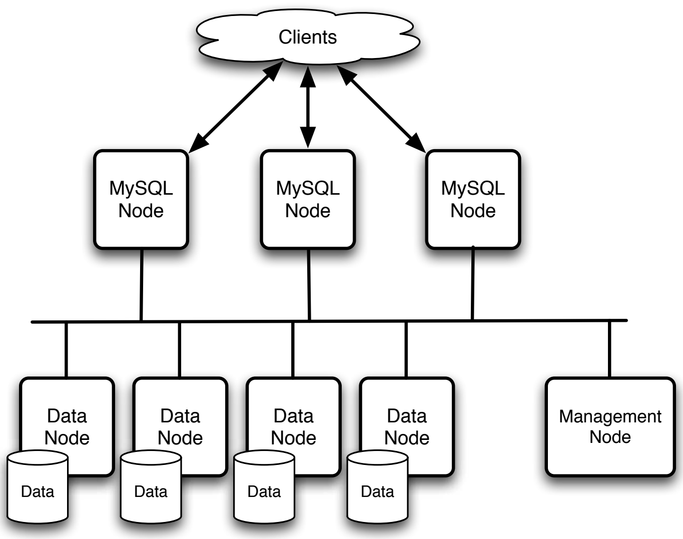

Fig. 5. MySQL Cluster

section 3.3, as both are physically shared-nothing, but they logically they act as a shared-storage.

In MySQL Cluster there's three kinds of nodes: data nodes, MySQL nodes, and management nodes. The bulk of the processing, data replication and fail-over-mechanism is done on the data nodes. We will discuss the data nodes in the next subchapter in more detail.

MySQL nodes, which run on standard MySQL servers, are the nodes that applications can connect to. Use of NDB as a pluggable storage engine is the only separating factor from standard MySQL server. Essentially, SQL nodes provide the "face" to the redundant storage cluster. They connect via the NDB API to data nodes, which do the early processing on the queries and return the results for final processing at the MySQL nodes. When a query comes into a standard MySQL server process, it is sent to a background server, the storage node.

Management nodes have two important roles. There's only one management node active at a time, but there might be several stand-by nodes ready to replace it. Management nodes' primary role is to act as a coordinator for nodes in the cluster, maintaining configurations and status database. In addition the active node is used to determine whether the cluster has quorum. In situations where the cluster is split, the segment with active management node remains active.

As the features of MySQL cluster are evolving, we're focusing on version 5.1 which is currently in beta. It has several significant improvements over previous version, such as support to variable length records and disk-based tables [8, 1]. In previous versions NDB has been strictly main-memory database, which greatly limited the amounts of data a storage node could handle. With disk-backed storage this limit has grown by order of magnitude. Unfortunately, also the recovery of failed data node has become more difficult proportionally to the increase of stored data.

4.2 NDB Storage Engine

NDB Storage Engine is a replacement for commonly used storage engines, such as MyISAM and InnoDB. While MyISAM and InnoDB store data on a local filesystem, NDB uses a group of nodes for this. It also achieves data redundancy by saving multiple copies to several nodes.

As is customary with MySQL's pluggable storage engines, NDB's set of supported features also varies from other storage engines. For example, NDB doesn't support foreign keys or full-text indexes. There are also some more subtle limitations, such as inability to handle on-line schema changes properly.

The operation in normal conditions is synchronous. The transactions are committed when all nodes expected to store the data have committed to it. In case of node failure the NDB cluster recognizes the missed heartbeats and drop the node from their assumed synchronous operation [4].

The inclusion of disk-backed storage for data also caused different approach to be required for recovery after data node failure. With database limited to main memory recovery and requried data resynchronization after failure was simple operation of copying everything from other node in the node group. This approach is incapaple of copying probable data amounts fast enough when using disk-backed storage. The solution from this problem comes from modified iSCSI synchronization algorithm, as it provides simple operations for block level synchronization which can be adapted for use with row-based database [9].

