the Availability Digest

MySQL Clusters Go Active/Active

December 2006

Introduction

MySQL (www.mysql.com) is without a doubt the most popular open source database in use today. Developed by MySQL AB of Sweden and Cupertino, California, it boasts over 4,000,000 installations around the world.

MySQL has recently entered the continuous processing arena with its introduction of the MySQL Cluster. Based on technology acquired from Alzato, a small venture capital company, MySQL cluster was introduced in late 2004. It is available under an open source licensing agreement as a plug-in module for the standard MySQL environment.

It incorporates an active/active NDB storage engine to achieve availabilities in the five 9s range. A unique characteristic of a MySQL Cluster is that the database is totally memory-resident. Therefore, it offers very high performance with database operations measured in just a few milliseconds.[1]

Though all nodes in a MySQL Cluster must be collocated, disaster tolerance can be provided by connecting multiple geographically-dispersed MySQL Clusters in a distributed active/active network using asynchronous replication.

Architecture

MySQL Cluster employs a multinode shared-nothing architecture to achieve its high availabilities. A node is a process running on a host computer. A host is any commodity computer that can support MySQL.[2]

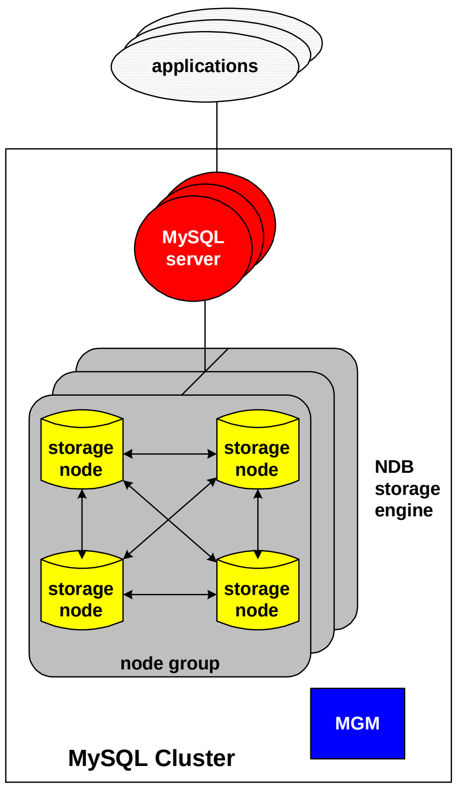
A host can support multiple nodes. The allocation of nodes to hosts provides a level of "redundancy tailoring" to meet the specific availability needs of an application. The greatest availability is achieved by having each node reside on its own host computer. However, important availability/cost tradeoffs can be achieved by allocating multiple nodes to a single host.

There are three types of nodes in a MySQL Cluster – storage nodes, server nodes, and management nodes. A cluster can contain up to 64 nodes, 48 of which can be storage nodes.

[1] A Guide to High Availability Clustering, MySQL Business White Paper; June, 2004.
[2] Ronström, M., Thalmann, L., MySQL Cluster Architecture Overview, MySQL Technical White Paper, April, 2004.

Storage Nodes

The storage nodes form the heart of a MySQL Cluster. There can be up to 48 storage nodes in a cluster. These nodes are organized into *node groups* of up to four storage nodes each. A node group is a single logical storage unit. The set of node groups in a MySQL Cluster forms its highly reliable NDB storage engine.



Each storage node stores its database in main memory. Therefore, access is very fast, with response times measured in a few milliseconds. Periodic asynchronous checkpoints of the database are made to disk to support recovery. However, main-memory storage means that the amount of data that a node group can store is limited by the amount of available memory in its storage node hosts.

To alleviate this problem, large tables can be partitioned across multiple node groups. Partitioning may be user-specified or may be done by hashing provided by MySQL. Furthermore, though MySQL Clusters support multiple storage nodes on a single host, it is usual to assign each storage node its own host computer because of the memory limitation problem

A node group typically contains two storage nodes. This provides standard mirroring of storage devices. However, node groups can contain one to four storage nodes. If there is only one storage node in a node group, that node group is not fault-tolerant. More than two storage nodes in a node group provide extreme availabilities for that node group (uptimes measured perhaps in centuries). Alternatively, more than two storage nodes in a node group removes the maintenance pressure to immediately repair a failed storage node and still achieve high availability.

The storage nodes within a node group are kept in synchronism via synchronous replication. All updates to a transaction are sent to all nodes during the execution of the transaction. When the transaction is ready to be committed, a standard two-phase commit protocol is used. First, all nodes are asked if they are ready to commit. If they all agree, the transaction is committed across all storage nodes. Otherwise, it is aborted.

Since all updates are synchronously applied to all nodes, each node will present the same view of the database at any given time. Furthermore, the database is always in a consistent state. Since updates are either made to all nodes or not made at all, there is no data loss following a node failure (the recovery point objective, or RPO, is zero).

A node group is a form of an active/active local network. Any user (which we will see is a MySQL server node) can access data via any of the nodes in the application network (the storage nodes in this case). Should one node in the application network go down, there are others to immediately take its place. Failover is fast and is measured in subseconds.

