Spat. Inf. Res.
DOI 10.1007/s41324-016-0059-1

Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries

Sarthak Agarwal[1] · **K. S. Rajan**[1]

Received: 16 February 2016 / Revised: 16 October 2016 / Accepted: 18 October 2016

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. Moreover, this is gaining ground in the context of increased shift towards Geospatial Web Services on both the Web and mobile platforms especially in the user-centric services, where there is a need to improve the query response time. This paper attempts to evaluate the performance of an existing NoSQL database 'MongoDB' with its inbuilt spatial functions with that of an SQL database with spatial extension 'PostGIS' for two fundamental spatial problems—line intersection and point containment problem, across a range of datasets, with varying feature counts. 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.

Keywords Spatial databases · Sql · Nosql · Intersection · Containment · Spatial queries

This paper was revised from the paper initially presented in FOSS4G Seoul 2015 Conference.

✉ Sarthak Agarwal
sarthak.a@research.iiit.ac.in

K. S. Rajan
rajan@iiit.ac.in

[1] Lab for Spatial Informatics, International Institute of Information Technology, Gachibowli, Hyderabad 500032, India

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 Geo-Spatial domain we use Spatial Databases. Satellite images are one prominent example of spatial databases [1]. To extract spatial information from a satellite image, it has to be processed on 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. While these have been able to store and manage very large datasets, their inherent need to per-define its structure, database schema, makes it difficult to remodel or change it as per the evolving need, especially in the context of user-driven data collation and collaboration. 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.

NoSQL is an umbrella term for a loosely defined class of non-relational data stores and best described as 'Not only SQL' [2]. The recent past has seen the development of NoSQL non-relational databases, which are now being adopted for spatial object storage and handling too. Moreover, this is gaining ground in the context of increased shift towards GeoSpatial Web Services on both the Web and mobile platforms especially in the user-centric

Published online: 01 November 2016

services, where there is a need to improve the query response time, these 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.

Relational databases are not designed to function with distributed systems; combining multiple tables across different computers is challenging and gives mostly a decrease in performance while NoSQL is distributed databases which can be spread over multiple servers and queried simultaneously [3]. Relational databases are good for structured data. For example, sales figures fit well in related tables. Unstructured data, such as points and lines over a network are not very suitable for a RDBMS, while NoSQL database systems are schema-less database systems where we can append any number of geometry columns in a table and store multiple geometries within the same column in a table which is an essential property with respect to spatial context.

According to Kolb [4], 90 % of the worldwide generated data until 2012, which is 2.5 Exabyte per day, was generated within the previous two years. To use, store and analysis the data of this magnitude we need very efficient database systems. When compared to relational databases, NoSQL databases are more scalable and provide superior performance, and their data model addresses several issues that the relational model is not designed to meet [5].

For the satisfaction of the user's significant characteristics of a database like scalability, performance and latency play a crucial role. Especially social media projects, like Facebook and Google+, with high user traffic, use other database management systems like 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 [6].

NoSQL data stores may provide advantages over relational databases, but 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 [7–9] and some practical implementation of those theories [10–12].

This study attempt to evaluate the performance of an existing NoSQL database and SQL database with respect to two of the primary geometric queries by scaling up the size of their datasets. We here are interested in observing which data model performs better with these problems in terms of time complexity without accounting for indexing of the data.

The traditional routing algorithm requires a server to process the query and send them to the client, but here we are focusing on querying the client itself. The motivation behind this work is to minimize the dependency for server during mobile-platform based routing. We want the client itself to act as a server during routing and for that we need a database system that can be easily ported to the mobile devices and can serve as a server, so here we are comparing the performance of SQL versus NoSql databases since NoSQL databases are document based databases which are very light and can exploit the increasing memory of our mobile smartphones. The spatial operations that are fundamental to a scenario described above is the line intersection queries. While the other most common spatial query is the search of a location and the zone, region or city it belongs too. Hence, we have chosen these two queries for this analysis as shown in Fig. 1.

2 Review of spatial databases

Similar to traditional databases, Spatial databases were also written initially in RDBMS but as technology evolved they are now adapted to NoSQL.

