MySQL Cluster

MySQL Cluster Evaluation Guide

Getting the most out of your MySQL Cluster Evaluation

A MySQL® Technical White Paper

June 2013

Table of Contents

1 INTRODUCTION 4

2 WHAT IS MYSQL CLUSTER? 4

2.1 MySQL Cluster Architecture 4

3 KEY FEATURES OF MYSQL CLUSTER 7.3 6

4 POINTS TO CONSIDER BEFORE STARTING AN EVALUATION 7

4.1 Will MySQL Cluster "out of the box" run an application faster than my existing MySQL database? 7

4.2 How do I install MySQL Cluster? 7

4.2.1 MySQL Auto-Installer 7

4.3 Does the database need to run on a particular operating system? 8

4.4 Will the entire database have fit in memory? 8

4.5 Does the application make use of complex JOIN operations? 8

4.6 Does the application make use of full table scans? 9

4.7 Does the database require Foreign Keys? 10

4.8 Does the application require full-text search? 10

4.9 How will MySQL Cluster perform compared to other storage engines? 10

4.10 Does MySQL Cluster's data distribution add complexity to my application and limit the types of queries I can run? 12

4.11 What does my application observe during the (sub-second) failover of a node? 12

4.12 Where does MySQL Cluster sit with respect to the CAP (Consistency, Availability & Performance) Theorem? 12

5 EVALUATION BEST PRACTICES 13

5.1 Hardware 13

5.2 Performance metrics 15

5.3 Test tools 15

5.4 SQL and NoSQL interfaces 15

5.5 Data model and query design 17
5.6 Using disk data tables or in-memory tables 18
5.7 User defined partitioning and distribution awareness 19
5.8 Parallelizing applications and other tips 19
6 ADVICE CONCERNING CONFIGURATION FILES 20
7 SANITY CHECK 21
7.1 A few basic tests to confirm the Cluster is ready for testing 22
8 TROUBLESHOOTING 23
8.1 Table Full (Memory or Disk) 23
8.2 Space for REDO Logs Exhausted 24
8.3 Deadlock Timeout 24
8.4 Distribution Changes 25
9 CONCLUSION 25
10 ADDITIONAL RESOURCES 26

1 Introduction

The purpose of this guide is to enable you to efficiently evaluate the MySQL Cluster database and determine if it is the right choice for your application, whether as part of a new project or an upgrade to an existing service.

This guide presents a brief overview of the MySQL Cluster database and new features in the latest 7.3 GA (General Availability) production release before then discussing:

- Considerations before initiating an evaluation
- Evaluation best practices
- Configuration options and sanity checking
- Troubleshooting

By following the recommendations in this Guide, you will be able to quickly and effectively evaluate MySQL Cluster and ultimately accelerate the time to market of your new service.

2 What is MySQL Cluster?

MySQL Cluster is a write-scalable, real-time, ACID-compliant transactional database, combining 99.999% availability with the low TCO of open source. Designed around a distributed, multi-master architecture with no single point of failure, MySQL Cluster scales horizontally on commodity hardware with auto-sharding to serve read and write intensive workloads, accessed via SQL and NoSQL interfaces.

MySQL Cluster's real-time design delivers predictable, millisecond response times with the ability to service millions of operations per second. Support for in-memory and disk-based data, automatic data partitioning (sharding) with load balancing and the ability to add nodes to a running cluster with zero downtime allows linear database scalability to handle the most unpredictable workloads.

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 SQL. Data can also be accessed and manipulated directly using MySQL Cluster's native NoSQL NDB API. This C++ interface provides fast, low-level connectivity to data stored in a MySQL Cluster.

Additional NoSQL interfaces are available that access the cluster through the NDB API. 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 – 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 tables in the MySQL Cluster database. In MySQL Cluster 7.2, a plug-in for Memcached has been released which enables Memcached clients to directly access tables via the standard Memcached protocol. MySQL 7.3 adds a driver for Node.js which provides direct access (not via SQL) to the data for javascript code running in an application server.

2.1 MySQL Cluster Architecture

Architecturally, MySQL Cluster consists of three different node types, each serving a dedicated role in delivering 99.999% availability with real time performance and linear scalability of both read and write operations.

Figure 1 shows a simplified architecture diagram of a MySQL Cluster consisting of four Data Nodes split across two node groups.

