NoSQL and SQL: Blending the Best of Both Worlds

Andrew Morgan
Oracle
UK

Keywords:

MySQL, NoSQL, Java, JSON, Memcached, MySQL Cluster, InnoDB, High Availability

Introduction

The ever increasing performance demands of web-based services has generated significant interest in providing NoSQL access methods to MySQL - enabling users to maintain all of the advantages of their existing relational database infrastructure, while providing blazing fast performance for simple queries, using an API to complement regular SQL access to their data. This session looks at the existing NoSQL access methods for MySQL as well as the latest developments for both the InnoDB and MySQL Cluster storage engines. See how you can get the best of both worlds - persistence, consistency, rich queries, high availability, scalability and simple, flexible APIs and schemas for agile development.

New Technologies Driving New Database Demands

It should come as no surprise that data volumes are growing at a much faster rate than anyone previously predicted. The McKinsey Global Institute estimates this growth at 40% per annum [http://www.mckinsey.com/mgi/publications/big_data/pdfs/MGI_big_data_full_report.pdf]

The databases needed to support such a massive growth in data have to meet new challenges, including:

- Automatically scaling database operations, both reads and writes, across commodity hardware
- High availability for 24x7 service uptime
- Choice of data access patterns and APIs introducing new levels of flexibility and convenience for developers
- Rapidly iterate and evolve databases and schemas to meet user demands for new functionality and dynamically scale for growth
- Ease-of-Use with reduced barriers to entry enabling developers to quickly innovate in the delivery of differentiated services.

Of course it can be argued that databases have had these requirements for years –but they have rarely had to address all of these requirements at once, in a single application. For example, some databases have been optimized for low latency and high availability, but then don’t have a means to scale write operations and can be difficult to use. Other databases maybe very simple to start with, but lack capabilities that make them suitable for applications with demanding uptime requirements.

With these types of requirements, it becomes clear that there is no single solution to meet all needs, and it is therefore essential to mix and match technologies in order to deliver functionality demanded by the business.

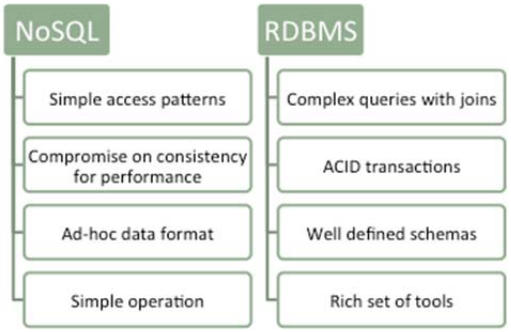

Figure 1 Combining the best of NoSQL & RDBMS

Through a combination of NoSQL access methods, technologies for scaling out on commodity hardware, integrated HA and support for on-line operations such as schema changes, MySQL is able to offer solutions that blend the best of both worlds.

NoSQL Key-Value Access to MySQL with Memcached

Providing NoSQL interfaces directly on top of a MySQL storage engine offers several major advantages:

- Delivers high performance for simple queries by bypassing the SQL layer
- Enables developers to work in their native programming languages and choose if and when to apply schemas when handling data management within the application. Faster development cycles result in accelerated time to market for new services.

Using NoSQL interfaces to MySQL, users can maintain all of the advantages of their existing relational database infrastructure including the ability to run complex queries, while providing blazing fast performance for simple Key/Value operations, using an API to complement regular SQL access to their data.

MySQL has previewed key value access to both the InnoDB and MySQL Cluster (NDB) storage engines, using the Memcached API, one of the most broadly adopted Key-Value interfaces in use today.

Using the Memcached API, web services can directly access InnoDB and MySQL Cluster without transformations to SQL, ensuring low latency and high throughput for read/write queries. Operations such as SQL parsing are eliminated and more of the server's hardware resources (CPU, memory and I/O) are dedicated to servicing the query within the storage engine itself.

