Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal data

Antonios Makris
Dept. of Informatics and Telematics, Harokopio University of Athens
Athens, Greece
amakris@hua.gr

Konstantinos Tserpes
Dept. of Informatics and Telematics, Harokopio University of Athens
Athens, Greece
tserpes@hua.gr

Giannis Spiliopoulos
MarineTraffic
London, United Kingdom
giannis.spiliopoulos@marinetraffic.com

Dimosthenis Anagnostopoulos
Dept. of Informatics and Telematics, Harokopio University of Athens
Athens, Greece
dimosthe@hua.gr

ABSTRACT

Several modern day problems need to deal with large amounts of spatio-temporal data. As such, in order to meet the application requirements, more and more systems are adapting to the specificities of those data. The most prominent case is perhaps the data storage systems, that have developed a large number of functionalities to efficiently support spatio-temporal data operations. This work is motivated by the question of which of those data storage systems is better suited to address the needs of industrial applications. In particular, the work conducted, set to identify the most efficient data store system in terms of response times, comparing two of the most representative of the two categories (NoSQL and relational), i.e. MongoDB and PostgreSQL. The evaluation is based upon real, business scenarios and their subsequent queries as well as their underlying infrastructures, and concludes in confirming the superiority of PostgreSQL. Specifically, PostgreSQL is four times faster in terms of response time in most cases and presents an average speedup around 2 in first query, 4 in second query and 4,2 in third query in a five node cluster. Also, we observe that the average response time is significantly reduced at half with the use of indexes almost in all cases, while the reduction is significantly lower in PostgreSQL.

1 INTRODUCTION

The volumes of spatial data that modern-day systems are generating has met staggering growth during the last few years. Managing and analyzing these data is becoming increasingly important, enabling novel applications that may transform science and society. For example, mysteries are unravelled by harnessing the 1 TB of data that is generated per day from NASA's Earth Observing System [1], or the more than 140 GB of raw science spatial data every week generated by space Hubble telescope [2]. At the same time, numerous business applications are emerging by processing the 285 billion points regarding aircraft movements per year gathered from the Automatic Dependent Surveillance Broadcast (ADS-B) system [3] and the 60Mb of AIS and weather data collected every second by MarineTraffic's on-line monitoring service [4] or the 4 millions geotagged tweets daily produced at Twitter [5].

Distributed database systems have been proven instrumental in the effort to dealing with this data deluge. These systems are distinguished by two key-characteristics: a) system scalability: the underlying database system must be able to manage and store a huge amount of spatial data and to allow applications to efficiently retrieve it; and, b) interactive performance: very fast response times to client requests. Some systems that natively meet those requirements for spatial data are: Hadoop-GIS [6] and SpatialHadoop [7], MD-HBase [8], GeoMesa [9], SMASH [10] and systems that use spatial resilient distributed datasets (SRDD) such as SpatialSpark [11], GeoTrellis [12] and GeoSpark [13].

The plethora of available systems and underlying technologies have left the researchers and practitioners alike puzzled as to what is the best option to employ in order to solve their big spatial data problem at hand. The query and data characteristics only add to the confusion. It is imperative for the research community to contribute to the clarification of the purposes and highlight the pros and cons of certain distributed database platforms. This work aspires to contribute towards this direction by comparing two suchlike platforms for a particular class of requirements, i.e. those that the response time in complex spatio-temporal queries is of high importance.

In particular, we compare the performance in terms of response time between a scalable document-based NoSQL datastore-MongoDB [14] and an open source object-relational database system (ORDBMS)-PostgreSQL [15] with the PostGIS extension. PostGIS is a spatial extender that adds support for geographic objects. The performance is measured using a set of spatio-temporal queries that mimic real case scenarios that performed in a dataset provided by MarineTraffic[1]. The evaluation of the systems was examined in different scenarios; in a 5 node cluster setup versus 1 node implementation and with the use of indexes versus not. Each database system was deployed on EC2 instances on Amazon Web Services (AWS)[2] and for storing/retrieving the data we used the Amazon S3 bucket.

The results show that PostgreSQL with the PostGIS extension, outperforms MongoDB in all queries. Specifically, PostgreSQL is four times faster in terms of response time in most cases and presents an average speedup around 2 in the first query, 4 in the second query and 4,2 in a third query in a 5 node cluster. The 5 node cluster setup outperforms the one node implementation

© 2019 Copyright held by the author(s). Published in the Workshop Proceedings of the EDBT/ICDT 2019 Joint Conference (March 26, 2019, Lisbon, Portugal) on CEUR-WS.org.

[1]MarineTraffic is an open, community-based maritime information collection project, which provides information services and allows tracking the movements of any ship in the world. It is available at: https://www.marinetraffic.com

[2]Amazon Web Services, https://aws.amazon.com/

CEUR-WS.org/Vol-2322/BMDA_3.pdf

in each system although the reduction is much more noticeable in PostgreSQL. Finally the results demonstrate that indexing affects response times in query execution by reducing them at half almost in all cases while the reduction is significantly lower with the use of indexes in PostgreSQL.

The document is structured as follows: Section 2 provides details about the related work in spatio-temporal systems and benchmark analysis; Section 4 describes the technology overview; Section 4 describes the evaluation of spatio-temporal database systems used; Section 5 presents the experimental results while Section 6 presents the final conclusions of this study and future work.