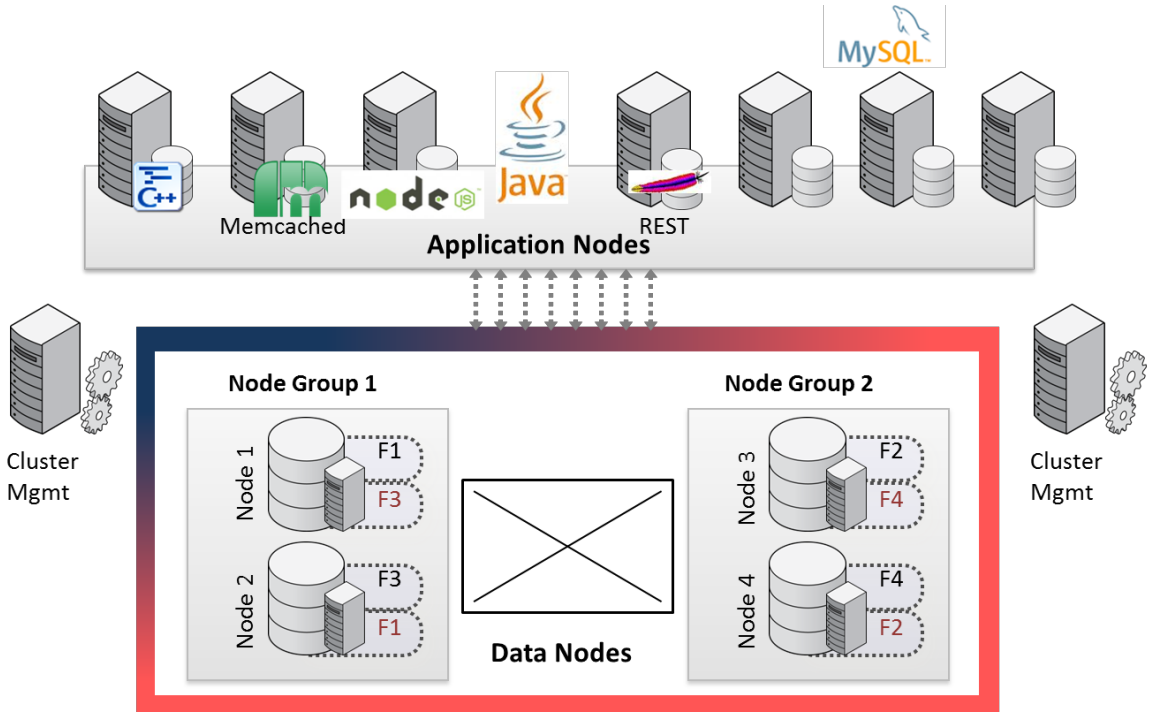


Figure 1 Four Data Node MySQL Cluster

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 (sharding) of tables
- Synchronous replication of data between data nodes
- Transactions and data retrieval
- Automatic fail over
- Automatic resynchronization after failure for self-healing

Tables are automatically sharded across the data nodes, and each data node is a master accepting write operations, making it very simple to scale write-intensive workloads across commodity nodes, with complete application transparency.

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.

It is possible to choose how to store data; either all in memory or with some on disk (non-indexed data only). 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 complete system failure – such as a power outage. 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 provide connectivity from the application logic to the data nodes. 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[1], 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.

As discussed above, the NDB API provides a layer through which additional NoSQL interfaces can directly access the cluster, bypassing the SQL layer, allowing for lower latency and improved developer flexibility.

All Application Nodes can access data from all Data Nodes and so they can fail without causing a loss of service as applications can simply use the remaining nodes.

Management Nodes are responsible for publishing the cluster configuration to all nodes in the cluster and for node management. 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"[2].

You can learn more about how the MySQL Cluster architecture enables the rapid scaling of highly available web, embedded and telecoms services from the Guide posted here:
http://mysql.com/why-mysql/white-papers/mysql_wp_scaling_web_databases.php

3 Key features of MySQL Cluster 7.3

MySQL Cluster 7.3 builds upon the proven capabilities of the 7.0, 7.1 and 7.2 releases with a range of new features designed to enable next generation web, game and telecoms services:

- **Foreign Keys:** Increased application flexibility and compatibility with InnoDB. MySQL Cluster Foreign keys can enforce referential integrity whether using SQL or any of the other supported data access APIs.
- **Auto-Installer:** Browser based installation and configuration for MySQL Cluster. Provide hints about the application together with the list of hosts and allow the auto-installer to do the rest – leaving you with a well-configured, running Cluster.
- **Node.js driver:** Javascript applications can access MySQL Cluster directly without conversion to SQL – store and retrieve javascript objects while being able to access this same data through all other supported APIs.
- **Connection thread scaling:** Increased throughput for each connection between a MySQL Server (or other application node) and the data node for greater performance and scalability together with simplified management.

For more information about all of these new features, refer to the following whitepaper:
http://mysql.com/why-mysql/white-papers/mysql_wp_cluster7_architecture.php

[1] http://www.mysql.com/downloads/connector/
[2] http://www.clusterdb.com/mysql-cluster/mysql-cluster-fault-tolerance-impact-of-deployment-decisions/

4 Points to Consider Before Starting an Evaluation

4.1 Will MySQL Cluster "out of the box" run an application faster than my existing MySQL database?

The reality is probably not without some optimizations 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" significant performance gains can be realized, especially as you start to scale write operations while maintaining very high levels of availability and low latency. You can download the performance optimization guide from here:
http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

4.2 How do I install MySQL Cluster?

Take a look at the Quick Start guides to get you up and running quickly:
http://mysql.com/products/cluster/get-started.html - quickstart

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/5.6/en/index.html) includes instructions on installing the Cluster and MySQL Server components.

You can also use the Bootstrap option within MySQL Cluster Manager to automatically configure and provision a 4-node cluster on a single host – making it ideal for developers building Proof of Concepts. You can learn more about this option here: http://www.clusterdb.com/bootstrap/
MySQL Cluster Manager is part of the commercial MySQL Cluster Carrier Grade Edition offering, but is available for download and use, without obligation, for 30-days from https://edelivery.oracle.com/.

