NoSQL and SQL - Why Choose? Enjoy the best of both worlds with MySQL

Andrew Morgan
Oracle
UK

Keywords:

mysql, nosql, mysql cluster, ha, high availability.

Introduction

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

MySQL Cluster Architecture

MySQL Cluster presents a single namespace to the application, allowing clients to connect to any node. Under the covers, there are three types of node which collectively provide service to the application:

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

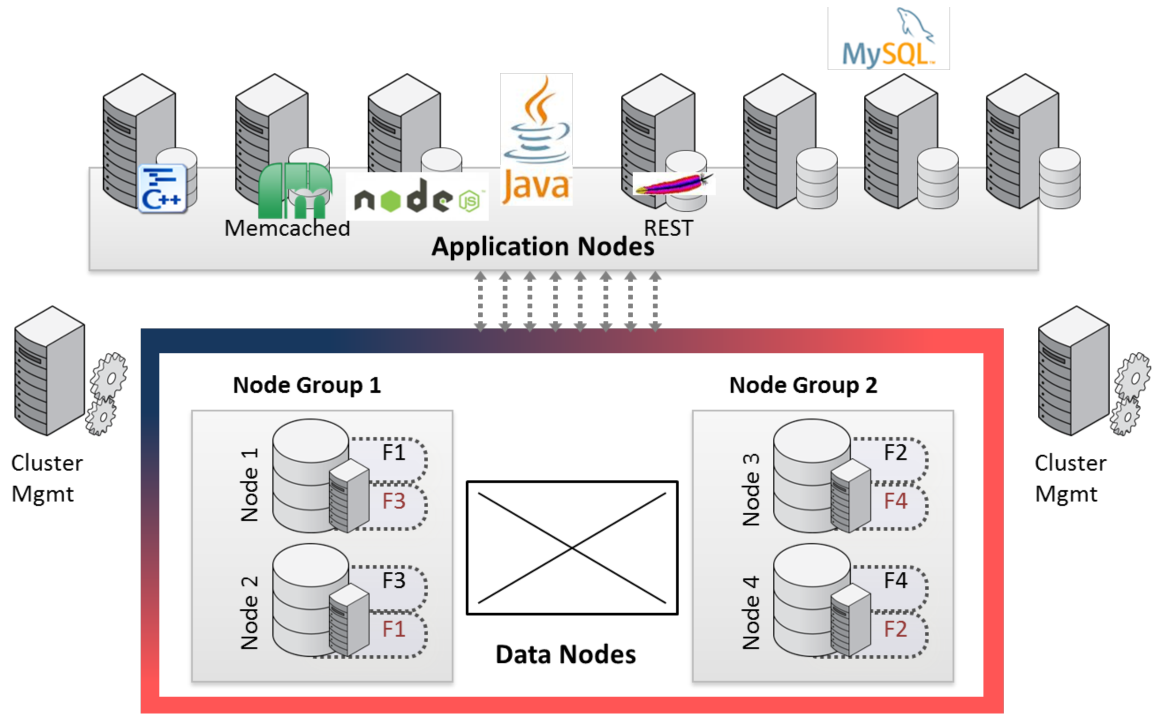

Figure 1 Four Data Node MySQL Cluster

Data Nodes are the main nodes of a MySQL Cluster. They provide the following functionality:

- Storage and management of both in-memory and disk-based data
- Automatic and user defined partitioning (sharding) of tables
- Synchronous replication of data between data nodes
- Transactions and data retrieval
- Automatic fail over
- Automatic resynchronization after failure for self-healing

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

By storing and distributing data in a shared-nothing architecture, i.e. without the use of a shared-disk, and synchronously replicating data to at least one replica, if a Data Node happens to fail, there will always be another Data Node storing the same information. This allows for requests and transactions to continue to be satisfied without interruption. Any transactions which are aborted during the short (sub-second) failover window following a Data node failure are rolled back and can be re-run.

It is possible to choose how to store data; either all in memory or with some on disk (non-indexed data only). In-memory storage can be especially useful for data that is frequently changing (the active working set). Data stored in-memory is routinely check pointed to disk locally and coordinated across all Data Nodes so that the MySQL Cluster can be recovered in case of a complete system failure – such as a power outage. Disk-based data can be used to store data with less strict performance requirements, where the data set is larger than the available RAM. As with most other database servers, a page-cache is used to cache frequently used disk-based data in the Data Nodes' memory in order to increase the performance.

