AgileLoad

White Paper

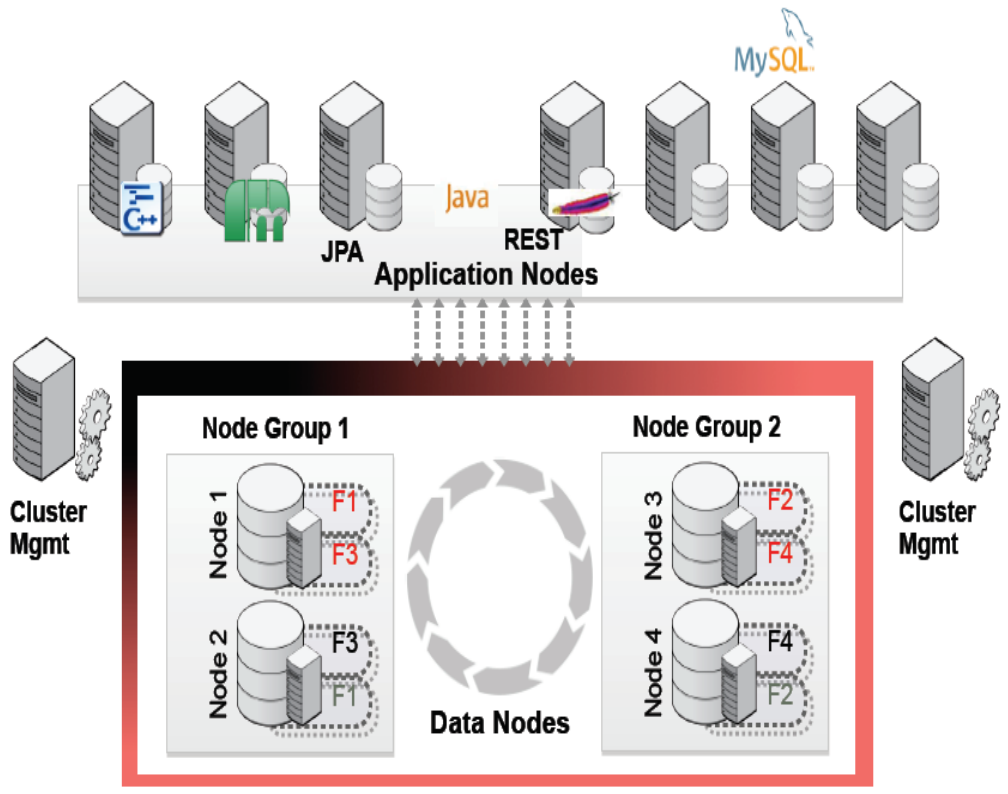
Optimizing the Performance Of MySQL Cluster

Table of Contents

Introduction and Background Information

MySQL Cluster can be enhanced to extend use cases into enterprise solutions including mobile payment systems and real time analytics.

MySQL Cluster is a real time, scalable and transactional database. It was actually designed as an embedded telecom database for applications requiring real time performance and carrier grade availability. But MySQL Cluster can be enhanced to extend use cases into enterprise solutions including mobile payment systems, real time analytics, seed streaming & analysis and content management. MySQL Cluster has the ability to scale horizontally to meet the needs of intensive workloads. Here is the architecture of MySQL Cluster:



The Cluster consists of 3 nodes: Data nodes, Application nodes and Management nodes.

- Data nodes usually manage the data access and storage.
- Application nodes provide a link to the data nodes from the application logic and many APIs are offered to the applications.
- Management nodes play a vital role in the cluster configuration and they also provide arbitration in case of network partitions.

This white paper explores most of the optimization methods for enhancing the performance of MySQL Cluster. With these methods, you can handle a variety of workload requirements.

Optimal Applications for MySQL Cluster

The Cluster can also leverage multiple and parallel SQL nodes, each node serving many connections.

MySQL Cluster can make sure that the updates performed by an application or SQL node are immediately available to every node accessing the cluster. The tables are partitioned across a bunch of commodity nodes and this enables the database to scale horizontally.

MYSQL Cluster can avoid the need to perform at the application layer by just partitioning the tables at the database layer. This can greatly simplify the development and maintenance of applications. In case of transactional workloads, MySQL Cluster can deliver great levels of write throughput. The cluster can also leverage multiple and parallel SQL nodes, each node serving many connections.

