Testing Spatial Data Deliverance in SQL and NoSQL Database using NodeJS Fullstack Web App

Dany Laksono
Department of Geodetic Engineering
Faculty of Engineering Universitas Gadjah Mada
Yogyakarta, Indonesia
danylaksono@ugm.ac.id

***Abstract* — During the last decade, we saw an explosion of geospatial data being produced. Most of which coming from GPS-enabled devices available for general consumers. The large amount of geotagged data coined the term 'Geospatial Big Data', indicating the semi-structured and unstructured nature of such data. SQL relational databases have been known in the past to handle geospatial data very well. However, the abundance of geospatial big data pushed forward the need for NoSQL database which is expected to perform better in terms of handling and storing geospatial big data. This paper discusses the quantitative comparison of performance between the SQL (i.e., PostGIS) and NoSQL (i.e., MongoDB) databases in handling geospatial big data. A NodeJS-based angular-framework web app was developed to test the real-world performance of MongoDB and PostGIS in handling a large amount of simulated geospatial data. A different number of points were generated for testing the geospatial data storing and loading capability of both the databases. The test was conducted by comparing the result of XHR (XML HTTP Request) of both databases in each case. The result showed that NoSQL database, i.e. MongoDB, performs better in loading big geospatial data compared to traditional SQL database using PostGIS.**

Keywords—NoSQL, MongoDB, PostGIS

I. INTRODUCTION

The inclusion of big data into the geospatial community is nothing new. In fact, the geospatial data itself is often characterized as 'big' in terms of volume, velocity, and variety [1]. Modern geospatial data are coming from mobile devices as geolocated data, geotagged images, sensors with GPS data, geocoded tweets, etc. This flood of geospatial information demands new kinds of data storage, processing, analysis and visualization.

Traditional SQL database has been known to performs well on storing and processing spatial data for web GIS applications. PostGIS is one of the most widely used relational spatial database capable of handling various spatial data analysis. However, the spatial big data brings new challenges to the traditional relational database, especially when the spatial data distribution via webGIS is concerned. For an effective user experience (UX) in a webGIS displaying large amount of spatial data, the loading time between server data storage, middleware, and client-side web application should be as short as possible. Thus, an emerging trend is to utilize NoSQL database for handling a large amount of spatial data [2].

NoSQL databases are designed to cater the needs for huge numbers of data. The structure of NoSQL (i.e. horizontally scaled, schema-less database) allows storage and analysis of large unstructured and semi-structured data. Although being relatively new for geospatial applications, the growing numbers of big spatial data open up new possibilities for analysis and visualization. Thus, the utilization of NoSQL database is gaining grounds in geospatial worlds. Investigations on the utilization of NoSQL database were performed in [3], where NoSQL is shown to perform effectively and efficiently in delivering spatial data. In [4], NoSQL and SQL database were both tested as a backend for OGC WMS request delivered using Geoserver. Other research such as [5] also suggests the efficiency of storing big geospatial data in a NoSQL database, especially MongoDB. Furthermore, [2] conducted a comparison between SQL and NoSQL database. This research, as well as the formers, showed that NoSQL outperforms SQL database for storing large amount of spatial data.

This paper aims to further suggest comparisons between SQL and NoSQL database in a NodeJS environment. The NodeJS Fullstack framework (i.e. NodeJS, ExpressJS, AngularJS, MongoDB and PostGIS) was chosen for its non-blocking nature and efficiency in handling big data. Furthermore, this paper investigates the required amount of time needed to load a large amount of POIs from both SQL (i.e. PostGIS) and NoSQL (i.e. MongoDB) database in NodeJS environment. Tailor-made POI data were generated using QGIS and imported to both databases. A web application based on MEAN framework (MongoDB, ExpressJS, AngularJS, and NodeJS) was built for testing the deliverance of this large geospatial data. Furthermore, we summarize the experience of building a MEAN web app for delivering spatial big data using SQL and NoSQL database.

The organization of this paper is as follows: Chapter II describes the related works and theoretical backgrounds related to SQL and NoSQL database in web application. Specifically, on the utilization of NoSQL database for geospatial applications. Chapter III explains the methodology used in this paper, Chapter IV reports our findings in the performance of PostGIS and MongoDB for serving large amounts of spatial data through the web app. Finally, Chapter V concludes our paper and provides suggestions for future research.

