School of Informatics

Informatics Research Review
Performance Analysis of PostgreSQL and MongoDB in Processing Vector Geospatial Data

[redacted]

January 2021

Abstract

The explosion of the number of geotagged data leads to more performance challenges in managing and processing spatial data. This phenomenon has also transformed the availability of geo functions from a nice-to-have feature to a necessity in a database. Solutions based on relational databases have predominated the market for a long time. With its advantages in processing massive amounts of data, NoSQL databases are like promising alternatives to relational databases. This paper reviews explicitly performance evaluation of PostgreSQL with PostGIS extension and MongoDB in querying geospatial data, focusing on the vector data format. Both advantages and disadvantages as geospatial databases will also be investigated.

Date: Tuesday 19th January, 2021

Supervisor: [redacted]

1 Introduction

In recent past years the number of data are growing significantly due to the increasing use of the internet and smartphones. Most websites and mobile apps enable geolocation tagging to detect where the traffic comes from. Mobile devices and vehicles are equipped with GPS to track user location and movement. These things have contributed to the increase of geospatial data. The amount of geospatial data is forecasted to rise annually by 20%. It accounts for a large proportion of the global data produced daily, about 2.5 quintillion bytes [1]. This trend brings more challenges in storing and querying this type of data [2]. It also transforms the availability of geo functions from a nice-to-have feature to a necessity.

To cater to the demand, database systems must have the capability to manage a massive amount of geospatial data, provide fast response time, and allow applications to retrieve it efficiently [3]. The two kinds of database, relational (RDBMS) and non-relational (NoSQL/Not-only-SQL), are now equipped with geo-functions to store and analyse geospatial data. RDBMS is the most widely used database in various industries and adopts a fixed structure. It remains a popular option because of its convenience and easiness to understand table structure [4]. Meanwhile, NoSQL databases come with different natures that support a flexible storage schema and are often called schemaless databases. This type of database aligns with current geospatial applications which require more flexible schema [1]. In the context of the number of geo-functions, NoSQLs are still far behind RDBMS, only providing a few basic features. However, NoSQL databases grow in popularity because of the ease of access, speed, and horizontal scalability.

The options of two type databases give rise to performance questions: which one offers better performance in processing geospatial data? We can measure the performance based on query time, disk usage, memory usage, or CPU usage. However, we only managed to obtain one literature that evaluates disk and memory usage. Because of this reason, we refine the question to which one offers better latency in processing geospatial data? Another question is, what are the benefits and limitations of each database in processing geospatial data.

For this literature review, we select one database of each type to be evaluated: PostgreSQL and MongoDB. PostgreSQL, with its extension PostGIS offers the most comprehensive geo-functionalities among relational databases with more than one thousand functions [4]. Meanwhile, not all types of NoSQL database provide geospatial capability. MongoDB is one of the NoSQL types that support more geometry functions and spatial indexes than most other NoSQL databases [5]. Document databases like MongoDB are proven to be very effective in distributed storing and processing geospatial data [6].

This literature review will also focus on analysing geospatial data performance in vector representation only, excluding raster data. We will review query performance testings that were done on a single node environment, and some on cluster mode. Furthermore, we will examine how indexing can help to improve query time.

2 Geospatial Data and Database

In this section, we will first describe geospatial data in vector format representation to illustrate data and query that were used to compare database performance. We will also discuss an overview of the NoSQL database, and the characteristics of Postgres with PostGIS extension and MongoDB as geospatial databases.

2.1 Vector Geospatial Data

In Geographic Information System (GIS), geospatial data is generally classified into two types: vector and raster [4]. Raster is cell-based data that represents a surface. One of the most common raster data is imagery satellite and digital pictures. In this review, we will focus on benchmarking querying time on top of vector data. Vector data has three main types of geometric objects: points, lines, or polygons [1].

Based on [7], points describe the location of individual objects such as restaurant, museum, or airport. It stores the objects in two dimensions in pairs (x,y). Lines depict linear objects which are created by connecting points over particular rules. Typical examples are street, river, or trails. Polygons are formed by a set of connected lines. The start and end point of a polygon should have the same coordinates. Polygons indicate enclosed areas, boundaries, or districts such as parks, lakes, or forests. Polygon features are two dimensional and can be used to measure the area and perimeter of a geographic feature [8].

The example of a spatial query for points is to find the closest gas station to the current location. For lines, it is to determine all cities which a given railway passes through. For polygons, it is to determine all the adjacent regions of a given district [9].

