MySQL Cluster 7.0 & 7.1: Architecture and New Features

A MySQL® Technical White Paper by Oracle

April 2010

Table of Contents

1 INTRODUCTION 3

2 NEW FEATURE OVERVIEW 3

3 MYSQL CLUSTER ARCHITECTURE OVERVIEW 3

4 ENHANCED SCALABILITY AND PERFORMANCE 10

4.1 Multi-Threaded Data Nodes 10
4.1.1 Benefits 11
4.1.2 Configuring NDB Data Nodes to be Multi-Threaded 12

4.2 On-Line Add Node 12

4.3 Multi-Threaded Disk Data File Access 16

4.4 Improved Large Record Handling 17

5 WINDOWS PLATFORM 17

6 SIMPLIFIED CLUSTER MONITORING AND MANAGEMENT 18

6.1 ndbinfo Data Node Statistics 18
6.1.1 ndbinfo.counters 18
6.1.2 ndbinfo.logbuffers 19
6.1.3 ndbinfo.logspaces 20
6.1.4 ndbinfo.memoryusage 20
6.1.5 ndbinfo.nodes 21
6.1.6 ndbinfo.transporters 21

6.2 MySQL Cluster Manager 22

6.3 Snapshot Backup Options for MySQL Cluster 22

6.4 Configuration Data Cached 24

6.5 Transactions for schema changes 25

7 CONNECTORS 25

7.1 MySQL Cluster Connector for Java 25

7.2 High Availability Directory Back-End 26

8 CONCLUSION 27

9 LINKS AND REFERENCES 28

1 Introduction

MySQL Cluster is a scalable, high-performance, clustered database, originally developed for some of the world's most demanding applications found in the telecommunications industry. Often these telecom applications required that the database's availability exceed 99.999%.

Since MySQL Cluster's introduction in 2004, new feature sets have been steadily introduced into the product. This has broadened its appeal for new application use-cases, markets and industries. MySQL Cluster is now being adopted not only as a database for traditional telecom applications like, HLR (Home Locator Registry) or SLR (Subscriber Locator Registry), it is now also being widely deployed for Service Delivery Platforms, Value-Added Services, VOIP, web applications, internet billing, session management, eCommerce sites, search engines and traditional back office applications. Industry use extends to include web, enterprise and government organizations.

With the release of MySQL Cluster 7.0 & 7.1, many new features and improvements have been introduced to the already popular MySQL Cluster 6.3. In this paper we will explore these new features which have been introduced in MySQL Cluster since the MySQL Cluster 6.3 release so you can better understand the opportunities and benefits using MySQL Cluster 7.0/7.1 can bring to your high availability applications and services.

2 New Feature Overview

The following set of enhancements does not represent the complete list of new features introduced in this latest version of MySQL Cluster. Please see the documentation for the full feature set: http://dev.mysql.com/doc/#cluster

Feature	Category	Release
Multi-Threaded Data Node	Scalability/Performance	7.0
On-Line Add Node	Scalability/Performance	7.0
Multi-Threaded Disk Data File Access	Scalability/Performance	7.0
Improved Large Record Handling	Scalability/Performance	7.0
Windows Platform	Platform	7.0 & 7.1
Carrier-Grade Directory Back-End	Connectors	7.0
Snapshot Options for MySQL Cluster	Monitoring & Management	7.0
Configuration data cached	Monitoring & Management	7.0
Transactions for schema changes	Monitoring & Management	7.0
ndbinfo – statistic reporting	Monitoring & Management	7.1
MySQL Cluster Manager	Monitoring & Management	7.1
MySQL Cluster Connector for Java	Connectors	7.1

3 MySQL Cluster Architecture Overview

Before embarking on a technical examination of MySQL Cluster's new features, it makes sense to quickly review the product's architecture and how it functions.

MySQL Cluster is a high availability database built using a unique shared-nothing architecture and a standard SQL interface. The system consists of a number of communicating processes, or nodes that can be distributed across hosts to ensure continuous availability in the event of server or network failure. MySQL Cluster uses a storage engine, consisting of a set of data nodes to store data which can be accessed using standard SQL with MySQL Server, through the NDB API for real-time access or via a number of other access methods (such as LDAP or JPA) that access the NDB API.

The NDB API is an object-oriented application programming interface for MySQL Cluster that implements indexes, scans, transactions, and event handling. NDB transactions are ACID-compliant in that they provide a means to group together operations in such a way that they succeed (commit) or fail as a unit (rollback).

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

MySQL Cluster consists of three kinds of nodes:

1. ***Data Nodes*** store all the data belonging to the MySQL Cluster. Data is instantaneously replicated between these nodes to ensure it is continuously available in the event one or more nodes fail. These nodes also manage database transactions. Increasing the number of *replicas* yields additional data redundancy. Applications utilizing the NDB API access the Data Nodes directly rather than through a MySQL Server.

2. ***Management Server Nodes*** handle system configuration at startup and are leveraged when there is a change to the cluster. At a minimum, just one Management Node can be configured but for maximum availability you should run additional nodes. A management node is required when starting, stopping or reconfiguring the cluster; it also acts as the arbitrator in certain data node failure scenarios to ensure that a 'split brain' scenario is avoided.

3. ***MySQL Server Nodes*** enable SQL access to the clustered Data Nodes. This provides developers a standard SQL interface to program against. MySQL Server in turn, handles sending requests to the Data Nodes, thus eliminating the need for cluster specific, low-level programming within the application. Additional MySQL Server Nodes are typically added in order to increase performance. This arrangement naturally lends itself to an implementation of the Scale-out methodology for increasing scalability, capacity and performance.

Applications that need the maximum real-time performance from the MySQL Cluster database should use the NDB API directly rather than going through a MySQL Server. This C++ API is described in detail in the *MySQL Cluster API Developer Guide*; available at: http://dev.mysql.com/doc/#cluster

There are also additional connector technologies for applications to use that access the data through the NDB API, providing ultimate developer independence and the easy integration of MySQL Cluster to a broad range of web and enterprise applications while benefiting from the performance benefits of the NDB API.

The database is internally broken down into partitions so that it can be split amongst Data Nodes. To avoid a single point of failure, two or more Data Nodes hold all of the data for a particular partition and they are referred to as a Node Group.

Below in Figure 1 is an illustration of a basic MySQL Cluster configuration consisting of:

- One MySQL Server Node
- One Management Server Node
- Two Data Nodes (forming one Node Group) for extra availability

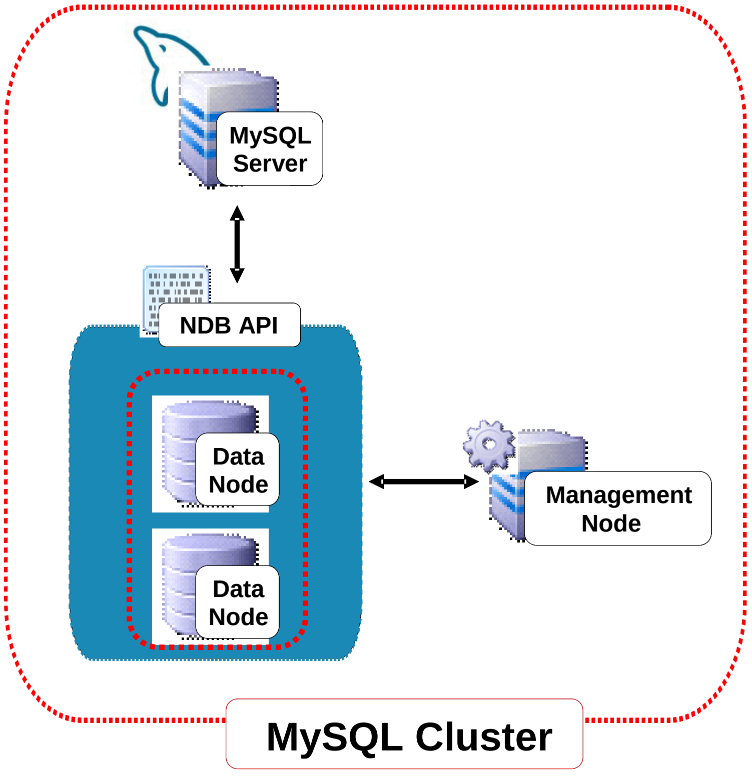


Figure 1 Basic MySQL Cluster Configuration

In Figure 2 we illustrate a MySQL Cluster configuration in which we have employed Scale-out in order to increase performance & capacity. We have done so by adding two additional MySQL Server Nodes, as well as 2 extra data nodes and an additional Management Server for process redundancy. This configuration now consists of:

- Three MySQL Server Nodes
- Two Management Server Nodes (added for redundancy of maintenance operations rather than capacity or performance)
- Four Data Nodes (forming two Node Groups) for extra performance, capacity and availability

