COMPARATIVE EVALUATION OF NOSQL AND RELATIONAL DATABASES PERFORMANCE WHILE ANALYZING SEMI-STRUCTURED GEOSPATIAL DATA

DOI: http://dx.doi.org/10.18509/GBP.2019.64
UDC: 004.65-047.44:[711.2:528.9

Mohamad Hasan
Evgeny Panidi
Vladimir Badenko
Saint Petersburg State University, Institute of Earth Sciences,
Department of Cartography and Geoinformatics, **Russia**

ABSTRACT

Non-relational (NoSQL) databases have gained popularity in the recent years, especially in Web applications, where semi-structured data formats (e.g., JSON, XML) that used widely to store data on the Web are more suitable to be managed by NoSQL database management systems. Web mapping software (OpenLayers, Leaflet, MapServer, GeoServer, etc.) implement geospatial extensions of such data formats, for example GeoJSON that is a standardized JSON document type, which can be used to represent simple geographical features alongside with their non-spatial attributes. In such a context, different relational database management system (RDBMS) vendors implemented JSON support in their software to provide greater flexibility for used relational database models. In this paper, a processing performance of semi-structured geospatial data in different databases management systems (DBMS) is analyzed. The analysis is performed on the example of GeoJSON datatype for different NoSQL DBMSs categories (MongoDB, Cassandra, CouchDB and Neo4J), in parallel with analysis of the PostgreSQL which is RDBMS with JSON processing capabilities. The results are presented for GeoJSON writing latency, geospatial querying based on location with and without spatial indexing, and querying based on attributes alongside with querying based on location. The conclusions can be used to support content-based estimations of the demands to the DBMS and its restrictions at the database design stage.
The results of the analysis show that in writing latency parameter MongoDB and CouchDB demonstrate the highest results. Additionally, the results demonstrated that organizing of the geoJSON data in a materialized view in PostgreSQL shows fastest results for both location querying and location combined with attributes querying, but it requires to use 23% more of storage size. Both MongoDB and Cassandra returned fast results without any additional disk space. Finally, when using geospatial index (supported only in MongoDB and PostgreSQL), PostgreSQL reading latency is reduced by a factor of 10% when querying geospatial location using the spatial indexes, while MongoDB shows no significant advantage of spatial index use.

Keywords: Geospatial Databases, NoSQL, RDBMS, GeoJSON

INTRODUCTION

Implementation of Web 2.0 has popularized semi-structured data types (e.g., JSON and XML). JSON (JavaScript Object Notation) is widely spread being a common standard for many applications on the Web. GeoJSON is a JSON document type that has keys

contained geospatial data. GeoJSON format is very common in Web mapping applications. Such a popularity of semi-structured data encouraged traditional relational databases management systems vendors to include JSON and GeoJSON to its supported types [14].
NoSQL databases was designed to overcome the problem of unstructured and semi-structured data storage and retrieve, presented in traditional relational database. NoSQL databases have very simple data model and are schema-free (i.e., the data is not stored using fixed table schemas as in relational databases) [11]. As NoSQL databases started to gain popularity in last decade, some of NoSQL databases started also to support geospatial data analysis. Most of NoSQL databases support only point-geometry and geospatial distance when providing data queries.

DATABASE ARCHITECTURES

The domain of database management systems has been diversified over the recent years, due to the new database architectures were emerged. While relational databases are prevalent in the majority of use cases, unstructured data calls for new approaches [13]. Some DBMSs cannot be assigned solely to a single architecture type, while these DBMSs may provide features specific for other groups. However, differentiation of DBMS architectures is useful to understand spectrum of data base facilities and applications.
RDBMS: Relational databases currently are the most widespread type of databases, with well-known DBMS PostgreSQL as an example. In the relational databases model data are stored in tables with relations in a mathematical sense. Each relational database model implements an n-ary relation, by specifying n columns each of that stores a certain type of data for all entries. Columns provide scheme of the table. The data inside the table are stored as row entries in form of n-tuples. This design approach makes relational databases adaptable to many application domains. It allows to define certain columns as unique key values and to create references between relations. If utilized correctly, this reduces redundancy of data [3].
A huge advantage of relational database model over other models, is the support of ACID transactions which are a set of conditions to solve concurrency and reaction to failures. A database is being ACID compliant, by fulfilling the following principles [4]:

- Atomicity: transactions are considered as a single unit;
- Consistency: results are consistent database states;
- Isolation: no side-effects among parallel transactions;
- Durability: their results are persistent.

