

MySQL Cluster

An Introduction

Geert Vanderkelen

O'Reilly MySQL Conference & Expo 2010
Apr. 13 2010

In this presentation we'll introduce you to MySQL Cluster. We'll go through the MySQL server, the storage engine and show how your data is getting stored in MySQL Cluster.

Retro MySQL

MySQL comes with a rather unique feature: storage engines. It allows you to specify how data is stored and retrieved per tables. Storage engines like MyISAM and InnoDB are writing and reading from local hard drives. We have also an engine called BLACKHOLE which doesn't writing anything at all.

Unless you use some sort of redundant system like RAID10, the health of your data depends on how good the disks are. A point of failure in this case is the hard drive or the disk controller.

```
CREATE TABLE attendees (  
  id INT NOT NULL,  
  name VARCHAR(100),  
  PRIMARY KEY (id)  
) ENGINE=InnoDB
```

There is another storage engine which is not shipped with the normal MySQL server, but is part of the MySQL Cluster product.

The NDBCluster engine

NDB stands for Network DataBase. The NDBCluster engine, as the name then suggest, is using a network connection to store and fetch data. Unlike the classic engines, it doesn't store any data on the

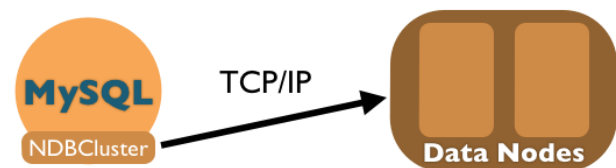
disk where the MySQL server is running.

```
CREATE TABLE attendees (  
  id INT NOT NULL,  
  name VARCHAR(100),  
  PRIMARY KEY (id)  
) ENGINE=NDBCluster
```

NDBCluster is a transaction-safe engine and is ACID compliant.

When you use the NDBCluster storage engine to create your tables, you tell the MySQL server that the data has to be stored in your network, on other machines. These machines are running processes called Data Nodes.

With MySQL Cluster, you can have lots of MySQL Servers connecting to the same Data Nodes. This means that all these MySQL instances can read and update the same data.



MySQL Storage Engine stores data into your network on Data Nodes.

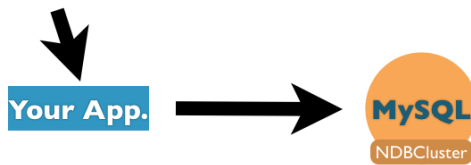
Data Nodes

The machines where the data gets stored are running processes called Data Nodes. They are responsible to make sure that your data is safely stowed away and can be retrieved at all time.

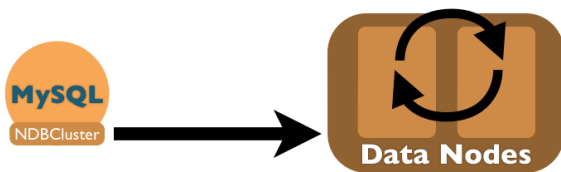
To illustrate what happens to your data, let's look how a new record into the attendee table, which was created using the NDBCluster storage engine.

1. Your application connects to the MySQL Server
2. It issues an SQL statement: `INSERT INTO attendees (name) VALUES ('John Doe')`

id	name
5	John Doe



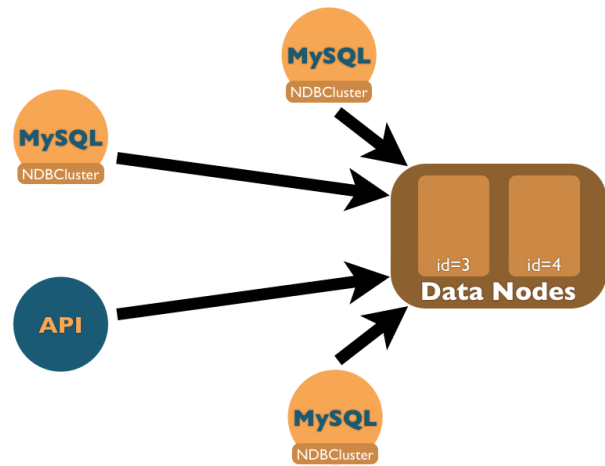
3. The NDBCluster engine kicks in, and starts to communicate with a data node for storing this new record. This particular data node is the transaction coordinator.
4. The coordinator will make sure that the data is stored and will tell the MySQL server when it's done.