4.2.1 MySQL Auto-Installer

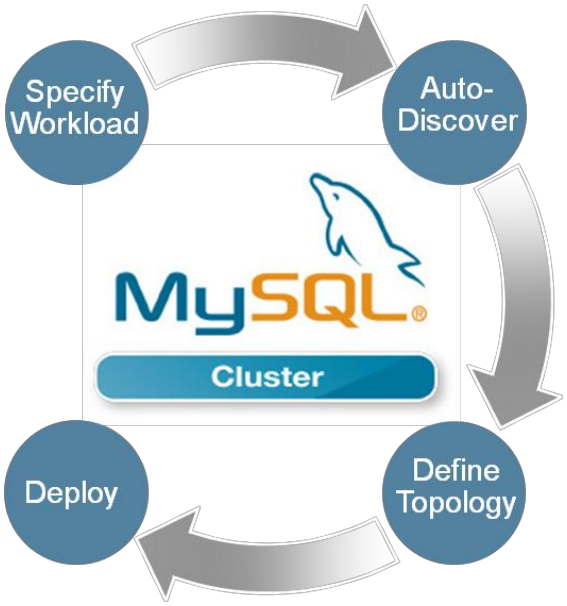


Figure 2 Auto-Installer steps

This is a new option that is part of the MySQL Cluster 7.3 but it can also be used to configure and deploy MySQL Cluster 7.2.

A single command launches the browser-based wizard which then steps you through configuring and deploying the cluster.

The process flow is illustrated in Figure 2 – the user supplies some simple information about the application and provides the names of the hosts to be used in the Cluster. The auto-installer will then discover the resources on those hosts and will propose a topology (what nodes to run on each host) and set of configuration parameters. The user can choose to accept the proposal or make adjustments to it before clicking a button to deploy and start the Cluster.

A tutorial video and a full walkthrough of using this tool is provided in a blog post[1]. From

[1] http://www.clusterdb.com/mysql-cluster/mysql-cluster-7-3-auto-installer/

MySQL Cluster 7.3, the auto-installer is part of the MySQL Cluster 7.3 packages (`bin/ndb_setup` or `setup.bat` on Windows).

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

All MySQL Cluster nodes (Data Nodes, MySQL Server nodes and Management Nodes) must run on a supported Linux, Windows or UNIX operating system. For the latest list of supported platforms, please see:
http://www.mysql.com/support/supportedplatforms/cluster.html

All hosts 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 directing a data node that is running on an x86 host. This restriction does not apply to machines simply running mysql or other clients that may be accessing the cluster's SQL nodes. It is possible to mix application nodes on 32 bit hosts with data nodes on 64 bit hosts or vice-versa.

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

4.4 Will the entire database have 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/5.6/en/mysql-cluster-disk-data.html

To see how much of the configured memory is currently in use by the database, you can query the `ndbinfo.memoryusage` table: http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-ndbinfo-memoryusage.html If using MySQL Cluster CGE then you can view this information over time in a MySQL Enterprise Monitor graph.

If designing a completely new database, the following calculations can be used to help determine the approximate memory sizing requirements for the data nodes:

*(in memory) Data Size * Replicas * 1.25 = Total Database Memory Requirements*

Example: 50 GB * 2 * 1.25 = 125 GB

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

Example: (2 GB * 2 * 1.25)/4 = 31.25 GB

4.5 Does the application make use of complex JOIN operations?

MySQL Cluster 7.2 GA included two new features, which when combined, can improve the performance of joins by a factor of 70x (or even higher) over earlier versions of MySQL Cluster:

- The Index Statistics function enables the SQL optimizer to build a better execution plan for each query. In the past, non-optimal query plans required a manual enforcement of indexes via `USE INDEX` or `FORCE INDEX` to alter the execution plan. `ANALYZE TABLE` must first be run on each table to take advantage of this.

- Adaptive Query Localization (AQL) allows the work of the join to be distributed across the data nodes (local to the data it's working with) rather than up in the MySQL Server; this allows more computing power to be applied to calculating the join as well as dramatically reducing the number of messages being passed around the system.

You can learn more about these features and a sample query here:
http://www.clusterdb.com/aql/

There are certain join operations that won't be distributed down to the data nodes, and therefore not benefit from AQL and so it may be necessary to tune your application to get the best performance.

The use of AQL is controlled by a global variable - `ndb_join_pushdown` – which is on by default.

In order for a join to be able to exploit AQL (in other words be "pushed down" to the data nodes), it must meet the following conditions:

1. Any columns to be joined must use exactly the same data type. (For example, if an `INT` and a `BIGINT` column are joined, the join cannot be pushed down). This includes the lengths of any `VARCHAR` columns.
2. Joins referencing `BLOB` or `TEXT` columns will not be pushed down.
3. Explicit locking is not supported; however, the NDB storage engine's characteristic implicit row-based locking is enforced.
4. In order for a join to be pushed down, child tables in the join must be accessed using one of the `ref`, `eq_ref`, or `const` access methods, or some combination of these methods. These access methods are described in http://dev.mysql.com/doc/refman/5.6/en/explain-output.html
5. Joins referencing tables explicitly partitioned by [`LINEAR`] `HASH`, `LIST`, or `RANGE` currently cannot be pushed down
6. If the query plan decides on `Using join buffer` for a candidate child table, that table cannot be pushed as child. However, it might be the root of another set of pushed tables.
7. If the root of the pushed join is an `eq_ref` or `const`, only child tables joined by `eq_ref` can be appended. (A `ref` joined table will then likely become a root of another pushed Join)

These conditions should be considered when designing your schema and application queries – for example, to comply with constraint 4, attempt to make any table scan that is part of the Join be the first clause.

