MySQL Cluster Evaluation Guide

Getting the most out of a MySQL Cluster evaluation

A MySQL® Technical White Paper by Oracle

April 2010

Table of Contents

1 Introduction

The purpose of this guide is to present several key points to consider during the planning and execution of an evaluation of MySQL Cluster Carrier Grade Edition. This will enable you to efficiently evaluate MySQL Cluster CGE and determine if it is the right choice for your new project, or database migration for an existing application.

2 What is MySQL Cluster?

MySQL Cluster is a relational database technology which enables the clustering of in-memory and disk-based tables with shared-nothing storage. The shared-nothing architecture is a distributed computing architecture where each node is independent and self-sufficient, and there is no single point of contention across the system. This shared-nothing architecture allows the system to work with commodity hardware and software components, such as the standards based AdvancedTCA platform running the Windows, Linux or Solaris operating systems.

MySQL Cluster integrates the standard MySQL server with a clustered storage engine called NDB. The data within a MySQL Cluster can therefore be accessed via various MySQL connectors like PHP, Java or .NET as well as standard MySQL. Data can also be accessed and manipulated directly using MySQL Cluster's native NDB API. This C++ interface provides fast, low-level connectivity to data stored in a MySQL Cluster. Java applications can get easy and efficient access to the data by using `MySQL Cluster Connector for Java which provides` two object-relational mapping persistence solutions – ClusterJ or ClusterJPA which is a plugin for OpenJPA – both frameworks go directly to the NDB API rather than using JDBC and the MySQL Server. Both ClusterJ and JPA directly map Java objects to relational table sin the MySQL Cluster database. Data can also be accessed using LDAP via a directory server front end which again uses the NDB API directly.

2.1 MySQL Cluster Architecture

Architecturally, MySQL Cluster consists of three different types of nodes, each providing a specialized role to deliver 99.999% availability with real time performance and linear scalability.

Figure 1 shows a simplified architecture diagram of a MySQL Cluster consisting of four Data Nodes.

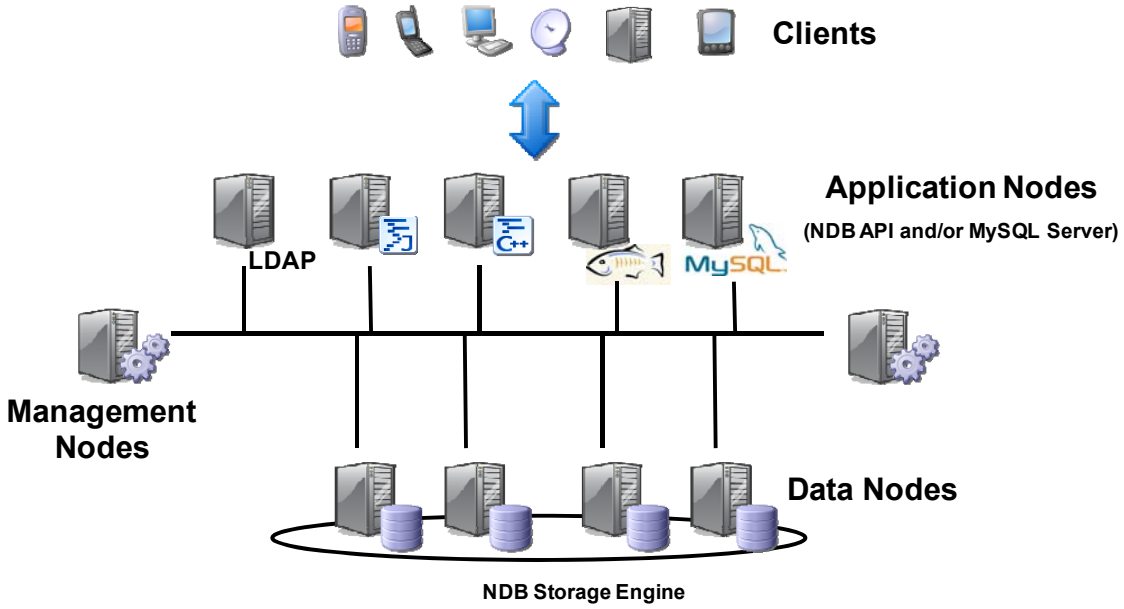


Figure 1 Four Data Node MySQL Cluster

Data Nodes are the main nodes of a MySQL Cluster. They provide the following functionality:

- Storage and management of both in-memory and disk-based data
- Automatic and user defined partitioning of data
- Synchronous replication of data between data nodes
- Transactions and data retrieval
- Fail over
- Resynchronization after failure

By storing and distributing data in a shared-nothing architecture, i.e. without the use of a shared-disk, and synchronously replicating data to at least one replica, if a Data Node happens to fail, there will always be another Data Node storing the same information. This allows for requests and transactions to continue to be satisfied without interruption. Any transactions which are aborted during the short (sub-second) failover window following a Data node failure are rolled back and can be re-run.