2 RELATED WORK

The volume of spatial data is increasing exponentially on a daily basis. Geospatial services such as GPS systems, Google Maps and NASA's Earth Observing system are producing terabytes of spatial data every day and in combination with the growing popularity of location-based services and map-based applications, there is an increasing demand in the spatial support of databases systems. There are challenges in managing and querying the massive scale of spatial data such as the high computation complexity of spatial queries and the efficient handling the big data nature of them. There is a need for an interactive performance in terms of response time and a scalable architecture. Benchmarks play a crucial role in evaluating the performance and functionality of spatial databases both for commercial users and developers.

In [16] are presented some database benchmarks such as Wisconsin which was developed for the evaluation of relational database systems [16], AS^3AP that contains a mixed workload of database transactions, queries and utility functions and SetQuery that supports more complex queries and designed to evaluate systems that support decisions making. Above benchmarks measure the performance of the system in general, but there are also benchmarks that are explicitly designed to evaluate the capabilities of spatio-temporal databases such as SEQUOIA 2000 [17] and Paradise Geo-Spatial DBMS (PGS-DBMS) [18]. SEQUOIA 2000 propose a set of 11 queries to evaluate the performance while PGS-DBMS presents 14 queries (the first nine queries are the same in both benchmark systems). Although above benchmarks seems to be adequate to evaluate a spatial database, things change when the evaluation consists the temporal factor. Only a few queries from both benchmarks have a temporal component. The 3-Dimensional spatio-temporal benchmark, expands the benchmarks into 3 dimensions in order to simulate real life scenarios. The main difference from the other systems is the addition of two new features: temporal processing/temporal updates and three dimensional support.

Another benchmark for spatial database evaluation is presented in [19]. Although a number of other benchmarks limited to a specific database or application, Jackpine presents one important feature, portability in terms that can support any database (JDBC driver implementation). It supports micro benchmarking that is a number of spatial queries, analysis and loading functions with spatial relationships and macro benchmarking with queries which address real world problems. Also includes all vector queries from the SEQUOIA 2000 benchmark.

In [20] the authors compare five Spark based spatial analytics systems (SpatialSpark, GeoSpark, Simba, Magellan, LocationSpark) using five different spatial queries (range query, kNN query, spatial joins between various geometric datatypes, distance join, and kNN join) and four different datatypes (points, linestrings, rectangles, and polygons). In order to evaluate these modern, in-memory spatial systems, real world datasets are used and the experiments are focusing on major features that are supported by the systems. The results show the strengths and weaknesses of the compared systems. In specific, GeoSpark seems to be the most complete spatial analytic system because of data types and queries supported.

In [9] are presented and evaluated two distributed database technologies, GeoMESA which focuses on geotemporal indexes and Elasticsearch which is a document oriented data store that can handle arbitrary data which may have a geospatial index. In general GeoMesa is an open-source, distributed, spatio-temporal database built on a number of distributed cloud data storage systems, including Accumulo, HBase, Cassandra, and Kafka. It can provide spatio-temporal indexing for BigTable and its clones (HBase, Apache Accumulo) using space filling curves to project multi-dimensional spatio-temporal data into the single dimension linear key space imposed by the database. On the other hand Elasticsearch uses Z-order spatial-prefix-based indexes that work for all types of vector data (points, lines and polygons) as well as a Balanced KD-tree which works better for point data. For batch processing, GeoMESA leverages Apache Spark and for stream geospatial event processing, Apache Storm and Apache Kafka.

A computing system for processing large-scale spatial data called GeoSpark, is presented in [13]. Apache Spark is an in-memory cluster computing system that provides a data abstraction called Resilient Distributed Datasets (RDDs) which consist of collections of objects partitioned across a cluster. The main drawback is that it does not support spatial operation on data. This gap is filled by GeoSpark which extends the core of Apache Spark to support spatial data types, spatial indexes and computations. GeoSpark provides a set of out-of-the-box Spatial Resilient Distributed Dataset (SRDD) types that provide support for geometrical and distance operations and spatial data index strategies which partition the SRDDs using a grid structure and thereafter assign grids to machines for parallel execution.

SMASH [10] is a highly scalable cloud based solution and the technologies involved with SMASH architecture are: GeoServer, GeoMesa, Apache Accumulo, Apache Spark and Apache Hadoop. SMASH is a collection of software components that work together in order to create a complete framework which can tackle issues such as fetching, searching, storing and visualizing the data directly for the demands of traffic analytics. For spatio-temporal indexing on geospatial data, GeoMesa and GeoServer are used. GeoMesa provides spatio-temporal indexing on top of the Accumulo BigTable DBMS and is able to provide high levels of spatial querying and data manipulation, leveraging a highly parallel indexing strategy using a geohashing algorithm (three-dimensional Z-order curve). GeoServer is used to serve maps and vector data to geospatial clients and allows users to share, process and edit geospatial data.

Finally in [6] is presented a system called Hadoop-GIS, a scalable and high performance spatial data warehousing system which can efficiently perform large scale spatial queries on Hadoop. It provides spatial data partitioning for task parallelization through MapReduce, an index-driven spatial query engine to support various types of spatial queries (point, join, cross-matching and nearest neighbor), an expressive spatial query language by extending HiveQL with spatial constructs and boundary

handling to generate correct results. In order to achieve high performance, the system partitions time consuming spatial query components into smaller tasks and process them in parallel while preserving the correct query semantics.

3 TECHNOLOGY OVERVIEW

In order to evaluate the set of spatio-temporal queries, two different systems are employed and compared: MongoDB and PostgreSQL with PostGIS extension.