Optimal applications for MySQL Cluster include:

- Trading platforms / systems.
- Payment gateways.
- User profile management.
- Multiple online games.
- IMS services.
- DHCP for broadband access.
- VOIP and video conferencing.

Identifying the Performance Issues

It is always recommended to adapt repeat strategy in measuring the performance i.e. latency and the transaction throughput. It is also essential to consider certain changes to the database and its usability.

The performance should be measured before the implementation of changes and also after the introduction of each optimization technique.
In many instances, you may come across a specific requirement that has to

be met. So with an iterative procedure, you can track the progress towards your goal.

Besides measuring the application's overall performance, you are supposed to consider individual transactions. Some queries may take too long to execute.

If you are using SQL to query the database with access to MySQL Enterprise Monitor then you can make use of the Query Analyzer that can track all expensive transactions.

If you don't have access to Enterprise Monitor then MySQL's slow query log can definitely help you in identifying the lengthy transactions. You can just use *long_query_time* variable to mention how slow a query should be before including it in the query log. Here the value is in seconds and specifying '0' can cause all the queries to be logged.

As the changes made to the *long_query_time* variable do not show immediate effect on active connections, you need to drop the connections and then re-establish them.

MySQL Cluster gives all the details within the data node. This data is available through a virtual database called NDBINFO. For instance, NDBINFO presents information on Disk Page Buffer usage. The Disk Page Buffer is a cache on each data node that is used while operating disk based tables. As usual, the higher the hit rate the better the performance. Tuning this cache size can have a powerful effect on your system. You can access the data apparent for the Disk Page Buffer from the *mysql* command line.

Optimizing the Performance

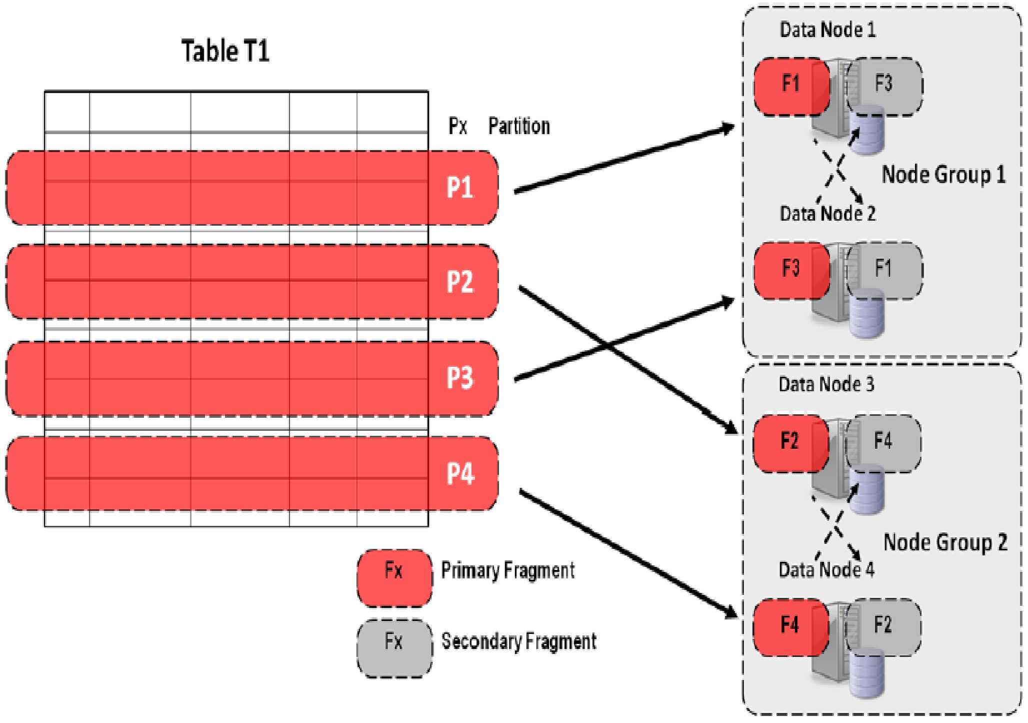
Access Patterns

You can just use long_query_time variable to mention how slow a query should be before including it in the query log.

