A Selection Process of Graph Databases based on Business Requirements

Víctor Ortega[1], Leobardo Ruiz[1], Luis Gutierrez[1], Francisco Cervantes[1]

[1] ITESO Jesuit University of Guadalajara,
Department of Electronics, Systems and Information Technology,
45604 Tlaquepaque, México
{vortega, ms716695, lgutierrez, fcervantes}@iteso.mx

Abstract. Several graph databases provide support to analyze a large amount of highly connected data, and it is not trivial for a company to choose the right one. We propose a new process that allows analysts to select the database that suits best to the business requirements. The proposed selection process makes possible to benchmark several graph databases according to the user needs by considering metrics such as querying capabilities, built-in functions, performance analysis, and user experience. We have selected some of the most popular native graph database engines to test our approach to solve a given problem. Our proposed selection process has been useful to design benchmarks and provides valuable information to decide which graph database to choose. The presented approach can be easily applied to a wide number of applications such as social network, market basket analysis, fraud detection, and others.

Keywords: Graph Databases, Benchmarking, Selection process

1 Introduction

In the last decade, the large amount of information generated from multiple data sources such as social networks and mobile devices has led the relational database management systems to their limit. As larger the dataset, it becomes more difficult to process using traditional data processing applications like relational database management systems and data warehousing tools. The challenges include analysis, capture, curation, search, sharing, storage, transfer, and visualization [1]-[3]. This fact has driven to look for alternatives such as non-relational databases. NoSQL databases provide several options for storing, accessing and manipulating data at big scale; examples are key-value, document-based, column-based, and graph-based stores, all of these improves the querying performance that relational SQL databases have on large and unstructured datasets [4].

One of the most promising approaches is the Graph Database (GDB). This kind of databases includes support for an expressive graph data model with heterogeneous vertices and edges, powerful query and graph mining capabilities, ease of use, as well as high performance and scalability [5]. All these features are needed to manipulate and

study highly connected data points. Other advantages of graph database systems include but not limited to the following topics: high-performance query capabilities on large datasets, data storage support in the order of petabytes (10^{15}), intuitive query language, appropriate for agile development, support of new types of data and suitable for irregular data, and optimized for data mining operations [6]. Overall, these features can be classified into three major areas: performance, flexibility, and agility.

Performance is a relevant factor when it comes to data relationship handling. Graph databases massively improve the performance when dealing with network examination and depth-based queries traversing out from a selected starting set of nodes within a graph database. They outperform the needed depth of join operations, which traditional relational databases implement in languages such as SQL when they need to go into a complex network to find how data relates to each other.

Flexibility plays an important role; it can allow data architects to modify database schemas, which can quickly adapt to significant changes because of new business needs. Instead of modeling a domain in advance, when new data need to be included in the graph structure, the schema can be updated on the fly as data are pushed to the graph.

Agility is highly appreciated due to most software companies adopting work methodologies such as Scrum and Kanban. This graph database feature aligns with such lean interactive development practices, which lets the graph evolve with the rest of the application and evolving business needs.

From an operational perspective, there are three types of graph database: true graph databases, triple stores (also named RDF stores), and conventional databases that provide some graph capabilities. True graph databases support index-free adjacency, which allows graph traversals without needing an index, while triple stores require indexing to perform traversals [7]-[8]. True graph databases are designed to support property graphs where these properties are applied either to vertices or edges, and recently some triple stores have added this capability.

Most of the graph databases support a unique main query language, which allows us to include scenarios such as language type, ease of usage, SQL similarity [9], and evaluate graph data and queries portability across other platforms.

So far, we have described the benefits of using graph databases to address certain types of problems where it is necessary to analyze large amounts of highly related data. However, for companies, it is not trivial to identify which GDB is better according to their needs. Thus, the selection of a GDB becomes a challenge that can impact at an operational level and even at an economic level.