4.3 Data Partitioning

In NDB Storage Engine, the data is evenly divided among the storage nodes. This partitioning is done by using a hash function over the primary key to determine which storage nodes should be responsible for that row. If the table doesn't have primary key, one is created while creating the table. This way, every storage node ends up having r/n parts of the data, where r is the number of replicas (two or more), and n is the number of storage nodes (2 to 48).

NDB data nodes are automatically split into node groups. The configured number of replicas determines the number of nodes in each group. Each node in node group will have complete and identical set of data [1]. Therefore the server remains operable without data loss as long as one node in each node group stays operational.

4.4 Performance Considerations

Considering performance, the NDB has potential to scale well by increasing the number of noders. The key consideration is whether the queries are of nature which can benefit from parallel data nodes, each having a partial set of data.

As processing a simplest of queries require contacting data nodes over the network, the network's performance becomes essential. The vendor claims Gigabit Ethernet to be sufficient for setups with up to eight nodes, after that using a dedicated cluster interconnect is suggested.

It would be premature to make claims of performance as MySQL Cluster 5.1 is still in pre-release versions, and its features significantly differ from 5.0. It is reasonable to expect that 5.1 will have different performance characteristics, as it parts from previous limitation of being constrained to main memory. Vendor claims MySQL Cluster will be fast when the queries are of nature benefiting from the parallel nature, and respectively slow when the queries require lots of coordination from different nodes [7].

In theory, the NDB should remain operational without data loss as long as one replica of data remains accessible in data nodes. According to the vendor, a failure within the means of survival shouldn't render NDB inoperable for more than three seconds [6].

5 Summary

It is difficult to choose a flawless approach for making MySQL highly available, even from this plethora of available methods. Replication is intended for use over long distances, and it's design of connections initiated by slaves is sound unless certainty is required whether the transactions have been replicated to slaves. Still, it can be a good companion for other methods.

Shared storage is always problematic when used with databases which are not designed for it. As MySQL is one of those, this approach is deducted to moving the mount from active system to standby system when required, to guarantee there are no rogue writes to the data.

Replication with DRBD might be good low-cost alternative to shared storage, but it is limited to Linux. Running the database on data mounted from remote system isn't either very beneficial if any performance is desired, requiring switching over the active server in any failure.

MySQL Cluster is interesting, and on paper it delivers impressive promises about performance and recovery. However, its features as a storage engine are even more limited than MyISAM's, which might be problematic. Another interesting detail is lack of detailed, independent case studies. Not only is this lack apparent with currently unreleased MySQL Cluster 5.1, but also the previous versions. With history dating back to 1996 [1], this should raise questions.

However, the lack of silver bullets is perfectly understandable. For most requirements one of the methods could prove satisfactory, even if the five nines' level of availability remains unaccomplished. MySQL Cluster is definitely the best bet in such quest, and it might prove iinfluential for future designs of HA-DBMSs if it lives to deliver all the promises made.

References

1. Davies, A., Fisk, H.: MySQL Clustering
   MySQL Press, (2006)

2. Drake, S.:, Hu, W., McInnis, D. M., Sköld, M., Srivastava, A., Thalmann, L., Tikkanen, M., Oystein, T., and Wolski, A.: Architecture of Highly Available Databases ISAS (2004) 1–16

3. Ellenberg, L. and Reisner, P.: DBRD v8: Replicated Storage with Shared Disk Semantics Proceedings of the 12th International Linux System Technology Conference (2005)

4. Lentz, A., Smith, S.: Introduction to MySQL Cluster: Architecture and Use
   5th System Administration and Network Engineering Conference
   `http://www.sane.nl/sane2006/program/final-papers/R7.pdf`

5. MySQL 5.1 Reference Manual
   `http://www.mysql.com/5.1/reference`

6. MySQL and DRBD High Availability Architectures
   MySQL Technical White Paper (2007)

7. MySQL Cluster Evaluation Guide
   MySQL Technical White Paper (2007)

8. New Features in MySQL CLuster 5.1
   MySQL Technical White Paper (2006)

9. Ronström, M., Oreland, J.: Recovery Principles of MySQL Cluster 5.1
   Proceedings of the 31st international conference on Very large data bases (2005) 1108–1115