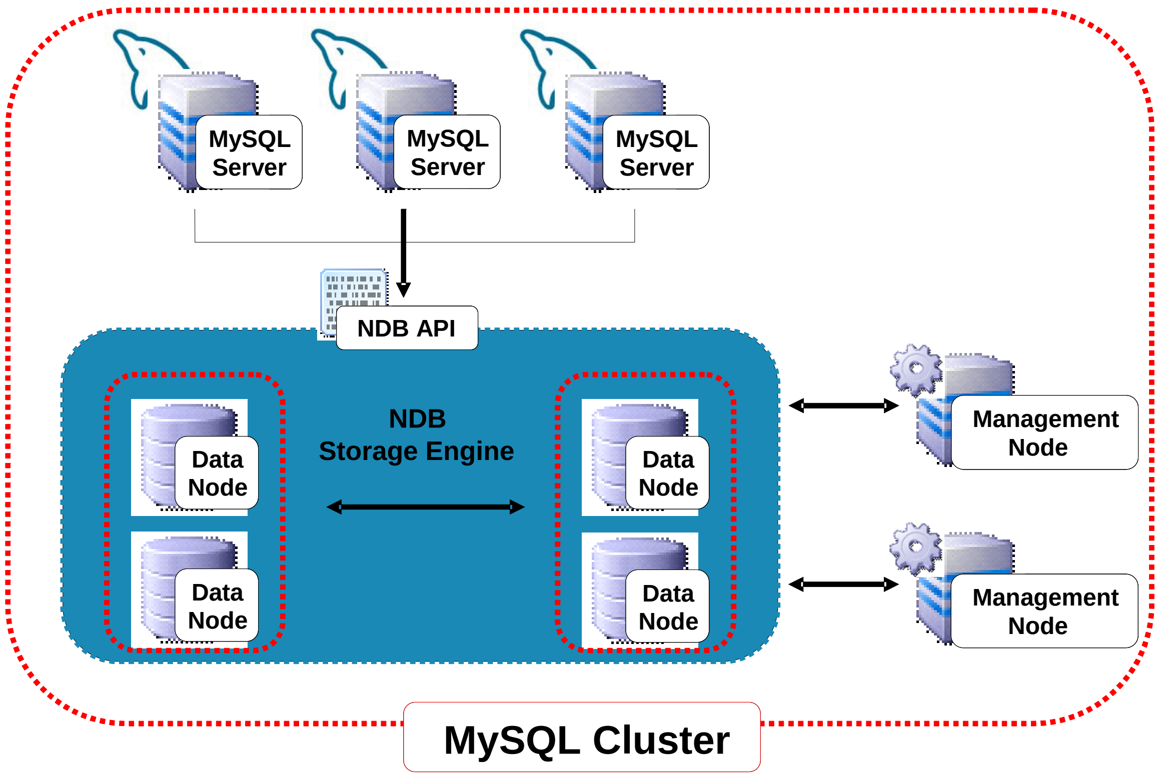


Figure 2 Scaling Out SQL Capacity & Performance

All MySQL Servers in a MySQL Cluster are connected to all Data Nodes as illustrated in Figure 2. All updates made by the MySQL Servers are stored in the data nodes and become immediately visible to other connected MySQL Servers at transaction commit time giving a single database image. This means that as soon as a transaction has been executed on one MySQL server, the result is visible through all MySQL servers in the Cluster.

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

- If a Data Node fails, then the MySQL Server can use any other Data Node in the node group to execute transactions.
- The data within a Data Node is replicated on all nodes within the Node Group. If a Data Node fails, then there is always at least one other Data Node storing the same information.
- Duplicate Management Server Nodes can be deployed so that no management or arbitration functions are lost if a single Management Server fails.

Designing the cluster in this way makes the system reliable and highly available since single points of failure have been minimized. Any node can be lost without it affecting the system as a whole. An application can, for example, continue executing even though a Data Node is down provided that there are one or more surviving nodes in its node group. Techniques used to increase the reliability and availability of the database system include:

- Data is synchronously replicated between all data nodes in the node group. This leads to very low fail-over times in case of node failures.
- Nodes execute on multiple hosts, allowing MySQL Cluster to operate even during hardware failures.
- Nodes are designed using a shared-nothing architecture. Each Data Node has its own disk and memory storage.

- Single points of failure have been minimized. Any one (in some cases more) node can be lost without any loss of data and without stopping applications using the database. Similarly, the network can be engineered such that there is no single point of failure in the interconnects.

In addition to the site-level high-availability achieved through the redundant architecture of a Cluster, geographic redundancy can be achieved using asynchronous replication between 2 or more Clusters.

Using row-based replication you have the ability to replicate from a MySQL Cluster to another MySQL Cluster or to other non-Cluster MySQL databases. It is possible to create the following master/slave configurations:

- MySQL Cluster to MySQL Cluster
- MySQL Server (MyISAM, InnoDB, etc) to MySQL Cluster
- MySQL Cluster to MySQL Server (MyISAM, InnoDB, etc)

In addition, it is possible for a single MySQL Cluster database to be replicated to one or more MySQL Cluster deployment as well as to one or more databases using different storage engines.

If the goal is to achieve the highest possible availability, a Cluster to Cluster replication configuration with redundant replication channels, will be the ideal.

Some popular reasons for implementing replication include:

- Replication to achieve higher availability within the data center or across a geographic WAN
- A replicated database for fail-over
- Provide lower latency data access in different geographies.
- A replicated database for complex data analysis that mustn’t impact the performance of the main, real-time production database

First, let’s review some basics about MySQL replication regardless if you use MySQL Cluster or not. Replication includes a master server and a slave server, the master being the source of the operations and data to be replicated and the slave being the recipient of these. This configuration is illustrated in Figure 3. A single server can be a Master or a Slave at different times, and in some topologies, it is a Master and Slave at the same time.

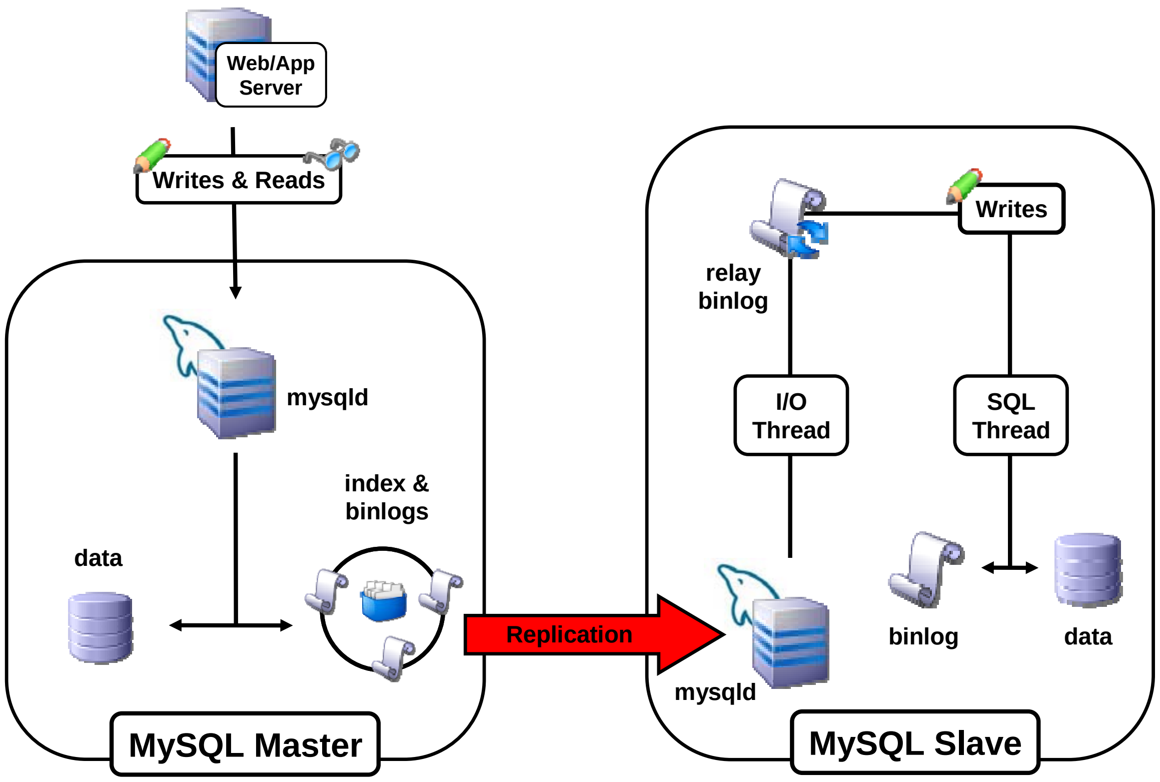


Figure 3 Generic MySQL Replication

Although the implementation of replication within MySQL Cluster is architecturally similar, there are a few differences which should be explained. A MySQL Cluster to MySQL Cluster replication configuration is illustrated below in Figure 4:

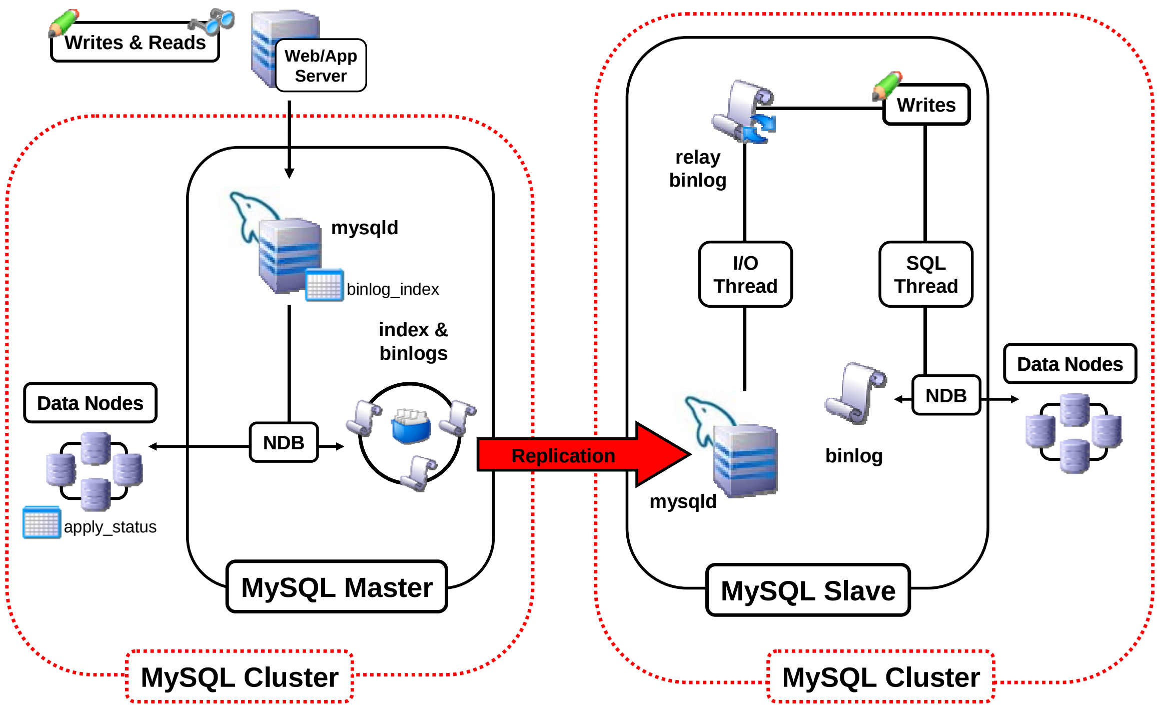


Figure 4 MySQL Cluster Geographic Replication

In the above configuration, the replication process is the one in which consecutive states of a master cluster are logged and saved to a slave cluster. This process is achieved by a thread known as the *NDB binlog injector thread*, which runs on each MySQL server and creates a binary log (also known as a binlog). The NDB binlog injector thread guarantees that all changes in the cluster producing the binary log - and not just those changes that are effected via the MySQL Server - are inserted into the binary log in the correct order. This is important because the NDB storage engine supports the NDB API which allows you to write applications that interface directly with the NDB kernel, bypassing the MySQL Server and normal SQL syntax.