In this work, we propose a new selection process for guiding analysts to select a suitable GDB to satisfy the business needs. The selection process flow consists of five main stages: the problem analysis, requirements analysis, GDB analysis, benchmarking, and GDB selection.

Our contributions are 1) a new selection process to select a graph database aligned to business requirements, 2) a detailed description of the proposed selection process in a particular scenario.

We organized the rest of this document as follows: Section 2 describes how other studies benchmark graph databases. Section 3 presents and describes in detail our proposed selection process to choose a graph database. Section 4 shows the application

of the selection process on a case study with two datasets and their results. Finally, in Section 5, we present our conclusions.

2 Related Work

There are several commercial GDB engines that enterprises and organizations could use. However, choosing the best option is not trivial. Several authors as Batra et al. [10] and Nayak et al. [11] have described the advantages of NoSQL databases and how those are solving common issues on legacy systems which rely on Relational Databases (RDB). These works provide an overall description of what graph databases are offering as a solution and remark the advantages and disadvantages of each type of systems.

Batra in [10] performs a comparative between MySQL and Neo4j. It provides the following evaluation parameters: level of support/maturity to know how well tested the system is, security to understand what mechanisms are available to manage restrictions and multiple users, flexibility which allows system designers to extended new features through time-based on schema changes, and performance-based upon predefined queries on a sample database. The obtained results are that both systems performed well on the predefined queries with slight advantages for Neo4j, the graph databases are more flexible than relational databases without the need to restructure the schemas.

Nayak in [11] describes each type of NoSQL database type. These are classified into five categories, one of them being Graph databases, the categories are key-value store databases such as Amazon DynamoDB and RIAK, column-oriented databases like Big Table and Cassandra, document store databases using MongoDB and CouchDB as examples, and graph Databases where Neo4j is the only analyzed system. No specific criteria were used to compare features. However, the paper provides the advantages and disadvantages of each of the data stores. Some of the advantages of NoSQL over Relational are that they provide a wide range of data models to choose, easily scalable, DB admins are not required, some provide handlers to react to hardware failures, they are faster, more efficient and flexible. The disadvantages of NoSQL over Relational include that they are immature in most of the cases, no standard query language; some of them are not ACID compliant, no standard interface, maintenance is difficult. Nayak provides a proper classification of NoSQL databases and Batra a usefully comparative between an RDB and Graph database, but any of them provide a formal process to select a graph database aligned to business requirements.

There are other works, which have compared only graph database systems [12]-[14]. The first one is providing a high-level description of AllegroGraph, ArangoDB, InfiniteGraph, Neo4j, and OrientDB [12]. The criteria, considered in work use the following elements: Schema flexibility, Query Language, Sharding (ability to break up large datasets and distribute it across several replicated shards), Backups, Multi-model, Multi-architecture, Scalability, and Cloud Ready. Each of the elements is graded for all the databases with a score which can be Great, Good, Average, Bad or Does not support. The obtained results are just remarking that there is no perfect graph database for any type of problem. An important element which is not evaluated in this work is the performance on graph operations through the use of their supported query languages.

The other works comparing only graph databases that go beyond the high-level features and evaluate performance tend to have the support of a specific development group or vendor. One of them is a post from Dgraph comparing the loading data capacities and querying performance against Neo4j [13], the dataset is in RDF format, which is not supported natively by Neo4j. Therefore, it had to be loaded through data format conversions. Then a set of specific queries are executed on each database where Dgraph outperforms Neo4j in most cases. The other graph comparison work is benchmarking TigerGraph, Neo4j, Neptune, JanusGraph, and ArangoDB [14], the work is developed by the creators of TigerGraph, the comparison includes data loading times, storage size, and the response of graph queries for full graph or subsets of it. It uses a couple of graphs, one which is considered a small size and then a large dataset. The conclusion favors TigerGraph in most of the compared features claiming that their development is based on a natural, real-time, and Massively Parallel Processing platform approach with a high-level language introduced by the platform. Even though the comparison presents some useful metrics, it does not provide a guide to choose the suitable GDB according to the business requirements.
As far as we know, in the literature, there is not a transparent process for the election of a GDB driven by the problem, goals, requirements, and constraints. Our work has no intention of favoring any of the evaluated systems, and it provides a guide to choose the suitable GDB according to business needs.

