STUDIA OECONOMICA POSNANIENSIA 2017, vol. 5, no. 12

DOI: 10.18559/SOEP.2017.12.1

Krzysztof Węcel, Bartosz Perkowski, Agata Filipowska,
Uniwersytet Ekonomiczny w Poznaniu, Wydział Informatyki i Gospodarki Elektronicznej, Katedra Informatyki Ekonomicznej

Dawid Węckowski
Agora S.A.

Piotr Zwolenkiewicz
Valeant Pharmaceuticals

Autor do korespondencji: Krzysztof Węcel, krzysztof.wecel@ue.poznan.pl

BENCHMARKING OF DATABASES FOR BIG DATA EXPLORATION IN THE SOCIAL GRAPH ANALYSIS

Abstract: In this article we present the results of the benchmarking of various data and knowledge-based systems for use while analysing a big social graph. MySQL, Neo4J, Titan and Virtuoso have been tested on data describing communication events among ca. 7 million users. We have proposed queries regarding the real-world scenarios derived from the requirements analysis of the data owner. Execution time has been measured for basic, aggregation, and networking types of queries. While MySQL was good at queries requiring calculations, Virtuoso outperformed it in graph-related queries.

Keywords: benchmarking, graph data, MySQL, Neo4J, Titan, Virtuoso.

JEL classification: C40, C55, C89, L86.

BADANIE WYDAJNOŚCI BAZ DANYCH NA POTRZEBY EKSPLORACJI DUŻYCH SIECI SPOŁECZNYCH

Streszczenie: W niniejszym artykule prezentujemy wyniki benchmarkingu różnych systemów baz danych potencjalnie użytecznych przy analizie dużych sieci społecz-

nych. Wykorzystując dane o zdarzeniach komunikacyjnych pomiędzy 7 milionami użytkowników, porównywaliśmy następujące rozwiązania: MySQL, Neo4J, Titan oraz Virtuoso. Zaproponowaliśmy przykładowe zapytania odpowiadające rzeczywistym scenariuszom wynikającym z analizy wymagań właściciela danych. Czasy odpowiedzi były mierzone w grupach zapytań podstawowych, agregujących i sieciowych. MySQL najlepiej sprawdził się w zapytaniach wymagających obliczeń, podczas gdy Virtuoso dominował w zapytaniach sieciowych.

Słowa kluczowe: benchmarking, bazy grafowe, MySQL, Neo4J, Titan, Virtuoso.

1. Introduction and motivation

The main objective of our research was to define a framework for an analysis of a social graph extracted from telecommunication data, being in this case the billing data of mobile customers. By the framework, we understand a set of tools that should support data load, analysis and presentation of the social graph. The above operations have been identified as crucial to acquire the knowledge about a social network and perform data mining. Moreover, from the point of view of diverse applications, the knowledge-based system (KBS) must deal with the problem of big data in an effective manner. In this article, we benchmark available tools that support an analysis of large amounts of data, especially graph data and provide insights on the effectiveness of these tools for various scenarios.

Achieving the main objective of the article is strictly determined by fulfilling the following goals:

a) identification of a data model,
b) identification of various tools that may support an analysis of data and building the model reflecting associations among different users as well as among users and the services they use,
c) analysis of effectiveness of data processing using different tools and with different assumptions behind the methods that will exploit the knowledge of the graph.

The article is well motivated from both business and technical point of view. Business users are interested in discovering various relations, dependencies and patterns in the gathered data. Typical analyses carried out concern customers' behaviour, like the usage of various communication channels, services, groups that a user belongs to or churn management. The business requirements for the data analysis software impact the way and how the data is processed and therefore raises research challenges.

The technical research focuses on data storage systems that should achieve an acceptable performance from an end-user perspective. In order to understand the trade-off among various database engines, a review of independent benchmarks needed to be carried out. Our idea while launching the project was to benefit from the benchmark studies published elsewhere or those that are available on the Web. Unfortunately, the benchmarks for various tools were mostly incomparable as specific data structures have to be taken into account. It is important to determine which tool is the most efficient to execute data-specific queries. In this paper, we focus on benchmarking methodology taking into account specific structures of telecommunication data – huge datasets requiring both heavy aggregations and forming social network graphs.

