Guide to Scaling Web Databases with MySQL Cluster

Accelerating Innovation on the Web

A MySQL® White Paper

July 2011
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>MySQL Cluster Architecture</td>
<td>3</td>
</tr>
<tr>
<td>Scaling Write-Intensive Web Services</td>
<td>4</td>
</tr>
<tr>
<td>Auto-Sharding</td>
<td>5</td>
</tr>
<tr>
<td>Scaling Across Data Centers</td>
<td>6</td>
</tr>
<tr>
<td>Delivering a Real-Time User Experience</td>
<td>7</td>
</tr>
<tr>
<td>Benchmarking Performance on Commodity Hardware</td>
<td>8</td>
</tr>
<tr>
<td>Scaling Operational Agility</td>
<td>8</td>
</tr>
<tr>
<td>On-Line, On-Demand Scaling</td>
<td>9</td>
</tr>
<tr>
<td>On-Line Cluster Maintenance</td>
<td>9</td>
</tr>
<tr>
<td>On-Line Schema Evolution</td>
<td>10</td>
</tr>
<tr>
<td>Scaling Database Access: SQL &amp; NoSQL</td>
<td>10</td>
</tr>
<tr>
<td>SQL or NoSQL: Selecting the Right Interface</td>
<td>10</td>
</tr>
<tr>
<td>Schema-less Data Store with memcached API</td>
<td>12</td>
</tr>
<tr>
<td>Scaling with Continuous Availability</td>
<td>13</td>
</tr>
<tr>
<td>Resilience to Failures</td>
<td>13</td>
</tr>
<tr>
<td>Eliminating Planned Downtime</td>
<td>15</td>
</tr>
<tr>
<td>MySQL Cluster Carrier Grade Edition</td>
<td>16</td>
</tr>
<tr>
<td>MySQL Cluster Manager</td>
<td>17</td>
</tr>
<tr>
<td>Oracle Premier Support</td>
<td>17</td>
</tr>
<tr>
<td>Conclusion</td>
<td>17</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>17</td>
</tr>
</tbody>
</table>
Introduction

The realities of today’s successful web services are creating new demands that many legacy databases were just not designed to handle. These include:

- The need to automatically scale writes, as well as reads, both within and across geographically dispersed data centers;
- The need to scale operational agility to keep pace with demand. This means being able to add capacity and performance to the database, and to evolve the schema – all without downtime;
- The need to scale queries by having flexibility in the APIs used to access the database – including SQL and NoSQL interfaces;
- The need to scale the database while maintaining continuous availability.

9 of the top 10 most trafficked web properties on the planet including Facebook, Google, YouTube and Yahoo power their sites using MySQL. For example, Facebook achieves extreme scalability with MySQL and manages 750 million users¹. MySQL has helped Zappos become one of the most trafficked e-commerce sites with over $1B in sales. Google relies on MySQL to power Google AdWords.

This gives MySQL a unique insight into the challenges of scaling web databases, which in turn has driven the development of MySQL Cluster, integrating key technologies to enable the scaling of write-intensive web databases, including:

- Auto-sharding for write-scalability;
- Real-time responsiveness;
- Active / active geographic replication;
- Online scaling and schema upgrades;
- SQL and NoSQL interfaces;
- 99.999% availability.

This Guide explores the technology that enables MySQL Cluster to deliver web-scale performance with carrier-grade availability, and provides the resources to get you started in building your next successful web service.

MySQL Cluster Architecture

MySQL Cluster is a write-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 to serve read and write intensive workloads, accessed via SQL and NoSQL interfaces.

MySQL Cluster's real-time design delivers predictable, millisecond response times with the ability to service millions of operations per second. Support for in-memory and disk-based data, automatic data partitioning (sharding) with load balancing and the ability to add nodes to a running cluster with zero downtime allows linear database scalability to handle the most unpredictable web-based workloads.