As of MySQL Cluster version 5.1, it is possible to choose how to store data; either all in memory or some on disk.. In-memory storage can be especially useful for data that is frequently changing (the active working set). Data stored in-memory is routinely check pointed to disk locally and coordinated across all Data Nodes so that the MySQL Cluster can be recovered in case of a system failure. Disk-based data can be used to store data with less strict performance requirements, where the data set is bigger than the available RAM. As with most other database

servers, a page-cache is used to cache frequently used disk-based data in the Data Nodes' memory in order to increase the performance.

Application Nodes are the applications connecting to the database. Applications can access the database using SQL through one or many MySQL Servers performing the function of SQL interfaces into the data stored within a MySQL Cluster. When going through a MySQL Server, any of the standard MySQL connectors can be used, offering a wide range of access technologies.

Alternatively, a high performance (C++ based) interface called NDB API can be used for extra control, better real-time behavior and greater throughput.

When using the NDB API, there are some additional low level optimizations for high performance, real-time applications and there is a good description of some of these in the "Benchmarking Highly Scalable MySQL Cluster white paper from *http://www.mysql.com/why-mysql/benchmarks/CGE-Intel-Dolphin.html*

A common approach is to access the data for real time applications using the NDB API, and perform operations and maintenance tasks using the SQL interface, where real time performance is not critical.

In addition, there are a growing number of alternate access methods that exploit the performance of the NDB API without the overhead of using SQL and the MySQL Server nodes; these include LDAP, http and MySQL Cluster Connector for Java.

Data Nodes do not require any specific Application Nodes to be available and running in order to service requests from other Application Nodes. This means there is no interdependence between Application Nodes and Data Nodes. In this way, by minimizing the interdependency of nodes, the MySQL Cluster is able to minimize any single points of failure.

Management Nodes manage and make available to other nodes cluster configuration information. The Management Nodes are used at startup, when a node wants to join the cluster, and when there is a system reconfiguration. Management Nodes can be stopped and restarted without affecting the ongoing execution of the Data and Application Nodes. By default, the Management Node also provides arbitration services, in the event there is a network failure which leads to a "split-brain" or a cluster exhibiting "network-partitioning".

3 The Benefits of MySQL Cluster

The shared-nothing architecture employed by MySQL Cluster offers several key advantages:

3.1 Scalability

MySQL Cluster offers scalability on five different levels:

- If more storage or capacity is needed, Data Nodes can be dynamically added incrementally without interruption to service
- Application Nodes can be dynamically added to increase performance and parallelization
- Clients connecting to Application Nodes can also be dynamically added online
- Additional CPUs, cores or threads in the data nodes can be exploited using the multi-threaded NDB process (ndbmtd)
- The database can be replicated to other database hosts (for example, one using the MyISAM storage engine) for read-access or generating complex reports

3.2 Performance

MySQL Cluster's architecture, which offers scalability on five tiers, can deliver unprecedented performance when used in conjunction with:

- NDB API, MySQL Cluster Connector for Java or OpenLDAP
- Primary key lookups (for both reads and writes)
- Distribution-aware application design
- User-defined partitioning
- Parallelization
- Transaction batching
- High performance network interconnects (SCI)

This paper provides an introduction to some techniques to get the best performance out of a MySQL Cluster database but this is examined in more detail in the “Guide to Optimizing Performance of the MySQL Cluster Database” from http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

3.3 High-Availability

Data Nodes can fail, and resynchronize automatically, without affecting service or forcing the Application Nodes to reconnect. Moreover, it is also possible to have redundant Management Servers and Application Nodes to maximize service availability. It is also possible to replicate asynchronously between MySQL Clusters to allow for geographic redundancy.

4 Key features of MySQL Cluster CGE 7.0 and 7.1

MySQL Cluster 7.1 builds on the innovations delivered by MySQL Cluster 7.0 which introduced several new features that lend themselves to building a high performance, scalable and highly available system. These include:

- Multi-Threaded Data Node: A single Data Node can effectively use up to 8 CPUs/cores/threads
- On-Line Add Node: Scale performance and capacity by adding additional node groups to the cluster with no loss of service
- Multi-Threaded Disk Data File Access: Increased performance for disk-based table data
- Improved Large Record Handling: Increased performance

MySQL Cluster CGE 7.1 adds the following new features:

- NDBINFO presents real-time status and usage statistics from the MySQL Cluster Data Nodes as SQL tables and views, providing developers and administrators a simple and consistent means of pro-actively monitoring and optimizing database performance and availability.MySQL Cluster Manager: a new management interface to make it simpler, faster and less error-prone to configure, run and manage a MySQL Cluster database. This is described in more detail at http://www.mysql.com/cluster/mcm
- MySQL Cluster Connector for Java: In addition to JDBC (via one or MySQL Servers) data can now be accessed more simply and faster using either JPA (using the ClusterJPA plugin) or through MySQL Cluster's own object-relational mapping persistence layer (ClusterJ). This is described in more detail in the whitepaper posted as follows: http://www.mysql.com/why-mysql/whitepapers/mysql_wp_cluster_connector_for_java.php

For more information about all of these 7.0 & 7.1 features, please refer to the "MySQL Cluster 7.1: Architecture and New Features" white paper which is available from http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster7_architecture.php

5 Points to Consider Before Starting an Evaluation

5.1 Will MySQL Cluster "out of the box" run an application faster than another database?

The reality is it probably will not without some modifications to the application and the database. However, by following the guidelines and best practices in this paper and in the "Guide to Optimizing Performance of the MySQL Cluster Database" from http://www.mysql.com/why-mysql/white-

papers/mysql_wp_cluster_perfomance.php, significant performance gains over other databases can be realized.

