Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings

Volume 17 *Boston, USA* Article 4

2017

Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries

Sarthak Agarwal
International Institute of Information Technology Hyderabad Gachibowli, Hyderabad, India

KS Rajan
International Institute of Information Technology Hyderabad Gachibowli, Hyderabad, India

Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries

Sarthak Agarwal[a,*], KS Rajan[a]

[a] *International Institute of Information Technology Hyderabad Gachibowli, Hyderabad, India*

Abstract: Relational databases have been around for a long time and spatial databases have exploited this feature for close to two decades. The recent past has seen the development of NoSQL non-relational databases, which are now being adopted for spatial object storage and handling, too. While SQL databases face scalability and agility challenges and fail to take the advantage of the cheap memory and processing power available these days, NoSQL databases can handle the rise in the data storage and frequency at which it is accessed and processed - which are essential features needed in geospatial scenarios, which do not deal with a fixed schema(geometry) and fixed data size. This paper attempts to evaluate the performance of an existing NoSQL database 'MongoDB' with its inbuilt spatial functions with that of a SQL database with spatial extension 'PostGIS' for two problems spatial and aggregate queries, across a range of datasets, with varying features counts. All the data in the analysis was processed In-memory and no secondary memory was used. Initial results suggest that MongoDB performs better by an average factor of 10x-25x which increases exponentially as the data size increases in both indexed and non-indexed operations. Given these results, NoSQL databases may be better suited for simultaneous multiple-user query systems including Web-GIS and mobile-GIS. Further studies are required to understand the full potential of NoSQL databases across various geometries and spatial query types.

*Corresponding author
Email address: sarthak.a@research.iiit.ac.in (Sarthak Agarwal)

1. Introduction

Traditionally Databases were designed to structure and organize any form of data. However, as the database size increased and to optimize databases for the geospatial domain we use spatial databases. Satellite images are one prominent example of spatial databases Govind and Sharma 2013. To extract spatial information from a satellite image, it has to be processed in a spatial frame of reference. However, satellites are not the only type of spatial databases. Maps are also stored in the spatial database.

Like other database systems, spatial databases have also relied on the relational databases to handle and manage spatial objects and their associated attribute information. They have been of great value in cases where we have a defined structure of our schema, including geometrical characteristics of the spatial objects. In addition, in many spatial applications, we do not always have a fixed schema, there can be many geometries with different shape and the requirements evolve as the data size increases depending on the case scenario where relational Databases can limit the potential use or design of the solution.

For the satisfaction of the user's significant characteristics of a database such as scalability, performance and latency play a crucial role. Especially social media projects, such as Facebook and Google+, with high user traffic, use other database management systems. such as Apache Cassandra or Google BigTable. Instead of the relational approach, a Not-only-SQL (NoSQL) method is used. NoSQL-databases are increasingly used to deal with simultaneously high read and write requests related to large datasets.

While SQL databases face scalability and agility challenges and fail to take the advantage of the cheap memory and processing power available these days, NoSQL databases can handle the rise in the data storage and frequency at which it is accessed and processed - which are essential features needed in geospatial scenarios, which do not deal with a fixed schema(geometry) and fixed data size Loureno et al. 2015.

NoSQL data stores may provide advantages over relational databases, However, they generally lack the robustness of relational databases for those advantages. The aim of this paper is to discover some benefits of a selected NoSQL data store as compared to a traditional relational database when storing and querying spatial vector data. Our work is influenced by the work done previously on the similar theories de Souza Baptista et al. 2011, Xiao and Liu 2011, Schmid et al. 2015 and some practical implementation of those theories Popescu and Bacalu 2009, Steiniger and AndrewJ.S.Hunter 2012, van der Veen et al. 2012.

One of the very important problems we face is of aggregate queries i.e when we combine the normal non-spatial data with the spatial data. Here we are trying to solve a similar problem based on our previous studies. The aim of this problem is to combine the real time spatial and non-spatial scenarios into one and then analyze the performance of SQL vs NoSQL databases in such scenarios.

The two problems are carefully chosen which are, first the total number of restaurants in an area which is vegetarian only. This problem has two very important features. Firstly we have to use containment query to find out the total number of restaurants in an area and apply sum aggregate query on such restaurants which are vegetarian. The second problem is getting the total number of distinct cuisine in an area. This problem also has very important aggregate feature i.e. distinct in a column. So using these queries we are able to simulate most of the real case scenarios and analyzing the performance of each database.