MongoDB [21] is an open source document based NoSQL datastore which is supported commercial by 10gen. Although MongoDB is non-relational, it implements many features of relational databases, such as sorting, secondary indexing, range queries and nested document querying. Operators like create, insert, read, update and remove as well as manual indexing, indexing on embedded documents and index location-based data also supported. In such systems, data are stored in collections called documents which are entities that provide some structure and encoding on the managed data. Each document is essentially an associative array of a scalar value, lists or nested arrays. Every document has a unique special key "ObjectId", used for explicitly identification while this key and the corresponding document are conceptually similar to a key-value pair. MongoDB documents are serialized naturally as Javascript Object Notation (JSON) objects and stored internally using a binary encoding of JSON called BSON [22]. As all NoSQL systems, in MongoDB there are no schema restrictions and can support semi-structured data and multi-attribute lookups on records which may have different kinds of key-value pairs [23]. In general, documents are semi-structured files like XML, JSON, YALM and CSV. For data storing there are two ways: a) nesting documents inside each other, an option that can work for one-to-one or one-to-many relationships and b) reference to documents, in which the referenced document only retrieved when the user requests data inside this document. To support spatial functionality, data are stored in GeoJSON which is a format for encoding a variety of geographical data structures [24]. GeoJSON supports: a) Geometry types as Point, LineString, Polygon, MultiPoint, MultiLineString and MultiPolygon, b) Feature, which is a geometric object with additional properties and c) FeatureCollection, which consist a set of features. Each GeoJSON document is composed of two fields: i) Type, the shape being represented, which informs a GeoJSON reader how to interpret the "coordinates" field and ii) Coordinates, an array of points, the particular arrangement of which is determined by "type" field. The geographical representation need to follow the GeoJSON format structure in order to be able to set a geospatial index on the geographic information. First, MongoDB computes the geohash values for the coordinate pairs and then indexes these geohash values. Indexing is an important factor to speed up query processing. MongoDB provides BTree indexes to support specific types of data and queries such as: Single Field, Compound Index, Multikey Index, Text Indexes, Hashed Indexes and Geospatial Index. To support efficient queries on geospatial coordinate data, MongoDB provides two special indexes: 2d index that uses planar geometry when returning results and 2dsphere index that use spherical geometry to return results. A 2dsphere index supports queries that calculate geometries on an earth-like sphere and can handle all geospatial queries: queries for inclusion, intersection and proximity. It supports four geospatial query operators for spatio-temporal functionality: $geoIntersects, $geoWithin, $near and $nearSphere and uses the WGS84 reference system for geospatial queries on GeoJSON objects.

On the other hand, PostgreSQL is an open source object-relational database system (ORDBMS). There is a special extension available called PostGIS that integrates several geofunctions and supports geographic objects. PostGIS implementation is based on "light-weight" geometries and the indexes are optimized to reduce disk and memory usage. The interface language of the PostgreSQL database is the standard SQL. PostGIS has the most comprehensive geofunctionalities with more than one thousand spatial functions. It supports geometry types for Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultipPolygons and GeometryCollections. There are several spatial operators for geospatial measurements like area, distance, length and perimeter. PostgreSQL supports several types of indexes such as: BTree, Hash, Generalized Inverted Indexes (GIN) and Generalized Search Tree (GiST) called R-tree-over-GiST. The default index type is BTree that can work with all datatypes and can be used for equality and range queries efficiently. For general balanced tree structures and high-speed spatial querying, PostgreSQL uses GiST indexes that can be used to index geometric data types, as well as full-text search.

Feature	Description
ship_id	Unique identifier for each ship
latitude, longitude	Geographical location in digital degrees
status	Current position status
speed	Speed over ground in knots
course	Course over ground in degrees with 0 corresponding to north
heading	Ship's heading in degrees with 0 corresponding to north
timestamp	Full UTC timestamp

Table 1. Dataset Attributes

4 EVALUATING SPATIO-TEMPORAL DATABASES

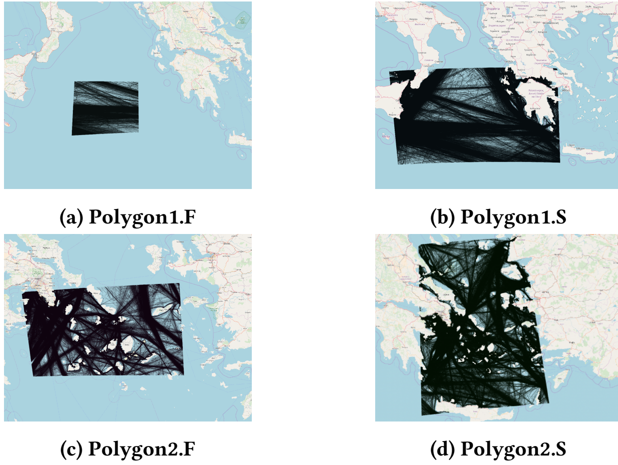
4.1 Dataset Overview

In order to evaluate the performance of the spatio-temporal databases, we employed a dataset (11 GB), which was provided to us by the community based AIS vessel tracking system (VTS) of MarineTraffic. The dataset provides information for 43.288 unique vessels and contains 146.491.511 AIS records in total, each comprising 8 attributes as described in Table 1. The area that our dataset covers is bounded by a rectangle within Mediterranean sea. The vessels have been monitored for a 3 months period starting at May 1st, 2016 and ending at July 31th, 2016.

