Polish Cartographical Review
Vol. 51, 2019, no. 4, pp. 167–179
DOI: 10.2478/pcr-2019-0014
Received: 14.10.2019
Accepted: 25.11.2019

MARCIN PIETROŃ
Warsaw University of Technology
Faculty of Geodesy and Cartography
Department of Cartography
orcid.org/0000-0001-7773-8925
e-m: mjj.pietron@gmail.com

Analysis of performance of selected geospatial analyses implemented on the basis of relational and NoSQL databases

Abstract. Databases are a basic component of every GIS system and many geoinformation applications. They also hold a prominent place in the tool kit of any cartographer. Solutions based on the relational model have been the standard for a long time, but there is a new increasingly popular technological trend – solutions based on the NoSQL database which have many advantages in the context of processing of large data sets. This paper compares the performance of selected spatial relational and NoSQL databases executing queries with selected spatial operators. It has been hypothesised that a non-relational solution will prove to be more effective, which was confirmed by the results of the study. The same spatial data set was loaded into PostGIS and MongoDB databases, which ensured standardisation of data for comparison purposes. Then, SQL queries and JavaScript commands were used to perform specific spatial analyses. The parameters necessary to compare the performance were measured at the same time. The study's results have revealed which approach is faster and utilises less computer resources. However, it is difficult to clearly identify which technology is better because of a number of other factors which have to be considered when choosing the right tool.

Keywords: spatial analyses, spatial data, MongoDB, PostGIS, efficiency

1. Introduction

Spatial databases are nowadays the basic source of data for cartographic studies, and the systems for managing these databases are an important tool for cartographic work. Quick analysis of very large data sets is particularly important for cartographers. The study presented in this paper was meant to analyse the impact of relational and non-relational (NoSQL) database models on the performance of spatial analyses. It was difficult to compare these two approaches due to their completely different natures. The precisely defined data structure of the relational model strongly contrasted with the unstructured quality of the NoSQL model. Another important issue was the need to define the criteria which would dictate how the study should be performed.

It is widely believed that the first database was created in the 1960s. The term was first used in that decade, and Charles Bachman developed the first system of database management at that point. An important breakthrough came in 1970 when Edgar Frank Codd presented the concept of a relational database model in the publication titled *A relational model of data for large shared data banks*. Products based on this solution did not appear on the market until ten years later, but it became the standard for many years to come. The theory of relational databases is built upon the mathematical theory of sets in accordance to which data is stored in tables of records which have identical structure and are internally linked by means of specific relations. Relational databases are distinguished by their isolation, data consistency, model normalisation and transaction support (J.D. Ullman, J. Widom 2000). These advantages determined their popularity. Nevertheless, the technical progress and changing market needs gradually made their drawbacks more noticeable. When many users use a relational database at the same time, it no longer works so well, and when many applications have to be serviced, the database becomes very complex. With very large amounts of data, the lack of possibility of horizontal scaling became an issue. It should also be noted that this approach used simple data formats, often different from those used by applications. This hindered the collection and processing of i.a. spatial data. In many scenarios, the relational model proved to be insufficient, but a way of storing a huge amount of unstructured data had to be found. Many IT specialists have tried to adapt old technologies to new challenges, but new approaches have also been developed. On 11 June 2009, in San Francisco, databases based on non-relational models were first presented during a "No SQL Meetup" conference. Since then, NoSQL databases have been steadily gaining popularity. The widespread use of non-relational models was determined by their many advantages, such as theoretically unlimited scalability of memory storage, a guaranteed system response to any request, as well as partition tolerance (M. Fowler, P.J. Sadalage 2015). In June 2019, db-engines.com reported that the top ten most popular databases included four NoSQL solutions. MongoDB, the most popular of them, took the fifth place on the list. This shows that non-relational databases are not merely a curiosity in the IT industry, but an important and functional solution (M. Wyszomirski 2018).

The differences between relational and non-relational technologies in the context of spatial databases have already been researched. S. Agarwal and K.S. Rajan (2017) from the International Institute of Information Technology in Hyderabad compared the speed with which MongoDB and PostGIS systems performed a specific task. However, they limited themselves only to finding a specific type of restaurant in a given area. Elena Baralis, Paolo Garza and Andrea Dalla Valle (2017) from the Polytechnic University of Turin focused on both qualitative and performance differences between Azure SQL Database and Azure DocumentDB. Dany Laksano (2018) studied the loading time of spatial data stored in the MongoDB and PostGIS databases performed with a NodeJS Fullstack web application. The same database management systems were used by Michał Lupa and Adam Piórkowski (2019) who focused on analysing the dependence of the query execution time on the number of objects in the set. In this paper, the above-mentioned considerations were expanded by using larger data sets and more diverse spatial analyses. They also took into consideration the issue of consumption of the CPU processing power during the query execution.