Nonetheless, some works focused on selection processes for other types of software tools such the proposed by Maxville et al. [15], where they implement a framework for the selection of open source software from the repositories. The context-driven component evaluation framework (CdCE) uses a specific set of templates, criteria, ontologies, and data representations, with classifiers and test generators for filtering and evaluation. The proposal has three main phases: filtering, evaluation, and ranking items.

Lee et al. [16] developed a tool based on the Analytic Hierarchy Process (AHP) to adopt open-source software. AHP is a decision-making method that includes qualitative and quantitative techniques to support decisions. Their selection process has four phases: 1) identify the goal/project, selection criteria and product alternatives, 2) justify judgment matrix for each criterion, 3) justify judgment matrix for the alternative products, and 4) results for the final ranking and decision making.

Lourenço et al. [1] presented a qualitative evaluation of NoSQL databases and described the main characteristics of each type of databases based on the literature review. This paper focused on performance requirements of the databases and quality attributes like availability, consistency, durability, maintainability, read and write performance, reliability, robustness, scalability, stabilization time, and recovery. Finally, Lee presents a timeline of the evolution of NoSQL databases.

The last works above provide a set of stages, metrics, and use cases as a guideline to choose diverse kind of technology, but are not focused on graph databases. Therefore, it is required a well-defined selection process of graph databases based on business requirements.

3 Selection Process of GDB

In this section, we describe a new selection process that guide analysts to select a GDB that satisfies the business needs. The selection process flow consists of five main stages: problem analysis, requirements analysis, GDB analysis, benchmarking, and GDB selection (Figure 1).

3.1 Problem analysis (Stage 1)

The goal of this stage is to define the problem that must be solved using a GDB. This stage has two steps: the problem and goals definition. We assume that, in general, the problem consists of analyzing a large amount of highly connected data.

In the problem definition step (step 1.1), it must be described the business needs in terms of the GDB scope. In this step, the analyst should answer the following question: why do I need a GDB?

Once the analyst defined the problem, he must determine a set of goals in terms of the capabilities that a GDB can offer, for example, the supported amount of processing data, the kind of analysis to perform on the connected data, filters, dynamic analysis, graph algorithms, and memory.

3.2 Requirements analysis (Stage 2)

Based on the established goals, it is required to define a set of requirements (step 2.1) that can be quantitative and qualitative that allow analysts to evaluate them throw metrics. Requirements can be of three types: performance, flexibility, and agility (described in Section 1).

Furthermore, the analysts could identify a subset of hard requirements (step 2.2). In this work, hard requirements are called constraints. The GDB must satisfy these constraints to become a candidate in the pre-selection stage. Some examples of constraints are to support: declarative query languages, specific programming languages, and particular graph algorithms.

Besides, the analysts must determine the metrics to measure each one of the requirements previously defined (step 2.3). Some metrics examples are load data efficiently, storage efficiently, and natively graph algorithms execution.

In order to assign a score to each GDB, we suggest a scoring function. The analyst must assign a weight for each metric (each metric is related to one requirement). The score is obtained by summarizing the product of the weights and the normalized metrics results, as it is shown in equation 1.

$$S_d^c(M, W) = \sum_{i=1}^{n} w_i \,||\, m_i \,||, \tag{1}$$

where S_d^c means the score of evaluating a candidate GDB denoted by c for a given dataset d. The function has as input a list (M) of evaluation results of each metric ($m_i \in M$), and a list of weights (W) that determine the importance of each metric. Besides, the weights have the following property $\sum_{i=1}^{n} w_i = 1$ where $w_i \in W$, $n = |W|$ and each w_i is a value between 0 to 1.