4.2 Use Case - Queries

To test the performance of each spatial database we utilize a set of queries that mimic real world scenarios. We employed 3-three complex spatio-temporal queries and the reason of the selection of these particular queries is because they contain spatial and temporal predicates:

(1) Find coordinates of different amount of vessels from 1/May/2016 - 31/July/2016 (entire time window) within the whole bounded area, Q1



(a) Polygon1.F (b) Polygon1.S (c) Polygon2.F (d) Polygon2.S

Figure 1. Geographical polygons with trajectories of the vessels

Amount of ships	Records returned
43.288	146.491.511
21.644	72.349.832
10.822	36.928.530
5.481	18.909.184

Table 2. Q1 **results**

Time window	Records Returned	Amount of ships
2 months	95.332.760	37.368
1 month	48.884.829	31.531
10 days	14.362.160	22.403
1 day	1.142.337	11.122

Table 3. Q2 **results**

(2) Find coordinates of vessels for different time windows within the whole bounded area, Q2
(3) Find coordinates of vessels for different geographical polygons within the entire time window, Q3

Specifically, Q1 fetches coordinates for an increased amount of vessels as shown in Table 2. The query is performed for all unique vessels in the dataset, for half of them, for 1/4 and finally for 1/8 of them, in order to examine the scalability of the database systems. Q2 fetches coordinates of vessels for different time windows; 1 day, 10 days, 1 month and 2 months as shown in Table 3. Q3 fetches coordinates for different geographical polygons as shown in Table 4. The polygons (polygon1.F, polygon2.F, polygon1.S, polygon2.S) are bounding boxes within Mediterranean Sea. Figure 1 shows a graphical representation of these polygons with the trajectories of the vessels inside. We used QGIS [3], a tool that visualize, analyses and publish geospatial information.

4.3 System Architecture

We have deployed a MongoDB cluster that contains a master node server (primary) and four replication slaves (secondaries) in Replica Set mode. One way to achieve replication in MongoDB

[3]QGIS: A Free and Open Source Geographic Information System, https://qgis.org/en/site/

Polygons	Records Retuned	Amount of ships
polygon2.S	15.502.808	8.854
polygon1.S	11.564.115	10.730
polygon2.F	5.194.874	6.185
polygon1.F	745.902	4.182

Table 4. Q3 **results**

is by using replica set. While MongoDB offers standard primary-secondary replication, it is more common to use MongoDB's replica sets. A replica set is a group of multiple coordinated instances that host the same dataset and work together to ensure superior availability. In a replica set, only one node acts as primary that receives all write operations while the other instances called secondaries and apply operations from the primary. All data are replicated from the primary to secondary nodes. If the primary node ever fails or becomes unavailable or maintained, one of the replicas will automatically be elected through a consortium as the replacement. After the recovery of failed node, it joins the replica set again and works this time as a secondary node.

The problem with replica set configuration is the selection of a member to perform a query in case of read requests. The question is which node to choose, the primary or a secondary and if a request queries a secondary, which one should be used. In MongoDB it is possible to control this choice with *read preferences* solution. When an application queries a replica set, there is the opportunity to trade off consistency, availability, latency and throughput for each kind of query. This is the problem that read preferences solve: how to specify the read preferences among above trade offs, so the request falls to the best member of the replica set for each query. For the distribution of queries, the read preference provides the solution. The categories of read preferences are: i) PRIMARY: Read from the primary, ii) PRIMARY PREFERRED: Read from the primary if available, otherwise read from a secondary, iii) SECONDARY: Read from a secondary, iv) SECONDARY PREFERRED: Read from a secondary if available, otherwise from the primary and finally v) NEAREST: Read from any available member. Because our goal is to achieve maximum throughput and an evenly distribution of load across the members of the set we used NEAREST preference and we set the value *secondary_acceptable_latency_ms* very high in 500ms. This value is used in MongoDB to track each member's ping time and queries only the "nearest" member, or any random member that is no more than the default value of 15ms "farther" than it. In general, load balancing and query distribution consist one of the most important factors in distributed systems. An even distribution of load within the nodes of the system reduces the probability that a node turns to a hotspot and his property also acts as a safeguard to the system reliability [25].

Additionally, we have deployed a PostgreSQL cluster that contains a master server and four slaves in Streaming Replication mode. Slaves keep an exact copy of master's data, apply the stream to their data and staying in a "Hot Standby" mode, ready to be promoted as master in case of failure. Streaming replication is a good tactic for data redundancy between nodes but for load balancing a mechanism is required that splits the requests between the copies of data. For this reason we use Pgpool-2[4], a

[4]Pgpool-2, http://www.pgpool.net/mediawiki/index.php/Main_Page

middleware that works between PostgreSQL servers and a PostgreSQL database client. Pgpool-2 examines each query and if the query is read only, it is forwarded to one of the slave nodes otherwise in case of write it is forwarded to the master server. With this configuration the read load splits between the slave nodes of the cluster, achieving thus an improved system performance. Pgpool-2 provides the following features, Connection Pooling: connections are saved and reused whenever a new connection with the same properties arrives, thus reducing connection overhead and improving system throughput, Replication: replication creates a real time backup on physical disks, so that the service can continue without stopping servers in case of a disk failure, Load Balancing: the load on each server is reduced by distributing SELECT queries among multiple servers, thus improving system's overall throughput and Limiting Exceeding Connections: connections are rejected after a limit on the maximum number of concurrent connections.