2. NoSQL vs SQL in spatial context

2.1. NoSQL vs SQL

SQL Server and relational databases have been the go-to databases for over 20 years. The increased need to process higher volumes, velocities, and varieties of data at a rapid rate has altered the nature of data storage needs for application developers. SQL Server and relational databases were the most popular databases. However, in order to enable the present day processing needs, NoSQL databases have gained popularity due to their ability to store unstructured and heterogeneous data at scale. Relational databases still remain a popular default option due to their easy to understand table structure, however there are many reasons to explore beyond relational databases.

NoSQL is a category of databases distinctly different from SQL databases. NoSQL is often used to refer to data management systems that are Not SQL or an approach to data management that includes Not only SQL". There are a number of technologies in the NoSQL category, including document databases, key value stores, column family stores, and graph databases, which are popular with gaming, social, and IoT apps.

2.2. NoSQL in spatial context

If we talk in the context of spatial Informatics, we need databases at multiple instances. We need databases to store maps, in maps particularly the schema can be vague. We also need the databases to store images from satellites and another kind of spatial data transmitted by a satellite such as weather info. So in short spatial informatics needs to store a lot of data. So to solve this problem we need some kind of data store.

2.3. Current systems, A case study

There are currently many NoSQL databases systems available but only few support spatial databases. In this section, we are going to discuss some of those systems and what all function so each of them offers.

MongoDB uses currently two geospatial indexes, 2d, and 2dsphere. The 2d index is used to calculate distances on a plane surface. The 2dsphere index calculates geometries over an Earth-like sphere. The coordinate reference system is currently limited to the WGS84 datum. MongoDB computes the geohash values for the coordinate pairs and then indexes the geohash values. A precise description of the indexing techniques of the geohash values is not available at the moment Anonymous 2011-2015.

CouchBase supports indexing of two-dimensional data using an R-Tree index. CouchBase, therefore, provides spatial views which enable a geospatial query using bounding boxes Ostrovsky and Rodenski 2014. A precise description of the indexing techniques in CouchBase is currently also not available.

Besides the storage and indexing of spatial data, the query process is an important aspect. For querying spatial data several geo-functions are available in relational databases. They enable different queries with geo-context at the database level using SQL. An example for a geo-function is the calculation of a buffer around a point feature or a line feature.

For the relational-database PostgreSQL, there is a special extension available, PostGIS, for integrating several geo- functions. MongoDB and CouchBase don't have a separate extension at the moment but they support some geo-functions. Table 1 compares the geo-functions of the three databases.

PostgreSQL/PostGIS inherits more than one thousand geo- functions. Table 1 includes only a selection of them. MongoDB only supports three geo-functions, $geoWithin, $geoIntersects and $near. The MongoDB $geoWithin operator corresponds to the ST_Within function in PostgreSQL/PostGIS, and the MongoDB $geoIntersects operator corresponds to the function ST_Intersects in PostgreSQL/PostGIS.

The function $near delivers the next located geometry for a predefined point. The $near function can be used in combination with a $maxDistance parameter. In that case, MongoDB delivers all geometries within a certain distance ordered by the distance. PostgreSQL can calculate this using the ST_DWithin function. The results however, need to be additionally ordered by the distance.

CouchBase can only query point geometries within a BoundingBox (BBox). The BBox-function of CouchBase can be compared to the $geoWithin (MongoDB) and ST_Within (PostGIS) functions, however, MongoDB and PostGIS can use different polygons, not only an axial parallel polygon.

The overview of the implemented geo-functions shows that PostgreSQL with its extension PostGIS has the most comprehensive geo-functionalities with more than one thousand functions. For a complete list of all implemented functions, it is referred to the PostGIS handbook. The two NoSQL databases have very limited implemented geo-functions. MongoDB just implements three functions whereas CouchBase just implements one geo-function.

2.4. Scope of improvements

The NoSQL spatial database is still at their inception. There is a lot of research and development yet to be done in both theoretical part and practical part of them. Currently, they support only a limited type of geometry and very fewer functions to manipulate them. We have to build the design of these databases in such a way that adding more functions can be flexible. We can take inspiration from the functions currently present in PostGIS and can start implementing those functions in MongoDB as well. spatial databases also currently support for various indexes, Xiang et al. 2016 illustrates the implementation of R-trees in MongoDB for spatial indexing.

