MongoDB vs. MySQL & ScaleBase:

A Comparison of Scalability, Data Distribution and Query Model

---

A ScaleBase Whitepaper

ScaleBase

info@scalebase.com
www.scalebase.com

Table of Contents

1. Introduction

Whether it is for extremely large data volumes, high transaction throughput or a massive number of concurrent users, today's internet and mobile applications require the ability to respond to a new level of performance and usability. Databases need to be flexible enough to quickly accommodate growth and changes as they arise.

A lot has been written to compare MongoDB and MySQL. Both are solid solutions for DBAs to use. Many developers and database professionals are investigating NoSQL solutions, and the most popular NoSQL database is MongoDB.

At a high level, MongoDB is a NoSQL document database. It is considered highly scalable and cloud-ready. MySQL is the leading open source relational database.

This whitepaper compares MongoDB and MySQL across three main areas:

1. Scalability, throughput and cloud-readiness.
2. Data distribution, reads / writes, joins and data rebalancing
3. Query model

Because, MySQL is not natively a distributed database, ScaleBase levels the playing field. ScaleBase is a distributed relational database platform that uses MySQL (or MariaDB or Percona Sever) for data storage. Adding ScaleBase into the MySQL equation makes MySQL a distributed database too, but it's still a relational database supporting SQL properties and ACID compliance.

MongoDB and MySQL, with ScaleBase, are both positioned to take software engineering teams and their applications to the next level of data scalability.

This comparison can help you understand which technology might be best for your new projects. Ultimately, data architects, dev-ops and developers need to choose the database technology that best meets their application's requirements.

2. Scalability, throughput and cloud-readiness

MongoDB and MySQL with ScaleBase each leverage several proven techniques when it comes to scaling out applications.

Auto-Sharding

A powerful technique that can greatly extend the scalability of a database is called "sharding." A lot has been written about sharding, and the difficulties inherent in homegrown, do-it-yourself horizontal data partitioning approaches, so this document won't cover those topics in detail again here.

As you may know, sharding is a process that splits large data sets on a single server into smaller data sets, or horizontal partitions, placed onto multiple servers. The goal of this division of the data is to *divide workloads* between multiple servers, so as to:

1) increase TPS throughput,
2) allow more concurrent users and
3) allow for much larger overall database size.

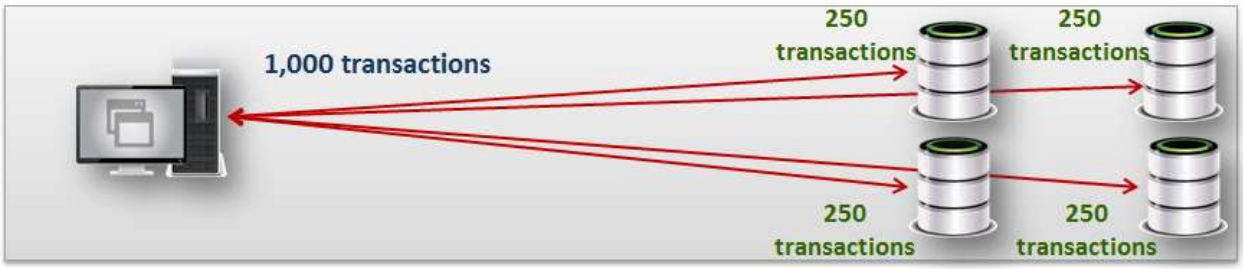

Figure 1 Scalability via auto-sharding

MongoDB Sharding

Sharding is an important part of how MongoDB achieves its scalability. In MongoDB, data is split between shards using a shard key. The shard key is a single field (or a set of composite fields) that identify the data to be sharded.

The shard stores a "replica set" of the application data. This is a subset of all the data and is identified by the particular shard key. MongoDB controls which replica set is stored on a particular shard.

If you look directly into an individual shard to see what it contains, you will see a random subset of the data. Which data replica is stored in which shard or partition is somewhat random to the user, and unknown to the application. That's why the client always needs to connect to the mongos and let MongoDB find the data you need.

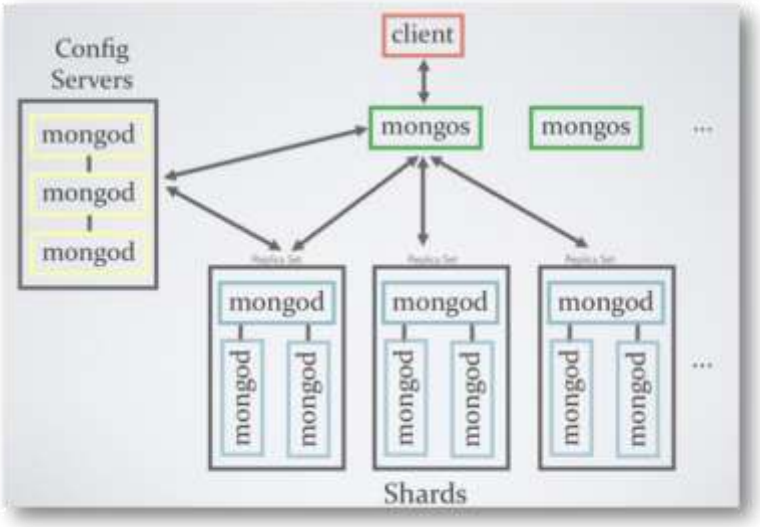

Figure 2 MongoDB Architecture (taken from the MongoDB web site)

In the MongoDB architecture diagram above, the configuration of the cluster is stored in the Config Servers (which are mongod databases). Queries, and data, going back and forth between the client and the cluster are sent through the MongoDB routers, mongos. The mongos will route a query to the appropriate backend database or databases.

