The Comparison of Processing Efficiency of Spatial Data for PostGIS and MongoDB Databases

Dominik Bartoszewski, Adam Piorkowski[2][0000−0003−4773−5322], and Michal Lupa[1][0000−0002−4870−0298]

[1] Department of Geoinformatics and Applied Computer Science
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Cracow, Poland
mlupa@agh.edu.pl
[2] Department of Biocybernetics and Biomedical Engineering
AGH University of Science and Technology,
A. Mickiewicza 30 Av., 30–059 Cracow, Poland

Abstract. This paper presents the issue of geographic data storage in NoSQL databases. The authors present the performance investigation of the non-relational database MongoDB with its built-in spatial functions in relation to the PostgreSQL database with a PostGIS spatial extension. As part of the tests, the authors designed queries simulating common problems in the processing of point data. In addition, the main advantages and disadvantages of NoSQL databases are presented in the context of the ability to manipulate spatial data[3]

Keywords: spatial databases · NoSQL · GIS · PostGIS · MongoDB · query preformance.

1 Introduction

According to IBM estimates, 90% of the world's data has been created in the last two years. Other forecasts say that in 2025 there will be 175 ZB of data in the world, which means an increase from 33 ZB in 2018. This data is big data. This increase also applies to spatial data, which have particularly gained importance due to mobile devices equipped with geolocations [28, 32, 18], constellations of various geostationary and non-geostationary satellites [27, 23, 25, 20] Volunteer

[3] This is the manuscript of:

Bartoszewski D., Piorkowski A., Lupa M.: The Comparison of Processing Efficiency of Spatial Data for PostGIS and MongoDB Databases. BDAS 2019. Springer, CCIS, vol. 1018, 2019, pp 291–302.

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-19093-4_22 .

Geographic Information [9], or finally Global Positioning System [14]. It should also be added that spatial data have a decidedly different character than alphanumeric information, hence the increasing amount of this type of data forces the use of a dedicated approach to handle it [6, 24, 19, 31, 12, 26, 17, 11].

The processing of spatial data has been discussed in the literature since the beginnings of GIS [22]. Analyzing the solutions available in the literature, it is worth paying attention to two issues. First, the strategies for optimizing geospatial queries are very different from the classic optimization methods [7, 10]. The second issue is that all classical methods of geospatial queries are tested on data whose size is significantly different from the size considered in the context of Big Data.

The problem of large data amount is inseparably connected with the NoSQL databases [30, 13, 16]. As recent years have shown, NoSQL is a great alternative for RDBMS in the context of web applications based on large data sets [15, 8]. However, it should be borne in mind that the price for this is the lack of fulfillment of assumptions ACID. In the field of NoSQL databases and spatial data, it is worth quoting the authors' research [33], where MongoDB capabilities were tested for processing data from shapefiles. The article [21] presents the method of indexing spatial data in document based NoSQL.

Analyzing the above issues, the question arises: what strategy do we have to adopt when there is a need to process a dozen TB of GPS logs saved in the form of PointGeometry? Is the replacement of RDBMS by NoSQL crucial in this case? The natural answer to this question seems to be transferring the entire database to the NoSQL. Nevertheless, as mentioned earlier, spatial data and the way they are processed require a completely different approach. Moreover, as shown in [29], even well-known RDBMS systems have problems with the implementation of geospatial functions. Therefore, in addition to the profits related to queries speed, there also should be examined the available geospatial functions offered by NoSQL systems.

In this work we are clearing the abovementioned issues. First of all, the performance of RDBMS and NoSQL for classic spatial queries, which are based on point and polygon data, were compared. What is more, the functionalities offered by the most popular free RDBMS and NoSQL systems were also verified.

2 Test environment

According to the DB-Engines ranking, the most popular open source and non-relational data base for handling spatial information is the MongoDB document database [1]. It is written in C ++ and is licensed under the GNU AGPL open license [5]. It uses objects in the GeoJSON format to store spatial data. MongoDB supports also geospatial indexes as (2D indexes) and spherical indexes (2D Sphere). The RDBMS that has been selected for research purposes is PostgreSQL with the PostGIS spatial extension. The PostgreSQL database is a popular object-relational database management system (ORDBMS).