Where a query involves multiple levels of joins, it is perfectly possible for some levels to be pushed down while others continue to be executed within the MySQL Server.

If your application consists of many of these types of join operations which cannot be made to exploit AQL, other MySQL storage engines such as InnoDB will present a better option for your workload.

4.6 Does the application make use of full table scans?

If so, then unless you are willing to modify your application, you will likely experience lower performance levels compared to other MySQL storage engines for those queries. This is due to the manner in which data is partitioned and distributed across many Data Nodes. Applications that mostly (but not exclusively) perform primary key operations will typically benefit most from MySQL Cluster's distribution of data.

4.7 Does the database require Foreign Keys?

Foreign keys are supported from MySQL Cluster 7.3. In most cases, Foreign Keys (FKs) will behave in exactly the same way for MySQL Cluster and InnoDB – there are a small number of exceptions:

- InnoDB doesn't support "No Action" constraints, MySQL Cluster does
- You can choose to suspend FK constraint enforcement with InnoDB using the `FOREIGN_KEY_CHECKS` parameter; at the moment, MySQL Cluster ignores that parameter.
- You cannot set up FKs between 2 tables where one is stored using MySQL Cluster and the other InnoDB.
- You cannot cause a Primary Key to be updated by a FK constraint when using MySQL Cluster.
- InnoDB doesn't permit the use of FK constraints on table where user-defined partitioning is being used – MySQL Cluster does. This is significant as user-defined partitioning is commonly used with MySQL Cluster to create distribution-aware applications (see section 5.7) in order to scale linearly.

The use of Foreign Keys for MySQL Cluster is described in this post[1].

4.8 Does the application require full-text search?

Full-text search is currently not supported by the MySQL Cluster storage engine. A common approach is to provide full text search by replicating the clustered database into a read-only MyISAM (or from MySQL 5.6 - InnoDB) slave server or to use an external search engine with MySQL Cluster, i.e. Sphinx or Lucene.

4.9 How will MySQL Cluster 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 MyISAM table and when we altered it to InnoDB it took several seconds to migrate. When we altered the table to NDB (MySQL Cluster), it took five times as long. We expect it to be faster."

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

The above scenarios ignore the fundamental difference in the way in which MySQL Cluster storage 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 and have high concurrency, but it incurs a cost in network access when accessing data between a MySQL Server and tables distributed across Data Nodes.

In the following figure below you notice that in order to satisfy the query, multiple Data Nodes must participate in the retrieval of data. Adaptive Query Localization discussed above can push certain classes of queries down into the data nodes for local processing, reducing the network messaging overhead and accelerating query performance.

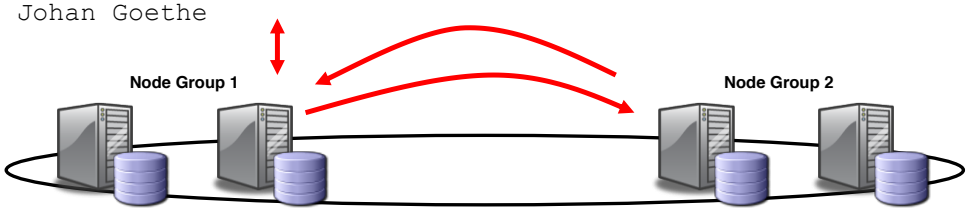
[1] http://www.clusterdb.com/mysql-cluster/foreign-keys-in-mysql-cluster/

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 3 Data access across Data Nodes

As a point of comparison, in the following Figure, we can see that with traditional storage engines, there is no networking communication overhead incurred to 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 4 Data access with a traditional storage engine

MySQL Cluster provides the best performance when used to process large volumes of relatively small transactions (i.e. less than a thousand operations per transaction) and so adapting your application to follow that pattern will produce the best results.

4.10 Does MySQL Cluster's data distribution add complexity to my application and limit the types of queries I can run?

The answer to both these questions is "no"!

By default, tables are automatically partitioned (sharded) across data nodes by hashing the primary key. Other partitioning methods are supported, but in most instances the default is acceptable.

As the sharding is automatic and implemented at the database layer, application developers do not need to modify their applications to control data distribution – which significantly simplifies scaling.

In addition, applications are free to run complex queries such as join operations across the shards, therefore users do not need to trade functionality for scalability.

4.11 What does my application observe during the (sub-second) failover of a node?

If a management node should fail, there is no observable impact to the application unless it was the only management node and the application then attempts to make a new connection to the data nodes using the native C++ API – such an attempt would fail until at least one management node is running; again, all existing connections would be unaffected.

If a MySQL Server should fail then the application will observe an error or a timeout and should send its database requests to another MySQL Server in the cluster – in many cases, this can be handled by the connectors (for example for PHP and JDBC).

If a data node should fail then the application may see a temporary error. On receiving such an error, it is the responsibility of the application to either:

- Retry the operation
- Accept that the operation has failed and reflect this back up the chain (for example, informing the user that they need to retry their request).

4.12 Where does MySQL Cluster sit with respect to the CAP (Consistency, Availability & Performance) Theorem?

The CAP Theorem[1] states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

1. Consistency (all nodes see the same data at the same time)
2. Availability (a guarantee that every request receives a response about whether it was successful or failed)
3. Partition tolerance (the system continues to operate despite arbitrary message loss)

According to the theorem, a distributed system can satisfy any two of these guarantees at the same time, but not all three.

