Optimizing Query Performance in Maritime Surveillance using MongoDB

Georgios Mageirias
Department of Mechanical Engineering
University of Thessaly
Athens, Attiki, Greece
magirias123@gmail.com

ABSTRACT

In this project, MongoDB is used as the storage layer for a heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance, that have been collected by an Automatic Identification System (AIS). The goal is to support efficient relational, spatial, and spatio-temporal queries. Indexing is used to improve the performance of these queries by creating indexes on relevant fields, so that MongoDB can quickly locate the relevant data. The use of MongoDB allows for flexible and scalable data management, as well as the ability to handle a variety of data types and perform complex queries with the help of indexing. Additionally, modeling aspects have been respected to ensure accurate and meaningful representation of the data. This dataset can be used to support decision making in maritime operations, such as tracking vessels and identifying potential threats.

KEYWORDS

MongoDB, Maritime Intelligence, Relational queries, Spatio-temporal queries, Indexing, Tracking vessels, AIS

1 Introduction

Automatic Identification System (AIS) is a widely used marine system that allows for the identification and tracking of vessels worldwide. It provides real-time vessel traffic data and information for tracking and identification of vessels, including information such as name, boat speed, and destination. AIS data is used for various purposes such as collision avoidance, navigation efficiency, and communication between vessels and shore-based stations. The process of storing, processing, and retrieving data from the AIS system is crucial for many applications. However, traditional relational databases may not be able to handle the large volume and variety of AIS data effectively. Therefore, there is a need to investigate alternative methods for managing AIS data.

The main problem that this study addresses is the efficient management of AIS data using a non-relational database. The goal is to investigate the use of MongoDB for storing, processing, and retrieving AIS data, and to evaluate its performance and suitability for managing AIS data.

There are many existing studies that have investigated the use of MongoDB for managing marine data. The paper by C. Rayab, R Dréo describes a dataset that contains ship information collected through the Automatic Identification System (AIS), as well as additional data sources such as environmental and geographic data. The dataset covers a six-month period from October 1st, 2015 to March 31st, 2016, and provides ship positions in the Celtic Sea, North Atlantic Ocean, English Channel, and Bay of Biscay. It is designed for easy integration with relational databases using the open-source PostgreSQL system and the geospatial extension PostGIS.

In "Maritime Traffic Management using MongoDB" by M. Huang et al. presented a system that utilizes MongoDB for collecting and analyzing maritime traffic data to support maritime traffic management. The system is able to handle real-time data streams, perform complex queries and improve maritime situational awareness.

"MongoDB and Ocean Data" by A. Chen et al. highlighted the capability of MongoDB to handle ocean data, which is usually big, diverse and with high-frequency. The authors proposed a system that combines MongoDB and other open source technology to collect, store, and analyze ocean data to support maritime security, maritime domain awareness, and maritime traffic management.

These studies demonstrate that MongoDB is a suitable technology for handling and analyzing maritime data, and its potential for improving decision making in maritime operations through real-time data processing and complex queries.

2 Data and Queries

Dataset Description

The dataset used in this study is collected from the AIS data available on Zenodo. The dataset provides a detailed and comprehensive view of ship movements over a specific period of time. It includes information such as vessel name, speed, course, position, and destination, as well as additional vessel-related data such as ship type, length, width, and international identifier. The data is collected in real-time and provides a high-resolution view of ship movements, making it suitable for various applications such as collision avoidance, navigation efficiency, and communication between vessels and shore-based stations. The dataset is large, with millions of records and a high rate of data generation, and it is designed to be easily integrated with other data sources to provide a comprehensive view of maritime activities.

Supported Queries:

- Relational Queries: These are queries that retrieve information based on the relationships between the data, such as finding all the ships that belong to a specific shipping company or finding the average speed of ships in a specific area.
- Spatial Queries: These are queries that retrieve information based on the location of the data, such as finding all ships within a certain radius of a specific location or finding the number of ships in a specific area.
- Spatio-Temporal Queries: These are queries that retrieve information based on both location and time, such as finding all ships that passed through a specific area at a specific time or finding the number of ships that passed through a specific area in a specific time period.