Data distribution occurs at the database level – it's not defined by the application. The application actually "sees" a single database and MongoDB handles retrieving data from each shard partition (mongod). As a non-relational database, there are no database joins (this will be explored more later in this document), so joins are achieved in the application. Most of the query processing is done in the data layer with aggregations done in the MongoDB middleware.

ScaleBase Sharding

Now that we've reviewed sharding in MongoDB, you might be surprised to learn this is very similar to how MySQL with ScaleBase works. ScaleBase also automates sharding. Like MongoDB, ScaleBase's data distribution is handled at the database level (and not in the application). Importantly, because ScaleBase maintains a relational data environment, ScaleBase will ensure that data that is accessed together is also stored together.

Like MongoDB, with ScaleBase is "transparent" and the application "sees" a single database and not a bunch of shards/partitions. Like MongoDB, ScaleBase also provides agility, flexibility, and growth. Database optimization is dynamic. Rebalancing (adding and removing nodes to accommodate growth and to handle database "hotspots") is dynamic, and can be executed while the entire system is online. All this without changes to the application design or codebase.

However, there are two ways that ScaleBase's data sharding differs.

1. Retaining ACID / SQL

   With ScaleBase, joins are supported, even in a distributed environment. As MySQL is a relational database, ScaleBase embraces and retains data relations expressed in joins and transactions. With ScaleBase, queries can have local joins (within one shard) as well as cross-database joins. Unlike MongoDB where joins are accomplished in your application, ScaleBase will automatically aggregate data across shards for you. In this way, ScaleBase gives you ACID transactions and 2-phase commits across a sharded cluster of MySQL database nodes. This is unique.

2. MySQL Ecosystem

   The second way ScaleBase is a bit different is it preserves rich MySQL skills and ecosystem technologies. It provides this scalability to your existing MySQL applications, if they are growing beyond the capabilities of your server (or Amazon RDS instance). Maybe more importantly, ScaleBase gives new MySQL apps a modern, distributed and relational data framework with near-linear scalability.

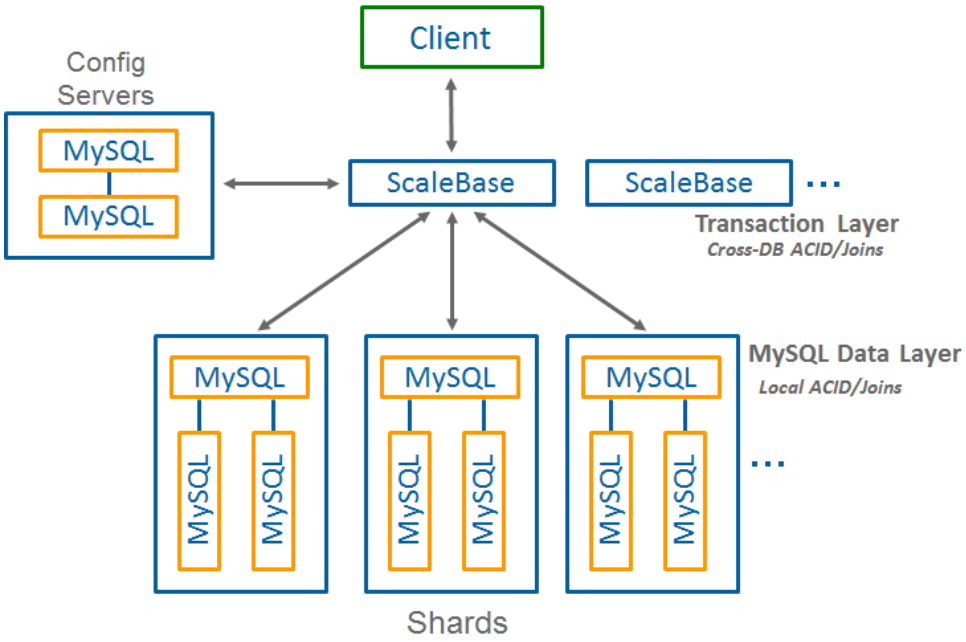

Figure 3 ScaleBase Architecture

Looking at a ScaleBase architecture diagram and relating back to MongoDB, the ScaleBase Config Servers here are the equivalent of the MongoDB mongod Config Servers. And ScaleBase Controller is the equivalent of the MongoDB router, the mongos. The similarities are striking.

3. Data Distribution

As a reminder, the goal of dividing data into horizontal partitions is to cause workloads to become distributed across multiple servers. This allows an application to experience higher TPS throughput, allow more concurrent users, and to allow for much larger overall database size, all while avoiding database bottlenecks.

For the best performance, any data that is logically related should be stored together. In this way, queries can be satisfied in a single fetch. But, that ideal situation isn't always available. What happens when a query needs data from multiple partitions? Let's take a look at a few example application scenarios and see how the *way* you distribute data can impact performance.

Reads and Writes in MongoDB and in MySQL with ScaleBase

Imagine a blogging application, with authors, articles, users, comments and tags. The table below identifies four typical blogging activities (scenarios) and indicates how frequently they typically happen.

#	Scenario	Frequency
1	Add article	High
2	Add tag	Low
3	Query articles with author user details	High
4	Query an article's comments with commenters user details	Low

Figure 4 Example Blogging Application Scenarios

As you can see, we have two "write" activities and two "read" activities. And, two scenarios occur frequently, and two occur less often.