The diagrams show a single MySQL Server (mysqld) acting as the SQL front-end as well as the replication master.

MySQL Server supports Statement Based Replication (SBR) and Row Based Replication (RBR); MySQL Cluster asynchronous replication always uses RBR.

Please refer to the “MySQL Cluster Reference Guide” for more details on advanced features of Cluster replication – including different Master/Slave topologies (such as active-active), conflict detection and resolution, multiple replication channels, synchronizing of the binlog with online Cluster backup.

Not all data needs to have the same read/write performance possible using MySQL Cluster in-memory storage. For that subset of data, MySQL Cluster can store it on disk rather than in-memory, giving you the ability to create even larger database clusters and still manage them effectively. Note that with “in-memory” data, changes are asynchronously checkpointed to disk to ensure that the data can be recovered even if the memory copy is lost for all of the data nodes in a node group.

Data that does not necessarily demand the very high performance characteristics of RAM-based data will be the best candidates for leveraging disk-data. Also, for those who have run into “storage ceilings” because of the hard limits imposed by the operating system or hardware migrating the in-memory data to disk or developing your new applications with MySQL Cluster's data on disk can be a very good solution.

Applications connecting to the MySQL Cluster using a MySQL Server or the NDB API gain the following benefits:

- Data independence, meaning that the application can be written without requiring knowledge of the physical storage of the data. Data is stored in a storage engine which handles all of the low level details such as data replication and automatic fail over.
- Network and distribution transparency, which means that the application program depends on neither the operational details of the network, nor the distribution of the data on the data nodes,
- Replication and partition transparency, meaning that applications can be written in the same way regardless of whether the data is replicated or not, and independent of how the data is partitioned.

In addition, using the NDB API gives the advantages of:

- Increased performance
- Reduced, predictable latency

Connecting through the MySQL Server provides:

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

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

4 Enhanced Scalability and Performance

As Telecommunications, Enterprise and Government organizations deliver new services to a host of new users and devices across both wired and wireless networks, previously unforeseen demands have been placed on underlying database platforms. For example, much higher update rates driven by presence and location based services are common, coupled with much larger database sizes driven by complex, composite on-line services. Availability and latency requirements become ever more stringent and the ability to scale instantly to handle massive demand is paramount.

This ability to deploy a database with extremely high throughput and low latency while also being able to start with a small, inexpensive system and then scale up is a key feature of MySQL Cluster and it remains a central focus for MySQL Cluster 7.0 & 7.1 releases.

4.1 Multi-Threaded Data Nodes

MySQL Cluster has always been highly effective at "scaling-out" – for example, you can start on small uni-processor COTS (Commodity Off The Shelf) Cluster hosts, adding nodes incrementally as your workload grows, all the while keeping acquisition costs ultra-low.

MySQL Cluster 7.0 adds a new option - "scaling-up" by allowing the data nodes to better exploit multiple threads, cores or CPUs in a single system. Each NDB data node is able to make effective use of up to 8 CPU cores/threads. Data is partitioned among a number of threads within the NDB process; in effect duplicating the Cluster architecture with the data node. This design maximizes the amount of work that can be done independently by threads and minimizes their communication.

This is achieved by allowing the Local Query Handler to run in a multi-threaded mode while other key activities (Transaction Coordination, replication, IO and transporter) are run in separate threads. Each of the LQH threads is responsible for one primary sub-partition (a fraction of the partition that a particular data node is responsible for) and one replica sub-partition (assuming that NoOfReplicas is set to 2). The Transaction Coordinator for an NDB data node is responsible for routing requests to the correct LQH thread.

Figure 5 illustrates this for an 8 core system where 4 threads are used for running instances of the Local Query Handler (LQH).

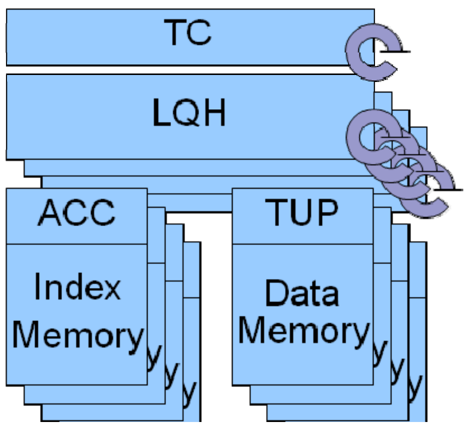


Figure 5 Multi-Threaded Data Node

The Transaction Coordinator handles co-ordination of transactions and timeouts; it serves as the interface to the NDB API for indexes and scan operations.

The Access Manager handles hash indexes of primary keys providing speedy access to the records.

The Tuple Manager handles storage of tuples (records) and contains the filtering engine used to filter out records and attributes.

The 4 LQH threads are in turn allocated work by the single Transaction Coordinator (TC) thread. The Access Manager (ACC) and the Tuple Manager (TUP) are run within each of the LQH threads.

4.1.1 Benefits

Multi-threaded/core/CPU systems have become common-place, even a low-end PC now comes with 2-4 cores and higher end systems have many more. In order to take advantage of those resources in earlier releases, the system had to be set up to run more than one NDB data node instance on the same host which has a number of effects:

- Each of the instances comes with a memory overhead which consumes that valuable resource, leaving less for storing application data. Other inefficiencies come from more process scheduling and open files and sockets which impact performance.
- Extra complexity from having to manage an increased number of data nodes
- There is a limit to the number of data nodes supported in a Cluster (currently 48) and this applies to the total number of data node instances rather than the number of physical hosts that they run on.

If building a new system then this feature means that less hardware will be needed to deliver the required performance (and possibly capacity). This represents an obvious saving in hardware purchase costs and data center space and power consumption but it also reduces software license and support fees as well as maintenance complexity/cost leading to a significant reduction in TCO.

In an existing system, you can use this extra bandwidth to store more data without buying new hardware; additionally you can remove the workaround of running multiple NDB data node instances on the same machine.

This feature can deliver 4x[1] more operations per second than previous Cluster releases on the latest multi-core hardware. Users can support larger and more complex workloads with 4x fewer data nodes and less hardware resource. This enables users to decrease capital hardware costs, reducing

[1] http://www.mysql.com/why-mysql/benchmarks/mysql-cluster/

energy consumption of the Cluster by 3x[2] and rack space by 4x[3], while simplifying on-going operations.

4.1.2 Configuring NDB Data Nodes to be Multi-Threaded

ndbmtd is a multi-threaded version of ndbd, the process that is used to handle all the data in tables using the NDB data node. ndbmtd is intended for use on host computers having multiple CPU cores/threads.

In almost all ways, ndbmtd functions in the same way as ndbd; and the application therefore does not need to be aware of the change. Command-line options and configuration parameters used with ndbd also apply to ndbmtd. ndbmtd is also file system-compatible with ndbd. In other words, a data node running ndbd can be stopped, the binary replaced with ndbmtd, and then restarted without any loss of data. This all makes it extremely simple for developers or administrators to switch to the multi-threaded version.

Using ndbmtd differs from using ndbd in two key respects:

- You must set an appropriate value for the MaxNoOfExecutionThreads configuration parameter in the config.ini file. If you do not do so, ndbmtd runs in single-threaded mode — that is, it behaves like ndbd.
- Trace files are generated for critical errors in ndbmtd processes in a somewhat different fashion from how these are generated by ndbd failures.

Number of execution threads. The MaxNoOfExecutionThreads configuration parameter is used to determine the number of execution threads spawned by ndbmtd. Although this parameter is set in [ndbd] or [ndbd default] sections of the config.ini file, it is exclusive to ndbmtd and does not apply to ndbd. This parameter takes an integer value from 2 to 8 inclusive. Generally, you should set this to the number of CPU cores/threads on the data node host, as shown in the following table:

Number of cores	Recommended MaxNoOfExecutionThreads Value
2	2
4	4
8+	8

Clearly, the performance improvement realized by this capability will be partly dependent on the application. As an extreme example, if there is only a single application instance which is single threaded, sending simple read or write transactions to the Cluster then no amount of parallelism in the data nodes can speed it up. So while there is no requirement to change the application, some re-engineering may help to achieve the best improvement.

4.2 On-Line Add Node

One of the most compelling attributes of MySQL Cluster is the ability to start with a small, low cost Cluster and then scale-out by adding more blades or servers as the capacity and/or performance demands grow over time. This growth could be driven by more users of the application, new

[2] Based on 4 x Sun Fire x4150 servers, equipped each equipped with 1 x dual core Intel Xeon 5260 3.33GHz processor (8 cores total) consuming 236 watts each, or 944 watts total versus 1 x Sun Fire x4150 Server, equipped with 2 x quad core Intel Xeon 5460 3.16GHz processors (8 cores total), consuming 308 watts. Each server equipped with 4 x 4GB DIMMs, 2 x 146GB PCI-E cards, running at 50% utilization. Power consumption estimated from x4150 power calculator on January 27th 2009: http://www.sun.com/servers/x64/x4150/calc/

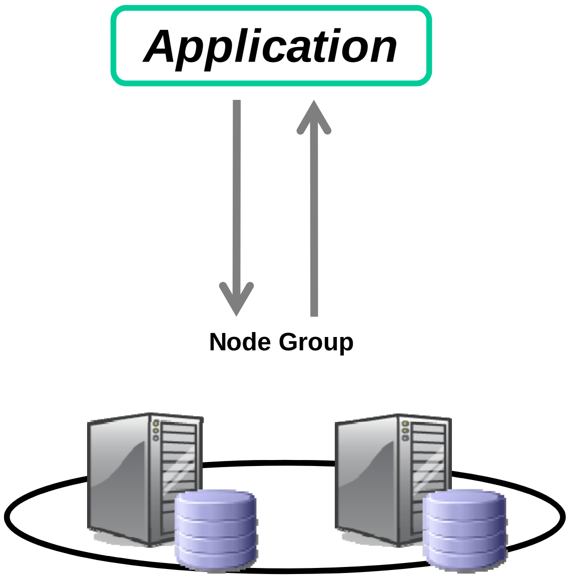
[3] Sun Fire x4150 = 1 Rack Unit high. 4 x single processor servers requiring 4 RU vs 1 x dual processor server requiring 1RU of rack space

applications sharing the same Cluster database or extra functionality/complexity being added to the original application.

