Week 10

Table of Contents

I. MongoDB

"MongoDB (from "humongous") is an open source document-oriented database system developed and supported by 10gen. It is part of the **NoSQL** family of database systems. Instead of storing data in tables as is done in a "classical" relational database, MongoDB stores structured data as JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster."

NoSQL

- The model is non-relational
- May be stored as JSON, key-value, etc. (depending on type of NoSQL database)
- Not every record has to be of the same nature, making it very flexible
- Add new properties to data without disturbing anything
- No schema requirements to adhere to
- Support for ACID transactions can vary depending on which NoSQL DB is used
- Consistency can vary
- Scales well horizontally

In recent years, SQL and NoSQL databases have even begun to merge. For example, PostgreSQL now supports storing and querying JSON data, much like Mongo. With this, you can now achieve much of the same with Postgres as you can with Mongo, but you still don't get many of the Mongo advantages (like horizontal scaling and the simple interface, etc.). As this article puts it:

If your active data sits in the relational schema comfortably and the JSON content is a cohort to that data then you should be fine with PostgreSQL and it's much more efficient JSONB representation and indexing capabilities. If though, your data model is that of a collection of mutable documents then you probably want to look at a database engineered primarily around JSON documents like MongoDB or RethinkDB.

1. Install

Head to the MongoDB download page http://www.mongodb.org/downloads and get version 2.2.2 (or latest stable version) for your OS (by the way I'm using OSX for this tutorial so *nix guys should have no difficulty following along but Windows users may need to change some bits - sorry!).

We need to make sure that the PyMongo distribution installed. If so, in the Python shell, the following should run without raising an exception:

```
pip install pymongo
```

2. Artchitecture, Characters and Replationship

MongoDB is a document-oriented, open-source database program that is platform-independent. MongoDB, like some other NoSQL databases (but not all!), stores its data in documents using a JSON structure. This is what allows the data to be so flexible and not require a schema.

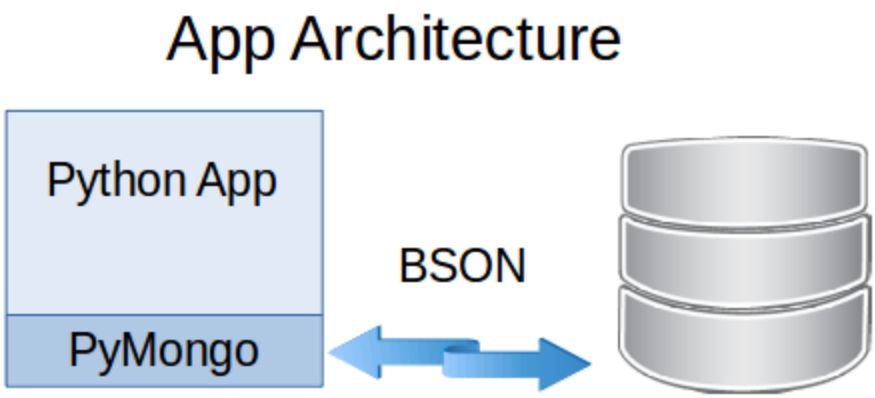
Some of the more important features are:

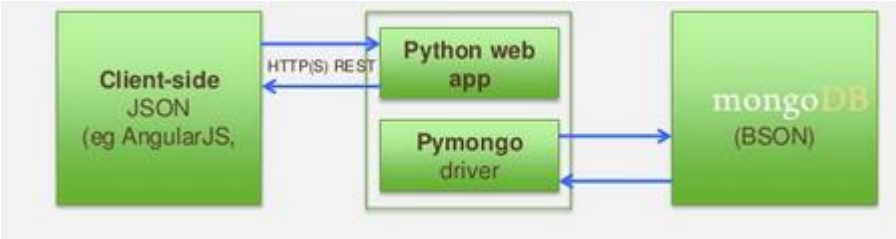
- You have support for many of the standard query types, like matching (==), comparison (<, >), or even regex
- You can store virtually any kind of data – be it structured, partially structured, or even polymorphic
- To scale up and handle more queries, just add more machines
- It is highly flexible and agile, allowing you to quickly develop your applications
- Being a document-based database means you can store all the information regarding your model in a single document
- You can change the schema of your database on the fly
- Many relational database functionalities are also available in MongoDB (e.g. indexing)

If you take advantage of the indexing features, much of this data will be kept in memory for quick retrieval. And even without indexing on specific document keys, Mongo caches quite a bit of data using the least recently used method.