2. Aim and concept of the study, and its research method

Spatial analyses are operations aimed at extracting new information from data which had spatial references (J. Adamczyk, A. Konieczny 2010). They have many applications and are crucial for the work of most cartographers and geographical information systems specialists. Spatial data must be stored in a secure manner that provides adequate access. GIS software makes it possible to create, modify and analyse data stored in databases, and to generate appropriate geovisualizations. The most popular applications of this type support solutions based on the relational model. However, the amount of available spatial data is constantly increasing, and new technologies' capabilities of analysing said data are also growing. Because of this, NoSQL databases are employed with an increasingly frequency. They have almost unlimited scalability, are usually made available under open source licenses, and in the case of aggregate-oriented ones, provide faster and easier access to data.

Non-relational databases have evolved in response to the changing needs. Very often, they were developed on the basis of systems which themselves failed to succeed (G. Harrison 2019). This means that the differences between a given NoSQL database and a relational database are often extremely large, despite the fact that they meet the same or similar goals. Is it possible to say which solution is better? This question is certainly not the best-formulated one, because there are significant differences in quality even among various relational databases. The question should be asked in a more specific way – Is X solution more efficient than Y solution? The answer is extremely important. Higher level of performance efficiency, and thus the ability to complete the same task quicker, can be the key to greater profit. However, there is another issue which has to be taken into account in this context.

Some performance benefits will become irrelevant if a given operation can be carried out faster, but at the same time consumes such enormous amounts of computing power that weaker equipment would not be able to satisfy such demands.

The third important aspect to consider when comparing the two database technologies is the choice of source data. A proper comparison requires all operations to be carried out on a basis of a single data set. Fulfilling this condition is not easy. Structures in relational and non-relational databases differ significantly from one another. Considering the above, it was decided that the study would be done using a NoSQL technology which allowed for storing resources in a way that was most similar to the method employed in a relational model. The data set was transferred by means of an appropriate algorithm from one storage to another. The reversibility of this process was ensured, which in turn guaranteed that there would be an algorithmic relationship between the two sets of data.

3. Preparation of data and computer equipment

A part of the vector Database of Topographic Objects (BDOT10k), depicting the area within the administrative boundaries of the city of Warsaw, was chosen as a set of data for the spatial analyses. The content and the level of detail of BDOT10k correspond to those of 1:10,000 topographic maps. The database's content consists of topographic objects with spatial references, descriptive characteristics, cartographic codes, and metadata. This referential database was implemented in relational-model-based solutions because of its conceptual model. The creators of BDOT10k have adapted it to the standards valid for the field of geographical information. In practice, this meant restrictions and the need to use specific solutions in the process of transformation into a relational model. The data for the Database of Topographic Objects were sourced from the registers kept by state authorities and institutions, as well as the results of field inspections (D. Gotlib 2013).

The BDOT10k data for Warsaw (test area) included 84 classes saved in the XML document format. The total size of the set is about 1.33 GB. It should be noted, however, that because some individual files had no geometry and some were not supported by the GIS application, the study was ultimately based on 71 classes. Indeed, the amount of data was very small. Even relational bases are able to handle much larger volumes. Another special class of objects was created in order to further increase the scale of the analysed resource. One million points were randomly generated within the smallest rectangle limiting the administrative boundaries of the capital city of Warsaw in the QGIS software, with the help of the "Random points in range" function. The so-created class was the most numerous of all the classes used for the tests (it had twice as many objects as all other classes together). This type of data is usually obtained in the course of modelling of a particular phenomenon, but they were not assigned any special significance for the purposes of the study.

The limits of the potential of the relational database and the scale of its non-relational competitor have not been checked due to hardware limitations and the fact that it was impossible to use distributed architecture. However, the obtained sample is sufficient to meet the objectives of the study. The results will allow to formulate relevant conclusions using sufficiently accurate measurement tools.

All operations were performed on a hardware set consisting of: a quad-core Intel Core i5-7600K processor with a base frequency of 3.8 GHz, two DDR4 Ballistix Sport RAM chips, 8 GB in total, a SSD ADATA SU800 120 GB drive and a HDD Toshiba HDWD110 1 TB drive. 64-bit Windows 10 (education version) was the operating system on which the software was installed and the tests were conducted.

4. Software configuration and data import

Many programmes were used to carry out the study for this article. They are presented in table 1.

QGIS and scripts created using Python were the key tools in the process of importing data into PostGIS and MongoDB (fig. 1). The nature of the imported data did not generate the need to create relationships between classes. The data loaded into QGIS consists of tables resulting from combining many inheriting classes. Each of them represented a completely different type

Tab. 1. Software used in research

No.	Software	Version	Description
1.	PostgeSQL with PostGIS extension	PostGIS – 2.5.2. PostgreSQL – 11.2.	PostgreSQL is an object-relational database management system. He uses the SQL language. PostGIS is an extension of PostgreSQL that develops it to a spatial database. It supports geographic features and queries based on their location (PostGIS Development Group, August 2019).
2.	pgAdmin	4.2.	The pgAdmin software is the PostgreSQL database administration platform. This is a very functional graphical interface (pgAdmin Development Team, August 2019).
3.	MongoDB	4.0.6.	MongoDB is an aggregation oriented JSON document database. It is based on JavaScript as the query language. This allows to create shorter and more simple queries than in SQL (MongoDB Inc., September 2019).
4.	QGIS	3.8. Noosa	QGIS is a functional Geographic Information System. The support for the tools of creating plugins and scripts is very important because it additionally expand the capabilities of this software (OSGeo, August 2019).
5.	Python	3.7.	Python is a high-level, interpreted, object-oriented and dynamically typed programming language. (Python Software Foundation, August 2019). It is imortant that two APIs (Application Programming Interface) were used in the research: PyMongo and PyQGIS.

of topographic objects, therefore there were no relationships between them that would have to be modelled.