The structure of the article is as follows. In Section 2 the state of the art concerning available tools that support an analysis of large amounts of data, especially data in the form of graphs, is presented. As a result, tools for further evaluation are identified. Section 3 describes our data model and the specification of hardware requirements. Then, the definition of the tool evaluation procedure follows: specific steps to be performed, data to be included, detailed experiment setup, etc. Section 4 describes the experiments that were performed over the data sample. The last section summarizes the results obtained and presents our future work.

2. Related work

Nowadays, more and more information systems target at collecting, transforming, analysing and utilising large quantities of data, or so-called big data. There are several well-known areas of the economy, in which big data is generated as a result of core business activities. The most prominent examples include social networks, telecommunications, finance and traffic monitoring. The big data concepts are also constantly being adopted in many other business areas.

Our main research interests are applications concerning social contexts, in which the big data is generated upon users' interactions of different forms. In this group one of the most significant examples is the telecommunication industry [Deng et al. 2012; Magnusson and Kvernvik 2012; Bouillet et al. 2012] as well as online social networks [Menon 2012]. In both cases the collected data describe some kind of interaction among users while utilising various services. Thus, the data can be organised as a graph, where different artefacts like for e.g. user ties, user profiles and stereotypes, events or services can be

represented by nodes and various connections among them can be shown in the form of edges. Such specific features impact the choice of the database technology for performing a big data analysis.

There are several database technologies that can be considered as a potential solution for social interaction data storage. First of all, there are relational database management systems (RDBMS), having a long history in information systems development. The most popular RDBMSs, e.g. MySQL (http://www.mysql.com/), PostgreSQL (http://www.postgresql.org/), Oracle (http://www.oracle.com/us/products/database/overview/index.html), are considered to be well-established solutions and the current advances introduce improvements in various aspects of their performance, including capacity or query speed [Cattell 2011].

Taking into account the structure of telecommunication data, a graph database seems to be a natural choice for storing social network data [Angles and Gutierrez 2008]. Many graph database implementations have been developed recently. Some of them are already mature and have gained a considerable audience, like Neo4j (http://www.neo4j.org/) or DEX (http://www.sparsity-technologies.com/dex). Other projects are slowly becoming more popular, e.g. OrientDB (http://www.orientdb.org/) or Titan (http://thinkaurelius.github.io/titan/), that try to build a graph layer over other well-established solutions, such as HBase (https://hbase.apache.org/) or Cassandra (https://cassandra.apache.org/).

Another approach suitable for supporting the analysis on the graph data model is the use of a triple store, storing data in a subject-predicate-object model, which makes it suitable to operate on RDF (http://www.w3.org/TR/REC-rdf-syntax/) the representation of information. One of the most popular triple-store solutions is Virtuoso [Erling and Mikhailov 2010], which i.e. is used to serve DBpedia resources [Lehmann et al. 2015].

In order to assess the suitability of the graph databases, a comparison on specific data has to be carried out. In [Vicknair et al. 2010] two databases were compared – MySQL and Neo4j. The motivation of this study was to determine which tool would be more suitable to serve as a data persistence layer in a data provenance system. The experiment included the evaluation of subjective and objective criteria. The first group of criteria concerned maturity and level of support, ease of programming, flexibility, and security. The tests concerning the objective criteria evaluated the processing speed, disk space requirements and scalability. All tests were carried out using artificially generated graphs. The experiment revealed that Neo4j was much faster than

MySQL in all structural queries, but also much slower when processing the data analysis queries. These results and authors' assessment of the subjective criteria indicated that a graph database would not be an acceptable solution for their data provenance system.

In [Dominguez-Sal et al. 2010] four graph databases were compared using the implementation of the HPC Scalable Graph Analysis Benchmark (HPC--SGAB). The tools taken under consideration were: Neo4j, Jena, HypergraphDB and DEX. For the purpose of this experiment the R-MAT algorithm was used to generate graphs of a desired size and edge density. HPC-SGAB presents the results of four sequential stages that evaluate different aspects of graph databases: data loading, finding a subset of edges, building subgraphs and traversal performance. The experiment showed that for most of the stages DEX was the most efficient tool, second to Neo4j only while assessing the traversal performance. Jena and HypergraphDB turned out to be unscalable.