5.2 Is there an easier, more automated way to get everything installed?

Oracle provides RPM as well non RPM packages for installing Cluster binaries as well as the option to download and compile the source code. The MySQL Reference Guide (http://dev.mysql.com/doc/refman/5.1/en/index.html) includes instructions on installing the Cluster and MySQL Server components.

It is possible to script installation and configuration of MySQL Cluster. While Oracle does not publish such scripts, they can be built by the end user, or are available from 3rd parties. A 3rd party (and free) tool is available at http://www.severalnines.com/config/ which requests a few details on the required configuration and then produces a customized script that will download (and if required, compile), install and configure a complete Cluster. It can also configure the Cluster to act as a master or slave for geographic replication. These scripts can enable a Cluster to be installed and up and running within a few minutes. Note that this tool is not supported by Oracle.

5.3 Does the database need to run on a particular operating system?

All the core components of the MySQL Cluster, the MySQL Servers, Data Nodes, and Management Server/Client must all run on a supported Linux or UNIX operating system. MySQL Cluster 7.0 introduces the ability to run these on Windows for development systems (not supported for deployed systems).

All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a SPARC host which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running mysql or other clients that may be accessing the cluster's SQL nodes.

For the latest list of supported platforms, please see:

http://www.mysql.com/support/supportedplatforms/cluster.html

MySQL Clients (using JDBC, .NET, PHP, LDAP, etc) can run on any operating system and access a MySQL Cluster via a MySQL Server running on a supported platform.

5.4 Will the entire database fit in memory?

MySQL Cluster supports disk-based as well as in-memory data. However, there are some limitations on disk storage to be aware of with the current implementation. For example, non-indexed data can reside on disk but indexed columns must always be in memory. The larger the database, the more likely you will need more memory/hardware. For additional information concerning the disk data implementation, please see:

http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-disk-data.html

There are few basic tools you can use to help determine how much memory you will need. First we recommend you use the ndb_size.pl script (which ships with the MySQL Server installation) to help you determine how much memory would be required if an existing MySQL database was converted to the NDB storage engine. Please note that this tool assumes converting your existing database into an all in-memory database without leveraging disk-based tables. For more information on this script see:

http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-utilities-ndb-size.html

An alternative tool is available from http://www.severalnines.com/sizer/index.php which relies on the tables having been created in Cluster. Note that this tool is from a 3rd party and is not supported by Oracle.

If designing your database from scratch, some rough calculations can be used to help determine the approximate sizing requirements for the systems:

*Data Size * Replicas * 1.25 = Total Memory*

Example: 2 GB * 2 * 1.25 = 5 GB

*(Data Size * Replicas * 1.25)/Nodes = RAM Per Node*

Example: (2 GB * 2 * 1.25)/4 = 1.25 GB

5.5 Does the application make use of complex JOINs or full table scans?

If so then unless you are willing to rewrite your application, you will likely experience lower performance characteristics compared to other MySQL storage engines. This is due to the manner in which data is partitioned and distributed across many Data Nodes. Applications which perform mostly primary key look-ups will typically benefit the most from MySQL Cluster's distribution of data.

One strategy to consider is using a set of stored procedures to simplify the amount and impact of joins your application may require. This may help minimize the amount of application modifications which may be required.

Future MySQL Cluster releases may improve the performance of these broader operations.

5.6 Does the database require foreign keys?

Foreign keys are currently not supported by the NDB storage engine. If using MySQL server to access your data then this limitation can be worked around by implementing triggers (the constraints would not be enforced for any clients using the NDB API). This method is described in http://forge.mysql.com/wiki/ForeignKeySupport#Appendix_A:_Triggers_implementing_foreign_key_constraints

5.7 Does the application require full-text search?

Full-text search is currently not supported by the NDB storage engine, a common approach is to provide full text search by replicating the clustered database into a read-only MyISAM slave server.

5.8 How will NDB perform compared to other storage engines?

"An INSERT into a MySQL Cluster takes longer then if I just use a MEMORY table, why is this?"

"We have a large INNODB table and when we altered it to MyISAM it took several seconds to migrate. When we altered the table to NDB, it took five times as long. We expect it to be faster."

"We perform similar requests against INNODB and NDB and the INNODB request is always faster, why?"

