Performance investigation of selected SQL and NoSQL databases

Stephan Schmid
University of the Bundeswehr
Werner-Heisenberg-Weg 39
Neubiberg, Germany
stephan.schmid@unibw.de

Eszter Galicz
University of the Bundeswehr
Werner-Heisenberg-Weg 39
Neubiberg, Germany
eszter.galicz@unibw.de

Wolfgang Reinhardt
University of the Bundeswehr
Werner-Heisenberg-Weg 39
Neubiberg, Germany
wolfgang.reinhardt@unibw.de

Abstract

In the today's high-tech world the amount of data and especially spatial data is growing from day to day. Databases are one of the best ways to store data. Several database concepts exist for storing data, while the mostly used is the relational database concept. Relational databases are very well in use for the storage of spatial and non-spatial data. But for example in social media, like Facebook or Twitter, relational databases often reach their limit of performance. For huge amounts of data and frequent data changes NoSQL-databases can be used. The paper gives an overview of selected SQL and NoSQL-databases according to their geo-functionalities. In addition it compares two document based NoSQL-databases with a relational database by several performance tests.

Keywords: NoSQL, Databases, SQL, MongoDB, CouchBase, PostgreSQL.

1 Introduction / Motivation

A characteristic of the today's high-tech world is the vast amount of stored data. According to Kolb, 90% of the world wide generated data until 2012, which is 2.5 Exabyte per day, was generated within the previous two years [6].

Different database systems can be used for the storage, use and analysis of this growing amount of data. According to Paul Campaniello from MySQL-Scalability mostly relational database management systems are used [9].

For the satisfaction of the users 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) approach is used. NoSQL-databases are increasingly used to deal with simultaneously high read and write requests related to large datasets.

In many fields, spatial data fulfills the criteria of fast changing, large datasets which makes continuous indexing of the data necessary. It might be expected that future data storage concepts for spatial data are more often based on NoSQL-databases more often. Besides querying the growing amount of spatial data, there is a need for performant analysis.

The paper focuses on the current status of NoSQL-databases for their usability in geo-applications. Therefore it presents a comparison between two document-based NoSQL-databases (MongoDB, CouchBase) and a relational-database (PostgreSQL). This mainly includes an overview of existing geo-functionalities as well as several performance tests.

Outline: Section 2 introduces two NoSQL-databases with a common definition for NoSQL-databases and a classification based on their main characteristics. Section 3 compares the differences between the relational data model and the NoSQL-approach. It describes storage concepts and compares the geo-functionality between a relational database and the selected NoSQL-databases. Section 4 describes the test setup and test procedure for the performance tests. Further it analyzes the test results. Section 5 summarizes the paper and gives an outlook to future work.

2 NoSQL-Databases

The term "NoSQL" already exists since 1998. Carlo Strozzi named an open-source database "NoSQL" to make clear, that his project does not support any SQL interface [12]. The underlying concept of his NoSQL-databases waives relations therefore the expression NoREL would be more appropriate. Nowadays "NoSQL" stands for "Not only SQL". It is no common definition for NoSQL-databases available but Edlich et.al. [3] point out 7 important characteristics. NoSQL-databases:

- are not based on a relational approach,
- scale horizontal,
- are often open-source products (although proprietary products are available),
- don't need a defined schema,
- provide an API for the integration in other software products,
- use a decentralized architecture for the easy replication of data,
- follow the BASE principle (**B**asically **A**vailable, **S**oft State, **E**ventually Consistent).

According to their characteristics NoSQL-databases can be divided into four groups [5].

Key value stored databases: this kind of NoSQL-databases use a simple schema based on key-value pairs

Column stored databases: data is stored in columns instead of rows.

Document stored databases: data is not stored in tables but in documents. Documents refer to structured files, like JSON, YAML or RDF.

Graph stored databases: data is stored as graph or tree structures which link the different data aspects.

3 NoSQL-DB's for geoapplications

The increasing amount and volume of spatial data leads to new challenges in storing geospatial data. Introduced in the 1970s, the relational database, mathematically based on the relational algebra, offers ways for structuring, keeping, and analysing/using spatial and non-spatial data. Therefore a data model is needed to logically structure the data that is being stored. These models are the first step and the biggest determiner of how a database application will work and handle the information because data is kept in relations. In the early 2000s the NoSQL approach occurred as an unstructured approach which aim is to eliminate the limitations of strict relations.

But which NoSQL-databases are capable of storing and processing geospatial data? According to recently performed investigations for this paper from the four categories of NoSQL-databases mentioned in section 2, only the document stored databases and graph stored databases are widely used for storing spatial data. For graph stored databases Neo4j includes a spatial extension which supports all simple feature geometry types and can be used for route analysis or proximity searches [1].