The PostGIS extension provides more than a thousand geospatial functions and contains all of the 2D and 3D spatial data types. Compared to PostGIS, MongoDB has only three geospatial functions: $geoWithin$, $geoIntersects$ and $nearSphere$. The function $geoWithin$ corresponds to the function S_Within in PostGIS, where the $geoIntersects$ is related to the $ST\_Intersection$. The $nearSphere$ function, combined with the $maxDistance$ parameter, returns all of the geometries at a certain distance sorted by distance. PostGIS is able to perform an analogous operation using the $ST\_DWithin$ function (Tab. 1).

PostGIS	MongoDB
ST_Within	$\$geoWithin$
$ST\_Intersection$	$\$geoIntersects$
$ST\_DWithin$	$\$nearSphere + \$maxDistance$

Table 1. Geospatial functions in the PostGIS and MongoDB databases.

PostGIS has a huge variety of geospatial functions. A full list and description of these functions can be found in the PostGIS documentation [3].

3 Experiments

The document database MongoDB v3.6.3 and thePostgreSQL version 10.1 were selected for the experiments. The queries were counted on a computer with the following specification:

- Intel Core i7 2600 3,4 GHz,
- 4 GB RAM,
- 1000 GB HDD,
- Windows 7 Professional.

3.1 The experiments methodology

For both database systems, the authors were prepared scripts that performed the appropriate commands after running. The purpose of the tests was to measure the time of performing the queries. During the tests, all the processes requiring high computing power and background systems tasks were excluded. Test scripts were two .bat files - DOS/Windows shell scripts executed by the command interpreter (cmd). The first of them was used to test the PostgreSQL performance, while the second one was used to test the performance of the MongoDB database. The performance tests of both databases were done in the following way:

- each query has been repeated 10 times,
- the average and standard deviation were calculated for each query,
- the result of each script has been saved to a text file.

3.2 Query efficiency tests

For the tests purposes were used the basic elements provided with a database system. In the software downloaded with the system, you can find the pgbench application, which provides the following possibilities:

- repeating the query a specified number of times,
- calculation of the average and standard deviation from the query times,
- counting the number of queries performed per unit of time,
- calculation of the query execution time,
- using the specified number of threads to process the query.

3.3 The MongoDB experiments

In the MongoDB database, the explain() method is used to check the performance of the queries. It provides a lot of information about the query. The explain() method can be called with or without a parameter. In the case of the second option, we have three parameters available the "queryPlanner", the "executionStats" and the "allPlansExecution". They determine the level of detail of the displayed information. The default mode is the "queryPlanner" [2].

3.4 The comparison of the query times

Test 1 The first test was to check whether the points are within a certain polygon feature. The coordinates of the points were determined randomly, according to the uniform distribution. The tests included collections that contained 1000, 5000, 10000, 50000, 100000, 500000 or 1,000,000 points. The code for both queries is presented in the Table 2. The visualization of the query is shown on Figure 1. Query times were collected in Table 3.

MongoDB	`var alaska = db.alaska.findOne();`
	`var statsOt = db.otpoints.find({`
	`geometry:{$geoWithin: {$geometry: alaska.geometry }}}).`
	`explain`
	`("executionStats");`

PostGIS	`SELECT * FROM otpoints,alaskasimple` `WHERE ST_Within (otpoints.geom, alaskasimple.geom);`

Table 2. Query #1 - selecting the points contained in the previously selected polygon.

This case gives a clear advantage to the NoSQL database. In MongoDB, the same queries run on average 3x faster than in PostGIS. The standard deviation is very low.