2.2 NoSQL Database

The challenges in storing and processing massive volume and fast velocity data with traditional databases have led to the rise of NoSQL databases [9]. This kind of database is designed to be horizontally scalable, which means that it can handle increased data by simply adding more servers. Thus, by distributing the load, it can deal with simultaneously high read and write requests to large datasets [2]. NoSQL is an excellent alternative for RDBMS in the context of web applications that use large datasets [10]. It has been used a lot to deal with high read and write parallel requests related to large datasets [4]. One of the pioneers of NoSQL databases is Google BigTable.

The key characteristic of this type of database that differs from RDBMS is the way data is organized. Relational databases store data in a table format, whereas NoSQL databases are classified into several categories according to their approach to storing the data [5]. Some groups of NoSQL databases are key-value, wide-column, document, and graph databases [9].

Resonating with its name, NoSQL database does not use SQL language to query the data. Each NoSQL has its unique query language. Apart from that, NoSQL is a schemaless database. It is, therefore, a better choice to store unstructured data like documents or text. Nevertheless, it does not always guarantee ACID properties like RDBMS [11]. In terms of geospatial databases, only a few NoSQL databases support geospatial functions and geospatial indexing [5].

2.3 Review of Geospatial Database

2.3.1 PostgreSQL with PostGIS Extension

PostgreSQL is an object-relational database management system. To store GIS objects, both raster and vector, PostgreSQL requires PostGIS to extend the basic data type and query operator to enable geospatial objects and operations [12]. This PostGIS extension provides more than one thousand geospatial functions which can construct and manage geometric objects, do geospatial measurement, and carry out many more actions [13]. It also supports spatial indexes to reduce disk and memory and speed up spatial querying [14]. Three kinds of its indexes are B-trees (binary trees), R-trees (sub-rectangles trees), and GiST (Generalized Search Trees or R-tree-over-GiST). Based on [12], PostGIS is the most efficient open-source solution for managing geospatial data.

2.3.2 MongoDB

MongoDB is a document-based distributed database written in C++. It uses GeoJSON objects to store spatial geometries. Each GeoJSON document consists of two fields: type and coordinates. The type field specifies GeoJSON reader how to interpret the coordinates field. The value of type can be a point, line string, polygon, multi-point, multi-line string, and multi-polygon. Meanwhile, coordinates field specifies the geometric objects' coordinates, such as a pair of longitude and latitude [15].

Currently MongoDB only provides four geospatial functions, which are $geoWithin, $geoIntersects, $near, and $nearSphere. $geoWithin is a function to select geometry objects that exist entirely within a specified geo object, $geoIntersects is to select geo objects which intersect with a specified geo object, and $near or $nearSphere are to select geo objects in proximity to a point or to a point on a sphere [16]. In contrast, PostGIS offers a more huge variety of geofunctions, including those four functions. MongoDB also has two types of geospatial indexes: 2D and 2Dsphere index. It does not support R-tree index like PostGIS. 2D index is used to calculate geometries on a 2D plane surface, and 2Dsphere index calculates geometries on a spherical shape like earth [16]. In MongoDB operations Create, Read, Update, Delete can be performed faster even on a large dataset [17].