Prior to MySQL Cluster 7.0, it was not possible to add node groups to an existing, in-service Cluster (in reality, there existed workarounds if you used Geographic Redundancy but they added complexity to the process and not every Cluster deployment has a redundant site).

Beginning with MySQL Cluster 7.0, it is possible to add new node groups (and thus new data nodes) to a running MySQL Cluster without shutting down and reloading the cluster. Additionally, a new capability has been added to repartition the data (moving a subset of the data from the existing node groups to the new one).

This section will step through an example of extending a single node group Cluster by adding a second node group and repartitioning the data across the 2 node groups. Figure 6 shows the original system.



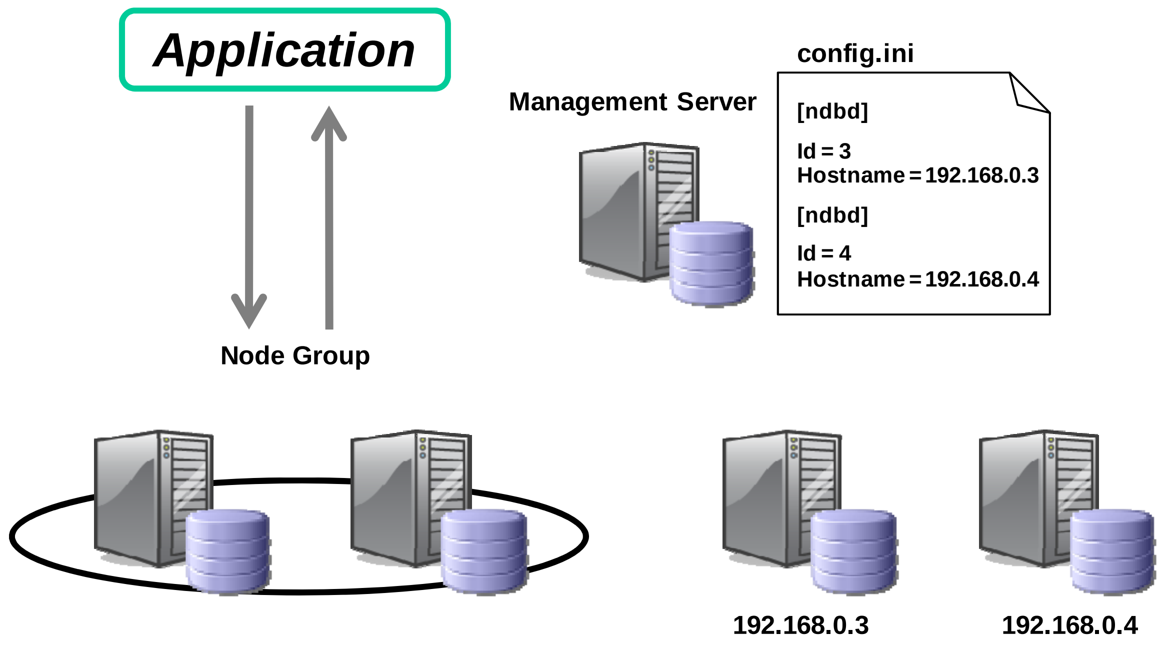
"author" Table

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

Figure 6 Initial Configuration - Single Node Group

This example assumes that the 2 servers making up the new node group have already been commissioned and focuses on the steps required to "bring them into the Cluster".

Step 1: Edit config.ini on all of the management servers as shown in Figure 7.



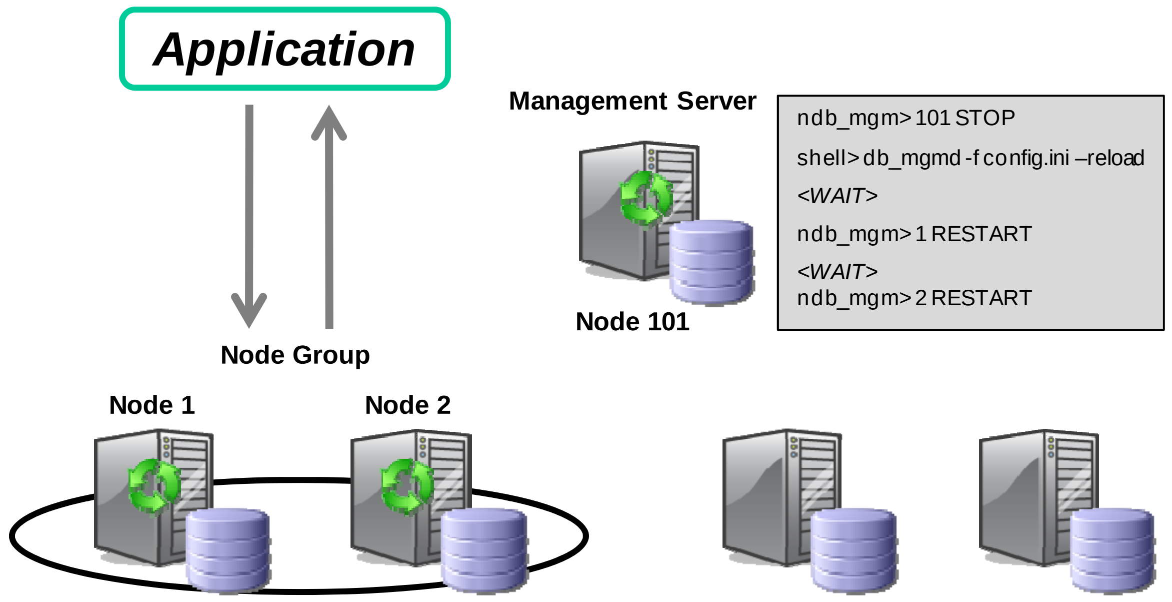
config.ini

```
[ndbd]
Id = 3
Hostname = 192.168.0.3
[ndbd]
Id = 4
Hostname = 192.168.0.4
```

Figure 7 Update config.ini on all management servers

Step 2: Restart the management server(s), existing data nodes and MySQL Servers.

Figure 8 illustrates how to restart the management server and existing data nodes. Note that you will be made to wait for each restart to be reported as complete before restarting the next node so that service is not interrupted.



```
ndb_mgm> 101 STOP
shell> db_mgmd -f config.ini –reload
<WAIT>
ndb_mgm> 1 RESTART
<WAIT>
ndb_mgm> 2 RESTART
```

Figure 8 Rolling Restart of Management and Data Nodes

In addition, all of the MySQL Server nodes that access the cluster should be restarted. That would be done by issuing the following commands (exact syntax would be dependent on how much information has been included in the my.cnf file) on each server:

```
  shell> mysqladmin -uroot -ppassword shutdown
  shell> mysqld_safe
```

If creating a new Cluster with the expectation that it will grow in the future then you can include data for the (at that time) non-existent data nodes in the config.ini file (refer to the MySQL Cluster Reference Guide for details). As there is no service impact from the rolling restart, you may decide to not worry about this advance preparation.

Step 3: Create the new node group

You are now ready to start the 2 new data nodes and then create the node group from them and so include them in the Cluster as shown in Figure 9. Note that in this case, there is no need to wait for Node 3 to start before starting Node 4 but you must wait for both to complete before creating the node group.

