A Guide to the MySQL Alternative

If you are a developer or DBA who has built a rapidly growing application on MySQL, it is now time for you to investigate a scalable, cost-effective alternative. Read this guide to understand the common methods of scaling MySQL and why you should examine NuoDB as a simpler, more scalable path to web-scale. We'll make it easy for you to migrate.

MySQL Pain Points:

- MySQL client-server architecture does not scale out to meet the requirements of Internet and mobile apps.
- Large-scale MySQL deployments are complicated to set up and expensive to operate.
- MySQL async binlog replication delays can cause consistency faults and compromise high availability.
- MySQL application-level sharding is hard to use, error-prone, and forces developers to reinvent query and transaction processing logic.

Scale Out… And In… No Sharding Required

With the adoption of Internet applications like Facebook, Twitter and thousands of others, unprecedented volumes of data are being generated. This sea change has driven software developers and database administrators to seek out new database management systems (DBMS) capable of easily handling all this data.

Established companies with web-scale apps have developed customized systems to handle this exponential growth and the corresponding need to scale their databases. For example, Facebook was initially built as a single monolithic database with MySQL and sharded over 4,000 times for scalability. Thousands of servers and storage are required for all these shards. Industry estimates have put the cost to support the Facebook infrastructure at somewhere north of $1 Billion.

If you are a developer or DBA who has built a rapidly growing application on MySQL, but you're short that $1 Billion, is now the time for you to investigate a scalable, cost-effective alternative?

This guide explains the common methods of scaling MySQL and why you should examine NuoDB as a simpler, more scalable path to web-scale. We'll make it easy for you to migrate.

Scaling MySQL

Many apps are built using a programming language like PHP, Ruby, Python, C#, or Java and the MySQL relational database management system (RDBMS). A load-balancing front-end directs traffic to a pool of web and applications servers, that in turn connect to a group of one or more MySQL databases for insert, update and delete read-write operations, and usually a larger group of caching servers for read-only access. For fault-tolerance and high availability, MySQL servers are usually configured as master-slave or multi-master replicas and along with the other servers are hosted in multiple datacenters run by Amazon, Google, Rackspace or another public or private infrastructure provider, as shown in the diagram below.

Fundamentally, MySQL is a client-server database designed in the 1990s to run on a single server host. It was simply not designed to scale out and run as a fully-distributed DBMS. As a result, the basic architecture does not meet the throughput, latency and elastic performance requirements of Internet-scale applications.

The CPU, memory and I/O limitations of a single server host place a hard upper limit on the workload capacity of each MySQL instance. Performance can scale up only to the point where it hits the wall imposed by these hard resource limits.