One of the objectives of this article was to ensure that the process of moving data from a relational database to a non-relational one was reversible. This condition has been met, because it is technically very easy to download a JSON document from MongoDB and import it into PostGIS.

5. Process of performance testing

5.1. Selection of spatial operators and designing spatial queries

Spatial analyses performed directly in DBMS were carried out by SQL queries or lines of JavaScript code. Spatial operators were crucial in this context, as all spatial queries were based on them, because they determined how the user wanted to process the data. Both PostGIS and MongoDB have allowed for the use of implemented operators. Unfortunately, the sets of clauses for these systems were diametrically opposed. PostGIS had dozens of operators which returned spatial relationships, allowed for taking measurements, constructing geometric objects, managing geometry attributes and carrying out many more actions (PostGIS Development Group, August 2019). MongoDB contained only a few operators, the most important of which were $geoIntersects, $geoWithin, $geoNear $near and $nearSphere. The last two are very similar, so the small differences between them were not considered in the course of this study. Operators such as $minDistance, $maxDistance or $geometry were very important for building query syntax and defining its geometric elements (MongoDB Inc., August 2019a).

The need to achieve comparability of analyses carried out in the two systems has limited the number of spatial operators which could be used in the study.

The designed queries were repeated for four object classes (tab. 2) and three administrative units (tab. 3) in order to analyse how the amount of data that the system must search and the

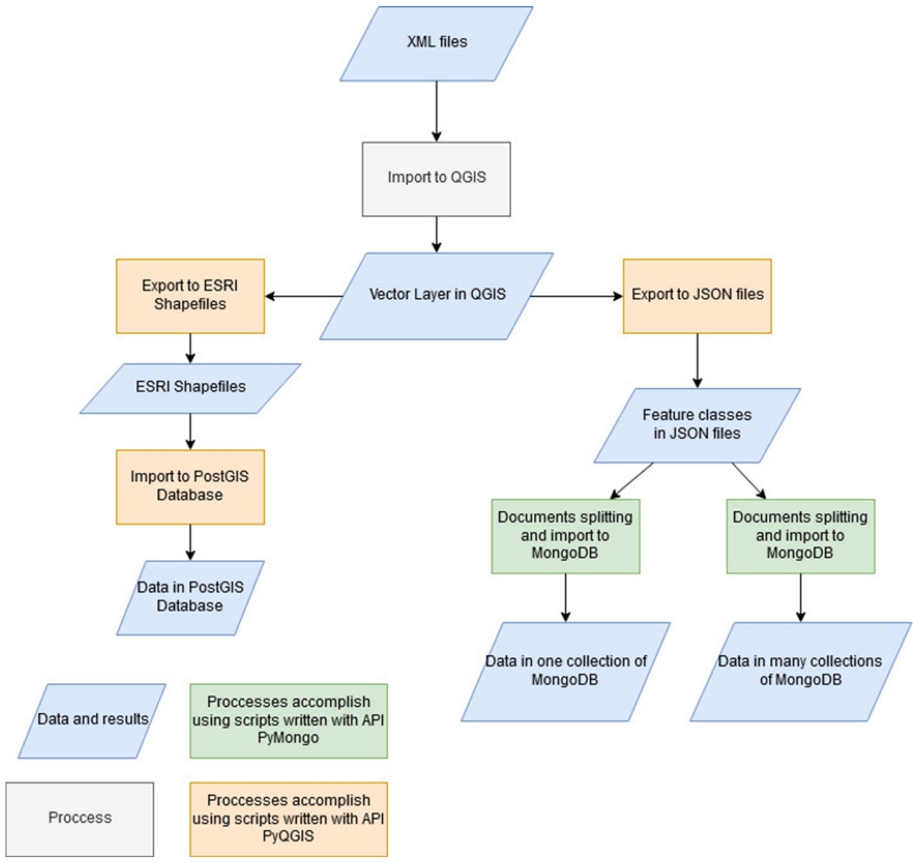
XML files
Import to QGIS
Vector Layer in QGIS
Export to ESRI Shapefiles
Export to JSON files
ESRI Shapefiles
Feature classes in JSON files
Import to PostGIS Database
Documents splitting and import to MongoDB
Documents splitting and import to MongoDB
Data in PostGIS Database
Data in one collection of MongoDB
Data in many collections of MongoDB
Data and results
Proccesses accomplish using scripts written with API PyMongo
Proccess
Proccesses accomplish using scripts written with API PyQGIS

Fig. 1. Block diagram of the process of data import into PostGIS and MongoDB databases

size of the search area impact the performance efficiency.