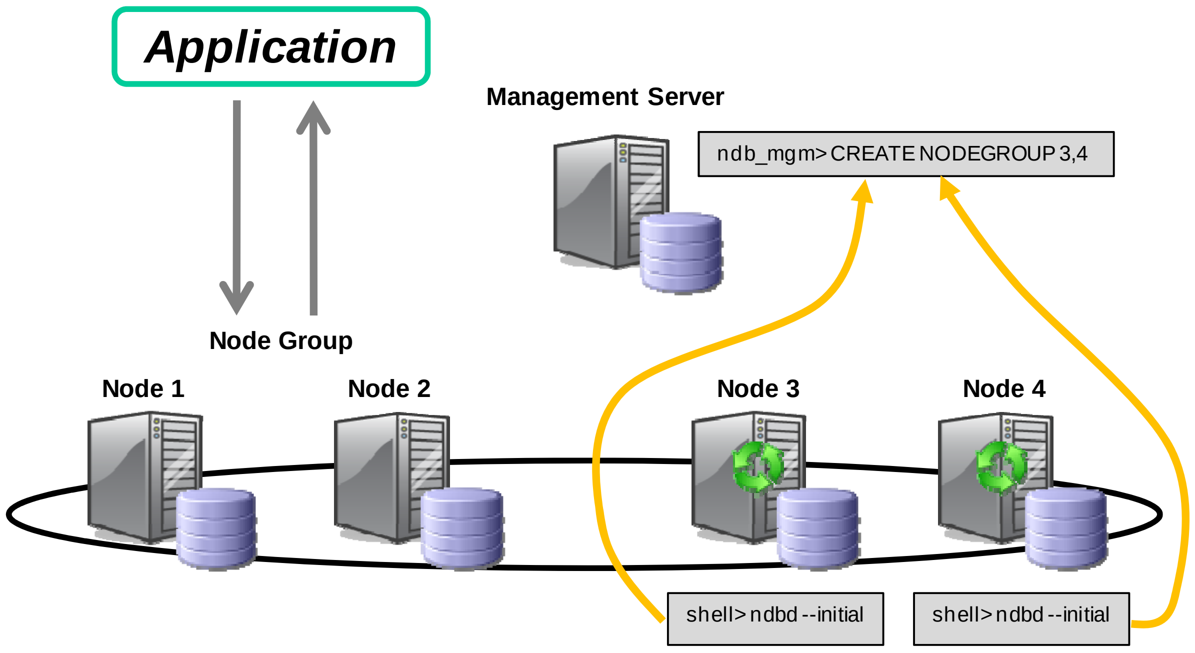
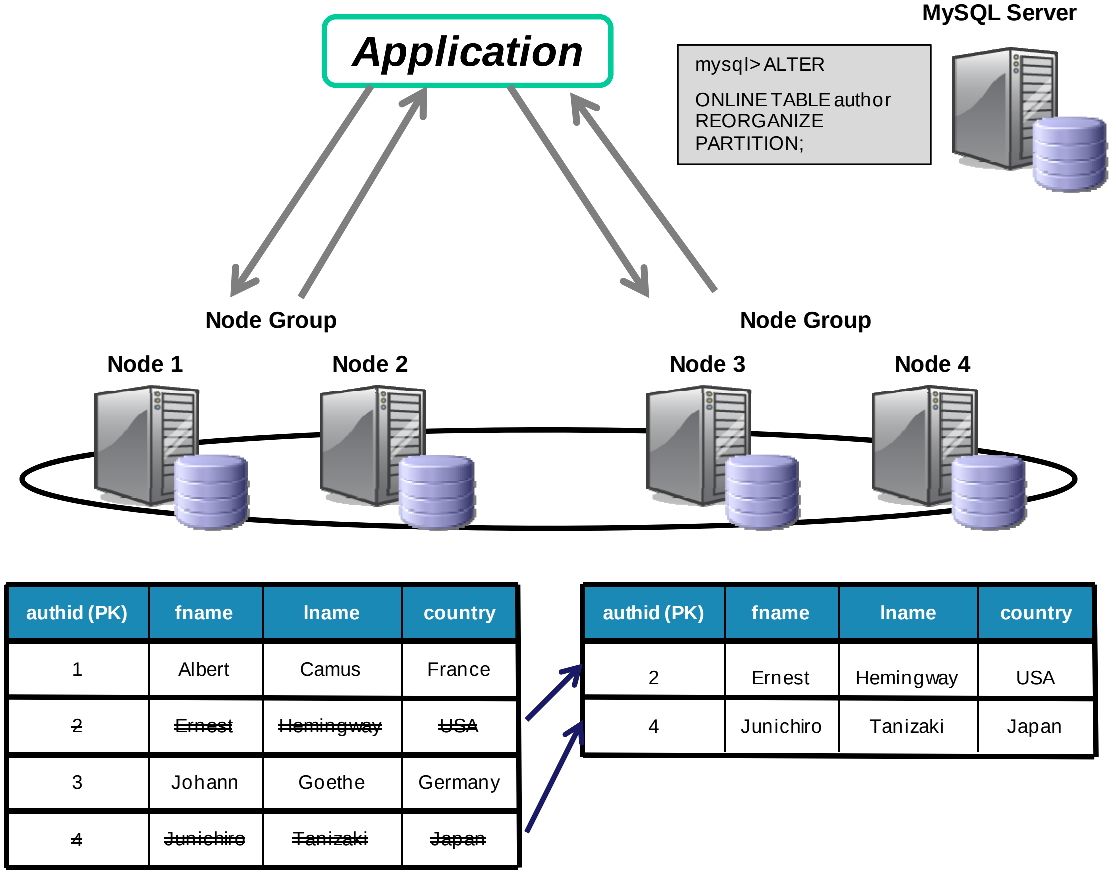


Figure 9 Start the new data nodes and add as node group in Cluster

Step 4: Repartition Data

At this point the new data nodes are part of the Cluster but all of the data is still held in the original node group (Node 1 and Node 2). Note that once the new nodes are added as part of a new node group, new tables will automatically be partitioned across all nodes.

Figure 10 illustrates the repartitioning of the table data (disk or memory) when the command is issued from a MySQL Server. Note that the command should be repeated for each table that you want repartitioning.



```
mysql> ALTER
ONLINE TABLE author
REORGANIZE
PARTITION;
```

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

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

Figure 10 Data Repartitioned across all available node groups

It should be noted that the ONLINE REORGANIZE and the add CREATENODEGROUP operations are transactional and so a node or cluster failure while they are running would not corrupt the database.

At this point, memory for the partitioned out rows has not been recovered within the original node group – this can be remedied by issuing an OPTIMIZE TABLE command from a MySQL Server.

4.3 Multi-Threaded Disk Data File Access

The existing support for disk based table data allows much larger capacity databases to be created. MySQL Cluster 7.0 modifies the file access architecture to improve the performance for working with this data.

All access to the files used by MySQL Cluster is through I/O threads and in previous versions there was a 1-to-1 mapping between open files and I/O threads. If a file is accessed frequently then the single I/O thread could become a bottleneck and reduce system throughput. One workaround to reduce this contention was to break these large files into smaller ones but that created more administrative overhead.

MySQL Cluster 7.0 introduces a pool of I/O threads where each thread is not tied to an open file. A file that is accessed very frequently could have a number of I/O threads allocated to it at a particular point in time which increases the throughput as shown in Figure 11**.**

The main benefit of this enhancement is increased I/O for heavily used files which improves the throughput and response times for disk-based table data.

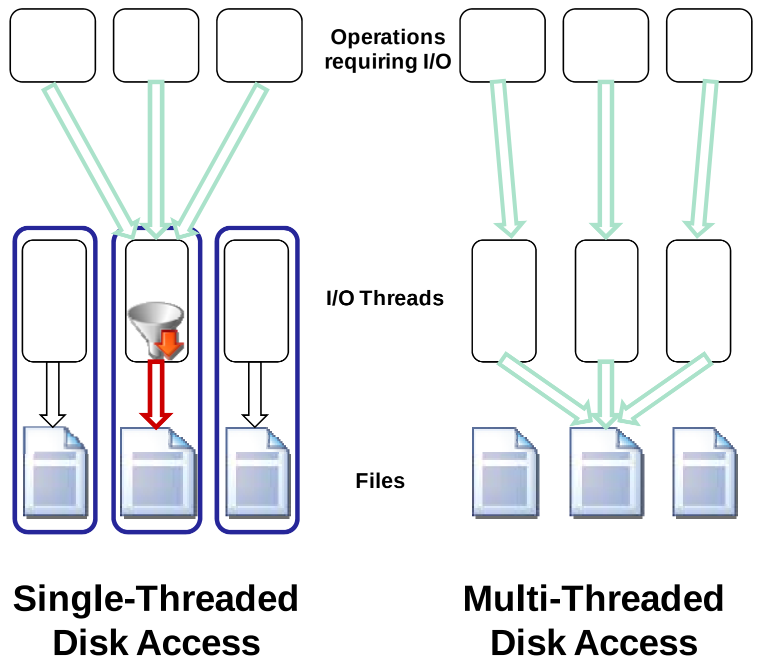


Figure 11 Multi-threaded disk data access

As I/O threads are no longer dedicated to all open files (whether they are in the process of being accessed or not), it is now possible to reduce the overall number of I/O threads in the system which reduces memory overhead.

It should be noted that some files continue to use single-threaded disk access (notably the redo log). This makes sense where data is streamed in and the latency of each access is critical.

4.4 Improved Large Record Handling

The performance of the Cluster can be limited by the messaging that passes between the client and the NDB data node and between the data nodes themselves. As Clusters grow, more and more traffic must pass over these network connections and in many cases, the performance of these network connections can limit the overall performance of the Cluster. In addition, the messaging consumes CPU resources on both sides of the connections and so reducing the size or number of messages can free up that CPU time for other work.

MySQL Cluster 7.0 introduces changes that optimize the use of the network connections by addressing the size and number of messages. These changes can increase the performance of the Cluster (and hence the application) and in some cases remove the need to migrate to a higher performance/more costly network infrastructure.

5 Windows Platform

As enterprise and consumer dependence on communications networks and services grows, high availability solutions continue to gain widespread market adoption, with demand to support an ever increasing range of developer environments and deployment platforms. As the industry's leading open source, high availability database for real-time, mission critical applications, MySQL Cluster has always been built around open standards, available on a wide range of COTS platforms. MySQL Cluster 7.0 extended this choice with support for the Microsoft Windows operating system.

MySQL Cluster had previously supported Windows-based clients through MySQL Server Connectors. With the release of MySQL Cluster 7.0, the Cluster itself could now run on Microsoft Windows. This support allows broader developer choice, enabling them to use their favorite platforms and tools to quickly and simply develop compelling high-availability database applications. This capability also extends the range of deployment options for DBAs and Systems Administrators to consider Windows platforms, in addition to Linux and UNIX environments.

In MYSQL Cluster 7.0, Windows is supported for development systems but not for live, deployed system. MySQL Cluster 7.1 extends this support by allowing MySQL Cluster nodes to run on Windows in live deployments.

The list of supported platforms is published at http://www.mysql.com/support/supportedplatforms/cluster.html and the binary and source downloads are available from http://dev.mysql.com/downloads/cluster/

6 Simplified Cluster Monitoring and Management

We often focus on the headline features of the database – capacity, performance, transaction interfaces and high availability but it is important to keep in mind the other functionality which makes the database more usable in real-world situations.

MySQL Cluster 7.0 & 7.1 enhance some of these key areas.

6.1 ndbinfo Data Node Statistics

ndbinfo is a read-only database presenting information on what is happening within the cluster – in particular, within the data nodes. This database contains a number of views, each providing data about MySQL Cluster node status, resource usage and operations.

Because this information is presented through regular SQL views, there is no special syntax required to access the data either in its raw form ("SELECT * FROM ndbinfo.counters") or manipulating it by filtering on rows and columns or combining (joining) it with data held in other tables. This also makes it simple to integrate this information into existing monitoring tools (for example MySQL Enterprise Monitor).

The ndbinfo views include the node-id for the data node that each piece of data is associated with and in some cases the values will be different for each data node.