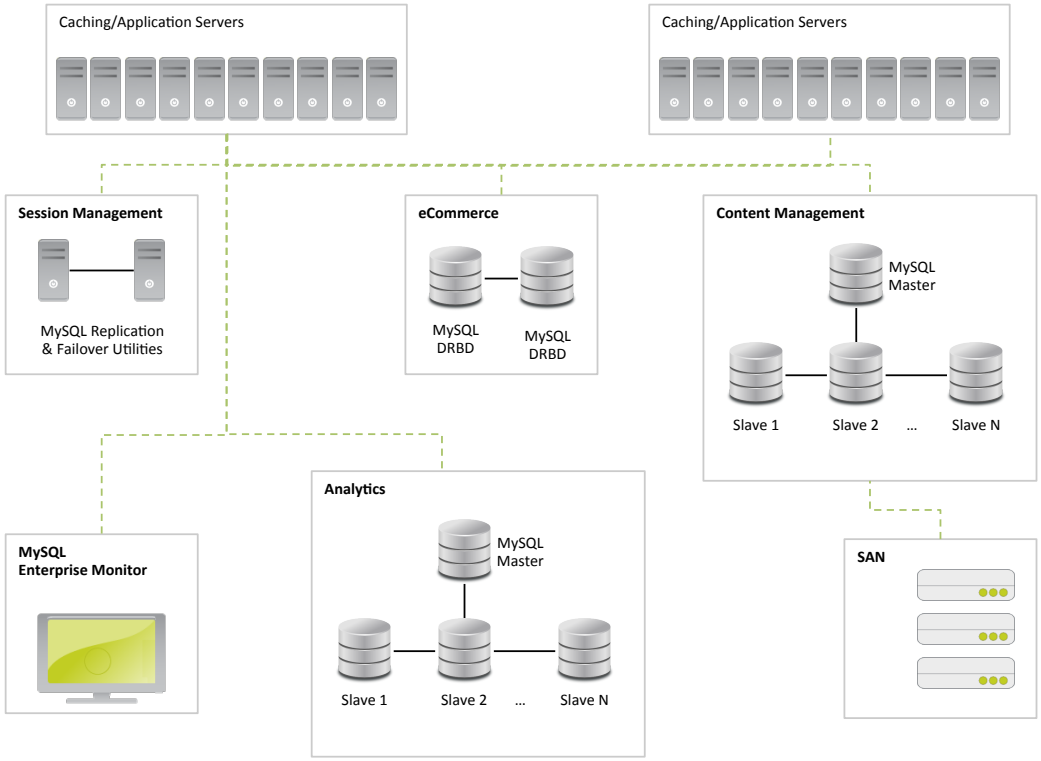

Figure 1: Typical sharded MySQL deployment architecture

To shard or not to shard, that is the question

Sharding across multiple hosts can solve some of the performance limitations of client-server DBMS. But when sharding is done in application layer code, as it is with MySQL, it opens a Pandora's box of transaction processing complexity and other complications. Those all fall squarely into the laps of developers instead of being handled in the database and storage infrastructure layers.

We confess we are not fans of sharding. Our objections include:

1. **It doesn't work in the general case**. It's easy to contrive simple cases in which a database can be readily partitioned into several databases and "scale-out" can be achieved. A good example are single-record Primary Key lookups/updates. If all you want to do is GET/PUT a record in the database system then that would work. But the moment you want to run a QUERY/UPDATE that touches rows in multiple partitions things start getting very difficult. It is probably very slow and the results are non-transactional. When someone tells you they have a sharded database solution that goes very fast you should take careful note of exactly what workloads they are running against exactly what schema.

2. **Sharding locks you into a solution to yesterday's problem.** Today most organizations do agile development, and it is a tenet of "agile" that you simply don't know what the system might have to do this time next year. Limiting the range of database interactions by sharding the system is a pretty harsh up-front decision. It's ironic that at a time when application developers place a high priority on flexible schemas and other don't-need-to-plan-ahead requirements for modern database systems, sharding is still seen as a solution to scaling a database.

3. **You no longer have a single consistent database.** Not only are there multiple databases in terms of storage, but the state of the database is not transactionally managed. Or if it is transactionally managed then you have to use 2-phase commit protocols, which are both slow and likely to freeze out other activity in the system for the duration of the transaction. Lack of transactional consistency creates major downstream costs in relation to data errors and recovery. And of course it increases application development and maintenance costs.

4. **A sharded system has N-times more complexity of administration**. You now have to have N-times more fail-over machines which have to handle complex partial-failover scenarios. You have N-times more backups, which again are not likely to represent any actual global state of the system at any known point in time. You have to upgrade, optimize, tune, modify schemas, etc. for N machines. You also have to manage security on N machines. Some people enjoy that stuff, but that doesn't make it optimal. Consider what needs to happen when N shards are not enough – the sharding algorithm would need to change in careful orchestration with a data migration activity.

5. Cost, cost, cost! The future is surely about systems that require less and less maintenance. We want to build systems in which a single administrator can manage thousands of databases. This already exists for the Web server layer of the stack, the application server layer, the storage layer, the network layer, etc. Sharding takes the database layer in the opposite direction. For reasons outlined in (4) above, sharded database solutions are very costly to maintain. Like most costly things, you should only do this if you have to. But if you don't have to then it makes no sense.

What is the MySQL alternative?

There are more modern alternatives to MySQL in the database world that were specifically architected for web-scale. First, there are the NoSQL data stores with claims of schema-less developer agility and scale-out performance. These systems do scale and are suitable for applications that do not require strong transactional consistency. However developers using NoSQL must undertake the effort to rewrite applications using proprietary APIs for handling queries and hierarchical data definitions. Sometimes this means reinventing query and transaction processing in the application code. These systems also have the disadvantage of not being able to use many of the tools and apps in the SQL software ecosystems; integration capabilities that are critical for many enterprise applications.

Before switching to NoSQL consider the following. For example, how many new skills must your staff learn? How many training courses must you put them in? How many new tools will you have to acquire if you ditch your entire SQL legacy? These are numbers that the finance-minded executive will want to quantify.

Zero percent will support all the features usually found in SQL systems, such as data security, data integrity and high-level query languages. So time and money will have to be invested to address these points.

The other alternative is NewSQL. The immediate upside is NewSQL's comprehensive support for SQL syntax. This new breed of RDBMS is designed to bring the power and traditional benefits of the old relational model to new distributed architectures and improve the performance of relational databases at web-scale.