$$\|m_i\| = \begin{cases} \dfrac{m_i - \min(M)}{\max(M) - \min(M)} & maximize\ m_i \\ 1 - \dfrac{m_i - \min(M)}{\max(M) - \min(M)} & minimize\ m_i \end{cases} \quad (2)$$

In this work, we suggest using the min-max normalization or the complement of this normalization in the case of minimizing the use of some resource as it is described in equation 2. This normalization function results in 0 if it is the lowest score and 1 to the highest score. This function will be used in the GDB selection stage (stage 5). Furthermore, the score function and the normalization function can be customized according to the problem to solve.

3.3 GDB Pre-selection (Stage 3)

In this stage, the inputs are a set of constraints defined in the previous stage (stage 2) and a GBD catalog. The catalog consists of all those databases that analyst believe can solve the problem defined in stage 1. Only the GDBs that accomplish all the constraints will be part of the pre-selected GDBs which are going to be considered for the benchmark. The pre-selected GDB catalog will be the output of this stage and input for the next one.

3.4 Benchmarking (Stage 4)

This stage has the pre-selected GDB catalog and the metric definition for each requirement as inputs. The purpose of the stage is to search or build datasets useful to evaluate the GDBs with the metrics established in step 2.3. Moreover, in this stage, the environment of the benchmark has to be defined and configured, such as operating system, RAM, GDB configuration, create indexes on data, among others.

On the other hand, to guarantee a fair comparison between GDBs, it is required to design and implement a set of tests that make possible to use the same set of data for all pre-selected GDBs; for example, a loading performance test requires that all the GDBs load the same amount of data in all tests.

To automate the evaluation of the metrics for each GDB, we suggest the use of some scripts that execute the benchmarks and gather the information of each metric for all the scenarios.

The output of this stage is a table per scenario with the results of the evaluation for each metric for all GDB.

3.5 GDB Selection (Stage 5)

In this stage, the inputs are the results of the evaluations performed over the GDBs of the previous stage, and the scoring function defined in stage 2. Once the scoring function has been applied to all GDBs with each dataset, the final score per GDB is computed by averaging the result of all scenarios(datasets). Hence, it is obtaining a single value per GDB, which is stored as an ordered list of scores and sorted from maximum to minimum. The highest score indicates that the GDB accomplish better the established requirements. Finally, the analysis has all the information to choose one GDB that will be used to solve the problem defined in stage 1.

