Guide to NoSQL with MySQL

Delivering the Best of Both Worlds for Web Scalability

A MySQL® White Paper

June 2013

Table of Contents

Introduction

The explosive growth in user load and data volumes is driving new levels of innovation in database technologies, reflected in the rise of Big Data platforms and NoSQL data stores, coupled with the continued evolution of Relational Database Management Systems (RDBMS).

For the first time in many years, software architects, developers and administrators are faced with an array of new challenges in managing this data growth, and a bewildering array of potential solutions.

MySQL is the leading open source database for web, mobile, social and cloud services. Oracle's MySQL team has therefore been at the forefront in understanding these challenges, and innovating with capabilities that are designed to blend both relational and NoSQL technologies.

This Guide introduces these capabilities and demonstrates how organizations can achieve:
1. Fast data acquisition rates with flexible development, rapid iteration and ease-of-use;
2. While maintaining data integrity and the ability to execute rich queries against the data.

The result are solutions that deliver the best of both relational and NoSQL technologies, enabling organizations to harness the benefits of data growth while leveraging established and proven industry standards and skill-sets across their development and operational environments to capture, enrich and gain new insight from big data.

New Technologies Driving New Database Demands

Increasing internet penetration rates, social networking, high speed wireless broadband connecting smart mobile devices, new Machine to Machine (M2M) interactions and cloud computing are just some technologies fueling massive growth in user load and data volumes.

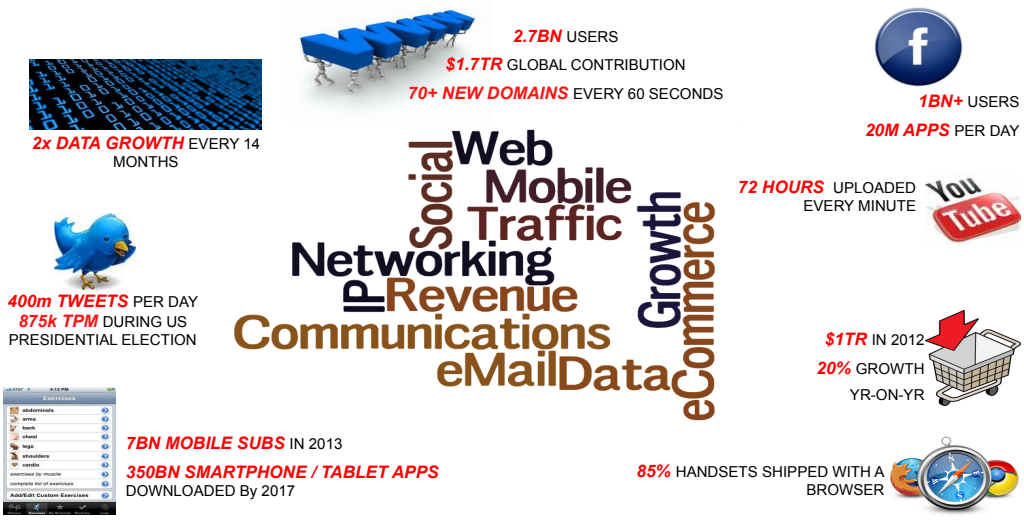


Figure 1: Key Trends Driving Growth in User Load and Data Volumes

The databases needed to support these trends have to meet new challenges, including:

- **Automatically scaling database operations**, both reads and writes, across commodity hardware, especially for high velocity data ingestion;
- **High availability** for 24x7 service uptime;
- **Choice of data access patterns and APIs** introducing new levels of flexibility and convenience for developers and users alike;

- **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.

It is also important to recognize that not all data is created equal. It is not catastrophic to lose individual elements of log data or status updates – the value of this data is more in its aggregated analysis. But some data is critical, such as ecommerce transactions, financial trades, customer updates, billing operations, order placement and processing, gaming state, access and authorization, service entitlements, etc.

Services generating and consuming this type of data still need the underlying data-store to meet the scalability and flexibility challenges discussed above, while still:

- Preserving transactional integrity with ACID[3] compliance;
- Enabling deep insight by running complex queries against the data, often in real-time;
- Leveraging the proven benefits of industry standards and skillsets to reduce cost, risk and complexity;
- Maintaining a well-structured data model, allowing simple data re-use between applications.

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.

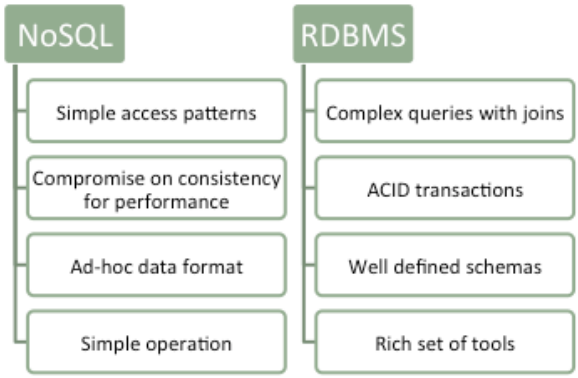


Figure 2: Combining the best of NoSQL & RDBMS

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

With data volume and velocity exploding, it is vital to be able to ingest and query data at high speed. For this reason, MySQL has implemented NoSQL interfaces directly to the InnoDB and MySQL Cluster (NDB) storage engines, which bypass the SQL layer completely. Without SQL parsing and optimization, Key-Value data can be written directly to MySQL tables up to 9x faster, while maintaining ACID guarantees.

