Graph Databases: Neo4j Analysis

José Guia[1], Valéria Gonçalves Soares[2] and Jorge Bernardino[1,3]
[1]*ISEC, Polytechnic of Coimbra, Rua Pedro Nunes, Coimbra, Portugal*
[2]*Informatics Centre, Federal University of Paraiba, Brazil*
[3]*CISUC - Centre for Informatics and Systems of the University of Coimbra, Portugal*

Keywords: Graph Databases, NoSQL Databases, Neo4j.

Abstract: The volume of data is growing at an increasing rate. This growth is both in size and in connectivity, where connectivity refers to the increasing presence of relationships between data. Social networks such as Facebook and Twitter store and process petabytes of data each day. Graph databases have gained renewed interest in the last years, due to their applications in areas such as the Semantic Web and Social Network Analysis. Graph databases provide an effective and efficient solution to data storage and querying data in these scenarios, where data is rich in relationships. In this paper, it is analyzed the fundamental points of graph databases, showing their main characteristics and advantages. We study Neo4j, the top graph database software in the market and evaluate its performance using the Social Network Benchmark (SNB).

1 INTRODUCTION

Relational databases are good at managing transactional data and are still prevalent in data storage. However, with the recent growth of applications rich in relationships (e.g., social networks), graphs are becoming the preferred choice as the data model for representing and storing this new type of data. Current platforms must deal with huge amounts of data and the growth of interconnected information (Lourenço et al., 2015a). In the last years, a new database family, the NoSQL model (Deka, 2015; Abramova et al., 2014a, 2014b), has gained widespread popularity, especially because of the need to deal with huge volumes of data connected to each other, to store and to recover them effectively (Lourenço et al., 2015b). So, a special type of NoSQL database arises that fits this paradigm: the *graph databases* (Larriba-Pey et al., 2014).

Graph databases are a type of non-relational databases that provide an effective and efficient solution for the information storage in the current context, where data are very strongly interconnected. Graph databases can be defined as databases that use graph structures with nodes, edges and properties to store data (Wang et al., 2015).

The interest in graph models has been increasing in the last few years, due to their applications in areas like the Semantic Web and the Analysis of Social Networks (Dietrich et al., 2014). The main advantage is the lightning-fast access to complex data, founded per example in social networks, recommendations engines, data mining operations and network systems.

This type of database is too easy to understand because its concept is based on graph theory. This theory is based on graphs (Rodriguez et al., 2010), which are mathematical structures used to model relations between objects. In this context, a graph is a structure organized by nodes, also called vertices (the entities), by edges (the relations) represented by the lines that connect the various nodes and by properties that represent the information related to the nodes and/or with the edges. Therefore, the graph databases can be described simply as a way to represent and store data using their structures: nodes, edges and properties. The simplicity of the storage representation in their structures and quick access to data make graph databases, a very practical database type to use and manage. The graph databases are optimized to store and query graph structures.

The problem of graph databases is that sometimes they are not particularly effective in all desired operations, for example, data representation from the relational models. They are not a general

Guia, J., Soares, V. and Bernardino, J.
Graph Databases: Neo4j Analysis.
DOI: 10.5220/0006356003510356
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 1*, pages 351-356
ISBN: 978-989-758-247-9

replacement for relational databases, but are in fact an efficient solution when dealing with huge volumes of data that contain interconnected data.

The focus of this paper is the study of the characteristics of graph databases, referring to its advantages, and the evaluation of the most popular graph database: Neo4j.

This paper is structured as follows: Section 2 describes the advantages and use cases of graph databases. Section 3 describes the main characteristics of the Neo4j database. Section 4 present Social Network Benchmark (SNB). Section 5 presents the experimental evaluation of Neo4j. Finally, Section 6 presents the main conclusions drawn from this study, as well as proposals for possible future work in this topic.