While at first Mongo may seem like it's the solution to many of our database problems, it isn't without its drawbacks. One common drawback you'll hear about Mongo is its lack of support for ACID transactions. Mongo does support ACID transactions in a limited sense, but not in all cases. At the single-document level, ACID transactions are supported (which is where most transactions take place anyway). However, transactions dealing with multiple documents are not supported due to Mongo's distributed nature.

Mongo also lacks support for native joins, which must be done manually (and therefore much more slowly). Documents are meant to be all-encompassing, which means, in general, they shouldn't need to reference other documents. In the real world this doesn't always work as much of the data we work with is relational by nature. Therefore many will argue that Mongo should be used as a complementary database to a SQL DB, but as you use MongoDB you'll find that is not necessarily true.





2.1 How Replication Works in MongoDB

MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

▫ Replica set is a group of two or more nodes (generally minimum 3 nodes are required).

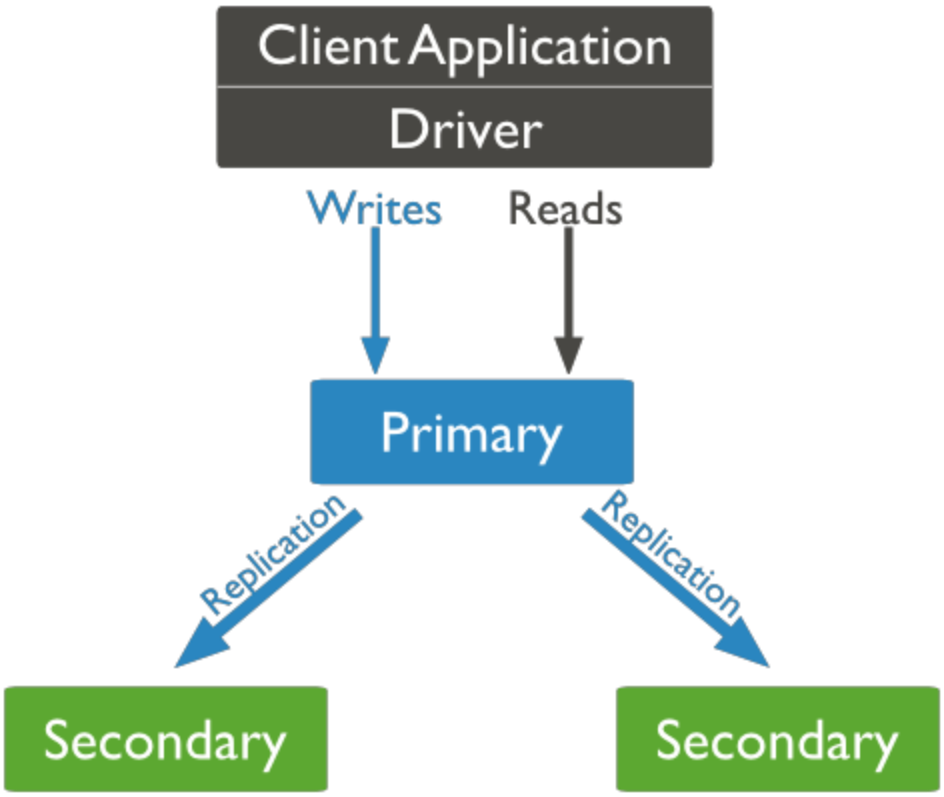
▫ In a replica set, one node is primary node and remaining nodes are secondary.