	MongoDB		PostGIS	
Point number	Time [ms]	Std. dev.	Time [ms]	Std. dev.
TEST 1				
1 000	**51.41**	0.51	112.21	0.51
5 000	**196.77**	1.33	496.53	2.58
10 000	**396.23**	2.29	987.61	3.75
50 000	**1855.42**	8.14	5282.78	29.58
100 000	**3793.71**	17.60	11895.12	55.90
500 000	**18520.42**	233.80	61195.34	257.01
1 000 000	**38128.00**	192.35	131895.52	725.42
TEST 4				
5 000	**52.27**	0.63	304.66	5.35
10 000	**103.71**	0.94	612.33	32.44
50 000	**515.84**	1.87	3017.80	24.14
100 000	**1037.41**	3.50	6022.55	72.56
500 000	**5166.74**	14.62	30525.22	770.25

Table 3. The comparison of the test 1 queries.

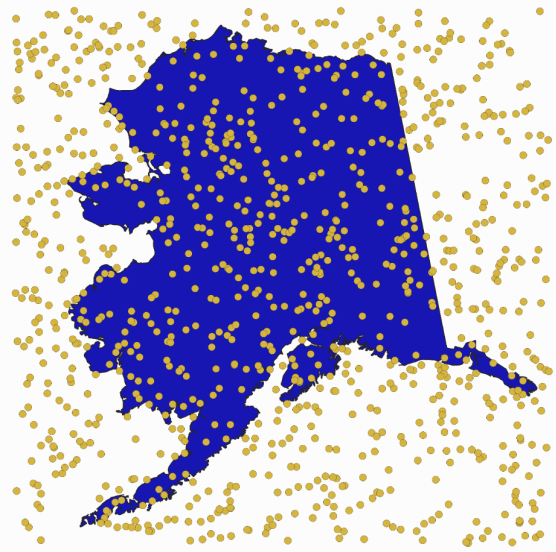

Fig. 1. Visualization of the used polygon and randomly drawn points using the QGIS [4]. The picture shows a case with a thousand points. The query will return only the points inside the blue polygon.

Test 2 The second test was to the points located within a given distance from the selected coordinates. The collections included 50,000, 100,000, 500,000, 1,000,000 points. The authors were determined four test scenarios:

- points within a maximum distance of 100 km from centroid,
- points within a maximum distance of 200 km from centroid,
- points within a maximum distance of 500 km from centroid,
- points within a maximum distance of 1000 km from centroid.

The visualization of the query is shown on Figure 2, the code of queries in the Table 4. Query times are collected in the Table 5.

MongoDB	`var centroid = db.centroid.findOne();`
	`var statsOt = db.otpoints.find({`
	`geometry : { $nearSphere:`
	`$geometry: centroid.geometry, $maxDistance: 100000} } }).`
	`explain`
	`("executionStats");`

PostGIS	`SELECT * FROM otpoints WHERE ST_DWithin`
	`(otpoints.geom, ST_GeographyFromText`
	`('SRID=4326; POINT(-153.138 64.731)'), 100000);`

Table 4. Query #2 - selecting points within a certain distance from point.

The test performed faster in MongoDB, taking into account the radius of 100, 200 and 500 kilometers. In the case of the 1000 kilometers circle radius, the PostGIS proved to be about 3 times faster. We observed very clearly that the query time in the MongoDB database grows much faster with more and more points to count. This situation presents completely different conclusions in relation to the theoretical use of the NoSQL databases, which are tailored to the processing of large amounts of data. The standard deviation is very low for both database systems.

Test 3 The third query is based on a very similar scheme to the previous ones, but now random polygons are examined instead of random points - each of them is a square about 50 kilometers. The sets of 5000, 10000, 50000, 100000 and 500000 polygons were taken into account. The authors were determined four test scenarios:

- polygons within a maximum distance of 100 km from centroid,
- polygons within a maximum distance of 200 km from centroid,
- polygons within a maximum distance of 500 km from centroid,
- polygons within a maximum distance of 1000 km from centroid.

The visualization of the query is shown on Figure 3, the code of queries in the Table 6. Query times are collected in the Table 5.