6.1.1 ndbinfo.counters

This view presents information on the number of events since the last restart for each data node:

```
mysql> select * from ndbinfo.counters;
+---------+------------+----------------+------------+--------------+------+
| node_id | block_name | block_instance | counter_id | counter_name | val  |
+---------+------------+----------------+------------+--------------+------+
|       3 | DBLQH      |              1 |         10 | OPERATIONS   | 2069 |
|       3 | DBLQH      |              2 |         10 | OPERATIONS   |   28 |
|       4 | DBLQH      |              1 |         10 | OPERATIONS   | 2066 |
|        4 | DBLQH      |              2 |          10 | OPERATIONS    |   27 |
|        3 | DBTC       |              0 |           1 | ATTRINFO      |  140 |
|        3 | DBTC       |              0 |           2 | TRANSACTIONS  |   19 |
|        3 | DBTC       |              0 |           3 | COMMITS       |   19 |
|        3 | DBTC       |              0 |           4 | READS         |   19 |
|        3 | DBTC       |              0 |           5 | SIMPLE_READS  |    0 |
|        3 | DBTC       |              0 |           6 | WRITES        |    0 |
|        3 | DBTC       |              0 |           7 | ABORTS        |    0 |
|        3 | DBTC       |              0 |           8 | TABLE_SCANS   |    0 |
|        3 | DBTC       |              0 |           9 | RANGE_SCANS   |    0 |
|        4 | DBTC       |              0 |           1 | ATTRINFO      |    2 |
|        4 | DBTC       |              0 |           2 | TRANSACTIONS  |    1 |
|        4 | DBTC       |              0 |           3 | COMMITS       |    1 |
|        4 | DBTC       |              0 |           4 | READS         |    1 |
|        4 | DBTC       |              0 |           5 | SIMPLE_READS  |    0 |
|        4 | DBTC       |              0 |           6 | WRITES        |    0 |
|        4 | DBTC       |              0 |           7 | ABORTS        |    0 |
|        4 | DBTC       |              0 |           8 | TABLE_SCANS   |    1 |
|        4 | DBTC       |              0 |           9 | RANGE_SCANS   |    0 |
+---------+------------+----------------+------------+--------------+------+
```

The block_name refers to the software component that the counter is associated with – this gives some extra context to the information:

- **DBLQH**: Local, low-level query handler block, which manages data and transactions local to the cluster's data nodes, and acts as a coordinator of 2-phase commits.
- **DBTC**: Handles distributed transactions and other data operations on a global level

The block_instance field is used to distinguish between the different Local Query Handles threads when using the multi-threaded data node (refer to section 4.1).

The counter_name indicates what events are being counted and the val field provides the current count (since the last restart for the node).

This information can be used to understand how an application's SQL queries (or NDB API method calls) are mapped into data node operations and can be invaluable when tuning the database and application.

6.1.2 ndbinfo.logbuffers

This view presents information on the size of the redo and undo log buffers as well as how much of that space is being used:

```
mysql> select * from ndbinfo.logbuffers;
+---------+----------+--------+----------+----------+--------+
| node_id | log_type | log_id | log_part | total    | used   |
+---------+----------+--------+----------+----------+--------+
|       3 | REDO     |      0 |        1 | 67108864 | 131072 |
|       3 | REDO     |      0 |        2 | 67108864 | 131072 |
|       3 | DD-UNDO  |      4 |        0 |  2096128 |      0 |
|       4 | REDO     |      0 |        1 | 67108864 | 131072 |
|       4 | REDO     |      0 |        2 | 67108864 | 131072 |
|       4 | DD-UNDO  |      4 |        0 |  2096128 |      0 |
+---------+----------+--------+----------+----------+--------+
```

This information can be used to identify when the buffers are almost full so that the administrator can increase their size – preempting potential issues. If the redo log buffers are exhausted then applications will see 1221 "REDO log buffers overloaded" errors. In extreme cases, if the undo log buffers fill too quickly then the database may be halted.

When using the multi-threaded data node (ndbmtd), the log_part column is used to distinguish between different LQH threads.
The sizes of the redo buffer can be increased using the RedoBuffer parameter. The size of the undo buffer is specified in the undo_buffer_size attribute when creating the log file.

6.1.3 ndbinfo.logspaces

This view provides information on the disk space that has been configured for the log spaces for redo and undo logs:

```
mysql> select * from logspaces;
+---------+----------+--------+----------+-----------+--------+
| node_id | log_type | log_id | log_part | total     | used   |
+---------+----------+--------+----------+-----------+--------+
|       3 | REDO     |      0 |        0 | 536870912 |      0 |
|       3 | REDO     |      0 |        1 | 536870912 |      0 |
|       3 | REDO     |      0 |        2 | 536870912 |      0 |
|       3 | REDO     |      0 |        3 | 536870912 |      0 |
|       3 | DD-UNDO  |      4 |        0 |  78643200 | 169408 |
|       4 | REDO     |      0 |        0 | 536870912 |      0 |
|       4 | REDO     |      0 |        1 | 536870912 |      0 |
|       4 | REDO     |      0 |        2 | 536870912 |      0 |
|       4 | REDO     |      0 |        3 | 536870912 |      0 |
|       4 | DD-UNDO  |      4 |        0 |  78643200 | 169408 |
+---------+----------+--------+----------+-----------+--------+
```

For the redo log space, there is 1 row for each of the 4 file sets for each data node and the total colum is equal to the NoOfFragmentLogFiles configuration parameter multiplied by the FragmentLogFileSize parameter while the used column is the amount actually used. If the files fill up before a local checkpoint can complete then error code 410 (Out of log file space temporarily) will be observed. That error can now be avoided by increasing NoOfFragmentLogFiles and/or FragmentLogFileSize if used approaches total.

For the undo log space, the total column represents the cumulative size of all of the undo log files assigned to the log group, as added using create/alter logfile group or the InitialLogFileGroup configuration parameter. Adding extra undo files if used approaches total can be done to avoid getting 1501 errors.

6.1.4 ndbinfo.memoryusage

This view compares the amount of memory and index used to the amount configured for each data node:

```
mysql> select * from memoryusage;
+---------+--------------+--------+------------+-----------+-------------+
| node_id | memory_type  | used   | used_pages | total     | total_pages |
+---------+--------------+--------+------------+-----------+-------------+
|       3 | Data memory  | 917504 |         28 | 104857600 |        3200 |
|       3 | Index memory | 221184 |         27 |  11010048 |        1344 |
|       4 | Data memory  | 917504 |         28 | 104857600 |        3200 |
|       4 | Index memory | 221184 |         27 |  11010048 |        1344 |
+---------+--------------+--------+------------+-----------+-------------+
```

For data memory, the total column represents the amount of configured memory (in bytes) for the data node – set using the DataMemory configuration parameter and used represents the amount currently in use by the Cluster tables.

For index memory, the total column represents the amount of configured index memory (in bytes) for the data node – set using the IndexMemory configuration parameter and used represents the amount currently in use by the Cluster tables.

The same information is also provided using memory pages rather than bytes.

To avoid exhausting the memory, monitor this table and if used approaches max then either:

- Delete obsolete table rows and run OPTMIMIZE TABLE
- Increase the value of IndexMemory and/or DataMemory
- Consider whether it is practical to alter some tables or columns to be stored on disk rather than in memory

6.1.5 ndbinfo.nodes

This view shows status information for every data node in the cluster:
mysql> select * from ndbinfo.nodes;

```
mysql> select * from ndbinfo.nodes;
+---------+--------+---------+-------------+
| node_id | uptime | status  | start_phase |
+---------+--------+---------+-------------+
|       3 |  16548 | STARTED |           0 |
|       4 |     17 | STARTED |           0 |
+---------+--------+---------+-------------+
```

The uptime column shows the time in seconds since the data node was started or last restarted.

The status is one of NOTHING, CMVMI, STARTING, STARTED, SINGLEUSER, STOPPING_1, STOPPING_2, STOPPING_3, or STOPPING_4. If the status is set to STARTING then the start phase is also displayed.

6.1.6 ndbinfo.transporters

This is a view showing the status of the connection between each data node and each of the other nodes in the cluster (data nodes, management nodes, MySQL Server nodes and any NDB API applications):

```
mysql> select * from transporters;
+---------+----------------+--------------+
| node_id | remote_node_id | status       |
+---------+----------------+--------------+
|       3 |              1 | CONNECTED    |
|       3 |              3 | DISCONNECTED |
|       3 |              4 | CONNECTED    |
|       3 |            102 | CONNECTED    |
|       3 |            103 | CONNECTING   |
|       3 |            104 | CONNECTING   |
|       4 |              1 | CONNECTED    |
|       4 |              3 | CONNECTED    |
|       4 |              4 | DISCONNECTED |
|       4 |            102 | CONNECTED    |
|       4 |            103 | CONNECTING   |
|       4 |            104 | CONNECTING   |
+---------+----------------+--------------+
```

6.2 MySQL Cluster Manager

MySQL Cluster Manager provides the ability to control the entire cluster as a single entity, while also supporting very granular control down to individual processes within the cluster itself. Administrators are able to create and delete entire clusters, and to start, stop and restart the cluster with a single command. As a result, administrators no longer need to manually restart each data node in turn, in the correct sequence, or to create custom scripts to automate the process.

MySQL Cluster Manager automates on-line management operations, including the upgrade, downgrade and reconfiguration of running clusters, without interrupting applications or clients accessing the database. Administrators no longer need to manually edit configuration files and distribute them to other cluster nodes, or to determine if rolling restarts are required. MySQL Cluster Manager handles all of these tasks, thereby enforcing best practices and making on-line operations significantly simpler, faster and less error-prone.

MySQL Cluster Manager is able to monitor cluster health at both an Operating System and per-process level by automatically polling each node in the cluster. It can detect if a process or server host is alive, dead or has hung, allowing for faster problem detection, resolution and recovery.

To deliver 99.999% availability, MySQL Cluster has the capability to self-heal from failures by automatically restarting failed Data Nodes, without manual intervention. MySQL Cluster Manager extends this functionality by also monitoring and automatically recovering SQL and Management Nodes. This supports a more seamless and complete self-healing of the Cluster to fully restore operations and capacity to applications.