II. THEORETICAL BACKGROUND

A. NoSQL Database for Geospatial Applications

Today's database could be roughly divided into two large categories: SQL and NoSQL database. The SQL database such as PostGIS, MySQL or Oracle are so-called traditional RDBMS which store data as tables related to each other. NoSQL (better suited as 'Not only SQL') differs from traditional SQL database in the way data are organized in a database. The NoSQL approach to storing data could be divided into four categories [6]:

a. Key-value storage, where data are stored as pairs of keys and values. Example of this kind of the database is DynamoDB used by Amazon

b. Column storage, where data are stored in columns associated with row keys. Google uses this kind of data for its BigTable database

c. Document storage, where data are stored in a single document. Example of this data is MongoDB and CouchDB

d. Graph Database, which uses relation between nodes to form network of nodes. Neo4J is one of the database using this model.

Only a few of NoSQL database support geospatial operations. Examples such as MongoDB, CouchDB or Azure DocumentDB are able to store and perform some kinds of spatial analysis. The capabilities are including, but not limited to [2]:

- Storing various kinds of spatial data format
- Spatial analysis (spatial queries and geoprocessing)
- Spatial indexing

Overall, the NoSQL databases have some advantages over traditional relational SQL databases, such as its ability to store and query large amount of unstructured data. However, relational database such as PostGIS also possesses advanced geospatial query and analysis suitable for most of geospatial data.

B. NodeJS Fullstack Framework

NodeJS is a Javascript implementation of server-side application framework. NodeJS's nature of event-driven, non-blocking I/O model suits well for real-time communication between client and server in a web application [7]. The non-blocking nature of NodeJS is the basis for building web app capable of serving big data to the web. A NodeJS framework consists of server-side and client-side applications built on top of NodeJS into a fully capable web application

ExpressJS serves as the middleware which connects client and server via ExpressJS routes. The functionalities of ExpressJS is vital in an MVC (model-view-controller) web application since middleware separates the model (database) from the view (web interface). For the view (interface) of the web app, AngularJS is one of the popular alternatives. These Javascript libraries, when combined with MongoDB database are collectively named MEAN framework.

The whole structure of a MEAN framework differs from the traditional LAMP framework (Linux, Apache, MySQL and PHP). One obvious advantage is that the framework uses single dynamic programming language, i.e. Javascript. The framework, including middleware, is used to connect the controller to database model. Given the Javascript nature of MongoDB (which uses Binary JSON format), the CRUD (Create, Read, Update, Delete) operations can be performed faster albeit conducted on a large dataset [8].

An implementation of NodeJS fullstack application is Angular-Fullstack [9]. The framework is used to build MEAN/SEAN framework using NodeJS, AngularJS, ExpressJS and either Mongoose or Sequelize to connect the app to MongoDB or PostgreSQL, respectively. Mongoose and Sequelize are ORM (object-relational mapping) libraries which translates the request form a web app controller into database queries [10].

LAMP framework with PostgreSQL/PostGIS database has been a de-facto solution for building web GIS apps. With the emergence of unstructured big spatial data, the MEAN/SEAN framework should be able to perform better, hence this paper. In the following chapter, we discuss about the methodology used in this paper to conduct the testing.

III. METHODOLOGY

In order to conduct the testing, some tailor-made POI data were generated and imported into a MEAN/SEAN web app. The web app was developed using Angular-fullstack framework using Mongoose and Sequelize to handle both MongoDB and PostgreSQL/PostGIS database. Chrome Developer Tools were utilized to perform XHR testing of the POIs using both the database for each simulation. The following sections discuss about the implementations.

