CS 488/588 Cloud and Cluster Data Management
Fall 2020

Project Part 1: System Investigation

Part 1 Deliverables:

1) **Team:** The DJS: Dominique Moore, Jane Seigman, Santiago Tobon

   - **Santiago Tobon** - *DB Admin* (setting up, configuring, managing the database system)
   - **Jane Seigman** - *Data Architect* (designing the schema and storage structure - including sharding, replication, etc.)
   - **Dominique Moore** - *ETL specialist* (ingesting and cleaning the data)
   - **ALL** - *Data Analyst* (running queries and analyzing the data)

2) **System Investigation Table:**

	MongoDB	*SimpleDB*	*Cassandra*
Data Model	Records are stored in documents, which is a data structure made of field and value pairs	NoSQL, stored in domains	NoSQL, key-value store
Indices	Creates an _id for each record in the DB upon creation of the record by default. Uses a B-Tree data structure. Supports the creation indices with a custom name. Supports different index types: **single field, compound, multikey, geospatial, text, and hashed**	Automatically indexes data.	Column index used to search data rows. One column per index but multiple indices allowed on a table.
Consistency	Reads and writes are issued to the primary member of a replica set. Applications can optionally read from	Keeps multiple copies of each domain. A successful write guarantees that all	Consistency levels can be configured to manage availability versus data accuracy for a session or per

	secondary replicas, where data is eventually consistent by default. Applications can also read from the closest copy of the data (as measured by ping distance) when latency is more important than consistency.	copies of the domain will durably persist.	individual read or write operation.
Sharding / Partitioning	Distributes data across a cluster of machines. Also supports creating zones of data based on a shard key. It directs reads and writes covered by a zone only to those shards inside the zone.	Can partition your data set among multiple domains to parallelize queries and operate on smaller data sets. Can use hashing algorithms to create a uniform distribution of items among multiple domains	Each row in a table has a primary key. The primary key contains the partition key and clustering key(optional). A unique partition key represents a set of rows in a table which are managed within a server.
Replication	Has a replica set, which is a group of mongod instances that maintain the same data set. This replica set contains several data nodes, and one of these nodes is presumed to be the primary node while the others are secondary nodes.	Automatic, geo-redundant replication. Everytime a data item is stored, multiple replicas are created in different data centers within the Region selected.	Multiple masters. Stores replicas on multiple nodes to ensure reliability and fault tolerance. A replication strategy determines the nodes where replicas are placed.
Joins	Uses $lookup and aggregation. These are equivalent to doing Left Outer Joins in a relational DB	Sacrifices complex transactions and relations (joins) in order to provide unique functionality and performance characteristics.	No joins built in. Must do these application-side using tools like Apache Spark's SparkSQL.

3) **Selected System:** MongoDB

4) **Selected System Description:**

1. Summary
   - MongoDB is a NoSQL, distributed, document-oriented database that uses collections of JSON-type documents which support embedded fields for the storage of related data.
   - High performance, high availability, and easy scalability are made possible through the invention of BSON: a binary representation of JSON that optimizes speed, space, and flexibility when storing and accessing data.
   - Open source and cross platform
   - MongoDB is often considered "schema-less" because the end-user doesn't need to outline specifications for tables, rows, columns, or data types. This is all dynamic. A document is simply a set of key-value pairs, no strings attached. However, for reference, general comparisons can be made to RDBMS terminology:

Table	A **<->** of	**Collection**	{ _id: ObjectId(123), field: "value" }
Rows	**<->** identified by a	**Documents**	{ _id: ObjectId(124), field: "value" }
Primary Key	**<->** and contains	**_id:ObjectId()**	
Columns	**<->**	**Fields**	

2. Justification
   - *Simplicity with High Performance*: Storage of JSON document's nested key-value pairs in binary format provides quick interpretation for both humans and machines. Although Cassandra's multiple masters boasts uninterrupted data availability, the simplicity of MongoDB's single master, when paired with its BSON format and powerful MQL, is worth a few seconds for a slave node to be promoted.
   - *Flexibility for Fluctual Data*: Other databases have constrictions for datatypes and schemas and downtimes for making modifications. MongoDB allows you to enter data and change schemas whenever you want without interrupting operations. Real-world data isn't cookie-cutter. The BSON data format allows documents in a collection to have varying sets of fields. It is designed for change.
   - *Powerful Data Access*: In many ways, MongoDB removes the need for a "middle-man". Documents are already objects so accessing data from programs is code-native. As for querying the data, MongoDB Query Language (MQL) is designed specifically for accessing data in BSON documents, while also supporting complex queries such as joins.

- *We're Beginners:* MongoDB is popular because it's easy to learn and is easily transferable to front-end and back-end systems we are familiar with.