[1] http://en.wikipedia.org/wiki/CAP_theorem

MySQL Cluster survives many network partition scenarios, but a sufficiently serious partitioning must result in either availability or consistency being sacrificed. In terms of CAP, MySQL Cluster will sacrifice availability to maintain consistency in the event of an unsurvivable network partition. The nodes will shut-down and attempt to restart and form a cluster; this is sometimes referred to as "failing fast". More details can be found at http://messagepassing.blogspot.co.uk/2012/03/cap-theorem-and-mysql-cluster.html

There are steps that can be taken when architecting your Cluster to ensure that no single point of failure should result in partitioning serious enough to cause the cluster to shut-down – refer to http://www.clusterdb.com/mysql-cluster/mysql-cluster-fault-tolerance-impact-of-deployment-decisions/ for details.

5 Evaluation Best Practices

Whether you are migrating an existing database to MySQL Cluster or building a new application, there are guidelines that should be observed in ensuring a productive and successful evaluation.

For example, if you are considering migrating to MySQL Cluster, it is highly likely that the current data model and access patterns are optimized for the original storage engine. Hence, you may have to modify certain schemas and queries to achieve the highest levels of performance from MySQL Cluster.

The following section presents the key best practices you should consider before embarking on an evaluation.

5.1 Hardware

Do you have enough hosts configured with sufficient resources to conduct the evaluation?

For early evaluations and prototyping, it is often sufficient to deploy all nodes of the cluster onto a single physical host. Consider using the Bootstrap option in MySQL Cluster Manager to get started quickly (see "How do I install MySQL Cluster" in the previous section) or the new Auto-Installer.

If you want to test the full performance and HA (High Availability) characteristics of MySQL Cluster, we recommend the following minimal hardware setup:

- 2 hosts each running one Data Node
- 2 hosts each running one Application Node and Management Server

This will ensure a minimum level of redundancy on all processes that constitute a MySQL Cluster.

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

The recommended hardware specification for each node type is as follows:

Data Nodes

- Up to 32 x x86-64 bit CPU cores. Use as high a frequency as possible as this will enable faster processing of messages between nodes
- Large CPU caches assist in delivering optimal performance
- 64-bit hosts with enough RAM to store your in-memory data set - see the section above for the recommended formula for estimating memory requirements
- Linux, Solaris or Windows operating systems
- 2 x Network Interface Cards and 2 x Power Supply Units for hardware redundancy

To avoid a single point of failure from the data centre, the 2 Data Nodes making up a Node Group should use different infrastructure (for example power and cooling).

It is important to ensure systems are configured to reduce swapping to disk whenever possible.

As a rule of thumb, have 7x times the amount of `DataMemory` configured for disk space for each data node. This space is needed for storing 2 Local Checkpoints (LCPs), the Redo log and 3 backups. You will also want to allocate space for table spaces if you are making use of disk-based data – including allowing extra space for the backups.

Disk-drive recommendations are as follows. Having a fast, low-latency disk subsystem is very important and will affect check pointing and backups.