4.4 Data Ingestion

The dataset used for the experiments was initially in CSV format. As we mentioned above, in MongoDB the geographical representation needs to follow the GeoJSON format structure in order to be able to set a geospatial index on the geographic information, thus the first step was the conversion of the data into the appropriate format. For data ingestion we used the *mongoimport* tool to import data into MongoDB database. The total size the dataset occupied in the collection in MongoDB is 116 GB and each record has a size of about 275 bytes.

In PostgreSQL there is a copy mechanism for bulk loading data that can achieve a good throughput. The data must be in CSV format and the command can accept a number of delimiters. The next step after data loading into database, is the conversion of latitude and longitude columns to PostGIS POINT geometry. These columns must be converted into geometry data which subsequently can be spatially queried. We created a column called *the_geom* using a defined PostGIS function, which in essence contains the POINT geometry created from latitude and longitude of each record. The spatial reference ID (SRID) of the geometry instance (latitude, longitude) is 4326 (WGS84). The World Geodetic System (WGS) is the defined geographic coordinate system (three-dimensional) for GeoJSON used by GPS to express locations on the earth. The latest revision is WGS 84 (also known as WGS 1984, EPSG:4326) [26]. The total size the dataset occupied in PostgreSQL db is 32 GB and each row's size is about 96 bytes while in MongoDB is almost 4 times larger.

The reason for this behavior is that the data stored in MongoDB are in GeoJson format and each record consist of many extra characters and a unique auto created id called ObjectId. Thus, each record is significant bigger in size than it was in its original CSV format. On the other hand, in PostgreSQL the data ingested in database as CSV, with the addition of *the_geom* column that contains the POINT geometries of each latitude and longitude.

4.5 Cluster Setup - AWS

To benchmark MongoDB and PostgreSQL we deployed each database system on Amazon Web Services (AWS) EC2 instances. For storing the dataset, we used the S3 bucket provided also by AWS. The configuration used is described below:

MongoDB. The MongoDB cluster consists of 5 x r4.xlarge instances within a Virtual Private Cloud (Amazon VPC). One node is the primary and the other played the role of the replicas. We installed MongoDB 3.6.5 version in Replica Set mode. Each instance operates on Amazon Linux 2 AMI OS and consist of 4 CPUs x 2.30 GHz, 30.5 GB DDR4 RAM, 500 GB of general purpose SSD storage type EBS, up to 10 Gigabit network performance and IPv6 support. Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in the AWS Cloud. Each EBS volume is automatically replicated within its Availability Zone to protect from component failures, thus offering high availability and durability. Also the instances are EBS-optimized which means that they provide additional throughput for EBS I/O and as a result an improved performance.

PostgreSQL. Exactly the same configuration is used in PostgreSQL. We installed PostgreSQL 9.5.13 and PostGIS 2.2.1 in streaming replication mode. One node is the master while the others play the role of slaves. Also an extra instance for Pgpool-2 was deployed.

5 EXPERIMENTS

This section contains a detailed experimental evaluation that examines the run time performance of the spatio-temporal queries Q1, Q2 and Q3 in MongoDB and PostgreSQL through seven experiments. Five consecutive separate calls are conducted, in order to gather the experimental results and collect the average values concerning response time of above queries in different case scenarios. We compare the response time in a 5-node cluster versus a 1-node implementation. We also test the settings when indices are used against the case that they are not.

For Q1 a regular BTree index is created in MongoDB and PostgreSQL for attribute "ship_id". The size of index varies between the different database systems. In MongoDB the index size is about 6 GB while in PostgreSQL the size is 3,1 GB. For Q2 we implemented also a BTree index of size 6 GB in both DBMSs for attribute "timestamp". Finally, for Q3 we implemented an index in field "$geometry" in MongoDB with size 6 GB. As mentioned above the data are stored in MongoDB as GeoJSON and the "$geometry" field that is created contains the coordinates values, latitude and longitude. Because these data are geographical, we create a 2dsphere index type which supports geospatial queries. In PostgreSQL we created an index of type GiST in field *the_geom* which contains the POINT geometry created from latitude and longitude of each record with size 8,2 GB. For high-speed spatial querying, PostgreSQL uses GiST indexes that can be used to index geometric data types. The index size varies between the two database systems even for the same attribute that performed. The two systems store data differently and the concept of "index" is different too.

Figure 2 illustrates the average response time concerning the set of queries Q1, Q2 and Q3 in 5 node cluster between MongoDB and PotgreSQL. It's quite clear that PostgreSQL outperforms MongoDB by a large extent in all queries. The response time is almost 4 times faster in some cases (Q2, Q3) comparing to MongoDB. Only in Q1 the response time presents smaller fluctuations between the DBMSs. Exactly the same behavior is observed in the 1 node implementation as shown in Figure 3. The response time is significant lower in case of PostgreSQL.