▫ All data replicates from primary to secondary node.

▫ At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.

▫ After the recovery of failed node, it again joins the replica set and works as a secondary node.

A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.



2.2 How Sharding Works in MongoDB

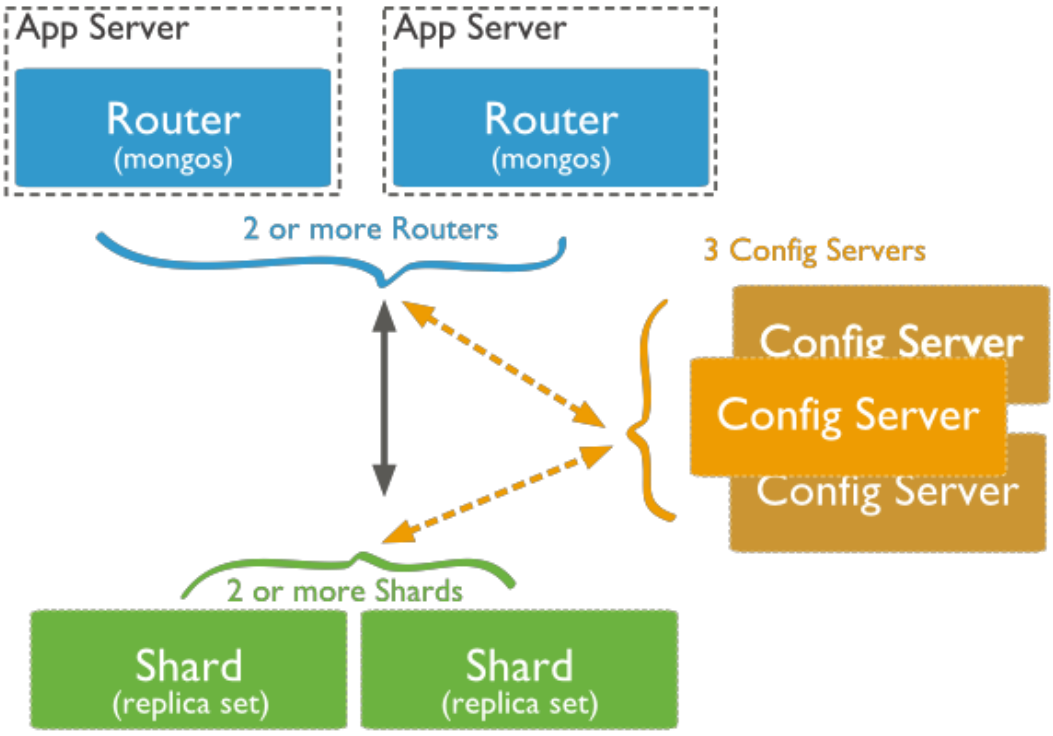
The following diagram shows the sharding in MongoDB using sharded cluster.

there are three main components:

▫ Shards: Shards are used to store data. They provide high availability and data consistency. In production environment, each shard is a separate replica set.

▫ Config Servers: Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.

▫ Query Routers: Query routers are basically mongo instances, interface with client applications and direct operations to the appropriate shard. The query router processes and targets the operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally, a sharded cluster have many query routers.



3. Robomongo

Robomongo is a shell-centric cross-platform MongoDB management tool. Unlike most other MongoDB admin UI tools, Robomongo embeds the actual mongo shell in a tabbed interface with access to a shell command line as well as GUI interaction.