Both classes and search areas were selected in such a way as to ensure that the differences concerning the number of objects and areas were significant and could illustrate how the increase in these parameters results in increased difficulties in execution of queries. In the case of the $near or $geoNear operators, the "near" range was 1 km or 10 km. The changing fragments are marked in the presented examples (scripts 1 and 2) – red indicates class, and green

Tab. 2. Selected feature classes

Feature class	**Count of objects**	**Geometry type**
Random generated points	1,000,000	point
bubd_a	149,731	Building – area
oipr_p	64,071	Nature object – point
suln_l	6,728	Overhead line – line

Tab. 3. Selected areas of queries

Region	Area in km^2
Ochota	9,713
Wawer	79,641
Warsaw	516,759

Similar analyses were carried out for the variant in which objects belonging to specific classes were placed in separate collections of the MongoDB databse, which required the use of other queries (script 5).

The next set of tests included searching for objects in polygons defined directly in the query (scripts 6 and 7). Only one, most numerous, class of randomly generated points and three polygons determined on the basis of vertex coordinates were used for these analyses. Pre-defined search areas were diversified on the basis of the area size in order to analyse how any increases in range affect performance efficiency (tab. 4). The data stored in the collection, along with the BDOT10k data, were used first at this stage.

Script 1. A PostGIS database query concerning buildings located within or on the border of the Ochota district

```
SELECT bu.*
FROM bdot.bubd_a AS bu, bdot.adja_a AS ad
WHERE ST_INTERSECTS(bu.geom, ad.geom) AND ad.nazwa = 'Ochota'
```

Script 2. A MongoDB database query concerning buildings located within or on the border of the Ochota district

```
var ar1 = db.Warsaw.findOne( { name: "ADJA_A", "features.properties.nazwa":
"Ochota" } )
db.Warsaw.find( { "features.geometry": { $geoIntersects: { $geometry: ar1.
features[0].geometry } }, name: "BUBD_A" } )
```

– the search area. Orange indicates aliases which are also subject to change, but depend directly on the selected table.

The problem of finding all buildings located within one (scripts 3 and 4) or ten kilometres from the selected point and arranging them from the nearest to the furthest was posed in order to observe the performance efficiency of queries constructed on the basis of the $geoNear operator.

Script 3. A PostGIS database query concerning buildings located within 1 km from the selected point, ordered by distance

```
SELECT bu.*
FROM bdot.bubd_a AS bu, (SELECT * FROM bdot.adms_p AS ap WHERE nazwa =
'Warszawa') AS p1
WHERE ST_INTERSECTS(bu.geom, ST_BUFFER(p1.geom, 1000))
ORDER BY ST_Distance(bu.geom, p1.geom)
```

Script 4. A MongoDB database query concerning buildings located within 1 km from the selected point, ordered by distance

```
var ar1 = db.Warsaw.findOne( { name: "ADMS_P", "features.properties.nazwa":
"Warszawa" } )
db.Warsaw.aggregate( [ { $geoNear: { near: ar1.features[0].geometry,
spherical: true, minDistance:0, maxDistance:1000, query: {name: "BUBD_A"},
distanceField: „calcDistance" } } ] )
```

Script 5. A MongoDB database query concerning buildings overlapping with the Ochota district area

```
var ar1 = db.adjaa.findOne( { "features.properties.nazwa": "Ochota" } )
db.bubda.find({„features.geometry": { $geoIntersects: { $geometry: ar1.
features[0].geometry }}})
```

Script 6. A PostGIS database query concerning points located within or on the border of the pre-defined polygon

```
SELECT ran.*
from bdot.rndpoints as ran, (select ST_Transform(ST_Polygon(ST_GeomFromText
('LINESTRING(20.857016 52.331373, 21.113418 52.343302, 21.271051 52.154767,
20.894588 52.124527, 20.857016 52.331373)'), 4326),2180) as p1) as g1
where ST_INTERSECTS(ran.geom, p1)
```

Script 7. A MongoDB database query concerning points located within or on the border of the pre-defined polygon

```
db.Warsaw.find({"features.geometry": { $geoIntersects: { $geometry: { type:
"Polygon", coordinates: [[ [20.857016, 52.331373], [21.113418, 52.343302],
[21.271051, 52.154767], [20.894588, 52.124527], [20.857016, 52.331373] ]] }
}}, name: "random_points"})
```

Tab. 4. Predefined polygons and their areas

	Area in km²
Small polygon	45,642
Medium polygon	179,700
Large polygon	491,216

Queries concerning data stored in a separate collection were also designed. The differences in syntax were very small. The data selection condition after the "name" attribute was not included, because queries were run on collections containing only one class (script 8).

The above-presented examples apply only to the $geoIntersects and ST_INTERSECTS operators. However, more operators were used to carry out more diverse spatial analyses (tab. 5).