ACID is necessary if a series of critical changes is made that can only be allowed to complete fully or have no effect at all. While transactions are no inherent attribute of relational databases, it usually distinguishes them from newer approaches to data storage. Whether or not transactions are needed is subject of the application domain.
NoSQL databases: Although relational databases are the most common type of databases and adaptable to many scenarios, not all data models fit well with a relational schema. Relational databases enforce a schema that has to be changed due to redesigns in the underlying application. In addition, the support of ACID transactions to ensure data consistency, becomes sometimes unnecessary and critical in contexts of big amount of data handling. This has become an issue for online services over the recent years, which need database systems that can be distributed [9].
NoSQL databases solve these problems with different approaches to data organization. The term NoSQL does not describe a specific group of databases, instead it is a umbrella

term for systems that break the classic relational approach. Without the relational basis, NoSQL databases do not provide a common query language. In order to improve the scalability, many NoSQL databases abandoned ACID transactions [7]. Subsequently, the most important types of NoSQL databases are described below.

Key-value stores: Key-value databases are the simplest category of NoSQL databases. The key-value category could be considered as the first emerging type of the NoSQL databases. Even though the key-value vendors build DBMSs in different ways, they have many common features. A HashMap is a simple data structure which can hold a set of key-value pairs and they all store data as maps. The HashMap concept was inspired by Amazon in their DynamoDB storage model [6]. The APIs of such databases offer the match query options. A match query extracts the value associated with a certain key, such as:

- Get (key) – extracts the value given a key;
- Put (key, value) – creates or updates the value given its key;
- Delete (key) – removes the key and its associated value.

The simplicity of the available actions is the reason why they are easy to scale and generally have great performance. The aggregates of keys can be stored into one single bucket which is a namespace for keys. However, this approach increases the chance of key conflicts [6].

Document-Oriented Databases: IBM introduced first document-oriented databases through its Lotus Notes database. The database is basically built on key-value data store where the value is a document, while the document is a collection of other key-value collections. Oppositely to the document-oriented database, key-value stores store only scalar values and cannot embed an object into another object. A document can be addressed by unique URL and have to be readable through the application programming interface (API) of the DBMS. Usually documents are stored in Web-compatible formats (JSON, XML, Binary JavaScript Object Notation – BSON). Documents can also contain attachments, which makes document stores useful for content management. Additionally, the document is usually structured and DBMS-readable. An advantage of the document-oriented database is that querying is possible either by the key or by the fields of the value [8].

The document-oriented databases are the most wide-spread type of the databases because they are more suitable for low latency with high performance. Unfortunately, these databases are not suitable when multi-document transactions, or authentication, or complex joins across collections, or finally extreme compression are needed. Also, document-oriented databases have a lack of complex security (user roles, document level security), and support ACID transactions only at the document level [15]. Examples of document-oriented databases are the MongoDB, CouchDB, Couchbase, DjonDB.

Column-Oriented Databases: This type of NoSQL databases is based on Google's BigTable model, consequently these databases are also referred to as BigTable clones. These databases are built to store and process very large data amounts. The database in this case is also a form of key-value pairs, but storage is organized in a semi-schematized and hierarchical pattern. BigTable is a map that is stored in sparse, distributed, persistent and multi-dimensional way. The map is indexed using row-keys, column-keys and timestamps. Each value in the map is an un-interpreted array of bytes [5]. The BigTable model later influenced the Cassandra, Hypertable, and HBase column-oriented databases.

Graph databases: The Relational databases are not designed to effectively process hierarchical or graph data. RDBMSs in such a case require many one-to-many and many-to-many relationships, which can't be modelled efficiently in a relational database. Graph databases are designed to represent data while the relationships are paramount, as these databases are ideal for capturing any data consisting of complex relationships such as social networks or product preferences. In other words, this type become best suited when the application has datasets with complex interconnections and do not offer support for ad-hoc queries [16]. Therefore, graph databases are the best choice for applications that need fast and extensive reference following, especially when data fits in memory [10]. In graph databases, entities are represented as nodes and relationships between them as edges. Nodes and edges can have attributes. These databases focus more on the relationships between entities than on the entities themselves. Edges can be added or removed at any time allowing one-to-many and many-to-many relationships to be expressed easily and avoiding anything like an intermediate relationship table needed in a relational database to accommodate many-to-many joins [16]. Neo4J is an example for graph database.

