Architecture of NDB

Mikael Ronström
Senior MySQL Architect

MySQL AB
MySQL Cluster Architecture

Scalability on three levels:
1. # Applications
3. # MySQL servers
4. # Database nodes

**Application**

MySQL Server

MySQL Server

MySQL Server

MySQL Cluster

NDB Kernel
(Database nodes)

DB

DB

DB

DB
MySQL Cluster Overview

- **Main memory**
  - Complemented with logging to disk

- **Clustered**
  - Database distributed over many nodes
  - Synchronous replication between nodes
  - Database automatically partitioned over the nodes
  - Fail-over capability in case of node failure

- **Update in one MySQL server, immediately see result in another**

- **Support for high update loads (as well as very high read loads)**
MySQL Cluster Value

• Bringing clustered data management to a broader range of customers
  – Affordable
  – Ease of use
  – Upgrade path from current MySQL installations

• Making high-availability main-stream
  – On-going standardization efforts assume off-the-shelf components
  – New database technology using main-memory storage with replication gives persistency and short fail-over times
MySQL Cluster Features (1)

- Support for mixed main memory and disk-based columns (indexed columns always in main memory)
- Transactions
- Synchronous Replication
- Auto-synch at restart of NDB node
- Recovery from checkpoints and logs at cluster crash
- Subscription to row changes
- Indexes (Unique hash and T-tree ordered)
MySQL Cluster Features (2)

- On-line Backup
- On-line Add Index
- On-line Drop Index
- On-line Add Column
- On-line SW Upgrade
- Asynchronous replication between clusters
- Conflict Resolution in Multi-Master Replication
History

- Requirement collection, prototypes and Research 1991-1996
- Draws from Ericsson’s extensive experience in building reliable solutions for the Telecom/IP industry
- Development start 1996 H2
- Version 0.1 demo august 1997 (primary key lookups and updates)
- Version 0.2 february 1998 (transaction support)
- Version 0.3 october 1998 (Cluster crash support)
- Version 1.0 april 2001 (Perl DBI, scan, ..)
- Version 1.1 august 2001 (First production release)
- Version 1.41 may 2002 (Node Recovery, Production release)
- Version 2.11 may 2003 (ODBC, On-line Backup, Unique Index)
- Version 2.12.3 february 2004 (production release for B2)
- Version 3.4.5 14th April 2004 (MySQL integration, Ordered Index, Online SW Upgrade) put in production by B2
MySQL Cluster Architecture

- Application
- Application
- Application
- Application

MySQL Server
- NDB API

Application
- NDB API

MySQL Server
- NDB API

NDB Kernel
(Database nodes)

MGM Server
(Management nodes)

Management Client
- MGM API

Copyright 2007 MySQL AB
The World's Most Popular Open Source Database
Data distribution

Horizontal fragmentation of Table 1 (4 fragments)

- Fragments distributed on nodes

Logical configuration

Node group 1

F1

F3

Node 1

Node 2

Node group 2

F2

F4

Node 3

Node 4

NDB Cluster: 4 node configuration on 2 dual processor machines

Physical configuration

Dual CPU

Node 1

Node 2

Node 3

Node 4

TCP/IP or SCI

Table 1

<table>
<thead>
<tr>
<th>Pnr</th>
<th>AccNo</th>
<th>Val</th>
<th>$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Synchronous Replication: Low failover time

messages = 2 x operations x (replicas + 1)

Example showing transaction with two operations using three replicas

1. Prepare F1
2. Commit F1
1. Prepare F2
2. Commit F2
On-line Backup & Restore

Logical configuration

Node group 1
F1
F3
Node 1
F1
F3
Node 2

Check pointing & log files (Data files, UNDO log REDO log)

Node group 2
F2
F4
Node 3
F2
F4
Node 4

Back-up files (Meta-data, Data, Change Log)

Restore

Transactions

System Restart
Failure detection: Heartbeats, lost connections

- Nodes organized in a logical circle
- Heartbeat messages sent to next node in circle
- All nodes must have the same view of which nodes are alive
TC coordinates fragment operations

Example with two replicas

Running Node (Primary)