2.1 RDBMS (PostGIS/PostgreSQL)

PostgreSQL is a powerful, object-relational database management system (ORDBMS). It is released under a BSD-style license and is thus free and open source software. PostGIS adds support for geographic objects to the PostgreSQL object-relational database. The interface language of the PostgreSQL database is the standard SQL, which allows for inserts, updates and queries of data stored in relational tables. Data operations for PostgreSQL are created in SQL.

PostgreSQL with its extension PostGIS has the most comprehensive geo-functionalities with more than one thousand spatial functions [13]. For a complete list of all implemented spatial functions, it is referred to the PostGIS handbook [13].

Features:

- It supports geometry types for Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultipPolygons and GeometryCollections.

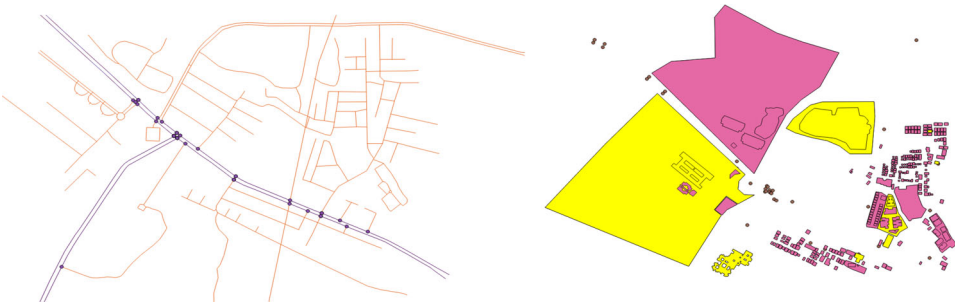

Fig. 1 Line intersection problem and point containment problem

- It supports spatial operators for determining geospatial measurements like area, distance, length, and perimeter.
- PostGIS also support R-tree-over-GiST (Generalized Search Tree) spatial indexes for high-speed spatial querying.
- The PostGIS implementation is based on "light-weight" geometries and indexes optimized to reduce disk and memory footprint.

Why PostgreSQL/PostGIS

Concerning spatial databases, PostgreSQL was one of the first databases to adopt the spatial databases. Being one of the first and major implementation of spatial functions PostGIS is highly optimized for spatial queries.

PostGIS has a large number of spatial functions which makes it relevant for this study. And being an open source software we can make any type of modifications and optimizations according to our need.

2.2 NoSQL (MongoDB)

MongoDB is a document-oriented data store. It is written in C ++ and its primary focus lies on high performance and retaining some friendly properties of SQL.

Currently, MongoDB uses GeoJSON objects to store spatial geometries. GeoJSON is an open-source specification for the JSON-formatting of shapes in a coordinate space. The GeoJSON spec is used in the geospatial community, and there is growing library support in most popular languages [14].

Each GeoJSON document (or sub-document) is composed of two fields [15]:

1. Type—the shape being represented, which informs a GeoJSON reader how to interpret the "coordinates" field.
2. Coordinates—an array of points, the particular arrangement of which is determined by "type" field [14].

Overview of MongoDB's current implementation

- There are multiple geospatial indexes per collection in MongoDB-2d, 2dsphere, and geoHaystack.
- Data can be imported from CSV files by converting it into GeoJSON objects.
- However, MongoDB does not support R-trees.

Currently in MongoDB, containment and intersection tests are supported for both flat and spherical geometries. Sorting by distance is supported too. This is because these operations are the most popular operational queries for a modern application. They meet most requirements of ideal location based applications.

With using the GeoJSON data structures the schema-less approach got some restrictions. However, the geographic representation needs to follow the GeoJSON structure in order to be able to set a geospatial index on the geographic information [15]. 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 WGS84 datum. MongoDB computes the geohash values for the coordinate pairs and then indexes the geohash values [12].

Why MongoDB

NoSQL databases currently lack support for spatial functions if we compare it to their counterpart. However, MongoDB and some other document based NoSql DB systems have started supporting some spatial functions and spatial indexes.

We choose MongoDB because it is the only document based NoSQL database that supports line intersection ($geoIntersects) and point containment ($geoWithin) queries (CouchDb only support Building Box query).

MongoDB is also an open source software which makes it convenient to add more spatial functions in the future.