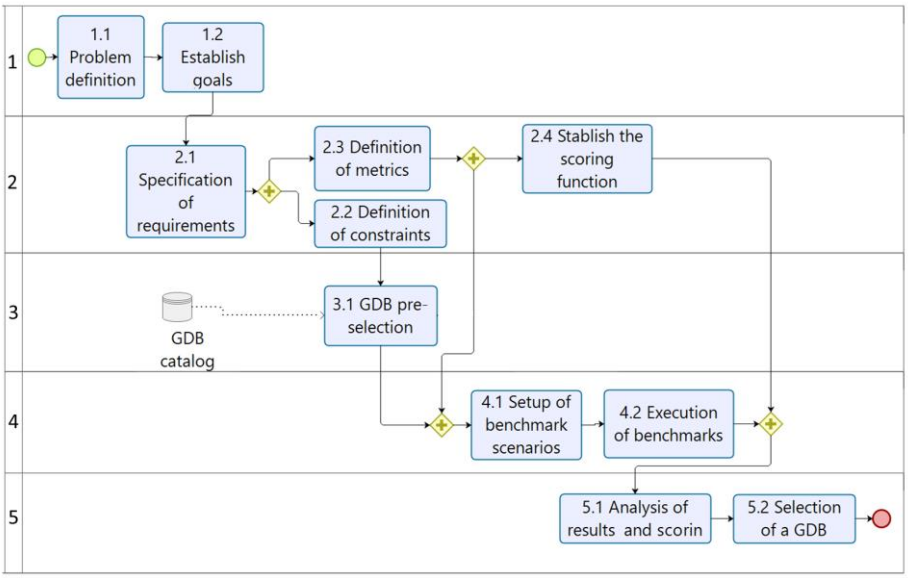

Fig. 1. The process to select the GDB that suits best the business requirements. It is composed of five stages: Problem Analysis (1), Requirement Analysis (2), GDB Pre-selection (3), Benchmarking (4), and GDB Selection (5).

4 Case Study and Results

To show how the selection process of GDB can be applied in a specific case, each stage of the process is described step by step until we can identify the most appropriate GDB to solve a defined problem.

In stage 1, the problem has to be specified by answering the question of why the GDB is needed? In our case study, the problem consists of analyzing a social network to identify the weakly connected components, to obtain the most influential users, and the possibility to explore the neighborhood of some given node. In the next step of this stage, the goals need to be declared. The goals extracted from the previously defined problem are: to store data extracted from a social network, identify weakly connected

components, identify opinion leaders or critical users, and to explore the neighborhood of the elements in the social network.

Stage 2 starts with the previously defined goals, and the requirements are specified. In this case study, the requirements are related to GDB performance, as listed in Table 1. However, it is possible to include requirements associated with flexibility and agility.

Table 1. List of performance requirements with its corresponding metrics and weights used in the case study.

Requirements of performance	Metric	Weights
large-scale data loading	Loading time (min.)	25%
use of disk storage	Disk usage (GB)	20%
Obtain the neighborhood in 1 hop	execution time (sec.)	5%
Obtain the neighborhood in 2 hops	execution time (sec.)	5%
Obtain the neighborhood in 3 hops	execution time (sec.)	5%
Execution of weakly connected components	execution time (sec.)	20%
Execution of PageRank algorithm	execution time(sec.)	20%

Furthermore, in this stage, define the constraints to pre-select the GDB catalog. In this case, the restrictions for GDB are to be an open-source or free, provide a declarative query language, support Java as a programming language, and compute graph algorithms like neighborhood to k-hops, PageRank, and weakly connected components.

Another step (2.3) in the same stage is the definition of the metrics. For this example, we define the metrics in terms of execution time and the required storage space, as shown in Table 1.

Finally, in this stage, the score function is established. In this case study, we use the function suggested in section 3, and the associated weight for each requirement is shown in Table 1.

Table 2. Results of the constraints evaluation on the initial GDB catalog. The first three GDBs accomplish all the constraints. Therefore, these GDBs are pre-selected for the next stages of the selection process.

Graph database	Graph query language	Native graphs triplets	Graph algorithms	Supports Java	Open-source
JanusGraph	X	X	X	X	X
Neo4j	X	X	X	X	X
TigerGraph	X	X	X	X	X
GraphDB	X	X	-	X	X
Mark Logic	X	X	-	X	X
SQL server	-	-	-	X	X

In stage 3, for demonstration purposes, we chose six accessible databases that support graphs. The databases are JanusGraph, Neo4j, TigerGrpah, GraphDB, MarkLogic, and Microsoft SQL. All databases have to satisfy the constraints of the previous stage. In Table 2, it can be observed that only JanusGraph, Neo4j, and

TigerGraph accomplish all constraints, and these can be considered as the pre-select GDBs to be evaluated in the next stage.

In stage 4, the scenarios for evaluation must be defined and configured. For this study, the benchmark setup has been based on the TigerGraph project [9]. The benchmark is conformed of three main steps: the setup, the data loading tests, and the query performance tests.

In this work, we use two datasets published by TigerGraph [9]. The first is a real dataset of the social network Twitter, and the second is a synthetic dataset. The used datasets in this work are described in Table 3.

Table 3. Datasets description used to benchmark pre-selected GDBs.