Copying

Restarting Node (Backup)

Copied fragments

If there are two fragment copies, then both are part of transaction

Update, insert, and delete

Read operations only read from one replica
Access methods for NDB tables

• 1) Primary key access
• 2) Full table scan (parallel)
• 3) Unique Index Key access
• 4) Parallel Range Scan (on ordered index)
• Many Primary Key and Unique Index Key accesses can be performed in parallel (in the same or different transactions)
Primary Key Access

• Define Table, Operation Type
• Call equal on all primary key attributes
• Columns to get/set
• Support for very simple interpreter functions in update/delete (e.g. \( AMOUNT = AMOUNT + 10 \))
• All tables have Primary Key (hidden if no primary key was specified in CREATE TABLE)
• Primary Key or part of it used as distribution key
• Linear hashing variant used locally for mapping to data page and row locks
Full table scan

- Define Table and type of scan
- Define columns to get
- Support for very simple filters (e.g. \texttt{AMOUNT < 10})
- Support for take over of locked row to updating primary key access
- Scan is performed in parallel on many fragments
Unique Key Index Access

• Specify Index Table and Table and Operation Type
• Call equal on unique index attributes
• Rest as Primary Key access
Parallel Range Scan

• Specify Ordered Index name and Table name and scan type
• Specify lower and upper bound (or equal) of range scan
• Rest is the same as full table scan
Unique Hash Index

```sql
CREATE TABLE SERVICE
(INTEGER SID PRIMARY KEY,
VARCHAR IPADDR, INTEGER PORT,
UNIQUE(IPADDR, PORT));
```

<table>
<thead>
<tr>
<th>IPADDR</th>
<th>PORT</th>
<th>NDB$PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>213.114.20.235</td>
<td>4000</td>
<td>2</td>
</tr>
<tr>
<td>213.114.20.229</td>
<td>4000</td>
<td>17</td>
</tr>
<tr>
<td>213.114.20.207</td>
<td>4010</td>
<td>13</td>
</tr>
<tr>
<td>213.114.20.218</td>
<td>4007</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>IPADDR</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>213.114.20.229</td>
<td>4000</td>
</tr>
<tr>
<td>24</td>
<td>213.114.20.218</td>
<td>4007</td>
</tr>
<tr>
<td>2</td>
<td>213.114.20.235</td>
<td>4000</td>
</tr>
<tr>
<td>13</td>
<td>213.114.20.207</td>
<td>4010</td>
</tr>
</tbody>
</table>
CREATE INDEX SINDX ON SERVICE (IPADDR, PORT);
Index Support

• Index maintenance through internal triggers (insert, delete, update) on primary table
  - Insert on primary table (insert into index)
  - Delete from primary table (delete from index)
  - Update of primary table (delete from and insert into index)

• Explicit usage through NDB API
  - Unique hash index accessed in similar manner as primary key access
  - Ordered index accessed in similar manner as scan
Index Support

• Implicit usage through SQL
  - Unique hash index chosen if primary key is not bound and unique hash index is bound
  - Ordered index is scanned if a range scan is needed
  - Query optimization can re-order query plans and use indexes
Indexes, Implementation Overview

- Unique hash index
  - Stored as ordinary table
  - Data is stored in index (copies of attributes)
  - Distributed differently than primary table
- Ordered indexes
  - Stored as T-trees in each node
  - Data is not stored in index (direct reference to attributes)
  - Distributed in the same way as the primary table
Fragmentation of Primary Table

Horizontal fragmentation of SERVICE table (4 fragments)

- Fragments distributed on nodes

Logical configuration

Node group 1

Node 1

F1

F4

Node 2

F1

F4

Node group 2

Node 3

F2

F3

Node 4

F2

F3

Physical configuration

Sun E220

Node 1

Node 2

Node 3

Node 4

TCP/IP or SCI

NDB Cluster: 4 node configuration on 2 dual processor machines

2 copies of data
Fx – primary replica
Fx – secondary replica
Unique Hash Index, Fragmentation of Index Table