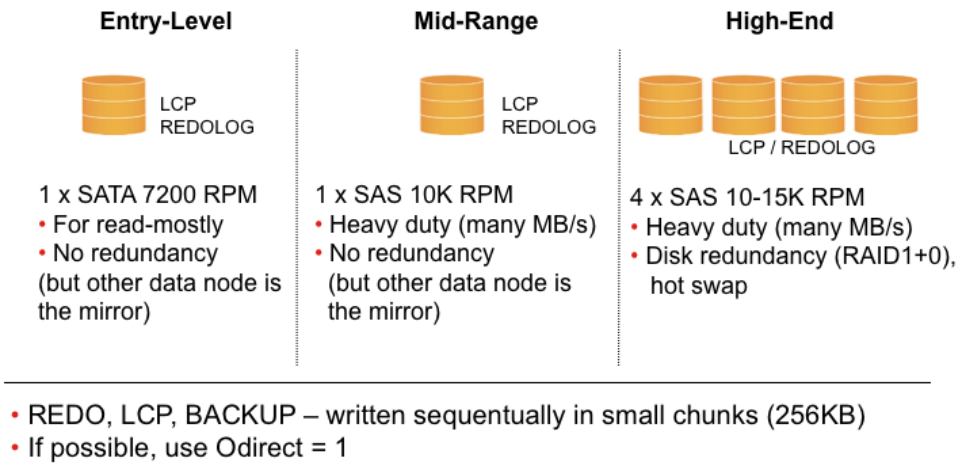


Figure 5 Recommended Disk Drive Configurations for Checkpointing & REDO Logs

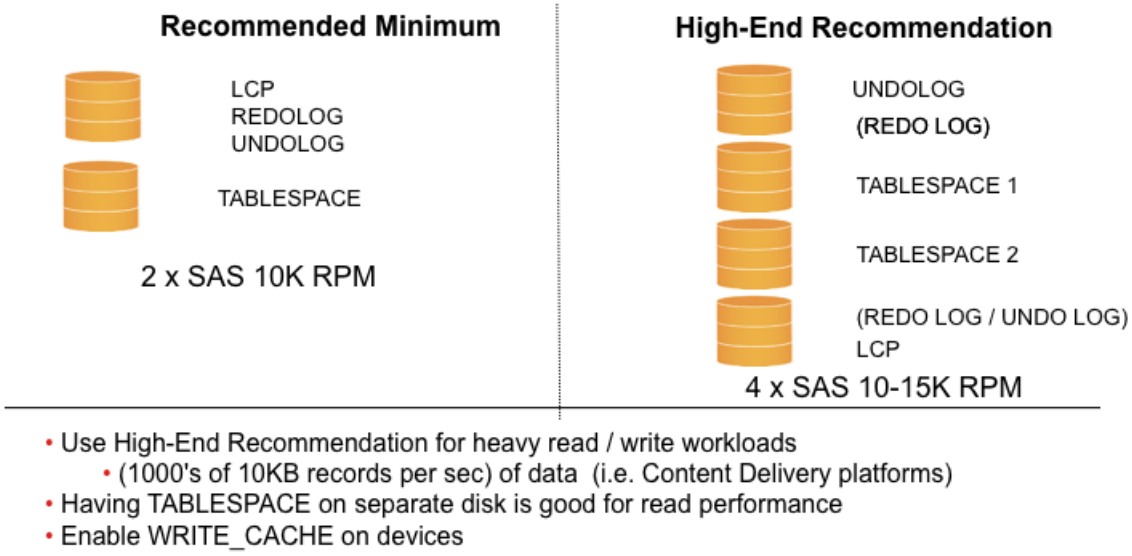


Figure 6 Recommended Disk Drive Configurations for Disk-Based Tables

MySQL Servers/Application Nodes

- 4 - 32 x86-64 bit CPU cores
- Minimum 4GB of RAM. Memory is not as critical at this layer, and requirements will be influenced by connections and buffers
- 2 x Network Interface Cards and 2 x Power Supply Units for hardware redundancy

Deployment of the Cluster

For evaluation purposes, it is recommended to deploy the data and application nodes on a dedicated network (i.e. IP addresses starting at 10.0.1.0), using 1 Gigabit Ethernet as a minimum transport. The MySQL servers would then also be attached to the public network for application access and optionally geographic replication to a second data center. It is also possible to locate the data nodes across a WAN link, but this is not recommended for initial evaluation purposes.

To provide resilience to network failures, it is recommended to configure each server with bonded and redundant Network Interface Cards (NICs), connected to redundant switches. If multiple data nodes are deployed onto a single host (i.e. Oracle Sun T-series systems), it is recommended to bond four or more NICs together to handle the increased network bandwidth requirements.

5.2 Performance metrics

Key metrics may include:

- How many transactions per second are required?
- What is the required response time?
- How is performance scaled if I add an application node or a data node?
- How is performance impacted if the cluster loses an application or data node?
- How is performance impacted when running on-line maintenance activities such as backups and schema changes?
- How is the operation of the database effected when introducing faults such as pulling network cables, killing database processes or turning off the power?

5.3 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.

5.4 SQL and NoSQL interfaces

As MySQL Cluster stores tables in data nodes, rather than in the MySQL Server, there are multiple interfaces available to access the database.

Developers have a choice between:

- SQL for complex queries and access to a rich ecosystem of applications and expertise
- Simple Key/Value interfaces bypassing the SQL layer for fast reads & writes
- Real-time interfaces for micro-second latency

With this choice of interfaces, developers are free to work in their own preferred environments, enhancing productivity and agility, enabling them to deliver new services to market faster. If required,

multiple access methods can be used for the same data set (for example provisioning through SQL and network access through a key-value API.

The following figure aims to summarize the capabilities and use-cases for each API.

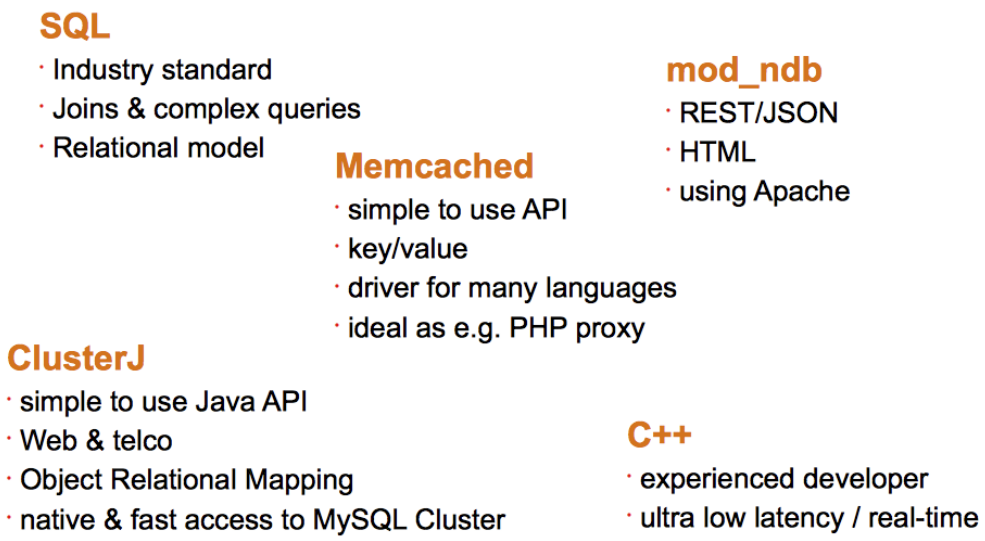


Figure 7 MySQL Cluster API Cheat-Sheet

A common choice of API is to use the MySQL Server to present a standard SQL interface to the data nodes, providing connectivity to all of the standard MySQL connectors including:

