MySQL Cluster Architecture Overview

High Availability Features of MySQL Cluster

A MySQL® Technical White Paper
April, 2004

Mikael Ronström, MySQL AB
Lars Thalmann, MySQL AB

Table of Contents

Executive Summary

This paper describes the high availability and reliability features of MySQL Cluster, an in-memory clustered distributed database management system. MySQL Cluster is built on a shared-nothing architecture and includes advanced features such as failover, node recovery, synchronous data replication and no single-point-of failure. This paper describes different usage scenarios and describes the architecture and approach used to provide high availability.

Introduction to MySQL Cluster

MySQL Cluster is designed to provide 99.999% availability using a distributed node architecture with no single point of failure. The system consists of multiple nodes that can be distributed across machines and regions to ensure continuous availability in case of node or network failure. MySQL Cluster uses a storage engine, consisting of a set of storage nodes to store data which can be accessed using standard SQL with MySQL Server.

MySQL Cluster consists of three kinds of nodes (Figure 1):

1. The storage nodes (SN) are the main nodes of the system. All data is stored on the storage nodes. Data is replicated between storage nodes to ensure data is continuously available in case one or more storage nodes fail. The storage nodes handle all database transactions.

2. The management server nodes (MGM) handle the system configuration and are used to change the setup of the system. Usually only one management server node is used, but there is also a possibility to run several. The management server node is only used at startup and system re-configuration, which means that storage nodes are operable without the management nodes.

3. The MySQL Server nodes are MySQL Servers accessing the clustered storage nodes. MySQL Server provides developers a standard SQL interface and MySQL Server in turn handles sending requests to the storage nodes eliminating the need for low level programming within the application.

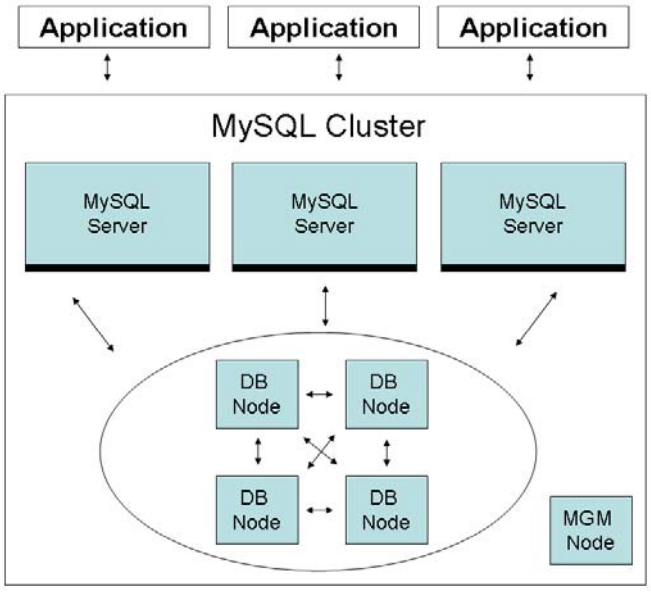

Figure 1: Node Architecture

A MySQL Server in a MySQL Cluster is connected to all storage nodes and there may be multiple MySQL servers in the same MySQL Cluster. All transactions executed on the MySQL servers are handled by the common set of storage nodes. This means that as soon as a transaction has been executed on one MySQL server, the result is visible through all MySQL servers connected to the MySQL Cluster.

High Availability System Architecture

The node architecture of MySQL Cluster has been carefully designed for high availability:

- MySQL Servers are connected to all clustered storage nodes. If a storage node fails, then the MySQL server can easily use any other storage node to execute transactions.
- Storage node data is replicated on multiple storage nodes. If a storage node fails, then there is always another storage node storing the same information.
- Management server nodes can be killed and restarted without affecting the ongoing execution of the storage nodes. The management server node is only used to send configuration information to the database and MySQL Server nodes, and when the storage nodes are up and running, the management servers can be stopped or killed.

The storage and MySQL Server nodes need the management server node when they start, since they begin by connecting to the management server to get the configuration of the MySQL Cluster.