3.1 Download and Install

https://robomongo.org/download

3.2 Create User in Mongodb & assign Roles

http://tgrall.github.io/blog/2015/02/04/introduction-to-mongodb-security/

3.2.1 Create Admin User

3.2.2 Connect with the Admnistrator user

3.2.3 Create/Connect with the application user

3.2.4 Create a reporting user (Read Only)

3.2.5 Add new role to a user

3.2.6 Create and use custom roles

4. Connecting and accessing MongoDB

4.1 Accessing a Database

4.2 Accessing a Collection

4.3 Documents in dictionary

4.4 Document Insert/Delete/Update

MongoDB Update Modifiers

Modifier	**Meaning**	**Example**
$inc	Atomic Increment	“$inc”:{"score”:1}
$set	Set Property Value	“$set”:{"username”:"niall"}
$unset	Unset (delete) Property	“$unset”:{"username”:1}
$push	Atomic Array Append (atom)	“$push”:{"emails”:"foo@example.com"}
$pushAll	Atomic Array Append (list)	“$pushall”:{"emails”:["foo@example.com”,"foo2@example.com"]}
$addToSet	Atomic Append-If-Not-Present	“$addToSet”:{"emails”:"foo@example.com"}
$pop	Atomic Array Tail Remove	“$pop”:{"emails”:1}
$pull	Atomic Conditional Array Item Removal	“$pull”:{"emails”:"foo@example.com"}

$pullAll	Atomic Array Multi Item Removal	“$pullAll”:{"emails”:["foo@example.com”, “foo2@example.com"]}
$rename	Atomic Property Rename	“$rename”:{"emails”:"old_emails"}

5. Querying MongoDB

MongoDB queries are represented as a JSON-like structure, just like documents. To build a query, you specify a document with properties you wish the results to match. MongoDB treats each property as having an implicit boolean AND. It natively supports boolean OR queries, but you must use a special operator ($or) to achieve it. In addition to exact matches, MongoDB has operators for greater than, less than, etc.

MongoDB Query Operators

Operator Meaning	**Example**	**SQL Equivalent**
$gt	Greater Than “score”:{"$gt”:0}	>
$lt	Less Than “score”:{"$lt”:0}	<
$gte	Greater Than or Equal “score”:{"$gte”:0}	>=
$lte	Less Than or Equal “score”:{"$lte”:0}	<=
$all	Array Must Contain All “skills”:{"$all”:["mongodb”,"python"]}	N/A
$exists	Property Must Exist “email”:{"$exists”:True}	N/A
$mod	Modulo X Equals Y “seconds”:{"$mod”:[60,0]}	MOD()
$ne	Not Equals “seconds”:{"$ne”:60}	!=
$in	In “skills”:{"$in”:["c”,"c++"]}	IN
$nin	Not In “skills”:{"$nin”:["php”,"ruby”,"perl"]}	NOT IN
$nor	Nor “$nor”:[{"language”:"english"},{"country”:"usa"}]	N/A
$or	Or “$or”:[{"language”:"english"},{"country”:"usa"}]	OR
$size	Array Must Be Of Size “skills”:{"$size”:3}	N/A

Notice the use of the special “$gt” operator. The MongoDB query language provides a number of such operators, enabling you to build quite complex queries.

5.1 Querying list of documents - find()

5.2 How many documents match a query - count()

5.3 Range Queries

II. Kafka

Kafka is an open source distributed streaming platform that simplifies data integration between systems. A stream is a pipeline to which your applications receives data continuously. As a streaming platform, Kafka has two primary uses:

- **Data Integration**: Kafka captures streams of events or data changes and feeds these to other data systems such as relational databases, key-value stores or data warehouses.
- **Stream processing**: Kafka accepts each stream of events and stores it in an append-only queue called a log. Information in the log is immutable hence enables continuous, real-time processing and transformation of these streams and makes the results available system-wide.

Compared to other technologies, Kafka has a better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution for large-scale message processing applications.

Kafka system has three main components:

1 Producer

The service that emits the source data.

2 Broker

Kafka acts as an intermediary between the producer and the consumer. It uses the power of API's to get and broadcast data

3 Consumer

The service that uses the data which the broker will broadcast.

III. House Keeping

Install SQLite on Windows

https://www.codeproject.com/Articles/850834/Installing-and-Using-SQLite-on-Windows