Below are typical data models for this simple application: MongoDB uses typical document type store, and ScaleBase's typical relational store.\

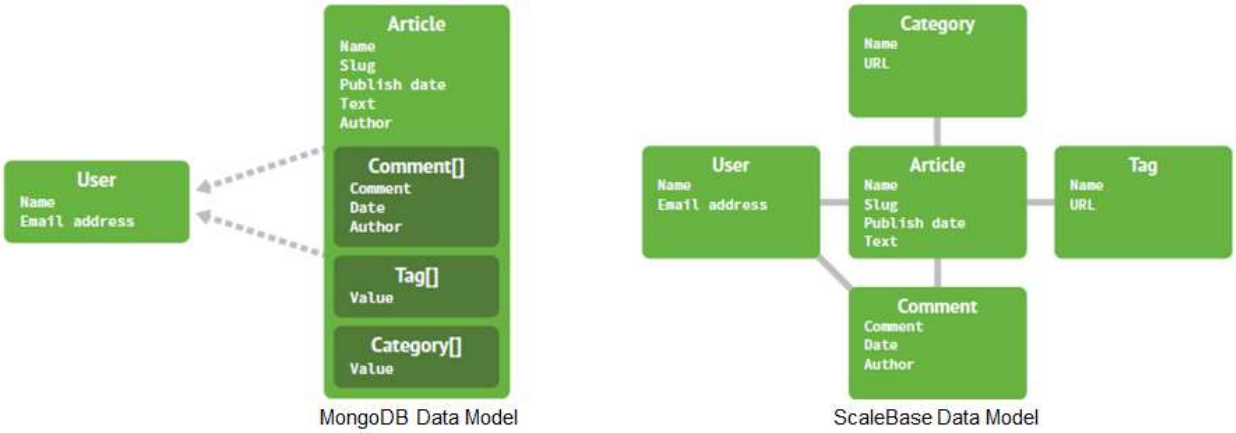

Figure 5 Example Blogging Application Data Models for MongoDB and ScaleBase

The model on the left is the MongoDB data model (represented via BSON) and the model on the right is a relational model. While the exact way that the data is stored is different, both models accomplish the same thing.

So, with the groundwork of our example understood, let's examine how MongoDB and ScaleBase execute the four blogging application scenarios we outlined earlier.

Scenario Execution with MongoDB – Joins are in the App

You can see that for the two "write" scenarios, in MongoDB, with that model, adding an article, a high frequency activity, and adding a tag, are both easily accomplished in a single call to a single database. For example, we already know the user so it is simply a matter of inserting the relevant documents.

However, both of the "read" scenarios require that a "join" be made in the application. For example, if we want to query all of an author's articles, along with that author's details, we would need to first request the Users collection (table) and then query the Article collection (table). The application would then have to join those two sets of data and return it to the client.

Depending on how the Article collection is sharded, MongoDB may need to query multiple shards and then aggregate that data before returning it to the blogging application, and for that data to then also be joined with the User collection.

#	Scenario	Frequency	Execution Steps	Local / Cross
1	Add article	High	Add Article document	Local
2	Add tag	Low	Update Article document	Local
3	Query articles with author user details	High	• Query Article document • Query User document (author)	Local + Local "Join" is made in app
4	Query an article's comments with commenters' user details	Low	• Query Article document • Query User (document commenter)	Local + Local "Join" is made in app

Figure 6. Read / Write Scenario Execution in MongoDB

Scenario Execution with ScaleBase with MySQL – Joins are in the Database

In ScaleBase with MySQL, the data may be distributed across articles, but the distribution process will co-locate logically related user-data with an author's articles. In cases where data is not co-located, ScaleBase will join the data at the *database* level. This means the application does not need to code those joins. In this way, three of the four scenarios are satisfied in a single call.

#	Scenario	Frequency	Execution Steps	Local / Cross
1	Add article	High	Add Article row and related rows	Local
2	Add tag	Low	Add Tag row	Local
3	Query articles with author user details	High	Query Article table joined with related tables	Local
4	Query an article's comments with commenter's user details	Low	• Local join of Article and Comment tables • Cross DB join of result and commenter User table	Cross-DB

Figure 7. Read / Write Scenario Execution in ScaleBase and MySQL

As you can see, both MongoDB and ScaleBase data models obviously can support our example Blogging application. In MongoDB, there can be read scenarios where a join needs to be accomplished in the *application*. In ScaleBase, for the one scenario that requires it, the join is accomplished by ScaleBase in the *database* layer, and not within the application.

Data Rebalancing in MongoDB, and ScaleBase with MySQL

There can come a time when you need to rethink how data is distributed across servers. For example, application usage patterns may evolve. And, as the number of concurrent users goes up and/or transaction and data volumes increase, you may find that certain data nodes, or partitions, become "hotspots" receiving more read/write workloads than other parts of the distributed database cluster. When this happens, you will want to adjust how data is distributed in the future, and possibly also *redistribute* existing data to achieve a more balanced distributed workload.

Data Chunks

In both MongoDB and ScaleBase, data is distributed in "chunks". In MongoDB, a data chuck is a set of documents and the default size for a data chunk 64 megabytes. In ScaleBase, a data chuck is a logical data slice from an entire relational hierarchy.

Splitting Shards and Rebalancing Partitions

When you want to alleviate the workloads from a database hotspot, there are two typical data redistribution use-cases: splitting partitions and rebalancing partitions.

They are easily explained in a few pictures.