2 MAIN ADVANTAGES AND USES OF GRAPH DATABASES

Graph databases are effective for several industries from telecommunications to financial services, logistics, hospitality, and healthcare. *Graph databases* are essential in main areas, such as analysis of behavior in social networks, data management, and census-related studies. According to Forrester Research, graph databases is the fastest growing category in database management systems and will reach more than 25 percent of enterprises by 2017. Following, we describe the main characteristics and advantages of a *graph database* (Robinson et al., 2013):

- Information search far more optimized than compared to relational databases, since it takes advantage of the proximity data from one or more root (main nodes) of the *graph database.*
- Quite intuitive, due to their natural form of information representation - the graphs.
- Support the data storage in the order of *petabytes* (10^{15}).
- They are very agile in development since they can be easily adapted over time, either in the insert or in the deletion of information.
- Allow new types of data.
- Suitable for data connected to each other, usually involved in real-world cases.
- Optimized for *data mining* operations.
- High performance in terms of *querying* very deep searches when compared to relational databases.

For example, Walmart and eBay adopt a graph database to understand the behavior and preferences of online buyers with adequate speed and enough depth to make real-time and personalized recommendations (Retail Technology, 2016). By using a graph database, these companies can connect rapidly masses of complex buyer and product data to gain insight into customer needs and product trends.

3 Neo4J ANALYSYS

Neo4j is considered the reference software in this area (Predictive Analytics Today, 2016), and it is one of the most used graph databases in areas such as health, government, automotive production, military area, among others.

Neo4j is an *open-source graph database* implemented in Java. The founders of Neo4j describe it as a fully transactional database, a persistent Java engine where it is possible to store structures in the form of graphs instead of tables (Webber, 2012). The Neo4j is considered the most popular and used *graph database* worldwide, the largest reference area and this is our choice due to this general recognition (Predictive Analytics Today, 2016).

This software was first released in 2007, and is divided into three broad categories: Community, Government, and Enterprise. The Community Edition is the trial version, which is basically the version that any user can test. The Enterprise Edition, where there is the possibility of testing a more complete version than the Community for 30 days. It is therefore the commercial version of this software, and there is still the Government Edition, which is like an upgrade to the Enterprise version. This release is highly focused on government services. The major differences between the two main versions of Neo4j (Community and Enterprise) are: the existence of *online backup,* high performance level of memory cache, detailed monitoring system, strong managing of *locks* on the database, and the greater database scalability, among other advantages of Enterprise Edition.

3.1 Neo4j Main Features

The main advantages of Neo4j are (Bruggen, 2015):

- Horizontal scalability (in the Enterprise version) - allows easily adding more nodes to the system. In the Community version, the scalability is vertical.
- Neo4j has its own language, created by the company for its query methods - Cypher language. It is through this language that we can handle all the information of *graph database.*
- The storage is *disk- based* - through proprietary file systems.
- Its integrity is ACID guaranteed.
- It has a very intuitive and very accessible interface.

In Neo4j interface (see Figure 1) we can see a *graph database* in the central image menu – and the graph with multiple nodes, the various properties, and the edges that make up the *graph database.* On the left side of the interface, we can see the name of the nodes, the edges, the attributes and the version of the program. We can also access through an option on interface, visualize the size of the disk *graph database,* follow various program tutorials, and access the specific settings of the program, among other options. In the center, above the *graph database,* we can find a place to write queries in Cypher language.

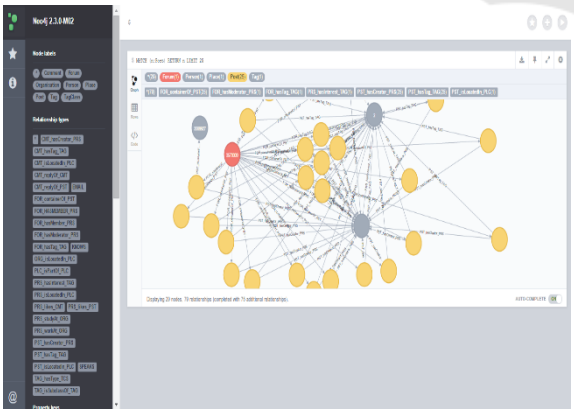

Figure 1: Neo4j interface (source: http://neo4j.com).

3.2 Query Methods

The query methods in Neo4j are written in Cypher language that is the query language created by Neo Technology (Vukotic et al., 2014).

The Cypher language allows an expressive and efficient query execution and update of graph database. It is a relatively simple language, but very powerful. Very complex queries running in a relational database can be easily executed in Cypher. This allows users to focus on your domain instead of getting lost in database access.

