Norbert ZANIEWICZ[1],
ORCID: 0000-0002-9516-7529
Andrzej SALAMOŃCZYK[1]
ORCID: 0000-0003-1562-6762

[1] Siedlce University of Natural Sciences and Humanities
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

Comparison of MongoDB, Neo4j and ArangoDB databases using the developed data generator for NoSQL databases

DOI: 10.34739/si.2022.26.04

Abstract. NoSQL databases are gaining more and more popularity and are an important alternative to relational databases. Examples of such databases are Neo4j, MongoDB and ArangoDB. These databases are described in this article, compared with each other, and the performance results for adding an object to the database, deleting an object, searching and populating the database are presented. Results show that the fastest database is MongoDB, except for one measurement of removal.

Keywords. NoSQL databases, Neo4j, MongoDB, ArangoDB

1. Introduction

Database solutions, from relational, through network, hierarchical, object-oriented, to non-relational, are dynamically developing with the use of new technologies. Companies constantly adapt their solutions to the requirements of customers, trying to stay ahead of the competition and gain even a temporary advantage. Thus, the database environment is one of the most progressive. The rejection of the relational database model by the pioneers of non-relational solutions was a response to the market demand in the segment of more efficient solutions with huge amounts of data. Companies such as Google and Facebook began working on various ways to tackle the growing amount of all kinds of information they must store and process, which has become increasingly costly both financially and efficiently.

Successive database systems were developed in the same period and met more and more different needs for data storage. Graphs, which facilitate relationship mapping and finding patterns, are currently used as purchasing scenario creation systems, e.g. on eBay. Document databases that made it easier for application developers to store and process JSON documents without the need to interfere with their structure.

The emergence of NoSQL databases was also associated with new database management systems and, more importantly, new languages or query structures, often significantly different from standard SQL. The flagship example are graph databases, based on relations that are described edges of the graph between nodes. Graph languages such as Cypher, GraphQL, Gremlin and SPARQL were created especially for the needs of these databases. Queries in document databases look a bit more friendly to SQL users, where the query language is similar, and sometimes almost identical to its relational counterpart or programming languages.

Adaptations of new solutions bring measurable benefits in the form of resource savings or a significant improvement in operating efficiency. Therefore they are constantly updated, but they try to keep their basics unchanged and are very often backward compatible. NoSQL databases are an example of dynamic development, which was forced by the development of the entire information technology industry. Demands had to be met with an appropriate response. We can expect the emergence of many hybrid systems or activities for the compatibility of operation of various solutions. Rarely, in large solutions, one database approach is enough. Therefore, it is worth comparing various database systems, including measuring their effectiveness.

2. NoSql Databases

In this chapter we will discuss three examples of NoSql databases, they are Neo4j, MongoDB, ArangoDB, which will be compared with each other.

2.1. Neo4j

Neo4j is not only a graph database system, but also the company responsible for it. In 2000, future founders of Neo4j, influenced by the performance problems encountered in relational databases, started working on a prototype of a graph database. The creators formalized their activities only after seven years, establishing the company, and then the first versions of this database appeared. However, only in 2010, version 1.0 appeared, starting a new stage of database solutions.

The database implemented in Java was dynamically developing, and the following years brought revised versions enriched with new functionalities. The creators collected more and more funds and large clients using their solution. Currently, Neo4j is the most popular solution among graph databases. It owes such a position on the market to its rich functionality. Compliance with ACID (Atomicity, Consistency, Isolation, Durability), i.e. a set of properties that guarantee the consistent execution of transactions in the database, support for clustering, and ease of scaling. These are the features that affect the ever-increasing quality of this graph database.

The offered paid solution, Enterprise version, provides excellent horizontal scaling, i.e. it is very easy to add resources via new machines in the resource pool. Scaling is performed by two types of clustering, which are high-availability clustering and casual clustering. The first type of clustering enables the use of distributed machines, where one of the servers acts as a supervisor over several slave machines. In the event of a failure, a new supervisor is selected, and the database operates continuously, which yields high availability.

Casual clustering, on the other hand, provides transferable read-only replicas. When one of the database instances fails, the primary servers provide read and write access, including data modification. However, very often graph databases are used to search for information, hence a significant number of replicas with read-only functionality. Such a structure allows for efficient resource management and ensuring high-class security and availability. This solution also has a load balancer in order to increase the comfort of use by evenly loading all available resources.