As with any synchronous active/active network, there exists a chance for deadlocks. It is possible that two different MySQL servers will lock the same data item in two different nodes at the same time, and neither can then complete the commit of their transaction across the node group network. To avoid this, one storage node in a node group is designated to be the primary partition for each table or table partition. All updates are routed to that node. Should the storage node hosting the primary partition for a table fail, MySQL Cluster automatically designates another storage node to fulfill that role.

Server Nodes

As we have seen, the "users" of the storage engine are the MySQL server nodes. They provide a SQL interface to the storage engine for the benefit of the applications.

Each server node is connected to each node group. It therefore has access to the data held by each node group even in the presence of storage node failures so long as there is at least one functioning storage node in a node group.

A server node/node group connection is implemented as a direct connection to a specific storage node in the node group. Should that storage node fail, the server node automatically reconnects to another storage node in the node group.

There is typically a number of MySQL server nodes configured in a MySQL Cluster. The number of server nodes is dependent upon the anticipated load and the availability requirements. If load balancing is an issue, there can be multiple server nodes contained on one host since a server node will be paused while waiting for responses from the storage engine. However, if availability is the issue, then server nodes must be on separate hosts. Resulting configurations are typically a mix satisfying these issues.

The MySQL servers provide an SQL interface to the applications. An application can connect to any server node. However, for load balancing purposes, applications are typically assigned to server nodes for the duration of a transaction on a round-robin basis.

Management Nodes

A management node (an MGM node in Cluster-speak) is needed only for startup configuration and for any subsequent reconfiguration. Otherwise, the Cluster can run properly without an operational management node.

When the system is started, all nodes communicate with the MGM to obtain their respective configuration information. Thereafter, should a node fail and be restarted (either automatically or after repair), that node will obtain its configuration information from the MGM.

Because of its central role in failure recovery, not only should an MGM always be functional (you never know when you will need it), but there should be more than one MGM running on different hosts. In this case, one MGM is designated as the active management node. Should its host fail, MySQL Cluster will designate a surviving MGM to be the active management node.

Via the MGM, nodes and hosts can be added and removed with no Cluster downtime.

Node Failure and Recovery

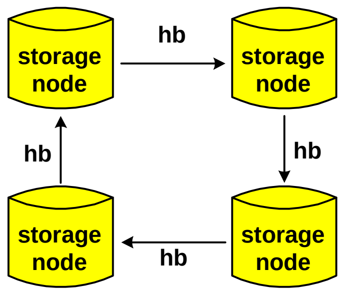
MySQL Cluster provides automatic fault detection and recovery for any of its nodes.

Storage Node Failure

The storage nodes within a node group monitor each other to determine node-group health. This is done by confirming connections and by heartbeats. If a storage node loses its connection to another storage node in its node group, it assumes that the node is down.

However, there are storage node failures that do not result in connection loss, such as an overloaded processor, a disk problem or a memory problem. To detect these conditions, heartbeats are sent among the storage nodes in a circular fashion. Each storage node sends a heartbeat to the next storage node in the ring. It the sending storage node does not get a response in three attempts, it declares that node down.

When a storage node detects a node failure, it notifies the other storage nodes in the node group. Together they verify that the node is, in fact, down. It is possible that a communication fault has not taken down a node but has isolated that node (or nodes) from the other nodes in the node group.



Heartbeats

In this case, to avoid split-brain operation, in which two sets of nodes in the node group are independently processing transactions, a network partitioning protocol is invoked to determine which segmented node group is the largest. The nodes in the other segment are stopped. They will automatically restart and attempt to join the node group.

If the two node-group segments are of equal size, an arbitrator decides which node segment to stop. The arbitrator can be configured on any server node or any management node and may be backed up by another arbitrator.

Should a storage node be taken out of service, it is automatically restarted. If this is successful, the node will get its configuration from the MGM and will then request that its memory-resident database be reloaded from one of the other storage nodes. Since this is a memory-to-memory transfer, it is very fast. Reconfiguration following a storage node failure typically takes less than one second.

Server Node Failure

Should a server node fail, it is simply removed from the MySQL Server pool. No further requests are routed to it.

It is restarted automatically, if possible. If this is successful, it rejoins the server pool.

Management Node Failure

If a management node fails, there is no immediate problem since it is not needed unless another node should fail while it is down. It is restarted, if possible. If there is a backup MGM configured, the backup is promoted to be the active management node.

System Failure and Recovery

Should all storage nodes in a node group fail, the Cluster is down. Since the database is memory-resident, it has been lost and must be recovered.