SELECTION OF DBMS AND THEIR GEOSPATIAL CAPABILITIES

The DBMSs for testing are selected basing on their popularity in each category. PostgreSQL with PostGIS extension is the most popular for querying geospatial data in the RDBMS category [1]. MongoDB and CouchDB are the most popular among document–oriented databases, while Cassandra and Neo4J are the most widespread among column–oriented and graph databases, accordingly to the Popularity Ranking of Database Management Systems (http://db-engines.com/en/ranking).

The spatial capabilities within PostgreSQL come from the PostGIS extension. PostGIS is the most complete implementation of the Simple Features (officially Simple Feature Access), which is an Open Geospatial Consortium (OGC) standard. Great number of GIS software (such as ArcMap, ArcGIS Server, QGIS, Grass, and GeoServer and many others) support PostGIS. ESRI deploys PostgreSQL with its GeoEvent Server product and recommends the use of PostgreSQL as the back-end for the temporal storage [12].

MongoDB supports geospatial querying in JSON documents for all sets of shapes (i.e., points, lines and polygons) and querying distances, intersections, within object and nearest object queries. MongoDB also supports geospatial indexing – 2d and 2dsphere indexing (https://docs.mongodb.com/manua).

CouchDB has third-party extension called GeoCouch that supports some geospatial capabilities (in fact, weak capabilities), and supports indirect spatial indexing implemented using GeoHash software to generate hash values and B-tree data model to store hash values as keys (https://developer.couchbase.com/documentation/server/5.0/fts/fts-geospatialqueries .html#geospatial_queries__introduction-to-geospatial-querying).

Cassandra has geospatial functions for points and polylines, including intersection, within check, disjoint check and containing queries (https://docs.datastax.com/en/ datastax_enterprise/5.0/datastax_enterprise/srch/queriesSpatial.html). Cassandra also supports geospatial indexing, but it is not a build-in function. Indexing could be enabled using third-party extensions [2].

Neo4j supports geospatial functions using programming in Java. Neo4j supports point-type geometry, and distance queries as a build-in function (https://neo4j.com/docs/developer-manual/current/cypher/functions/spatial).

DBMS PERFORMANCE ANALYSIS

The performance analysis was done using a single node computer with 12 GB RAM, Intel Core i7-6500u processor with 2.59 GHz clock speed, 64-bit Windows 10 operating system, solid state hard drive. The analysis was done on PostgreSQL 10.4 with PostGIS extension 2.4.2, MongoDB version 4.0.2., Neo4j 3.4.7, Cassandra 3.0.9 and Couchbase 2.2.0.

The dataset used for analysis incorporated points of interest in USA collected from OpenStreetMap (OSM). OpenStreetMap is an open access map and Web service, which provides maps for large areas all over the world. All features in the dataset are represented as points and have properties. The data are saved in the databases using GeoJSON file format. GeoJSON is an JSON-based open standard and format designed for representing simple geographical features, alongside with their non-spatial attributes. The dataset has around 13 million records and occupies almost 3.5 gigabytes of disk storage.

The GeoJSON files has the following structure:

{"id": integer, "type": "Feature", "geometry": {"type": "Point", "coordinates": (X, Y)}, "properties": {"code": integer, "name": string, "type": string, "address": string}}

The performance analysis was built upon the testing of queries most common for desktop GIS and Web-GIS (i.e., selection by attribute, selection based on location within a distance to specific coordinates, and combination of both selections). Additionally, since the application of the database (in the context of our studies) is to save the data mining results when obtained from social media, a performance analysis for inserting records was provided. Therefore, four queries where selected for testing.

Firs query – Q1 is conducted to analyse the performance of writing latency of geoJSON document in the database. The second query – Q2 tests geospatial querying without geospatial indexing. Q2 results is the time conducted to return the points within a distance from a certain point. The third query – Q3 is the same to Q2, but conducted in the databases with support geospatial indexing (i.e., MongoDB and PostgreSQL). The forth query – Q4 tests selection using a complex selection based on location within a certain distance and on attribute selection.