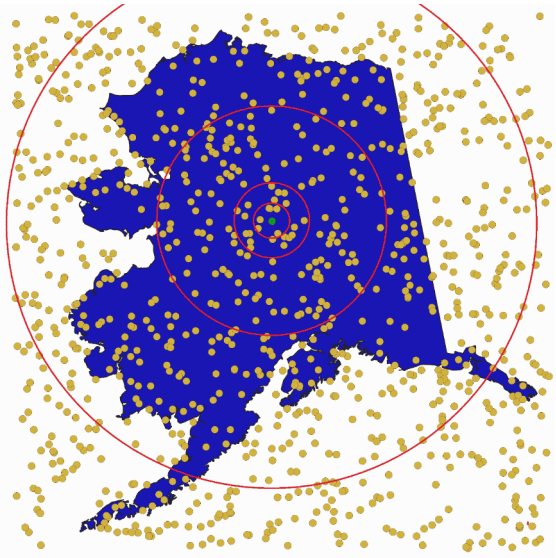

Fig. 2. Visualization of the query using the QGIS [29]. The first circle represents 100km, the second 200km, the third 500km, and the last 1000km from the central point. Visible case concerns a thousand points.

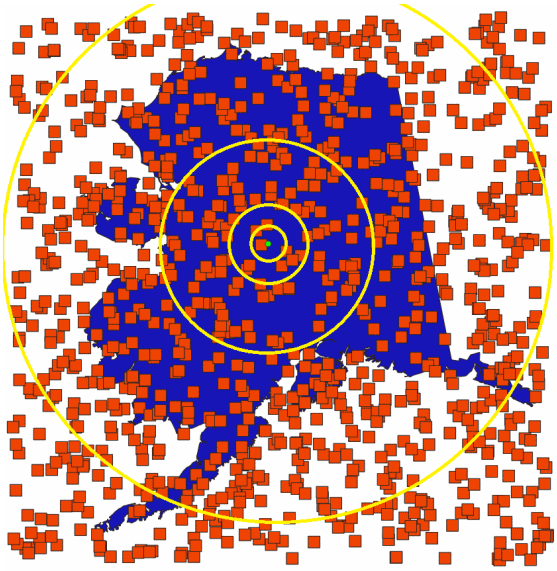

Fig. 3. Visualization of the query using the QGIS [29]. The first circle represents 100km, the second 200km, the third 500km, and the last 1000km from the central point. Visible concerns a thousand polygons.

	MongoDB		PostGIS	
Circle radius/Num. of points	Query time [ms]	Std. dev.	Query time [ms]	Std. dev.
TEST 2				
100 km				
50 000	**5.11**	0	12.17	0.03
100 000	**9.94**	0	53.97	0.04
500 000	**47.12**	0.31	104.75	0.21
1 000 000	**93.03**	0.52	500.43	0.67
200 km				
50 000	**4.20**	0.32	13.24	1.36
100 000	**18.77**	0.42	62.30	1.33
500 000	**37.54**	0.48	124.28	1.42
1 000 000	**194.56**	1.05	643.14	1.44
500 km				
50 000	**22.21**	0	29.38	1.34
100 000	**107.44**	0.37	142.81	1.43
500 000	**227.17**	0.43	285.86	1.37
1 000 000	**1208.36**	1.50	1424.83	2.69
1000 km				
50 000	320.21	0.42	**56.37**	0.18
100 000	677.45	0.51	**275.75**	0.32
500 000	3856.75	4.05	**554.95**	1.51
1 000 000	8489.68	11.95	**2711.01**	2.27
TEST 3				
100 km				
5000	**5.13**	0.14	29.48	0.62
10000	**8.81**	0.21	49.46	5.14
50000	**44.17**	0.91	250.95	11.31
100000	**86.12**	2.11	495.29	18.66
500000	**708.45**	18.24	2535.06	108.24
200 km				
5000	**15.37**	0.67	46.37	0.94
10000	**29.51**	1.26	85.30	1.56
50000	**163.15**	4.40	362.23	7.34
100000	**354.37**	7.63	737.35	14.50
500000	**2531.62**	45.34	3629.29	72.44
500 km				
5000	**93.88**	1.81	104.03	2.50
10000	**184.11**	7.13	209.69	4.18
50000	**979.57**	23.14	1029.64	20.36
100000	2136.14	64.69	**2094.75**	51.15
500000	12717.45	523.85	**10329.26**	210.67
1000 km				
5000	248.67	3.54	**104.03**	3.22
10000	516.77	9.21	**209.69**	8.66
50000	2862.18	40.52	**1029.64**	108.51
100000	6041.14	112.47	**2094.75**	210.32
500000	36434.61	727.81	**10329.26**	410.22