3 Data Modeling

In this project, two csv files were used as the data source. The first one, called anfr.csv, contains information about vessels' registration and identification, including fields such as maritime_area, registration_number, imo_number, ship_name, callsign, mmsi, ship_type, length, tonnage, tonnage_unit, material_onboard, atis_code, radio_license_status, date_first_license, and date_inactivity_license. This dataset did not require any preprocessing and was directly loaded into MongoDB, a non-relational database management system (NoSQL) for storage and querying.

The second csv file, called nari_dynamic, contains information about vessels' dynamic characteristics, including fields such as mmsi, status, turn, speed, course, heading, lon, lat, and timestamps. This dataset required some preprocessing to convert the timestamp in UNIX epochs format to date time format and to define the longitude and latitude as coordinates. These preprocessing steps were done after the dataset was imported to MongoDB.

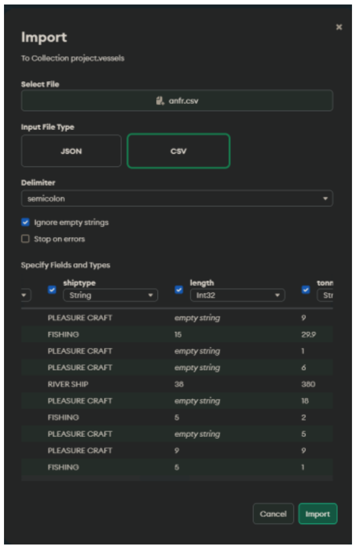
The choice of using MongoDB as the storage layer was made due to its ability to handle unstructured and semi-structured data, scalability, and support for complex queries. MongoDB also allows for flexible data modeling, making it suitable for the maritime domain where data often needs to be integrated from multiple sources and used for a variety of purposes.

Overall, the data modeling approach used in this project aimed to provide a comprehensive view of maritime activities by integrating multiple data sources and ensuring data quality and integrity.

4 Implementation

The datasets were loaded into MongoDB, using the MongoDB Compass import feature, and they were stored in the collections “dynamic_locs” and “vessels”, with the former collection containing dynamic information of the ships such as positions, speed and course, and the latter collection containing static information of the ships such as names, ship type and international identifier.

`



Import

To Collection project.vessels

Select File: anfr.csv

Input File Type: JSON [✓ CSV]

Delimiter: semicolon

☑ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

shiptype (String)	length (Int32)	tonn (Str
PLEASURE CRAFT	*empty string*	9
FISHING	15	29.9
PLEASURE CRAFT	*empty string*	1
PLEASURE CRAFT	*empty string*	6
RIVER SHIP	38	380
PLEASURE CRAFT	*empty string*	18
FISHING	5	2
PLEASURE CRAFT	*empty string*	5
PLEASURE CRAFT	9	9
FISHING	5	1

With import feature of MongoDB Compass, CSV or JSON files can be loaded in the database, and the types of fields can be specified manually. The only preprocessing step that had to be done after importing was to convert timestamps to the appropriate format and defining longitude and latitude as coordinates.

The conversion of UNIX epochs timestamps to Dates in "dynamic_locs" collection was done with the following commands in MongoSH:

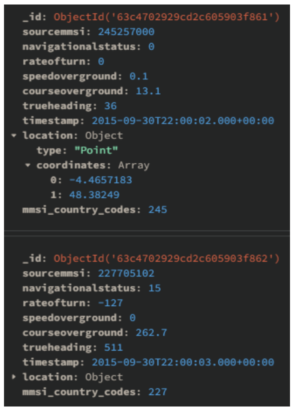
```
var new_docs= db.dynamic_locations.aggregate({$set:
{"timestamp": {$toDate: {$multiply: ['$t',
1000]}}}}).toArray()

db.dynamic_locs.insertMany(new_docs)
db.dynamic_locs.updateMany({}, {$unset: {t:""}})
```

MongoSH (Mongo Shell) is the command-line client for MongoDB, a popular, open-source NoSQL database. The Mongo Shell is a JavaScript interface to MongoDB and provides a way to interact with and query the database using JavaScript commands. It also allows you to perform administrative tasks such as creating, modifying, and deleting databases and collections.