Alcatel-Lucent, BT Plusnet, Cisco, Docudesk, Neckermann, Shopatron, Telenor, Zillow.com and many more deploy MySQL Cluster in highly demanding web, broadband and mobile communications environments for services including eCommerce and billing, user profile and session management, content management and caching, social networking, on-line gaming, location and presence services, messaging and collaboration, hosting, etc.²

¹ Based on Facebook estimates, July 2011
² http://mysql.com/customers/cluster/
MySQL Cluster is also a key component of the MySQL Web Reference Architectures—a collection of repeatable best practices for building highly scalable and available web services, developed with the world’s leading web properties. Deployment scenarios are provided for MySQL Cluster which powers the eCommerce, user profile management, sessions and look-up / shard catalogs within the Reference Architectures. The Web Reference Architectures are discussed in more detail later in this Guide.

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/JSON, C++ (NDB-API), Java, JPA and LDAP.
- Management nodes are used to configure the cluster and provide arbitration in the event of a network partition.

The roles played by each of these nodes in scaling a web database are discussed in the following sections of this Guide.

**Scaling Write-Intensive Web Services**

MySQL Cluster is designed to support write-intensive web workloads with the ability to scale from several to several hundred nodes using the technologies discussed in the following section of the guide. This means services can start small and scale quickly as demand takes-off.

---


4 Memcached API for MySQL Cluster is currently provided as a preview release.
Auto-Sharding

MySQL Cluster is implemented as an active/active, multi-master database ensuring updates made by any application or SQL node are instantly available to all of the other nodes 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. Up to 255 nodes are supported, of which 48 can be data nodes. As demonstrated later in this section, MySQL Cluster excels at both aggregate and per-node performance, eliminating the requirement to provision and manage racks of poorly utilized hardware.

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. Developers can also add "distribution awareness" to applications by partitioning based on a sub-key that is common to all rows being accessed by high running transactions. This ensures data that is used to complete transactions is localized on the same shard, thereby reducing network hops.

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. In the MySQL Cluster 7.2 Development Milestone Release, Adaptive Query Localization pushes JOIN operations down to the data nodes where they are executed locally and in parallel, significantly reducing network hops and delivering 20-40x higher throughput and lower latency\(^5\). This enables users to perform complex queries, such as real-time analytics, across live OLTP data sets, opening up a myriad of new possibilities to unlock greater value from user behavior and preferences.

![Figure 2: Auto-Sharding in MySQL Cluster](http://dev.mysql.com/tech-resources/articles/mysql-cluster-labs-dev-milestone-release.html)
The figure above demonstrates how MySQL Cluster shards tables across data nodes of the cluster.

From the figure below, you will also see that MySQL Cluster automatically creates “node groups” from the number of replicas and data nodes specified by the user. Updates are synchronously replicated between members of the node group to protect against data loss and enable sub-second failover in the event of a node failure. This is discussed in more detail in the “Scaling with Continuous Availability” section of the Guide.

The following figure shows how MySQL Cluster creates primary and secondary fragments of each shard. The user has configured the cluster to use four physical data nodes with two replicas, and so MySQL Cluster automatically created two node-groups.

![Diagram showing automatic creation of node groups and replicas]

**Figure 3: Automatic Creation of Node Groups & Replicas**

To ensure optimum performance, developers should always define a primary key, otherwise MySQL Cluster will create a hidden key. Even if the application doesn’t have a primary key, users should create an integer column for their table and auto-increment it. You can learn more about performance and tuning optimization of MySQL Cluster from the following whitepaper: [http://mysql.com/why-mysql/white-papers/mysql_wp_cluster_performance.php](http://mysql.com/why-mysql/white-papers/mysql_wp_cluster_performance.php)

**Scaling Across Data Centers**

While it is currently recommended that all data nodes of a cluster are located on the same local network, web services are global and so developers will want to ensure their databases can scale-out across regions. MySQL Cluster offers Geographic Replication which distributes clusters to remote data centers, serving to reduce the affects of geographic latency by pushing data closer to the user, as well as providing a capability for disaster recovery.

Geographic Replication is implemented via standard asynchronous MySQL replication, with one important difference: support for active / active geographically distributed clusters. Therefore, if your applications are attempting to 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.
Figure 4: Scaling Across Data Centers with Geographic Replication

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 real-time data, with full performance isolation from the live data store.