[3] http://en.wikipedia.org/wiki/ACID

In addition, users can continue to run complex queries with SQL across the same data set, providing real-time analytics to the business or anonymizing sensitive data before loading to big data platforms such as Hadoop, while still maintaining all of the advantages of their existing relational database infrastructure.

A native Memcached API is part of MySQL 5.6 and MySQL Cluster. By using the ubiquitous Memcached API for writing and reading data, developers can preserve their investments in Memcached infrastructure by re-using existing Memcached clients, while also eliminating the need for application changes.

As discussed later, MySQL Cluster also offers additional NoSQL APIs beyond Memcached, including JavaScript, Java, JPA, HTTP/REST and C++. Developers are free to work in their native programming language, allowing faster and more agile development cycles.

Speed, when combined with flexibility, is essential in the world of big data. Complementing NoSQL access, support for on-line DDL (Data Definition Language) operations in MySQL 5.6 and MySQL Cluster enables Developers and DBAs to dynamically update their database schema to accommodate rapidly changing requirements, such as the need to capture additional data generated by their applications. These changes can be made without database downtime.

Using the Memcached interface, developers do not need to define a schema at all when using MySQL Cluster.

Over and above higher performance and faster development, there are a number of additional benefits in using Memcached as a NoSQL API to MySQL:

- Extends Memcached functionality by integrating persistent, crash-safe, transactional database back-ends offering ACID compliance, rich query support and extensive management and monitoring tools;
- Simplifies web infrastructure by optionally compressing the caching and database layers into a single data tier, managed by MySQL;
- Reduces service disruption caused by cache re-population after an outage;
- 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.

The following sections of the Guide discuss the Memcached implementation for both InnoDB and MySQL Cluster.

Memcached Implementation for InnoDB

The Memcached API for InnoDB was released with MySQL 5.6.

As illustrated in the following figure, 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.

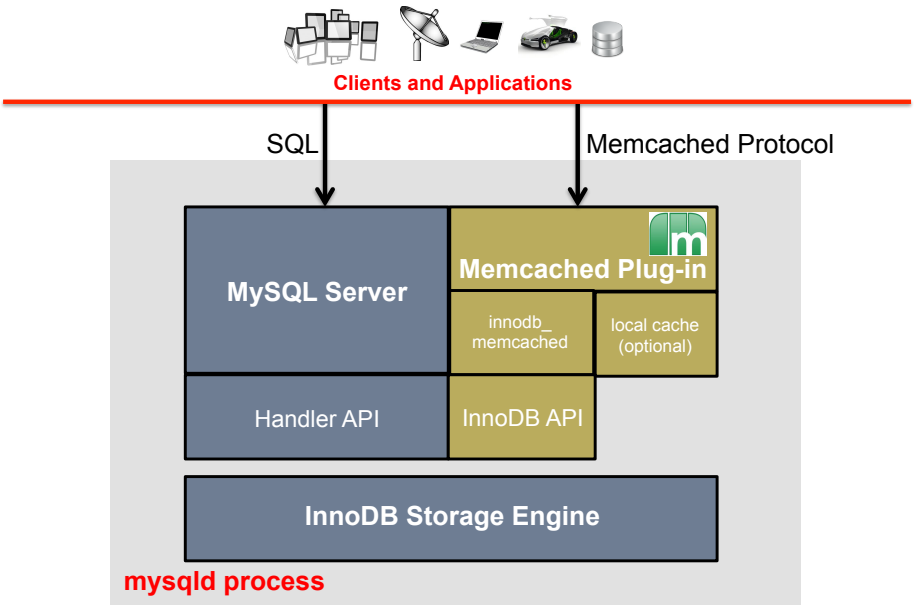


Figure 3: Memcached API Implementation for InnoDB

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 InnoDB tables via the memcached API 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, Full Text Search, XA transactions and complex JOIN operations.

Additional Memcached API features provided 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.

Benchmarks demonstrate that the NoSQL Memcached API for InnoDB delivers up to 9x higher performance than the SQL interface when inserting new key/value pairs, with a single low-end commodity server[4] supporting nearly 70,000 Transactions per Second.

[4] The benchmark was run on an 8-core Intel server configured with 16GB of memory and the Oracle Linux operating system.

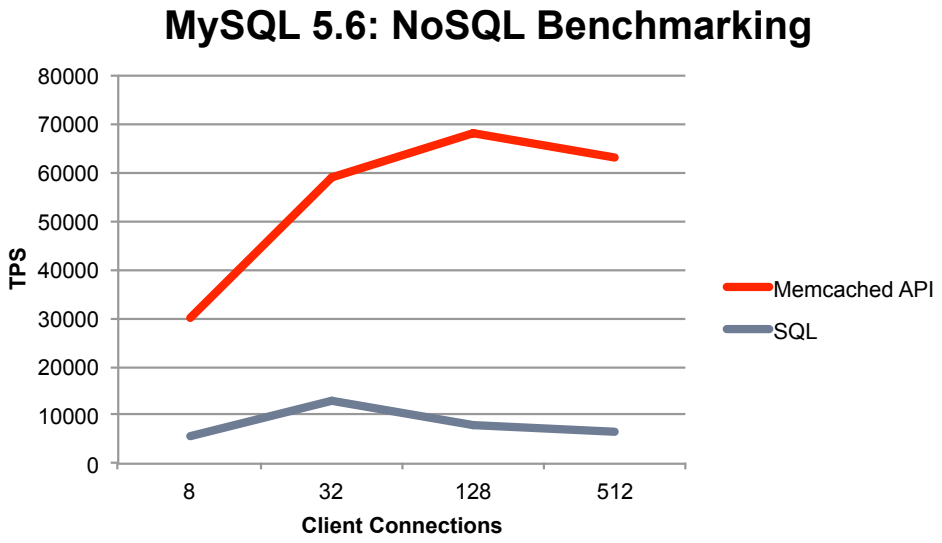


Figure 4: Over 9x Faster INSERT Operations

The delivered performance demonstrates MySQL with the native Memcached NoSQL interface is well suited for high-speed inserts with the added assurance of transactional guarantees.