This paper concentrates on document-stored databases. At this point in time there are two widely used open-source NoSQL-databases which support geospatial data, CouchBase and MongoDB. When using document based NoSQL implementations, they don't use any database schemas or tables, they use documents to store data (it is expected that the schema is part of the application layer). Documents are semi-structured standardized files, like JSON, YAML or XML. The two investigated NoSQL-databases use JSON (JavaScript Object Notation) as documents. JSON documents can be constructed in two different ways:

The first way is nesting documents inside each other. This option can work for one-to-one or one-to-many relationships. As an example features can be stored as one feature per document or in a Feature Collection, where all features are nested documents.

The second option is to store a reference to another document. This is done by setting one field in the JSON document as the reference key, where the value of this field is the id of the referenced document. NoSQL-databases will only retrieve the referenced document when the user requests data inside the referenced document. NoSQL-databases don't need an additional collection (table) for joining the data. Referencing another document is comparable to the foreign key concept of relational databases.

Spatial data can be stored in GeoJSON which is a format for encoding a variety of geographic data structures [2]. A GeoJSON document may represent a:

-**Geometry** which is a GeoJSON object where the type value is one of the following ISO/OGC geometry types: "Point", "MultiPoint", "LineString", "MultiLineString", "Polygon", "MultiPolygon" or "GeometryCollection".

-**Feature** which is a GeoJSON object with the type "Feature". A feature must have a "geometry" and several "properties".

-**Collection** of features which is a GeoJSON object with the type "FeatureCollection". A "FeatureCollection" must have some "features" which are organized corresponding to "features" as defined above.

With using the GeoJSON data structures the schema free 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 [7]. Indexing is important to speed up query processing. Different NoSQL-databases use different indexing techniques.

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. A precise description for the indexing techniques of the geohash values is not available at the moment [8].

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 [10]. 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 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. Table1 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.

Table 1: Geo-Functions of the investigated databases

PostGIS (selection)	MongoDB	CouchBase
ST_Within	$geoWithin	BBOX
ST_Intersects	$geoIntersects	
	$near	
ST_DWithin + Order by distance	$near + parameter (maxDistance)	
ST_Area		
…		

The overview of the implemented geo-functions show 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 [11].
The two NoSQL-databases have very limited implemented geo-functions. MongoDB just implements three functions whereas CouchBase just implements one geo-function.

4 Performance tests for vector data

4.1 Testsetup

Xiao [13] already investigated the performance of storing raster data in NoSQL databases. This paper concentrates on vector data, which is to our knowledge not yet available. For the performance tests a virtual machine with the following hardware configuration was used:

- 10GB RAM
- 8 CPU 2,5 GHz
- Microsoft Windows Server 2008 R2

This hardware was used for all tested databases, no shared server system setup was used. To test the performance two typical queries were defined, one queries attributes of the objects and the other one calculates spatial data using the geo-function “within”. In PostgreSQL the requests were performed using SQL; and the standard PostGIS Gist-Index was used.
MongoDB uses the 2dsphere index while the requests were taken using JavaScript.
In CouchBase the requests were performed by using the REST API, because the existing JavaScript API currently does not support geo-queries. In CouchBase it was necessary to generate views for requesting the data, a view acts like an index.
Test data from OpenStreetMap with different sizes were imported into the databases. An overview of the data is given in Table 2.

Table 2: Test data used from OpenStreetMap

Level	Region	Size
Subregion	Niederbayern	38,9 MB
State	Bayern	501 MB
Country	Germany	2,1 GB

The tests didn’t investigate the memory usage or the storage overhead. The data was imported to PostgreSQL using the GDAL importer. For MongoDB the mongoimport-tool was used CouchBase doesn’t have any import tool. Therefore it is necessary to develop a JavaScript-file based on NodeJS. GDAL offers an easy way to convert the data from OpenStreetMap to JSON. Hence, the result of that standard data conversion is a flat structure in the JSON document according to the GeoJSON specification, no nested or complex structures were used.

All databases are installed according to their standard installation instructions. In all databases an index for the geometry is used. PostgreSQL uses the GIST (Generalized Search Tree)-index, MongoDB uses the 2D-sphere index. For CouchBase several views on data were created which act like an index.