5. Data Nodes have successfully stored the data, the MySQL server gives the application an OK that all is well.



We only showed 1 MySQL Server, but you can have multiple MySQL Server inserting, updating, or reading data in parallel.



Lots of MySQL Server can access the data, stored in Data Nodes

Partitioning & Replicas

Data Nodes keep your data highly available by splitting up your data into partitions and making sure there are two copies of each record. You need at least 2 of them to make this work.

Inside the Data Nodes, the table 'attendees' gets partitioned automatically based on the primary key. When you have 2 Data Nodes, your table is split in 2 parts. Each Data Node will be responsible for one of these partitions.

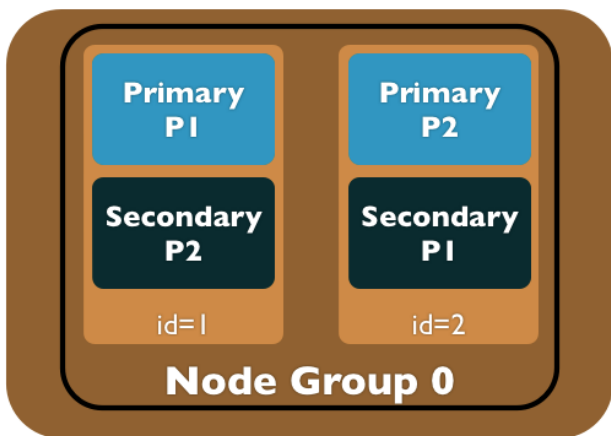
id	name	
1	John	Partition 1
2	Geert	
3	Jan	
4	Michel	
5	Marta	Partition 2
6	Ann	
7	René	
8	Monique	

2 Data Nodes and 2 Replicas, tables are partitioned in 2

MySQL Cluster is all about High Availability and replicas play an essential role. When configured with

NoOfReplicas=2 you data will be stored 2 times.

Lets look again at the 'attendees' table. You have 2 data nodes, each responsible for a partition. With the option NoOfReplicas set to 2, you ask Cluster to keep a copy of another partition. Each Data Node, in addition to be responsible for its part of the table, will also have a copy of the other Data Node. They will have a Primary and a Secondary partition.



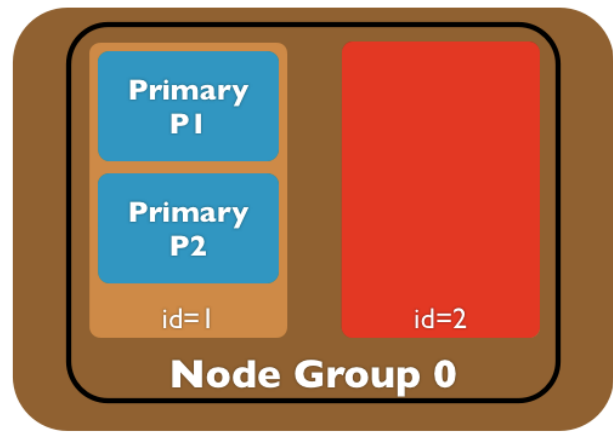
Each Data Node is responsible for 1 partition, the partner keeps a copy as secondary

At this point, in a 2 Data Node setup, both have the exact same data and are grouped as a Node Group. Lets look how this helps us.

High Availability of Your Data

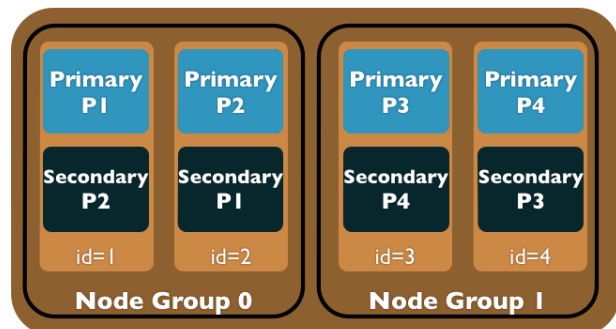
We have now 2 Data Nodes, both holding the exact same data. Each of them responsible only for their own Primary partition.