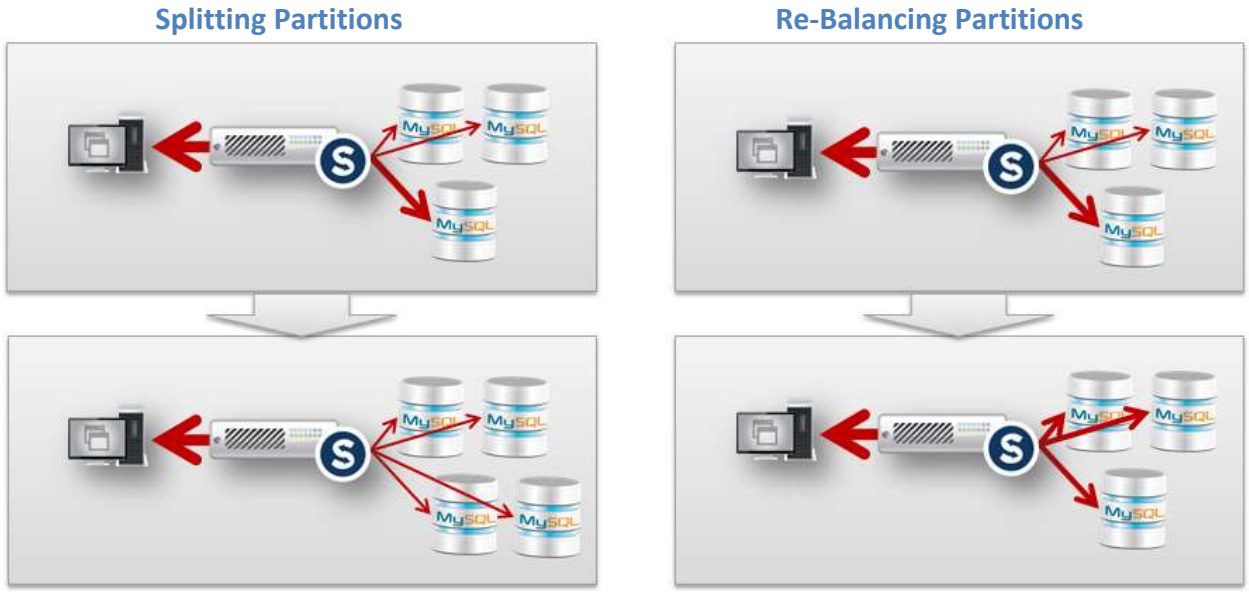

Figure 8. Data Redistribution: Splitting and Rebalancing Shards

Simply stated, in *splitting partitions* you are adding resources to your distributed database, whereas *in re-balancing partitions* you are redistributing data across already existing server resources.

Both MongoDB and ScaleBase can automatically redistribute data across partitions, online and without downtime, by adding and removing nodes as needed, to handle data volume changes and hotspots.

A new server might need to be added for more capacity, but it may also be possible to move "hot" data from an overused server node to an underused server node. Remember that in MongoDB, a chunk is the smallest logical set of data. When migrating between over- and under-used nodes, MongoDB moves entire chunks from one server to the other. Whereas, in ScaleBase, a data chuck is a logical data slice from an entire relational hierarchy. ScaleBase will similarly move entire chunks of data in a way that maintains logical data relations.

The best part, for both MongoDB users and ScaleBase users, is that the logic for data redistribution and rebalancing happens behind the scenes, automatically. data is moved using logic at the data layer. Once again, your application does not need to contain code for that functionality..

4. Query Model Comparison

Now let's compare query models, and see how MongoDB and MySQL with ScaleBase answer application calls and queries.

The Challenge – Aggregating query results from several database nodes

Efficiently satisfying queries in a distributed database system can be challenging. While accessing a single collection, or a document, is easy enough, building result-sets from data spread across multiple nodes and collections has traditionally involved a manual coding process.

At a high-level, we can say a query can be filtered either by ID (and return a single result) or a range (and return multiple results).

In a distributed database, the challenge is for a query to efficiently and accurately access several data partitions, where each partition gives a partial results, and then to aggregate results efficiently so as to provide one answer to the application and user.

At a high level we can say there are several *operations* that aggregate query results:

- Aggregate functions: count, min, max, sum
- Distinct
- Group
- Sort

The challenge is to execute these operations in a distributed data environment and so that the applications still "sees" one database.

Figure 9. A query in a distributed database environment

For example, for a distributed database with four partitions, a simple count of records across the entire database needs four queries, run in parallel on four databases, which results in four numbers that need to be summed up and returned to the application as a single summed number.

As I mentioned earlier, since MongoDB does not do joins, discrete docs need to be "joined" together inside the application after several roundtrips to the database.

Aggregations can make this even more complex, but both MongoDB and ScaleBase do most of the aggregation work for you. An aggregation is an operation that scans over a set of documents and returns computed values such as Sum, Average, or Count.

So, what options do MongoDB and ScaleBase offer to aggregate query results from across several database partitions?

MongoDB Query Aggregation Options

MongoDB provides two main methods of aggregation: the aggregation pipeline (and simple single purpose pipeline operations), and map-reduce.

MongoDB Aggregation Pipeline

The *aggregation pipeline* is a framework for aggregates built into MongoDB (since version 2.2). You can think of the pipeline framework as working something like the UNIX pipe command. You stream documents through the pipeline, filtering and transforming as needed on each operator.

