What's new in MySQL Cluster 7.3

Andrew Morgan

Oracle

UK

Schlüsselworte
MySQL, MySQL Cluster, Real-Time, High Availability, JavaScript, Node.js, NoSQL

Einleitung

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

Originally designed as an embedded telecoms database for in-network applications demanding carrier-grade availability and real-time performance, MySQL Cluster has been rapidly enhanced with new feature sets that extend use cases into web, mobile and enterprise applications deployed on-premise or in the cloud, including:

- High volume OLTP
- Real time analytics
- Ecommerce, inventory management, shopping carts, payment processing, fulfillment tracking, etc.
- Online Gaming
- Financial trading with fraud detection
- Mobile and micro-payments
- Session management & caching
- Feed streaming, analysis and recommendations
- Content management and delivery
- Communications and presence services
- Subscriber / user profile management and entitlements

The purpose of this whitepaper is to explore the latest enhancements delivered as part of the MySQL Cluster 7.3 release, enabling users to:

Get started quickly:

- MySQL Cluster Auto-Installer – a browser-based GUI that will provision a well configured, distributed Cluster in minutes, ready for test, dev or production environments.

Agile Development:

- Foreign Keys (FKs) allow simpler application development with referential integrity implemented in the database rather than in the application code. High levels of compatibility with InnoDB FKs simplifies the process of migrating applications to MySQL Cluster.
- NoSQL Node.js driver allows JavaScript applications to read and write MySQL Cluster data directly without transformations to SQL. Developers can re-use JavaScript from the client to the server, all the way through to a distributed, fault-tolerant, transactional database.
- MySQL Server 5.6 support. Maximizing application flexibility by enabling developers to select the best MySQL storage engine for each table of their database, using the very latest MySQL GA release. Some tables will benefit from InnoDB, while others requiring auto-sharding, 99.999% uptime, etc. can be configured with a single variable to use MySQL Cluster.

Simplify administration:

- MySQL Cluster Manager extended to provide centralised, on-line backup and restore.
- Transactional slave positioning improves the robustness of asynchronous replication between Clusters.

Increase performance and scale-out:

- Improved thread scalability reduces the number of connections that need to be configured between the application and data nodes – simplifying management and supporting the on-line addition of more application nodes as the application scales
- SQL optimizations deliver higher query throughput with lower latency

Ihre Überschrift

The following table summarizes the most important new features within the MySQL Cluster 7.3 release. Please see the documentation for the full feature set: http://dev.mysql.com/doc/#cluster

Feature	Release
MySQL Cluster Auto-installer	MySQL Cluster 7.3
Foreign Keys	MySQL Cluster 7.3
NoSQL API: JavaScript for Node.js	MySQL Cluster 7.3
Connection Thread scalability	MySQL Cluster 7.3
MySQL 5.6 Integration	MySQL 5.6
SQL Performance	MySQL Cluster 7.3 (MySQL 5.6)
MySQL Replication enhancements	MySQL Cluster 7.3 (MySQL 5.6)
Centralized backup & restore	MySQL Cluster Manager 1.2

MySQL Cluster Auto-Installer

A major priority for this release is to make it much easier and faster to provision a cluster that is well tuned for your application and environment; we want you to focus on exploiting the benefits of MySQL Cluster in your application rather than on figuring out how to install, configure and start the database. The MySQL Cluster Auto-Installer provides a browser-based GUI which steps you through creating a Cluster tailored to your requirements. For a really good view of how the tool works, a tutorial video is available[1].

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

Figure 1 illustrates the steps that the auto-installer walks you through:

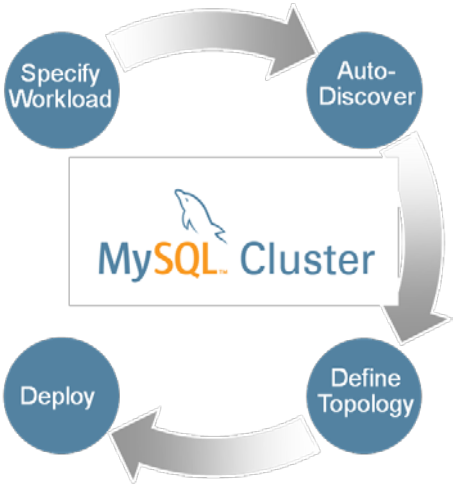

Figure 1 MySQL Cluster Auto-Installer

- Specify the anticipated workload type (for example – real-time application which is write-intensive) together with the list of hosts the Cluster will run on
- The installer will then optionally connect with each of the hosts (dependent on SSH access) to discover what resources are available (operating system, memory & CPU cores)
- A topology (which nodes/processes should form the Cluster and which hosts they should run on) is then proposed which the user may then accept or modify
- Based on all of the provided and discovered information, configuration settings are proposed – again the user can accept or modify these
- Finally the installer will optionally (dependent on SSH access) deploy and start the Cluster and show the status of the nodes as the Cluster comes into service. If SSH is not available then the Cluster can be configured and started manually using the commands that the auto-installer displays

Foreign Keys

Foreign Keys (FKs) are a way of implementing relationships/constraints between columns in different tables. For example, in Figure 2, we want to make sure that the value of the county column in the towns table has an associated entry in the counties table. In this way, no-one can place a town in a non-existent county and similarly no one can remove a county and leave orphaned towns behind.

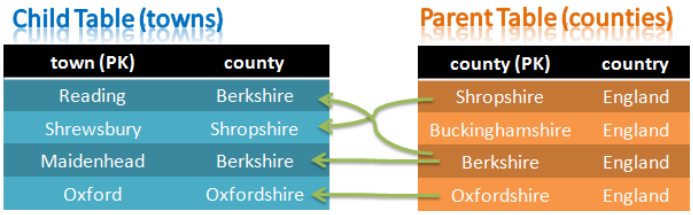

Figure 2 Foreign Key constraints between tables

We refer to the towns table as the *child* and the counties table as the *parent*.

We believe that this is going to enable a whole new set of applications to exploit the advantages of MySQL Cluster:

- Developers want to simplify their application by pushing referential checks down into the database
- The application is built upon 3rd party middleware that is dependent on FKs
- The application is already so dependent on FKs that it would be too complex to remove them

NoSQL API: JavaScript Driver for Node.js

Node.js is a platform that allows fast, scalable network applications (typically web applications) to be developed using JavaScript. Node.js is designed for a single thread to serve millions of client connections in real-time – this is achieved by an asynchronous, event-driven architecture – just like MySQL Cluster, making them a great match.

The MySQL Cluster NoSQL Driver for Node.js is implemented as a module for the V8 engine, providing Node.js with a native, asynchronous JavaScript interface that can be used to both query and receive results sets directly from MySQL Cluster, without transformations to SQL. As an added benefit, you can direct the driver to use SQL so that the same API can be used with InnoDB tables.

With the MySQL Cluster JavaScript Driver for Node.js, architects can re-use JavaScript from the client to the server, all the way through to a distributed, fault-tolerant, transactional database supporting real-time, high-scale services such as:

- Process streaming data from digital advertising and user tracking systems
- Gaming and social networks, powering the back-end infrastructure for serving mobile devices

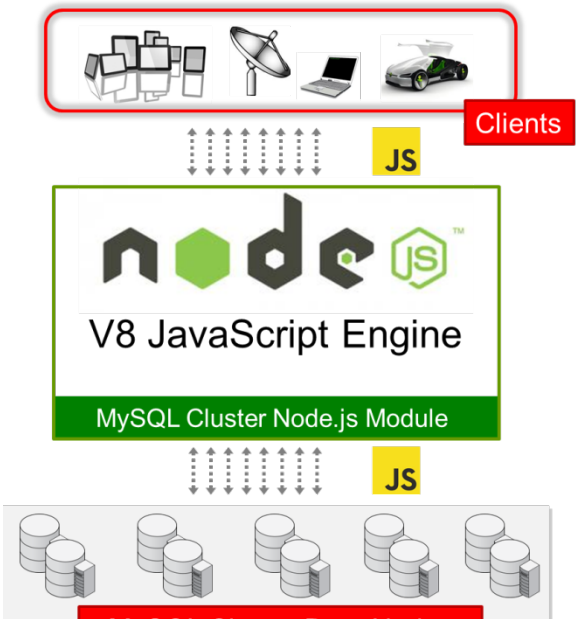

Figure 3 Real-time access to data using JavaScript API

JavaScript with Node.js joins a growing portfolio of NoSQL APIs for MySQL Cluster, which already includes Memcached, Java, JPA and HTTP/REST. And of course, developers can still depend on SQL to execute complex queries and access the rich ecosystem of connectors, frameworks, tooling and skills.