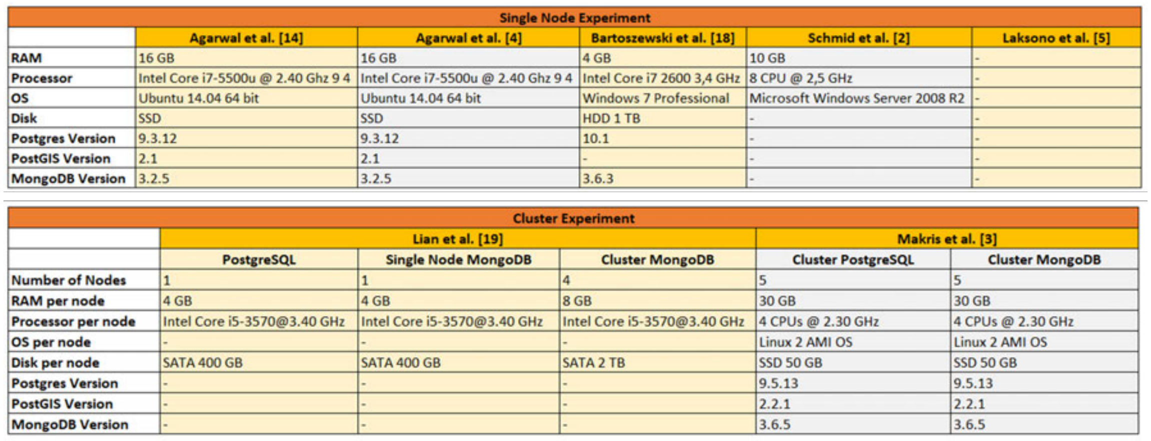
3 Literature Review

In this section, we will review experiments from seven literature to evaluate query performance between PostgreSQL and MongoDB in processing vector spatial data. Five of them ran performance testing on a single node machine, and two on a cluster environment. All of these studies used diverse datasets and queries. Although all the queries were different, some shared the same task, such as intersection or containment task. In the real application, querying or analyzing over geospatial data can be varied and not specific to a particular query type. As such, this variety of experiments indirectly mimics the real-world application. The seven performance tests were conducted on different hardware specifications and software versions as shown in Figure 1. Nevertheless, some of them did not mention the hardware and software specification.

In the following sections, we will first discuss the single node experiments and later the cluster experiments. Within each literature, we will also compare to others when they have common experiment.

3.1 Line Intersection and Point Containment Scenario by Agarwal et al (2016)

Agarwal et al. issued two papers in 2016 [14] and 2017 [4] with different experiments to examine PostgreSQL and MongoDB's performance. In the 2016 experiment [14], they used scenarios that possibly happen in the real world, such as line intersection and point containment [14]. The example situations respectively are finding all small roads attached to a highway and reporting



Single Node Experiment					
	Agarwal et al. [14]	Agarwal et al. [4]	Bartoszewski et al. [18]	Schmid et al. [2]	Laksono et al. [5]
RAM	16 GB	16 GB	4 GB	10 GB	-
Processor	Intel Core i7-5500u @ 2.40 Ghz 9 4	Intel Core i7-5500u @ 2.40 Ghz 9 4	Intel Core i7 2600 3,4 GHz	8 CPU @ 2,5 GHz	-
OS	Ubuntu 14.04 64 bit	Ubuntu 14.04 64 bit	Windows 7 Professional	Microsoft Windows Server 2008 R2	-
Disk	SSD	SSD	HDD 1 TB	-	-
Postgres Version	9.3.12	9.3.12	10.1	-	-
PostGIS Version	2.1	2.1	-	-	-
MongoDB Version	3.2.5	3.2.5	3.6.3	-	-

Cluster Experiment					
	Lian et al. [19]			Makris et al. [3]	
	PostgreSQL	Single Node MongoDB	Cluster MongoDB	Cluster PostgreSQL	Cluster MongoDB
Number of Nodes	1	1	4	5	5
RAM per node	4 GB	4 GB	8 GB	30 GB	30 GB
Processor per node	Intel Core i5-3570@3.40 GHz	Intel Core i5-3570@3.40 GHz	Intel Core i5-3570@3.40 GHz	4 CPUs @ 2.30 GHz	4 CPUs @ 2.30 GHz
OS per node	-	-	-	Linux 2 AMI OS	Linux 2 AMI OS
Disk per node	SATA 400 GB	SATA 400 GB	SATA 2 TB	SSD 50 GB	SSD 50 GB
Postgres Version	-	-	-	9.5.13	9.5.13
PostGIS Version	-	-	-	2.2.1	2.2.1
MongoDB Version	-	-	-	3.6.5	3.6.5

Figure 1: Hardware and software specification of all experiments

how many houses are located in a particular area to analyse the changes. The motivation behind this work was to reduce the dependency for the server during mobile routing.

For line intersection, they broke down the experiment into several scenarios: one line versus one line intersection, one line versus multiple lines intersection, and multiple lines versus multiple lines intersection. The number of lines ranged from 10, 100, 1000, to 10,000. The maximum intersection was around 19 million generated from intersections between 10,000 and 10,000 lines. In total, there were ten queries in the line intersection case.

For point containment, they used a dataset with two independent layers. The first layer was horizontal lines with the size varied from 10 to 80 lines, and the second layer consisted of square boxes of the different perimeter. There were some lines which were entirely inside the box, and the rest are outside the boxes. The maximum number of lines within the box was nearly 300,000 lines. In total, there were 23 queries in the point containment case. For this case, the authors assumed that all the lines were made up of millions of points.

They ran the experiment with index and without index on both databases. In PostGIS, they used the GIST index method, and in MongoDB, they used 2D Sphere. Based on the result, MongoDB performed faster than PostgreSQL in all scenarios, including with and without index. As the data increased, the average query time of PostgreSQL rose more dramatically than MongoDB. They also observed that the performance difference between indexed and non-indexed dataset was relatively small in MongoDB compared to PostgreSQL, which had exponentially reduced time.