Designing the system in this way makes the system reliable and highly available since there is no single point of failure. Any node can be killed without affecting the system as a whole. An application can, for example, continue executing even though a storage node is down.
As will be described soon, other techniques and algorithms are used to increase reliability and availability of the database system including:

- Data is synchronously replicated between all storage nodes. This leads to very low fail-over times in case of node failures.
- Nodes execute on multiple hosts, making MySQL Cluster operate even during hardware failures.
- Nodes are designed using a shared-nothing architecture. Each storage node has its own disk and memory storage. (There is also the option to share disk and memory when running several storage nodes on the same computer.)
- There is no single point of failure. Any node can be killed without any loss of data and without stopping applications using the database.
- Applications are connected to MySQL Cluster using MySQL Server which gives the application:
    - Data independence, meaning that the application can be written without requiring knowledge of the physical storage of the data. Data is stored in a storage engine which handles all of the low level details such as data replication and automatic failover.
    - Network and distribution transparency, which means that the application program depends on neither the operational details of the network, nor the distribution of

the data on the storage nodes,

- Replication and partition transparency, meaning that applications can be written in the same way regardless of whether the data is replicated or not, and independent of how the data is partitioned.

- A standard SQL interface that is easy for developers and DBAs to use without requiring any low level programming to achieve high availability.

These features make it possible for MySQL Cluster to dynamically reconfigure itself in case of failures without the need for custom coding in the application program.

A Sample Configuration

MySQL Cluster is also highly configurable so you can distribute and configure the MySQL Cluster to suit your particular situation. You can change the number of computers it runs on, the number of storage nodes, the number of applications, how the storage nodes should be distributed on the computers, and much more. As an example configuration (Figure 2), lets say we have four storage nodes $SN_1, \ldots, SN_4$ executing on four computers $C_1, \ldots, C_4$. Lets suppose we want to run two MySQL Server instances M_1 and M_2 on two other computers C_5 and C_6. A management server node MGM_1 is executed on computer C_5.

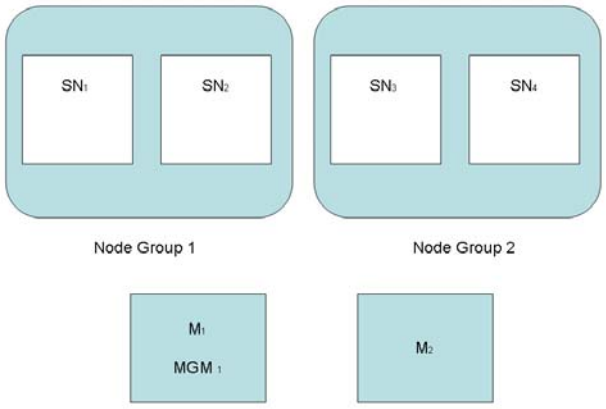

Figure 2: Example Configuration

All database tables are horizontally divided into partitions, which are stored on the storage nodes. To introduce data redundancy, let us use two system replicas (the system may by configured to have anything from one up to four system replicas). This means that all data exists in two copies. In the example configuration above, storage nodes SN_1 and SN_2 will store the same partitions and so will SN_3 and SN_4. The sets $\{SN_1, SN_2\}$ and $\{SN_3, SN_4\}$ are called node groups. A node group is responsible for part of the data stored in the system.

If all nodes in a node group fail and their file systems are cleared, then the partitions (data) they are responsible for its lost. In our example configuration there are two nodes in each node group so there is always a backup node in case a node fails.

Synchronous Replication

All data in the database is replicated on several storage nodes in all storage nodes of the same node group. The number of replicas is decided when MySQL Cluster is started by the database administrator. It can be set from one (no replication) up to four replicas.

Data is synchronously replicated during transactions, i.e. the effect of each transaction is propagated to all the appropriate storage nodes during the transaction.

When the transaction is going to be committed, a request is sent to all storage nodes being involved in the transaction. When all nodes indicate that they are ready, the transaction becomes committed and the application gets informed of the success of the transaction.

The synchronous replication of data means that if a node fails, the fail-over time for another node to take-over is less than a second since all data is already replicated.

To ensure that the database is always consistent, if a storage node fails during a transaction, the transaction is aborted and the application is informed about the transaction so that it can restart the transaction as appropriate.

Failure Detection

There are two ways to detect failed nodes: communication loss and heartbeat failure. In both cases, a message is sent to all storage nodes and a network partitioning protocol is used to determine if there are enough nodes left to continue running the MySQL Cluster. Note that if some nodes are reported as failed, it might in fact be the case that there are two parts of the cluster which have lost all connections between them. In this case, we can not allow that both of them to stay alive, since this could cause database inconsistency. MySQL Cluster ensures applications remain available using the network partitioning protocol to automatically selecting one part of the cluster which continues to execute. All nodes in the other part of the cluster are automatically restarted and connect to the cluster as new nodes.