Subsequently, we compared the average response time in the set of queries for the 5-nodes cluster versus the 1-node implementation for the two database systems as shown in Figure 4 and Figure 5 respectively. The results show that the average response

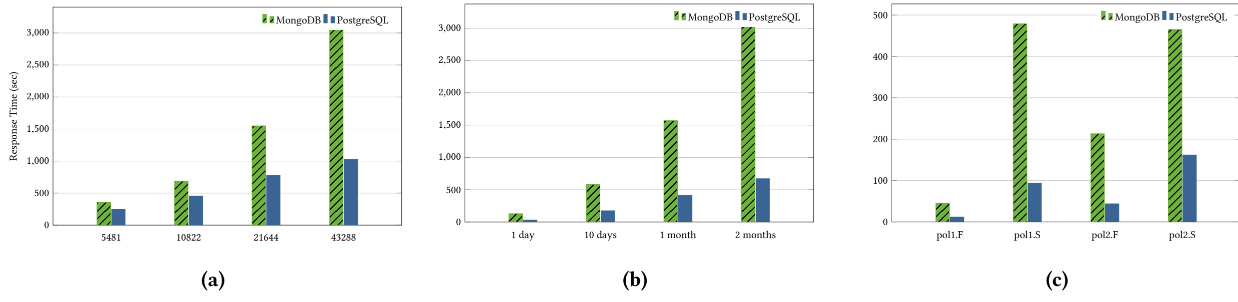


Figure 2. Average response time of Q1 (a), Q2 (b) and Q3 (c) in 5 node cluster between MongoDB and PostgreSQL.

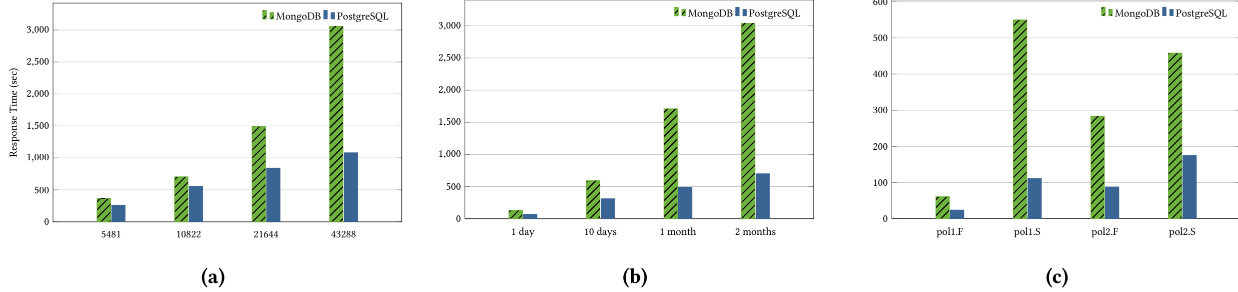


Figure 3. Average response time of Q1 (a), Q2 (b) and Q3 (c) in 1 node implementation between MongoDB and PostgreSQL.

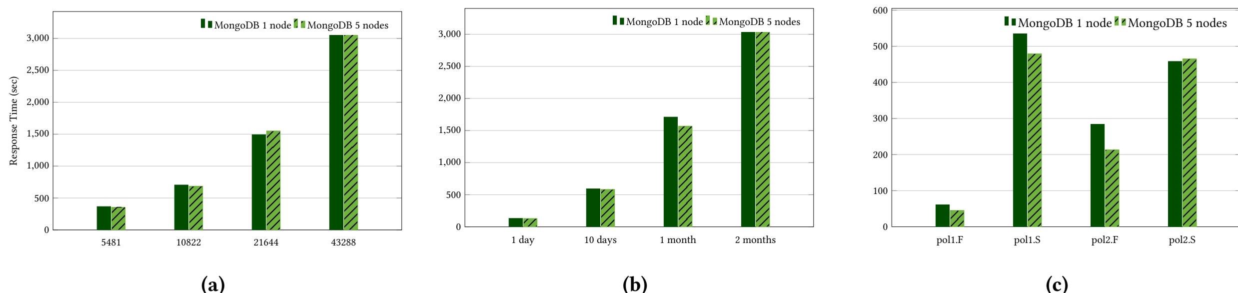


Figure 4. Average response time of Q1 (a), Q2 (b) and Q3 (c) between 5 nodes cluster and 1 node implementation in MongoDB.

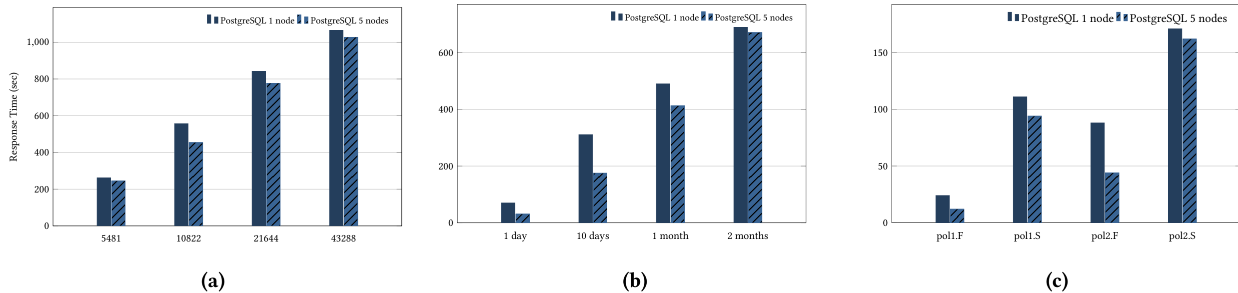


Figure 5. Average response time of Q1 (a), Q2 (b) and Q3 (c) between 5 nodes cluster and 1 node implementation in PostgreSQL.