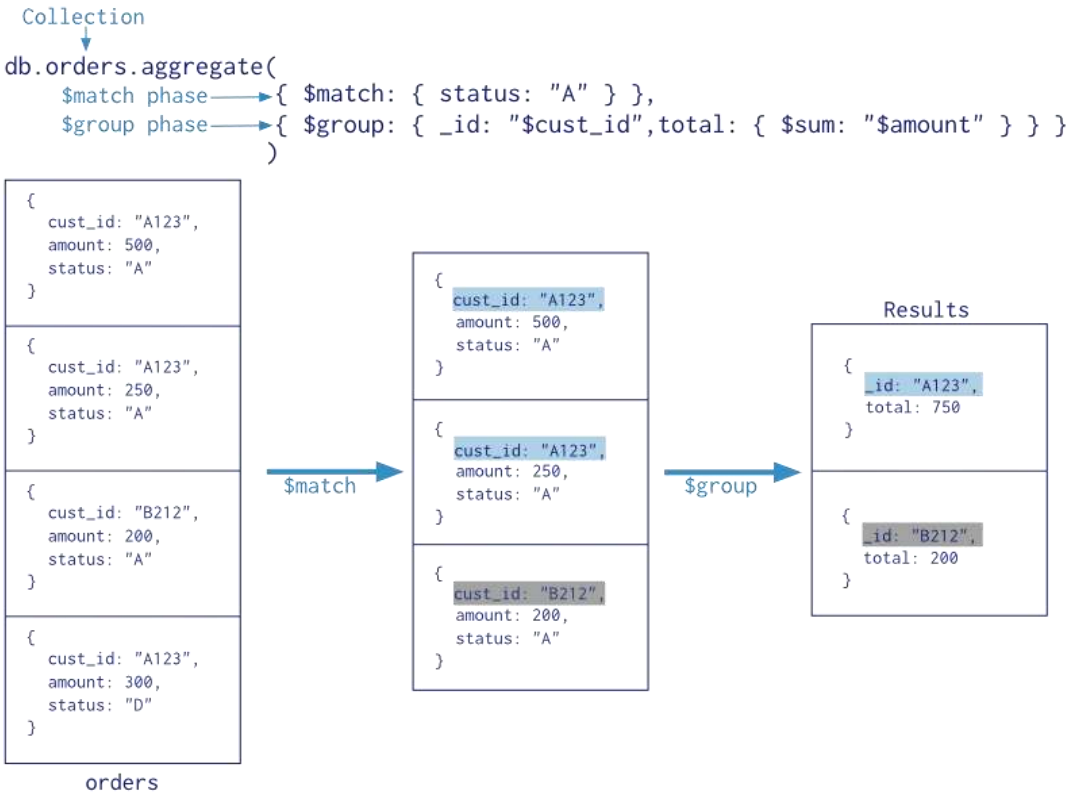

Figure 10. Annotated Aggregation Pipeline from Mongodb.com documentation http://docs.mongodb.org/master/_images/aggregation-pipeline.png

Each pipeline operator can do things like skip, match, sort and even geospatial style matching. You can improve performance by doing filtering (match) at the beginning of the pipeline, thus reducing

the amount of data being scanned and manipulated. The aggregation pipeline can use indexes that are available.

Single Purpose Aggregation Operations

The single purpose aggregation operations are very simple procedures that return a very specific set of data. Examples would be count on a single value, grouping, a distinct list of values, etc.

However, it's important to note that in MongoDB, the 'Group' function does NOT support distributed database nodes or partitions. This is a major difference. Secondly, all single purpose aggregation operation results must be less than 16MB.

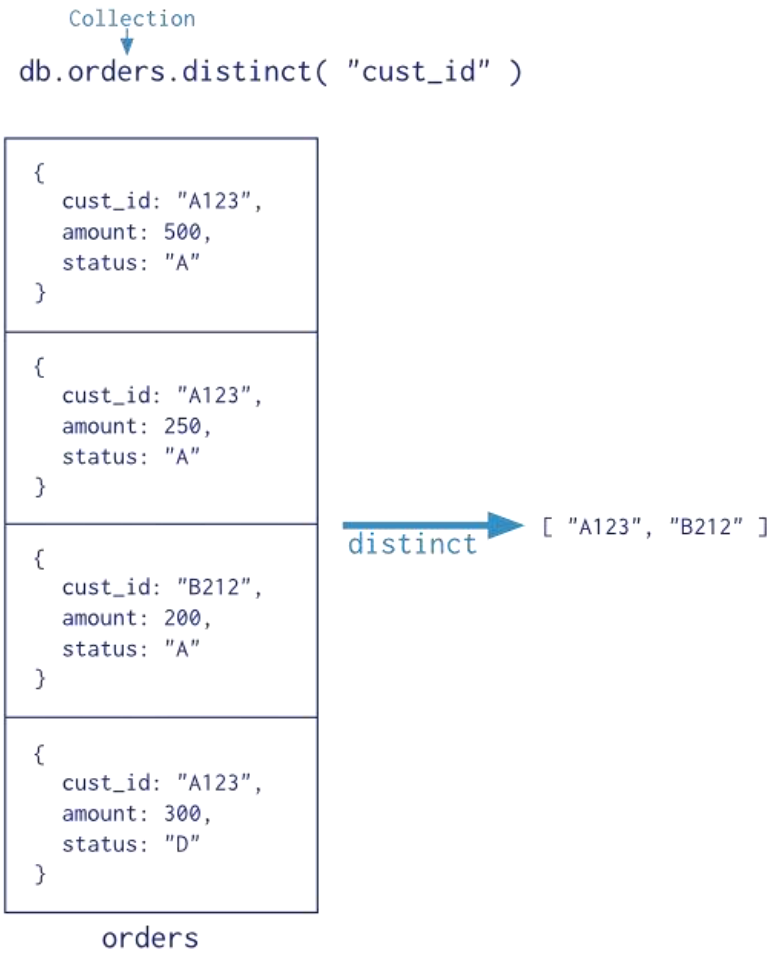

Figure 11. Annotated Single Purpose Aggregation Operation from Mongodb.com documentation http://docs.mongodb.org/master/_images/distinct.png

MongoDB Map-Reduce

MongoDB's Map-Reduce capability provides programmatic query processing flexibility not available in Aggregation Pipeline, but at a cost to performance and coherence. Map-Reduce is a massively parallel process for manipulating and condensing large volumes of data down to something more useful. MongoDB provides the 'mapReduce' command to process map-reduce scripts.