Overall, the query on the indexed dataset finished faster than non-indexed dataset on both databases. These results suggested that MongoDB performed better by an average factor of 25 times than PostgreSQL in both indexed and non-indexed data and line intersection and point containment cases. They also discussed the limitation on MongoDB as geospatial databases which it only supports very basic geofunctions. PostgreSQL with PostGIS extension is still a superior geospatial database in terms of the number of features.

3.2 Aggregation of Point Containment Scenario by Agarwal et al (2017)

In the 2017 paper of Agarwal et al. [4], they benchmarked the two databases' performance using different approaches. This time the authors measured the execution time of the aggregation query. They created two point containment scenarios combined with aggregate queries that involved spatial and non-spatial data. The first scenario was to calculate the total number of vegetarian restaurants within a particular area. The second was to count the number of unique cuisines within a specific area.

The working dataset consisted of two layers. The first layer contained the list of restaurants represented as points and have non-spatial attributes such as time, price, and cuisine. The second layer contained squares of boxes with different perimeters that some points were located inside the box and some outside the box. This box represented the area of interest. The number of restaurants varied from 10 to 80, and the boxes ranged from 5 to 90. In total, there were 23 combinations of the first and second layer with different values for each.

Similar to the 2016 experiment [14], the authors tested the two scenarios on both indexed and non-indexed databases. According to the result, MongoDB always outperformed PostgreSQL in all cases, from small to large dataset. The increased amount of dataset has affected PostgreSQL performance negatively. Queries on PostGIS took much longer to finish when the size of the dataset increased.

Overall findings were very similar to the author's previous experiment [14]. The gap latency between indexed and non-indexed dataset was more prominent when the query ran on PostgreSQL than on MongoDB. Therefore from both studies of Agarwal et al. [14, 4], we can conclude that MongoDB performance is better and more stable than PostgreSQL as a geospatial database.

The testing methodology of both Agarwal et al. papers [14, 4] was not explained in detail. Both experiments measured the querying time average, but they did not mention how many times they run the query. It is not easy to measure how representative enough the experiment results were. They also did not provide the query statement, which makes it difficult to compare to other papers.

3.3 Containment, Distance Measure, and Intersection by Bartoszewski et al

The approach of Bartoszewski et al. [18] attempted to answer whether NoSQL is a suitable replacement of RDBMS to process massive GPS logs that are saved in PointGeometry format. The authors composed four test scenarios to compare PostgreSQL and MongoDB performance on non-indexed databases. Bartoszewski et al. [18] explained more detailed experiments than the two aforementioned papers. The authors clearly stated the method to measure query speed, the number iterations of query execution, and the query statement for each scenario and database. They used the pgbench application and explain() method to measure PostgreSQL and MongoDB latency, executed each query for ten times and then calculated the average and standard deviation of execution time.

The first scenario was a containment case which checked whether the points were within a particular polygon area. They used seven different numbers of points which ranged from 1,000 to 1,000,000 points. MongoDB clearly excelled in this case in which it delivered approximately three times faster performance than PostgreSQL. This scenario was similar to point containment testing of Agarwal et al. (2016) [14]. Both results agreed that MongoDB performed faster than PostgreSQL.

The second scenario was to discover points that were located within a certain distance from the given coordinates. To measure the distance, PostgreSQL query uses ST_DWithin function and MongoDB uses $nearSphere and $maxDistance. The points collections included 50,000 until 1,000,000 points, and the distance was 100, 200, 500, and 1000 km from a centroid. The product of all point collections and radius yielded 16 cases on the second test. The authors observed that in cases of radius 100 to 500 km, query execution in MongoDB consistently took faster than in PostgreSQL. However, when scrutinizing the query time's growth, execution time in MongoDB grew much faster with a larger radius to calculate compared to PostgreSQL. This condition made PostgreSQL finally outperformed MongoDB in case of radius 1,000 km for all point collections.

The third scenario was similar to the second one, but this time the authors aimed to find polygons instead of points. The polygons collections varied from 5,000 to 100,000 polygons and the radius setting was the same with the second scenario. The result demonstrated a similar pattern with the second test. The larger the radius, the much slower MongoDB's performance compared to PostgreSQL. This condition indicated that the high radius number had a more negative impact on MongoDB compared to PostgreSQL. These findings were somewhat opposed to the NoSQL databases theory, which excelled at processing large amounts of data.

The fourth scenario used a more complex query related to the intersection case. It selected the closest polygons to a centroid, then checked whether those polygons intersect with other polygons. This compound query also applied the distance function to retrieve points located within 2,5 km from a centroid. As the radius was not as large as the second and third test, there was no performance issue in MongoDB. It performed around six times faster than PostgreSQL for all points numbers which ranged from 5,000 to 500,000.