For the latest Memcached / InnoDB developments and benchmarks, check out:
https://blogs.oracle.com/mysqlinnodb/entry/new_enhancements_for_innodb_memcached

You can learn how to configure the Memcached API for InnoDB here:
http://dev.mysql.com/doc/refman/5.6/en/innodb-memcached-setup.html

Memcached Implementation for MySQL Cluster

Memcached API support was introduced with MySQL Cluster 7.2, and joins an extensive range of NoSQL interfaces that are already available for MySQL Cluster (discussed later in this Guide).

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.

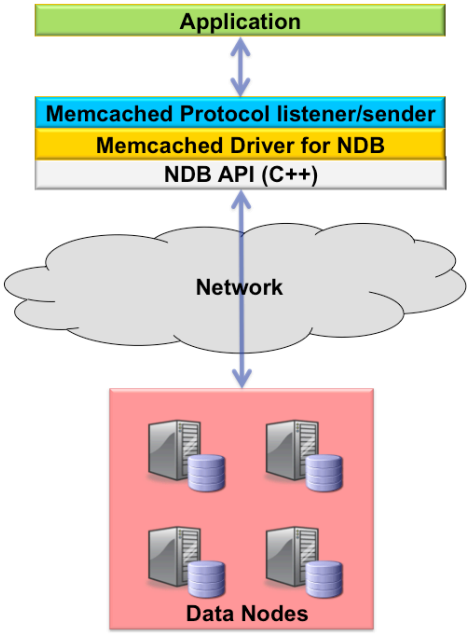


Figure 5: Memcached API Implementation with MySQL Cluster

Implementation is simple:
1. The application sends reads and writes to the Memcached process (using the standard Memcached API).
2. This invokes the Memcached Driver for NDB (which is part of the same process)
3. The NDB API is called, providing 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 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 Memcached.

The benefit of this flexible approach to deployment 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 value is linked to a pre-defined column 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.

Scaling MySQL with Replication and Sharding

When it comes to scaling MySQL for highly demanding web applications with large data volumes and high write-rates, replication and sharding (or horizontal partitioning) are typically part of the solution. As discussed below, when using the MySQL Server with InnoDB, replication can be configured asynchronously or semi-synchronously, and sharding is implemented at the application layer.

Using MySQL Cluster, discussed in the next section of this Guide, sharding is handled automatically at the database layer with support for active-active, multi-master replication, thereby providing an alternative model for applications requiring the highest levels of write throughput and availability.

MySQL Replication

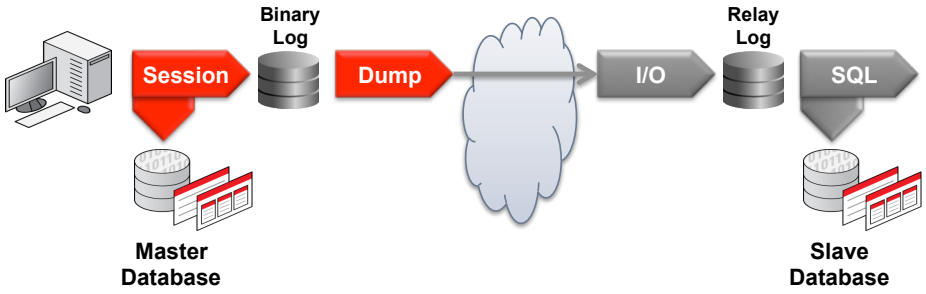
One of the ways many new applications achieve scalability is via replicating databases across multiple commodity nodes in order to distribute the workload.

For over ten years replication has been a standard and integrated component of MySQL, providing the foundation for extreme levels of database scalability and high availability in leading web properties including some of the world's most highly trafficked Web sites including Facebook, YouTube, Google, Twitter and Tumblr.

Replication enables MySQL to copy changes from one instance to another (i.e. from the "master" to one or more "slave" instances). This is used to increase both the availability and scalability of a database, enabling MySQL to scale-out beyond the capacity constraints of a single system by distributing queries across the replication cluster.

MySQL Replication is implemented by the master logging changes to data (DML: INSERT, UPDATE and DELETE) and changes to object structures (DDL, i.e. ALTER TABLE, etc.), which are then sent and applied to the slave(s) immediately or after a set time interval (when using Time Delayed Replication).

The figure below represents the implementation of MySQL replication.



- Session thread: processes queries from the application – writes data to master database & associated events to binary log
- Dump thread: reads events from binary log and sends them to a slave
- I/O thread: receives replication events and stores them in slave's relay log
- SQL thread: reads replication events from slave's relay log and applies them to slave database

Figure 6: *MySQL Replication Workflow*

Replication is often employed in a scale-out implementation so that requests which simply "read" data can be directed to slave servers. This allows transactions involving writes to be exclusively executed on the master server. This typically leads to not only a performance boost but a more efficient use of resources, and is implemented either within the application code or by using the appropriate MySQL connector (e.g. the Connector/J JDBC or mysqlnd PHP drivers).