For communication between the database and applications, a binary protocol Bolt created in 2015 is used, which gained wider interest a year later by using it in version 3.0 of the Neo4j server. This protocol, which works on TCP sockets, allows to send messages with parameters. The server replies to such a query with a reply message with an optional stream of results. Neo4j databases can be implemented as server solutions, and also embedded locally, e.g. in a Java application (Fig. 1). This means that in the case of local solutions or implementation for mobile devices, the burden of network communication is reduced. Neo4j also provides native Java API for data manipulation in applications.

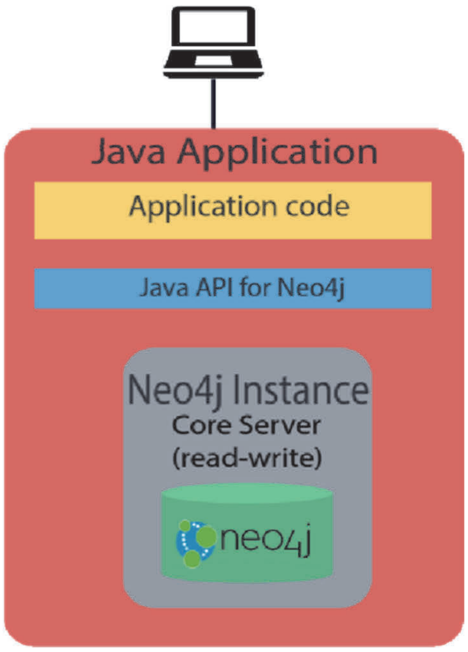

Figure 1. Java application with locally embedded Neo4j database instance. Source: [8]

Providing such support from the manufacturer allows programmers to easily build an application based on the provided API without involving server resources. Neo4j adopted a graph data model, i.e. sets of nodes and edges between them described with values. Nodes can store information about the object they represent. Edges are directed and described connections of nodes in the graph. Such a connection allows you to easily perform such operations as, for example, counting steps between the nodes of interest, as well as returning dependent nodes in a characteristic manner, specified in the query.

The distinguishing feature of Neo4j is the use of its own query language Cypher. This declarative query language was developed in the course of work on the graph database system. Initially, it was intended to be the native query language only in Neo4j, but in 2015 it was transformed into the openCypher initiative, which aims to provide the graph database segment with a fully documented universal query language for all datasets of this type. Cypher is a systematized translation of the description of the relationship of nodes and edges in the graph into a query. Currently, it uses many keywords present in SQL, but often in a different sense from its relational competitor. The entire language operates on relatively expensive patterns, which, when properly used, allow to obtain results that are not available for classic database engines.

Neo4j is currently used by big online stores for contextual suggestions. Changes to the graph are tracked, such as items from the wish list, and based on them, more are proposed. A similar mechanism is used to suggest items on auction sites based on searched phrases.

2.2. MongoDB

MongoDB is one of the document solutions in the NoSQL database segment. This means that the system operates on storing data in documents similar to JSON. The document database is popular. It is used by such large companies as Adobe, eBay and Electronic Arts. It is offered in several versions, including free Community and paid Enterprise. The paid option includes more database management functions and additional security options, e.g. authentication using the Kerberos protocol. Like its competitors, it offers horizontal scalability and replication mechanisms to ensure data security.

The scalability of this database is basically only limited by the amount of resources connected in a distributed network. Thanks to the use of the Google BigTable sharding system, i.e. the division into master and slave servers, the data is divided and spread over various servers, which can be added to the pool of resources endlessly in theory. Data reading is controlled by an appropriate process having knowledge of the data localization, which carries out the request addressed to the server. The sharing system automates many distribution processes and load care, thanks to which reading even from different servers is fast and can be carried out simultaneously from different machines. Replication - that is, creating a server copy to ensure redundancy based on the same solution as scaling, slave servers have a copy of data from master servers, this ensures the desired redundancy and safety in case of failure. Additional replicas can be used as additional read servers to increase performance.

MongoDB also provides database drivers for many programming languages, including C++, C#, Java, PHP or Python which allows for its wide application. Communication with the server is based on the http or REST interface, which allows for easy use in network applications. The latest drivers allow to choose synchronous or asynchronous work with the database and are developed to support other languages.

MongoDB is a NoSQL database and like other solutions of this type, it departed from certain relational assumptions in favor of a more efficient system. This database uses documents as the main data storage unit. As understood by this database system, a document is an equivalent of a row in relational databases, with the difference that it can have nested values, which in the case of standard databases must be implemented using foreign keys and additional tables. There is also an option to use references to other documents, which makes it easier to map relational database schemas in which we strive to normalize the data. The normalization process consists in creating tables and relationships appropriate to the imposed rules, which is to prevent redundancy and ensure data consistency.