All four performance tests of Bartoszewski et al. indicated that queries involving within and intersection operations took faster in MongoDB. In contrast, PostgreSQL showed more advantages when distance measuring function applied for a larger radius. The reason behind why larger radius affects MongoDB more negatively than PostgreSQL is not well-explained in the paper. Bartoszewski et al. might need to run the same experiments on the indexed dataset to find whether indexing could help to solve this issue

3.4 Filtering Non-Spatial Attributes and Containment Scenario by Schmid et al

Performance tests done by Schmid et al. [2] covered more exhaustive scenarios than the other experiments. They evaluated the execution time of two types of geospatial queries run simultaneously by a certain number of users on different datasets. The first query was to retrieve all geometry objects (points, lines, or polygons) based on particular non-spatial attributes. The second query was to retrieve all geometry objects within a specific polygon (containment case). The authors ran the queries on indexed databases using GIST and 2D Sphere indexes for PostgreSQL and MongoDB. On this experiment, they used three dataset with different size obtained from OpenStreetMap: Niederbayern dataset (subregion level with size of 38,9 MB), Bayern (state level with size of 501 MB), and Germany (country level with size of 2,1 GB).

Schmid et al. designed a script to run the query multiple times in parallel as if there are multiple users running the query. The objective was to simulate a condition in the real world. They used three different numbers of users: 100, 250, and 500 users. In reality, most users do not run queries exactly at the same time, but usually have a small time difference. To mimic this situation, the authors set three types of time offsets: no time offset (simultaneously), 0.5 second, and 1 second. The variety of non-spatial attributes and geometry objects in the query, dataset, number of users, and time offset had generated 486 test cases. Schmid et al. declared that the testing did not investigate the memory usage or storage overhead. They used JMeter to measure the execution time.

Paper by Schmid et al. did not provide detailed results of each test case. Instead, they summarized the analysis of the work based on the query. On the first query, PostgreSQL always needed a longer response time than MongoDB regardless of the geometry objects, number of users, time offsets, and size of the dataset.

This behaviour differed from the second query, which involved *within* operation. For more complex geometry objects such as lines and multipolygons, PostgreSQL execution time was faster than MongoDB on Niederbayern (small) and Bayern (medium) dataset. Nevertheless, it rapidly increased when dealing with Germany (large) dataset. On the other hand, the size of the dataset did not affect query time in MongoDB as much as in PostgreSQL. This type of NoSQL thus had a more stable performance no matter the number of users and time offsets. Schmid et al. argued that MongoDB performed better in general because of its 2D index that calculates geohash values. They speculated that searching R-Tree that is used in GIST index might be slower than searching geohash values. However, they did not take further research to prove this hypothesis.

3.5 Web GIS Loading Time Scenario by Laksono et al

The methodology of Laksono et al. [5] to quantitatively compare the two databases are different from other discussed papers. Their approach was dictated by the need of web GIS applications that usually display large amounts of geospatial data. They thus built a NodeJS web application to visualize geospatial data and quantified the loading time of accessing the data through this web to investigate PostgreSQL and MongoDB's performance. The loading time was measured starting from the request being fired, to when the response was delivered to the client's browser, i.e. the XMLHttpRequest (XHR) response time. They used Firefox's Developer Tools to obtain detailed responses and saved it into the HAR file to analyze the XHR time. They did not state hardware and software specifications for their experiment.

Laksono et al. paper [5] did not execute any geospatial-related query like other papers. However, the paper still contributed to evaluating PostgreSQL and MongoDB as a geospatial database using different perspectives and approaches. The experiment randomly generated five different numbers of POI dataset, starting from 50 to 500,000 POIs using QGIS Toolbox's random point generator. They indexed PostgreSQL and MongoDB using B-tree and 2D Sphere respectively. The resulting test showed that MongoDB always had a shorter response time than PostgreSQL. The time difference increased more significantly as bigger POIs. At POIs of 500,000, the web failed to load the data from PostgreSQL. The reason behind this issue was not explained explicitly. We speculated that query time in PostgreSQL took longer than the duration of request timeout.

3.6 Overall Single Node Experiment Review

According to the five reviewed papers [14, 4, 18, 2, 5], MongoDB beats PostgreSQL in most scenarios regardless the data size, especially queries related to intersection tasks (i.e. using *intersect* geofunction) and containment tasks (i.e. using *within* geofunction). In the study of Laksono et al. [5], which does not involve geospatial query, the result also demonstrates that query time in MongoDB is faster than PostgreSQL. However, there is a containment case in which PostgreSQL is more superior than MongoDB. In Schmid et al. [2], PostgreSQL showed advantages at speed on small and medium dataset, but it significantly slowed down when processing large dataset. This experiment actually involved other criteria such as the number of multi-user query and time offsets, and this condition might make the result different from other containment cases. In the Bartoszewski et al. scenarios of discovering objects within a certain distance, PostgreSQL is better than MongoDB when dealing with a high radius of 1000 km [18].