In [Garulli 2011] the authors provide a comparison of the results of TinkerPop Blueprints Test Cases for OrientDB and Neo4j. Although it is not a benchmark, but rather a set of tests for checking the compliance with TinkerPop Blueprints standards, it provides some insights into the performance of databases. The experiments performed on different platforms and hardware configurations showed that OrientDB is from 1.8 to nearly 10 times faster than Neo4j.

To the best of our knowledge, some databases have not yet been compared with others, although individual tests were carried out. One of these databases is Titan, for which a simulation of a social service similar to Twitter was performed to evaluate its efficiency [Rodriguez 2012]. This experiment proved that Titan is scalable and can support thousands of concurrent users.

There are also other comparative benchmarks [Ciglan, Averbuch and Hluchy 2012; Angles et al. 2013; Barmpis and Kolovos 2012; Macko, Margo and Seltzer 2013; Morsey et al. 2011], although providing a comparison or a summary of their findings is difficult. There are many aspects of the database benchmarking design, which can be very influential, especially with respect to the graph databases [Paradies 2012], e.g. data characteristics, a data model or a workload definition. Also query languages that are used for data processing have different features and various expressiveness [Holzschuher and Peinl 2013], which not only affects the execution time, but also the time spent for query preparation. As we can see, the benchmarks' results are highly dependent on the data application domain, which poses the need for new comparative studies in different areas of business.

3. Experiment's description

3.1. Data model

In the case of big data not only the query time has to be measured for benchmarking purposes but also the time of data loading. It may happen that one solution stores data linearly without any additional calculations, while the other optimizes storage, precalculates aggregations, builds indexes and other auxiliary structures. The loading time will be significantly shorter in the first case, while querying apparently in the latter. Therefore, two types of experiments have been designed in order to measure both aspects.

The data model of the datasets used for benchmarking is presented in Figure 1. It was directly used by graph databases. A relational database applied a model reflecting this data, but after applying normalisation rules. The primary key is defined by a connection node, which stores date and duration. Each connection has such properties as a sender, a receiver, a connection channel and a location, with specific attributes assigned to an each node.

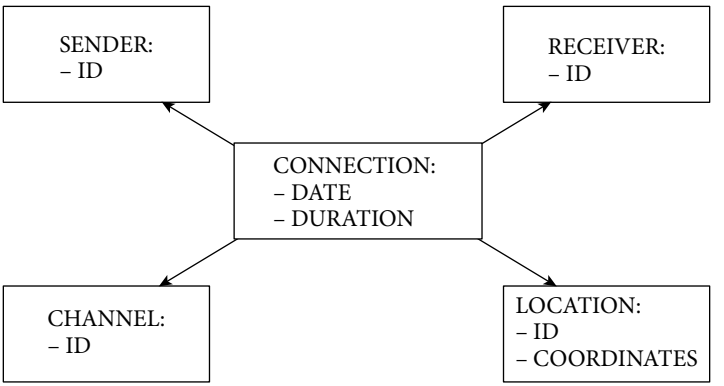

Figure 1. The graph model of the billing data

We assume that the billing data from one day have to be loaded quicker than within a day, which is not obvious when data is collected in a distributed manner. For measuring the load time, three datasets of billing data of various sizes were prepared, where the biggest one represented the whole day. Moreover, three groups of queries in basic, aggregating and networking scenarios were repeated for each size of the dataset. These scenarios were identified and based on the initial requirements analysis (i.e. future applications of the graph) as well as the literature analysis.

The decision to limit our experiments to one-day is justified by the size of data. The graph that has been built from the source data contained roughly 7.6 million nodes and 5.4 million edges. This was a representative sample for the initial experiments regarding the effectiveness of data processing.