- Common web development languages and frameworks, i.e. PHP, Perl, Python, Ruby, Ruby on Rails, Spring, Django, etc
- JDBC (for additional connectivity into ORMs including EclipseLink, Hibernate, etc)
- .NET
- ODBC

In environments where real-time performance is critical, we recommend considering the use of the native NoSQL APIs, such as the NDB API (C++ API), memcached, Node.js (javascript) 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 tune response time-critical requests
- Batch interface – it is possible to batch requests such as 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 lower latency compared to SQL, often more depending on the types of requests, and whether the batch interface is used

You can learn more about the NDB API from the developer guide:
http://dev.mysql.com/doc/ndbapi/en/index.html

You can learn more about the Memcached API for MySQL Cluster here:
http://www.clusterdb.com/memcached/

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

The MySQL Cluster 7.3 adds a driver for Node.js which allows a JavaScript application to have direct access (not passing through SQL) to the data held within the data nodes. This new functionality is documented in the MySQL Cluster API Developer Guide[1] and a tutorial[2] is also available.

5.5 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. Ensuring that joins meet the requirements for AQL and avoiding full table scans can help with this
- Looking up data in a hash table is a constant time operation, unaffected by the size of the data set
- Looking up data in a tree (T-tree, B-tree etc) structure is logarithmic (O (log n)).

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. The network identifiers for the subscriber are then mapped to this id. For every subscriber table, the subscriber id is either:

- The sole primary key in the table participating in the subscriber profile is the global subscriber id (gsid)
- or part of the 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 a composite primary key then there are two possibilities:

Use an ordered index scan to fetch the data:

```
SELECT * FROM service WHERE gsid=1202;
```

[1] http://dev.mysql.com/doc/ndbapi/en/ndb-nodejs.html
[2] http://www.clusterdb.com/mysql/mysql-cluster-with-node-js/

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);
```

By selecting the gsid to be the partitioning key for each of the tables, any transaction involving any tables for the same subscriber can be executed within a single data node – this is referred to as making the application distribution aware (see section 5.7 for details).

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.

Aggregated information on what the application queries are causing to happen in the data nodes can be viewed through the `ndbinfo.counters` table – described in http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-ndbinfo-counters.html

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” available at: http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

5.6 Using disk data tables or in-memory tables

You can have a bigger dataset than you have physical 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, 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 becoming 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.

The `ndbinfo.diskpagebuffer` table provides information on the effectiveness of this cache; users of MySQL Cluster CGE can also see this information presented over time through MySQL Enterprise Monitor graphs as described in http://www.clusterdb.com/mem+/

5.7 User defined partitioning and distribution awareness

As discussed above, when adding rows to a table that uses MySQL Cluster as the storage engine, each row is assigned to a partition (shard) where that partition is mastered by a particular data node in the Cluster. The best performance is often achieved when all of the data required to satisfy a transaction is held within a single partition. This means that it can be satisfied within a single data node rather than being bounced back and forth between multiple nodes, resulting in extra latency.

By default, MySQL Cluster partitions the data by hashing the primary key.
We can override the default behavior by telling MySQL Cluster which fields from the Primary Key should be fed into the hash algorithm. In this way, you can engineer the system such that primary key look ups for related data over multiple tables can be satisfied within a single data node. In a similar way many index scans can be contained within a single data node – a process referred to as partition pruning.

The benefits from partition pruning are dependent on the number of data nodes in the Cluster and the size of the result set – with the best improvements achieved with more data nodes and less records to be retrieved.

Figure 8 shows how partition pruning (orange bars) reduces latency for smaller result sets, but can actually increase it for larger result sets. Note that shorter bars/lower latency represents better performance.

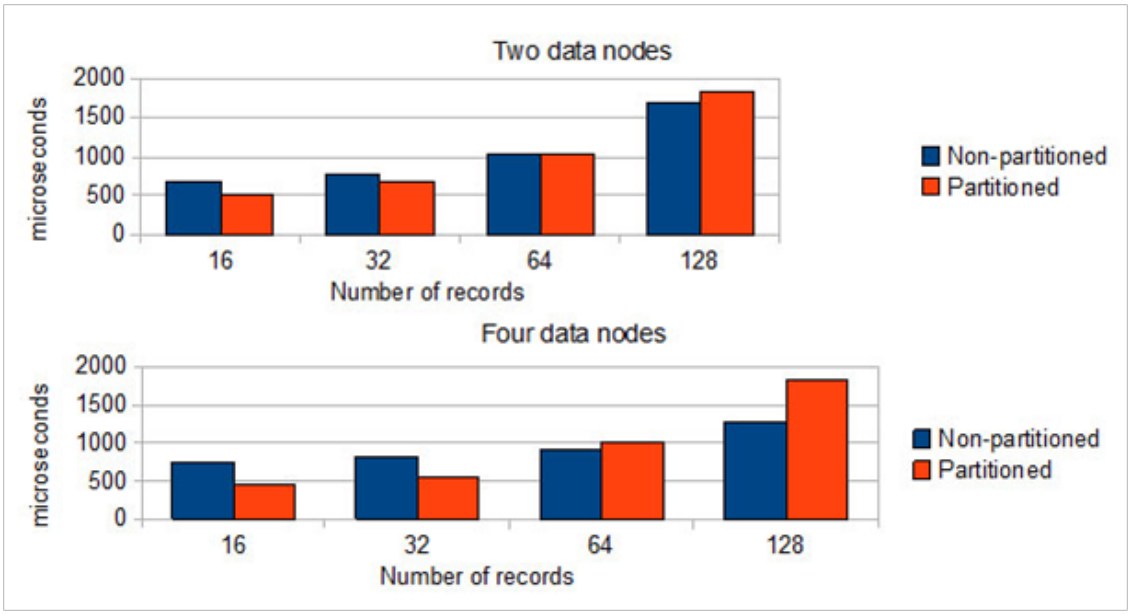


Figure 8 Effects of index scan partition pruning

5.8 Parallelizing applications and other tips

As mentioned MySQL Cluster is a distributed, auto-sharded database. This means that there is often more than one Data Node that can work in parallel to satisfy application requests.

Additionally, MySQL Cluster 7.2 enhanced multi-threading so data nodes can now effectively exploit multiple threads / cores. To use this functionality, the data nodes should be started using the `ndbmtd` binary rather than `ndbd` and `config.ini` should be configured correctly - http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-programs-ndbmtd.html.

Parallelization can be achieved in MySQL Cluster in several 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
- Utilizing multiple connections between each Application Node and the Data Nodes (connection pooling)

How many threads and how many applications are needed to drive the desired load has to be studied by benchmarks. One approach to 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 you have access to additional performance enhancements that allow better utilization on multi-core / thread CPUs, including:

- Reduced lock contention by having multiple connections from one MySQL Server to the Data Nodes (`--ndb-cluster-connection-pool=X`): http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-program-options-mysqld.html#option_mysqld_ndb-cluster-connection-pool. Note that from MySQL Cluster 7.3 the lock contention has been greatly reduced and so you are likely to need far fewer connections and in many cases just one
- Setting threads to real-time priority: http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-ndbd-definition.html#ndbparam-ndbd-realtimescheduler
- Locking Data Node threads (kernel thread and maintenance threads to a CPU): http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-ndbd-definition.html#ndbparam-ndbd-lockexecutethreadtocpu
- The "Guide to Optimizing Performance of the MySQL Cluster Database"[1] includes guidelines for configuring the correct number of threads for the data nodes

6 Advice Concerning Configuration Files

With the introduction of the auto-installer in MySQL Cluster 7.3 (see section 4.2.1) you will typically be able to avoid creating your own configuration files; this section provides some tips if you prefer to create your own.

[1] http://www.mysql.com/why-mysql/white-papers/guide-to-optimizing-performance-of-the-mysql-cluster/

Below is a 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

7 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 communicating with 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

7.1 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 from the management server (ndb_mgm)