Dataset	Description	Nodes	Edges	Raw size (MB)
Twitter	A directed graph of social network users.	41.6 M	1.47 B	24375
Graph500	Synthetic Kronecker graph	2.4 M	67 M	967

Table 4. Benchmarks results using the Twitter dataset. The last three columns are the outcomes of metric evaluation on pre-selected GBDs. The last row shows the score computed for each GDB.

Metric	JanusGraph	Neo4j	TigerGraph
Loading time (min.)	238.400	74.000	40.300
Disk usage (GB)	33.000	30.000	9.500
K-Neighborhood (sec.)	1.420	0.377	0.017
Two-Hop K-Neighborhood (sec.)	40398.000	5.260	0.330
Three-Hop K-Neighborhood (sec.)	1600.600	99.700	2.900
Weakly connected components (sec.)	86400.000	1545.100	47.900
Page rank query (sec.)	86400.000	614.900	166.030
Score (0 - 1)	0.000	0.763	1.000

The Twitter dataset has over 41 million nodes and almost 1.5 billion edges; the dataset represents real user connections taken from the social network while Graph500 is a synthetic graph dataset which has 2.4 million nodes and 67 million connections across the data.

Table 5. Benchmarks results using the Graph500 dataset. The last three columns are the outcomes of metric evaluation on pre-selected GBDs. The last row shows the score computed for each GDB.

Metric	JanusGraph	Neo4j	TigerGraph
Loading time (min.)	19.200	6.000	3.700
Disk usage (GB)	2.500	2.300	0.500
K-Neighborhood (sec.)	0.195	0.018	0.003
Two-Hop K-Neighborhood (sec.)	13.950	4.400	0.066
Three-Hop K-Neighborhood (sec.)	1965.500	58.500	0.405
Weakly connected components (sec.)	1491.400	65.500	2.880
Page rank query (sec.)	2279.051	31.200	12.800
Score (0 - 1)	0.000	0.752	1.000

In both cases, the graph's raw data is formatted as a single tab-separated edge list, and no properties or attributes are attached to edges.

In order to obtain a reliable result, it is required to perform a fair comparison between GDBs. The first condition in order to satisfy such equity is to install the systems on the same hardware and software conditions.

Once the benchmark is set up, it is essential to select an environment which satisfies minimum hardware and software requirements for each database, and this can be checked on each database official requirements publication. Our experiments run on Amazon EC2 instances; each system uses the same number of CPUs, memory, and network bandwidth. The selected operating system must support a native installation of all pre-selected GDBs. In this case, we use Ubuntu 18.04 LTS. Once the databases are configured, the scripts are executed. The results of the benchmark are shown in Table 4 for the first scenario (Twitter), and Table 5 for the second scenario (Graph500). In the case of the Twitter dataset, for JanusGraph, we stopped the execution of the algorithms Page Rank and weakly connected components after 24 hours.

Finally, in stage 5, the score function suggested in section 3 is applied to the normalized values, which are the results from the previous stage. Therefore, we obtain, for each scenario, a score per GDB as can be seen in the last row of Tables 4 and 5.

The resulting scores of both tables are averaged, and thus giving a final score that evaluates the GDB in the general point of view. In our case study, the final scores of the pre-selected GDBs are JanusGraph 0.000, Neo4j 0.758, and TigerGraph 1.000. Therefore, according to these results, TigerGraph accomplishes better the requirements. Thus it is selected to solve the defined problem.

5 Conclusions

While getting information about the most popular graph database engines, we have realized that it is common that new products appear and try to get a piece of this evolving market. The proposed selection process shows that interested people in graph database systems can get a quick idea of the strengths and limitations that each database exposes. Our work can help analysts to understand these pros and cons and help them in the process of using an adequate product based on their needs.

The proposed selection process consists of the following stages: problem definition, requirements analysis, GDB pre-selection, benchmarking, and finally, the GDB selection.