Whether you are replicating within the cluster, across data centers or between storage engines, all replication activities are performed concurrently – so users do not need to trade-off one scaling strategy for another.

Delivering a Real-Time User Experience

MySQL Cluster was originally designed for real-time applications and so kept all data in memory, with continuous Redo logging and check-pointing to disk for durability. Support for paging data out to disk was added in 2007, which increased the size of database that could be managed by MySQL Cluster far beyond the combined memory of all data nodes.

The original real-time design characteristics remain a central part of MySQL Cluster including:
- Data structures optimized for in-memory access, rather than disk-based blocks;
- Persistence of updates to logs and check-points on disk as a background, asynchronous process, therefore eliminating the I/O overhead that can throttle other databases;
- Binding of threads to CPUs to avoid CPU sleep-cycles.

Through the characteristics discussed above, MySQL Cluster is able to deliver consistently low levels of latency as users interact with the web service in real time, for example delivering status updates as soon as they occur, without duplicates or reappearances.

Users also have the flexibility of tuning the level of durability by de-configuring check-pointing and logging. Tables stored in this way would still survive a node failure (as they are synchronously replicated to other nodes in the node group), but not a failure of the entire cluster – for example a power failure – unless they were geographically replicated to a remote site. This mode of operation is commonly used in session management applications.
Response times are highly application and environment-dependent, but in recent OLTP tests\textsuperscript{6} using the TPC-C like DBT2 test suite, MySQL Cluster delivered an average latency of just 3 millisecond across SQL queries – and would have been faster if using one of the NoSQL access methods which will be discussed in a following section of the Guide.

**Benchmarking 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?

The MySQL Cluster development team recently ran a series of benchmarks\textsuperscript{7} that characterized performance across eight commodity dual socket (2.93GHz), 6-core Intel servers, each equipped with 24GB of RAM, running Oracle Linux.

As seen in the figure below, MySQL Cluster delivered just under 2.5 million updates per second and over 4.25m reads per second with two data nodes configured per physical server. Synchronous replication between node groups was configured, enabling both high performance and high availability – without compromise.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{mysql_cluster_performance.png}
\caption{MySQL Cluster performance scaling-out on commodity nodes.}
\end{figure}

Across 16 Intel servers, MySQL Cluster achieved just under 7 million queries per second.

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**

Scaling performance is just one dimension – albeit a very important one – of scaling a web database. As a web 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.

\begin{itemize}
\item \textsuperscript{6}http://www.clusterdb.com/mysql-cluster/mysql-cluster-database-7-performance-benchmark/
\item \textsuperscript{7}http://mysql.com/why-mysql/benchmarks/mysql-cluster/
\end{itemize}
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. These capabilities are discussed in the following section of this guide.

**On-Line, On-Demand Scaling**

MySQL Cluster allows users to scale both database performance and capacity by adding Application and Data Nodes on-line, enabling users to start with small clusters and then scale them on-demand, without downtime, as a service grows. Scaling could be the result of more users, new application functionality or more applications needing to share the database.

In the following example, the cluster on the left is configured with two application and data nodes and a single management server. As the service grows, the users are able to scale the database and add management redundancy – all of which can be performed as an online operation. An added advantage of scaling the Application Nodes is that they provide elasticity in scaling, so can be scaled back down if demand to the database decreases.

![On-Line Database Scaling](image)

**Figure 6: Doubling Database Capacity & Performance On-Demand and On-Line**

When new data nodes and node groups are added, the existing nodes in the cluster initiate a rolling restart to reconfigure for the new resource. This rolling restart ensures that the cluster remains operational during the addition of new nodes. Tables are then repartitioned and redundant rows are deleted with the OPTIMIZE TABLE command. All of these operations are transactional, ensuring that a node failure during the add-node process will not corrupt the database.

The operations can be performed manually from the command line or automated with MySQL Cluster Manager®, part of the commercial MySQL Cluster Carrier Grade Edition.