The total time for processing the requests was measured using the Apache JMeter, which is a Java based performance measurement tool [4].
The following two queries were defined:

1. Queries on attribute-information: One feature of each geometry type (point, line, and polygon) is selected based on its attribute (OSM_id). For example, from all point objects the point with the OSM_ID=1082817686 is selected.

```
Select * from points WHERE osm_id = '1082817686'
```

2. Requests using a geo-function “within”

The second query uses the geo-function “within” to calculate data on database level. It delivers all points within the defined polygon. The polygon is of the same size for all requests.

```
Select * from points WHERE
(ST_Within (wkb_geometry, ST_GeomFromGeoJSON('
{
    "type": "Polygon",
    "coordinates": [
      [[12.782592773437498,
        48.38817819201506 ],
       [12.782592773437498,
        48.54843286654265,
       [13.1231689453125,
        48.54843286654265],
       [13.1231689453125,
        48.38817819201506],
       [12.782592773437498,
        48.38817819201506]]],
    "crs": {
    "type": "name",
    "properties": {
    "name": "EPSG:4326" } } }
')) is true)
```

For simulating realistic conditions the tests were performed with an increasing amount of users. Therefore three user categories were tested:

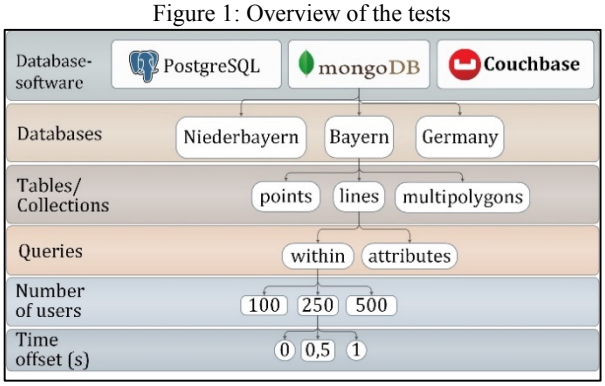
- 100 users,
- 250 users,
- 500 users.

In reality the user requests are often conducted with a small time offset. There is always an unknown time offset between the requests. Simultaneous requests occur randomly. Therefore the tests were executed with different time offsets:

- simultaneously, no time offset
- with half second time offset
- with one second time offset

The tests with an increasing number of users and time offset as well as requests in attribute and geo-functions were conducted with all datasets and different file sizes. In total this leads to 486 tests. Figure 3 gives an overview of the performed tests. Due to space limitations not all results of the tests are given in the paper.

Figure 1: Overview of the tests



4.1.1 Test results:

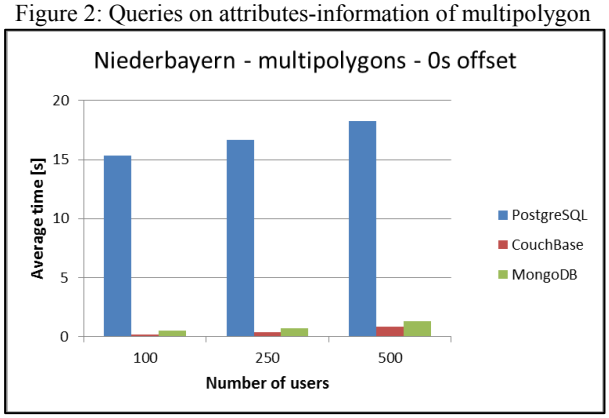
In the following the results of the performance tests are exemplarily shown and discussed. This includes results for queries on attribute-information as well as queries using the geo-function "within".

Queries on attribute-information

Figure 4 shows the response time for requesting attribute-information of a multipolygon. It compares the three tested databases. All tests were conducted at the same time.

It gets clear that the response time for multipolygon of the NoSQL-databases are less than for PostgreSQL. The PostgreSQL response time increases with the number of users from 15 sec./100 users to 18 sec./500 users. The response times for the NoSQL-databases, both MongoDB and CouchBase, slightly increase from 1 sec./100 users to 4 sec/500 users. MongoDB and CouchBase behave almost similar while CouchBase is a little faster. Further test results show that PostgreSQL always needs more time for answering the queries than both of the NoSQL-databases regardless of the different geometry types or size of the datasets.

Figure 2: Queries on attributes-information of multipolygon



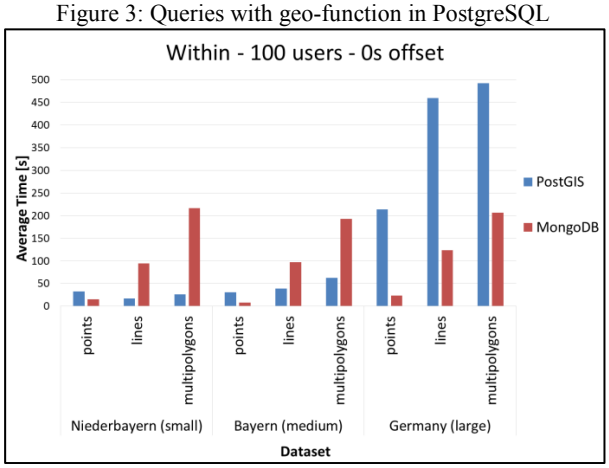
Queries using the geo-function „within"

The investigation with the geo-function "within" could not be carried out using CouchBase. The database crashed regularly. Figure 5 compares the results for MongoDB and PostgreSQL. The diagram shows the different datasets and different geometry types for 100 users. All requests were performed at the same time.