Tab. 5. Operators used in the research

Operator in MongoDB	Operator in PostGIS
$geoIntersects	ST_INTERSECTS
$geoWithin	ST_WITHIN
$geoNear	ST_BUFFER+ST_INTESECTS +ST_DISTANCE
$near	ST_BUFFER+ST_INTESECTS

The final stage of testing consisted in running queries on all classes at the same time. All objects in contact with the defined point, located within the Ochota district, and within 100 metres from another specified point, were identified. The properties of a non-relational database have proved to be very valuable. The ability to

Script 8. A MongoDB database query concerning points located within or on the border of the pre-defined polygon

```
db.rndpoints.find({"features.geometry": { $geoIntersects: { $geometry:
{ type: "Polygon", coordinates: [[ [20.857016, 52.331373], [21.113418,
52.343302], [21.271051, 52.154767], [20.894588, 52.124527], [20.857016,
52.331373] ]] } }}})
```

store all data in one collection made it possible to avoid a very long query syntax.

5.2. Method of measuring performance parameters

The assumptions of this paper identified two basic parameters which would allow for determining performance efficiency. Quick task performance is a key issue for most users, so the study focused on the query execution time. It was measured on the basis of functionalities implemented directly in DBMS. The pgAdmin application displayed the execution time at the end of the query. In the case of MongoDB, this information was saved in server logs, i.e. files which recorded subsequent activities.

Windows 10 task manager was used as a CPU usage monitoring tool. All unnecessary processes were turned off during the execution of the query in order to achieve the maximum possible measurement accuracy. The query was run several times to make sure that other programmes were not adversely affecting the results. Values of this parameter were recorded as a percentage of total CPU usage on all cores.

The default DBMS MongoDB settings were meant to record only those results which exceeded one second. The queries conducted for this study ended much earlier. The time limit threshold had to therefore be lowered to one millisecond (MongoDB Inc., August 2019b).

Measuring the CPU processing power consumption was a big problem. It was impossible to find a tool to register this type of load generated by database queries. Most of potential solutions were designed for Linux systems or were not released under open-source licenses. Only few free options allowed to measure the CPU consumption caused by queries made only in the PostGIS system. Ultimately, relatively imprecise task manager of the Windows operating system was used. The fact that the consumption of processing power differed a lot depending on the amount of data and DBMS made it possible to notice discrepancies in the operation of selected systems. However, one should remember that this solution provides results burdened with a sizeable error.

6. Results

Visualizations presented below concern only the queries containing the operator searching for the common part of objects, i.e. $geoIntersects or ST_INTERSECTS, and carried out for the data contained in one collection of the MongoDB database, due to the very large number of sets and charts resulting from numerous analyses.

The results revealed that the MongoDB database had a clear advantage in terms of

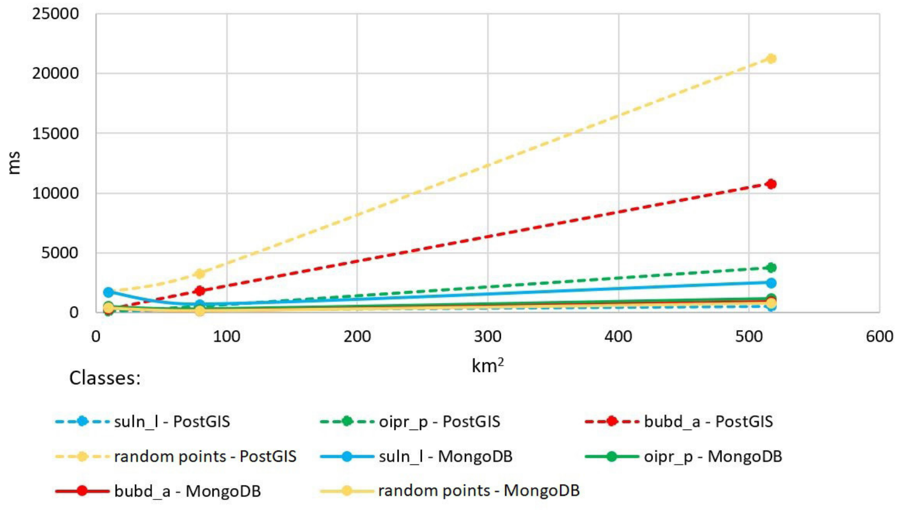

Fig. 2. The query execution time in dependence of the size of the search area for the Intersects operator

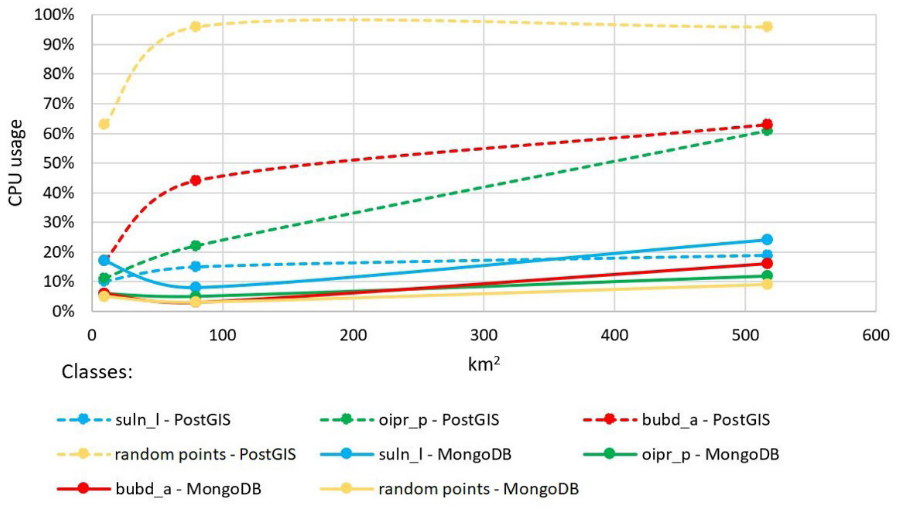

Fig. 3. The CPU usage of query in dependence of the size of the search area for the Intersects operator