In the case of MongoDB, denormalized data models simplify the work , so is possible manipulate the entire document in one operation. A document in such a database is a hierarchically arranged data structure using an appropriate format, e.g. JSON (JavaScript Object Notation) in visual representation and BSON - a JSON document in binary form. The use of JSON documents made the default query language used in MongoDB to be JavaScript. Moreover, each driver uses the structure of the language it was written into. In practice, this means that the choice of language affects how the queries are built.

2.3. ArangoDB

ArangoDB is the name of a database system under a free and open source license, created in 2011. Initially, it was published under the name AvocadoDB, which faced legal problems, and a year later it got its current name. Proposed by ArangoDB. The database system is one of the most universal because it supports as many as three data models: key-value, documents and graphs.

The motivation for this direction of development was the combination of models already offered by NoSQL databases in one solution. Neo4j uses graphs, MongoDB uses documents, and the creators of ArangoDB decided to create a base with a multi-model approach to overcome the need to use different solutions for different types of data. This relatively new database system has many clients, incl. this is Cisco or Thomson Reuters, which is one of the largest message delivery companies. It uses ArangoDB to build its platform for internal information exchange, analysis and intelligence management. The AQL query language in particular helps here.

The motivation for this direction of development was the combination of models already offered by NoSQL databases in one solution. Neo4j uses graphs, MongoDB uses documents, and the creators of ArangoDB decided to create a base with a multi-model approach to overcome the need to use different solutions for different types of data. This relatively new database system has many clients, incl. this is Cisco and Thomson Reuters, one of the world's largest news delivery companies. It uses ArangoDB to build its platform for internal information exchange, analysis and intelligence management, AQL query language helps in such tasks.

Despite its young age, ArangoDB has been equipped with many functionalities that competitors have. Providing, among others, vertical scaling - that is, the use of more and more efficient servers, and vertical scaling, i.e. the creation of clusters from many servers. Like MongoDB and Neo4j, it uses many servers, each of which has some data and depending on it can be a leader (parent server) or a slave server. The entire system also uses coordinators who act as an intermediary between the ArangoDB cluster and clients, thanks to which they perform all or part of the query depending on the data they have. Such a structure will ensure easy scalability by adding new servers to the pool of machines. The model used is also an important aspect of scalability. Different models scale better vertically or horizontally.

ArangoDB also supports many programming languages, Java, JavaScript, PHP, .NET and Scala are just some of them. In communication with the server, it can use the http protocol or the binary VelocyStream protocol introduced in version 3.2, which in some solutions of short queries gains an advantage over the http protocol. To support many models and use several of them in one query, ArangoDB has implemented the declarative AQL (ArangoDB Query Language). This language uses English words as keywords.

Currently, ArangoDB is becoming popular for undecided companies that are in the process of defining their data storage needs, or are confident that a uniform solution will not meet their needs. The discussed NoSQL databases are among the most popular and it was decided to focus on them, test and compare them directly.

2.4. Testing and comparing NoSQL databases

Analysing the bibliography in terms of comparing databases, we most often encounter SQL databases. When examining SQL databases, we can take into account various factors like performance and security. For example Barczak et al., [1]. researched influence of indexing methods on effective functioning of the database and Skaruz, [2], analysed the security of databases combining neural networks and classification approach.

A performance comparison of both SQL and NoSQL databases is described in [3]. Li and Manoharan compared read, write, delete, and instantiate operations on key-value stores implemented by NoSQL and SQL databases. Their results showed that not all NoSQL databases perform better than SQL databases.

More detailed studies on performance are described by Martins, Abbasi and Sá, [4]. They tested and evaluated: MongoDB; Cassandra; HBase; OrientDB; Voldemort; Memcached and Redis databases. Also five NoSQL databases (Redis, MongoDB, Couchbase, Cassandra, HBase) were tested by Tang and Fan, [5], using a measurement tool - YCSB (Yahoo! Cloud Serving Benchmark).

Another set of NoSQL databases (Couchbase, MongoDB and RethinkDB) was tested by Pereira et al., [6]. They tested single thread and multiple threads scenarios. The results show that Couchbase had a better performance at most of the operations, except for retrieving multiple documents and inserting documents with multiple threads, operations in which MongoDB yields better result.

Gupta et al., [7], selected database for different type of data, MongoDB (document stores), Cassandra (wide column stores), Redis (key value stores), and Neo4j (graph databases) and compared them with each other. Their results show that write and delete operations are fast for databases MongoDB, Redis and Cassandra, whereas read operation is comparatively slow in Cassandra.