Table 5. Test 3: query execution times comparison.

MongoDB	`var centroid = db.centroid.findOne();`
	`var statsOt = db.otpolygons.find({`
	`geometry : { $nearSphere:`
	`{ $geometry: centroid.geometry, $maxDistance: 100000} } }).`
	`explain`
	`("executionStats");`

PostGIS	`SELECT * FROM otpolygons WHERE ST_DWithin`
	`(otpolygons.geom, ST_GeographyFromText`
	`('SRID=4326; POINT(-153.138 64.731)'), 100000);`

Table 6. Query #3 - selecting polygons within a set distance from point.

An analogous situation as in the previous test with points. When the number of polygons is smaller, MongoDB has the advantage. However, when there is a larger radius of a circle, MongoDB begins to clearly slow down.

Test 4 The fourth and last test case is a compound query. First, in the inner query, the polygon closest to the centroid is selected, then it is investigated whether the returned polygon does not intersect with other polygons. The visualization of the query is shown on Figure 4, the code of queries in the Table 7. Query times are collected in the Table 3.

MongoDB	`var centroid = db.centroid.findOne();`
	`var statsOt = db.otpolygons.find({ geometry : {`
	`$geoIntersects : { $geometry : db.otpolygons.findOne( {`
	`geometry : { $geoNear : { $geometry : db.centroid.findOne`
	`()`
	`.geometry, $maxDistance: 2500 } } } ).geometry } } } ).`
	`explain("executionStats");`

PostGIS	`SELECT ST_Intersection`
	`(otpolygons.geom, (SELECT otpolygons.geom FROM otpolygons`
	`WHERE ST_DWithin`
	`(otpolygons.geom, ST_GeographyFromText`
	`('SRID=4326; POINT(-153.138 64.731)'), 2500)`
	`LIMIT 1)) FROM otpolygons;`

Table 7. Query #4 - Compound query: selecting polygon closest to the certain point and next checking if the polygon is intersecting with other polygons.

The tests gave results similar to those in test 1 - a clear advantage in favor of MongoDB. The NoSQL queries take about 6 times shorter. The standard deviation is too low to be visible on the graph (except for one result in PostGIS for the 500,000 points).

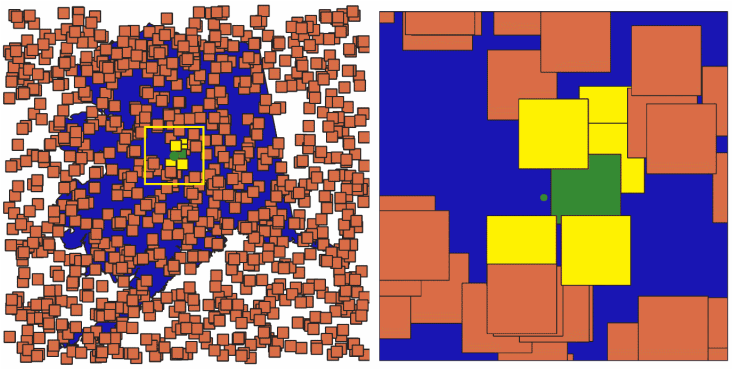

Fig. 4. Visualization of the query using the QGIS [29]. The green polygon represents the result of the internal query (search polygon closest to centroid - here marked with a green point), yellow polygons represent the result of an external query (searching for the intersecting polygons).