performance efficiency. As the search area increased, so did the query execution time, which was expected. The algorithm had to analyse a larger area and find more objects (fig. 2). It should be noted that the time difference between the analysis of data stored in a relational and a non-relational database was very large. A notable exception was the "suln_l" overhead line class, because the query run on the MongoDB database was executed faster for the Wawer district than for Ochota.

Resource utilisation had different distribution than time (fig. 3). In the case of the PostGIS system, an increase of the search area was linked to an increasing demand for CPU power. The differences were very significant for queries concerning data with location in Ochota and Wawer. However, the trend stabilised later.

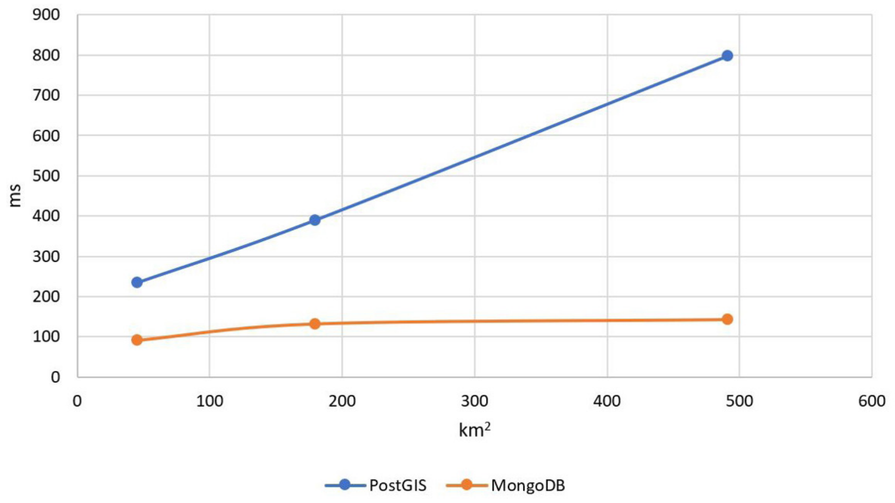

Fig. 4. The query execution time in dependence of the size of the search area for the Intersects operator

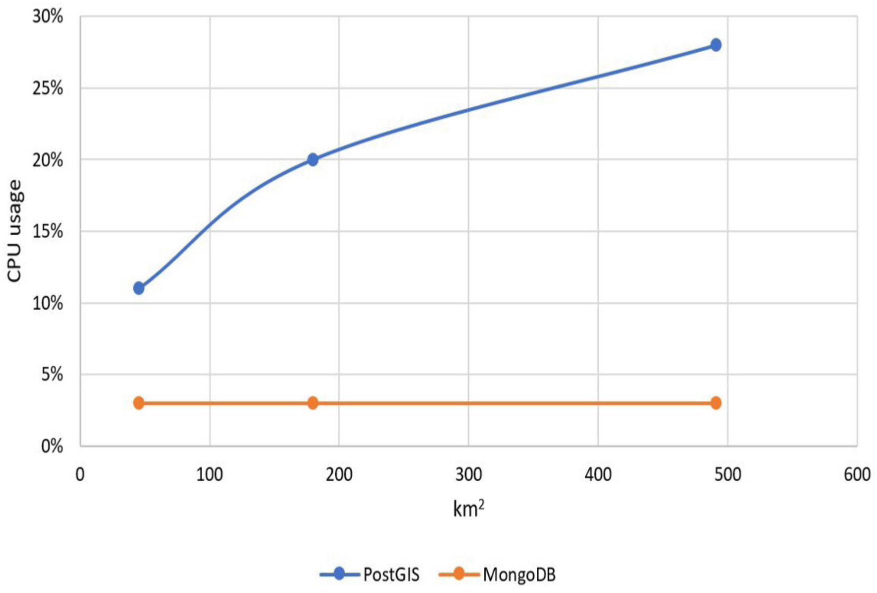

Fig. 5. The CPU usage of query in dependence of the size of the search area for the Intersects operator

The exception were the results of the queries concerning the class of "oior_p" natural objects, where the increase was similar to a linear function. The MongoDB system outclassed its competitor in this aspect as well. There was even a decrease in the demand for processing power for the data queries within the Wawer district.

In the case of the within operator, the distribution of time and consumption of the CPU processing power, reflecting the change in the size of the search area, were similar to those presented above. Using the "near" operator ($geoNear) radically changed the results, but the MongoDB database still retained its overwhelming advantage in every aspect. Queries run on a database storing data in separate collections were even more efficient.