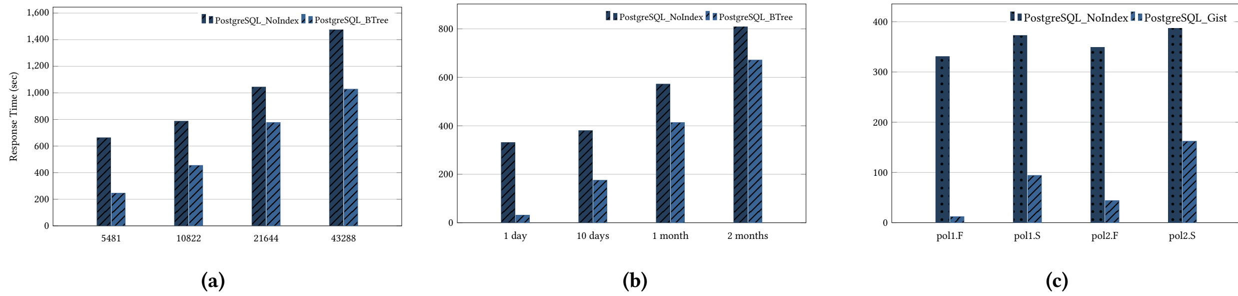


Figure 6. Average response time of Q1 (a), Q2 (b) and Q3 (c) in 5 nodes PostgreSQL cluster with indexing versus not.

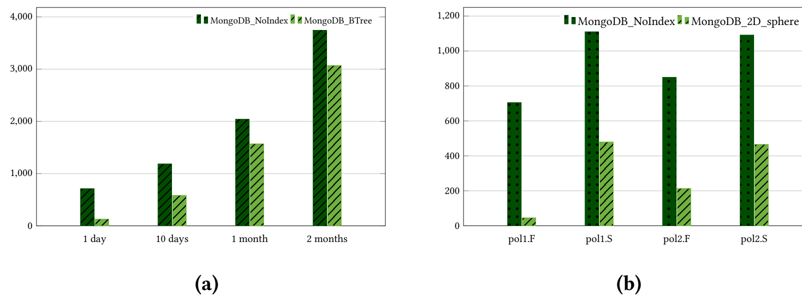


Figure 7. Average response time of Q2 (a) and Q3 (b) in 5 nodes MongoDB cluster with indexing versus not.

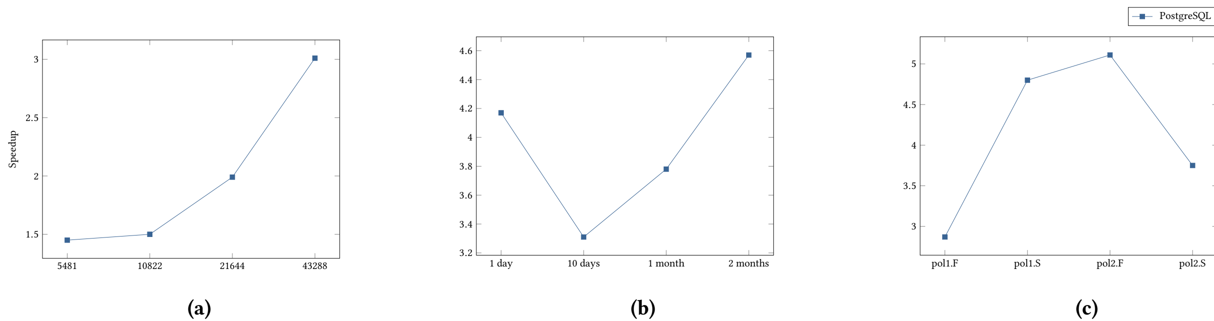


Figure 8. Average speedup of PostgreSQL over MongoDB in Q1 (a), Q2 (b) and Q3 (c) in 5 node cluster with the use of indexes.

time presents a small reduction when the experiments are performed in a 5-node MongoDB cluster versus the 1-node implementation. The reduction is much more noticeable in the case of PostgreSQL. The reason is that in the MongoDB replica set, the client requests are distributed using the read preference-Nearest option that can achieve a maximum throughput and an evenly distribution of load across the members of the set. In case of PostgreSQL, with the use of Pgpool-2 the load of read requests is distributed between the nodes of the cluster in a much more effective way, thus improving system's overall throughput.

In the next set of experiments we examined how indexes affect the response time in queries execution. As shown in Figure 6 the reduction is significantly lower, almost at half with the use of indexes in PostgreSQL. An index allows the database server to find and retrieve specific rows much faster than without an index. In Figure 7 concerning MongoDB we excluded Q1 because the response time was extremely high in case of no index (> 4 hours). In the remaining queries, also the response time is significantly reduced. Efficient query execution in MongoDB is supported using indexing. MongoDB can use indices to limit the number of documents it must inspect otherwise a scan operation is performed in every document in a collection, to select those documents that match the query statement (collection scan). Figure 8 presents the average speedups of PostgreSQL comparing to MongoDB in 5 node cluster with the use of indexes. As it shown the average speedup concerning Q1 is almost 2, 4 in case of Q2 and 4,2 in Q3.

6 CONCLUSIONS

In this paper, we analyzed and compared the performance in terms of response time between two different database systems, a document-based NoSQL datastore, MongoDB, and an open-source object-relational database system, PostgreSQL with PostGIS extension. Each database system was deployed on Amazon Web Services (AWS) EC2 cluster instances. We employed a replica set and a streaming replication cluster setup for MongoDB and PostgreSQL system respectively. For the evaluation between the two systems, we employed a set of spatio-temporal queries that mimic real world scenarios and present spatial and temporal predicates, with the use of a dataset which was provided to us by the community based AIS vessel tracking system (VTS) of MarineTraffic.