What happens when a Data Node process dies? When this happens, the Data Node that remains will take responsibility of the partition that was on the other Data Node. This way all data within MySQL Cluster is still available.



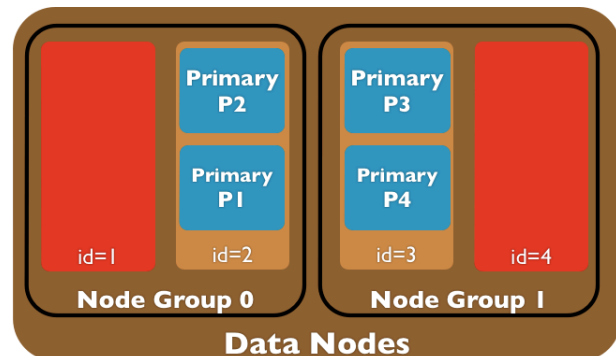
Data Node 2 goes down, Data Node 1 takes responsibility for Partition 2

Lets take it further and look at a setup which is using 4 Data Nodes. The table 'attendees' will be split up in 4 parts. Each data node will have its partition. Since we have asked to keep 2 replicas, MySQL Cluster splits up the 4 nodes in 2 Node Groups. Each Data Node within a Node Group will have the same data.



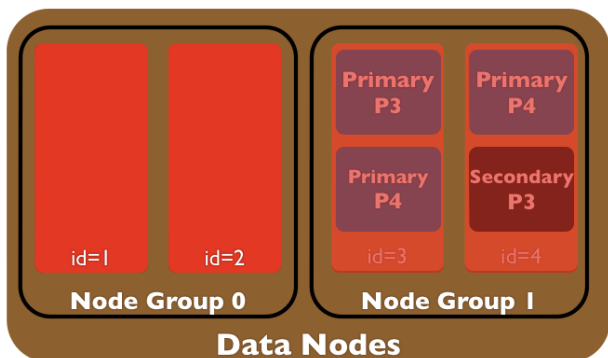
4 Data Nodes, you get 4 Partitions, grouped in 2

Just like with 2 Data Nodes, in a Node Group, if one Data Node goes down, the partner will take responsibility of the other partition and keep your data available.



All data is available with 2 Data Nodes down; 1 Data Node per group needs to be up

At least 1 Data Node per Node Group should be up. If all Nodes within a Node Group are down, Cluster will shutdown. You can't continue with half of your data not available.



All Data Nodes from Group 0 are down: Cluster fails, it can't work further

When a data is started again, it automatically re-sync with its partner. Depending how long it was out and how many changes were done, it will take a full dump of its partner or just what changed. So, it might take a while to get back online.

It's good practice to keep a machine ready to replace a failed Data Node. You can simply put in the new machine (with same IP address) and run the Data Node process. It will join the cluster get data from the other Data Node in the group.

Node Types

We know already Data Nodes, and we know that MySQL Servers. Lets have a look at all the types of nodes inside MySQL Cluster.

Note that nodes are processes, they are not machines. You can have multiple nodes on the same host.

SQL Nodes

These are the MySQL Servers. They use the NDBCluster storage engine to communicate with the Data Nodes. It's important to know that the SQL Nodes are connect with all Data Nodes.

Data Nodes

Processes running on hosts inside your network to which SQL Nodes connect. Data Nodes are responsible for storing records, meta data, index information, do transaction coordination, and lots more. All Data Nodes are sometimes referred too as the Kernel of MySQL Cluster.

Data Nodes store there data in-memory. Important to note, however, is that they are checkpointing what they have in RAM, to disk.

You can have up to 48 Data Nodes within a MySQL CLuster.