A. Architecture of The Web App

The web app developed for testing purposes was built using Angular-fullstack, which consist of the following components:

a) NodeJS: The web server handling requests and delivering responses

b) ExpressJS: NodeJS Middleware to redirects routes and managing ORMs

c) AngularJS: A Javascript MVC library for building user interfaces

d) Mongoose: NodeJS ORM for MongoDB

e) MongoDB: NoSQL database used for storing various numbers of POI data

f) Sequelize: NodeJS ORM for Relational Databases such as PostgreSQL, MySQL, MSSQL, etc.

g) PostgreSQL (with PostGIS extension installed): A relational database used to store geospatial data

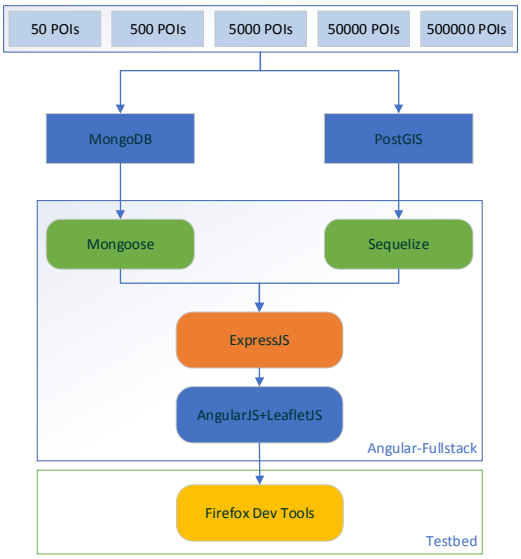

Figure 1: Web App Architecture

The architecture of the web app is depicted in Figure 1. Angular-fullstack serves as the main generator which builds the MVC (Model-View-Controller) of the web app. ExpressJS Endpoints were created for each sample data in both MongoDB and PostGIS (Table 1), which redirect requests to Mongoose and Sequelize, respectively. Both ORMs forward the request to each model (database) and return the test data which then displayed using LeafletJS in the client.

Table 1: Endpoints for testing in the WebApp

Endpoints	ORM
/api/Mongodb50s	Mongoose
/api/Mongodb500s	
/api/Mongodb5000s	
/api/Mongodb50000s	
/api/Mongodb500000s	
/api/postgis/50s	Sequelzie
/api/postgis/500s	
/api/postgis/5000s	
/api/postgis/50000s	
/api/postgis/500000s	

A single-page web app was built with Angular-Leaflet [11], an AngularJS Directive for LeafletJS, to receive and serve the loaded POIs into the map. The frontend was also built using Angular-fullstack, which also built the controller for managing the web app view to perform request to each endpoint.

B. Importing test data into PostGIS and MongoDB

Four different numbers of POIs, i.e. 50, 500, 5000, 50000 and 500000 were randomly generated using QGIS Toolbox's random point generator. Both sets of POIs were then imported into PostGIS and MongoDB. QGIS was employed to import the dataset into PostGIS, while a Python script [12] was used for loading the data into MongoDB. Figure 2 shows the study area and dataset generated in QGIS.

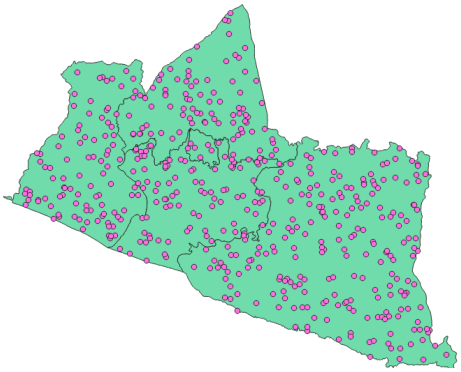

Figure 2: Test Dataset (shown here the 500 POIs)

Both databases were optimized using B-tree spatial index (PostGIS) and 2Dsphere Index (MongoDB). Each of the test datasets was written as PostgreSQL table and MongoDB collection containing large numbers of POIs for testing purposes.

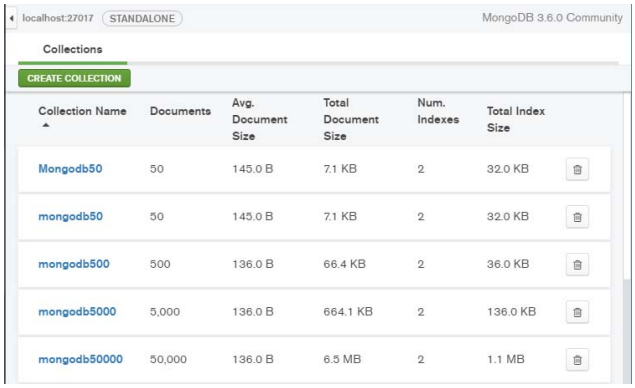

Figure 3: MongoDB collections of the test dataset

C. Testing Geospatial Data Loading

The web app was then configured to handle requests from the client. Leaflet-AngularJS was used to visualize the data and to perform AJAX request to ExpressJS routes handling each request to Mongoose and Sequelize, which subsequently converts the request into each database queries. The results are then delivered to the client's browser using XMLHttpRequest.