Horizontal fragmentation of IP_ADDR_1 table

- Fragments distributed on nodes
Unique Hash Index, Two-step Access

Step 1: Index table lookup

Step 2: Operation on primary table

TC: Transaction Coordinator
Unique Hash Index Maintenance

Step 1: Update on Primary Table

Step 2: Delete old index reference
Insert new reference

TC: Transaction Coordinator
Ordered Index, Parallel Range Scan

TC: Transaction Coordinator

Parallel Scan

Node group 1

Node 1
F1
F4

Node 2
F1
F4

Node group 2

Node 3
F2
F3

Node 4
F2
F3
Ordered Index Maintenance

1) Delete index entry (at commit)
2) Insert index entry (at update)

TC: Transaction Coordinator

Update on Primary Table
MySQL Cluster API’s

- All the MySQL API’s
- MGM API (C API for cluster management)
- NDB API (NDB table access API, C++)
NDB API: Highly optimized native C++ API

- Create Table, Drop Table, Create Index, Drop Index (only to be used from NDB storage handler)
- Read, Update, Delete, Insert and Write (= Replace)
- Parallel operations and transactions through synchronous interface
- Parallel operations/transactions through asynchronous interface
- Load Balancing towards NDB nodes when starting new transaction
- Possible to place transaction coordinator where data of transaction resides
- Always uses a live NDB node when starting a new transaction
- Adaptive Send algorithm for optimized performance
- Interpreted programs for scan filtering
- Event (detached trigger) support on operation and column level
Looking at performance

Five synchronous insert transactions
(10 x TCP/IP time)

Five inserts in one synchronous transaction
(2 x TCP/IP time)

Five asynchronous insert transactions
(2 x TCP/IP time)
When use which API in MySQL Cluster

• Most applications will be fine with MySQL API’s
• When performance is really critical use the synchronous NDB API interface, fairly easy to use
  ⇒ Around 3 times the performance
• If performance needs to be even better and the technical competence is there use asynch NDB API interface
  ⇒ Another 2 times better performance
Cluster Development

• CPU’s are getting faster as always
• Multiple cores per chip
• Multiple chip per board, 2 common
• Clusters of low-end servers
⇒ 4 dimensions of development
• MySQL Cluster uses ALL of these developments simultaneously by automatically parallelising the low-level database operations
MySQL Cluster Future Features

• On-line Add Node
• Multi-threaded NDB kernel nodes
• Mixed-endian Clusters
MySQL Cluster RT solution on Dual-Core computer

Memory Optimized Architecture

**CPU 0**
- Connection Threads
- Watch Dog thread
- FileSystem threads

**CPU 1**
- Main Thread
- Super Sockets
- Read/Write

Rest
MySQL Cluster RT solution on Quad-Core computer using 4 data nodes
CPU optimized architecture
Minimal Cluster, 2 data nodes, 2 computers

**DBT2 with 2 Mysql servers**

**Improvement is due to latency**

**Improvement is due to efficiency**

**Low load**

**Heavy load**

**Response time 2 MySQL servers**

- **Response time eth ms**
- **Response time sci ms**
- **Response time eth + rt ms**
- **Response time sci + rt ms**

Parallel activity
CPU-optimized architecture
Distribution aware (8 data nodes)

CPU-optimized architecture Distribution aware (8 data nodes)
Improvement is due to latency

Improvements compared to ethernet
CPU-optimized architecture Distribution aware (8 data nodes)
Architecture of NDB

Mikael Ronström
Senior MySQL Architect

MySQL AB
MySQL Cluster Architecture

Scalability on three levels:
1. # Applications
2. # MySQL servers
3. # Database nodes
MySQL Cluster Overview

- **Main memory**
  - Complemented with logging to disk
- **Clustered**
  - Database distributed over many nodes
  - Synchronous replication between nodes
  - Database automatically partitioned over the nodes
  - Fail-over capability in case of node failure
- **Update in one MySQL server, immediately see result in another**
- **Support for high update loads (as well as very high read loads)**
MySQL Cluster Value

- Bringing clustered data management to a broader range of customers
  - Affordable
  - Ease of use
  - Upgrade path from current MySQL installations