MySQL Cluster Manager is available as part of the commercial MySQL Cluster Carrier Grade Edition only.

For more information on MySQL Cluster Manager, refer to its dedicated white paper at http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_manager.php or browse the material at http://www.mysql.com/products/database/cluster/mcm/ A demonstration of installing, running and using MySQL Cluster Manager can be viewed at http://www.mysql.com/products/database/cluster/mcm/cluster_install_demo.html

6.3 Snapshot Backup Options for MySQL Cluster

MySQL Cluster supports on-line data back-up, ensuring service interruptions are avoided during this critical database maintenance task. The original Cluster back-up routine captured the database state at the time the back-up operation ended. If you needed to synchronize this backup with other activities (for example, backing up other, external databases), you could only estimate which point-in-time the Cluster backup would capture whereas it would be desirable to decide when that point-in-time should be. Using the new Point-in-Time Cluster Backup feature of MySQL Cluster 7.0, the database state can now also be captured when the back-up operation starts. This ensures users can capture a consistent state of their databases at any one time.

Figure 12 shows how the backup works if you choose to use the original Cluster backup strategy. If the database is updated while it is being copied to disk, the updated fields will be in an indeterminate state on disk and so a redo log is built up during the backup and then stored alongside the backup file.

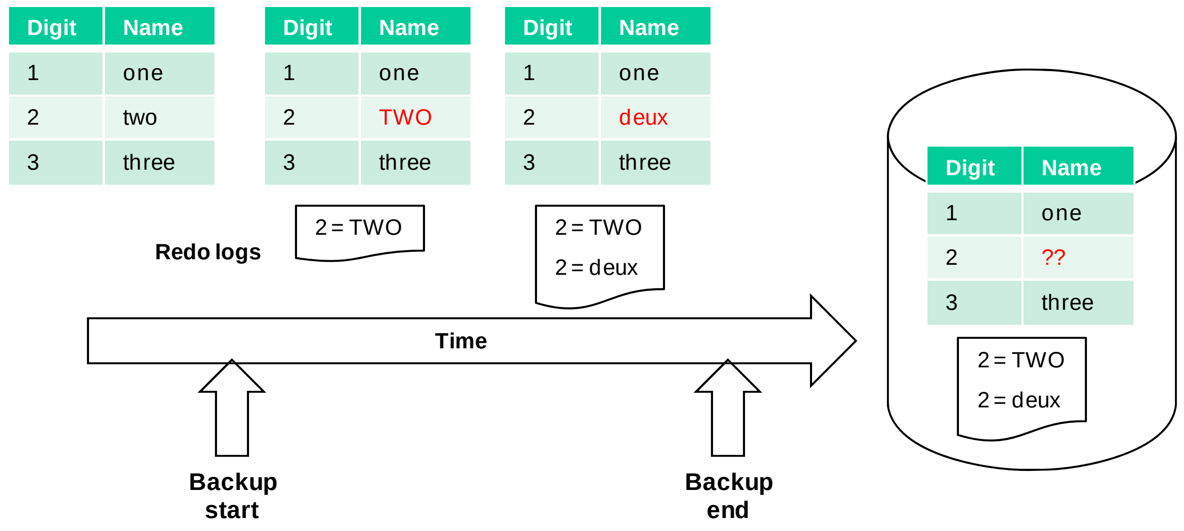


Figure 12 Backup aligned with end

Figure 13 shows the restore process. The backup file is read from disk into memory and then the updates in the redo log are applied from start to end. The result is that at the completion of the restore, the database looks as it did at the end of the backup.

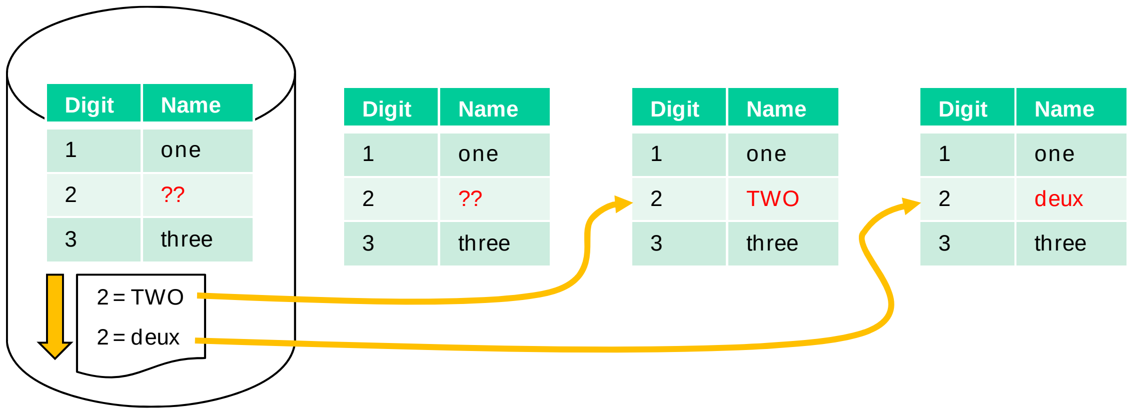


Figure 13 Restore Aligned with end of backup

Figure 14 shows how the backup process differs if you choose to have the backup match the database at the start of the backup rather than the end. Rather than writing changes to a redo log, the reverse of the changes are written to an undo log.

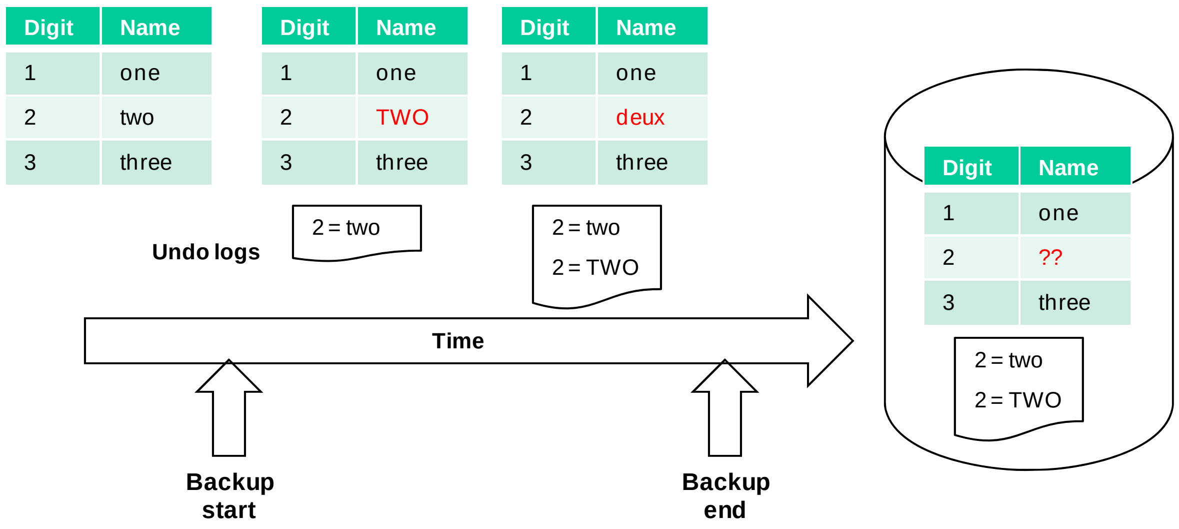


Figure 14 Backup aligned with start

As shown in Figure 15, when restoring from the backup, the undo log is applied from the end of the file backwards. The effect is that following the restore, the database looks as it did at the start of the backup.

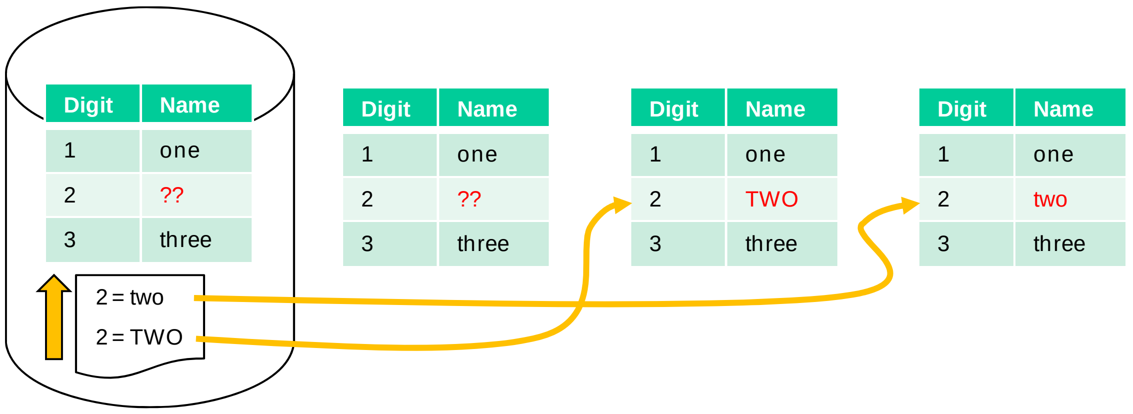


Figure 15 Restore Aligned with start of backup

MySQL Cluster 7.0 further adds to the restore flexibility by allowing the user to exercise more fine-grained control when restoring a MySQL Cluster from backup using ndb_restore. You can restore only specified tables or databases, or exclude specific tables or databases from being restored, using the new ndb_restore options --include-tables, --include-databases, --exclude-tables, and --exclude-databases.

6.4 Configuration Data Cached

Formerly, MySQL Cluster configuration was stateless - that is, configuration information was reloaded from the cluster's global configuration file (usually config.ini) each time ndb_mgmd was started. Beginning with MySQL Cluster 7.0, the cluster's configuration is cached internally and the global configuration file is no longer automatically re-read when the management server is restarted.

This has the benefit of maintaining a consistent view across the management servers rather than one autonomously picking up a new configuration when it restarts.

This behavior can be controlled via the three new management server options --configdir, --initial, and --reload.

