ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

Postgraduate Program

“Information System and Services”

“MONGO-DB Project”

Apostolidou Melina, me1920

Rapti Charikleia, me1941

Chatzigeorgiou Maria, me1949

INTRODUCTION

Approximately 80–90% of the different types of data in this modern time have been created in the past few years alone and every day, we create approximately 2.5 quintillion bytes of data that comes from everywhere like posts to social sites, collect weather information, digital images and videos, transaction information, and more. In such scenario, organizations need a web based solution that integrates the robust presentation features of a portal like user interfaces, collaboration, and secure access, with centralized as well as enormously scalable data storage as the back end, composing of different types of content such as Images, Audio, Video, Documents, Metadata in huge amount.

Reason for Fall of Other Databases

Many databases make you choose between a flexible data model, low latency at scale, as well as powerful access, but increasingly you need all mainly three at the same time.

A. Rigid Schemas

You should be able to analyze semi-structured, unstructured, as well as polymorphic data, and it should be easy to add new data. But problem is that these types of data do not belong in relational rows and columns. Relational schemas are hard to change incrementally, mainly without impacting performance or taking the database offline.

B. Scaling Problems

Relational databases were mainly designed for single-server configurations, not for horizontal scale-out. They were meant to serve approximately 100s of ops per second, not 100,000s of ops per second. With a lot of engineering hours, custom sharding layers, as well as caches, scaling an RDBMS is hard at best or impossible at worst.

C. Takes Too Long

Analyzing data in real time requires a break from the familiar ETL as well as data warehouse approach. You do not have time for lengthy load schedules. You need also to run aggregation queries against variably structured data.

NoSQL

NoSQL (Not only SQL) database provides a mechanism for retrieval as well as storage of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach contain simplicity of design, horizontal scaling, as well as finer control over availability. NoSQL databases are increasingly used in big data as well as real-time web applications. Many NoSQL database stores compromise consistency (in the sense of the CAP theorem) in favour of availability as well as partition tolerance.

MongoDB

MongoDB is a famous NoSQL database that is an open source, written in C++, crossplatform, high performance as well as document oriented database. MongoDB uses collections to store data as well as represent relationships between them and data is in the format of BSON documents. It is a binary format in that zero or more key/value pairs are stored as a

single entity i.e. as a document. BSON is based on JSON style documents. JSON (JavaScript Object Notation) is a format that is easy for computers to parse and generate.

Features of MongoDB

MongoDB features include like full index support, replication, high availability, and auto-sharding. Here we are discussing some important features of MongoDB such as:

A. Indexing

MongoDB supports secondary indexing, that makes retrieval faster as well as unique, compound and geospatial indexing is also possible.

B. Stored JavaScript

Users can also use JavaScript function as well as scripts on server side.

C. Aggregation

MongoDB supports MapReduce that is a very useful aggregation tool.

D. Horizontal Scaling

MongoDB scales horizontally. MongoDB scales out and up easily on a variety of platforms including in the cloud using services like Amazon EC2 and Rackspace.

E. Sharding

This is a process in which large databases are broken down into different tables so that they can be processed on multiple machines and in MongoDB, this is automatic. MongoDB's sophisticated sharding keys make balancing your data across large clusters easy and powerful.

MongoDB How Makes Easy

Many organizations are using MongoDB database for analytics because it lets them store any kind of data, analyze it in real time, as well as change the schema as they go.

A. New Data

MongoDB's document model enables you to store as well as process data of any structure like events, time series data, geospatial coordinates, text, binary data, and anything else. You can easily adapt the structure of a document's schema just by adding new fields, making it simple to bring in new data as it becomes available.

B. Horizontal Scalability

MongoDB's supports automatic sharding distributes data across fleets of commodity servers, with complete application transparency. Multiple options for scaling like range-based, hash-based as well as locationaware sharding, MongoDB can support thousands of nodes, petabytes of data, as well as hundreds of thousands of ops per second without requiring you to build custom partitioning and caching layers.