In a map-reduce process, you match data that you want to work with (the map process) and then you filter and/or condense (with the reduce process).

The "map" process creates a set of key->value pairs that you want to work with and the "reduce" process takes those values in. In MongoDB, a map process takes in a collection and the output of the reduce operation can be a collection or it can be returned inline. If you generate a collection, it can then be used as input to another map-reduce process.

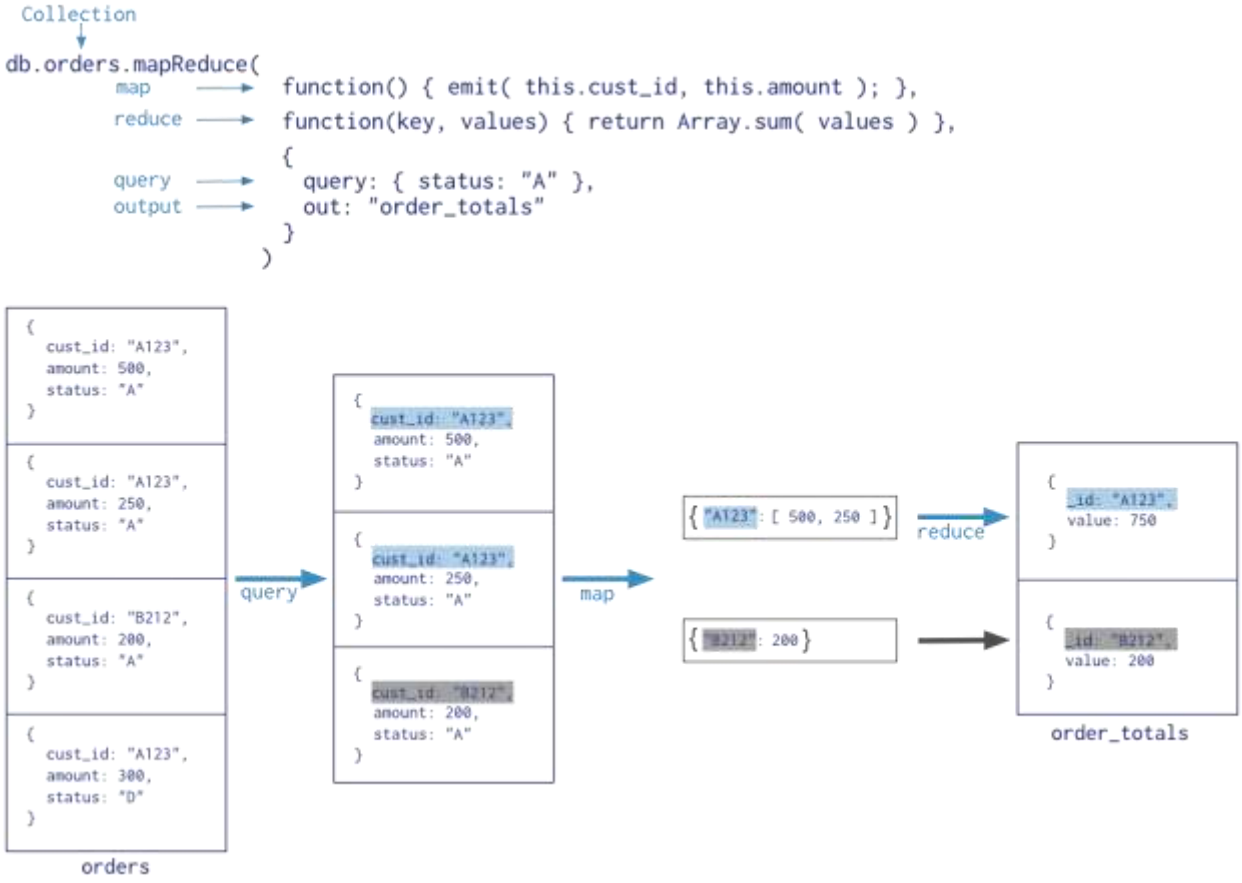

Figure 12. Annotated Map-Reduce from Mongodb.com documentation http://docs.mongodb.org/master/_images/map-reduce.png

Like the pipelined aggregation, map-reduce does support partitioned databases. MongoDB notes that while they have made improvements in map-reduce performance in later releases, the aggregation pipeline is usually more performant, though it may not be as dynamic and functional.

ScaleBase Query Aggregation Options

ScaleBase also provides two main methods of query data aggregation: ALL_DB Aggregation and an automatic and built-in “Map-Reduce-Like” capability that executes across distributed database partitions.

The primary difference is with ScaleBase you can use SQL and tools that you are most likely already familiar with.

Additionally, with ScaleBase you don't need to decide between an aggregation pipeline or a map reduce approach. Instead, you submit regular SQL query, and ScaleBase performs all the aggregations operations behind the scenes for you.