2.3 Comparative analysis of spatial functions

Each of the implementation of spatial databases has a different approach for handling geometries.

Geometry/flat shape PostGIS supports various operations for geometry, while MongoDB only supports storing points and containment/intersection queries with box/circle/polygon. 2d index is designed to speed up this case. Although it is possible to leverage open source libraries like GEOS, MongoDb does not plan to expand the features for flat geometry, because most LBS applications work with GPS data, which nicely fits with the spherical model.

Geography/spherical shape MongoDB supports storing and querying spherical shapes in GeoJSON. 2dsphere index is for this case. PostGIS only supports containment and intersection natively, which are backed by MongoDB as well. The distance between two objects is also supported in PostGIS.

3 Datasets

For this analysis, the dataset was custom generated. Here is the link to the dataset used for the study. Please give the reference to the paper while using the dataset.

3.1 Line intersection dataset

The line intersection problem works for line geometry in the space and reports whether or not that geometry is being intersected by some other geometry. This problem is crucial in real time scenarios. So we generated several synthetic datasets.

We here are taking the assumption that all the line can be broken into horizontal and vertical line segment. Moreover, observing the interaction between these line segments will give us an excellent picture of real time interaction between line segments.

The dataset D1 consisted of two independent layers of horizontal lines and vertical lines with incremental lengths and their size varied from ten lines to ten million lines in each layer. Each line intersects at least one line in the other layer with some lines intersecting multiple lines.

The above dataset emulates most of the real time cases. For this dataset we are observing the following situations:

1. One Line versus One Line intersection which is the primary case.
2. One Line versus Many Lines intersection, for example, the scenario where we want to find out the number of small roads attached to a highway.
3. Many versus Many lines, for instance, the situation where we want to see the interaction between railway lines and roads.

3.2 Point containment dataset

Like the previous problem point containment problem also 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 D2 consisted of two independent layers, one of horizontal lines of incremental length and the size of the layer varied from 10 lines to 10 million lines. Another layer consisted of square boxes of different perimeter scattered both sequentially and randomly over the space with few lines completely inside the box, few partially inside the box and others completely outside the box.

We here are taking the assumption that all the line is made up of millions of pints, and if we generate a dataset with many lines of various lengths, it can emulate many of the real time cases. And observing the interaction between these line segments and boxes will give us an excellent picture of real-time interactions.

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 are 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.

4.1 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 × 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 Fig. 2 we will notice that MongoDB works much faster than PostGis in all the cases (from small datasets to large datasets). The difference in

performance is in the order of 100 (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 to a conclusion that the performance of both indexed and non-indexed line intersection queries is better in case of MongoDB as compared to PostGis (Table 1).

Experiment 1. Performance of Line Intersection Query

In the 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 fails at huge datasets and time complexity increases exponentially. But after indexing the geometries performance of both the database engine improved by a substantial factor.

If we observe the results from the experiments, we will notice that indexed datasets perform better than non-indexed dataset and PostGIS time increases the exponentially as size of dataset increases whereas MongoDB still performs within some bounds (Fig. 3, Table 2).

Experiment 2. Performance of Point Containment Query

5 Discussions and conclusions

The above analysis was necessary to evaluate the performance of NoSQL with respect to spatial databases and to tell whether NoSQL performs better than SQL in our current scenario. These results suggest that MongoDB performs better by an average factor of 25× which increases exponentially as the data size increases in both indexed and non-indexed operations for both line intersection and point containment problem. 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 functions most commonly used in spatial queries, especially for routing. Therefore, we choose line intersection and point containment queries. 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 versus NoSql databases which are comparatively lighter.