The motivation behind all the five papers is based on the increasing amount of geospatial data. However, none of them did experiment using the amount of data that is representative enough to the current condition or on a cluster environment. The papers also only benchmark query time without evaluating disk or memory usage. To get more answers and perspectives, we review another two papers that carried out the performance testing on a cluster mode, including evaluating the disk or memory usage. The two papers we will discuss in the following section employed big enough dataset in the experiment. Nevertheless, instead of geospatial data, they use spatio-temporal data, it is geospatial data with temporal (timestamp) attribute.

3.7 Performance Testing on Sharded Cluster Mode by Lian et al

Study of Lian et al. [19] examined the disk usage, memory usage, and query time of single-node PostgreSQL, single-node MongoDB, and multi-node MongoDB with three shard nodes in processing spatio-temporal data. The dataset covers more than 30 years of climate data which consisted of latitude, longitude, six other features related to climate, and timestamp that spread from 1948 to 2012. Although the total data size was not mentioned, the 30 years of climate data was big enough for performance testing. They indexed the single-node and multi-node MongoDB and PostgreSQL based on timestamp attribute. The query selected all data filtered by various years, ranging from 1 to 30 years, to gradually increase the workload. Yet, the authors did not write the query statement.

In terms of disk usage, the storage size of PostgreSQL rose in linear as the data size increased, whereas single-node and multi-node MongoDB raised in steps, and they consumed a much bigger space than PostgreSQL. The authors explained that MongoDB occupies pre-allocated disk space to reduce disk fragmentation and ensure that documents are stored in a contiguous block on disk. When documents are inserted, the pre-allocated disk fills up. When there is not enough space for a new document, MongoDB will double the size of the pre-allocated disk [19]. The authors also stated that MongoDB does not use data compression and PostgreSQL applied LZ compression. Enabling compression in MongoDB needs to apply WiredTiger storage engine which provides snappy and Zlib compression [20]. We think that this disk usage comparison is not head-to-head and less valid as the authors benchmarked the disk usage between non-compressed MongoDB versus compressed PostgreSQL.

Regarding memory usage, the single and cluster mode MongoDB took up slightly more memory than PostgreSQL. All three databases occupied similar memory usage. According to Lian et al. [19], MongoDB adopts a memory-mapped file mechanism which associates data files with memory. When a query is executed, it will directly check into memory. This condition can trigger a page fault when some files are not mapped into memory and can deteriorate performance and cause query time less efficient as the consequence of using more memory. Another MongoDB characteristic is that it takes as much memory as it can to execute a query. However, as it has dynamic memory usage, it will produce cached memory when other processes need memory. This case makes MongoDB a little less efficient in terms of memory usage [19]. Memory usage between single-node and multi-node were almost similar throughout the 1 to 30 year's filter.

For the query time, single and multi nodes MongoDB performed significantly faster than PostgreSQL. When querying 1 to 20 years of data, single-node and multi-node MongoDB yielded almost similar query time, but after 20 years, both started to slow down due to memory swapping. Overall multi-node outperformed single-node.

3.8 Performance Testing on Non-Sharded Cluster Mode by Makris et al

Makris et al. [3] made performance comparison on single-node and cluster mode and tested the query time on both databases with and without index. They used 11 GB of spatio-temporal dataset, which was provided by a community-based AIS vessel tracking system (VTS) of Marine-Traffic. It contains eight attributes and 146 million records from 43.288 unique vessels. The cluster of MongoDB consisted of a master node (primary) and four replication slaves (secondaries) and PostgreSQL cluster also contained a master and four slaves with Streaming Replication. They deployed the databases on EC2 instances on AWS.

Cluster architecture MongoDB designed by Makris et al. is different from multi-node MongoDB on Lian et al. [19] paper. Makris et al. used replica sets, but not a sharded cluster, and configured 'read preference' setting to 'NEAREST' which means read from any available nodes (primary or secondary) with the least network latency. Meanwhile, Lian et al. [19] used a sharded cluster of three nodes but did not mention anything about the replica set and the 'read preference' setting.