ScaleBase ALL_DB Aggregation

ScaleBase supports single-phase and multi-phase ALL_DB aggregation.
For every ALL_DB command, ScaleBase performs:

- Parallel execution
- Local processing at each individual DB, such as Filter, local joins, local groups, local sort (That’s most of the processing!)
- Meta-Aggregation at ScaleBase Controller

For example: SELECT COUNT(*) FROM my_distributed _table;

ScaleBase delegates the same command to all databases in parallel. Databases perform it very efficiently in parallel, on smaller datasets, returns n counts to ScaleBase which in turn sums all the counts to one big count, to return to the client.

The same goes with “sum of sums”, “min of mins” and so on. ScaleBase even supports a global ALL_DB average aggregated function.

In addition, consider the following supported cases:

- Multi-phase aggregation:
  GROUP BY + HAVING + ORDER BY + LIMIT
- DDL (ALTER TABLE, CREATE INDEX)

ScaleBase Map-Reduce

As mentioned before, ScaleBase provides a Map-Reduce-Like capability. ScaleBase will deconstruct the query; run parallel scans, and aggregate results.

To compare Map-Reduce in MySQL with ScaleBase and MongoDB, let's look at the following figure.

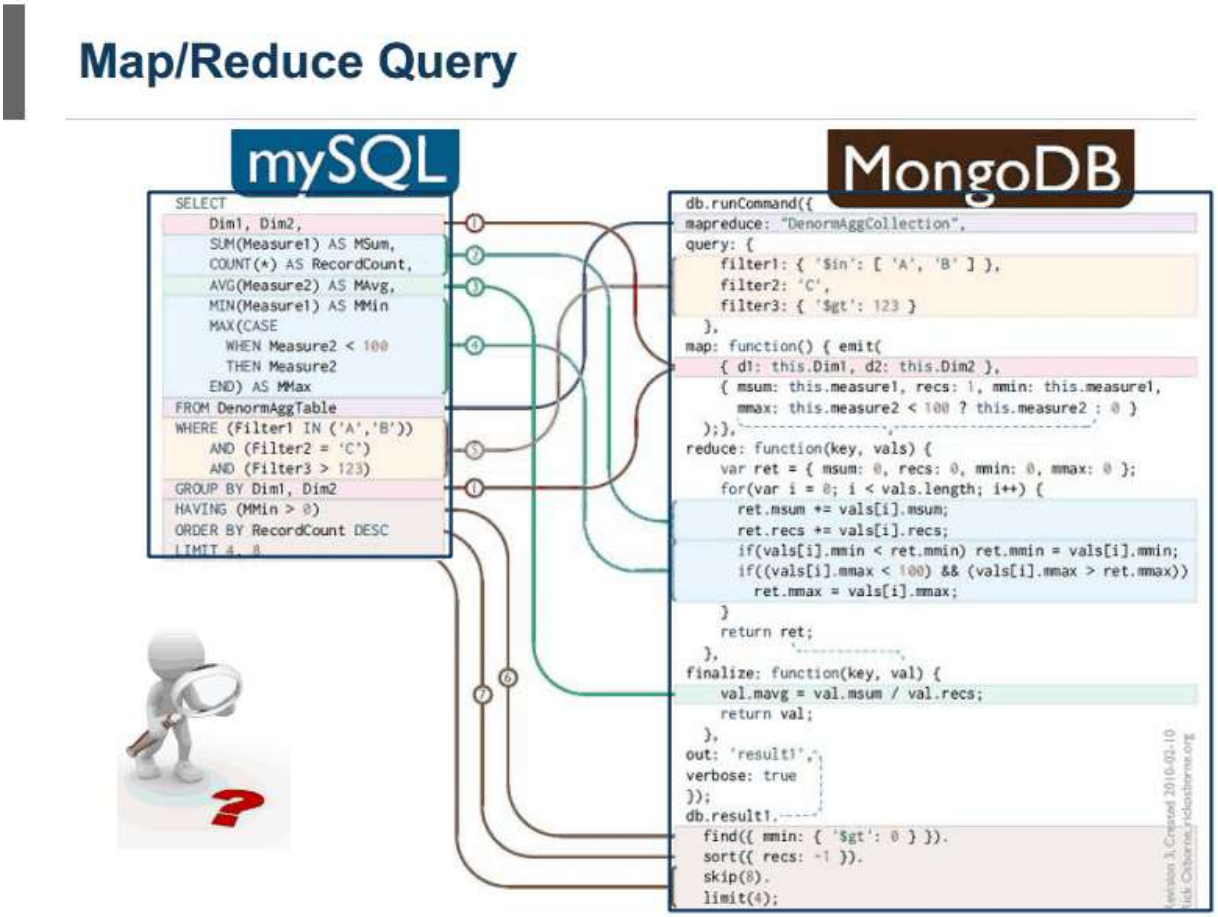

Figure13. MySQL Query and MongoDB Map-Reduce Script

Comparing a typical Map-Reduce query in MySQL and in MongoDB, you can see that to get the same functionality, a simple SQL query is a lot easier to read and to write. Ideally, less code can also mean less bugs and less maintenance.

The example above is for a very simple SQL command on only one table, with no joins, with three WHERE predicates and six SELECT predicates. As we know, queries can be much longer than that, but not more complicated. This is really the power of SQL.

You can see how the ScaleBase query flow resembles a map-reduce program in the figure below.