The longitude and latitude fields of collection "dynamic_locs" were designated as coordinates in order for MongoDB Compass to recognize and utilize them as geospatial data. The commands used were:

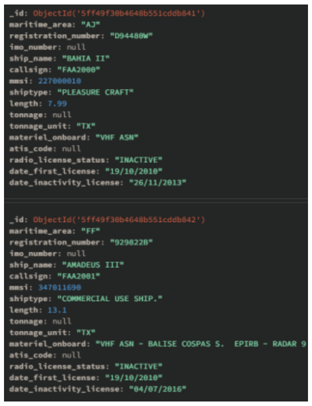
```
db.dynamic_locs.updateMany({},[{"$set":{"location":{type:
"Point",coordinates:["$lon", "$lat"] } }} ] )
db.dynamic_locs.updateMany({},{"$unset":{ lon: 1, lat: 1 }})
```

Here are some examples of the documents stored in the collections, that have been loaded into the MongoDB database.



```
  _id: ObjectId('63c4702929cd2c605903f861')
  sourcemmsi: 245257000
  navigationalstatus: 0
  rateofturn: 0
  speedoverground: 0.1
  courseoverground: 13.1
  trueheading: 36
  timestamp: 2015-09-30T22:00:02.000+00:00
▾ location: Object
    type: "Point"
  ▾ coordinates: Array
      0: -4.4657183
      1: 48.38249
  mmsi_country_codes: 245

  _id: ObjectId('63c4702929cd2c605903f862')
  sourcemmsi: 227705102
  navigationalstatus: 15
  rateofturn: -127
  speedoverground: 0
  courseoverground: 262.7
  trueheading: 511
  timestamp: 2015-09-30T22:00:03.000+00:00
▸ location: Object
  mmsi_country_codes: 227
```

"dynamic_locs" collection



```
_id: ObjectId('5ff49f30b4648b551cddb841')
maritime_area: "AJ"
registration_number: "D94480W"
imo_number: null
ship_name: "BAHIA II"
callsign: "FAA2000"
mmsi: 227000010
shiptype: "PLEASURE CRAFT"
length: 7.99
tonnage: null
tonnage_unit: "TX"
materiel_onboard: "VHF ASN"
atis_code: null
radio_license_status: "INACTIVE"
date_first_license: "19/10/2010"
date_inactivity_license: "26/11/2013"

_id: ObjectId('5ff49f30b4648b551cddb842')
maritime_area: "FF"
registration_number: "929822B"
imo_number: null
ship_name: "AMADEUS III"
callsign: "FAA2001"
mmsi: 347011690
shiptype: "COMMERCIAL USE SHIP."
length: 13.1
tonnage: null
tonnage_unit: "TX"
materiel_onboard: "VHF ASN - BALISE COSPAS S.  EPIRB - RADAR 9
atis_code: null
radio_license_status: "INACTIVE"
date_first_license: "19/10/2010"
date_inactivity_license: "04/07/2016"
```

"vessels" collection

In this section, the implementation of the three use cases of queries that were executed in the system, will be described. These use cases are:

USE CASE 1: A relational query that searches for the maritime area (maritime_area) that contains the most pleasure boats (shiptype: "PLEASURE CRAFT") with length between 5 and 6 meters.

```
db.vessels.explain.aggregate([{$match: {$and: [{shiptype:
"PLEASURE CRAFT"}, {length: {$gt: 5, $lt: 6}}]}}, {$group:
{_id: "$maritime_area", count: { $sum: 1 }}}, {$sort: {count:
-1}}, {$limit: 1}])
```