Over and above higher performance and faster development cycles, there are a number of additional benefits:

- Preserves investments in Memcached infrastructure by re-using existing Memcached clients and eliminates the need for application changes
- Flexibility to concurrently access the same data set with SQL, allowing complex queries to be run while simultaneously supporting Key-Value operations from Memcached
- Extends Memcached functionality by integrating persistent, crash-safe, transactional database back-ends offering ACID compliance, rich query support and extensive management and monitoring tools
- Reduces service disruption caused by cache re-population after an outage

- Simplifies web infrastructure by optionally compressing the caching and database layers into a single data tier, managed by MySQL
- Reduces development and administration effort by eliminating the cache invalidation and data consistency checking required to ensure synchronization between the database and cache when updates are committed
- Eliminates duplication of data between the cache and database, enabling simpler re-use of data across multiple applications, and reducing memory footprint
- Simple to develop and deploy as many web properties are already powered by a MySQL database with caching provided by Memcached.

Memcached Implementation for InnoDB

The Memcached API for InnoDB has been previewed as part of the MySQL 5.6 Development Milestone Release.

As illustrated in the ***Figure 2***, Memcached for InnoDB is implemented via a Memcached daemon plug-in to the mysqld process, with the Memcached protocol mapped to the native InnoDB API.

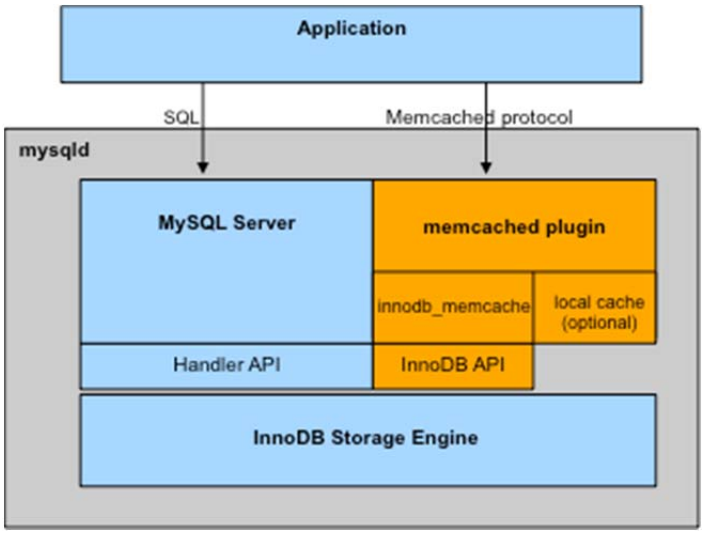

Figure 2 Memcached API Implementation for InnoDB

With the Memcached daemon running in the same process space, users get very low latency access to their data while also leveraging the scalability enhancements delivered with InnoDB and a simple deployment and management model. Multiple web / application servers can remotely access the Memcached / InnoDB server to get direct access to a shared data set.

With simultaneous SQL access, users can maintain all the advanced functionality offered by InnoDB including support for Foreign Keys, XA transactions and complex JOIN operations.

Additional Memcached API features provided in the preview release include:

- Support for both the Memcached text-based and binary protocols; the Memcached API for InnoDB passes all of the 55 memcapable tests, ensuring compatibility with existing applications and clients

- Supports users mapping multiple columns into a “value”. A pre-defined “separator” which is fully configurable separates the value
- Deployment flexibility with three options for local caching of data: “cache-only”, “innodb-only”, and “caching” (both “cache” and “innodb store”). These local options apply to each of four Memcached operations (set, get, delete and flush)
- All Memcached commands are captured in the MySQL Binary Log (Binlog), used for replication. This makes it very easy to scale database operations out across multiple commodity nodes, as well as provide a foundation for high availability.

Memcached Implementation for MySQL Cluster

Memcached API support for MySQL Cluster was released as part of the 7.2 Development Milestone Release [http://www.clusterdb.com/mysql-cluster/scalabale-persistent-ha-nosql-memcache-storage-using-mysql-cluster/], and joins an extensive range of NoSQL interfaces that are already available for MySQL Cluster (discussed later in this paper).