4 Conclusions

NoSQL databases are a relatively new technology in the context of spatial data processing. There are only a few such systems available that provide this kind of data. This paper shows that the mechanisms for handling spatial data, compared to relational systems, are much more limited. Support for geographical features includes only the most basic functionalities. Performance tests show that queries concerning within and intersection operations take less time in MongoDB, as opposed to operations to select objects in the neighbourhood of another object, where PostGIS has a big advantage.

This shows that many criteria should be used to work with geographic data. One of them is to determine the operations that will have to be done using the database. In the case of many different spatial analyzes, the most obvious choice will be to use RDBMS, where there are several hundred geospatial functions. Much smaller possibilities in the area of geospatial functions are provided by the non-relational databases, where in the case of MongoDB (which seems to best support the spatial data), there are only a few of these functions. Relational databases also have the advantage of many years of presence on the market, which led to their enormous popularity and the presence of many qualified professionals familiar with this subject.

Non-relational databases can be an alternative when working in dispersed environments that process a huge amount of data simultaneously. What is more, it should be noted that non-relational systems are constantly evolving, which in the future will probably result in an increase in the number of available geospatial functions. This may cause non-relational bases to take over part of the spatial data market.

Acknowledgement. This work was financed by the AGH - University of Science and Technology, Faculty of Geology, Geophysics and Environmental Protection as a part of a statutory project.

References

1. Db-engines ranking, `https://db-engines.com/en/ranking/`
2. MongoDB Docs - geospatial query operators, `https://docs.mongodb.com/manual/reference/operator/query-geospatial/`
3. PostGIS 2.5.2 dev manual, `https://postgis.net/docs/`
4. QGIS documentation, `https://qgis.org/en/docs/`
5. Agarwal, S., Rajan, K.: Performance analysis of mongodb versus postgis/postgresql databases for line intersection and point containment spatial queries. Spatial Information Research **24**(6), 671–677 (2016)
6. Akulakrishna, P.K., Lakshmi, J., Nandy, S.: Efficient storage of big-data for real-time gps applications. In: Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on. pp. 1–8. IEEE (2014)
7. Bajerski, P., Kozielski, S.: Computational model for efficient processing of geofield queries. In: Man-Machine Interactions, pp. 573–583. Springer (2009)
8. Burzańska, M., Wiśniewski, P.: How poor is the poor mans search engine? In: International Conference: Beyond Databases, Architectures and Structures. pp. 294–305. Springer (2018)
9. Chmielewski, S., Samulowska, M., Lupa, M., Lee, D.J., Zagajewski, B.: Citizen science and webgis for outdoor advertisement visual pollution assessment. Computers, Environment and Urban Systems **67**, 97–109 (2018)
10. Chromiak, M., Stencel, K.: A data model for heterogeneous data integration architecture. In: International Conference: Beyond Databases, Architectures and Structures. pp. 547–556. Springer (2014)
11. Chuchro, M., Franczyk, A., Dwornik, M., Lesniak, A.: A big data processing strategy for hybrid interpretation of flood embankment multisensor data. Geology, Geophysics and Environment **42**(3), 269–277 (2016)
12. Czerepicki, A.: Perspektywy zastosowania baz danych nosql w inteligentnych systemach transportowych. Prace Naukowe Politechniki Warszawskiej. Transport (92), 29–38 (2013)
13. Fraczek, K., Plechawska-Wojcik, M.: Comparative analysis of relational and non-relational databases in the context of performance in web applications. In: International Conference: Beyond Databases, Architectures and Structures. pp. 153–164. Springer (2017)
14. Goodchild, M.F.: Citizens as sensors: the world of volunteered geography. GeoJournal **69**(4), 211–221 (2007)
15. Harezlak, K., Skowron, R.: Performance aspects of migrating a web application from a relational to a nosql database. In: International Conference: Beyond Databases, Architectures and Structures. pp. 107–115. Springer (2015)
16. Hricov, R., Šenk, A., Kroha, P., Valenta, M.: Evaluation of xpath queries over xml documents using sparksql framework. In: International Conference: Beyond Databases, Architectures and Structures. pp. 28–41. Springer (2017)
17. Inglot, A., Koziol, K.: The importance of contextual topology in the process of harmonization of the spatial databases on example bdot500. In: 2016 Baltic Geodetic Congress (BGC Geomatics). pp. 251–256 (2016)