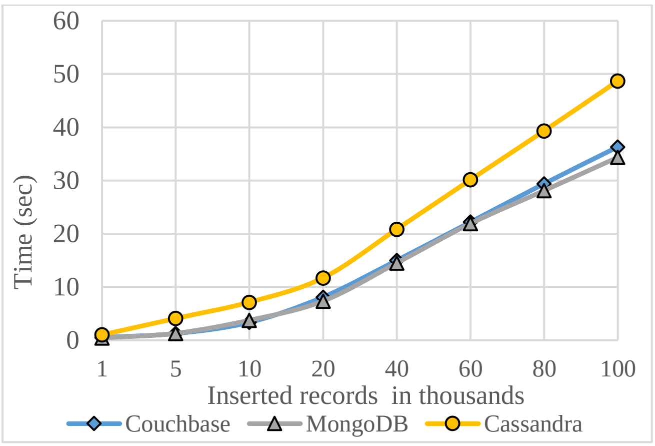

Figure 1. Inserting latency for GeoJSON records in Couchbase, MongoDB and Cassandra

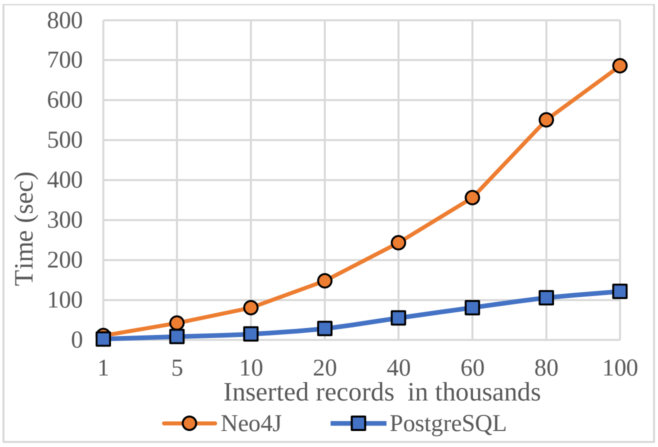

Figure 2. Inserting latency for GeoJSON records in PosgreSQL and Neo4J

Document-oriented (MongoDB and CouchBase) NoSQL databases returned the best writing latency for GeoJSON, this can be attributed to the nature of database which supports originally JSON data type. Neo4J scored the lowest writing results; this can be attributed to the type of the database, which is graph database. Graph databases store relations as separate objects, which in theory should return faster results but at cost of slower writing.
Q2 tests selecting points within a certain distance from a specific point. While not all databases support spatial indexes, all of the selected databases support spatial functions (except for Cassandra, where spatial function is processed in Python to obtain results). The materialized view in PostgreSQL returns the best results for Q2, while processing the JSONB data in PostgreSQL database returns the worst results. This could be attributed to PostgreSQL does not process the JSONB data directly. JSONB is stored as character type and needed to be parsed every time when querying the data.
Q3 repeated the same query as Q2 but with applying spatial indexes. The results for Q3 are the same as the previous query, moreover the spatial index did not significantly improve the read latency.

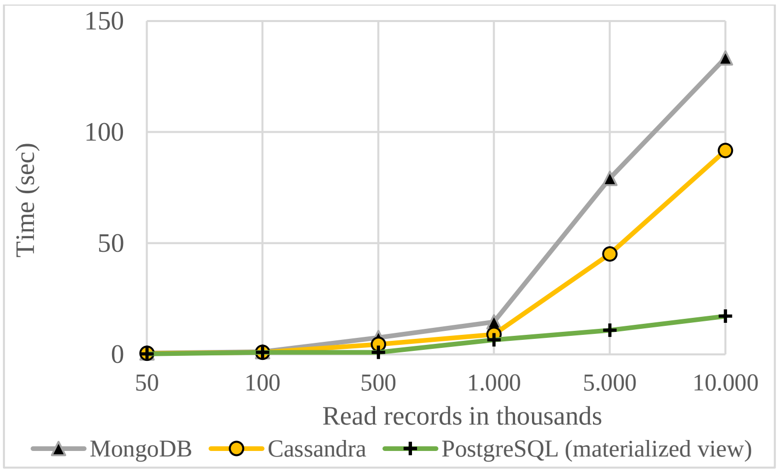

Figure 3. Reading latency Q2 for MongoDB, Cassandra and PostgreSQL (materialized view)

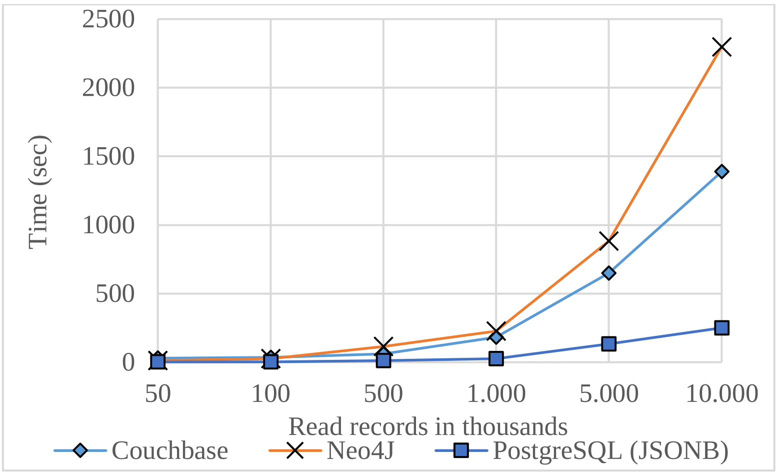

Figure 4. Reading latency Q2 for CouchBase, Neo4J and PostgreSQL (JSONB)