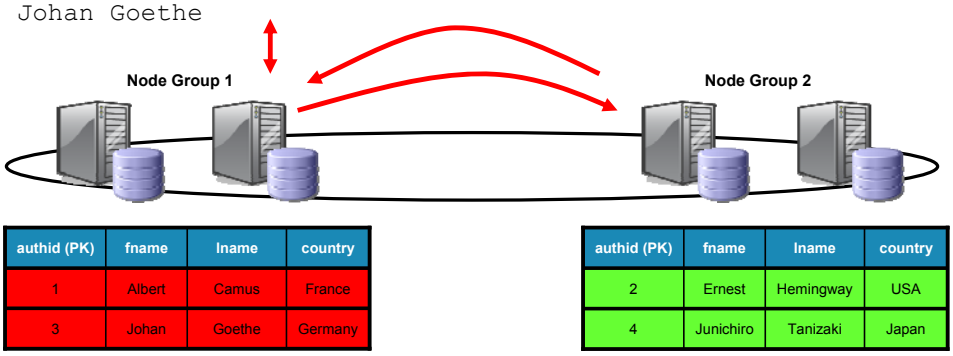
In the above three scenarios what is being forgotten is the fundamental difference in which the NDB storage engine is architected compared to other MySQL storage engines. MySQL Cluster is a distributed, shared-nothing data store that provides very high, scalable performance for applications that largely use primary key access but it incurs a cost in network access when accessing data between a MySQL Server and Data Nodes. Below in Figure 2 you notice that in order to satisfy the query, multiple Data Nodes must participate in the retrieval of data.

Query:

```
SELECT fname, lname
FROM author
WHERE authid BETWEEN 1 AND 3;
```

Result:

```
Albert Camus
Ernest Hemingway
Johan Goethe
```



authid (PK)	fname	lname	country
1	Albert	Camus	France
3	Johan	Goethe	Germany

authid (PK)	fname	lname	country
2	Ernest	Hemingway	USA
4	Junichiro	Tanizaki	Japan

Figure 2 Data access across Data Nodes

Meanwhile in Figure 3, we can see that with traditional storage engines, there is no networking communication overhead to try and satisfy the query, as the entire database/table is located on the same host.

Query:

```
SELECT fname, lname
FROM author
WHERE authid BETWEEN 1 AND 3;
```

Result:

```
Albert Camus
Ernest Hemingway
Johan Goethe
```

authid (PK)	fname	lname	country
1	Albert	Camus	France
2	Ernest	Hemingway	USA
3	Johan	Goethe	Germany
4	Junichiro	Tanizaki	Japan

Figure 3 Data access with a traditional storage engine

6 Setting Expectations and Evaluation Guidelines

Whether or not you are planning on migrating an existing database to MySQL Cluster, or building a new system from scratch you should prepare yourself for some pre-work in order to get the most performance out of your system and therefore benefit from MySQL's carrier-grade performance, scalability and availability. For example, if you are considering migrating another database technology to MySQL Cluster, it is very likely that the data model is optimized for the original database. Hence, you might have to do some modifications to schemas and queries if you want to get the most out of MySQL Cluster.

In this section we present guidelines that will increase the likelihood of making a successful evaluation.

6.1 Hardware

Do you have enough computers and resources available to conduct the evaluation?

MySQL recommends the following hardware specification:

Data Nodes

- Dual core or dual CPU machines as a starting point; from MySQL Cluster 7.0, a single data node can effectively exploit up to 8 cores/CPUs
- 64-bit machines with enough RAM to store your in-memory data set
- Big CPU caches are important for optimal performance
- Having a fast, low-latency disk subsystem is very important and will effect check pointing and backups

All nodes in the Cluster should be run on hosts on the same LAN; Geographical replication between remote Clusters can be used to achieve geographic redundancy.

MySQL Servers/Application Nodes

- Fast CPUs with big caches
- RAM is not as important as with Data Nodes. (4 GB should be more than enough.)

In addition, high-bandwidth, low latency network interconnects will help with the Cluster's performance – use Gigabit Ethernet or SCI connections.

Increased performance can sometimes be achieved by co-locating MySQL nodes and data nodes on same hosts to remove network delays.

Make sure you design your systems to avoid swapping whenever possible. This is very bad for Data Nodes. As a rule of thumb, have 12 times the amount of `DataMemory` configured for disk space for each data node. This space is needed for storing three Local Checkpoints (LCPs) and the Redo log. You will also want to allocate space for backups and of course, table spaces if you are making use of disk-based data. We recommend the following minimal setup when evaluating MySQL Cluster:

- 2 computers each running one Data Node
- 2 computers each running one Application Node and a Management Server

This should give you a minimum amount of redundancy on all processes which constitute a MySQL Cluster.

6.2 Performance metrics

How many transactions per second are required? What is the required response time?

6.3 Test tools

What tools do you have in order to verify the performance criterion?
Are they migrated from a legacy system and optimized for that environment? If so then consider reworking them so that a real 'apples to apples' comparison can be made.

6.4 Programming interfaces

In environments where performance is critical, as in real time systems, we strongly recommend considering the use of the native APIs, such as the NDB API (C++ API) or MySQL Cluster Connector for Java.

There are several advantages to using the NDB API over SQL:

- Lower latency and less overhead – no parsing or optimizing of queries required
- Fine grained control of the interaction between the Application and Data nodes
- Ability to hand optimize performance-critical requests
- Batch interface -possible to batch all types of requests like inserts, updates, reads, and deletes. These batches can be defined on one or many tables.
- By using the NDB API, you can expect at least three times faster compared to SQL, often more depending on the types of requests, and whether the batch interface is used

Refer to http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_connector_for_java.php fpr more information of the benefits of MySQL Cluster Connector for Java.