PostGIS	**MongoDB**	**CouchBase**
ST_Within	$geoWithin	BBOX
ST_Intersects	$geoIntersects	
ST_DWithin + Order by dist	$near + param(Distance)	
ST_Area		
....		

Table 1: Geo-Functions of the investigated databases.

3. Dataset

For this analysis, the dataset was custom generated.

3.1. All vegetarian restaurants in an area

Point Containment problem works for any geometry and reports whether the given geometry is completely inside another geometry or not. This is a vital and traditional problem on spatial databases. It is useful in the domain of map generations, modeling, analyzing spatial data over an area, for example, we want to report how the number of houses has changed over time in a city by analyzing the spatial data of that town. We can count the number of points (which represents each house in this case) in the polygon(city) for all the years using point within a polygon problem.

The dataset D1 consisted of two layers, the first layer simulating the restaurants which are represented as points having some non-spatial attributes such as timing, price, cuisine, etc. Another layer consisted of square boxes of different perimeter scattered both sequentially and randomly over the space with few points completely inside the box and others completely outside the box simulating the area in interest.

3.2. Distinct Cuisine in an area

Like the previous problem here also we use Point Containment query which works for any geometry and reports whether the given geometry is completely inside another geometry or not.

The dataset D2 consists of similar layers one simulating restaurants and other simulating the area in interest.

4. Performance

Tests were run on both of the database systems with same datasets one with Index and other without an index. The time recorded is in seconds for both indexed and non-indexed analysis. Lines and polygons in PostGIS were indexed using GIST index method and in MongoDB they were indexed using 2d Sphere indexing method.

Test Setup

All the data in the analysis was processed In-memory and no secondary memory was used.

Hardware used-

- Ram 16 Gb
- Processor - Intel Core i7-5500u CPU @ 2.40 Ghz x 4
- OS Ubuntu 14.04 64 bit
- Disk Solid state hard drive

Softwares used-

- PostgreSQL 9.3.12
- PostGIS 2.1
- MongoDB 3.2.5

For Post results analysis Libre Office and Google sheets were used to plot graphs.

If we observe the graph from figure 1 we will notice that MongoDB works much faster than PostGIS in all the cases(from small datasets too large datasets). The difference in performance is in the order of 10(approx.). Also, another observation we make from the graph is that the performance difference between indexed and non-indexed queries is small in MongoDB as compared to PostGIS. So we reach a conclusion that the performance of both indexed and non-indexed Line Intersection queries is better in the case of MongoDB as compared to PostGIS.

In experiment 2, firstly the performance was analyzed without indexing any geometry and time was observed which suggests that MongoDB performs better as the data size increases whereas PostGIS does not perform as well with huge datasets and time complexity increases exponentially. However after indexing the geometries performance of both the database engine improved by a substantial factor.

If we observe the results of the experiments, we will notice that indexed datasets perform better than non-indexed dataset and PostGIS time increases the exponentially as the size of dataset increases whereas MongoDB still performs within some bounds.

5. Discussions and Conclusions

The experiment analyses the performance difference and ease of implementation while developing real time application involving both spatial and non-spatial use cases. These results suggest that MongoDB performs better by an average factor of 10 which increases exponentially as the data size increases in both indexed and non-indexed operations for problems. Given these results, NoSQL databases may be better stated for simultaneous multiple-user query systems including Web-GIS and mobile-GIS.

This implies that non-relational databases are more suited to the multi-user query systems and has the potential to be implemented in servers with limited computational power. Further studies are required to identify its appropriateness and incorporate a range of spatial algorithms within non-relational databases.

The focus of our research is to mainly benchmark the real case scenarios including both spatial and non-spatial use cases. The motivation behind this work is to minimize the dependency for server during mobile routing. We want the client itself to act as a server during routing and for that we need a light database system that can be easily ported to the mobile devices, so here we are comparing the performance of SQL vs. NoSQL databases which are comparatively lighter.