As you expect maximum performance out of a MySQL Cluster deployment, it is important to know the database architecture. Here is the thing - the data for MySQL Cluster tables is not stored in the MySQL Server. It is actually partitioned across a pool of data nodes as shown in the following figure.

The table rows are divided into partitions. Each data node holds the primary fragment for one partition and a secondary fragment for the other.

If a query needs many network hops from the server to the data nodes or between the data nodes then performance may degrade and this affects scalability.



If the tables involved in the join are large then more time will be consumed for each trip to the data nodes.

So, achieving top performance from MySQL Cluster involves reducing the total number of network hops.

Partitioning is based on a hashing of the primary key, but that can be overridden to improve performance. Simple access patterns are key to building scalable and high performing solutions.

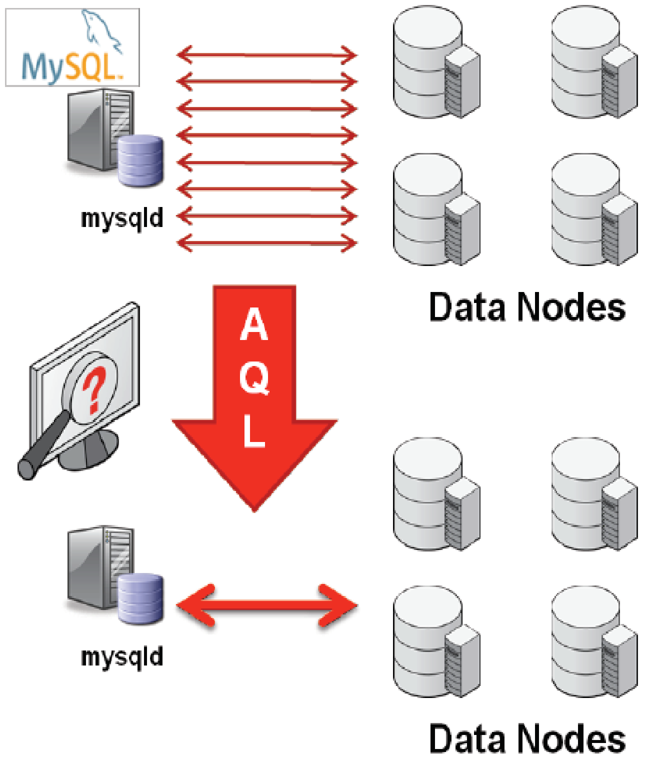
The number of network trips required is determined by the size of the results set from each stage and depth of the join. If the tables involved in the join are large then more time will be consumed for each trip to the data nodes.

The best performance can be seen while using Primary Key lookups that can be done in constant time irrespective of the database size and the number of data nodes.

Using AQL

By using AQL (Adaptive Query Localization), the performance improvements can be more dramatic.

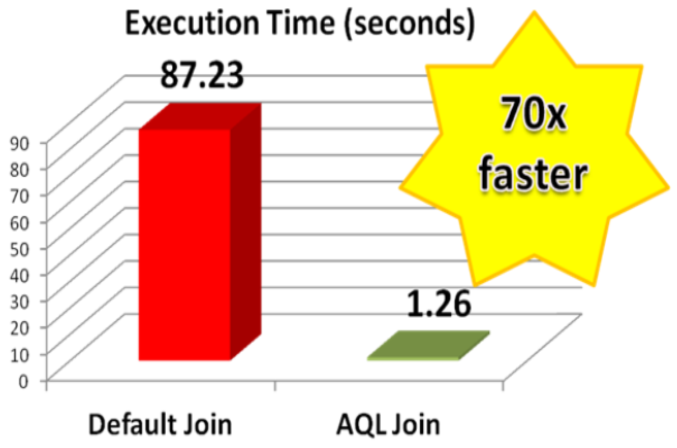
The AQL functionality dispatches the queries from the MySQL Server to all the data nodes. The query gets executed on local copies of the data. Later it returns the merged result set back to the MySQL Server. This process can eventually enhance the performance by minimizing the network trips.



AQL allows MySQL Cluster to effectively serve the use cases that involve intricate queries.

AQL allows MySQL Cluster to effectively serve the use cases that involve intricate queries.

To get better results, you can run OPTIMIZE TABLE <tab-name> on one of the MySQL Servers whenever you change the table schema, add or remove an index or perform significant changes.