3. Data Model
    a. What is the data model of your system?

Mongodb is a noSQL database. It is a document based database, that is schema-less. Mongodb stores the data records as BSON documents which is a binary representation of JSON documents. The documents are composed of a field and a value pair in the following structure:

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

You can use one of two data model designs to structure your documents, embedded data models or normalized data models. Embedded models allow you to embed the related data in a single structure or document. These schemas are known as "denormalized" models. Embedded data models allow applications to store related pieces of information in the same database record. This means applications can issue fewer queries and updates for a common operation. Normalized data models describe relationships using references between documents. They are better for representing more complex many to many relationships and to model large hierarchical data sets.

    b. Give an example of data stored in your system and the function to insert data.

Example of data stored:

```
{
 "name": "notebook",
 "qty": 50,
 "rating": [ { "score": 8 }, { "score": 9 } ],
 "size": { "height": 11, "width": 8.5, "unit": "in" },
 "status": "A",
"tags": [ "college-ruled", "perforated"]
}
```

Function to insert data:

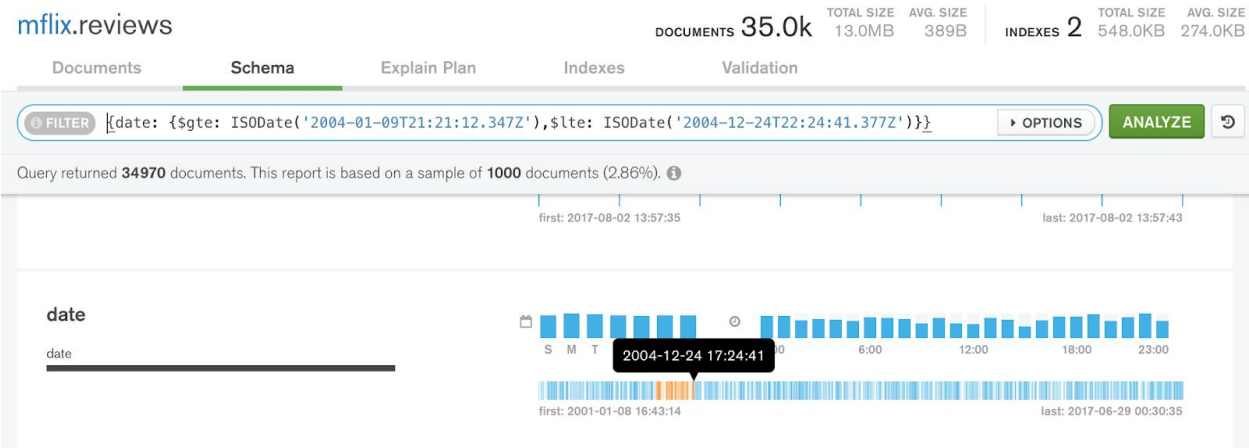
```
db.collection.insert(
        <document or array of documents>,
        {
        writeConcern: <document>,
        ordered: <boolean>
        }
)
```

4. User Interface
    a. Which UI will you use to interact with your system?

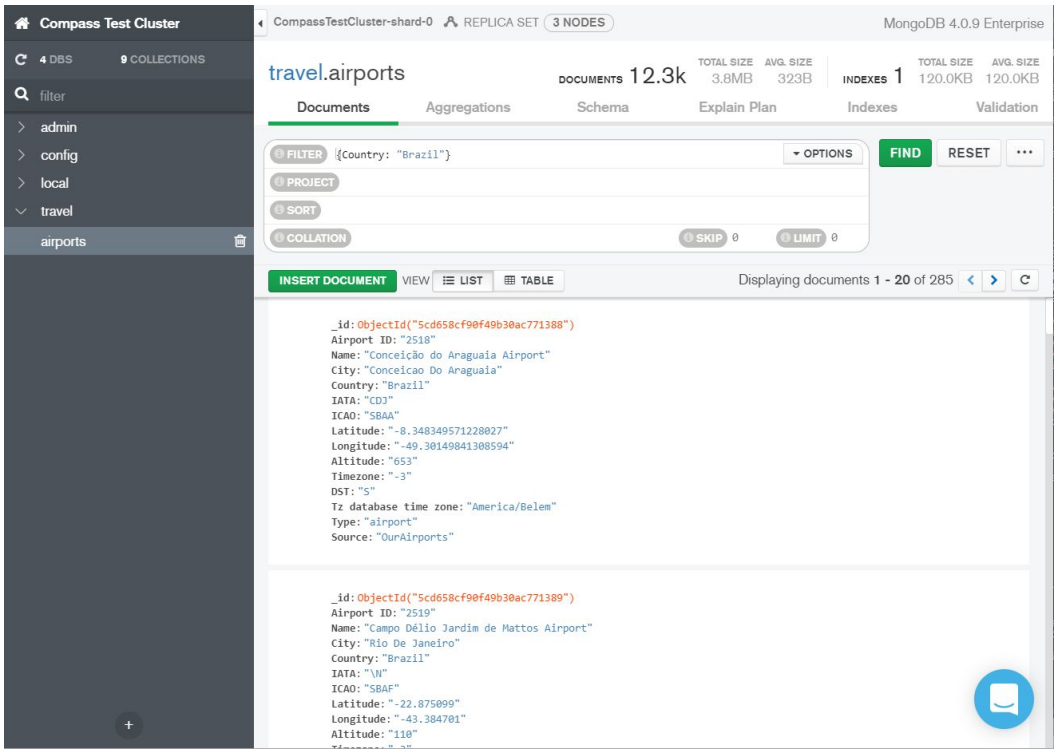
Mongodb Compass is the gui for Mongodb. It is used to help visually explore the data, run ad hoc queries, modify the data, and debug and optimize it.

b. Give two examples of interactions with your system. These interactions should be of different types – e.g. range queries or graph navigation queries.
(Wasn't sure how you wanted the examples since we don't have the UI installed yet, so I used screenshots from online.)