Many MySQL Cluster customers with real-time requirements make use of these direct APIs when implementing their network requests, and use the SQL interface offered by the MySQL server for provisioning and operation & maintenance tasks.

6.5 Data model and query design

Has the data model been developed specifically for MySQL Cluster or is it a legacy model?
Does the evaluation data set reflect something tangible and realistic?
Do you have queries matching the relevant use cases you want to test?
Are the use cases realistic or edge cases?

Having a good data model and sound queries are crucial for good performance. No evaluation can be successful unless a few fundamental points are understood regarding network latency, data algorithms and searching data structures.

- The data model and queries should be designed to minimize network roundtrips between hosts. It is in the MySQL Nodes where data aggregation takes place and joins are resolved.
- Looking up data in a hash table is a constant time operation. Using the big-O notation it costs O(1). This is a primary key lookup (exact match returning 0 or 1 records) in MySQL Cluster.
- Looking up data in a tree (T-tree, B-tree etc) structure is logarithmic (O (log n)). This is an ordered index lookup (returning 0 to many records) in MySQL Cluster.

For a database designer this means it is very important to choose the right index structure and access method to retrieve data. We strongly recommend application requests with high requirements on performance be designed as primary key lookups. This is because looking up data in a hash structure is faster than from a tree structure and can be satisfied by a single data node. Therefore, it is very important that the data model takes this into account. It also follows that choosing a good primary key definition is extremely important.

If ordered index lookups are required then tables should be partitioned such that only one data node will be scanned.

Let's take an example regarding subscriber databases, which are common in the telecom industry. In subscriber databases it is common to have a subscriber profile distributed across a number of tables. Moreover, it is typical to use a global subscriber id. For example IMSI, SIP, MSISDNs are mapped to such an id in every table related to the subscriber profile that is either:

- The sole primary key in the table participating in the subscriber profile is the global subscriber id (gsid)
- Part of a primary key. E.g. a subscriber might have many services stored in a service table so the primary key could be a composite key consisting of <gsid, serviceid>

For the case when the global subscriber id is the sole primary key in the table, reading the data can be done with a simple primary key request. But if the global subscriber id is part of the primary key then there are two possibilities:

Use an ordered index scan to fetch the data:

SELECT * FROM service WHERE gsid=1202;

Use prior knowledge in the query. Often a subscriber can't have more than a fixed number of services. This is often not liked by puritans but using that information can be a way of optimizing the query and is often faster and scales better than an ordered index scan)

SELECT * FROM service WHERE gsid=1202 AND serviceid IN (1, 2, 3, 4,5,6,7,8,10...64);

If you are using the NDB API you can combine the above with the batch interface in the NDB API and read up a subscriber profile in one network roundtrip (using SQL will render one roundtrip for each table being read).

The distributed nature of the Cluster and the ability to exploit multiple CPUs, cores or threads within nodes means that the maximum performance will be achieved if the application is architected to run many transactions in parallel. Alternatively you should run many instances of the application simultaneously to ensure that the Cluster is always able to work on many transactions in parallel.

By using the above design principles and combining it with partitioning and distribution awareness it is possible to design a very high performing and scalable database.

Designing data models and queries that maximize performance with MySQL Cluster is covered in more detail in the “Guide to Optimizing Performance of the MySQL Cluster Database” from http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

6.6 Using disk data tables or in-memory tables

You can have a bigger dataset than you have RAM, by identifying table data to be stored on disk. Non-indexed attributes can selectively be made to reside on disk

but indexed attributes are always in memory. The larger the database and the more indexes you have on the tables the more likely it is that you will need more memory or hosts. Similar to most disk-based DBMSs, there is a LRU (Least Recently Used) buffer cache that caches hot pages. When reading a record containing disk-based data a lookup is made in the buffer cache to see if the page exists there. If it does not exist there, then the record data has to be read from disk.

At some stage the buffer cache is check pointed and all the dirty pages in the buffer cache are written back to a table space. The table space together with the available RAM for the indexed attributes defines how much data you can store in a disk data table.

This means that the same performance limitations exist for disk data tables in MySQL Cluster as traditional disk-based DBMSs. The bigger the buffer cache the better, as there is a risk of getting disk I/O bound. Tables which are subject to a lot of random access, and have strong requirements on performance and response times are better designed as in-memory tables in order to avoid being disk I/O bound.

6.7 User defined partitioning and distribution awareness

In Figure 4 and Figure 5 we show the difference between an application that is distribution aware and one that is not. The purpose is to help illustrate the potential savings in inter-node communication.