- Making high-availability main-stream
  - On-going standardization efforts assume off-the-shelf components
  - New database technology using main-memory storage with replication gives persistency and short fail-over times
MySQL Cluster Features (1)

• Support for mixed main memory and disk-based columns (indexed columns always in main memory)
• Transactions
• Synchronous Replication
• Auto-synch at restart of NDB node
• Recovery from checkpoints and logs at cluster crash
• Subscription to row changes
• Indexes (Unique hash and T-tree ordered)
MySQL Cluster Features (2)

• On-line Backup
• On-line Add Index
• On-line Drop Index
• On-line Add Column
• On-line SW Upgrade
• Asynchronous replication between clusters
• Conflict Resolution in Multi-Master Replication
History

- Requirement collection, prototypes and Research 1991-1996
- Draws from Ericsson’s extensive experience in building reliable solutions for the Telecom/IP industry
- Development start 1996 H2
- Version 0.1 demo august 1997 (primary key lookups and updates)
- Version 0.2 february 1998 (transaction support)
- Version 0.3 october 1998 (Cluster crash support)
- Version 1.0 april 2001 (Perl DBI, scan, ..)
- Version 1.1 august 2001 (First production release)
- Version 1.41 may 2002 (Node Recovery, Production release)
- Version 2.11 may 2003 (ODBC, On-line Backup, Unique Index)
- Version 2.12.3 february 2004 (production release for B2)
- Version 3.4.5 14th April 2004 (MySQL integration, Ordered Index, On-line SW Upgrade) put in production by B2
MySQL Cluster Architecture

Application

MySQL Server
NDB API

Application
NDB API

MySQL Server
NDB API

NDB Kernel
(Database nodes)

MGM Server
(Management nodes)

Management Client
MGM API

MySQL
Server

Application

Application

Application

Application
NDB Cluster: 4 node configuration on 2 dual processor machines

2 copies of data
Fx – primary replica
Fx – secondary replica

Horizontal fragmentation of Table 1 (4 fragments)
• Fragments distributed on nodes
Synchronous Replication: Low failover time

Example showing transaction with two operations using three replicas

messages = 2 x operations x (replicas + 1)

1. Prepare F1
2. Commit F1
1. Prepare F2
2. Commit F2
On-line Backup & Restore

Logical configuration

Node group 1
- F1
- F3
- Node 1
- Node 2

Node group 2
- F2
- F4
- Node 3
- Node 4

Back-up files
- (Meta-data, Data, Change Log)

Check pointing & log files
- (Data files, UNDO log, REDO log)

Restore
- Transactions

System Restart

Back-up files
- (Meta-data, Data, Change Log)

Check pointing & log files
- (Data files, UNDO log, REDO log)
Failure detection: Heartbeats, lost connections

Nodes organized in a logical circle

Heartbeat messages sent to next node in circle

All nodes must have the same view of which nodes are alive
Node Recovery

TC coordinates fragment operations

Example with two replicas

If there are two fragment copies, then both are part of transaction

Update, insert, and delete
Read operations only read from one replica
Access methods for NDB tables

• 1) Primary key access
• 2) Full table scan (parallel)
• 3) Unique Index Key access
• 4) Parallel Range Scan (on ordered index)
• Many Primary Key and Unique Index Key accesses can be performed in parallel (in the same or different transactions)
<table>
<thead>
<tr>
<th>Primary Key Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Define Table, Operation Type</td>
</tr>
<tr>
<td>• Call equal on all primary key attributes</td>
</tr>
<tr>
<td>• Columns to get/set</td>
</tr>
<tr>
<td>• Support for very simple interpreter functions in update/delete (e.g. AMOUNT = AMOUNT + 10)</td>
</tr>
<tr>
<td>• All tables have Primary Key (hidden if no primary key was specified in CREATE TABLE)</td>
</tr>
<tr>
<td>• Primary Key or part of it used as distribution key</td>
</tr>
<tr>
<td>• Linear hashing variant used locally for mapping to data page and row locks</td>
</tr>
</tbody>
</table>
Full table scan