Screenshot of using a mouse to click and drag across a range of entries in the date field.



Screenshot of running a query and viewing the results.



5. Indexes -

MongoDB supports many different types of indices. It provides many different types of indices in order to specific types of data and queries. These indices include: Default id, Single Field, Compound Index, Multikey Index, and etc.

The default index supported is a unique id index which stops clients from inserting multiple documents with the same value for the id field. MongoDB creates this unique index upon creation of a collection.

Single field indices are user-defined increasing/decreasing indexes on a single field of a document. The sorted order of the index key is inconsequential because MongoDB is able to traverse the index in ascending or descending order.

A compound index is like a single field index, but on multiple fields. The order of fields listed in a compound index is important. For example if a compound index looks like: { id: 1, zip: 2}, then the index will sort first by "id" and then in each "id" value, it sorts by "zip".

Multikey indexes are used to index data stored in arrays. MongoDB creates index entries for every element of an array if a field that has an index holds an array value. These indexes allow queries to select documents that contain arrays by matching on elements of the arrays. A Multikey index is automatically created by MongoDB if the indexed field contains an array value, the multikey type doesn't need to be explicitly specified.

Two examples of creation of indexes in MongoDB:

1) db.collection.createIndex({ userId: 1 })

2) db.collection.createIndex({productId: -1, quantity: -1 })

6. Consistency -

MongoDB has something known as "Casual Consistency" where a client application can decide the type of consistency for read concerns and write concerns. The type of consistency allows MongoDB to say it is "consistent by default," meaning that reads and writes are issued to the primary member of a replica set. The options for reads are: majority and local. The options for writes are: majority and {w: 1}. The combination of these read and write options give the client applications different options for consistency.

Applications have the option to read from secondary replicas, where data will be eventually consistent by default. In cases where it's ok for data to be slightly out of date, reads from the secondary can be useful. If latency is more important than consistency then applications can read from the closest copy of the data, which would be measured by ping distance.

MongoDB supports multi-document transactions, which can also be called distributed transactions. In MongoDB, an operation on a single document is atomic.
Multi-document transactions are needed since embedded documents and arrays can be used to capture relationships between data in a single document structure. After a transaction commits, all changes to the data in the transaction are saved and visible outside the transaction. Any changes made in the transaction aren't visible outside the transaction until a transaction commits. If multiple shards are involved then some read operations will not have to wait for a transaction to commit to see results across the shards. If a transaction aborts then all data changes from the transaction will be thrown away, and won't become visible.

7. Scalability and Replication
    a. Does your system support replication?

Mongodb supports replication through a replica set. When the writer operations are sent to the primary server, it also applies the operations across secondary servers, thus replicating the data. If the primary server fails then a secondary server takes over.

    b. Does your system support sharding?

MongoDB uses sharding to support horizontal scaling for deployments with large data sets and high throughput operations. A sharded cluster contains a shard, mongos, and config servers. A shard can be deployed as a replica set containing a subset of the data, which is sharded at the collection level and are identified by immutable shard keys. "Mongos" route queries from client applications to the sharded cluster. Metadata and cluster settings are stored in the config servers.

    c. Give examples of how replication and sharding can be set up in your system.

Example of how to initiate a replica set:

```
rs.initiate( {
_id :  "rs0",
members: [
{ _id:  0, host:  "mongodb0.example.net:27017" },
{ _id:  1, host:  "mongodb1.example.net:27017" },
{ _id:  2, host:  "mongodb2.example.net:27017" }
]
})
```

Whether or not a collection itself is sharded, if it's in a sharded cluster, you must connect to a mongos router in order to connect and interact with it. This can be done via the mongo shell or a driver.

References:

https://phoenixnap.com/kb/cassandra-vs-mongodb

https://www.tutorialspoint.com/mongodb/mongodb_overview.htm#:~:text=MongoDB%20is%20a%20cross%2Dplatform,concept%20of%20collection%20and%20document.

https://www.mongodb.com/

https://www.sitepoint.com/an-introduction-to-mongodb/

https://docs.mongodb.com/manual/core/document/

https://www.studytonight.com/mongodb/data-modelling-in-mongodb

https://docs.mongodb.com/guides/server/introduction/

https://docs.mongodb.com/manual/reference/method/db.collection.insert/

https://www.mongodb.com/products/compass

https://docs.mongodb.com/compass/master/query/filter

https://www.mongodb.com/basics/replication#:~:text=With%20MongoDB%2C%20replication%20is%20achieved,secondary%20servers%2C%20replicating%20the%20data.