With this query language, we can delete/ insert/create, the basic elements such as nodes, edges/relations and properties. For example, the code to create a node with a name is:

CREATE (n: Person);

This code creates a node n, named Person, using the CREATE command.

4 SOCIAL NETWORK BENCHMARK (SNB)

In the experimental evaluation, it is used the Social Network Benchmark (SNB), created by LDBC (Erling et al., 2015). The SNB models a social network like Facebook. The *dataset* consists of people connected each other in a network relationship, where most of the data relates to messages that people post in various forums.

To accomplish the tests, we generated two large data sets. Then we show the settings used for the different tests in generating data on SNB. The modified parameters were: scaleFator (which is the scale factor of the data to be generated); the serializer (which is the format that the generated files out - supports the following: .csv, .ttl, .csv_merge_foreign); the compressed (that specifies whether the generated files must be compressed or not); the numThreads (which is the number of threads to use); updateStreams (option for DATAGEN generate streams updated to use) and outputDir (which is the location where the files to be generated will be stored on disk). Following we describe the two datasets that were generated with in the SNB. Besides the datasets, we also will describe the time to generate these files on the SNB.

In the first test, we generated one dataset with about 1.2 GB of data (Dataset 1). This test takes 11 minutes and 54 seconds. In the second test, we generated one dataset with about 11.6 GB of data (Dataset 2). This test takes 1 hour, 52 minutes and 26 seconds.

The two generated datasets have eight main entities, along with four other entities that are as subclasses of the main entities. Relations connect existing entities. The Tail and Head entities should

be understood as the entities connected between connection(s), and one is the end of the call and the other is the start of the relationship, respectively. Therefore, the relationship is oriented.

5 EXPERIMENTAL EVALUATION

The experimental evaluation analyzes the Neo4j behavior with different sizes of datasets, presenting their load times and showing the performance using SNB queries.

The experiment was executed with the following configuration:

- Operating System: Windows 8.1 - 64bit;
- Processor: Intel (R) Core (TM) i7-3630QM CPU @ 2.40GHz.

The tests were executed in order to load the two datasets generated by the SNB - upload them one by one to the Neo4j, then in each of the datasets loaded, we run the queries test. Each dataset has 33 .csv files - that are representative files of entities and relationships between these entities. Two types of tests were made with Neo4j. The first assessed the load time of the two datasets created by the SNB, as mentioned above. After that, we evaluate the performance of Neo4J executing the SNB benchmark queries.

5.1 Loading Time in Neo4j

In this step, we uploaded the files generated in the SNB in the *Dataset 1* that initially had 1.2 gigabytes of data. At the end of loading, the original data, which were generated by the SNB, increased from 1.2 *gigabytes* to 5.4 *gigabytes* in Neo4j when "built" in *graph database*. This variation corresponds to an overhead of 4.5 times compared to the original size, so there was an increase of around 450%. The total loading time of these files was 5h:12m: 24,81s.

In the second test, we load the files generated in the *Dataset* 2 that has about 11.6 *gigabytes* of data. The data that were generated by the SNB increased from 11.6 *gigabytes* to 52.35 *gigabytes* in Neo4j. A growth of about 451% compared to the size of the files generated SNB. It is important to note that during the loading time, the system blocked, and we had to start it again, increasing the total final time. The total loading time of this dataset was 160h: 28m: 4,74s.

In summary, we can say that relatively to the Dataset 1, the Dataset 2 took much longer to load all the files to the Neo4j. It went up for five hours in the first dataset for approximately 160 hours in the second dataset. This corresponds to an overhead of about more 32 times to complete the load of all files. This situation occurs because the volume of data has increased considerably in the second dataset.

5.2 Query Example

After completion of the Cypher language commands to load the entities and relationships, we execute the SNB queries in the two *datasets,* to assess the runtimes. For example, **Query 13**, which will look for people who know each other, traversing the shortest possible route in the graph database. It returns the length of this path taken in the search, and the code in Cypher is:

```
MATCH (p1: Person), (p2: Person)
OPTIONAL MATCH shortestPath path =
((p1) - [: KNOWS] - (p2))
RETURN path CASE IS NULL
WHEN THEN true -1
ELSE length (path)
END THE pathlength
```

We execute the 14 queries of the SNB's manual. It is through this type of test queries for data query the database graph that makes assessments such as the time it takes to query the data in the graph database (from: https://github.com/ldbc/ldbc_snb_docs).