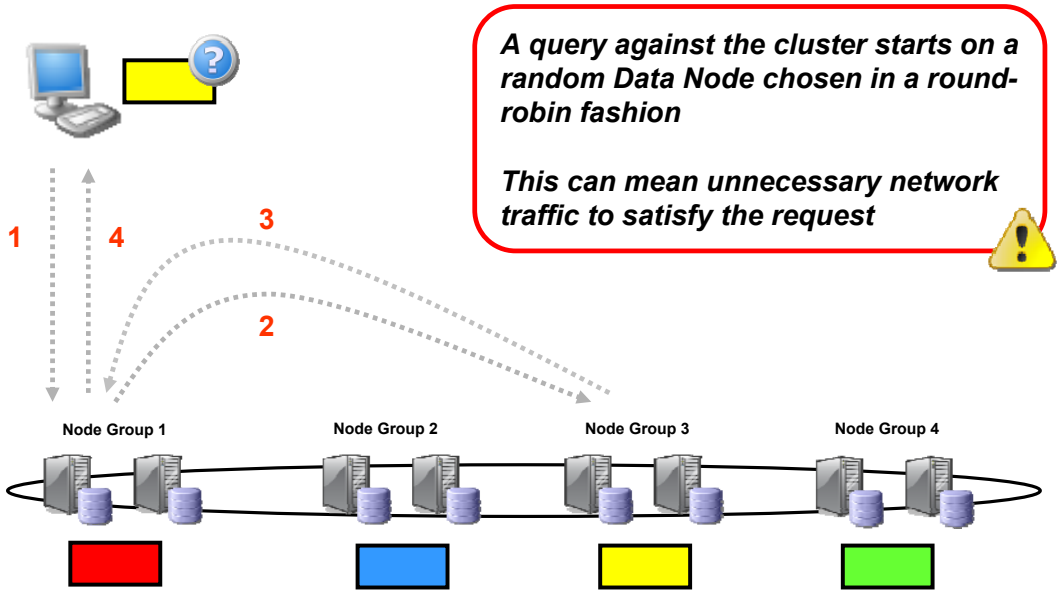


Figure 4 Non-distribution aware application

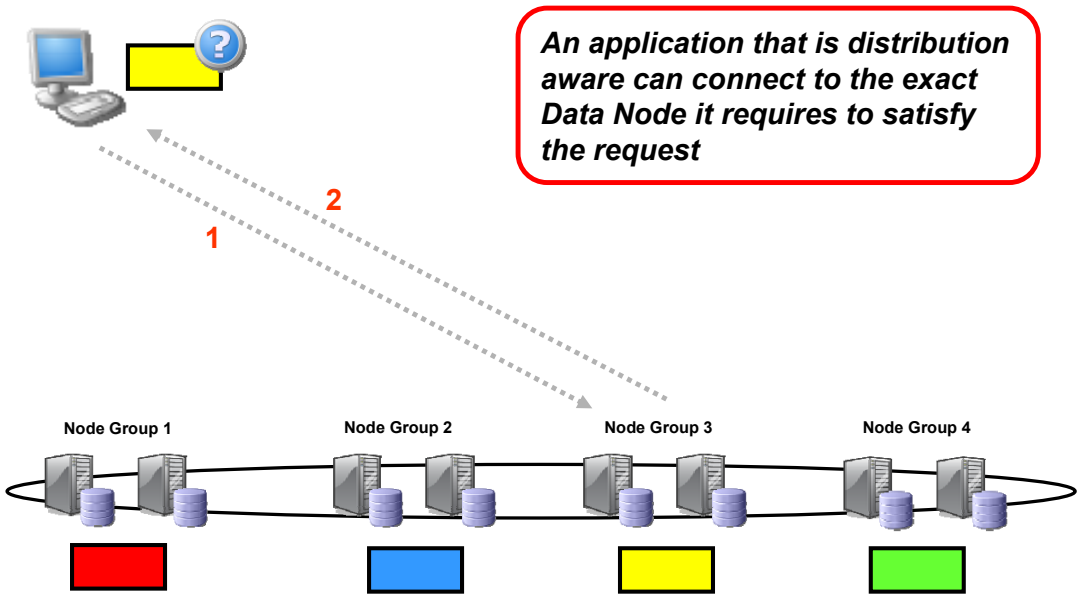


Figure 5 Distribution aware application

Using distribution awareness can give a huge increase in performance as show in Figure 6. For more information on these results see:

http://www.mysql.com/why-mysql/benchmarks/CGE-Intel-Dolphin.html

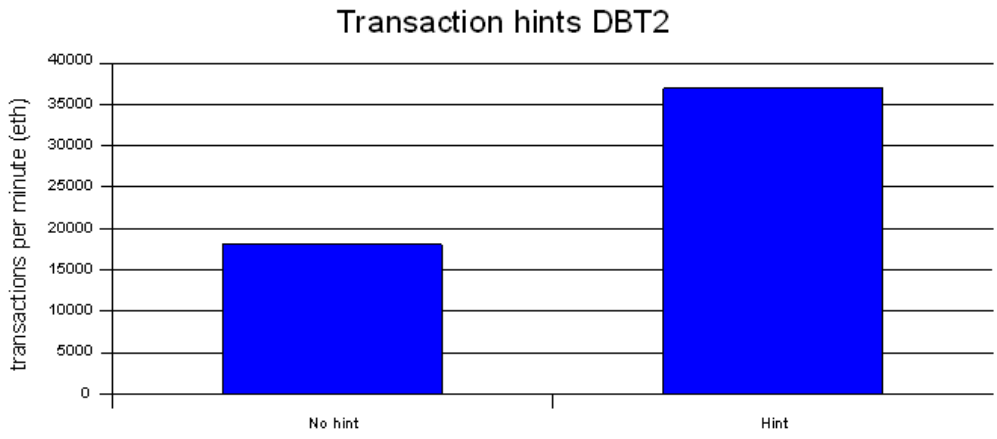


Figure 6 Distribution awareness performance with more than two Data Nodes

For applications accessing the database through an SQL Server, being distribution aware involves using a primary key wherever possible and then constructing any non primary key lookup queries to use the WHERE expression narrow the search