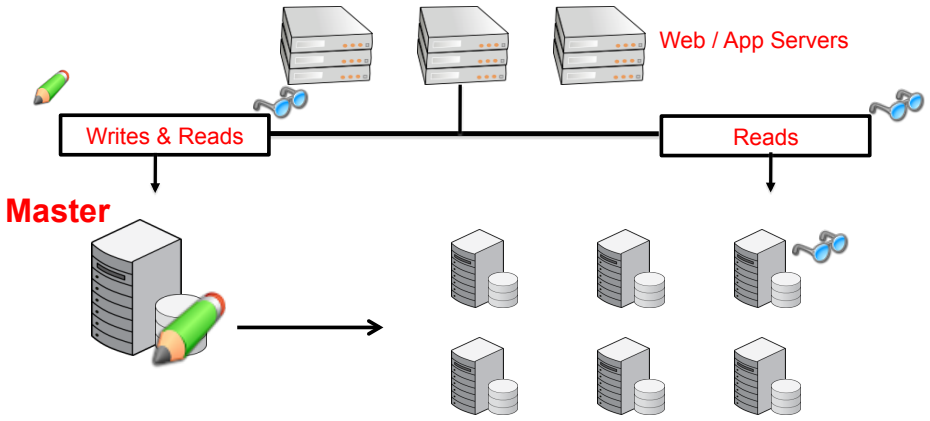


Figure 7: *Scaling Reads with MySQL Replication*

MySQL 5.6 includes a host of new capabilities that directly enhance replication performance, data integrity and availability, including:

- Global Transaction Identifiers which tag each replication transaction with a unique ID, making it simple to failover from a master to the most up-to-date slave, as well as create multi-master circular (ring) and n-tier (hierarchial) replication topologies;
- Multi-Threaded Slaves that improve performance by using multiple execution threads to apply replication events in parallel to slave servers;
- Crash-Safe Slaves which enable slaves to automatically recover from failures;
- Replication Event Checksums that protect the integrity of slave databases by automatically detecting corruption whether caused by memory, disk or network failures, or by the database itself;

Other key features include optimized row-based replication, binlog group commit, time-delayed replication and remote binlog backup, collectively enhancing performance, flexibility and ease of use.

MySQL replication can be deployed in a range of topologies to support diverse scaling and HA requirements. To learn more about these options and how to configure MySQL replication, refer to the following whitepaper: http://www.mysql.com/why-mysql/white-papers/mysql-replication-introduction/

Sharding

Sharding is commonly used by large web properties to increase the scalability of writes across their database. Sharding – also called application partitioning - involves the application dividing the database into smaller data sets and distributing them across multiple servers. This approach enables very cost effective scaling, as low-cost commodity servers can be easily deployed when capacity requirements grow. In addition, query processing affects only a smaller sub-set of the data, thereby boosting performance.

In high traffic Web and mobile environments such as social networking much of the content is generated by users themselves, thereby demanding high levels of write scalability, handling rapidly changing data. It must be remembered that reads will still predominate in these environments as typically each record will be read before an update is applied.

It is also important to emphasize that millions of users can still be serviced without requiring any database sharding at all, especially with the latest MySQL developments in scaling on multi-core commodity hardware and replication.

When using sharding, applications need to become “shard-aware” so that writes can be directed to the correct shard, and JOIN operations are minimized. If best practices are applied, then sharding becomes an effective way to scale relational databases supporting web-based write-intensive workloads.

Hashing against a single column (key) often proves the most effective means to shard data. In the Social Networking Reference Architecture we have sharded the user database by User ID, so Shard One handles the first third of users, the second shard handles the second third of users, and the remainder are stored in the third shard. An alternative approach used in some social networks is "functional sharding". In this scheme, each shard manages a different service or data type, with each shard storing the entire user base.

The approaches described above initially provide the most logical sharding mechanism and make it simple to scale-out as the service grows. However, it is likely that some shards will contain more active users than other shards; meaning re-balancing of shards may be required in the future. Both the application and the database need to be sufficiently flexible to accommodate change in the architecture. Implementing a sharding catalogue is a common way to achieve this. The central catalogue keeps track of data distribution and the application bases its sharding on this catalogue. This way it is easy to rebalance even individual data objects.

As sharding is implemented at the application layer, it is important to utilize best practices in application and database design. The Architecture and Design Consulting Engagement offered by the MySQL Consulting group in Oracle has enabled many customers to reduce the complexity of deployments where sharding is used for scalability.

By deploying MySQL with sharding and replication, leading social networking sites such as Facebook, Tumblr, Twitter and Pinterest have been able to exponentially scale their MySQL databases.

If applications involve high volumes of write operations coupled with the need for continuous operation, it is worthwhile also considering MySQL Cluster.

MySQL Cluster: Auto-Sharding, HA & Online Schema Maintenance

MySQL Cluster has many attributes that make it ideal for new generations of highly dynamic, highly scalable applications, including:

- Auto-sharding for write-scalability across commodity nodes;
- Cross-data center geographic synchronous and asynchronous replication;
- Online scaling and schema upgrades;
- SQL and NoSQL interfaces;
- Integrated HA for 99.999% uptime.

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.

MySQL Cluster is deployed in the some of the largest web, gaming, telecoms and embedded applications on the planet, serving high-scale, business-critical applications[10], including:

- High volume OLTP;

[10] http://mysql.com/customers/cluster/

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

MySQL Cluster Architecture

MySQL Cluster appears to clients as a single logical database. As illustrated in the figure below, MySQL Cluster actually 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, JavaScript, 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.