The response to each request was then analyzed using Firefox's Developer Tools and saved as HAR file for further analysis (Figure 4). Using the Angular-fullstack generator's web app, the testing was then performed in a local environment. HAR File was analyzed for XHR response's time for each POI dataset for both databases.

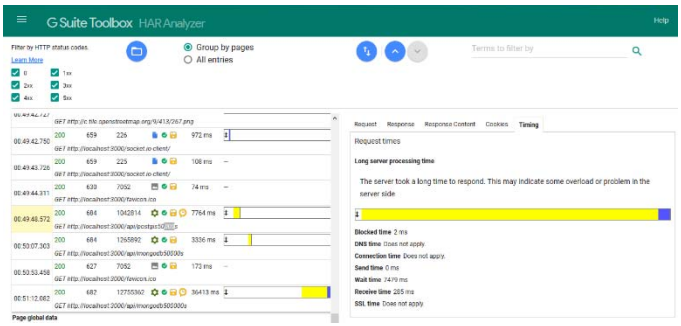

Figure 4: HAR Analyzer to Analyze the Testing Results

IV. Result and Discussion

The web app was built using Angular-fullstack generator run in localhost environment. Figure 5 shows the NodeJS Fullstack application used as the main interface to load the testing dataset from both databases. Each POI dataset was put into different ExpressJS endpoints to eliminate latency effects. Each request was performed using AngularJS frontend, and the response is then loaded into a LeafletJS web map. The response time, i.e. XMLHttpRequest (XHR) time was recorded and analyzed. Specifically, this paper used Firefox developer tool to perform network testing of XHR response time.

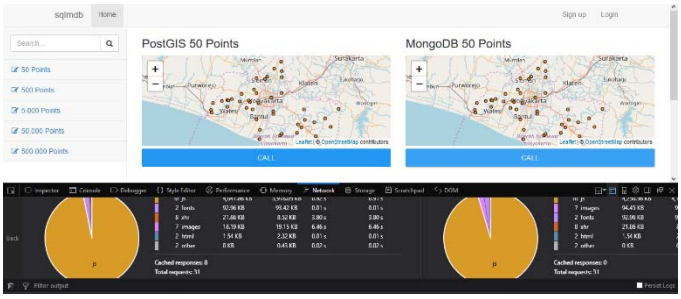

Figure 5: NodeJS Web App built for testing purposes

Requests from Angular-Leaflet are accepted by the controller, which perform the actual HTTPRequest to Endpoints. Requests are translated into native database queries using either Mongoose or Sequelize. Further, each database sends a response which is accepted by the controller and visualized through Angular-fullstack.

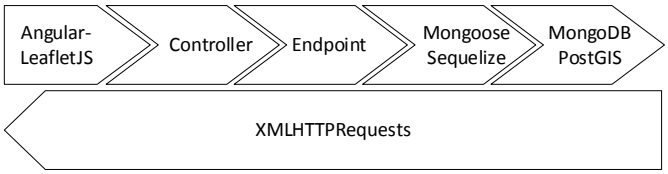

Figure 6: Route of Requests and Responses

The result of this research is shown in Table 2 below. The response time, expressed in milliseconds, was obtained through Firefox network developer tools using an XHR filter of JSON data. The response time represents the time it needs for the response to be sent from database to the client, which subsequently used to display the data.

Table 2: Result of Testing

No. of POIs	XHR Timing (*ms*)	
	PostGIS	MongoDB
50	206	83
500	632	239
5000	1276	621
50000	17458	3470
500000	∞ (failed)	36413

The result shows that the loading time for MongoDB is significantly smaller compared to that of PostGIS as the number of the data grows. On smaller dataset, the time difference, though noticeable, are less significant compared to the larger dataset. This result, obtained using NodeJS fullstack applications, showed that NoSQL database outperforms traditional SQL database (i.e. PostGIS) for delivering a large amount of spatial data through XHR.