**On-Line Cluster Maintenance**

With its shared-nothing architecture, it is possible to avoid database outages by using rolling restarts to not only add but also upgrade nodes within the cluster. Using this approach, users can:

- Upgrade or patch the underlying hardware and operating system;

---

- Upgrade or patch MySQL Cluster, with full online upgrades between releases.

MySQL Cluster supports on-line, non-blocking backups, ensuring service interruptions are again avoided during this critical database maintenance task. Users are able to exercise fine-grained control when restoring a MySQL Cluster from backup using ndb_restore. Users can restore only specified tables or databases, or exclude specific tables or databases from being restored, using ndb_restore options --include-tables, --include-databases, --exclude-tables, and --exclude-databases.

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.

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

Scaling Database Access: SQL & NoSQL

As MySQL Cluster stores tables in data nodes, rather than in the MySQL Server, there are multiple interfaces available to access the database.

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 micro-second latency.

With this choice of interfaces, developers are free to work in their own preferred environments, enhancing productivity and agility, enabling them to deliver new services to market faster.

SQL or NoSQL: Selecting the Right Interface

The following chart 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.
At the extreme right 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 left 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 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.

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/JSON and memcached APIs;
- Enterprise applications with the ClusterJ and JPA APIs;
- Directory service using the LDAP API;
- Real-time web services (i.e. presence and location based) using the NDB API.

The following figure aims to summarize the capabilities and use-cases for each API.

---

**Figure 7: Ultimate Developer Flexibility – MySQL Cluster APIs**

**Figure 8: MySQL Cluster API Cheat-Sheet**
Schema-less Data Store with memcached API

As part of the MySQL Cluster 7.2 Development Milestone Release, Oracle announced the preview of native memcached Key/Value API support for MySQL Cluster – enabling direct access to the database from the memcached API without passing through the SQL layer. As this is the most recent NoSQL API for MySQL Cluster, and widely used in web properties around the world, the following section provides a more detailed discussion of its capabilities.

Like memcached, MySQL Cluster provides a distributed hash table with in-memory performance for caching. 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. Duplication of data between the cache and a persistent database can be eliminated, enabling simpler re-use of data across multiple applications, and reducing memory footprint.

Using the memcached API, users can simplify their architecture by compressing the caching and database layers into a single data tier, managed by MySQL Cluster, enabling them to:

- Preserve their existing investments in memcached by re-using existing memcached clients and without requiring application changes;
- Deliver higher write performance for update intensive applications;
- Simplify scale-out (both at the Memcached and MySQL Cluster layers);
- Improve uptime and availability.

![Figure 9: Memcached API Implementation with MySQL Cluster](http://www.clusterdb.com/mysql-cluster/scalable-persistent-ha-nosql-memcache-storage-using-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:

- Data that is written to and read from frequently would be best stored just in MySQL Cluster;
- Data that is rarely updated but frequently read would be best cached in memcached as well stored in MySQL Cluster;
- Data that has a short lifetime and wouldn’t benefit from being stored in MySQL Cluster would be stored only in memcached.

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.

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 with Continuous Availability**

No matter how well a web service is architected for performance, scalability and agility, if it is not up and available, it will never be successful.

Data and transactional states are usually the hardest parts of a web service to make highly available. Implementing a database that is itself highly available makes it simpler for the application to become highly available as well. This approach permits delegating the complexity of data management and transactional states to the database layer. The clear advantage of this design is that the database will always be the most competent, efficient and reliable mechanism in handling these duties when compared to other components within the system.

The architecture of MySQL Cluster is designed to deliver 99.999% availability, which includes both regularly scheduled maintenance operations, as well as systems failures (collectively known as “planned” and “unplanned” downtime). The Guide has already discussed many of the capabilities within MySQL Cluster to deliver high availability, and in this section, we explore them in more detail.