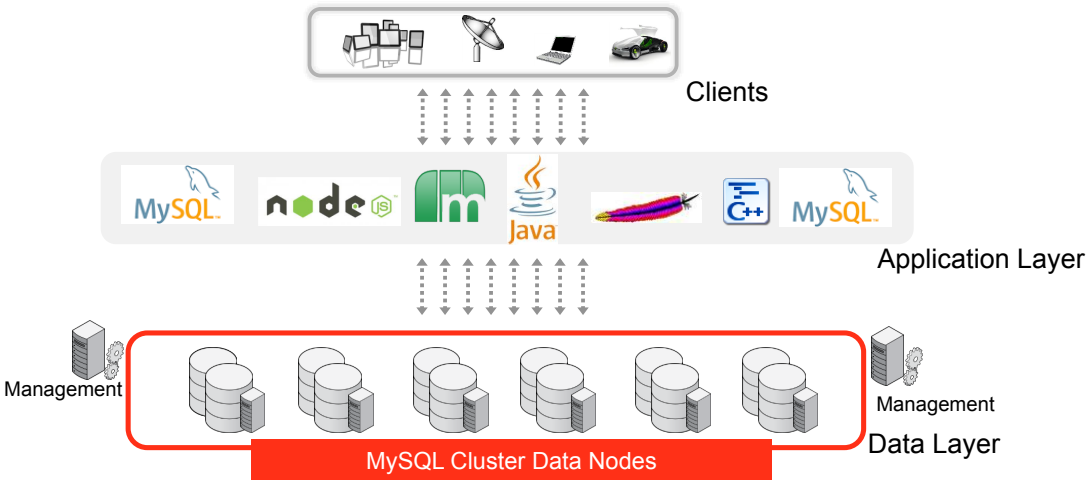


Figure 8: The MySQL Cluster architecture provides high write scalability across multiple SQL & NoSQL APIs

You can learn more about the architecture of MySQL Cluster from the Guide to Scaling Web Databases: http://www.mysql.com/why-mysql/white-papers/mysql_wp_scaling_web_databases.php

Auto-Sharding for High Write Scalability

MySQL Cluster is designed to support write-intensive workloads with the ability to scale across hundreds of nodes – allowing services to start small and expand quickly as demand takes-off.

MySQL Cluster is implemented as an active / active, multi-master database ensuring updates can be handled by any data node, and are instantly available to all of the other clients accessing the cluster.

Tables are automatically sharded across a pool of low cost commodity data nodes, enabling the database to scale horizontally to serve read and write-intensive workloads, accessed both from SQL and directly via NoSQL APIs.

By automatically sharding tables at the database layer, MySQL Cluster eliminates the need to shard at the application layer, greatly simplifying application development and maintenance. Sharding is entirely transparent to the application which is able to connect to any node in the cluster and have queries automatically access the correct shards needed to satisfy a query or commit a transaction.
By default, sharding is based on the hashing of the whole primary key, which generally leads to a more even distribution of data and queries across the cluster than alternative approaches such as range partitioning.

It is important to note that unlike other distributed databases, users do not lose the ability to perform JOIN operations or sacrifice ACID-guarantees when performing queries and transactions across shards. Adaptive Query Localization (AQL) pushes JOIN operations down to the data nodes where they are executed locally and in parallel, significantly reducing network hops and delivering 70x higher throughput and lower latency[12]. Referential integrity is maintained between tables with Foreign Keys. Those tables can live in different shards….even in different data centers!

As a result of technologies such as AQL, MySQL Cluster can serve real-time analytics across live data sets, along with high throughput OLTP operations. Examples include recommendations engines and clickstream analysis in web applications, pre-pay billing promotions in mobile telecoms networks or fraud detection in payment systems, such as those used by PayPal[13] who rely on MySQL Cluster to protect nearly $150bn of transactions per year.

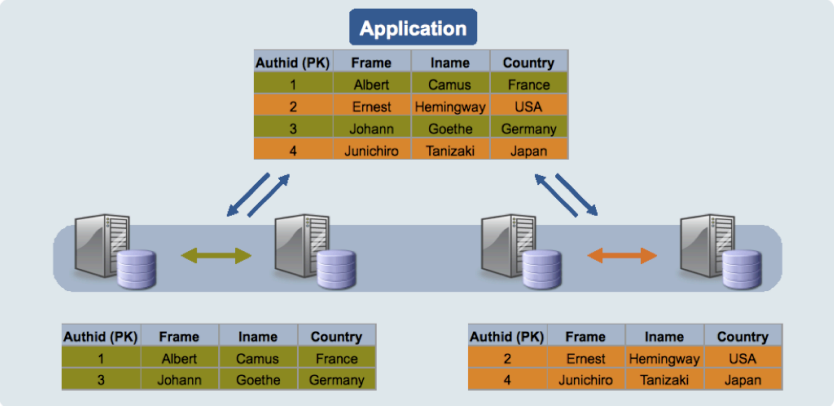


Figure 9: Auto-Sharding in MySQL Cluster, supporting cross-shard JOINs and Foreign Keys

Scaling Across Data Centers

Web services are global and so developers will want to ensure their databases can scale-out across regions. MySQL Cluster offers Geographic Replication that distributes clusters to remote data centers, serving to reduce the affects of geographic latency by pushing data closer to the user, as well as supporting disaster recovery.