C. Powerful Analytics

In Real Time, with rich index as well as query support like secondary, geospatial and text search indexes as well as the aggregation framework and native MapReduce (MR), MongoDB can easily run complex ad-hoc analytics and reporting in place.

Advantages of MongoDB

- Schema-less (without schema) design enables rapid introduction of new CDR (Call Detail Record) types to the system.

- Scale BillRun production site already controls several TB in a single table, w/o being limited by adding new fields.

- Rapid replica Set easily enables meeting regulation with easy to setup multi data center DRP as well as HA solution.

- Sharding enables linear as well as scale out growth w/o running out of budget.

- With over approximately 2,000/s CDR inserts, then MongoDB architecture is great for a system that must support high insert load. So you can easily guarantee transactions with findAndModify as well as two-phase commit.

- Supports developer oriented queries.

- Location based is being utilized to analyze users usage as well as determining where to invest in cellular infrastructure.

Dis-advantages of MongoDB

- The current database is locked when MongoDB is writing onto it, therefore this does not allow concurrent writes.

- MongoDB reports scalability constraints when the data exceeds hundreds of GB.

Indexing in MongoDB

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. Scans can be highly inefficient and require MongoDB to process a large volume of data. Indexes are special data structures that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index. MongoDB supports indexes that contain either a single field or multiple fields depending on the operations that this index type supports. By default, MongoDB creates the _id index, which is an ascending unique index on the _id field, for all collections when the collection is created. You cannot remove the index on the _id field.

Dataset Description

From the initial source of data, we chose the nari_dynamic, the nari_static and the Ship Types List. More specifically, the information we used from the nari_dynamic was lon, lat. From the nari_static we used shipname, shiptype, to_bow, to_stern, draught, destination and finally, from ship_types we used, shiptype_min, shiptype_max, ais_type_summary.

Data Analysis

Initailly, we loaded the files into Python. The files nari_dynamic and Ship_Types_List did not had any missing values. The file nari_static had missing values and especially the column “mothermmsi”, so we dropped it. The missing values from the other columns were filled with the mean of each column. In addition, for the files nari_dynamic and nari_static we used the first 100000 rows and so created two new files nari_dynamic_new and nari_static_new which we imported and used in MongoDB.

```
import pandas as pd
```

```
nari_static = pd.read_csv(r'C:\Users\user\Desktop\MongoData\AisData\nari_static.csv', low_memory=False)
ship_types = pd.read_csv(r'C:\Users\user\Desktop\MongoData\AisTypes\Ship_Types_List.csv', low_memory=False)
nari_dynamic = pd.read_csv(r'C:\Users\user\Desktop\MongoData\AisData\nari_dynamic.csv', low_memory=False)
```

```
nari_static.shape
```

```
(1078617, 14)
```

```
nari_dynamic.shape
```

```
(19035630, 9)
```

```
nari_static = nari_static[:100000]
```

```
nari_dynamic = nari_dynamic[:100000]
```

```
nari_static.shape
```

```
(100000, 14)
```

```
nari_dynamic.shape
```

```
(100000, 9)
```

```
total = nari_static.isnull().sum().sort_values(ascending=False)
percent = (nari_static.isnull().sum()/nari_static.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
#missing_data.head(20)
```

```
nari_static.drop(['mothershipmmsi'],axis=1,inplace=True)
```

```
nari_static.fillna(nari_static.mean())
```

```
nari_static.to_csv('nari_static_new.csv')
```

```
nari_dynamic.to_csv('nari_dynamic_new.csv')
```