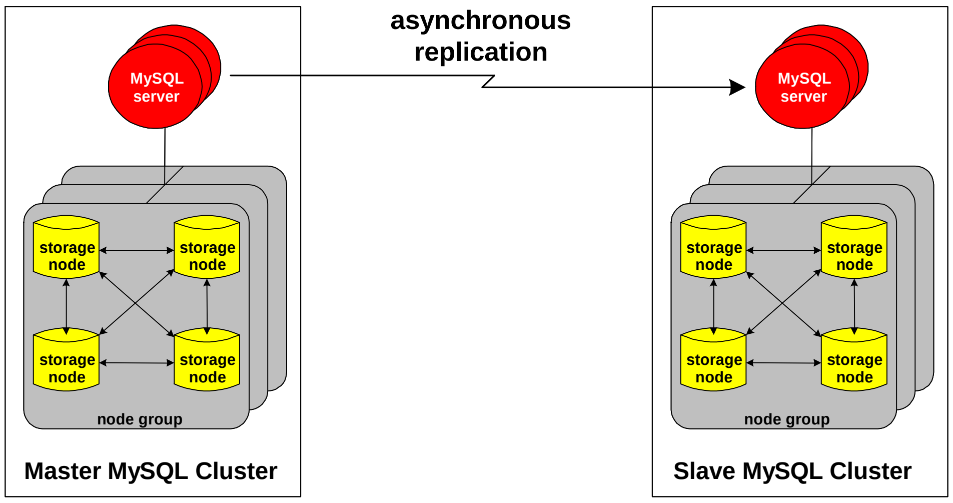
To support recovery, a log file of all database update activity is maintained in the memory of each storage node. This log file is periodically flushed to disk.

If no other action were taken, the log file could grow arbitrarily large. To control this, an asynchronous snapshot is taken of the memory-resident database by replaying the log file against the previous copy of this snapshot. The snapshot is brought up to the latest transaction-consistent state, and the superfluous log file tail is truncated. Thus, the disk snapshot represents a consistent snapshot of the memory-resident database at the time of the snapshot.

Should the node group fail, upon restart the database in each node can be reconstructed from the disk snapshot and the log file. If the memory-resident log file is still available, then no transactions are lost. Otherwise, the database can be reconstructed up to the last complete transaction contained in the log file. Transactions processed since the log file was last written to disk are lost.

Disaster Tolerance

Though the MySQL Cluster uses an active/active architecture to attain a very high availability, protection against a node failure is strictly local. If a disaster of some sort should take out the Cluster site, there is no backup. This situation can be corrected by building at geographically dispersed sites two or more Clusters synchronized by data replication so that if one site goes down, there is a backup site to use.



Disaster Tolerance

MySQL provides an asynchronous replication facility to keep two or more Clusters synchronized. This replication guarantees transaction consistency at the target database. It can replicate SQL statements or changed rows.

However, this replication engine is limited in that it cannot detect or resolve data collisions. Therefore, if bidirectional replication should be used, two users making changes to the same data item at two different clusters within the replication latency time of the replication engine will find that their change will overwrite that made in the remote node. In turn, the remote node change will overwrite their change.

Therefore, multiple Clusters interconnected by MySQL's replication engine are typically configured to operate in a master/slave configuration. One Cluster is designated the master, and all other Clusters are slaves. All updates are made only to the master and are replicated to the slaves. Should a master fail, a slave can be promoted to be the new master.

Therefore, the slaves can only be used as hot backups or for queries. Having multiple slave query nodes can be advantageous for load balancing, to provide data locality to remote communities of users, and for testing.

It is possible to configure bidirectional replication in which each Cluster is both a master and a slave, but the application shoulders the entire responsibility for data collision detection and resolution. However, there are some applications, such as insert-only applications, that do not suffer from data collisions and can benefit from this configuration. Also, it may be possible to partition the database across Clusters, with each Cluster acting as the master for one set of partitions and as a slave backup for other partitions.

In addition to data collisions, another problem with MySQL replication is that it is relatively slow. Each slave must query its master periodically for changes, read those changes, and apply them to its database. As a result, the target database could be several seconds behind the master database. Should the master fail, all of the transactions that are still in this multisecond replication pipeline will be lost.

Licensing

MySQL Cluster is available from MySQL AB under dual licensing. It can be licensed as open source using the GPL (GNU General Public License). Alternatively, it can be obtained under a traditional license for inclusion in proprietary products.

Summary

MySQL Clusters provide a highly available database (five 9s is claimed) using a localized active/active configuration. Failure detection and recovery is automatic and generally will be completed in subsecond time frames.

It is a memory-resident database. Therefore, it can be very fast, with response times measured in milliseconds. The memory-resident database is supported by asynchronous disk-resident checkpoints for system recovery.

However, because the database is memory-resident, its size is limited. The database can be distributed among several node groups, each being a configuration of one to four storage nodes. Up to 48 storage nodes can be included in one Cluster. In addition, there is a limit of 1,600 metadata objects (tables, indices, etc.).

Disaster tolerance can be achieved by connecting two or more geographically dispersed nodes in an active/active configuration with MySQL's asynchronous data replication engine. However, this replication engine can generally be used only in a master/slave configuration, relegating the slave nodes to hot standby or query purposes.

An excellent reference for further study is the MySQL 5.1 Reference Manual (http://dev.mysql.com/doc/)

MySQL Cluster is available from MySQL AB either under a GPL open source license or under a traditional license for proprietary use.