Geographic Replication is implemented via standard asynchronous MySQL replication, with additional support for active / active geographically distributed clusters. Therefore, if your applications are attempting to

[12] http://dev.mysql.com/tech-resources/articles/mysql-cluster-7.2.html
[13] http://www.mysql.com/customers/view/?id=1223

update the same row on different clusters at the same time, MySQL Cluster's Geographic Replication can detect and resolve the conflict, ensuring each site can actively serve read and write requests while maintaining data consistency across the clusters.

Geographic Replication also enables data to be replicated in real-time to other MySQL storage engines. A typical use-case is to replicate tables from MySQL Cluster to the InnoDB storage engine in order to generate complex reports from near real-time data, with full performance isolation from the live data store.

MySQL Cluster also offers Multi-Site Clustering, providing additional options for cross data center scalability and fault tolerance. Data nodes can be split across data centers with synchronous replication between them. Failovers between sites are handled automatically by MySQL Cluster. This deployment model works best for data centers that are within the same geographic region and connected by high quality WAN links.

Whether you are replicating within the cluster, across data centers or between storage engines, all replication activities are performed concurrently – so users can combine scaling and High Availability (HA) strategies.

Benchmarking Scalability & Performance on Commodity Hardware

All of the technologies discussed above are designed to provide read and write scalability for your most demanding transactional web applications – but what do they mean in terms of delivered performance?

Benchmarks executed by Intel and Oracle demonstrate the performance advantages that can be realized by combining NoSQL APIs with the distributed, multi-master design of MySQL Cluster[14].

1.2 Billion write operations per minute (19.5 million per second) were scaled linearly across a cluster of 30 commodity dual socket (2.6GHz), 8-core Intel servers, each equipped with 64GB of RAM, running Linux and connected via Infiniband.

Synchronous replication within node groups was configured, enabling both high performance and high availability – without compromise. In this configuration, each node delivered 650,000 ACID-compliant write operations per second.

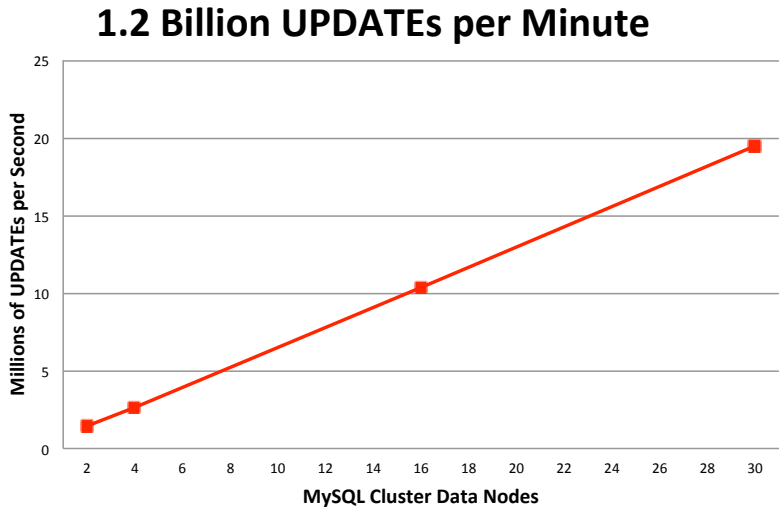


Figure 10: MySQL Cluster performance scaling-out on commodity nodes.

[14] http://mysql.com/why-mysql/benchmarks/mysql-cluster/

These results demonstrate how users can build a highly performing, highly scalable MySQL Cluster from low-cost commodity hardware to power their most mission-critical web services.

Scaling Operational Agility to Support Rapidly Evolving Services

Scaling performance is just one dimension – albeit a very important one – of scaling today's databases. As a new service gains in popularity it is important to be able to evolve the underlying infrastructure seamlessly, without incurring downtime and having to add lots of additional DBA or developer resource.

Users may need to increase the capacity and performance of the database; enhance their application (and therefore their database schema) to deliver new capabilities and upgrade their underlying platforms.

MySQL Cluster can perform all of these operations and more on-line – without interrupting service to the application or clients.

On-Line Schema Evolution

As services evolve, developers often want to add new functionality, which in many instances may demand updating the database schema.

This operation can be very disruptive for many databases, with ALTER TABLE commands taking the database offline for the duration of the operation. When users have large tables with many millions of rows, downtime can stretch into hours or even days.

MySQL Cluster supports on-line schema changes, enabling users to add new columns and tables and add and remove indexes – all while continuing to serve read and write requests, and without affecting response times.

Of course, if using the Memcached API, the application can avoid the need for data schemas altogether.

Unlike other on-line schema update solutions, MySQL Cluster does not need to create temporary tables, therefore avoiding the need to provision double the usual memory or disk space in order to complete the operation.

SQL and NoSQL Interfaces in MySQL Cluster

As MySQL Cluster stores tables in network-distributed 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.

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

At the extreme left hand side of the chart, 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 right hand side of the chart, MySQL presents a standard SQL interface to the data nodes, and provides connectivity to all of the standard MySQL connectors.