In the next step, the data of each dataset were uploaded into MongoDB as unique collections using the following command:

```
C:\Users\user>mongoimport --db queen --collection nari_static --file "C:\Users\user\Desktop\nari_static_new.csv" --type csv --headerline
2020-02-11T17:50:54.365+0200    connected to: mongodb://localhost/
2020-02-11T17:50:56.490+0200    100000 document(s) imported successfully. 0 document(s) failed to import.

C:\Users\user>mongoimport --db queen --collection nari_dynamic --file "C:\Users\user\Desktop\nari_dynamic_new.csv" --type csv --headerline
2020-02-11T17:53:31.158+0200    connected to: mongodb://localhost/
2020-02-11T17:53:32.845+0200    100000 document(s) imported successfully. 0 document(s) failed to import.

C:\Users\user>mongoimport --db queen --collection ship_types --file "C:\Users\user\Desktop\Ship_Types_List.csv" --type csv --headerline
2020-02-11T17:54:49.567+0200    connected to: mongodb://localhost/
2020-02-11T17:54:49.629+0200    38 document(s) imported successfully. 0 document(s) failed to import.
```

At first, we wanted to find information about shiptype_min and shiptype_max in case ais_type_summary was “Cargo”.

```
> db.ship_types.find( {ais_type_summary:"Cargo"}, {shiptype_min: 1, shiptype_max: 1} ).pretty()
{
        "_id" : ObjectId("5e42ce497701737a4d0527e8"),
        "shiptype_min" : 70,
        "shiptype_max" : 70
}
{
        "_id" : ObjectId("5e42ce497701737a4d0527e9"),
        "shiptype_min" : 73,
        "shiptype_max" : 73
}
{
        "_id" : ObjectId("5e42ce497701737a4d0527ea"),
        "shiptype_min" : 74,
        "shiptype_max" : 74
}
{
        "_id" : ObjectId("5e42ce497701737a4d0527ec"),
        "shiptype_min" : 75,
        "shiptype_max" : 79
}
{
        "_id" : ObjectId("5e42ce497701737a4d0527f0"),
        "shiptype_min" : 72,
        "shiptype_max" : 72
}
{
        "_id" : ObjectId("5e42ce497701737a4d0527f1"),
        "shiptype_min" : 71,
        "shiptype_max" : 71
}
```

Furthermore, we counted the number of ships going to “MONTOIR” with shiptype between 70 and 79 , which corresponds to “Cargo” ais_type_summary. The result was 81.

```
> db.nari_static.find({$and: [{"destination": {$eq: "MONTOIR"}}, {"shiptype": {$gte:70, $lte:79}}]}).count()
81
```

After that, we found the number of ships going to “MONTOIR” with shiptype 71 ,which refers to Cargo-Hazzard A(Major). The result was 11.

```
> db.nari_static.find({$and: [{"destination": {$eq: "MONTOIR"}}, {"shiptype": {$eq: 71}}]}).count()
11
```

Then we executed the same query but with *explain(“executionStats”)*, in order to get the the results of the execution time.

```
> db.nari_static.find({$and: [{"destination": {$eq: "MONTOIR"}}, {"shiptype": {$eq: 71}}]}).explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "queen.nari_static",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "destination" : {
                                                "$eq" : "MONTOIR"
                                        }
                                },
                                {
                                        "shiptype" : {
                                                "$eq" : 71
                                        }
                                }
                        ]
                },
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "$and" : [
                                        {
                                                "destination" : {
                                                        "$eq" : "MONTOIR"
                                                }
                                        },
                                        {
                                                "shiptype" : {
                                                        "$eq" : 71
                                                }
                                        }
                                ]
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 11,
                "executionTimeMillis" : 84,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 100000,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "$and" : [
                                        {
                                                "destination" : {
                                                        "$eq" : "MONTOIR"
                                                }
                                        },
                                        {
                                                "shiptype" : {
                                                        "$eq" : 71
                                                }
                                        }
                                ]
                        },
                        "nReturned" : 11,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 100002,
                        "advanced" : 11,
                        "needTime" : 99990,
                        "needYield" : 0,
                        "saveState" : 781,
                        "restoreState" : 781,
                        "isEOF" : 1,
                        "direction" : "forward",
                        "docsExamined" : 100000
                }
        },
        "serverInfo" : {
                "host" : "LAPTOP-6IMKU284",
                "port" : 27017,
                "version" : "4.2.1",
                "gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"
        },
        "ok" : 1
}
```

The executionTimeMillis was 84.