They created three spatio-temporal queries that involved geospatial and temporal attributes and mimicked real-world scenarios. Each query was applied on various types indexed PostgreSQL and MongoDB and was executed for five times. They ran the first query on both databases indexed with a regular B-tree on a non spatio-temporal attribute. The second query was run on databases with B-tree index of a temporal attribute. The third query indexed PostgreSQL using GIST and MongoDB using 2D Sphere on a spatial attribute.

In cluster mode comparison, the results showed that PostgreSQL outperformed MongoDB in all queries and its response time was about four times faster for the second and third query. The same behaviour happened in a single-node experiment. This result surprisingly contradicted all the papers mentioned above, including spatio-temporal experiment by Lian et al. The different design database architecture of Makris et al. and Lian et al.'s study mainly contributed to this opposite conclusion.

Regarding index evaluation, the query time after indexing was reduced significantly on both databases. Applying index in MongoDB can limit the number of documents that must be inspected and avoid scan operations in every document in a collection [3]. Index implementation accounted for a much more significant reduction in PostgreSQL than in MongoDB, especially in the third query in which it can lower almost at half query time. Agarwal et al. [14, 4] also concluded the same thing, PostgreSQL had more significantly reduced query time with indexing. From this outcome, we can infer that the GIST index in PostGIS is more efficient than the 2D Sphere index in MongoDB. Some research has been done to find faster indexes in MongoDB, such as R-Tree [21, 22].

4 Summary and Conclusion

This literature reviewed varying papers that benchmarked PostGIS and MongoDB's performance in querying geospatial data in vector format. In many ranges of scenarios, MongoDB is proven to be superior in terms of query time and has a more stable increase latency across varied data size. Nevertheless, there are some cases where Postgres has advantages over MongoDB.

Studies of Agarwal et al. (2016) [14] and Bartoszewski et al. [18] have attested that query time MongoDB is always faster than PostGIS in intersection scenarios with various data size. The result of containment cases also showed similar behaviour, except the experiment done by Schmid et al. [2]. It demonstrated that PostgreSQL excelled at speed when querying small and medium dataset, but its execution time became a magnitude slower in processing large dataset. However, aside from the dataset's size, some other elements affected the workload that was not used in other experiments, such as the number of parallel query and time offsets. This consideration can be the reason why the experiment of Schmid et al. had a different outcome. On the other hand, PostgreSQL has clear advantages over MongoDB in the scenario of discovering geometric objects (points and polygons) within a long distance of more than 1000 km.

The works of Agarwal et al. [14, 4], Bartoszewski et al. [18], Schmid et al. [2], and Laksono et al. [5] conducted on a single machine and employed not a big enough dataset. Meanwhile, Makris et al. [3] and Lian et al. [19] examined the latency on both a single and cluster machine using a much bigger dataset. However, they used spatio-temporal data instead of spatial data only. The only difference between these two types of data is the presence of temporal (timestamp) attribute. Makris et al. concluded that PostgreSQL outperformed MongoDB on both single-node and cluster mode. In contrast, Lian et al. deduced that MongoDB performs significantly faster than PostgreSQL. Nevertheless, these two outcomes can be seen as an unfair comparison due to their different design of architecture. Makris et al. implemented a non-sharded MongoDB cluster, whereas Lian et al. implemented a sharded cluster.

Another thing we can conclude is that indexing databases can help boost performance according to [4, 14, 3]. Furthermore, those papers agreed that the improvement is more significant for PostgreSQL compared to MongoDB. These results can imply that GIST index on PostgreSQL is more efficient than 2D Sphere in MongoDB. Aside from query time evaluation, we only found one paper which compared disk and memory usage between both databases. Thus, it is not enough to draw any conclusion in the context of the disk and memory usage solely based on one report. Hence, in this literature review, we can only evaluate the performance based on query time.

In conclusion, we learnt that there is no single geospatial database that can outperform the other in every kind of task. As the results of the evaluations, MongoDB offers better query speed in most containment and intersection tasks. It also has advantages as a distributed database and with its easiness of horizontal scaling. Nevertheless, MongoDB only supports four geo functions, outnumbered by PostGIS, which has thousands of geo features. This limitation has been mentioned since the 2015 paper of Schmid et al., and unfortunately there is no new function added based on the current official MongoDB website [16]. Considering its advantages and disadvantages, MongoDB is a better option when prioritizing latency but not requiring advanced spatial functions. Meanwhile, PostgreSQL with PostGIS extension is a good-to-go when looking for more complex functions, having a heavy task of discovering objects within a certain distance more than 1000 km, and preferring convenience and easiness of table structure format.