For PostgreSQL (with standard cache size) the response time increases with the size of the dataset. Especially for the large dataset (Germany) the response time reaches 200 seconds for requests even on point objects. In general queries on points can be answered faster than on lines and polygons.

MongoDB behaves differently. The size of the dataset doesn't play a big role. The response time is almost linear, just differs by some seconds. In general queries on points can also be answered faster than on lines and polygons.

Figure 3: Queries with geo-function in PostgreSQL



PostgreSQL has a good performance independently of the geometry type on the small (Niederbayern) dataset. That changes with an increasing amount of data. Whereas MongoDB keeps the performance even with large datasets, PostgreSQL response time rapidly increases with the size of the dataset. But for small datasets PostgreSQL performs better when

considering complex geometry types like lines and multipolygons. The reason for the better performance of MongoDB can't be explained in detail. A reason might be the the 2D index internals of MongoDB. It calculates geohash values for the 2D-index, so only a small amount of data has to be searched before delivering the data. According to the Gist-index an r-tree needs to be searched, which might be slower than searching the geohash values.

5 Conclusion and further work

NoSQL databases are a relatively new technology in the field of geoinformation. There are several different NoSQL-concepts available. The paper pointed out that there is still a lack of geo-functionalities within document-oriented NoSQL-databases. 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.

In direct comparison to the performance test of the two test cases the results show that queries with the use of a geo-functions take longer than queries on attribute-information, which was expectable. For requests purely on attribute information NoSQL-databases are very fast and are superior compared to relational databases.

For requests with geo-functions NoSQL-databases also perform very constant. The measured response times vary only about some seconds for an increasing amount of data. But for small datasets with complex geometry the relational database performed better.

In future work it needs to be investigated how the performance of the NoSQL-databases can be optimized. An optimization can be achieved by:

- Improvement of the indices
- Enhancement of the JSON schema
- General database improvements (i.e. Cache)

MongoDB usually is optimized for a shared setup over several servers. This possibility was not investigated in the tests but may still lead to some performance improvement.

Another important aspect is the investigation of NoSQL-databases as a basis for Geo Web Services. This includes different Web Services like WMS, WFS and WCS.

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.

References

[1] Baas, B; *NoSQL spatial: Neo4j versus PostGIS;* TU Delft, Delft. OTB Research Institute for the Built Environment; http://repository.tudelft.nl/view/ir/uuid%3Aa47d3b8e-650a-4152-a310-366db0773848/; accessed: 08.01.2015; 2012

[2] Butler, H. et. al.; *The GeoJSON Format Specification;* http://geojson.org/geojson-spec.html; accessed: 08.01.2015; 2008

[3] Edlich, S. et. al.; *NoSQL. Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*; Carl Hanser Verlag, München, XIV; 289 S.; ISBN: 978-3-446-42355-8; 2010

[4] Halili, E; *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*; Packt Publishing Limited, Birmingham; ISBN 9781847192950; 2008.

[5] Hecht, R., Jablonski, S; *NoSQL Evaluation. A Use Case Oriented Survey;* In: Proceedings of CSC '11 International Conference on Cloud and Service Computing; ISBN 9781457716355; 2011

[6] Kolb, L; *NoSQL-Datenbanken. Kapitel 1: Einführung;* Universität Leipzig; http://dbs.uni-leipzig.de/file/NoSQL_SS14_01_Intro.pdf; accessed: 27.08.2014; 2014

[7] MongoDB, Inc.; *2dsphere Indexes*; http://docs.mongodb.org/manual/core/2dsphere/; accessed: 27.08.2014; 2011-2015

[8] MongoDB, Inc.; *Geospatial Indexes and Queries*; http://docs.mongodb.org/manual/applications/geospatial-indexes/; accessed: 27.08.2014; 2011-2015

[9] MySQL Scalability; *The state of open source Database Markets: MySQL leads the way;* https://www.scalebase.com/the-state-of-the-open-source-database-market-mysql-leads-the-way/; accessed 14.01.2014; 2014

[10] Ostrovsky, D; Rodenski, Y; *Pro Couchbase Server;* Apress; 2014

[11] PostGIS Development Group; *Postgis Manual*; http://postgis.net/docs/index.html; accessed: 14.01.2014; 2014

[12] Strozzi, C; NoSQL - *A Relational Database Management System;* http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page; accessed: 14.01.2015

[13] Xiao, Z; Liu, Y; *Remote sensing image database based on NOSQL database;* In: 19th International Conference on Geoinformatics; Shanghai, China; S. 1–5; 2011