One can notice that comparisons and performance studies on different kits are popular and worth doing. The set of tested bases depends on the current popularity and usefulness in specific applications. Zaniewicz, [8], compared databases MongoDB, Neo4j and ArangoDB in his master thesis and some of the results are presented later in this article. For testing, he used a data generator developed by the author.

3. Creating a test dataset using the data generator

The "imdb-5000-movie-dataset.csv" file from www.kaggle.com was taken as input data. It is a site of a community of scientists and machine learning people collecting and sharing public data sets. The above-mentioned file contains a collection of almost five thousand films, each record is described by 27 columns containing, among other information, such information as the name and surname of the director, film title, actors' personalities, a link to a page in the IMDb film database, country of production, budget and many more.

The first step in the design approach was to determine on the basis of what features each database is to be built and what data will give interesting results in the graph and document approach. In the next stage of creating the file, the file was adapted to the program's requirements, this means adopting appropriate separators and checking whether they do not appear in English film titles, because some of them contain semicolons, colons, periods or apostrophes. Then, each record was marked with a number as an identifier, and in the next steps, records containing blank values were reduced to maintain the stability of the implemented solution. The data in this form had to be saved in the program memory in order to parameterize them for the purposes of the queries being performed, to avoid the impact of Java internal mechanisms on the query execution time.

In this way, a test set was obtained that was sent to individual queries and thanks to which three databases with very different approaches, having over three thousand records, were created.

4. Results

The difficult task is to choose a feature that will not favor a specific database solution. Each of the database systems described in this paper was created to fill the gap for the needs of SBD with a specific specification. In order to maintain comparable results, the basic time features for all databases were tested, such as database creation time, execution time of a simple query looking for a document or graph node, time of removing a record from the database and the time of updating the record.

The first tested parameter is the time of creating the database, taking into account the used document databases and the graph database, this is the time when it is filled with the test data set, and in the case of Neo4j also the creation of the graph edge, which is an important element generating a fairly large load. Another parameter examined is the execution time of a simple query selecting a record by title of the movie from each of the databases. The time of removing a given record and the time of editing the record were further tested.

All these parameters are also present in traditional, relational databases, which allows for a more accessible understanding of the obtained results based on the characteristics and preferred operations of specific solutions. All the results were related to information about the applications of each of the solutions and how it may affect the results.

All the tests carried out in order to obtain reliable values were repeated one hundred times. Based on the obtained results, the means, maximum and minimum values were drawn. All tests were performed on local servers using the same computer specification, i.e. Intel® CoreTM i5-6500 processor clocked at 3.2GHz, 16GB DDR4 2133MHz RAM and WD Blue 2TB 5400rpm hard drive. The databases also used their suggested default settings without optimizing them to improve performance.

Collecting the above results allows you to create a comparison that will present the results achieved by the tested databases. You have to bear in mind that despite the huge differences between them, they have the same goal - to organize and store data in an organized manner.

The first test is adding a new node / document to the database, i.e. its main organizational unit, the results are shown in Tab. 1.

Table 1. Summary of node / graph creation results in databases. Source: [8]

Parameter	**Neo4j**	**MongoDB**	**ArangoDB**
Mean	0,88 ms	0,75ms	1,01ms
Min	<1 ms	<1 ms	<1 ms
Max	3 ms	5 ms	6 ms

When analyzing Tab. 1, it can be noticed that all databases, creating their own types of organizational units, achieve quite similar times. The differences in average times are small, with only Neo4j having a slight advantage in the maximum time.

Tab. 2 shows the results of the query execution time removing the "record" from the databases. The results obtained together show that individual solutions differ significantly from each other.

Table 2. Summary of the results of removing a node/document from the database. Source: [8]

Parameter	**Neo4j**	**MongoDB**	**ArangoDB**
Mean	0,81ms	2,3 ms	1,13ms
Min	<1 ms	<1 ms	<1 ms
Max	2 ms	168 ms	6 ms

The results of this query are different from creating node, although both operations can be considered basic for each database. The removal times for MongoDB differ significantly from the results of adding it and the results achieved by competing solutions.

The maximum removal time in the case of MongoDB significantly deviates from the rest of the results and strongly affects the average, however, this sample, which constitutes 1% of all results, can be considered a statistical error caused, for example, by a temporary load of the test set or connection to the MongoDB server.

However, all the obtained values can be taken in the fast category. The average MongoDB, not taking into account the given outliers, is about 0.6 ms, which makes the database the fastest in terms of removal. Next test consisted in searching for a node or document in a given database solution, Tab. 3 summarizes these results.