18. Kopec, A., Bala, J., Pieta, A.: Webgl based visualisation and analysis of stratigraphic data for the purposes of the mining industry. Procedia Computer Science **51**, 2869–2877 (2015)
19. Kozioł, K., Lupa, M., Krawczyk, A.: The extended structure of multi-resolution database. In: International Conference: Beyond Databases, Architectures and Structures. pp. 435–443. Springer (2014)
20. Krawczyk, A.: A concept for the modernization of underground mining master maps based on the enrichment of data definitions and spatial database technology. In: E3S Web of Conferences. vol. 26, p. 00010. EDP Sciences (2018)
21. Li, Y., Kim, G., Wen, L., Bae, H.: Mhb-tree: A distributed spatial index method for document based nosql database system. In: Ubiquitous Information Technologies and Applications, pp. 489–497. Springer (2013)
22. Longley, P.A., Goodchild, M.F., Maguire, D.J., Rhind, D.W.: Geographic information systems and science. John Wiley & Sons (2005)
23. Loor, J.S., Fdez-Arroyabe, P.: Aerial and satellite imagery and big data: Blending old technologies with new trends. In: Big Data for Remote Sensing: Visualization, Analysis and Interpretation, pp. 39–59. Springer (2019)
24. Lupa, M., Kozioł, K., Leśniak, A.: An attempt to automate the simplification of building objects in multiresolution databases. In: International Conference: Beyond Databases, Architectures and Structures. pp. 448–459. Springer (2015)
25. Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., Jie, W.: Remote sensing big data computing: Challenges and opportunities. Future Generation Computer Systems **51**, 47–60 (2015)
26. Martins, P., Cecílio, J., Abbasi, M., Furtado, P.: Gisb: A benchmark for geographic map information extraction. In: Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery, pp. 600–609. Springer (2015)
27. Mirek, K., Mirek, J.: Non-parametric approximation used to analysis of psinsar[tm] data of upper silesian coal basin, poland. Acta Geodynamica et Geomaterialia **6**(4), 405–410 (2009)
28. Pavlicek, A., Doucek, P., Novák, R., Strizova, V.: Big data analytics–geolocation from the perspective of mobile network operator. In: International Conference on Research and Practical Issues of Enterprise Information Systems. pp. 119–131. Springer (2017)
29. Piorkowski, A.: MySQL Spatial and PostGIS–implementations of spatial data standards. EJPAU **14**(1), 03 (2011)
30. Płuciennik, E., Zgorzałek, K.: The multi-model databases–a review. In: International Conference: Beyond Databases, Architectures and Structures. pp. 141–152. Springer (2017)
31. Wyszomirski, M.: Przeglad mozliwosci zastosowania wybranych baz danych nosql do zarzadzania danymi przestrzennymi. Roczniki Geomatyki-Annals of Geomatics **16**(1 (80)), 55–69 (2018)
32. Xu, G., Gao, S., Daneshmand, M., Wang, C., Liu, Y.: A survey for mobility big data analytics for geolocation prediction. IEEE Wireless Communications **24**(1), 111–119 (2017)
33. Zhang, X., Song, W., Liu, L.: An implementation approach to store gis spatial data on nosql database. In: Geoinformatics (GeoInformatics), 2014 22nd International Conference on. pp. 1–5. IEEE (2014)