Like Memcached, MySQL Cluster provides a distributed hash table with in-memory performance. MySQL Cluster extends Memcached functionality by adding support for write-intensive workloads, a full relational model with ACID compliance (including persistence), rich query support, auto-sharding and 99.999% availability, with extensive management and monitoring capabilities.

All writes are committed directly to MySQL Cluster, eliminating cache invalidation and the overhead of data consistency checking to ensure complete synchronization between the database and cache.

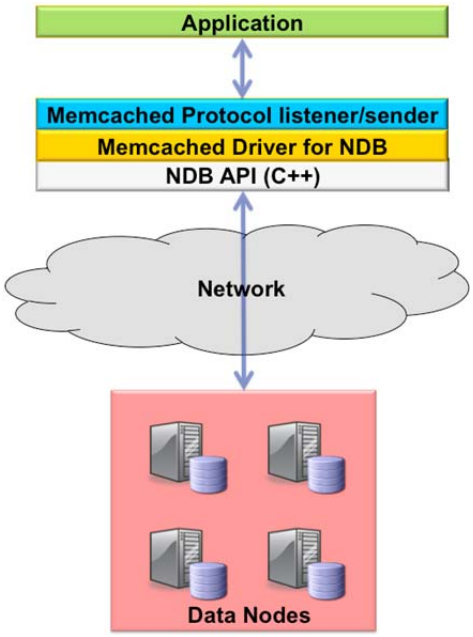

Figure 3 Memcached API Implementation with MySQL Cluster

Implementation is simple - the application sends reads and writes to the Memcached process (using the standard Memcached API). This in turn invokes the Memcached Driver for NDB (which is part of the same process), which in turn calls the NDB API for very quick access to the data held in MySQL Cluster’s data nodes.

The solution has been designed to be very flexible, allowing the application architect to find a configuration that best fits their needs. It is possible to co-locate the Memcached API in either the data nodes or application nodes, or alternatively within a dedicated Memcached layer.

Developers can still have some or all of the data cached within the Memcached server (and specify whether that data should also be persisted in MySQL Cluster) – so it is possible to choose how to treat different pieces of data, for example:

- Storing the data purely in MySQL Cluster is best for data that is volatile, i.e. written to and read from frequently
- Storing the data both in MySQL Cluster and in Memcached memory is often the best option for data that is rarely updated but frequently read
- Data that has a short lifetime, is read frequently and does not need to be persistent could be stored only in the Memcached cache.

The benefit of this approach is that users can configure behavior on a per-key-prefix basis (through tables in MySQL Cluster) and the application doesn't have to care – it just uses the memcached API and relies on the software to store data in the right place(s) and to keep everything synchronized.

Any changes made to the key-value data stored in MySQL Cluster will be recorded in the binary log, and so will be applied during replication and point-in-time recovery.

Using Memcached for Schema-less Data

By default, every Key-Value is written to the same table with each Key-Value pair stored in a single row – thus allowing schema-less data storage. Alternatively, the developer can define a key-prefix so that each key and value are linked to pre-defined columns in a specific table.

Of course if the application needs to access the same data through SQL then developers can map key prefixes to existing table columns, enabling Memcached access to schema-structured data already stored in MySQL Cluster.

MySQL Cluster Architecture

As illustrated in ***Figure 4***, MySQL Cluster comprises three types of node which collectively provide service to the application:

- Data nodes manage the storage and access to data. Tables are automatically sharded across the data nodes which also transparently handle load balancing, replication, failover and self-healing
- Application nodes provide connectivity from the application logic to the data nodes. Multiple APIs are presented to the application. MySQL provides a standard SQL interface, including connectivity to all of the leading web development languages and frameworks. There are also a whole range of NoSQL interfaces including memcached , REST/HTTP, C++ (NDB-API), Java and JPA
- Management nodes are used to configure the cluster and provide arbitration in the event of network partitioning.