Our novel selection process is a useful guide for who needs to evaluate and select a GDB, and it is based on metrics that make possible to ensure that the requirements are fulfilled.

We applied the selection process in a case study of the social network domain, using two datasets to perform the benchmark. Besides, we described how to apply each step of the process until obtaining the final score, which indicates the best GDB for our problem.

The process is flexible enough to be applied to several problems related to a large amount of highly connected data.

As future work, we consider the possibility of implementing a software tool for the (partial) automation of the proposed selection process. The tool can include a set of standard requirements with their corresponding metric. Moreover, we would like to consider the possibility to create building blocks by specifying the input, process, and output. These building blocks may be customized for each GDB and can be used with a friendly user interface where these blocks can be dragged and dropped during the benchmark definition. On the other hand, we would like to test our selection process using FOAF-based dataset in order to benchmark GDB against RDF graph databases.

References

1. J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation," Journal of Big Data, vol. 2, no. 1, p. 18, Aug. (2015)
2. Alazar Baharu and Durga Prasad Sharma, "Performance Metrics for Decision Support in Big Data vs. Traditional RDBMS Tools & Technologies" International Journal of Advanced Computer Science and Applications(IJACSA), 7(11), (2016)
3. Cattell, R. "Scalable sql and nosql data stores," SIGMOD Rec., vol. 39, no. 4, pp. 12–27, May (2011)
4. J. Han, E. Haihong, G. Le, and J. Du.: Survey on NoSQL Database. In: 6th International Conference on Pervasive Computing and Applications, pp. 363--366. IEEE, Beijing, China, (2011)
5. M. Junghanns, A. Petermann, M. Neumann, E. Rahm.: Management and Analysis of Big Graph Data: Current Systems and Open Challenges. Handbook of Big Data Technologies. Springer, Sydney (2017)
6. Guia, J., Soares, V., Bernardino, J.: Graph Databases: Neo4j Analysis. In Proceedings of the 19th International Conference on Enterprise Information Systems, pp. 351--356, (2017)
7. J. Hayes, C. Gutierrez.: Bipartite graphs as intermediate model for RDF. In International Semantic Web Conference. Springer, Berlin, Heidelberg, p. 47--61, (2004).
8. R. Angeles, C. Gutierrez.: Querying RDF data from a graph database perspective. In: European Semantic Web Conference. Springer, Berlin, Heidelberg, pp. 346--360, (2005).
9. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D. A.: Comparison of a Graph Database and a Relational Database: a Data Provenance Perspective. In: Proceedings of the 48th annual Southeast regional conference, pp. 42, ACM, Mississippi(2010)
10. Batra, S., Tyagi, C.: Comparative Analysis of Relational and Graph Databases. In: International Journal of Soft Computing and Engineering, pp. 509--512, (2012)
11. Nayak, A., Poriya, A., Poojary, D. Type of NoSQL databases and its comparison with relational databases. In: International Journal of Applied Information Systems, pp. 16--19, (2013)
12. Fernandes, D., Bernardino, J. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In: Proceedings of the 7th International Conference on Data Science, Technology and Applications, pp. 373--380, Porto, Portugal (2018)
13. Neo4j vs. Dgraph – The Numbers Speak for Themselves, https://blog.dgraph.io/post/benchmark-neo4j/

14. Benchmarking Graph Analytic Systems: TigerGraph, Neo4j, Neptune, JanusGraph, and ArangoDB, https://www.tigergraph.com/benchmark/
15. V. Maxville, J. Armarego, and C. P. Lam, “Applying a reusable framework for software selection,” IET Software, vol. 3, no. 5, pp. 369–380, Oct. (2009)
16. Y.-C. Lee, N.-H. Tang, and V. Sugumaran, “Open Source CRM Software Selection using the Analytic Hierarchy Process,” Information Systems Management, vol. 31, no. 1, pp. 2–20, Jan. (2014)