Management Nodes

Every other node in MySQL Cluster needs to first connect with the Management Node to fetch the configuration. You can run MySQL Cluster without them, but when not available, no other nodes can start. Management Nodes are also useful for monitoring and doing some administrative tasks.

You want to have 2 Management Nodes: 1 is fine, 3 is to much, 2 is good.

Management Nodes also act as Arbitrator, which prevents a Split Brain situation. Imagine you have 2 Data Nodes: each in a separate rack. Suddenly the network between them goes out and they can't see each other anymore. SQL Nodes can still reach both. The problem here is that data might be inserted with same primary keys and both data nodes would get inconsistent. This is called a Split Brain. To prevent this, MySQL Cluster elects an Arbitrator and usually this is a Management Node. The Data Nodes that can reach the Arbitrator will stay up, the ones that can't exit with an Arbitration Error.

API Nodes

MySQL Cluster comes with NDB API, a C++ API allow you to develop applications which connect directly with Data Nodes, thus bypassing the need of SQL. The

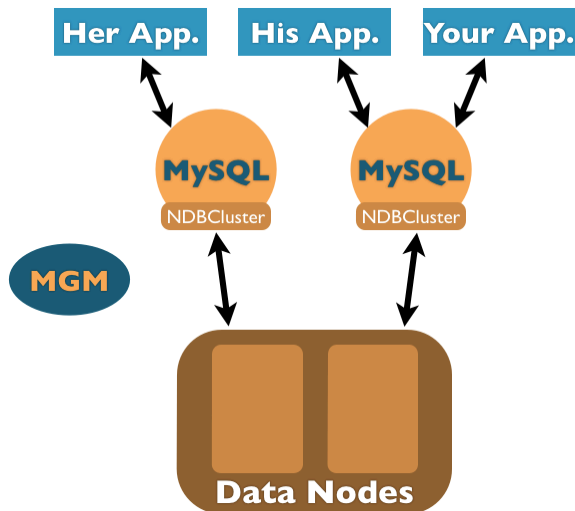
NDBCluster engine is such NDB API application. API and SQL Nodes are sometimes referred as one and the same.

Failure Handling

MySQL Cluster is using a shared-nothing approach. This means that nodes are independent from each other. For example, a Data Node has dedicated hardware: it's own hard drive, CPUs, etc..

Lets look at how MySQL Cluster handles failures of nodes. We assume that each node is running on its dedicated hardware.

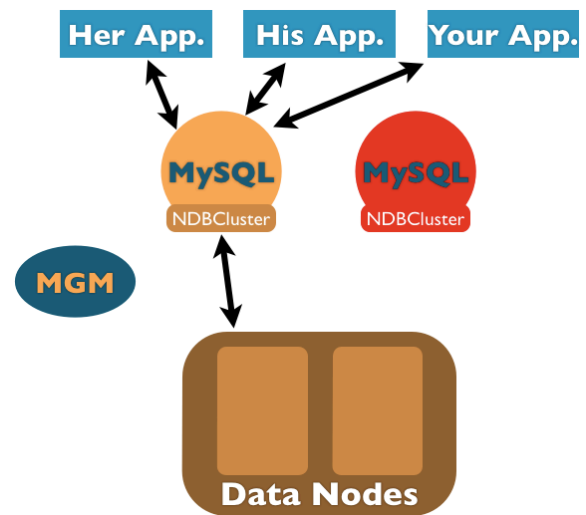
Situation: You have application connecting to MySQL Servers or SQL Nodes, which are in turn connecting to 2 Data Nodes.



Shared-Nothing: nodes don't share disks, memory, cpu, ..