NewSQL DBMS offer all the scale-out features of NoSQL on commodity hardware with transactional guarantees, like ACID, that are missing in NoSQL. NewSQL solutions are also compatible with familiar technologies like ODBC and JDBC.

NewSQL joins the classic benefits associated with traditional RDBMS and the scale-out performance brought to the market by NoSQL vendors. The NewSQL solutions essentially offer the best of both worlds.

But like any choice, some of the NewSQL solutions require compromises as well.

These compromises include auto-sharding, limited implementation of SQL, high-availability and many more. It might be tempting to accept these compromises in return for scale, but that would be akin to taking one step forward and two steps back. For example, if a database auto-shards your data, does that enable you to avoid accounting for new shards in your application code? Do updates to table structures automatically

propagate to all of the shards? Sadly the answer is no. Many NewSQL solutions only take you partway and then leave you to your own devices to figure out the rest.

Capability	MySQL	MongoDB	NuoDB
Standard SQL structured query language. Compatible with a broad range of tools, frameworks and application integrations. Large community of skilled software engineers and database administrators. Avoids proprietary API lock-in.	✓	✗	✓
ACID Transactions. Ensures that updates do not get lost and other concurrent access anomalies are prevented. Reduces application layer complexity and development effort.	✓	✗	✓
Distributed MVCC. High performance non-blocking access for operational analytics workloads. Optimistic concurrency control reduces transaction wait times and avoids need for application layer caching. Eliminates delays for ETL and ability to report near real-time metrics and Key Performance Indicators.	✗	✗	✓
Elastic scale-out performance. Provision new database processes to add (or remove) capacity on demand and start processing queries within seconds.	✗	✗	✓
Programming languages and frameworks. Support for Java, Python, PHP, C/C++, C# and other common languages, LINQ, Ruby on Rails, Hibernate, Zend, Node.js and other frameworks.	✓	✓	✓
Flexible schemas. Define data using non-relational information models. Support derived table types for managing non-uniform, semi-structured data sets and inheritance hierarchies.	✗	✓	✓[1]
Side-by-side deployment configurations. Supports gradual transition from MySQL and avoids forklift upgrades. Compatible with MySQL replication solutions.	✓	Partial	✓

[1] Derived tables is a feature targeted for release in 2013.

Storage layer partitioning. Eliminates complex and error-prone application layer sharding. Prevents database anomalies and reduces development and maintenance costs. Increases IIOPS and database size capacity.	✕	✓	✓[2]
High Availability. Out of the box support for fault-tolerant configuration of database transaction and storage processing nodes. Avoids Single Points Of Failure.	Partial	Partial	✓
Geo Distribution. Provisioning of domains that span multiple data centers and cloud-hosting providers, quality of service and access control parameters for adapting performance to bandwidth and latency constraints of WAN environments.	✕	✕	✓[3]
Built-in system management tools. Graphical, web-based console and APIs for network and element level operation, administration, maintenance and provisioning of the database management system.	✓	✕	✓

Figure 2: Feature comparison table between MySQL, MongoDB and NuoDB

NewSQL Without Compromise

NuoDB leads the industry with a proven NewSQL solution to solve scaling challenges without sacrificing the safety and familiarity of SQL or making the compromises other solutions require. In addition to scale-out performance, it also offers zero downtime and geo-distributed data management. It's an operational DBMS to handle transactions, interactions and observations anywhere.

Many applications already written in MySQL but in need of sharding in order to scale will be better off continuing to use SQL. For example, those applications in which query selection predicates and aggregation clauses are non-trivial and aren't merely simple key-value (KV) lookups. Even in the case of single key lookups, database performance can be optimized to be on par with that of simpler KV data stores, as NuoDB already demonstrated on Google Compute Engine, Amazon Web Services and other cloud infrastructures.

[2] Partitioned storage manager is a feature targeted for release in 2013.
[3] Geo-Distribution is a feature scheduled for release in 2013. Currently available for early access.

The NuoDB approach blends the benefits of de facto and de jure standards for database access that are broadly adopted and supported by a wide variety of software applications and middleware.

For developers who have existing MySQL apps that have, or are about to, hit the performance wall of client-server DBMS architectures, and developers who are considering writing new web scale MySQL apps, NuoDB offers a less risky approach than NoSQL alternatives. NuoDB provides a migration path that preserves the desirable elements of extended relational databases and leaves behind the non-scalable aspects of legacy DBMS architectures.