But 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

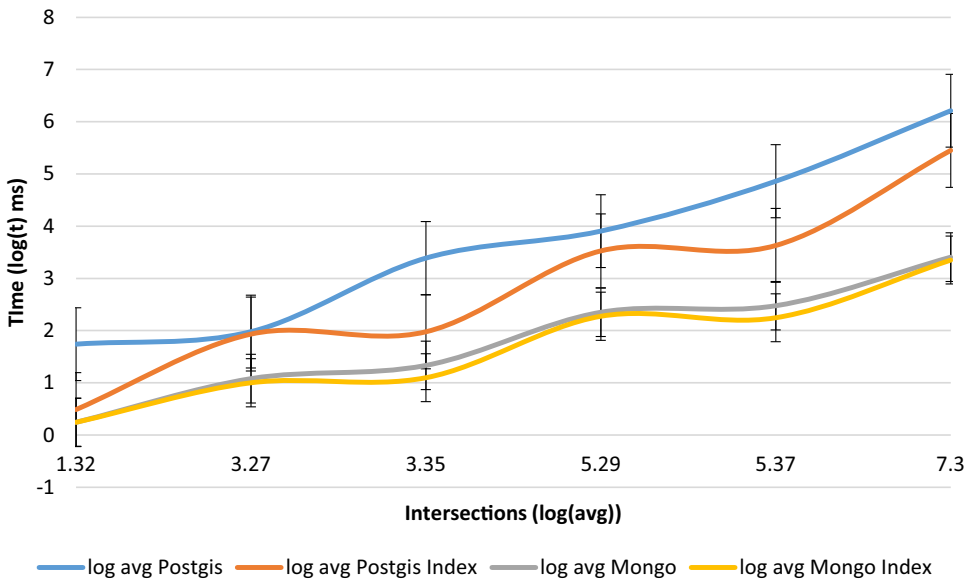

Fig. 2 Graph of intersections versus average time for all datasets in D1

Table 1 Average time for line-intersection query for D1 sub-datasets

Lines in layer 1	Lines in layer 2	No. of intersections	PostGIS	POSTGIS index	MongoDB	MongoDB index
10	10	21	3.59	2.943	1	1
10	100	21	4.652	3.045	1	1
10	1000	21	17.719	3.028	2	2
10	10,000	21	190.8	3.252	3	3
100	100	1875	95.2	85.579	12	10
100	1000	2231	408.573	90.31	22	11
100	10,000	2231	4474.7	99.252	21	14
1000	1000	195,691	8034.4	3355.436	224	188
1000	10,000	236,859	72,506.5	4258	299	176
10,000	10,000	19,731,383	1,655,740	287,469	2542	2236

Table 2 Average time for containment query for D2 sub-datasets

Lines in line layer	Boxes in box layer	No. of lines within	PostGIS	PostGIS index	MongoDB	MongoDB index
10	5	5002	8.69	8.92	6	8
10	10	17,004	16.68	15.29	12	14
10	20	17,004	16.69	15.21	27	28
10	40	17,004	15.18	17.03	52	51
20	5	5002	23.7	23.69	7	9
20	10	17,004	52.51	52.55	13	15
20	20	45,999	120.76	131.34	27	26
20	40	46,008	142.53	158.76	50	51
40	5	5002	66	72.71	8	7
40	10	5411	160.02	168.23	12	12
40	20	45,999	675.35	634.31	28	27
40	40	122,006	1184.25	1335.45	51	51
40	60	122,016	1339.23	1110.42	96	99
60	5	5002	113.44	113.84	6	7
60	10	17,004	306.05	287.46	13	12
60	20	45,999	1230.43	1129.02	28	29
60	40	122,066	2962.47	3069.11	51	54
60	60	204,024	3510.05	3566.72	98	106
60	80	204,024	2913.24	3286.71	99	105
80	40	122,006	4203.38	3423.05	53	55
80	60	204,024	5551.63	2509.91	102	107
80	80	298,023	NC	NC	106	108
80	90	298,032	NC	NC	144	116

user needs to calculate geoinformation on database level [9].

The results presented in the paper are only valid for the chosen database settings but they clearly show that No-SQL databases are a possible alternative, at least for querying attribute information.