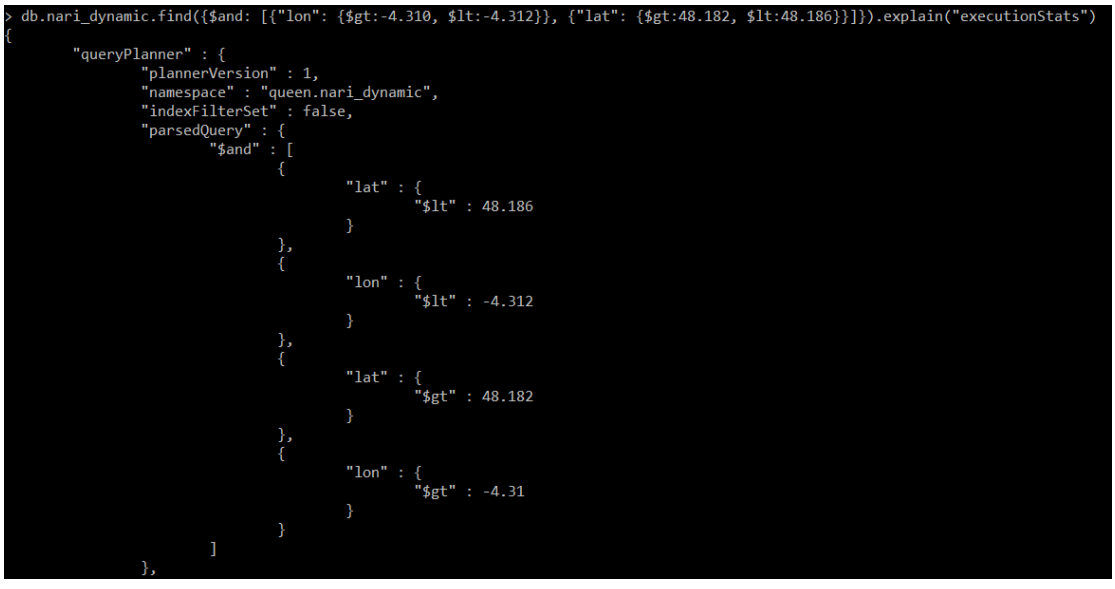
We executed the same query with descended index at “destination” and “shiptype” fields.

```
> db.nari_static.createIndex({"destination":1, "shiptype":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