Table 3. Summary of node / document search results in databases. Source: [8]

Parameter	**Neo4j**	**MongoDB**	**ArangoDB**
Mean	0,64 ms	0,18 ms	5,07 ms
Min	<1 ms	<1 ms	<4 ms
Max	12 ms	12 ms	8 ms

Searching in databases gave quite diverse results, the MongoDB document base was the best in this task, the worst, which may be a surprising result, ArangoDB. The graph solution is quite competitive in relation to MongoDB.

The high ArangoDB result is due to the lowest times achieved, a significant part of which reaches around 4 ms, which is a result four times higher than that of the competition, but retains a much lower maximum time.

The last parameter taken into account was the database creation along with the population. Such a choice was made due to the low time of creating the base structures of the database - the creation of a graph base or document collection is measured in milliseconds, and this process is often performed only once.

For this reason, the results of the creation itself would have a marginal impact on efficiency, however, taking into account filling the base in an appropriate manner according to the adopted patterns, we can simulate a change, for example, of the company's based solution. Tab. 4 summarizes the results of all databases and two variants of the Neo4j database, the first noticeable factor is the change of the unit to seconds (the MongoDB result has been properly formatted for easier reading).

Table 4. Summary of the results of creating and populating databases. Source: [8]

Parametr	**Neo4j with edges**	**Neo4j without edges**	**MongoDB**	**ArangoDB**
Mean	682,80 s	244,22 s	0,42 s	2,79 s
Min	382 s	204 s	0,4 s	2 s
Max	837 s	480s	1,07 s	7 s

5. Conclusions

In this work, a comparison of Neo4j, MongoDB and ArangoDB databases was made using data generated in a data generator prepared for this purpose. The article also presents a short description of these databases.

To evaluate the effectiveness, a data generator developed for the purpose of the topic was used, which, on the basis of a data set about movies from the IMDB website contained in a .csv file, created an appropriate structure to create the mentioned databases, fill them and test some aspects of the performance offered by NoSQL databases, which allowed for carrying out a database comparison. Analyzing the results presented in Results section, it is clear that graph bases were formed the longest. This result is not surprising due to the complexity of the graphs and the number of nodes in them. The shortest was the implementation of databases based on a denormalized document form, i.e. MongoDB and ArangoDB.

Looking at the averages of the results it is possible to say that the fastest solution is MongoDB, except for one measurement of removal, which definitely overestimated the average achieved by the base. Considering the versatility of ArangoDB (it also combines a graph engine), it can be concluded that it is a very competitive implementation of a document solution with the possibility of using graphs. It is important in the analysis of the results to take into account that the database creation process is usually carried out once, subsequent iterations involve modification or extension of the solution. Therefore, despite the considerable development time, graph databases are still a competitive and desirable data storage solution.

References

1. Barczak A., Zacharczuk D. and Korzeniecka A. (2019). The influence of indexing methods on effective functioning of the database. Studia Informatica. System and Information Technology, 17(1-2), 5–18.

2. Skaruz J. (2020). Database security: combining neural networks and classification approach. Studia Informatica. System and Information Technology, 23(1-2), 95–115. https://doi.org/10.34739/si.2019.23.06

3. Li Y. and Manoharan S., "A performance comparison of SQL and NoSQL databases," 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 2013, pp. 15-19, DOI: 10.1109/PACRIM.2013.6625441.

4. Martins P., Abbasi M., Sá F. "A Study over NoSQL Performance". In: Rocha Á., Adeli H., Reis L., Costanzo S. (eds) New Knowledge in Information Systems and Technologies. WorldCIST'19 2019. Advances in Intelligent Systems and Computing, vol 930. Springer, Cham. https://doi.org/10.1007/978-3-030-16181-1_57

5. Tang E. and Fan Y., "Performance Comparison between Five NoSQL Databases," 2016 7th International Conference on Cloud Computing and Big Data (CCBD), 2016, pp. 105-109, DOI: 10.1109/CCBD.2016.030.

6. Pereira D. A., Ourique de Morais W. and Pignaton de Freitas E. NoSQL real-time database performance comparison, International Journal of Parallel, Emergent and Distributed Systems, 33:2, 144-156, 2018, DOI: 10.1080/17445760.2017.1307367

7. Gupta A., Tyagi S., Panwar N., Sachdeva S. and Saxena U., "NoSQL databases: Critical analysis and comparison," 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), 2017, pp. 293-299, DOI: 10.1109/IC3TSN.2017.8284494

8. Zaniewicz N., "Porównanie baz danych MongoDB, Neo4j i ArangoDB z wykorzystaniem opracowanego generatora danych do baz NoSQL", Master Thesis, Siedlce University of Natural Sciences and Humanities, 2019, (in Polish)