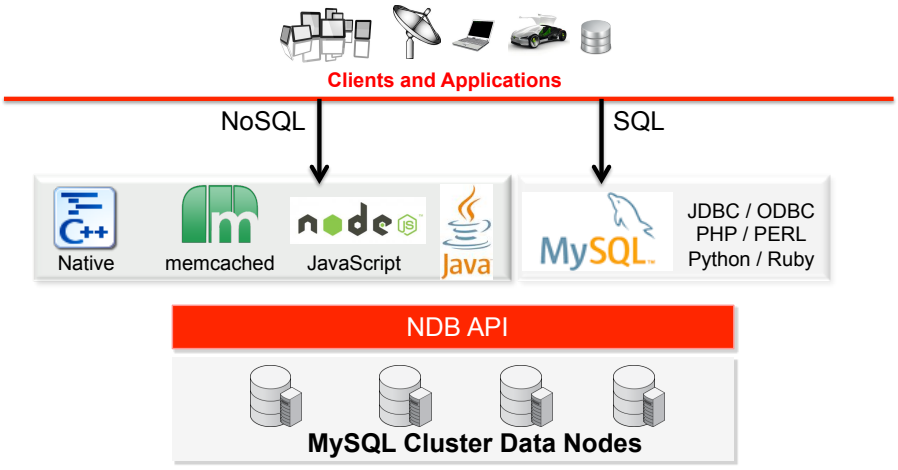


Figure 11: Ultimate Developer Flexibility – MySQL Cluster APIs

Whichever API is used to insert or query data, 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.

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 Big Data Platforms

Increases in data variety and volume are driving more holistic approaches to managing the data lifecyle – 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 heterogenous, with a requirement for simple integration between different data stores and frameworks, each deployed for a specific task.

As the leading open source database for web and cloud environments, MySQL has long offered comprehensive integration with applications and development environments. This integration is growing with a range of technologies and tooling that seamlessly enable integration between the MySQL database and Hadoop platform as part of a Big Data pipeline. These enable users to load or stream data into their Hadoop platform where it can be consolidated with data from other sources for processing, with result sets loaded back to MySQL to support decision making processes.

There are multiple approaches to integrating data between MySQL and Hadoop:

- Apache Sqoop (Batch, Bi-Directional)
- MySQL Applier for Hadoop (Real-Time, Uni-Directional)
- Custom methods (i.e. bash scripts)

The following figure shows MySQL integration with Hadoop, using the NoSQL APIs for high velocity data ingest, and then Sqoop and the Applier for Hadoop for data transfer.

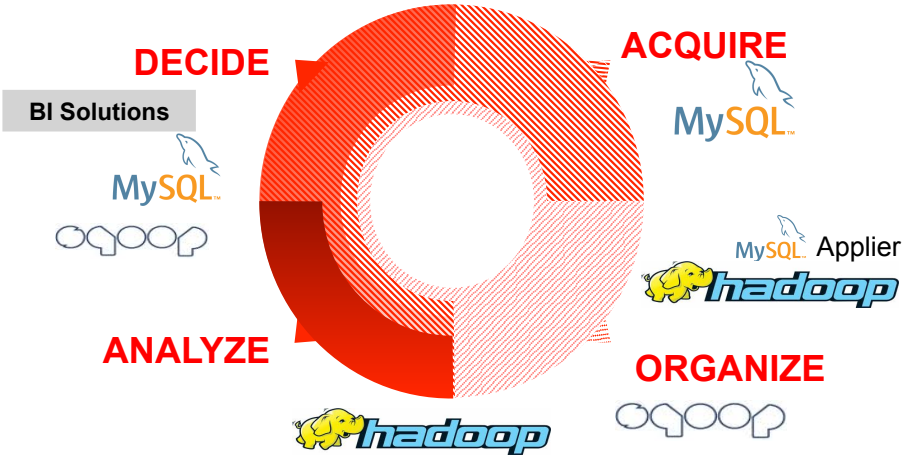


Figure 12: MySQL and Hadoop Integration

An overview of Apache and Sqoop and the MySQL Applier for Hadoop follow.

For more detail download the Guide to MySQL and Hadoop Integration:
http://www.mysql.com/why-mysql/white-papers/mysql_wp_hadoop.php

Apache Sqoop for Hadoop Integration

Apache Sqoop is a tool designed for efficiently transferring bulk data between Hadoop and structured datastores such as relational databases and data warehouses. Sqoop can be used to:

1. Import data from MySQL into the Hadoop Distributed File System (HDFS), or related systems such as Hive and HBase.
2. Extract data from Hadoop – typically the results from processing jobs - and export it back to MySQL tables.
3. Integrate with Oozie to allow users to schedule and automate import / export tasks.

Sqoop uses a connector-based architecture that supports plugins providing connectivity between HDFS and external databases. 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.

Apache Sqoop is a well-proven approach for bulk data loading. However, there are a growing number of use-cases for streaming real-time updates from MySQL into Hadoop for immediate analysis. In addition, the process of bulk loading can place additional demands on production database infrastructure, impacting performance.

MySQL Applier for Hadoop

The MySQL Applier for Hadoop is designed to address these issues by performing real-time replication of events between MySQL and Hadoop.

Replication via the Applier for Hadoop is implemented by connecting to the MySQL master reading events from the binary log[16] events as soon as they are commited on the MySQL master, and writing them into a file in HDFS. “Events” describe database changes such as table creation operations or changes to table data.

[16] http://dev.mysql.com/doc/refman/5.6/en/binary-log.html

The Applier for Hadoop uses an API provided by *libhdfs,* a C library to manipulate files in HDFS. The library comes precompiled with Hadoop distributions.

It connects to the MySQL master to read the binary log and then:

- Fetches the row insert events occurring on the master
- Decodes these events, extracts data inserted into each field of the row, and uses content handlers to get it in the format required
- Appends it to a text file in HDFS.