Communication Loss

MySQL Cluster nodes are connected via different communication protocols. Currently TCP, Scalable Coherent Interface, OSE, and Shared Memory are all implemented and used. Normally, all storage nodes are connected with each other and every application node is connected with every storage node. (Except for shared memory, these connections can be used to connect nodes residing on different computers.) If a storage node notices that a connection between two nodes is lost, then all other storage nodes are immediately informed and they jointly classify the node as failed. Failed nodes automatically restart and connect to the MySQL Cluster as a new node, leaving the application unaffected. Communication loss is the fastest way to detect that a node has failed.

Heartbeat Failure

There are also node failures which can not be detected by communication loss, e.g. disk problems, memory problems, and processor exhaustion. These failures cause a node to work improperly, but do not destroy the node connection to the rest of the MySQL Cluster. A heartbeat protocol is used to detect this kind of failure.

The storage nodes are organized in a logical circle (Figure 3). Each storage node sends heartbeat signals to the next storage node in the circle. If a storage node fails to send three

consecutive heartbeat signals, the next storage node will characterize the storage node as dead. The failed node is reported to all storage nodes, which jointly classify the node as failed.

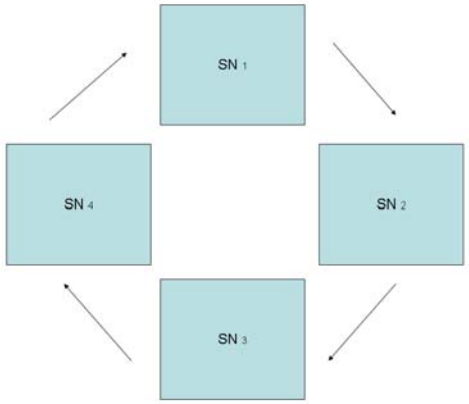

Figure 3: Heartbeat Protocol

Network Partitioning

Whenever there are failed storage nodes, the remaining nodes use a network partitioning protocol to check whether they are in majority by having more than one node in each node group or by having more than half of the nodes available in each node group. This ensures that transactions are fully committed by eliminating the possibility that one node group acts alone and only commits part of the transactions.

To explain what could happen if no network partitioning protocol was implemented, consider again the example configuration with four storage nodes. Suppose all connections were lost between {SN_1, SN_3} and {SN_2, SN_4}. Since both sets are storing all partitions, each set has access to the whole database. If both sets were allowed to survive, then each set could update different information, thus leading us to two inconsistent databases.

The network partitioning protocol solves this problem by ensuring that a surviving set of storage nodes must be a majority of the storage nodes. In the particular case, where a surviving set of storage nodes are exactly half of the nodes an arbitrator is used to cast an extra vote.

Any management server or application node can be configured to be an arbitrator. If several nodes are configured as arbitrators, then they take turns in being the official arbitrator. The storage nodes keep track of the current official arbitrator automatically and if the official arbitrator gets killed a new one is automatically assigned by the storage nodes.

For the example configuration, MGM_1, M_1, and M_2 could all have been configured as arbitrators and the official one would have supported either the set {SN_1, SN_3} or the set {SN_2, SN_4}, provided that at least one set is still connected to the official arbitrator. MySQL Cluster manages this automatically so no extra coding or management is required.

Determining Failure Order

In the rare occasions when multiple nodes fail at the same time, a two-phase commit failure determination protocol is used to determine the order in which the nodes have failed to ensure that they can be safely restarted.

The protocol executes in two steps. First, all nodes send a list of all the storage nodes they consider to have failed to a master storage node. (In many of the MySQL Cluster protocols there is a dedicated master node. In case the master fails, a new one is immediately appointed.) Second, the master determines and informs all nodes of the node failure ordering.

Node Recovery

If there are some failed storage nodes, then they automatically get restarted using a node recovery protocol which provides the restarting nodes with data from the surviving nodes.

Single Node Recovery

If one storage node fails, that node gets restarted by asking a backup node (i.e. a node belonging to the same node group as the failed node) for the partitioned data it should store. The backup node sends this information partition-by-partition to the restarting node.

The restarting node is ready to start serving transactions immediately when it restarts. If a transaction wants to read data from a partition, then it is already automatically directed to the primary node for that partition, i.e. the node mainly responsible for the partition. Since this node is restarting it is not primary and need not concern itself with this request.

If a transaction wants to update a partition already copied to the restarting node, then this is being done in the standard way. That is, firstly the primary partition is updated and then the backup partition.