```
> db.nari_static.find({$and: [{"destination": {$eq: "MONTOIR"}}, {"shiptype": {$eq: 71}}]}).explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "queen.nari_static",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "destination" : {
                                                "$eq" : "MONTOIR"
                                        }
                                },
                                {
                                        "shiptype" : {
                                                "$eq" : 71
                                        }
                                }
                        ]
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "destination" : 1,
                                        "shiptype" : 1
                                },
                                "indexName" : "destination_1_shiptype_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "destination" : [ ],
                                        "shiptype" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "destination" : [
                                                "[\"MONTOIR\", \"MONTOIR\"]"
                                        ],
                                        "shiptype" : [
                                                "[71.0, 71.0]"
                                        ]
                                }
                        }
```

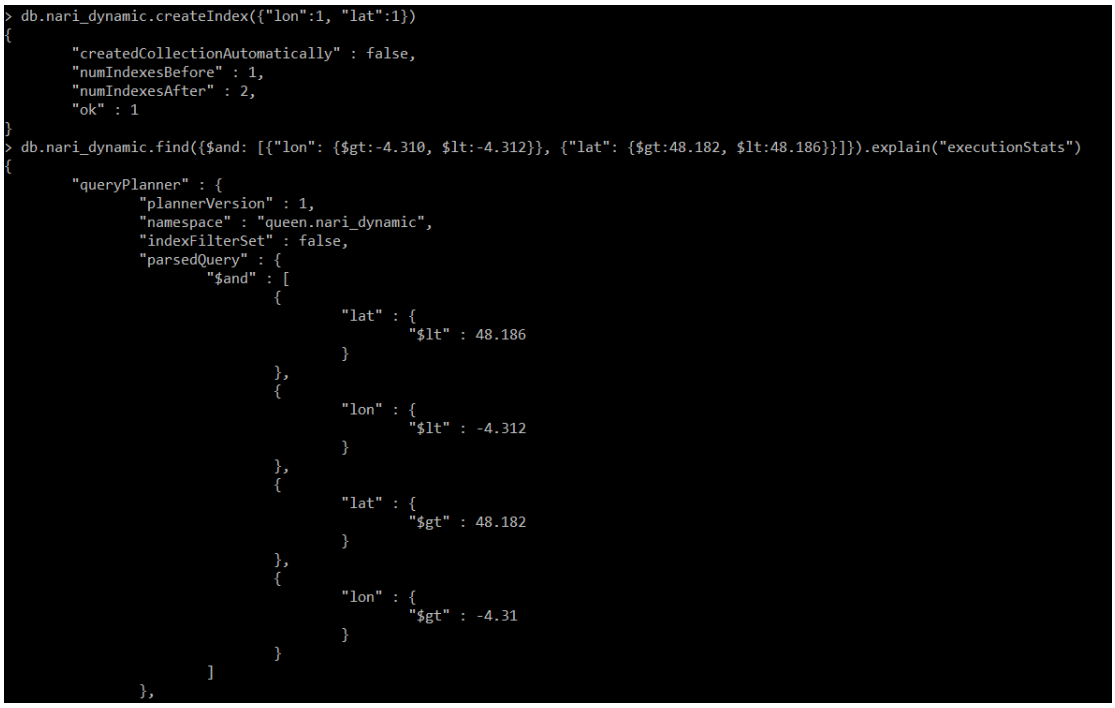
The executionTimeMillis was 9. Almost 9x faster with the index.

Another query we executed was to find information about the ships sailing in a specific area of “lot” and “lat”.Also,we wanted to find the results of the execution time.



The executionTimeMillis was 125.

We executed the same query with descended index at “lon” and “lat” fields.



```
> db.nari_dynamic.createIndex({"lon":1, "lat":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.nari_dynamic.find({$and: [{"lon": {$gt:-4.310, $lt:-4.312}}, {"lat": {$gt:48.182, $lt:48.186}}]}).explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "queen.nari_dynamic",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "lat" : {
                                                "$lt" : 48.186
                                        }
                                },
                                {
                                        "lon" : {
                                                "$lt" : -4.312
                                        }
                                },
                                {
                                        "lat" : {
                                                "$gt" : 48.182
                                        }
                                },
                                {
                                        "lon" : {
                                                "$gt" : -4.31
                                        }
                                }
                        ]
                },
```

The executionTimeMillis was 14. Approximately 9x faster with the index.

Sharding

MongoDB supports sharding by configuring a sharded cluster. Each cluster is composed of three main components namely shards, query routers and configuration servers. Query routers redirect the client query to the appropriate shards after looking up shard addresses from the config. servers. Query routers are also responsible for cluster balancing using the two primary operations of chunk splitting and balancing. MongoDB supports collection level partitioning of data. Each collection is distributed evenly across mongoDB sharded cluster in the form of chunks. Unfortunately, we were not able to perform sharding.

References

https://docs.mongodb.com/manual/tutorial/

https://en.wikipedia.org/wiki/MongoDB?fbclid=IwAR2ijESN4pwi5zujCemcyOAXA21pv8SP45XUx3aXpawMOHcNwo250eIZFNI

https://www.w3schools.com/Python/default.asp?fbclid=IwAR1Pt0Y44uuUTL3mvp2lf3hJdgRU4tHIcOqhdHT-OSgKs8zT1ZeyH9yP0-E

https://en.wikipedia.org/wiki/Automatic_identification_system?fbclid=IwAR3SutzLdhomhHjkNTGfCfGhrdtebxb192ZBZPGAo5yzGnxBajtBmoVrvEY