A real world testing has shown good performance gains of 70X across a range of queries. For instance, an online content store management system has a complex query joining 11 tables. This query generally took over 87 seconds to complete. But with AQL join it just took around 1.26 seconds.

The AQL can be controlled by a global variable *ndb_join_pushdown* that is on by default. In order for a join to be able to utilize the AQL, it has to follow the below mentioned rules:

- Columns to be joined should use exactly the same data type, including the lengths of VARCHAR columns.
- It is not possible to push down the joins referencing BLOB or TEXT columns.
- Explicit locking is actually not supported. This can be fulfilled by using implicit row based locking of NDB storage engine.
- Joins referencing the tables that are explicitly partitioned by HASH or RANGE can't be pushed down.

Distribution Aware Applications

MySQL Cluster partitions the data by hashing the primary key. We can override the default behavior

While adding rows to a table that uses MySQL Cluster, each row is assigned to a partition. Each partition is mastered by a particular data node in the cluster. The best performance can be achieved when all the data required to serve a transaction is grasped within a single partition. This means that it can be served within a single data node instead of bouncing back and forth among multiple nodes that may increase the latency.

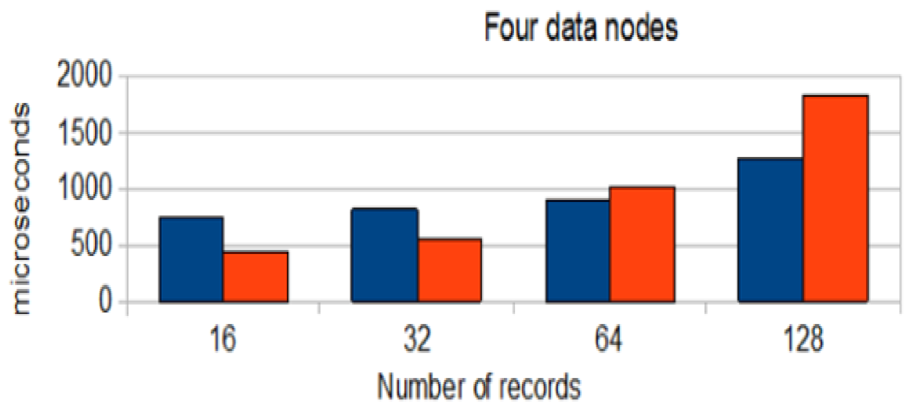
MySQL Cluster partitions the data by hashing the primary key by default. We can override the default behavior by just specifying the appropriate fields from the Primary Key that could to be fed into the hash algorithm.

You may also use partition pruning method. Partition pruning is the process of allowing an index scan to be performed by a single data node. The latency benefits from partition pruning are dependent on the number of data nodes and the size of the result set. You can achieve best improvements with more data nodes and less records to be retrieved.

The following figure shows the effect of partition pruning.

Partition pruning can reduce latency for smaller result sets and it can increase for larger result sets. In the figure below, partition pruning is shown in red bars.

In this manner, making your application distribution aware can be critical in enabling throughput to scale linearly as extra nodes are added.



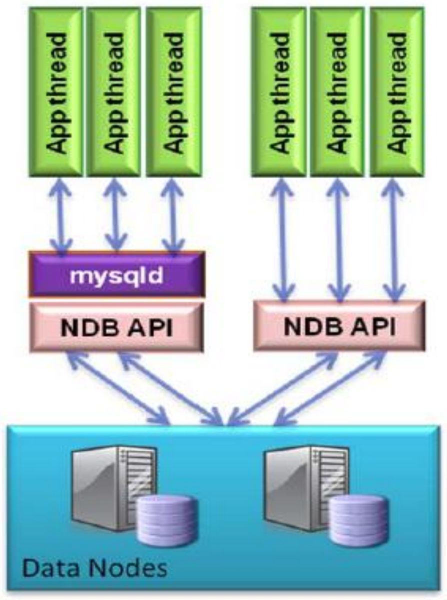
Using Connection Pools

To use connection pooling, each connection needs to have its own [mysqld] or [api] section in the config.ini file and then set ndbcluster-connection-pool to a value greater than 1 in my.cnf file

MySQL Cluster can deliver the best throughput especially when the data nodes are served with many operations in parallel. You must use multiple MySQL Servers together with multiple application threads accessing each of those MySQL Servers.