to one partition. For applications using the NDB API, the MySQL Cluster API Developer Guide http://dev.mysql.com/doc/ndbapi/en/index.html describes how to be distribution aware.

Detailed information on making an application distribution aware can be found the "Guide to Optimizing Performance of the MySQL Cluster Database" from http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

The best performance can be achieved by using the native Cluster API (C++ or MySQL Cluster Connector for Java). Using the the NDB API will typically yield performance that is 3 times faster than using SQL. Often the performance improvement can be even greater, depending on the request.

6.8 Parallelizing applications and other tips

As mentioned MySQL Cluster is characterized by a parallel data store. This means that there is often more than one Data Node that can work in parallel to satisfy application requests.

Additionally, MySQL Cluster 7.0 introduces multi-threaded data nodes that can effectively exploit up to 8 threads, CPUs or cores on a single host. To use this new functionality, the data nodes should be started using the new ndbmtd binary rather than ndb and config.ini should be set up correctly.

The key to making your application achieve higher performance characteristics is to make use of this functionality whenever possible. Parallelization can be achieved in MySQL Cluster in a few ways:

- Adding more Application Nodes
- Use of multi-threaded data nodes
- Batching of requests
- Parallelizing work on different Application Nodes connected to the Data Nodes

How many threads and how many applications are needed to drive the desired load has to be studied by benchmarks. One approach of doing this is to connect one Application Node at a time and increment the number of threads. When one Application Node cannot generate any more load, add another one. It is advisable to start studying this on a two Data Node cluster, and then grow the number of Data Nodes to understand how your system is scaling. If you have designed your application queries, and data model according to the best practices presented in this paper, you can expect close to double the throughput on a four Data Node system compared to a two Data Node system, given that the application can generate the load.

Try to multi-thread whenever possible and load balance over more MySQL servers. In MySQL Cluster CGE you have access to additional performance enhancements which allow the MySQL Cluster to make use of the CPUs better on multi-cpu/multi-core systems. These include:

- Reduced lock contention by having multiple connections from one MySQL Server to the Data Nodes (--ndb-cluster-connection-pool=X)
- Setting threads to real-time priority
- Locking Data Node threads (kernel thread and maintenance threads to a CPU)

7 Advice Concerning Configuration Files

Below is template for a good general purpose configuration of MySQL Cluster. Most parameters can initially be left as default, but some important ones to consider include:

config.ini

```
[TCP DEFAULT]
SendBufferMemory=2M
ReceiveBufferMemory=1M


[NDB DEFAULT]
NoOfFragmentLogFiles= 6 * DataMemory (MB) / 64
```

The above is heuristic, but can also be calculated in detail if you know how much data is being written per second

```
MaxNoOfExecutionThreads=8
```

This value is dependent on the number of cores on the data node hosts. For a single core, this parameter is not needed. For 2 cores, set to 3, for 4 cores set to 4 and for 8 or more cores set to 8.

```
RedoBuffer =32M
LockPagesInMainMemory=1
```

my.cnf

```
[mysqld]

ndb-use-exact-count=0
ndb-force-send=1
engine-condition-pushdown=1
```

Note that if using MySQL Cluster Manager, you define the configuration parameters there rather than editing the config.ini file; refer to www.mysql.com/cluster/mcm

8 Sanity Check

Before starting any testing it is a good idea to perform a sanity check on your environment. Ask yourself the following questions:

Can I ping all the machines participating in the MySQL Cluster based on the hostnames/IP addresses specified in the config.ini and my.cnf?

Do I have a firewall that may prevent the Data Nodes to talk to each other on the necessary ports?

Hint: Use e.g. traceroute to check that you don't have unnecessary router hops between the nodes

Do I have enough disk space?

Hint: Roughly 10-12 times the size of `DataMemory` is a good heuristic.

Do I have enough RAM available on my Data Nodes to handle the data set?

Hint: A Data Node will fail to start if it can't allocate enough memory at startup

Do I have the same version of MySQL Cluster everywhere?

Hint: ndb_mgm –e "show" and the cluster log will tell you if there is a mismatch

8.1 A few basic tests to confirm the Cluster is ready for testing

Below are a few basic tests you should perform on your MySQL Cluster to further validate it is fully operational.