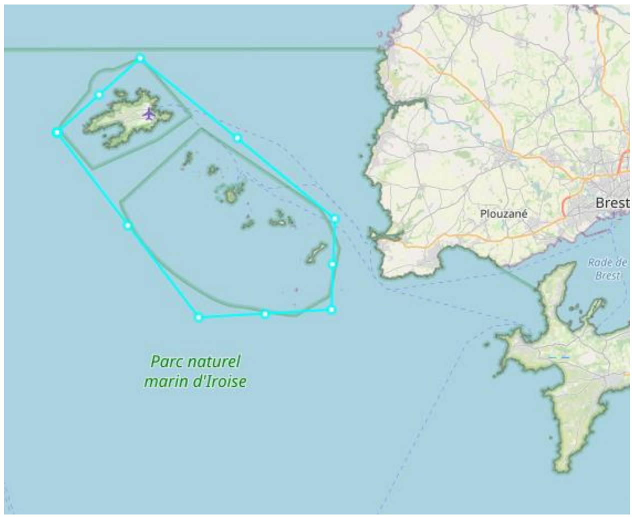
To support this query, indexes were created on the fields of shiptype, length and maritime_area.

```
db.vessels.createIndex({shiptype: 1, length: 1,
maritime_area:1})
db.vessels.createIndex({length: 1, shiptype: 1,
maritime_area:1})
db.vessels.createIndex({length: 1})
```

USE CASE 2: A spatial query that searches for vessel codes (sourcemmsi) that have been found at any time within the marine natural park of Iroise, which is located west of the French city of Brest.

db.dynamic_locs.aggregate([{$match: {location: {$geoWithin: {$geometry: {type: "Polygon" , "coordinates": [[[-5.1759338,48.448333],[-5.1759338,48.448333],[-5.0008392,48.2955285],[-4.8353577,48.302837],[-4.8319244,48.3772333],[-5.072937,48.5097813],[-5.1759338,48.448333]]]}}}}}, {$group: {_id: "$sourcemmsi"}},{$group: {_id: null,sourcemmsi: {$addToSet: "$_id"}}},{$project: {_id: 0, sourcemmsi: 1}}])

The Park naturel marin d'Iroise is a protected area. It is classified as a Marine Nature Reserve (Natura 2000) by the European Union and as a traditional Coastal Area by the French government. The purpose of the protection is to maintain and restore the environment and the natural human value of the area, as well as to protect the countless species that exist there. The polygon of the following image were used for the spatial query. This polygon surrounds the Park.



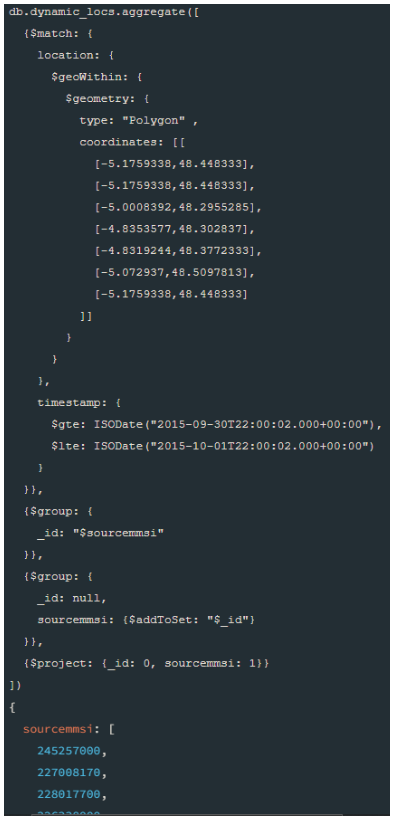
Marina Park Iroise

To support the use case 2 query, a 2dsphere index were created on the "location".

db.dynamic_locs.createIndex({"location":"2dsphere"})

A 2dsphere index is a type of index in MongoDB that is used to support spatial queries on geospatial data stored in the format of GeoJSON. The index is used to efficiently locate documents within a collection that are within a specified geographical area or that meet other spatial criteria.