The performance is measured in terms of response time in different case scenarios; in a 5 node cluster versus 1 node implementation and with the use of indexes versus not. The results are show that PostgreSQL outperforms MongoDB in all cases and queries and presents an average speedup around 2 in first query, 4 in second query and 4,2 in third query in a five node cluster. Also, the replica set mode implementation of 5 nodes in MongoDB as well as the streaming replication of 5 nodes in PostgreSQL outperforms the one node implementation in each system respectively. Subsequently, as demonstrated by the experimental results, the average response time is significantly reduced at half, almost in all cases with the use of indexes while again the reduction is significantly lower in PostgreSQL. Finally, the dataset size occupied in the system db, reduced 4x in case of PostgreSQL, since it stores data in a more effective way.

Our future plan include the extension of our system architecture to what it is called Shared Cluster. A Shared Cluster consists of shards which in turn contain a subset of the sharded data. Sharding is a method for distributing data across multiple machines. Every machine contains only a portion of the shared data and each machine replicated to secondaries nodes for data redundancy and for fault tolerance reasons. We plan to evaluate and compare the two types of cluster and draw conclusions of which system is best for different cases.

ACKNOWLEDGMENTS

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. no. 2158).

This work has been developed in the frame of the MASTER project, which has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 777695.

This work was supported in part by MarineTraffic which provided data access for research purposes.

REFERENCES

[1] G Leptoukh. Nasa remote sensing data in earth sciences: Processing, archiving, distribution, applications at the ges disc. In *Proc. of the 31st Intl Symposium of Remote Sensing of Environment*, 2005.

[2] Telescope hubble site. http://hubble.stsci.edu/the_telescope/hubble_essentials/quick_facts.php. Accessed: 2018-7-15.

[3] Automatic dependent surveillance-broadcast (ads-b). https://www.faa.gov/nextgen/programs/adsb/. Accessed: 2018-7-15.

[4] Iraklis Varlamis, Konstantinos Tserpes, and Christos Sardianos. Detecting search and rescue missions from ais data. In *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2018.

[5] Luke Sloan and Jeffrey Morgan. Who tweets with their location? understanding the relationship between demographic characteristics and the use of geoservices and geotagging on twitter. *PloS one*, 10(11):e0142209, 2015.

[6] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013.

[7] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1352–1363. IEEE, 2015.

[8] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Md-hbase: design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, 31(2):289–319, 2013.

[9] Andrew Hulbert, Thomas Kunicki, James N Hughes, Anthony D Fox, and Christopher N Eichelberger. An experimental study of big spatial data systems. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 2664–2671. IEEE, 2016.

[10] Yikai Gong, Luca Morandini, and Richard O Sinnott. The design and benchmarking of a cloud-based platform for processing and visualization of traffic data. In *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, pages 13–20. IEEE, 2017.

[11] Simin You, Jianting Zhang, and Le Gruenwald. Large-scale spatial join query processing in cloud. In *2015 31st IEEE International Conference on Data Engineering Workshops (ICDEW)*, pages 34–41. IEEE, 2015.

[12] Ameet Kini and Rob Emanuele. Geotrellis: Adding geospatial capabilities to spark. *Spark Summit*, 2014.

[13] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 70. ACM, 2015.

[14] Peter Membrey, Eelco Plugge, Tim Hawkins, and DUPTim Hawkins. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Springer, 2010.

[15] Neil Matthew and Richard Stones. *Beginning Databases with PostgreSQL*. Apress, 2005.

[16] David J DeWitt. The wisconsin benchmark: Past, present, and future., 1993.

[17] Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith. The sequoia 2000 storage benchmark. In *ACM SIGMOD Record*, volume 22, pages 2–11. ACM, 1993.

[18] Jignesh Patel, JieBing Yu, Navin Kabra, Kristin Tufte, Biswadeep Nag, Josef Burger, Nancy Hall, Karthikeyan Ramasamy, Roger Lueder, Curt Ellmann, et al. Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. In *ACM SIGMOD Record*, volume 26, pages 336–347. ACM, 1997.

[19] Suprio Ray, Bogdan Simion, and Angela Demke Brown. Jackpine: A benchmark to evaluate spatial database performance. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1139–1150. IEEE, 2011.

[20] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. How good are modern spatial analytics systems? *Proceedings of the VLDB Endowment*, 11(11):1661–1673, 2018.

[21] Mongodb. https://www.mongodb.com/. Accessed: 2018-7-15.

[22] Binary json. http://bsonspec.org/. Accessed: 2018-7-15.

[23] Antonios Makris, Konstantinos Tserpes, Vassiliki Andronikou, and Dimosthenis Anagnostopoulos. A classification of nosql data stores based on key design characteristics. *Procedia Computer Science*, 97:94–103, 2016.

[24] The geojson format specification. http://geojson.org/geojson-spec.html. Accessed: 2018-7-15.

[25] Antonios Makris, Konstantinos Tserpes, and Dimosthenis Anagnostopoulos. A novel object placement protocol for minimizing the average response time of get operations in distributed key-value stores. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 3196–3205. IEEE, 2017.

[26] B LOUIS Decker. World geodetic system 1984. Technical report, Defense Mapping Agency Aerospace Center St Louis Afs Mo, 1986.