- Define Table and type of scan
- Define columns to get
- Support for very simple filters (e.g. AMOUNT < 10)
- Support for take over of locked row to updating primary key access
- Scan is performed in parallel on many fragments
Unique Key Index Access

• Specify Index Table and Table and Operation Type
• Call equal on unique index attributes
• Rest as Primary Key access
Parallel Range Scan

- Specify Ordered Index name and Table name and scan type
- Specify lower and upper bound (or equal) of range scan
- Rest is the same as full table scan
Unique Hash Index

CREATE TABLE SERVICE
(INTEGER SID PRIMARY KEY,
 VARCHAR IPADDR, INTEGER PORT,
 UNIQUE(IPADDR, PORT));

<table>
<thead>
<tr>
<th>IPADDR</th>
<th>PORT</th>
<th>NDB$PK</th>
</tr>
</thead>
<tbody>
<tr>
<td>213.114.20.235</td>
<td>4000</td>
<td>2</td>
</tr>
<tr>
<td>213.114.20.229</td>
<td>4000</td>
<td>17</td>
</tr>
<tr>
<td>213.114.20.207</td>
<td>4010</td>
<td>13</td>
</tr>
<tr>
<td>213.114.20.218</td>
<td>4007</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>IPADDR</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>213.114.20.229</td>
<td>4000</td>
</tr>
<tr>
<td>24</td>
<td>213.114.20.218</td>
<td>4007</td>
</tr>
<tr>
<td>2</td>
<td>213.114.20.235</td>
<td>4000</td>
</tr>
<tr>
<td>13</td>
<td>213.114.20.207</td>
<td>4010</td>
</tr>
</tbody>
</table>
Ordered Index

CREATE INDEX SINDEX ON SERVICE(IPADDR, PORT);

<table>
<thead>
<tr>
<th>SID</th>
<th>IPADDR</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>213.114.20.229</td>
<td>4000</td>
</tr>
<tr>
<td>24</td>
<td>213.114.20.218</td>
<td>4007</td>
</tr>
<tr>
<td>2</td>
<td>213.114.20.235</td>
<td>4000</td>
</tr>
<tr>
<td>13</td>
<td>213.114.20.207</td>
<td>4010</td>
</tr>
</tbody>
</table>
Index Support

• Index maintenance through internal triggers (insert, delete, update) on primary table
  - Insert on primary table (insert into index)
  - Delete from primary table (delete from index)
  - Update of primary table (delete from and insert into index)

• Explicit usage through NDB API
  - Unique hash index accessed in similar manner as primary key access
  - Ordered index accessed in similar manner as scan
Index Support

- Implicit usage through SQL
  - Unique hash index chosen if primary key is not bound and unique hash index is bound
  - Ordered index is scanned if a range scan is needed
  - Query optimization can re-order query plans and use indexes
Indexes, Implementation Overview

- Unique hash index
  - Stored as ordinary table
  - Data is stored in index (copies of attributes)
  - Distributed differently than primary table
- Ordered indexes
  - Stored as T-trees in each node
  - Data is not stored in index (direct reference to attributes)
  - Distributed in the same way as the primary table
Fragmentation of Primary Table

Horizontal fragmentation of SERVICE table (4 fragments)
- Fragments distributed on nodes

NDB Cluster: 4 node configuration on 2 dual processor machines

Logical configuration
- Node group 1
  - Node 1: F1
  - Node 2: F4
  - Node 3: F2
  - Node 4: F3

Physical configuration
- Sun E220
  - Node 1
  - Node 2
  - Node 3
  - Node 4

TCP/IP or SCI

2 copies of data
Fx – primary replica
Fy – secondary replica
Unique Hash Index, Fragmentation of Index Table

Horizontal fragmentation of IP_ADDR_1 table
* Fragments distributed on nodes

Node group 1
Node group 2

Node 1
Node 2
Node 3
Node 4

F1
F2
F3
F4
Unique Hash Index, Two-step Access

Step 1: Index table lookup

Step 2: Operation on primary table

TC: Transaction Coordinator
Unique Hash Index Maintenance

Step 1: Update on Primary Table
Step 2: Delete old index reference
   Insert new reference