If the transaction wants to update a partition not already transferred to the starting node, then this gets updated in the primary node only, since the partition will be transferred to the restarting node later anyway. Since information is only required to be updated in one place, this may actually make the database faster compared to when the node has recovered and is up and running again.

Multiple Node Recovery

If multiple nodes need to be recovered, then the order determined by the failure determination protocol is used to determine the order in which they should be recovered. The master node then instructs one node at a time to do a node recovery in the same way as for single node recovery.

System Recovery

System recovery is the process of recovering the whole system after a system failure.

Logging

During normal operation a log is written to disk storing all database operations (inserts, deletes, updates, etc). The log is stored on the file system for each storage node and is only used in case of system failures, i.e. if all storage nodes fail at the same time. Since the log contains all database operations, it is used to replay history to get the database up-to-date during a system recovery.

Local Checkpoints

Since the log quickly grows with the number of operations, a local checkpoint protocol is used to remove the tail of the log. The local checkpoint algorithm creates a transaction consistent snapshot of all node data to disk. Applying the log after loading this snapshot gets the database up-to-date.

Global Checkpoints

Since MySQL Cluster is a main-memory database, transactions are first committed to main memory. As long as there are still live nodes in all node groups, the replication makes the data secure. To recover from system failures (when all nodes fail), MySQL Cluster flushes the log to disk during a global checkpoint, sometimes called a group commit. After the global checkpoint, all transactions are also committed to disk.

To control the commit state of a transaction, MySQL Cluster assigns a global checkpoint identity to each committed transaction. The global checkpoint identity specifies which global checkpoint the transaction belongs to and can be used to verify that the transaction has been committed to disk. If, for instance, global checkpoint 15 is finished, then the result of all transactions with GCI less than or equal to 15 have been saved to disk.

System Recovery

System recovery is handled in two steps. First the local checkpoint snapshot is loaded from disk by each storage node. Then the log is read from disk and executed on each storage node to bring the database up-to-date.

System recovery recovers all transactions with a global checkpoint identity less than or equal to that of the most recent finished global checkpoint. This ensures that all transactions up to the most recent global checkpoint are safely recovered.

Replication and Partitioning Transparency

When the application wants to execute a new transaction, MySQL Cluster uses one of the storage nodes to execute it. If the storage node is down, the transaction is automatically sent to another storage node. A round-robin algorithm automatically selects a storage node to use for the transaction.

Failure Scenarios

Let's consider various failures and how MySQL Cluster handles them.

MySQL Server Node Failure

If a MySQL Server crashes, it can be restarted and reconnected to the MySQL Cluster. The restarted MySQL Server can connect to any storage node. During the time this server is down, other MySQL Servers in the MySQL Cluster provide the same database service.

Storage Node Failure

If a storage node crashes, then all other storage nodes are informed about this either by lost communication link between the nodes or by lost heartbeats. Since data is typically replicated, there is always another storage node available to service the transaction requests.

Management Server Node Crash

Since the storage and MySQL Server nodes are not dependent on the management server for their execution, the management server may fail and be restarted any number of times without affecting the running MySQL Cluster.

Connection Failures

If the connections between the storage nodes are broken, the nodes get information about failed storage nodes. Connection loss is handled by the same mechanisms as node failure. First the two-phase node failure protocol is used to determine which nodes are unreachable, and then the MySQL Cluster reforms with the nodes possible to connect to.

The connections have a fail-over system to handle communication failures. Using this system, TCP/IP connections has a fail-over time of about 100 milliseconds, and the Scalable Coherent Interface connections has a failover time of about 100 microseconds. The connection fail-over system effectively hides PCI card problems, cable problems, and switch failures by routing messages through other connections.

Disk Failures

All storage nodes store their own data. If a node detects that its file system has become corrupted, it stops executing. After the file system has been cleared, the node is restarted using the node recovery protocol.

Conclusion

MySQL Cluster is a high availability database built using a unique shared nothing architecture and a standard SQL interface. We have described some of the technology behind the high availability features of MySQL Cluster.

MySQL Cluster tolerates failures of several storage nodes and reconfigures itself on the fly to mask out the failures. The self-healing capabilities, as well as the transparency of data distribution and partitioning from the application, result in a simple programming model that enables database developers to easily include high availability in the applications without complex low-level coding.

About MySQL

MySQL AB develops and markets 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 5 million active installations. Many of the world's largest organizations, including Yahoo!, Sabre Holdings, Cox Communications, The Associated Press 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, with dual licensing that supports open source values and methodology in a profitable, sustainable business. For more information about MySQL, please go to www.mysql.com.