However there are still some limitations on using NoSQL databases over SQL databases. There are not as many spatial functions in NoSQL as in SQL. The currently implemented geo-functions support only very basic operations. Relational databases are still far superior if the user needs to calculate geoinformation on database level Schmid et al. 2015. The results presented in the paper are only valid for the chosen database settings but they clearly show

that NoSQL databases are a possible alternative, at least for querying attribute information. PostgreSQL also has an implementation of NoSQL of its own, however it does not support PostGIS currently however we can export the results of PostGIS queries as GeoJSON objects. In future, we are planning on expanding our study to other spatial query functions as well as spatial algorithms such as shortest path problem to evaluate the performance of NoSQL on such platforms and also test the performance of MongoDB on distributed systems.

References

Anonymous, 2011-2015. Mongodb, inc.; geospatial indexes and queries. Online, online; accessed July 21, 2017.
URL http://docs.mongodb.org/manual/applications/geospatialindexes/

de Souza Baptista, C., de Oliveira, M. G., da Silva, T. E., 2011. Using ogc services to interoperate spatial data stored in sql and nosql databases. XII GEOINFO, Campos do Jordo, Brazil.

Govind, S., Sharma, A., 2013. Open source spatial database for mobile devices. Computer Engineering and Intelligent Systems 4(6).

Loureno, J. R., Cabral, B., Carreiro, P., Vieira, M., Bernardino, J., 2015. Choosing the right nosql database for the job: a quality attribute evaluation. Journal of Big Data 2:18.

Ostrovsky, D., Rodenski, Y., 2014. Pro couchbase server. Apress.

Popescu, A., Bacalu, A.-M., 2009. Geo nosql: Couchdb, mongodb, and tokyo cabinet. Online, online; accessed July 21, 2017.
URL http://nosql.mypopescu.com/post/300199706/geo-nosql-couchdb-mongodb-tokyo-cabinet

Schmid, S., Galicz, E., Reinhardt, W., 2015. Performance investigation of selected sql and nosql databases. AGILE 2015, Lisbon.

Steiniger, S., AndrewJ.S.Hunter, 2012. Free and open source gis software for building a spatial data infrastructure. Geospatial Free and Open Source Software in the 21st Century(Part 5).

van der Veen, J. S., van der Waaij, B., Meijer, R. J., 2012. Sensor data storage performance: Sql or nosql, physical or virtual. IEEE Fifth International Conference on Cloud Computing, Honolulu, Hawaii, USA.

Xiang, L., Shao, X., Wang, D., 2016. Providing r-tree support for mongodb. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic XLI-B4.

Xiao, Z., Liu, Y., 2011. Remote sensing image database based on nosql database. Geoinformatics.

Experiment 1. Performance of All vegetarian restaurants in an area

Restaurants in 1st Layer	Boxes in 2nd Layer	No. of vegan restaurants within	PostGIS time(s)	PostGIS Index time(s)	MongoDB time(s)	MongoDB Index time(s)
10	5	5	9.25	8.39	6	7
10	10	17	17.63	14.82	12	12
10	20	17	17.14	14.76	27	26
10	40	17	15.01	17.55	52	55
20	5	5	23.18	21.86	7	9
20	10	17	51.46	47.88	13	14
20	20	45	108.7	133.71	27	28
20	40	46	147.07	151.24	50	48
40	5	5	72.21	78.02	8	6
40	10	5	146.59	183.91	12	13
40	20	45	727.36	630.53	28	27
40	40	122	1092.52	1279.14	51	52
40	60	122	1272.77	1065.34	96	97
60	5	5	120.92	121.71	6	6
60	10	17	292.7	313.96	13	11
60	20	45	1130.5	1102.48	28	30
60	40	122	3111.69	2920.86	51	52
60	60	204	3753.86	3427.31	98	102
60	80	204	3069.14	3213.61	99	102
80	40	122	4573.4	3269.07	53	53
80	60	204	5325.57	2471.84	102	100
80	80	298	NC*	NC*	106	109
80	90	298	NC*	NC*	144	123

Table 2: Average Time for Vegan in an area problem (*NC- Not Computable).