USE CASE 3: A spatio-temporal query that searches for the codes of the ships (sourcemmsi) which had been present within the protected area of the Iroise park, in a specific time range.



```
db.dynamic_locs.aggregate([
  {$match: {
    location: {
      $geoWithin: {
        $geometry: {
          type: "Polygon" ,
          coordinates: [[
            [-5.1759338,48.448333],
            [-5.1759338,48.448333],
            [-5.0008392,48.2955285],
            [-4.8353577,48.302837],
            [-4.8319244,48.3772333],
            [-5.072937,48.5097813],
            [-5.1759338,48.448333]
          ]]
        }
      }
    },
    timestamp: {
      $gte: ISODate("2015-09-30T22:00:02.000+00:00"),
      $lte: ISODate("2015-10-01T22:00:02.000+00:00")
    }
  }},
  {$group: {
    _id: "$sourcemmsi"
  }},
  {$group: {
    _id: null,
    sourcemmsi: {$addToSet: "$_id"}
  }},
  {$project: {_id: 0, sourcemmsi: 1}}
])
{
  sourcemmsi: [
    245257000,
    227008170,
    228017700,
```

To support this query, the 2dsphere index was used. This index allowed efficient querying of the data based on spatial relationships. It should be noted that a compound index including timestamps was considered, but it was found to consume a significant amount of memory. Therefore, the 2dsphere index alone was used to support this query.

Unfortunately the resources and time available for the project were limited, and implementing partitioning and scalability would have required additional effort and resources.

5 Performance Analysis and Results

In this section, will be presented the results and performance evaluation of the three use cases of queries that were executed in the system.

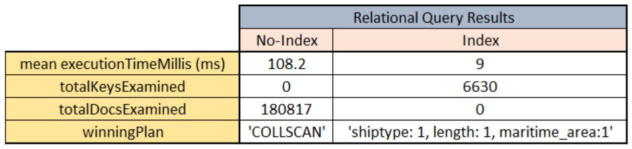
Mean execution time in milliseconds (ms) was the main metric used to evaluate the performance of the queries.

`

Additionally, the total number of keys and documents were also examined.

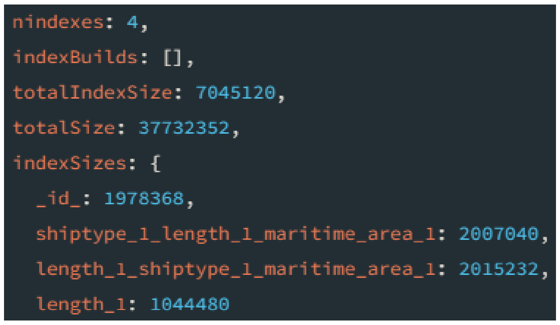
USE CASE 1, Results

For the relational query, the mean execution time for the query without an index was 108.2 milliseconds, with a total of 0 keys examined and 180817 documents examined. With an index on the shiptype, length and maritime_area fields, the mean execution time was significantly reduced to 9 milliseconds, with a total of 6630 keys examined and 0 documents examined.



	Relational Query Results	
	No-Index	Index
mean executionTimeMillis (ms)	108.2	9
totalKeysExamined	0	6630
totalDocsExamined	180817	0
winningPlan	'COLLSCAN'	'shiptype: 1, length: 1, maritime_area:1'

Also with **db.vessels.stats()** command, the size of the indexes can be shown in bytes.