Create a basic table

```
CREATE TABLE t1 (a integer, b char(20), primary key (a))
ENGINE=NDB;
```

Insert some data

```
INSERT INTO t1 VALUES (1, 'hello');
INSERT INTO t1 VALUES (2, 'hello');
INSERT INTO t1 VALUES (3, 'hello');
INSERT INTO t1 VALUES (4, 'hello');
```

Select the data out of the table

```
SELECT * FROM t1;
```

Restart a Data Node (ndbd) from the management server (ndb_mgm)[1]

```
3 restart -n
```

Check to see if the node restarted, rejoined the cluster and all nodes are connected[1]

```
show all status
```

Select the data out of the table

```
SELECT * FROM t1;
```

Repeat this process for all other Data Nodes

If you cannot complete the above steps, have problems getting data to return or failures when stopping or starting data nodes, you should investigate the error logs, the network and firewalls for issues.

9 Troubleshooting

If the evaluation exercise is looking to test the boundary condition for MySQL Cluster capacity and performance then it may well be that you hit issues related to your initial Cluster configuration. This section identifies some of the associated errors that may be seen, what they mean and how they can be rectified.

9.1 Table Full (Memory or Disk)

Error

```
ERROR 1114: The table '<table name>' is full
```

Cause

Either:

- All of the memory (RAM) assigned to the database has been exhausted and so the table cannot grow. Note that memory is used to store indexes even if the rest of a table's data is stored on disk

or

- The table holds disk-based data and the capacity of the associated TABLESPACE has been fully used.

[1] Operation should be performed from the MySQL Cluster Manager CLI using MySQL Cluster Manager

Solution

If memory has been exhausted:

If the size of the database cannot be reduced and no additional data can be stored on disk rather than in memory then allocate more memory.

If there is still extra physical memory available on the host then allocate more of it to the database by increasing the `IndexMemory` and `DataMemory` configuration parameters (refer to "MySQL Cluster Reference Guide" for details).

If there is no more physical memory available then either install more on the existing hosts or add a new node group running on new hosts (refer to "MySQL Cluster 7.1: Architecture and New Features" for details).

If the disk space associated with the TABLESPACE has been exhausted:

If there is still spare disk space available on the host then the `ALTER TABLESPACE` SQL command can be used to provision more storage space for the table or tables. Otherwise, extra hardware should be added before increasing the size of the TABLESPACE.

9.2 Space for REDO Logs Exhausted

Error

```
Temporary error: 410: REDO log buffers overloaded, consult online manual (increase RedoBuffer, and|or decrease TimeBetweenLocalCheckpoints, and|or increase NoOfFragmentLogFiles)
```

Cause

The amount of space allocated for the REDO buffer is not sufficient for the current level of database activity (the required space is dictated by update rates rather than the size of the database).

Solution

Configure additional log files by increasing the `NoOfFragmentLogFiles` parameter (defaults to 8). Each additional log file adds 64MB of space to the REDO log capacity. A rolling restart (with `–initial` option) will cause the changes to take effect.

9.3 Deadlock Timeout

Error

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try
restarting transaction
```

Cause

A node has had to wait for a period longer than specified by `TransactionDeadlockDetectionTimeout` (default is 1.2 seconds) and so has indicated an error to the client. The root cause could be one of:

- A failed node (which was holding onto a resource)
- The node is heavily overloaded
- Another operation is holding onto a resource for a long time (application design issue)

Solution

If a node has failed then recover it (replacing hardware if required).

Try increasing `TransactionDeadlockDetectionTimeout` to cope with overload conditions.

Examine the application design and find ways that it holds onto resources for shorter periods (for example, use multiple, shorter transactions where safe to do so).

9.4 Distribution Changes

Error

```
ERROR 1297 (HY000): Got temporary error 1204 'Temporary
failure, distribution changed' from NDBCLUSTER
```

Cause

This is a transient error which may be seen when a MySQL Server attempts to access the database while a data node is starting up.

Solution

The application should simply retry the operation when it receives this temporary error.

10 Conclusion

In this guide we have presented some key points which you should consider before embarking on an evaluation of MySQL Cluster. There are many assumptions that can be made about the way MySQL Cluster works which can lead to a less then productive evaluation of the product. Therefore we recommend taking a few minutes to look at the characteristics of your application and the available hardware you plan to use, before beginning in earnest. Of course you

can opt to engage MySQL's professional services division in order to accelerate evaluation and application prototyping, giving you faster time to market of new database applications. For more information about a MySQL Cluster Jumpstart, please see:

http://www.mysql.com/consulting/packaged/cluster.html

11 Additional Resources

MySQL Cluster Getting Started Guide:
http://www.mysql.com/products/database/cluster/get-started.html

MySQL Cluster Reference Guide:
http://dev.mysql.com/doc/mysql-cluster-excerpt/5.1/en/index.html

MySQL Reference Guide (includes Cluster storage engine as well as MySQL Server):
http://dev.mysql.com/doc/refman/5.1/en/index.html

MySQL Cluster API Developer Guide:
http://dev.mysql.com/doc/ndbapi/en/index.html

MySQL Cluster 7.1: Architecture and New Features white paper
http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster7_architecture.php

MySQL Cluster Connector for Java Whitepaper: http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_connector_for_java.php

MySQL Cluster Manager Whitepaper: http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_manager.php