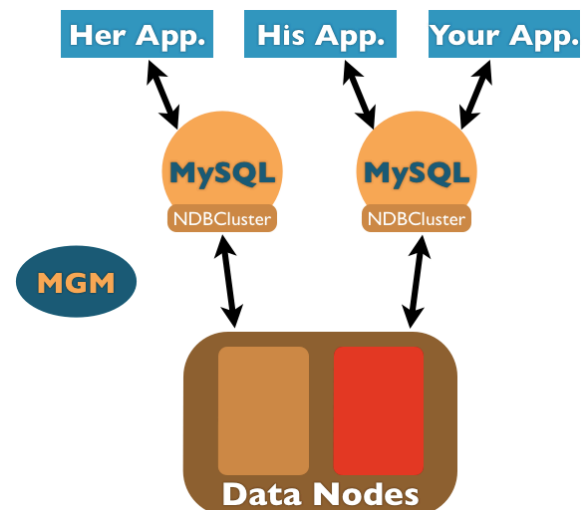
When an SQL Nodes fails, no other node is affected within MySQL Cluster. Your applications will need to figure out a way to connect to another MySQL Server. For example, in JDBC you could give different MySQL Servers to connect too. It's up to the developers of the applications to implement this.

Also, when a transaction was going on, it has to retry (using another MySQL Server).



Failing SQL Nodes doesn't affect other nodes; your applications need to handle it.

If a Data Node fails, not much happens either. The other Data Node will take responsibility for the other part of the data, and all data will be available. You'll need to act and make sure the Data Node is back soon. The reason is that you have now a Cluster with a single point of failure, namely the remaining Data Node. The applications doing transactions will have them rolled back. This a temporary error and they can retry them.



Data Node failure: all stays up. Transactions are rolled backed, apps can retry.

When the Management Node fails, not much happens. Monitoring will stop however, so you might want to get it backup sooner than later. Also, other nodes can't start because the configuration is not available. This can be solved having

2 Management Nodes available and the other nodes can be configured to use both in their connection string.

Limitations and Considerations

MySQL Cluster offers great benefits when it comes to high availability and performance. There are, however, a few limitations and considerations. For example, the maximum row size of 8052 bytes, the 128 column limit per table and performance problems when using BLOB/TEXT fields.

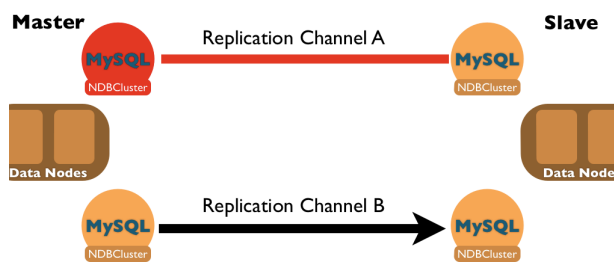
Since MySQL Cluster is an in-memory database, you will need lots of RAM. You could use disk-based tables, but they might have some performance drawbacks.

It's good to develop for MySQL Cluster instead of taking an existing application and try to make it fit.

Some Interesting Features

Cluster comes with Online Backup meaning there is no interruption when you take backups.

There is also support for Geographical Replication which allows you to duplicate your MySQL Cluster to another site, or to an other storage engine like InnoDB for example. It uses the normal MySQL Replication, but needs Row-Based binary logging. The reason is that API Nodes which are not using SQL, have to be logged as well. This would not work with Statement-Based binary logging. With Geographical MySQL Cluster replication you would have 2 MySQL servers doing binary logging. All changes done against NDBCluster tables will go into 2 binary logs (on different machines). The Slave MySQL Cluster can read from both, but applying only from only one. Having 2 replication channels, it makes it highly available. If one channel fails, you can quite easily continue replicating using the other channel.



Making Replication highly available between Clusters using different Channels.

Links

The manual:

<http://dev.mysql.com/doc/refman/5.1/en>

Download:

<http://dev.mysql.com/downloads/cluster/>

About

The author

Geert Vanderkelen is a member of the MySQL Support Team at Sun Microsystems. He is based in Germany and has worked for MySQL AB since April, 2005. Before joining MySQL he worked as developer, DBA and SysAdmin for various companies in Belgium and Germany. Today Geert specializes in MySQL Cluster and works together with colleagues around the world to ensure continued support for both customers and community. He is also the maintainer of Sun's MySQL Connector/Python.

Contact

Geert Vanderkelen

<geert.vanderkelen@sun.com>