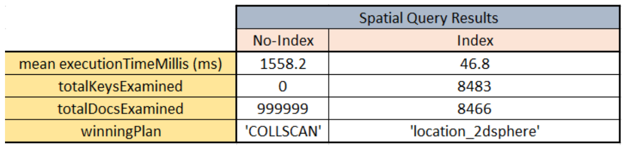


```
nindexes: 4,
indexBuilds: [],
totalIndexSize: 7045120,
totalSize: 37732352,
indexSizes: {
  _id_: 1978368,
  shiptype_1_length_1_maritime_area_1: 2007040,
  length_1_shiptype_1_maritime_area_1: 2015232,
  length_1: 1044480
```

Although it chooses the compound index during execution, it is clear that this index consumes a lot of memory.

USE CASE 2, Results

For the spatial query, the mean execution time for the query without an index was 1558.2 milliseconds, with a total of 0 keys examined and 999999 documents examined. With an index on field 'location' (2dsphere index), the mean execution time was significantly reduced to 46.8 milliseconds, with a total of 8483 keys examined and 8466 documents examined.

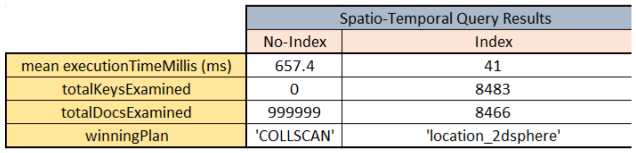


	Spatial Query Results	
	No-Index	Index
mean executionTimeMillis (ms)	1558.2	46.8
totalKeysExamined	0	8483
totalDocsExamined	999999	8466
winningPlan	'COLLSCAN'	'location_2dsphere'

It's also visible that the winningPlan after the creation of the 2dsphere index, is the 'location_2dsphere'.

USE CASE 3, Results

For the spatio-temporal query, the mean execution time for the query without an index was 657.4 milliseconds, with a total of 0 keys examined and 999999 documents examined. With an index on the location field, mean execution time was reduced to 41ms, with a total of 8483 keys examined and 8466 documents examined.



	Spatio-Temporal Query Results	
	No-Index	Index
mean executionTimeMillis (ms)	657.4	41
totalKeysExamined	0	8483
totalDocsExamined	999999	8466
winningPlan	'COLLSCAN'	'location_2dsphere'

6 Conclusion

In conclusion, this project demonstrates the effectiveness of using MongoDB as a storage layer for AIS data, and the importance of indexing in improving query performance. The relational query showed a significant improvement in execution time when an index was used, with an execution 12 times faster. The spatial and spatio-temporal queries also showed a significant improvement in execution time when a 2dsphere index was used. The spatial query was 33 times faster with index and the spatio-temporal was 39 time faster. Additionally, the use of indexing resulted in a reduction in the total number of keys and documents examined in all queries. Overall, the results of this study demonstrate the effectiveness of using MongoDB and indexing in managing AIS data and support efficient relational, spatial and spatio-temporal queries. This system can be used to support decision making in maritime operations, such as tracking vessels and identifying potential threats.

In future work, partitioning and sharding can be applied to the system to further improve its scalability and performance. This would allow for distributing the data and queries across multiple servers, which can handle larger amounts of data and improve query response times.

REFERENCES

[1] Rayab, C., Dréo, R., Camossi, E., Jousselme, A-L., & Ipharé, C. (2019). Heterogeneous integrated dataset for Maritime Intelligence, surveillance, and reconnaissance. Data in Brief, 22, 104141. https://doi.org/10.1016/j.dib.2019.104141

[2] Rayab, C., Dréo, R., Camossi, E., Jousselme, A-L., & Ipharé, C. (2016). Heterogeneous integrated dataset for Maritime Intelligence, surveillance, and reconnaissance. Zenodo. http://doi.org/10.5281/zenodo.1167595

[3] MongoDB. (n.d.). Query Documents. MongoDB Manual. https://www.mongodb.com/docs/manual/tutorial/query-documents/

[4] MongoDB. (n.d.). Indexes. MongoDB Manual. https://www.mongodb.com/docs/manual/indexes/