**Resilience to Failures**

The distributed, shared-nothing architecture of MySQL Cluster has been carefully designed to ensure resilience to failures, with self-healing recovery:

- MySQL Cluster detects any failures instantly and control is automatically failed over to other active nodes in the cluster, without interrupting service to the clients.
- In the event of a failure, the MySQL Cluster nodes are able to self-heal by automatically restarting, recovering, and dynamically reconfiguring themselves, all of which is completely transparent to the application.
- The data within a data node is synchronously replicated to all nodes within the Node Group. If a data node fails, then there is always at least one other data node storing the same information.
- In the event of a data node failure, then the MySQL Server or application node can use any other data node in the node group to execute transactions. The application simply retries the transaction and the remaining data nodes will successfully satisfy the request.
- Duplicate management server nodes can be deployed so that no management or arbitration functions are lost if a single management server fails.

Designing the cluster in this way makes the system reliable and highly available since single points of failure have been eliminated. Any node can be lost without it affecting the system as a whole. As illustrated in the figure below, an application can, for example, continue executing even though a Data Node is down, provided that there are one or more surviving nodes in its node group. Techniques used to increase the reliability and availability of the database system include:

- Data is synchronously replicated between all data nodes in the node group. This leads to very low fail-over times in case of node failures as there is no need to recreate and replay log files in order for the application to fail over.
- Nodes execute on multiple hosts, allowing MySQL Cluster to operate even during hardware failures.
- With its shared-nothing architecture, each data node has its own disk and memory storage, so a failure in shared storage does not cause a complete outage of the cluster.
- Single points of failure have been eliminated. Multiple nodes can be lost without any loss of data and without stopping applications using the database. Similarly, the network can be engineered such that there are no single points of failure between the interconnects.

![Diagram of MySQL Cluster](image)

**Figure 10:** With no single point of failure, MySQL Cluster delivers extreme resilience to failures

As demonstrated in the figure above, MySQL Cluster continues to deliver service, even in the event of catastrophic failures. As long as one data node from each node group and an application server remain available, the cluster will remain operational.

In addition to the site-level high-availability achieved through the redundant architecture of MySQL Cluster, geographic redundancy can be achieved using asynchronous replication between two or more Clusters. More information on Geographic Replication can be found in the “Scaling Across Data Centers” section of this Guide.
Eliminating Planned Downtime

Building on the shared nothing architecture discussed above, users are able to perform maintenance tasks to MySQL Cluster without having to bring the database down. These have already been discussed in the “Scaling Operational Agility” section of the guide, and are summarized below. All of these are on-line operations, meaning that query and insert, update and delete transactions continue to be processed throughout:

- Scale the cluster by adding new data, application and management nodes;
- Update the schema with new columns, tables and indexes;
- Re-sharding of tables across data nodes to allow better data distribution;
- Upgrade or patch the underlying hardware and operating system;
- Upgrade or patch MySQL Cluster, with full online upgrades between releases.

Through the capabilities described above, MySQL Cluster is able to eliminate both planned maintenance and unplanned downtime in order to deliver 99.999% availability demanded by web-based applications.

MySQL Web Reference Architectures

MySQL Cluster is a key component of the MySQL Web Reference Architectures – a collection of repeatable best practices for building highly scalable and available web services, developed with the world’s leading web properties.

The Reference Architectures profile four components common to most web properties (user authentication / session management; content management, ecommerce and analytics) and define the optimum deployment architecture for each. The reference architectures are categorized by “small”, “medium”, “large” and "extra large" (social networking) sites, based on the sizing and availability requirements for each environment, as demonstrated in the figure below.

<table>
<thead>
<tr>
<th></th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Extra Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries/Second</td>
<td>&lt;500</td>
<td>&lt;5,000</td>
<td>10,000+</td>
<td>25,000+</td>
</tr>
<tr>
<td>Transactions/Second</td>
<td>&lt;100</td>
<td>&lt;1,000</td>
<td>10,000+</td>
<td>25,000+</td>
</tr>
<tr>
<td>Concurrent Read Users</td>
<td>&lt;100</td>
<td>&lt;5,000</td>
<td>10,000+</td>
<td>25,000+</td>
</tr>
<tr>
<td>Concurrent Write Users</td>
<td>&lt;10</td>
<td>&lt;100</td>
<td>1,000+</td>
<td>2,500+</td>
</tr>
<tr>
<td>Database Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sessions</td>
<td>&lt;2 GB</td>
<td>&lt;10 GB</td>
<td>20+ GB</td>
<td>40+ GB</td>
</tr>
<tr>
<td>eCommerce</td>
<td>&lt;2 GB</td>
<td>&lt;10 GB</td>
<td>20+ GB</td>
<td>40+ GB</td>
</tr>
<tr>
<td>Analytics</td>
<td>&lt;10 GB</td>
<td>&lt;500 GB</td>
<td>1+ TB</td>
<td>2+ TB</td>
</tr>
<tr>
<td>Content Management</td>
<td>&lt;10 GB</td>
<td>&lt;500 GB</td>
<td>1+ TB</td>
<td>2+ TB</td>
</tr>
</tbody>
</table>