PostgreSQL also has an implementation of NoSQL of its own, but 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

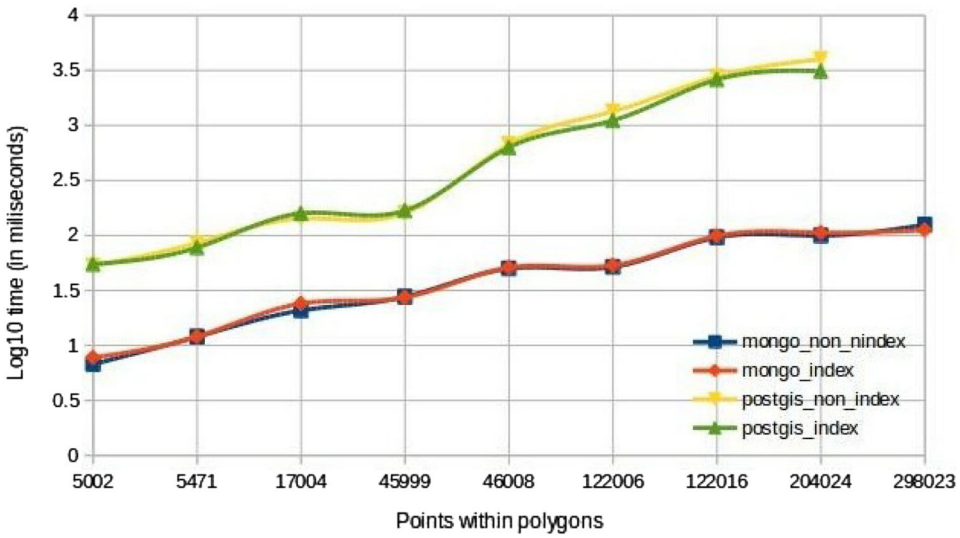

Fig. 3 Graph of lines within versus average time for all datasets in D2

such as shortest path problem to evaluate the performance of NoSQL on such platforms and also test the performance of MongoDB on distributed systems.

Acknowledgements We would like to acknowledge the responses and interactions with MongoDB developers through emails which helped during this research project.

References

1. Soni, G. (2013). Open source spatial database for mobile devices. *Computer Engineering and Intelligent Systems (IISTE)*.
2. Baas, B., Quak, W., Van Oosterom, P. et al. (2012). *NoSQL spatial: Neo4j versus PostGIS*. Master Thesis, Geographical Information Management and Applications.
3. Scholz, J. (2011). Coping with dynamic, unstructured data sets—NoSQL a buzzword or a savior? In M. Schrenk, V. V. Popovich, P. Zeile (Eds.), *Proceedings of 16th international conference on urban planning, regional development and information society REAL CORP* (pp. 121–129). ISBN: 978-3-9503110-1-3.
4. Kolb, L. (n.d.). *NoSQL-Datenbanken. Kapitel 1: Einführung; Universität Leipzig*. http://dbs.unileipzig.de/file/NoSQL_SS14_01_Intro.pdf.
5. NoSQL Databases Explained | MongoDB. White Paper (n.d.). https://www.mongodb.com/nosql-explained.
6. Lourenço, J. R., et al. (2015). Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data, 2*, 18. doi:10.1186/s40537-015-0025-0.
7. 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 Jordão, Brazil, November 27–29, 2011.
8. Xiao, Z. F., & Liu, Y. M. (2011). Remote sensing image database based on NOSQL database. *Geoinformatics, 2011*, 1–5.
9. Schmid, S., Galicz, E., Reinhardt, W. (2015). *Performance investigation of selected SQL and NoSQL databases*. AGILE 2015, Lisbon, June 9–12, 2015.
10. Popescu, A., Bacalu, A.-M. (2009). *Geo NoSQL: CouchDB, MongoDB, and Tokyo cabinet.* http://nosql.mypopescu.com/post/300199706/geo-nosql-couchdb-mongodb-tokyo-cabinet.
11. Steiniger, S., & Hunter, A. J. S. (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)*. doi:10.1007/978-3-642-10595-1_15.
12. van der Veen, J. S., van der Waaij, B., Meijer, R. J. (2012). Sensor data storage performance: SQL or NoSQL, physical or virtual. In: *IEEE fifth international conference on cloud computing*, Honolulu, Hawaii, USA, June 24–29, 2012.
13. PostGIS Development Group, Postgis Manual. http://postgis.net/docs/index.html. Accessed January 14, 2014.
14. Chris. (2014). *A primer on geospatial data and MongoDB—mLab Blog*. http://blog.mlab.com/2014/08/a-primer-on-geospatial-data-and-mongodb. Accessed August 19, 2014.
15. Butler, H. et al. (2015). *The GeoJSON format specification. Internet Engineering Task Force*.