This is demonstrated in the figure below:

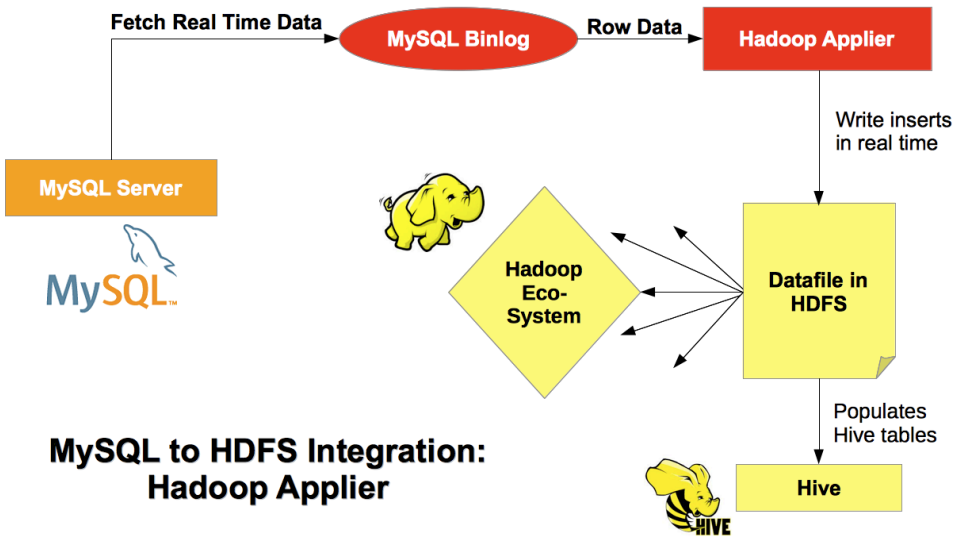


Figure 13: MySQL to Hadoop Real-Time Replication

Read more in the guide to MySQL and Hadoop Integration:
http://www.mysql.com/why-mysql/white-papers/mysql_wp_hadoop.php

Operational Best Practices

For business critical applications, it is recommended to rely on Oracle's MySQL Enterprise Edition or MySQL Cluster CGE, offering the most comprehensive set of advanced features, management tools and technical support to help you achieve the highest levels of MySQL scalability, security and uptime. MySQL Enterprise Edition and CGE include:

- **MySQL Database**
- **Oracle Premier Support for MySQL:** Staffed with MySQL Experts, Oracle's dedicated MySQL technical support team can help you solve your most complex database issues rapidly, and make the most of your MySQL deployments. Oracle MySQL Support Engineers are available 24/7/365 either online or by phone and have direct access to the MySQL developers. Oracle Premier Support does not include any limit on the number of support incidents, and you get support in 29 languages.
- **MySQL Enterprise Backup:** Reduces the risk of data loss by enabling online "Hot" backups of your databases.
- **MySQL Enterprise Scalability:** Enables you to meet the sustained performance and scalability requirements of ever increasing user, query and data loads. MySQL Thread Pool provides an

efficient, thread-handling model designed to reduce overhead in managing client connections, and statement execution threads.

- **MySQL Enterprise Security:** Provides ready to use external authentication modules to easily integrate MySQL with existing security infrastructures including PAM and Windows Active Directory.
- **MySQL Enterprise Audit:** Enables you to quickly and seamlessly add policy-based auditing compliance to new and existing applications. You can dynamically enable user level activity logging, implement activity-based policies, manage audit log files and integrate MySQL auditing with Oracle and third-party solutions
- **MySQL Enterprise High Availability:** Provides you with certified and supported solutions, including MySQL Replication, DRBD, Oracle VM Templates for MySQL and Windows Failover Clustering for MySQL.
- **MySQL Enterprise Monitor:** The MySQL Enterprise Monitor and the MySQL Query Analyzer continuously monitor your databases and alert you to potential problems before they impact your system. It's like having a "Virtual DBA Assistant" at your side to recommend best practices to eliminate security vulnerabilities, improve replication, optimize performance and more. As a result, the productivity of your DevOps and DBA team is improved significantly.
- **MySQL Cluster Manager:** Simplifies the creation and administration of the MySQL Cluster CGE database by automating common management tasks, including on-line scaling, upgrades and reconfiguration.
- **MySQL Workbench:** Provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, and much more.
- **MySQL Enterprise Oracle Certifications:** MySQL Enterprise Edition is certified with Oracle Linux, Oracle VM, Oracle Fusion middleware, Oracle Secure Backup, Oracle GoldenGate, and additional Oracle products.

Additional MySQL Services from Oracle to help you develop, deploy and manage highly scalable & available MySQL applications include:

Oracle University
Oracle University offers an extensive range of MySQL training from introductory courses (i.e. MySQL Essentials, MySQL DBA, etc.) through to advanced certifications such as MySQL High Availability and MySQL Cluster Administration. It is also possible to define custom training plans for delivery on-site.
You can learn more about MySQL training from the Oracle University here: http://www.mysql.com/training/

MySQL Consulting
To ensure best practices are leveraged from the initial design phase of a project through to implementation and sustaining, users can engage Professional Services consultants. Delivered remote or onsite, these engagements help in optimizing the architecture for scalability, high availability and adaptability. You can learn more at http://www.mysql.com/consulting/

Conclusion

The massive growth in data volumes is placing unprecedented demands on databases and frameworks. There are many new approaches that promise high scalability and availability with flexible development, rapid iteration and ease-of-use. As discussed, many applications also need to maintain transactional integrity of data, as well as the ability to run complex queries against their data to ensure business insight and competitive advantage.

Through developments in NoSQL interfaces, replication and MySQL Cluster, users can blend the best of NoSQL and relational data stores to provide a complete solution to application requirements, and enjoy the benefits of both technologies.