```
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.

Both forums and mailing lists are available to you in assisting with initial configuration issues. The MySQL Cluster forum can be reached at:
http://forums.mysql.com/list.php?25

The MySQL Cluster mailing list can be reached at:
cluster@lists.mysql.com

8 Troubleshooting

If the evaluation exercise is looking to test the boundary conditions 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.

8.1 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.

or

- The hash index is full, with over 49,000,000 rows per partition.

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

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`.

If the hash index is full:

By increasing the number of partitions, the number of rows per partition is lowered. By default MySQL Cluster automatically creates 1 partition per LQH thread and so the number of partitions can be implicitly increased by any combination of:

- Add more data nodes
- Increase the number of LQH processes per data node by using `ndbmtd` and setting `MaxNoOfExecutionThreads` appropriately http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-programs-ndbmtd.html#ndbparam-ndbmtd-maxnoofexecutionthreads

After performing those operations you should run:

```
mysql> ALTER TABLE mytab REORGANIZE PARTITION;
```

Alternatively (or in addition) you can give MySQL Cluster a hint that more partitions are required by providing a maximum (large) size for the table:

```
ALTER TABLE mytab MAX_ROWS=100000000;
```

Finally you have the option of explicitly indicating the number of partitions to use but this can limit what on-line repartitioning you can subsequently perform on the table. Therefore, this would not be the preferred approach in most cases:

```
mysql> ALTER TABLE mytab  PARTITION BY KEY (`pk`) PARTITIONS
32;
```

8.2 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 16). 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.

8.3 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).

8.4 Distribution Changes

Error

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

Cause

This is a transient error that 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.

As mentioned in the previous section, both forums and mailing lists are available to you in assisting with initial configuration issues. The MySQL Cluster forum can be reached at:
http://forums.mysql.com/list.php?25

The MySQL Cluster mailing list can be reached at:
cluster@lists.mysql.com

9 Conclusion

In this guide we have presented best practices that will enable you to quickly and efficiently evaluate the MySQL Cluster database. 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 engage Oracle's MySQL Cluster consulting group in order to accelerate evaluation and application prototyping, giving you faster time to market of new database applications. Oracle's MySQL Cluster consultants offer the deepest level of expertise anywhere, having worked with some of the world's largest web, telecoms and government groups in MySQL Cluster projects: http://www.mysql.com/consulting/

10 Additional Resources

MySQL Cluster Quick Start Guides:
http://www.mysql.com/products/cluster/get-started.html#quickstart

Bootstrapping MySQL Cluster:
http://www.clusterdb.com/bootstrap/

MySQL Cluster Documentation:
http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster.html

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

MySQL Cluster 7.3: New Features white paper
http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster7_architecture.php

Guide to MySQL Cluster Performance Optimization:
http://mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

MySQL Cluster User Forum and Mailing List:
http://forums.mysql.com/list.php?25
cluster@lists.mysql.com