Our tests were based on exactly the time that these queries are executed to return the desired results. In the next section we show the results in the evaluations using Neo4j and the two datasets generated by SNB benchmark.

5.3 SNB Queries Execution Time

In this section, we present the results of executing the 14 queries of the SNB benchmark using the two datasets. We execute all the queries four times and take the average of last three runs to eliminate cold-start.

We executed the 14 test queries in the Dataset 1, (when coming from SNB it had 1.2GB and then when it was built in the graph database Neo4j, it increases to 5.4 GB) and obtained the results shown in Table 1.

Table 1: SNB queries execution time of Dataset 1.

Queries Test	*Execution time (s)*
QUERY 1	19,27
QUERY 2	5,454
QUERY 3	41,221
QUERY 4	1,321
QUERY 5	23,720
QUERY 6	42,973
QUERY 7	8,037
QUERY 8	3,276
QUERY 9	12,169
QUERY 10	22,493
QUERY 11	0,270
QUERY 12	2,717
QUERY 13	50,328
QUERY 14	68,398
Total	**5m 1,65s**

It is noticed that Queries 6, 13 and 14 are the ones that took longer to perform, as they require more processing. This is due to the code they contain and calculations they have to make to the graph database. For example, calculations of shorter paths between certain nodes and entities. From Table 1, the average execution time of Dataset 1 was 5 minutes and 1.65 seconds.

After that, we run the same 14 *queries* test in Dataset 2 (when coming from SNB had 11.6GB and then when it was built in *graph database* in Neo4j, it increases to 52.35 GB). The results obtained are shown in Table 2.

Table 2: SNB queries execution time of Dataset 2.

Queries Test	*Execution time (s)*
QUERY 1	80,765
QUERY 2	92,978
QUERY 3	117,483
QUERY 4	18,04
QUERY 5	60,95
QUERY 6	119,174
QUERY 7	54,926
QUERY 8	49,927
QUERY 9	56,172
QUERY 10	70,153
QUERY 11	22,512
QUERY 12	23,001
QUERY 13	122,408
QUERY 14	129,338
Total	**16m 57,83s**

As previously observed with Dataset 1, the queries 6, 13 and 14 are the ones that took longer to run. The reason is because are the queries which have more calculations in the graph database. It was also found that in the Dataset 2, queries 2, 4, 7, 8, 11 and 12 were the queries that were executed in seconds. Although they take more time to search, comparing the times shown in the Dataset 1, but it is normal given the size that the graph database, which is about 52 gigabytes of data.

As we can see in Table 2, the total execution time for the 14 queries using the Dataset 2, is 16 minutes and 57.83 seconds.

6 CONCLUSIONS AND FUTURE WORK

The concept of non- relational databases has been growing in popularity and usability. The NoSQL databases bring advantages over the previously established databases, such as the manipulation of huge volumes of connected data. The NoSQL databases are divided into different models, each model with a set of features and enhancements (Abramova et al., 2015). Our study focused on only *graph databases*.

Our analysis allows to conclude that there are three very similar databases and with very similar characteristics, but still Neo4j stands out for its simplicity, despite the need to have prior knowledge of the Cypher language to create and manage any information in graph databases of Neo4j. Its interface also makes Neo4j one graph database reference for its accessibility. Our assessment allows to say that the software more robust and more practical is the Neo4j.

The experimental evaluation considered the performance of the data loads Neo4j to form a graph database, as well as testing the performance in terms of certain databases searches to those previously created graph. Accordingly, we tested the different times of loading the files, which were generated in the benchmark, and the different runtimes of created surveys conducted to graph databases. With this analysis, we can get the knowledge of how the Neo4j handles loading external data to the software along with the understanding of how the software behaves the level of performance when it has to consult certain information in a database. The results showed that the Neo4j, is a powerful tool, and according to the tests performed, we concluded that the software has a fairly acceptable behavior when dealing with different sizes of graph databases, as in our case, the various datasets we tested. We note that

where the software loses much time is on loading the files to the Neo4j, getting later in graph format. The data presented the first dataset (with 1.2GB) showed speed in loading files, since they did not have any file with more than 800MB to be loaded. In the case of the second dataset, which was lost over time, since there was files being uploaded that took almost one day to be completely loaded into the Neo4j. We can therefore say that the software has a great behavior in loading files, with up to a size of 700 to 800MB, because above this value, it is time consuming this process, as we proved with the Dataset 2. Another important aspect that also tested it was performance-level searches.