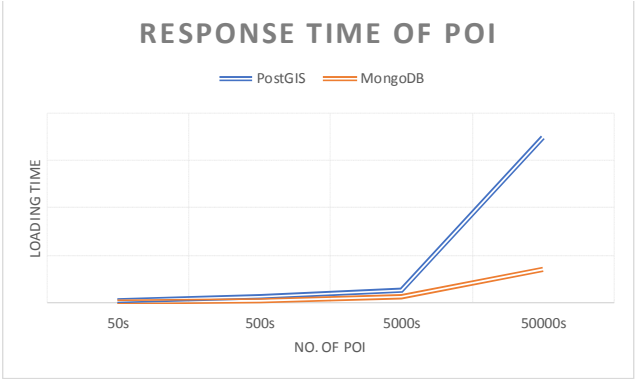

Figure 7: Loading time of POIs

This paper contributes to the comparison of spatial data deliverance between NoSQL database and SQL database by measuring XHR response time for a different number of test data. The real-world application of this research could be implemented when a large amount of spatial data is needed to be loaded and visualized into a web app. The test showed that NoSQL database performs much better on delivering large dataset compared to traditional SQL database through XHR. This result support previous studies on NoSQL performance over SQL database as previously mentioned. However, the result showed that merely delivering the data using XHR is not enough, as shown in the case of the larger amount of POI. Thus, either server-side or client-side preprocessing are needed in order to provide better user experience. Also, further tests should be performed on the geospatial-related queries for both databases, which may yield different results regarding the deliveries of data.

V. Conclusion

With the growing numbers of spatial data available, it is shown that traditional relational databases face a huge challenge in handling such big spatial data. Thus, for a web GIS application ensuring smooth user experience, an alternative should be considered. NoSQL database is known to be performing well in handling big data of semi-structured and unstructured data, and thus possesses great potential for geospatial applications on the web. This research shows that NoSQL database, i.e. MongoDB, outperforms relational database (PostGIS) in terms of loading simulated POI data. A NodeJS fullstack application was built to support testing of the loading. In general, loading time for MongoDB spatial data loading is faster compared to that of PostGIS using same dataset and environment.

References

[1] M. R. Evans, D. Oliver, X. Zhou, and S. Shekhar, "Spatial Big Data: Case Studies on Volume, Velocity, and Variety," 2013.

[2] E. Baralis, A. D. Valle, P. Garza, C. Rossi, and F. Scullino, "SQL versus NoSQL databases for geospatial applications," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 3388–3397.

[3] P. Amirian, A. Basiri, and A. Winstanley, "Efficient Online Sharing of Geospatial Big Data Using NoSQL XML Databases," in *2013 Fourth International Conference on Computing for Geospatial Research and Application*, 2013, pp. 152–152.

[4] S. Schmid, E. Galicz, and W. Reinhardt, "WMS performance of selected SQL and NoSQL databases," in *International Conference on Military Technologies (ICMT) 2015*, 2015, pp. 1–6.

[5] X. Zhang, W. Song, and L. Liu, "An implementation approach to store GIS spatial data on NoSQL database," in *2014 22nd International Conference on Geoinformatics*, 2014, pp. 1–5.

[6] J. Bhogal and I. Choksi, "Handling Big Data Using NoSQL," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, 2015, pp. 393–398.

[7] A. Q. Haviv, *MEAN Web Development*. Packt Publishing Ltd, 2014.

[8] D. Hows, P. Membrey, and E. Plugge, "MongoDB Basics," 2014.

[9] "angular-fullstack/generator-angular-fullstack: Yeoman generator for AngularJS with an Express server." [Online]. Available: https://github.com/angular-fullstack/generator-angular-fullstack. [Accessed: 02-Apr-2018].

[10] "Manual | Sequelize | The node.js ORM for PostgreSQL, MySQL, SQLite and MSSQL." [Online]. Available: http://docs.sequelizejs.com/manual/. [Accessed: 02-Apr-2018].

[11] "angular-ui/ui-leaflet: AngularJS directive to embed an interact with maps managed by Leaflet library." [Online]. Available: https://github.com/angular-ui/ui-leaflet. [Accessed: 09-Apr-2018].

[12] R. Ciesla, *geojson-mongo-import.py: Import GeoJSON file into MongoDB using Python*. 2018.