Spatial analyses taking into account the definition of the search area in the query syntax were made only with the help of "intersect" and "within" operators. It was not possible to include the $geoNear clause in the context of this part of the study due to its different mode of opera-

Tab. 6. Summary of CPU percentage usage and queries execution time for all classes

OPERATOR	POSTGIS			MONGO		
	Time [msec]	RESULT	CPU usage	Time [msec]	Result	CPU usage
$geoIntersects /ST_INTERSECT	217	10	15%	32	12	4%
$geoWithin /ST_WITHIN	2,142	24,085	64%	118	24,085	3%
$near /ST_BUFFER +ST_INTESECTS	227	82	14%	62	139	4%

tion. The tests included queries concerning one, most numerous class, which in the case of the MongoDB system was stored in a separate collection.

The results of measuring the time and the consumption of the processing power were very clear (figs. 4 and 5). Better performance of the MongoDB system was clearly noticeable. The time needed to execute queries increased with the increase in the area of pre-defined polygons. This increase was definitely faster in the case of the PostGIS system. A similar situation occurred for the CPU processing power consumption parameter. Once again, the results obtained for the within operator were very similar, and querying the database for data contained in separate collections further increased the performance efficiency.

The results of the query performance tests for all database classes were very clear once again. The MongoDB system performed its operations faster, and also consumed less CPU resources (tab. 6). It is worth noting that the queries concerning near locations provided completely different results, which may confirm either differences in functioning of these operators, or differences in the way the location data was saved. A similar situation occurred in the case of the “intersects” clause.

Building a query in PostGIS that could finding objects from each class required a very long code. A specific set of operations had to be performed for each class and, afterwards, the results had to be aggregated using the “UNION” clause. It was necessary to ensure the consistency of the result table, which meant a restrictive selection of attributes, leaving only “id”. This problem did not affect the MongoDB system. Storage of all data in one collection allowed for writing a very short query. This is undoubtedly an advantage of aggregate-oriented databases.

7. Problems, difficulties and further research development areas

The first difficulty was related to storing spatial data in the MongoDB database. QGIS software provides very good support for the PostgreSQL system, which made it possible to avoid major problems during the transfer. Importing data into a non-relational database was much more difficult. It required mastering Python programming language, at least at a basic level, as well as learning its libraries and APIs that allow for such operations. It should be noted that implementation of the designed scripts was very time-consuming, and the import of all data could take up to several hours.

Maintaining a uniform reference system also proved to be a challenge. The input data contained a reference to the national geodetic coordinate system (PUWG) “1992”. They have been imported in the unchanged configuration into the PostGIS database. The MongoDB system had some limitations in this respect. It was necessary to set up an index for the attribute containing information about the geometry. The default “2dsphere” was selected in this case, which meant that the queries returned an error if the data was in a different system than WGS-1984. It was therefore necessary to transform the coordinates.

This paper focuses on performance of queries run on databases limited to one local server. Modern commercial systems use solutions based on distributed architecture. Technical limitations prevented performance testing of this database variant.

MongoDB is not the only non-relational database. Similarly, PostgrSQL has many relational competitors. It would be worthwhile to expand the research to compare the possibilities offered by other non-relational technologies, e.g. Cassandra which is a column family type database, based on the key-value model, such as Redis or Neo4j, a graph database. In turn, SQL Server, Oracle, MySQL and SQLite are some of examples of other popular relational solutions.

Only minimal attention was paid to the aggregative nature of the MongoDB database in the course of the study. This potential was used in the context of queries concerning data from all classes at once, but its full scope is much larger. Future research could take this aspect into consideration. In particular, it is worth considering the possibilities of storing complex spatial data describing very complicated phenomena.

Definition of spatial operators is a very important issue that must be taken into account in the possible further course of research. In the case of the PostGIS system, the results differed from those obtained in MongoDB. The disparity was probably caused by different approaches to spatial relations checked by operators. Different level of precision in saving coordinates in databases could be another contributing factor. It was a crucial issue because it not only determined the possibility of comparing the performance of various queries, but also significantly affects users work.

8. Summary and conclusions

Many of the issues presented in the study are very important for modern cartographers and GIS analysts. The amount of available spatial data is growing extremely quickly and creates the need to use modern technologies for processing data in such large quantities. Although the relational model dominates in GIS, non-relational databases are gradually becoming more common and gaining popularity.

The obtained results clearly indicated that the analyses carried out in the MongoDB system were performed with greater efficiency, which confirmed the hypothesis of the study. MongoDB not only performed queries much faster, but also consumed less of CPU processing power. The fact that it was an aggregate-orientated database was also important and allowed for construction of much simpler, more concise queries. It had also a significant impact on performance itself. The query of a properly aggregated data set could be executed up to several dozen times quicker than of an analogous query in a relational database.

It should be emphasised that PostGIS allows for the use of an incomparably larger number of spatial operators. This is clearly crucial in the context of GIS analyses. In practice, most spatial data processing operations are carried out in dedicated software, such as QGIS. Therefore, the number of operators that can be used in a given DBMS is less important. As a rule, GIS software supports relational databases for much longer and in a much better way, and non-relational solutions are usually not supported not all.