Let's take a look at the architecture of an application using the NuoDB DBMS:

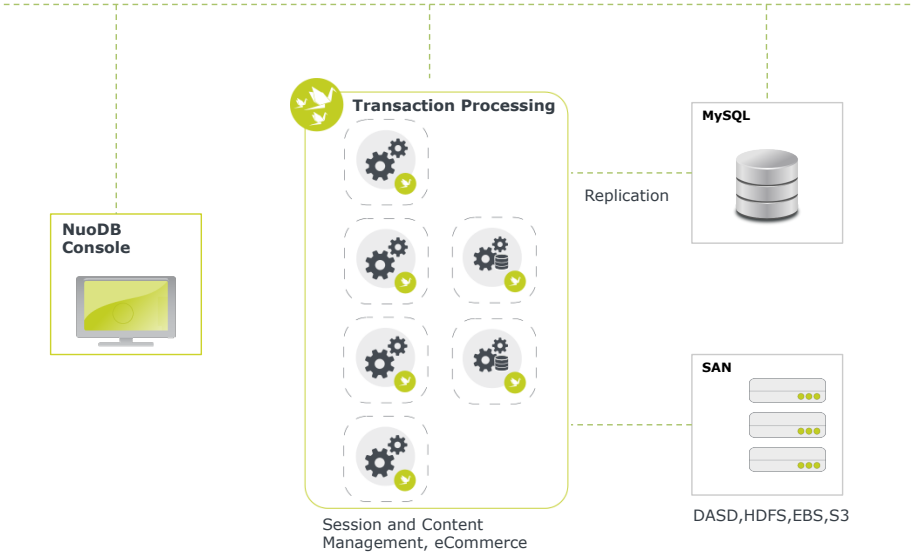

Figure 3: Typical NuoDB deployment architecture with MySQL data replication

Adding or removing NuoDB database transaction processing or storage management nodes in response to changes in application workload demand is straightforward and does not require complicated orchestration of provisioning and system management tasks, nor the lengthy startup times required to bring a new database server on-line. When additional NuoDB database processing nodes are started, they discover and peer with other NuoDB nodes and within a few seconds are ready to accept connections to run application queries. Compared to the MySQL deployment architecture, we've eliminated additional replication hardware and software components that are provisioned and standing idle for High Availability and all the host machines in catching layer used for off-loading read-only queries from MySQL servers. This streamlined, distributed DBMS architecture greatly simplifies the jobs of developers and operations staff.

Scale out…more… YCSB benchmark results

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-source performance evaluation tool and framework for cloud-based OLTP workloads. NuoDB recently demonstrated throughput of over 1.35 million transactions per second on YCSB workload B (5% writes, 95% reads), while maintaining sub- millisecond latencies, on a configuration of 32 NuoDB transaction engines running on Google Compute Engine (GCE) cloud infrastructure, as shown on the following chart:

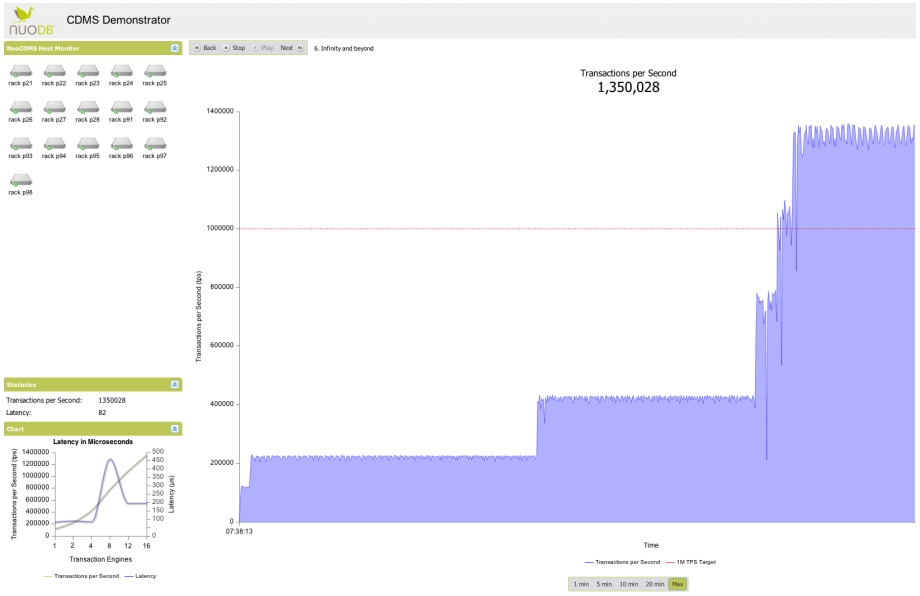

Figure 4: NuoDB YCSB benchmark results demonstrate throughput of 1.35M TPS

Another benchmark that illustrates NuoDB's linear scale-out of more complex queries and write-intensive workloads is DBT2, an open-source implementation of the TPC-C benchmark that simulates a traditional commerce OLTP workload. The following chart shows the performance of NuoDB running the DBT2 New Order transaction on a private cloud of commodity servers.