As a final note, comparing two databases is not a straightforward task as there are some techniques to tune the performance. For instance, set the 'read preference' parameter of MongoDB to NEAREST to select nodes with the least network latency. To benchmark two databases head-to-head, it is imperative to apply the same parameter settings or carry out the same performance tuning on both databases. Otherwise, it can yield incomparable evaluation, like what happened in the paper of Lian et al. [19], evaluating the disk usage of non-compressed MongoDB and compressed PostgreSQL. It is also crucial for the researchers to understand the best practice in designing both databases to result in fair benchmarking.

References

[1] Dongming Guo, Erling Onstein. State-of-the-art geospatial information processing in nosql databases. *ISPRS Int. J. Geo-Inf*, 2020.

[2] Stephan Schmid, Eszter Galicz, Wolfgang Reinhardt. Performance investigation of selected sql and nosql databases. 2015.

[3] Antonios Makris, Konstantinos Tserpes, Giannis Spiliopoulos, Dimosthenis Anagnostopoulos. Performance evaluation of mongodb and postgresql for spatio-temporal data. 2019.

[4] Sarthak Agarwal, KS Rajan. Analyzing the performance of nosql vs. sql databases for spatial and aggregate queries. *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*, 17(4), 2017.

[5] Dany Laksono. Testing spatial data deliverance in sql and nosql database using nodejs fullstack web app. *2018 4th International Conference on Science and Technology (ICST), Yogyakarta*, pages 1–5, 2018.

[6] Parinaz Ameri, Uda Grabowski, Jorg Meyer, Achim Streit. On the application and performance of mongodb for climate satellite data. *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 652–659, 2014.

[7] Deborah Yates, Jared Keller, Rachel Wilson, Leigh Dodds. The uk's geospatial data infrastructure: challenges and opportunities. `https://theodi.org/wp-content/uploads/2018/11/2018-11-ODI-Geospatial-data-infrastructure-paper.pdf`. Open Data Institute. Online; accessed 21 December, 2020.

[8] Caitlin Dempsey. Types of gis data explored: Vector and raster. `https://www.gislounge.com/geodatabases-explored-vector-and-raster-data/`. GIS Lounge. Online; accessed 21 December, 2020.

[9] Peng Yue, Zhenyu Tan. Gis databases and nosql databases. 2018.

[10] Katarzyna Harezlak, Robert Skowron. Performance aspects of migrating a web application from a relational to a nosql database. 2015.

[11] Ali Davoudian, Liu Chen, Mengchi Liu. A survey on nosql stores. *ACMComput. Surv*, 51(2), 2018.

[12] Elena Baralis, Andrea Dalla Valle, Paolo Garza, Claudio Rossi, Francesco Scullino. Sql versus nosql databases for geospatial applications. *2017 IEEE International Conference on Big Data (Big Data)*, pages 3388–3397, 2017.

[13] Marcin Pietron. Analysis of performance of selected geospatial analyses implemented on the basis of relational and nosql databases. *Polish Cartographical Review*, 51(4):167–179, 2019.

[14] Sarthak Agarwal, KS Rajan. Performance analysis of mongodb versus postgis/postgresql databases for line intersection and point containment spatial queries. *Spat. Inf. Res*, 24(6):671–677, 2016.

[15] MongoDB Official Site. Geojson objects. `https://docs.mongodb.com/manual/reference/geojson`. Online; accessed 27 December, 2020.

[16] MongoDB Official Site. Geospatial queries. `https://docs.mongodb.com/manual/geospatial-queries`. Online; accessed 27 December, 2020.

[17] David Hows, Peter Membrey, Eelco Plugge. *MongoDB Basics.* Apress, California, 2014.

[18] Dominik Bartoszewski, Adam Piorkowski, Michal Lupa. The comparison of processing efficiency of spatial data for postgresql and mongodb databases. *BDAS 2019, CCIS 1018*, page 291–302, 2019.

[19] Jie Lian, Sheng Miao, Michael McGuire, Ziying Tang. Sql or nosql? which is the best choice for storing big spatio-temporal climate data? *ER 2018 Workshops, LNCS 11158*, pages 275–284, 2018.

[20] ScaleGrid Blog. Enabling data compression in mongodb 3.0. `https://scalegrid.io/blog/enabling-data-compression-in-mongodb-3-0`. Online; accessed 8 January, 2021.

[21] Yan Li, Dongho Kim, Byeong-Seok Shin. Geohashed spatial index method for a location-aware wban data monitoring system based on nosql. *Journal of Information Processing Systems*, 12(2):263–274, 2016.

[22] Longgang Xiang, Juntao Huang, Xiaotian Shao, Dehao Wang. A mongodb-based management of planar spatial data with a flattened r-tree. *ISPRS International Journal of Geo-Information*, 5:119, 2016.