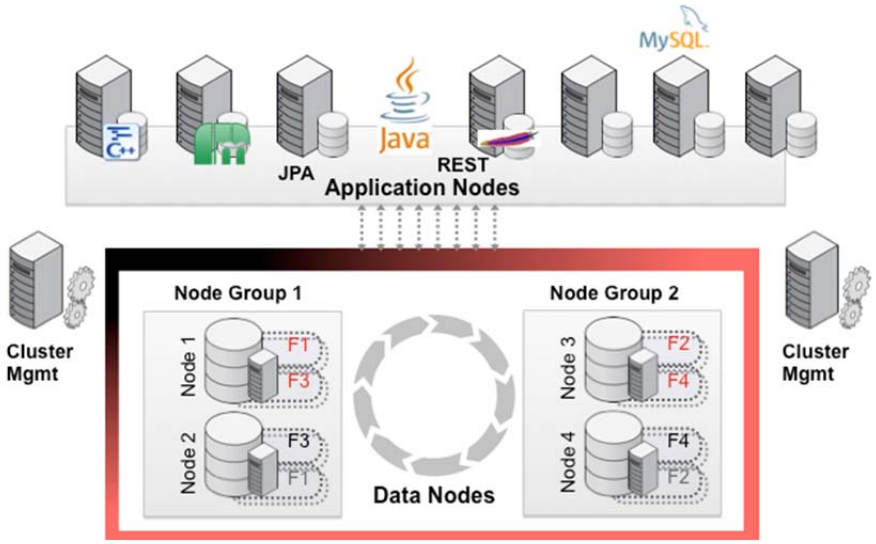

Figure 4 The MySQL Cluster architecture provides high write scalability across multiple SQL & NoSQL APIs

SQL and NoSQL Interfaces in MySQL Cluster

As MySQL Cluster stores tables in data nodes, rather than in the MySQL Server, there are multiple interfaces available to access the database. We have already discussed the Memcached API, but MySQL Cluster offers other interfaces as well for maximum developer flexibility.

Developers have a choice between:

- SQL for complex queries and access to a rich ecosystem of applications and expertise
- Simple Key-Value interfaces bypassing the SQL layer for blazing fast reads & writes
- Real-time interfaces for microsecond latency.

Figure 5 shows all of the access methods available to the database. The native API for MySQL Cluster is the C++ based NDB API. All other interfaces access the data through the NDB API.

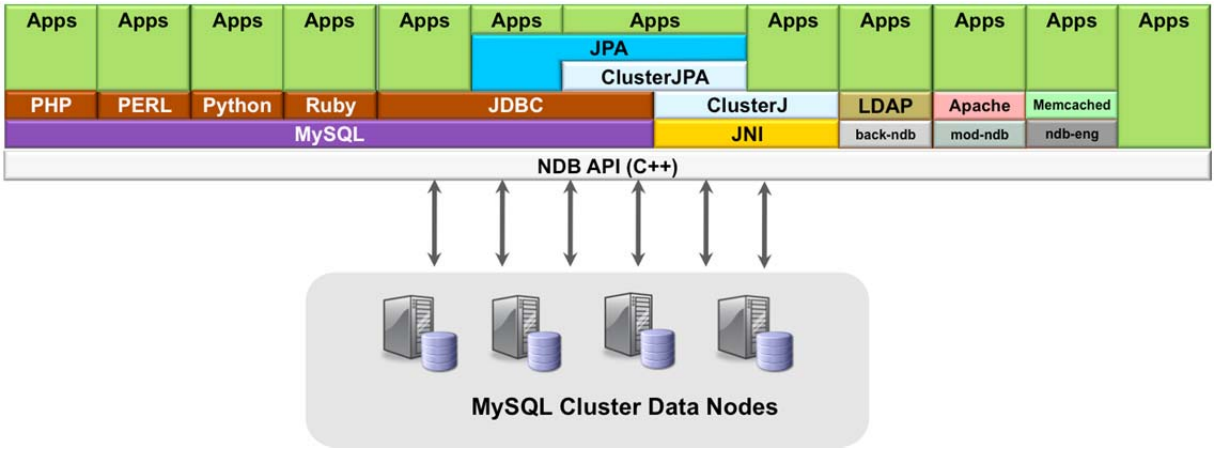

Figure 5 Ultimate Developer Flexibility – MySQL Cluster APIs

At the extreme right hand side of the figure, an application has embedded the NDB API library enabling it to make native C++ calls to the database, and therefore delivering the lowest possible latency.