Using the test queries withdrawn in SNB benchmark, one can see that in the two datasets where it loses more time is in the information query in the initial execution. It happens because the graph database leverages one of its main features that is the storage engine, that is optimized due to the fact that it stores adjacent registers by direct references, thus making access to quickly plays data in the next executions.

It is normal for the amount of information that has to go through that in a graph database with a huge volume of data to take longer in a given query test that a graph database with little information running the same query test and that this present almost immediately the respective output. One drawback encountered in Neo4j is their instability when it has to deal with a large volume of data, if the Dataset 2 (which in order to be all loaded, it became to the size of approximately 52GB) blocking often the system and causing the restart to load the data.. We cannot be sure if this issue was related to the Neo4j, or with any restrictions the hardware and also software of the machine where the tests were performed.

As future work, we intend to analyze the loading of files and query times in other graph databases.

REFERENCES

A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, J. Partner, 2014, “Neo4j in Action”, Book Neo4j in Action, 2014.

G. C. Deka, 2015, “Tutorial on NoSQL Databases”, Mobile Cloud Computing, Services, and Engineering (MobileCloud) IEEE International Conference, San Francisco, USA, April 2015.

I. Robinson, J. Webber and E. Eifrem. Graph Databases. O‟Reilly Media Inc., California, 2013

J. Dietrich, N. Jones and J. Wright, 2008, “Using social networking and semantic web technology in software engineering – Use cases, patterns and a case study”, Massey University, Institute of Information Sciences and Technology, Palmerston North, New Zealand, January 2008.

J. L Larriba-Pey, N. Martínez-Bazán, D. Domínguez-Sal, 2014, “Introduction to Graph Databases”, Reasoning Web. Reasoning on the Web in the Big Data Era Volume 8714 of the series Lecture Notes in Computer Science pp 171-194, 2014.

J. R. Lourenço, V. Abramova, M. Vieira, B. Cabral, J. Bernardino, “Nosql databases: A software engineering perspective”, New Contributions in Inform. Systems and Technologies, Springer, pp.741-750, 2015.

J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, J. Bernardino, “Choosing the right NoSQL database for the job: a quality attribute evaluation”, Journal of Big Data, Vol 2: 18, 2015.

J. Webber, 2012, “A programmatic introduction to Neo4j”, SPLASH’12, pages 217-218, ACM New York, USA, 2012.

M. A. Rodriguez, Neubauer, P., 2010, “Constructions from Dots and Lines” Bulletin of the American Society for Information Science and Technology, American Society for Information Science and Technology, volume 36, number 6, pages 35-41, August 2010.

O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M. Pham, and P. Boncz. The LDBC Social Network Benchmark: Interactive Workload. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). ACM, New York, NY, USA, 619-630.

Predictive Analytics Today, 2016, “Top 31 Graph Databases”, http://www.predictiveanalyticstoday.com /top-graph-databases/, accessed on 29th November, 2016

Retail Technology, “Walmart and eBay adopt graph database “http://www.retailtechnology.co.uk/item.php ?news_id=5187, accessed on 23-11-2016

Rik Van Bruggen, 2015, Learning Neo4j, Packt Publishing.

Shao-Ting Wang, Jennifer Jin, Pete Rivett, and Atsushi Kitazawa, “Technical Survey Graph Databases and Applications”, International Journal of Semantic Computing 2015 09:04, 523-545

V. Abramova, J. Bernardino, P. Furtado, “Experimental evaluation of NoSQL databases”, International Journal of Database Management Systems, Vol 6 (3), 2014.

V. Abramova, J. Bernardino, P. Furtado “Testing Cloud Benchmark Scalability with Cassandra”, 2014 IEEE World Congress on Services, pp. 434-441.

V. Abramova, J. Bernardino, P. Furtado, “SQL or NoSQL? Performance and scalability evaluation”, International Journal of Business Process Integration and Management, Vol 7 (4), pp. 314-321, 2015.