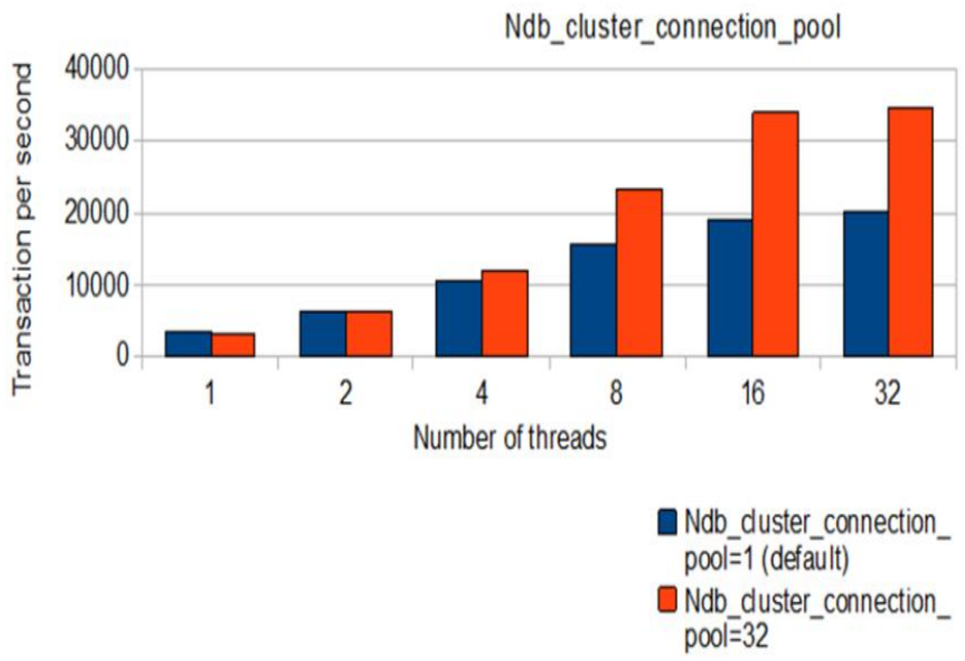
To access the data nodes, the *mysqld* process uses a single NDB API connection by default. As there are multiple application threads that are competing for that one connection, there is conflict on a mutex that prevents the throughput from scaling.

One solution would be to have each application thread use a dedicated *mysqld* process, but that would be wasteful of resources and can increase application complexity.



A more effective solution is to have multiple NDB API connections from the *mysqld* process to the data nodes as shown in the above figure.

To use connection pooling, each connection (from the mysqld) needs to have its own [mysqld] or [api] section in the *config.ini* file and then set *ndbcluster-connection-pool* to a value greater than 1 in *my.cnf* file.



In addition, you must not specify *ndb-node-id* as a command line option while starting the *mysqld* process. Here in this example, the single mysqld process has 4 NDB API connections.

The above figure shows the performance benefits of using connection pooling. Applications normally experience a 70% improvement but in some cases it may even exceed 150%.

Scaling by Adding Nodes

It is possible to add new MySQL Servers or node groups to a running MySQL Cluster.

MySQL Cluster has been designed to be scaled horizontally. By just adding additional MySQL Servers or data nodes, you can increase the performance.

It is also possible to add new MySQL Servers or node groups to a running MySQL Cluster. That is, you need not shut down and reload the cluster. The existing table data could be repartitioned across all the data nodes.

Using Adaptive Query Localization, tuning parameters and exploiting distribution awareness can enable MySQL Cluster to serve an ever broader portfolio of application requirements.

The safest method is to use MySQL Cluster Manager. The Cluster Manager automates the process of adding additional nodes to a running Cluster and you can make sure that there is no loss of service during the process.

Conclusion

In the last nine years (since 2004), MySQL Cluster has emerged to meet the needs of workloads demanding extreme levels of performance and 99.99% uptime.

As explained in the above sections, using Adaptive Query Localization, tuning parameters and exploiting distribution awareness can enable MySQL Cluster to serve an ever broader portfolio of application requirements.

Enhancements in both the core database technology along with real time monitoring and reporting capabilities can enable the database developers and administrators to extend their use cases for MySQL Cluster.