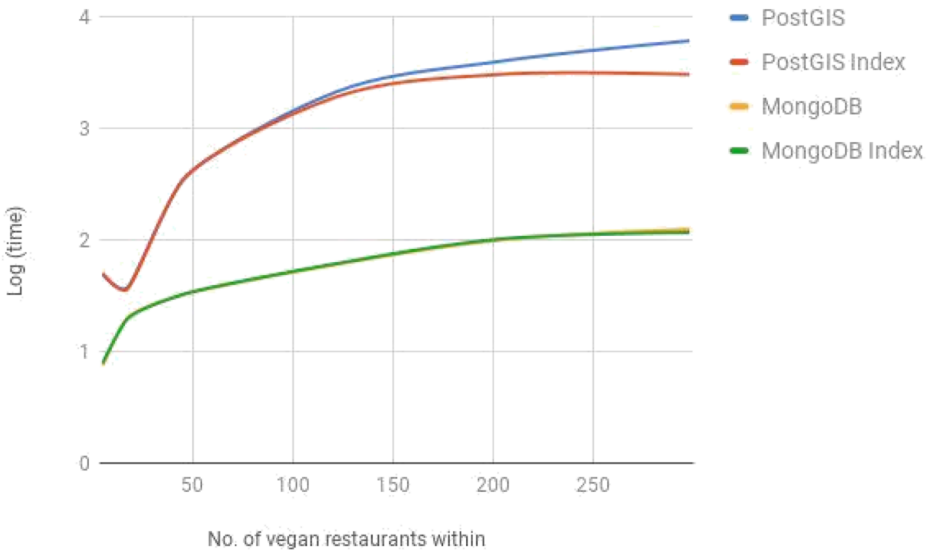

Figure 1: Graph of No. of restaurants vs average time for all datasets in D1.

Experiment 2. Performance of Distinct Cuisine restaurants in an area

Restaurants in 1st Layer	Boxes in 2nd Layer	No. of Distinct Cuisine restaurants within	PostGIS time(s)	PostGIS Index time(s)	MongoDB time(s)	MongoDB Index time(s)
10	5	5	8.96	8.72	5.89	7.26
10	10	17	18.7	13.98	12.59	12.05
10	20	17	16.95	14.36	27.77	28.27
10	40	17	14.87	17.8	55.8	59.5
20	5	5	24.62	20.31	6.52	8.29
20	10	17	46.6	49.34	11.98	15.35
20	20	45	104.67	142.97	28.15	28.28
20	40	46	142.55	142.75	54.09	50.08
40	5	5	75.45	75.49	8.01	6.19
40	10	5	136.68	190.67	11.91	11.99
40	20	45	704.79	584.84	30.17	27.82
40	40	122	1027.05	1361.62	49.23	55.49
40	60	122	1379.45	1075.67	94.82	104.5
60	5	5	114.92	126.7	5.66	6.26
60	10	17	301.36	337.02	11.94	10.67
60	20	45	1091.33	1119.77	26	29.18
60	40	122	3275.41	2746.24	49.6	50.24
60	60	204	3721.13	3302.4	98.38	98.32
60	80	204	3052.97	3186.87	103.85	104.62
80	40	122	4775.85	3169.52	57.3	49.25
80	60	204	4827.24	2249.57	111.63	105.23
80	80	298	6384.51	2276.98	113.78	114.94
80	90	298	6100.11	2518.85	134.74	122.6

Table 3: Average Time for Distinct Cuisine problem.

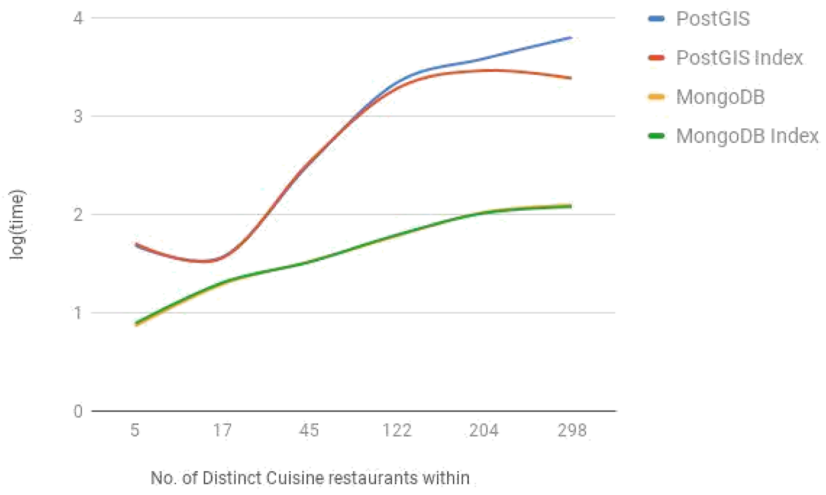

Figure 2: Graph of No. of restaurants vs average time for all datasets in D2.