6.5 Transactions for schema changes

Prior to MySQL Cluster 7.0, DDL (Data Definition Language) operations (such as CREATE TABLE or ALTER TABLE) were not protected from data node failures. In the event of a data node failure, MySQL Cluster 7 ensures such operations are now rolled back gracefully. Previously, if a data node failed while trying to perform a DDL operation, the MySQL Cluster data dictionary became locked and no further DDL statements could be executed without restarting the cluster. This enhancement makes it safer to make on-line changes to a live system.

7 Connectors

7.1 MySQL Cluster Connector for Java

Prior to MySQL Cluster 7.1, the choices for Java developers were somewhat limited. The only supported method was JDBC – either directly or through a 3rd party layer such as Java Persistence API (JPA) compliant middleware. If Java users wanted greater performance then they were responsible for writing their own Java Native Interface (JNI) layer sitting between their (Java) application and the (C++) NDB API.

With the introduction of MySQL Cluster 7.1, the options are much wider as shown in Figure 16. Working from left to right the options are:

- Java application makes JDBC calls through JDBC Driver for MySQL (Connector/J). This is a technology that many Java developers are comfortable and it is very flexible. However it requires that the application developer maps from their Java objects to relational, SQL statements. In addition, performance can be compromised as Connector/J accesses the data through a MySQL Server rather than directly accessing the data nodes through the NDB API.
- Applications can shift responsibility for the Object-Relational-Mapping (ORM) to a 3rd party JPA solution (for example, Hibernate, Toplink or OpenJPA). While this frees the Java developer from having to work with a relational data model, performance can still be limited if the JPA layer is having to access the data via JDBC.
- MySQL Cluster 7.1 introduces a plug-in for OpenJPA which allows most JPA operations to be performed through the NDB API (via a new "ClusterJ" layer) with the remainder using JDBC. This gives the Java developer the luxury of working purely with objects while still getting much of the performance benefits of using the NDB API.

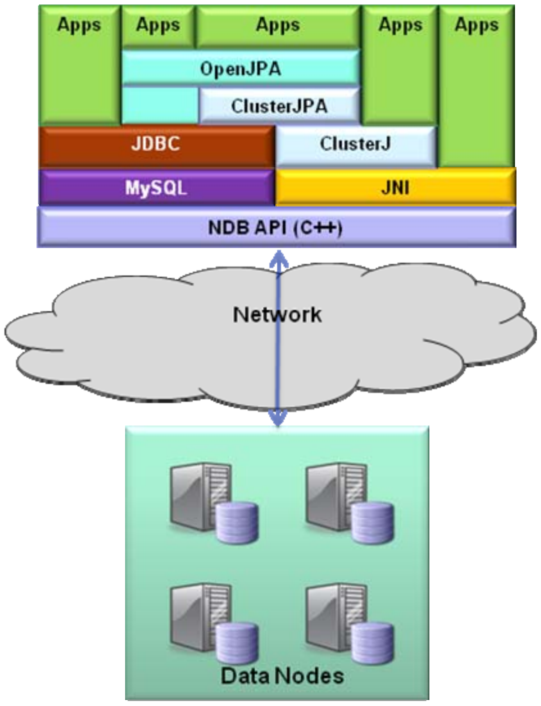


Figure 16 Options for Java Applications

- Applications developers can choose to bypass the JPA layer and instead go directly to the ClusterJ layer. This introduces some limitations but may be appropriate for developers wanting to get the best possible performance and/or want to avoid introducing additional 3rd party components (OpenJPA). ClusterJ is introduced by MySQL Cluster 7.1.

Finally, Java application developers still have the option to implement their own wrapper (typically using JNI) to act as the Java/C++ mediator.

For more details on ClusterJ and ClusterJPA, please refer to `http://www.mysql.com/why-mysql/whitepapers/mysql_wp_cluster_connector_for_java.php`

7.2 High Availability Directory Back-End

MySQL Cluster has been widely deployed for subscriber / user databases within telecommunications and web-based networks. Extending this capability, MySQL Cluster Carrier Grade Edition 7.0 can serve as the back-end data store for LDAP directory servers, allowing users to preserve and enhance their existing investments in LDAP technology. It allows organizations to embark on initiatives that fully exploit user and network data that is currently distributed across legacy applications and networks. In order to deploy a range of next generation, highly personalized services delivered over the network; operators need to expose subscriber and network data in a standardized way. User profiles are becoming richer as they capture network preference and media objects alongside traditional customer contact and service entitlement data. At the same time security and auditing requirements force data to be more transactional in nature. Using industry standard LDAP directories with MySQL Cluster serving as the directory data store, organizations can leverage standard LDAP interfaces for authentication and authorization of devices and subscribers with real-time performance, carrier-grade availability and a total solution that reduces cost, risk and complexity for large, transaction-intensive directory data sets.

Popular LDAP Directory OpenLDAP, provides a native NDB API driver for MySQL Cluster called back-ndb.

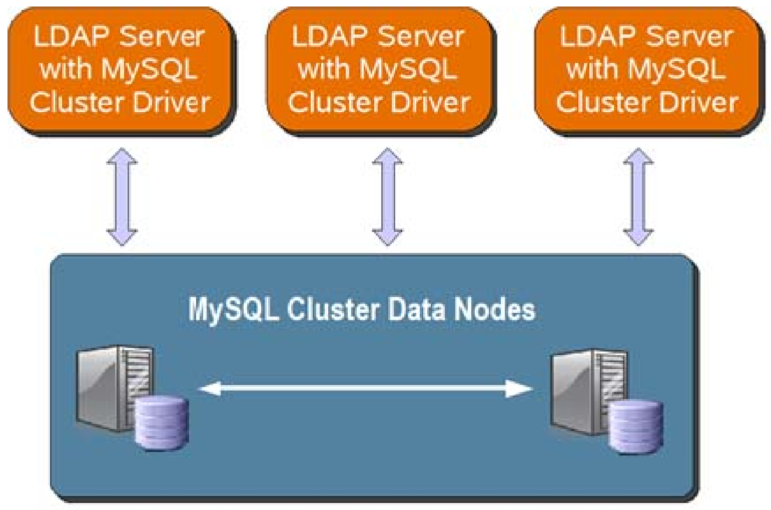

LDAP Server with MySQL Cluster Driver
LDAP Server with MySQL Cluster Driver
LDAP Server with MySQL Cluster Driver
MySQL Cluster Data Nodes

Figure 17 MySQL Cluster Used as Back-End Storage for LDAP Servers

Figure 18 shows how existing MySQL Cluster Deployments now also have the option of extending the data access methods to their data by adding a Directory Server as an additional front-end – enabling application access using LDAP rather than SQL or the NDP API directly.

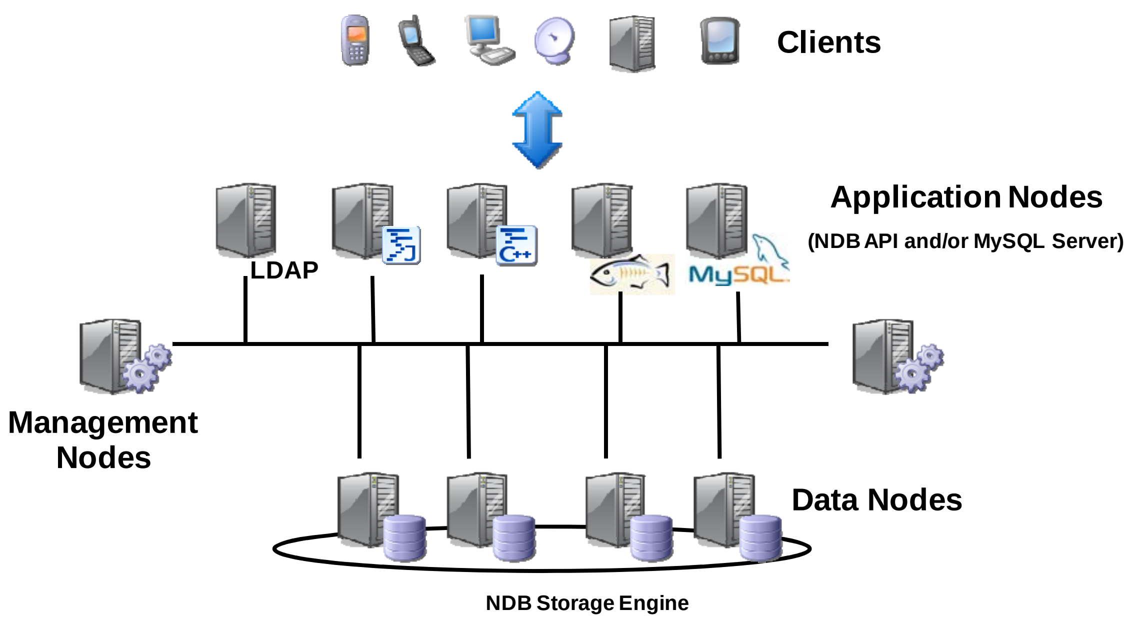


Figure 18 Access methods to data in the Cluster

8 Conclusion

In this paper we explored in detail some of the new features which have been introduced in MySQL Cluster 7.0 & 7.1:

- Multi-Threaded Data Nodes
- On-Line Add Node
- Multi-Threaded Disk Data File Access
- Improved Large Record Handling
- Windows Platform
- Point-in-Time Cluster Back-Up
- Configuration data cached
- Transactions for schema changes
- Carrier-Grade Directory Back-End
- Ndbinfo real-time reporting
- MySQL Cluster Connector for Java

For further information and the complete set of change logs for MySQL Cluster please refer to the online documentation available at www.mysql.com.

MySQL Cluster continues down a development path focused on delivering a dramatically lower TCO for a highly available real-time database and at the same time facilitating the ability to leverage a scale-out (and now also scale-up) methodology using commodity hardware and open source components.

9 Links and References

Below are links to additional high availability resources from MySQL.

MySQL Cluster Evaluation Guide:
http://www.mysql.com/why-mysql/white-papers/mysql_cluster_eval_guide.php

MySQL Cluster NDB 7.0 Reference Guide: http://dev.mysql.com/doc/#cluster

MySQL Cluster API Developer Guide: http://dev.mysql.com/doc/#cluster

MySQL Cluster Architecture Overview White Paper:
http://www.mysql.com/why-mysql/white-papers/cluster-technical.php

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

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