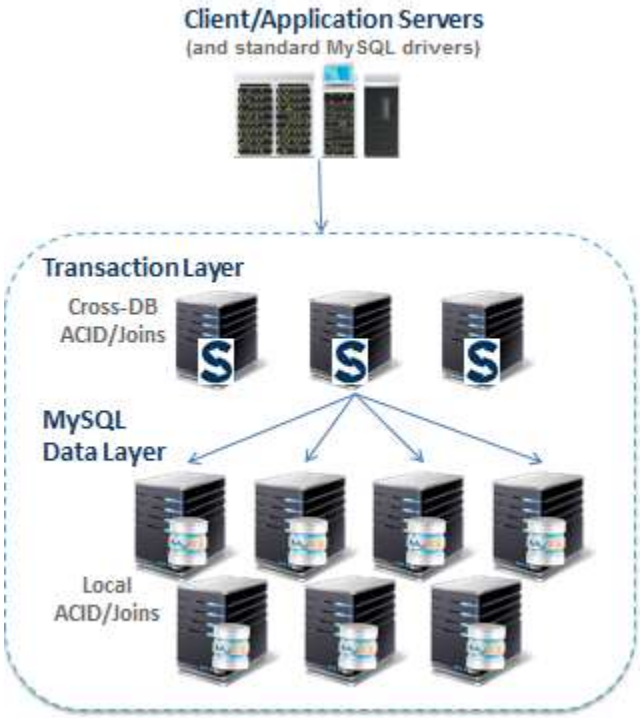

Figure 14. Map-Reduce Like Functionality in ScaleBase

With no extra coding, you get localized access at the partition/node level, running in parallel, with local sorts, joins and grouping. ScaleBase does the final aggregation for your application in the middle ScaleBase layer. Not in the application.

Aggregation is a fact of life in the data world. You don't want a tool that makes your life harder than it has to be. MongoDB and ScaleBase both have aggregation capabilities to make it as easy as possible for you.

In a distributed database environment you must have aggregation capabilities.

MongoDB and MySQL with ScaleBase both provide aggregation capabilities; MySQL with ScaleBase also retains SQL and ACID properties.

5. Summary

MongoDB was developed after SQL databases were around for quite a while and already and providing great query capabilities. MongoDB is more than a key/value database. It has great querying capabilities as well, some say equal to SQL databases, with very similar concepts of filters, indexes, results aggregation. The main difference is in the "how".

MongoDB is a NoSQL database. Some in the database world call it a web scale technology. MySQL is the world's the most widely used open source database, with a rich ecosystem of tooling and skilled users. ScaleBase is a modern relational database platform, build using standard MySQL, and optimized for the cloud, and takes advantage of your existing MySQL development skills and ecosystem technologies.

The figure below summarizes the comparison for MongoDB and ScaleBase auto-sharding, data distribution, join capabilities, data distribution, aggregation and map-reduce capabilities.

Capability	MongoDB	ScaleBase
Auto-sharding	Supported	Supported
Data distribution at the database level	Supported	Supported
Scalable distributed workload	Supported	Supported
The application "sees" one DB (transparency)	Supported	Supported
References / Relations	• Each collection is sharded independently • Joins not supported	• Relations analyzed and dictates data distribution policy • Joins are supported either locally in the database, and cross-DB
Data Redistribution	Supported	Supported
Aggregations	Supported	Supported, retaining ACID/SQL
Map-Reduce	Supported	Automated Map-Reduce Like Supported

Figure 15. Summary of MongoDB and MySQL with ScaleBase comparison

As you can see MongoDB and ScaleBase both equally support auto-sharding, data distribution at the *database* level, scalable workload distribution, and application transparency. Where they differ is at the references and relational level. In MongoDB, each document collection is sharded independently and joins are not supported, any joins across horizontal partitions need to be accomplished by the application In ScaleBase, MySQL relations are retained and form the basis of a data distribution policy, and joins are supported at the database level locally on each node, and across the entire distributed database cluster. MySQL with ScaleBase retains ACID compliance across the distributed database.

About ScaleBase

ScaleBase is a relational database platform built on MySQL and optimized for the cloud.

ScaleBase gives organizations the relational data integrity of MySQL combined with the scalability and flexibility of a modern distributed, multi-site database to support an unlimited numbers of users, larger data volumes and extremely high TPS.

Unlike other database systems that forgo ACID, SQL and joins, or rely on in-memory persistence and durability, or bank on risky asynchronous replication to achieve scalability and availability, ScaleBase provides an easy to manage horizontally scalable database cluster built on MySQL and that dynamically optimizes workloads across multiple nodes to reduce costs, increase database elasticity and drive development agility.

ScaleBase is the only distributed database cluster that uses MySQL and InnoDB storage. This unique capability provides the scalability and availability benefits of a NoSQL database combined with the strengths of an SQL database:

- Two-Phase commit and roll-back
- ACID compliance
- SQL query model, including cross-node joins and aggregations

Data distribution, transactions, concurrency control, and two-phase commit are all 100% transparent, so applications and tools continue to interact with your distributed database as if it were a single MySQL instance.

ScaleBase is the only distributed database platform that provides logical, policy-based data distribution optimized for individual applications.

ScaleBase gives you the capabilities you need for modern, 24/7, operational databases supporting cloud-scale apps.

- **Database scalability and elastic scale out** - continuously increase database size and throughput and stay ahead of application workload requirements
- **Database availability, reliability and resiliency** – protect users against downtime and delays ensuring users and applications remain connected to their documents, data files and business systems
- **Geo-distribution of database** – distribute the database so sub-sets of the database are closer to where that data is needed
- **Hybrid private/public cloud** – distribute the database so sub-sets of the database can be split across multiple sites, private and public cloud infrastructure

Migration from single MySQL database to a modern distributed shared nothing MySQL database cluster is automated, simplified and centrally managed.

ScaleBase software is available for free: http://www.scalebase.com/software/