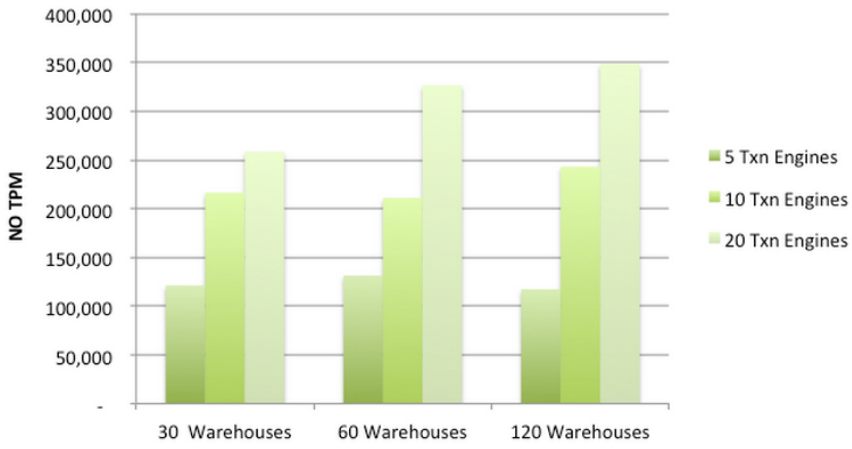

Figure 5: Performance results for New-orders measured in Transactions Per Minute [NOTPM] for 30, 60, and 120 Warehouses based on 5, 10, and 20 NuoDB transaction processes. All tests were performed with no simulated "think" time.

How big is big?

An ideal DBMS should scale elastically, allowing new machines to be introduced to a running database and become effective immediately.

NuoDB's distributed three-tier architecture is the foundation for just that kind of scale-out performance. It decouples management, transactions and storage, meaning each tier can scale a single, logical database elastically. It scales linearly to improve transactions per second performance and handle both concurrency and data volume. Out and in; by simply adding or removing processes.

Plus, NuoDB's geo-distributed data management lets you build an active/active, highly responsive database for high-availability and low latency. By bringing the database closer to your customers, they benefit from faster response times and you eliminate the need for complex replication, backup and recovery schemes. This functionality is the Holy Grail of databases.

Customers who would rather switch than shard

Here are examples of NuoDB customer applications that have been migrated from MySQL to NuoDB:

- **Social marketing sentiment analysis** and quantitative metrics for campaign optimization. Running on AWS, large multi-TB databases. Large working set read performance did not scale with MySQL.
- **Mobile Telecom - location sharing smartphone app.** Required very fast spin up and spin down of database nodes. MySQL monolithic architecture required too much time to configure and provision additional instances and could not meet SLA metrics during application demand spikes.
- **Internet – display advertisement serving.** Geo-distributed performance did not scale with MySQL. Considered NoSQL data stores, but did not want to give up SQL and transactional semantics.

Conclusion

Today's database market requires fast response, constant availability and the ability to scale to match app builders' dreams. The fact is that application loads vary a lot. So the ideal system scales dynamically, allowing new machines to be introduced to a running database and become effective immediately. On the other hand, a database application should not require even one dedicated machine if its load does not justify the cost in hardware, power, and heat of even a single machine.

At the same time, the system must provide the traditional guarantees: committed changes are durable, regardless of failures, even multiple failures. Concurrent transactions cannot overwrite each other's changes, even if they run on separate machines. Transactions must have a consistent view of their data and succeed or fail as a unit.

A database for this new market must run on hard iron, virtual machines, private clouds, public clouds, and across clouds. It must be designed for continuous operation.

That's a lot to ask for but it exists. Only NuoDB offers this unique combination of features. It's NewSQL without compromise.

For more information and a free download

- Download the free NuoDB developer edition at http://www.nuodb.com/register/
- Grab drivers from GitHub at http://nuodb.github.io/
- Visit our Developer Center at http://www.nuodb.com/devcenter/

215 First Street
Cambridge, MA 02142
+1 (617) 500-0001
www.nuodb.com