Application Nodes provide connectivity from the application logic to the data nodes. Applications can access the database using SQL through one or many MySQL Servers performing the function of SQL interfaces into the data stored within a MySQL Cluster. When going through a MySQL Server, any of the standard MySQL connectors can be used[1], offering a wide range of access technologies.

Alternatively, a high performance (C++ based) interface called NDB API can be used for extra control, better real-time behavior and greater throughput.

The NDB API provides a layer through which additional NoSQL interfaces can directly access the cluster, bypassing the SQL layer, allowing for lower latency and improved developer flexibility. Existing interfaces include Java, JPA, Memcached and HTTP/REST (via an Apache Module). The latest MySQL Cluster 7.3 adds JavaScript for Node.js.

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

Management Nodes are responsible for publishing the cluster configuration to all nodes in the cluster and for node management. The Management Nodes are used at startup, when a node wants to join the cluster, and when there is a system reconfiguration. Management Nodes can be stopped and restarted without affecting the ongoing execution of the Data and Application Nodes. By default, the Management Node also provides arbitration services, in the event there is a network failure which leads to a "split-brain" or a cluster exhibiting "network-partitioning"[2].

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

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

NoSQL APIs

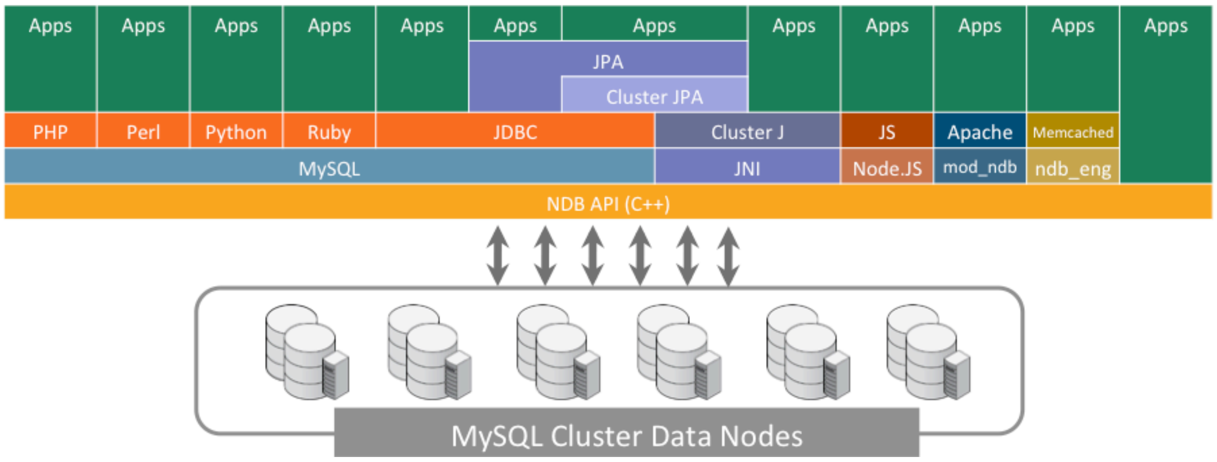

Figure 2 The NoSQL & SQL APIs offered by MySQL Cluster

One of the most recent APIs to be added is for JavaScript running within 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:

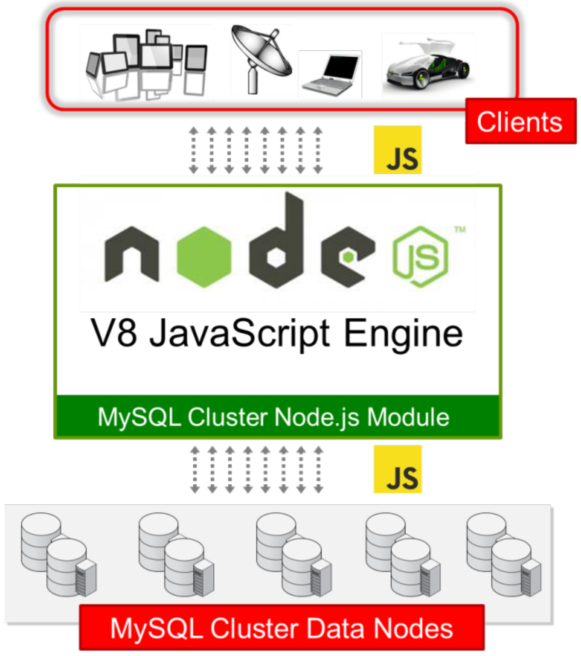

Figure 3 Real-time access to data using JavaScript API

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

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.

Contact address:

Name
Andrew Morgan, Oracle.

Email :	andrew.morgan@oracle.com
Internet:	www.clusterdb.com