On the extreme left hand side of the figure, MySQL presents a standard SQL interface to the data nodes, and provides connectivity to all of the standard MySQL connectors including:

- Common web development languages and frameworks, i.e. PHP, Perl, Python, Ruby, Ruby on Rails, Spring, Django, etc
- JDBC (for additional connectivity into ORMs including EclipseLink, Hibernate, etc)
- .NET, ODBC, etc.

Whichever API is chosen for an application, it is important to emphasize that all of these SQL and NoSQL access methods can be used simultaneously, across the same data set, to provide the ultimate in developer flexibility. Therefore, MySQL Cluster maybe supporting any combination of the following services, in real-time:

- Relational queries using the SQL API
- Key-Value-based web services using the REST/HTTP and memcached APIs
- Enterprise applications with the ClusterJ and JPA APIs
- Real-time web services (i.e. presence and location based) using the NDB API.

With auto-sharding, cross-data center geographic replication, online operations, SQL and NoSQL interfaces and 99.999% availability, MySQL Cluster is already serving some of the most demanding web and mobile telecoms services on the planet.

Integrating MySQL with Data Stores and Frameworks

Increases in data variety and volume are driving more holistic approaches to managing the data lifecycle – from data acquisition to organization and then analysis. In the past, one type of database may have been sufficient for most needs – now data management is becoming increasingly heterogeneous, with a requirement for simple integration between different data stores and frameworks, each deployed for a specific task.

MySQL has long offered comprehensive integration with applications and development environments. This integration is growing with new Oracle and community developed initiatives, including:

- The Binary Log (Binlog) API
- Apache Sqoop - for MySQL integration with Hadoop / HDFS.

The Binlog API enables developers to reduce the complexity of integration between different stores by standardizing SQL data management operations on MySQL, while enabling replication to other services within their data management infrastructure.

The Binlog API exposes a programmatic C++ interface to the binary log, implemented as a standalone library. Using the API, developers can read and parse binary log events both from existing binlog files as well as from running servers and replicate the changes into other data stores, including Hadoop, search engines, columnar stores, BI tools, etc.

You can download the code as part of MySQL Server Snapshot: mysql-5.6-labs-binary-log-api from http://labs.mysql.com

To demonstrate the possibilities of the Binlog API, an example application for replicating changes from MySQL Server to Apache Solr (a full text search server) has been developed. The example is available with the source download on Launchpad.

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. Users can use Sqoop to import data from MySQL into the Hadoop Distributed File System (HDFS) or related systems such as Hive and HBase. Conversely, users can use Sqoop to extract data from Hadoop and export it to external structured datastores such as relational databases and enterprise data warehouses.

Sqoop integrates with Oozie, allowing users to schedule and automate import and export tasks. Sqoop uses a connector-based architecture which supports plugins that provide connectivity to new external systems.

When using Sqoop, the dataset being transferred is sliced up into different partitions and a map-only job is launched with individual mappers responsible for transferring a slice of this dataset. Each record of the data is handled in a type safe manner since Sqoop uses the database metadata to infer the data types.

By default Sqoop includes connectors for most leading databases including MySQL and Oracle, in addition to a generic JDBC connector that can be used to connect to any database that is accessible via JDBC. Sqoop also includes a specialized fast-path connector for MySQL that uses MySQL-specific batch tools to transfer data with high throughput.

You can see practical examples of importing and exporting data with Sqoop here:
https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

Contact address:

Andrew Morgan
Oracle
UK

Phone: +44(0)7833-483595
Email: andrew.morgan@oracle.com
Internet: www.clusterdb.com
Twitter : @andrewmorgan, @clusterdb