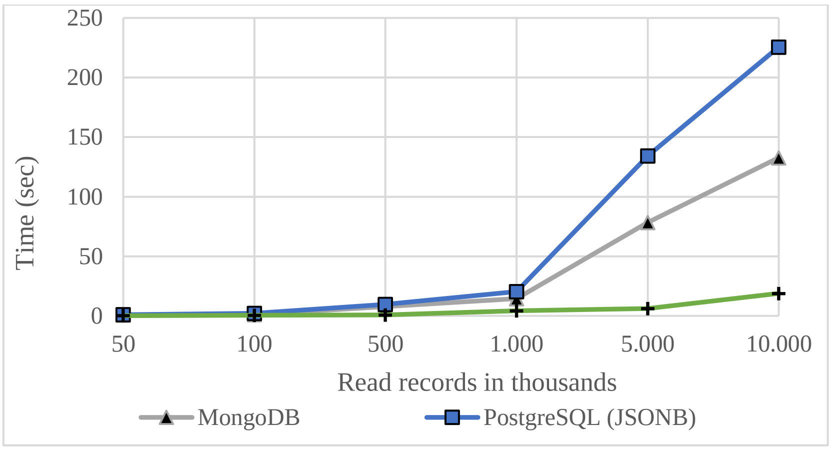

Figure 5. Reading latency for Q3 using for MongoDB and PostgreSQL (JSONB, materialized view)

Finally, the Q4 tests both selections, selection within a certain distance and attributes selection. Databases performance in Q4 was similar to the Q2 in the order of analyzing the data.

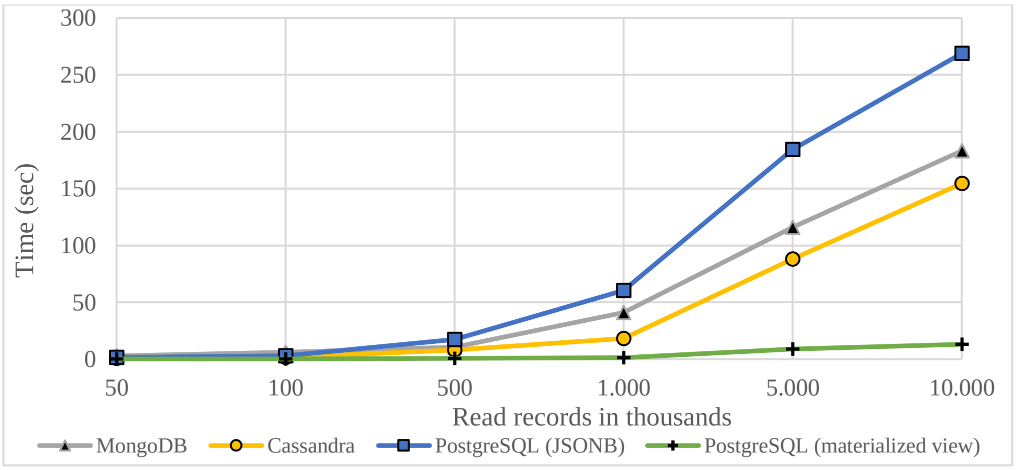

Figure 6. Reading latency for Q4 using for MongoDB, Cassandra and PostgreSQL (JSONB, materialized view)