Another significant limitation of the MongoDB system are its restrictive requirements for the coordinate system. This is related to the issue of spatial indexes. The index selected for this study required the data to be in the WGS-1984 system. PostGIS does not impose such restrictions.

Visualization of analysis results is particularly important in the context of GIS and cartography. The pgAdmin graphical interface allowed for the display of spatial query results. However, the so-obtained presentations were of poor quality and could not be edited, as they were meant to provide only very simplified information. However, the MongoDB database management system did not allow for creating even the simplest visualizations. It should be noted that no graphical interface was used in this case. All operations were carried out in the system console.

The non-relational MongoDB database is definitely more efficient than the relational PostGIS. This does not mean, however, that this solution is better, especially in the context of spatial data analysis, because currently support for processing of location data is poor. However, this technology can continue to develop. The PostGIS extension provides many more possibilities in the fields of reference systems, spatial operators and cooperation with leading GIS applications. Each of the examined database management systems can store and process location data. They have different advantages, but also significant drawbacks. When choosing a specific technology, one should carefully analyse the needs of a given project and specificity of the tasks involved. Relational databases are the current standard and in some areas of GIS they will certainly remain the default option. However, non-relational technologies are an attractive alternative, and in the case of very large volumes of complex, unstructured collections, they may even be a necessity.

Acknowledgements

I would like to offer my sincere thanks to my research supervisor, Professor of the Warsaw University of Technology Robert Olszewski, PhD Eng., Miłosz Gnat, MSc Eng., and Michał Wyszomirski, PhD Eng., for consultations and substantive assistance.

Literature

Adamczyk J., Konieczny A., 2010, *Rodzaje analiz przestrzennych*. In: K. Okła, *Geomatyka w Lasach Państwowych – Część I. Podstawy*. Warszawa: CILP, 214 pp.

Agarwal S., Rajan K.S., 2017, *Analyzing the performance of NoSQL vs. SQLdatabases for spatial and aggregate queries.* "Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings" Vol. 17, Article 4, Boston.

Baralis E., Garza P., Valle A.D., 2017, *SQL versus NoSQL Databases for Geospatial Applications.* In: *IEEE International Conference on Big Data*, Boston, IEEE, pp. 3388–3397.

Fowler M., Sadalage P.J., 2015, *NoSQL. Kompedium wiedzy.* Polish transl. J. Hubisz, Gliwice: Helion, pp. 19−36.

Gotlib D., 2013, *Ogólna koncepcja, cel budowy i zakres informacyjny BDOT10k i BDOO*. In: R. Olszewski, D. Gotlib (eds.), *Rola bazy danych obiektów topograficznych w tworzeniu infrastruktury informacji przestrzennej w Polsce*. Warszawa: GUGiK, pp. 51−57.

Harrison G., 2019, *NoSQL, NewSQL i BigData*. Polish transl. P. Pilch, Gliwice: Helion, pp. 29−32, 58−59, 70−77.

Laksano D., 2018, *Testing spatial data deliverance in SQL and NoSQL database using NodeJS Fullstack Web App*. *4th International Conference on Science and Technology (ICST)*, Yogyakarta, IEEE.

Lupa M., Piórkowski A., 2019, *The comparison of processing efficiency of spatial data for PostGIS and MongoDB Databases*. In: S. Kozielski et al., *Beyond databases, architectures and structures. Paving the road to smart data processing and analysis*. 15th International Conference, BDAS 2019, Ustroń: Springer International Publishing, pp. 291–302.

Ullman J.D., Widom J., 2000, *Podstawowy wykład z systemów baz danych*. Polish transl. M. Jurkiewicz, Warszawa: Wydawnictwo Naukowo-Techniczne, pp. 19−33, 118−122, 168−197.

Wyszomirski M., 2018, *Przegląd możliwości zastosowania wybranych baz danych NoSQL do zarządzania danymi przestrzennymi.* "Roczniki Geomatyki" T. 16, z. 1(80), pp. 55−69.

Internet sources

PostGIS Development Group, *PostGIS 2.5.4.dev Manual*. In: PostGIS Documentation [online], https://postgis.net/docs/manual-2.5/ (access August 2019).

pgAdmin Development Team, *Features*. In: pgAdmin homepage [online], https://www.pgadmin.org/features/ (access August 2019).

MongoDB Inc., *GeoJSON Objects*. In: MongoDB Documentation [online], https://docs.mongodb.com/manual/reference/geojson/ (access September 2019).

OSGeo, *QGIS – wiodący otwartoźródłowy system GIS.* In: Official QGIS project website, https://qgis.org/pl/site/about/index.html (access August 2019).

Python Software Foundation, *General Python FAQ*, 2010. In: Python Documentation, https://docs.python.org/3/faq/general.html (access August 2019).

MongoDB Inc., *Geospatial Queries.* In: MongoDB Documentation [online], https://docs.mongodb.com/manual/geospatial-queries/ (access August 2019a).

PostGIS Development Group, *PostGIS Reference*. In: PostGIS Documentation [online], https://postgis.net/docs/reference.html (access August 2019).

MongoDB Inc., *Database Profiler.* In: MongoDB Documentation [online], https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/ (access August 2019b).