TC: Transaction Coordinator
Ordered Index, Parallel Range Scan

TC: Transaction Coordinator

Parallel Scan

Node group 1

Node group 2

Node 1

Node 2

Node 3

Node 4

Copyright 2007 MySQL AB

The World’s Most Popular Open Source Database
Ordered Index Maintenance

1) Delete index entry (at commit)
2) Insert index entry (at update)

TC: Transaction Coordinator

Update on Primary Table
MySQL Cluster API’s

- All the MySQL API’s
- MGM API (C API for cluster management)
- NDB API (NDB table access API, C++)
NDB API: Highly optimized native C++ API

- Create Table, Drop Table, Create Index, Drop Index (only to be used from NDB storage handler)
- Read, Update, Delete, Insert and Write (= Replace)
- Parallel operations and transactions through synchronous interface
- Parallel operations/transactions through asynchronous interface
- Load Balancing towards NDB nodes when starting new transaction
- Possible to place transaction coordinator where data of transaction resides
- Always uses a live NDB node when starting a new transaction
- Adaptive Send algorithm for optimized performance
- Interpreted programs for scan filtering
- Event (detached trigger) support on operation and column level
Looking at performance

- Five synchronous insert transactions (10 x TCP/IP time)
- Five inserts in one synchronous transaction (2 x TCP/IP time)
- Five asynchronous insert transactions (2 x TCP/IP time)
When use which API in MySQL Cluster

• Most applications will be fine with MySQL API’s
• When performance is really critical use the synchronous NDB API interface, fairly easy to use
  ⇒ Around 3 times the performance
• If performance needs to be even better and the technical competence is there use asynch NDB API interface
  ⇒ Another 2 times better performance
Cluster Development

• CPU's are getting faster as always
• Multiple cores per chip
• Multiple chip per board, 2 common
• Clusters of low-end servers
⇒ 4 dimensions of development
• MySQL Cluster uses ALL of these developments simultaneously by automatically parallelising the low-level database operations
MySQL Cluster Future Features

- On-line Add Node
- Multi-threaded NDB kernel nodes
- Mixed-endian Clusters
MySQL Cluster RT solution
on Dual-Core computer
Memory Optimized Architecture

CPU 0
- Connection Threads
- Watch Dog thread
- FileSystem threads
- Rest

CPU 1
- Main Thread
- Super Sockets
- Read/Write

Copyright 2007 MySQL AB
The World’s Most Popular Open Source Database
MySQL Cluster RT solution on Quad-Core computer using 4 data nodes
CPU optimized architecture

CPU 0
- Connection Threads
- Watch Dog thread
- FileSystem threads
- Main Thread
- Super Sockets
- Read/Write

CPU 1
- Connection Threads
- Watch Dog thread
- FileSystem threads
- Main Thread
- Super Sockets
- Read/Write

CPU 2
- Connection Threads
- Watch Dog thread
- FileSystem threads
- Main Thread
- Super Sockets
- Read/Write

CPU 3
- Connection Threads
- Watch Dog thread
- FileSystem threads
- Main Thread
- Super Sockets
- Read/Write
Minimal Cluster, 2 data nodes, 2 computers

DBT2 with 2 Mysql servers

Improvement is due to latency

Improvement is due to efficiency

Low load

Parallel activity

Heavy load

Transactions per minute

eth
sci
eth + rt
sci + rt

Response time 2 MySQL servers

ms

Response time eth ms
Response time sci ms
Response time eth + rt ms
Response time sci + rt ms

Parallel activity

Copyright 2007 MySQL AB
The World's Most Popular Open Source Database
CPU-optimized architecture
Distribution aware (8 data nodes)

**Graph 1:**
- x-axis: Parallel activity
- y-axis: Improvement in latency
- Lines: eth, sci, eth+rt, sci + rt

**Graph 2:**
- x-axis: Parallel activity
- y-axis: Improvement (%)
- Lines: Improvement sci vs eth, Improvement eth+rt vs eth, Improvement sci+rt vs eth

Low load
Heavy load

Improvement is due to latency.