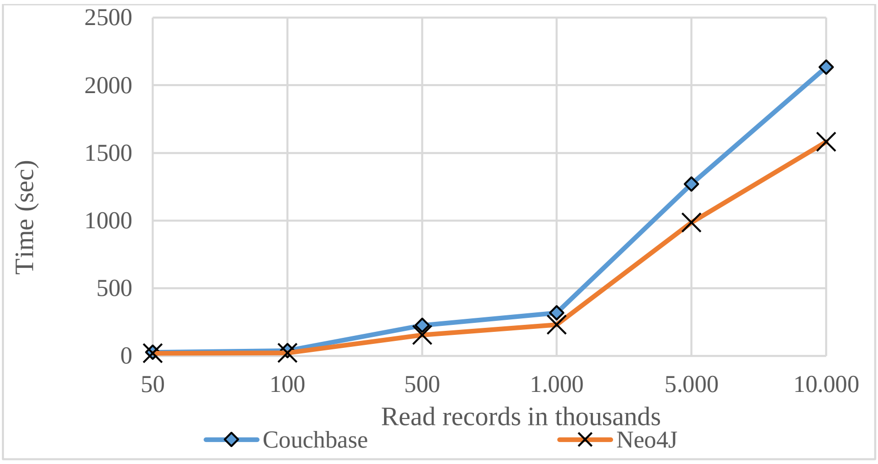

Figure 7. Reading latency for Q4 using for Couchbase and Neo4J

CONCLUSION AND FUTURE WORK

Results show that analysis of the data in materialized view in PostgreSQL returns the fastest results when querying, while direct analysis of JSONB data in PostgreSQL is performed significantly worse. Even though, materialized view performs better than other databases, a drawback of using a materialized view is a requirement of more disk space (in this case, 23% more). Moreover, PostgreSQL performs slower than other databases when writing JSON documents. Both Cassandra and MongoDB perform well when both reading and writing JSON records, while Neo4J and Couchbase perform worse than the other databases.

Since all of the performance analysis done on a single node, a more analysis should be done for the multiple nodes. Moreover, more analysis should be done to explore the performance of database management systems performance for other semi-structured datatypes such as XML.

REFERENCES

[1] Ballatore A., Tahir A. McArdle G. Bertolotto M., A comparison of open source geospatial technologies for Web mapping, International Journal of Web Engineering and Technology, 2011, vol. 6 (4), pp. 354-374.

[2] Brahim M.B., Drira W., Filali F., Hamdi N., Spatial data extension for Cassandra NoSQL database, Journal of Big Data, 2016, vol. 3 (11), pp. 1-11.

[3] Codd E. F., A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, NY, USA, 1970, vol. 13 (6), pp. 377–387.

[4] Coronel C., Morris S., Database Systems: Design, Implementation, and Management – 13th edition, Cengage Learning. Boston, USA. 2018, pp. 493–495.

[5] Espling D., Östberg P.O., Elmroth E., Integration and Evaluation of Decentralized Fairshare Prioritization (Aequus), IEEE International on Parallel & Distributed Processing Symposium Workshops (IPDPSW), Beijing, China, 2014, pp. 1198–1207.

[6] Fowler A., The State of NoSQL 2016: A quick guide to the NoSQL landscape, Addison-Wesley Professional. 1st edition. Boston, USA. 2016.

[7] Harrison G., Next Generation Databases: NoSQL, NewSQL, and Big Data. Apress, USA, 2015.

[8] He C., Survey on NoSQL Database Technology, Journal of Applied Science and Engineering Innovation. 2015, vol. 2 (2), pp. 50–54.

[9] Hecht R., Jablonski S., NoSQL evaluation: A use case-oriented survey, International Conference on Cloud and Service Computing, USA, 2011, pp. 336–341.

[10] Joishi J., Ashish Sureka A., Graph or Relational Databases: A Speed Comparison for Process Mining Algorithm, 19th International Database Engineering & Applications Symposium. Yokohama, Japan, 2015.

[11] Sapthami R., Rodrigues A.P., Study on Handling Semi-structured Data using NoSQL Database, International Journal of Engineering and Innovative Technology (IJEIT), vol. 5 (10), 2016, pp. 43–46.

[12] Steiniger S., Hay G.J., Free and open source geographic information tools for landscape ecology, Ecological Informatics, 2009, vol. 4, pp. 183–195.

[13] Leavitt N. Will NoSQL Databases Live Up to Their Promise?, Computers journal, 2010, vol. 43 (2), pp. 12–14.

[14] Liu C.H., Hammerschmidt B., McMahon D., JSON Data Management Supporting Schema-Less Development in RDBMS, SIGMOD-PODS conference, Utah, USA, 2014 pp. 1247–1258.

[15] Manoj V., Comparative study of NoSQL document, column store databases and evaluation of Cassandra, International Journal of Database Management Systems, 2014, vol. 6 (4), pp. 11–26.

[16] Varga V., Jánosi-Rancz K.T., Kálmán B., Conceptual Design of Document NoSQL Database with Formal Concept Analysis, Acta Polytechnica, Hungarica, 2016, vol. 13 (2), pp. 229–248.