Figure 11: Sizing for the “Large” Web Reference Architecture

---

The Large Web Reference Architecture is shown below, with MySQL Cluster powering the Session Management and eCommerce components, while MySQL 5.5 with InnoDB and memcached are deployed for content management.

![Diagram of MySQL Cluster powering Session Management and eCommerce](image)

**Figure 12: MySQL Cluster powering Session Management and eCommerce**

With 4 x data nodes, MySQL Cluster is supporting 6,000 sessions (page hits) a second, with each page hit generating 8 – 12 database operations.

By using the scale-out capabilities of MySQL Cluster, it is possible to combine the session management and ecommerce databases onto one larger cluster.

Master / Slave replication is used to scale-out the Content Management and Analytics components of the web property. The content management architecture would be able to scale to 100k+ concurrent users with around 30 slaves – dependent on traffic patterns of the workload.


**MySQL Cluster Carrier Grade Edition**

MySQL Cluster Community Edition is licensed under the GPL and freely downloadable for development and deployment.
In addition, Oracle also offers the MySQL Cluster Carrier Grade Edition (CGE) commercial subscription\(^{11}\) and licenses that includes the most comprehensive set of MySQL production, backup, monitoring, modeling, development and administration tools and services so businesses that are using MySQL Cluster can achieve the highest levels of performance, reliability, security and uptime.

Key components of MySQL Cluster Carrier Grade Edition include MySQL Cluster Manager and Oracle Premier Support.

**MySQL Cluster Manager**

MySQL Cluster Manager\(^{12}\) simplifies the creation and management of MySQL Cluster CGE by automating common management tasks. As a result, developers, DBAs and Systems Administrator are more productive, enabling them to focus on strategic IT initiatives that support the business and respond more quickly to changing user requirements. At the same time risks of database downtime, which previously resulted from manual configuration errors, are significantly reduced.

**Oracle Premier Support**

MySQL Cluster CGE provides 24x7x365 access to Oracle’s MySQL Support team, which is staffed by seasoned database experts ready to help with the most complex technical issues. Oracle’s Premier support provides you with:

- 24x7x365 phone and online support;
- Rapid diagnosis and solution to complex issues;
- Unlimited incidents;
- Emergency hot fix builds;
- Access to Oracle’s MySQL Knowledge Base;
- Consultative support services.

**Conclusion**

To be truly successful, a web service needs to scale in multiple dimensions:

- Performance (throughput and latency);
- Operational agility;
- Data access interfaces;
- Availability.

With auto-sharding, active / active 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.

This Guide has been designed to provide an overview of these capabilities, and the resources below will enable you to learn more in building out your next successful web service with MySQL Cluster.

**Additional Resources**


MySQL Cluster Manager Trial (see the Resources section):

\(^{11}\) [http://www.mysql.com/products/]

\(^{12}\) [http://www.mysql.com/products/cluster/mcm/]
http://mysql.com/products/cluster/mcm/

On-Line Demonstration - MySQL Cluster in Action:
http://www.mysql.com/products/cluster/cluster_demo.html

Whitepaper: MySQL Web Reference Architectures:

Whitepaper: MySQL Cluster - Architecture and New Features:

MySQL Cluster in the Web – Case Studies, On-Demand Webinars & Whitepapers:

Contact MySQL Sales:
https://mysql.com/about/contact/sales.html