Connection Thread scalability

MySQL Cluster thrives when it is offered as many operations in parallel as possible. To achieve this, parallelism should be configured at each layer. As illustrated in Figure 4, there should be multiple application threads sending work to the MySQL Server (or other API), there should be multiple MySQL Servers and finally multiple connections between the MySQL Server (or other API node) and the data nodes. This is explained in more detail in the MySQL Cluster Performance white paper[2].

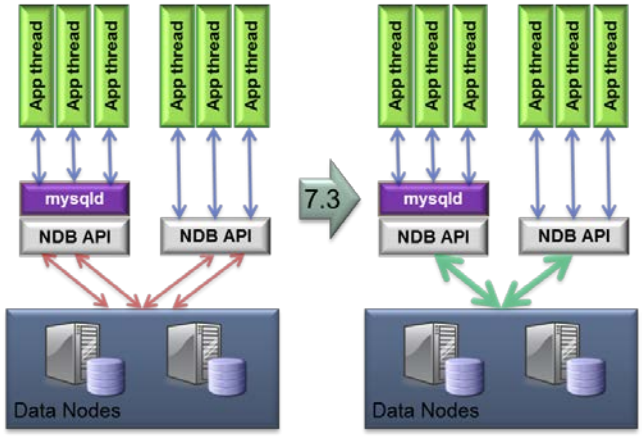

Figure 4 Greater throughput on each connection

Each of the connections to the data nodes consumes one of the 256 available node-ids and so in some scenarios they could cap the scalability of the Cluster. MySQL Cluster 7.3 greatly increases the throughput of each of these connections meaning that less connections (and therefore node-ids) are needed to

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

tackle the same workload; this in turn means that more API nodes and data nodes can be added to the Cluster to scale the capacity and performance even further.

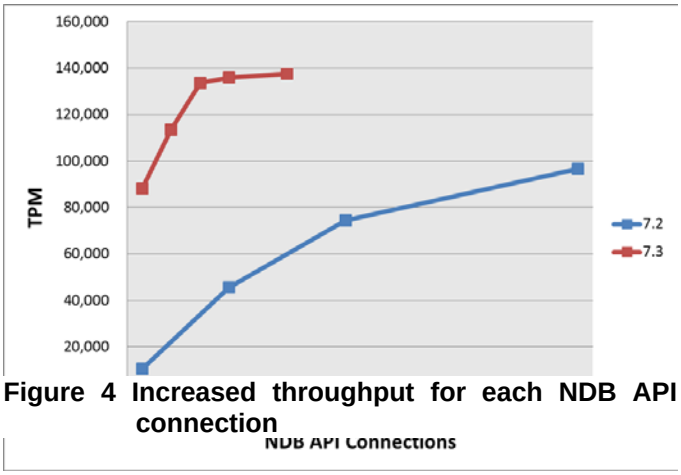

Figure 4 Increased throughput for each NDB API connection

Benchmarks have shown up to a 8.5x increase in throughput per connection as show in Figure 5. The graph shows how the overall throughput increases with the number of NDB API connections between the data nodes and application node (for example the MySQL Server); in MySQL Cluster 7.3 a single connection can handle 8.5x more transactions per minute and the throughput rises much more rapidly as extra connections are configured. This benchmark used a single MySQL Server running the DBT2 BM[3] and a single data node; this was running on an 8 socket, 8 (socket) x 6 (core) x 2 (thread) = 96 thread system[4] with 512 GB RAM.

Kontaktadresse:
Andrew Morgan
Oracle

E-Mail andrew.morgan@oracle.com
Internet: www.clusterdb.com
@clusterdb.com
@andrewmorgan

3 DBT-2 is an open source benchmark http://sourceforge.net/projects/osdldbt/files/dbt2/

4 Intel Xeon E7540 @ 2.00 GhZ