3.2. Infrastructure

The tests have been run on a virtual machine that used 8 Core CPU Intel Xeon, 16GB DDR3 RAM and 500GB of disk space. Each tool needed its own staging area, so altogether ca. 30GB of space was needed for a single day. The benchmark instance used the Debian 7.0 operating system with the customized configuration of each tool.

The experiments were carried out using three types of previously identified tools: relational databases–MySQL (because of its popularity); graph databases–Neo4j, OrientDB, Titan; triple stores for the RDF–Virtuoso. The initial tests showed that OrientDB is not suitable for the amount of data that is needed to be processed. Due to technical problems with loading even small samples of data, this solution has finally been excluded from further experiments.

3.3. Benchmarking procedure

The benchmarking queries were executed three times against each of the three datasets separately, and then an average time was calculated.

Data Loading Experiment Samples of three different sizes were created: 1 million (LD01), 5 million (LD02) and 25 million records (LD03). The scale factor of 5 was determined intentionally, so that the scalability of the solutions could be easily calculated, i.e. a fivefold increase in size should ideally result in less than a fivefold increase in the loading time. For dataset LD03 also lap times for each 1 million records were noted, so that it was possible to observe the impact of data load progress on efficiency.

Basic Queries Experiment The basic queries are related to the identification of distinct entities within the data. A tool has to count a number of distinct senders (*QB01*), distinct receivers (*QB02*) and a number of unique services that are used (*QB03*). Depending on the solution, the results could be calculated on demand or read from indices.

Aggregation Queries Experiment The aggregations are essential for analytical purposes. They can be calculated on-the-fly or in advance (like in a data

warehouse). Various tools can handle this crucial issue in different ways, therefore it is necessary to compare the efficiency with regard to the defined requirements. Some queries involve also sorting and identifying the top entities. The following queries were defined for this experiment:

QA01: A total number of connections with each service/within a channel.
QA02: The most active senders using a specific service/channel sorted by a total duration.
QA03: The most active receivers using a specific channel sorted by the number of connections.
QA04: The most active location with regard to the number of connections.
QA05: A number of users who initiated communication within a specified timespan and a channel.

Network Queries Experiment The overall purpose of our research is however to build a social graph based on given data. As the graph algorithms intuitively seem to be the most important in this case, therefore a separate sub-experiment was designed.

We assume that the contact between a caller and a receiver is a directed relation. Contacts of the 1st degree are people contacted by an initial user subject for analysis, while people contacted by receivers are 2nd degree contacts. However, in the identified requirements also undirected relations are important which is reflected in the query *QN06*.

The following queries were defined for this experiment[1]:

QN01: 25 users with the highest number of contacts.
QN02: The same as *QN01* with an additional restriction on the duration of a single communication (filter before).
QN03: The same as *QN01* with an additional restriction on the total duration of communication (filter after).
QN06: 25 users with the highest number of networkers restricted by a condition on the total duration of mutual conversations (undirected graph).
QN07: All contacts of the second degree of a given user.

3.4. Data import and querying languages

Each tool required its own import format and queries in a specific language. The programs in Python and Java were prepared to convert raw CSV files with billing data into a format required by the tools. The conversion time was negligible compared to the loading time and similar for all solutions,

[1] Queries *QN04* and *QN05* were excluded from this article as the results were very similar to queries *QN03* and *QN06* respectively.

therefore it is not included as a comparison. For querying, each tool used a different language, not only on the syntax level: MySQL–SQL, Titan–Gremlin, Neo4J–Cypher, and Virtuoso–SPARQL. The different design principles of these languages make it hard to formulate equivalent queries. For example, the expressiveness of the Gremlin query language was not sufficient to express all designed queries in a simple form.

4. Benchmark results and analysis

Based on several iterations of the initial tests, we carefully selected and designed effective algorithms for loading data into specific databases and then we queried these databases.

The sections below present the detailed results of the experiments that we carried out. In some cases, for a specific query and a specific database the results are marked as FAIL(ed). This means that e.g. there was not enough memory to complete the experiment.

4.1. Data loading results

According to the MySQL documentation, using the LOAD DATA INFILE MySQL statement is the recommended way to load large CSV files into a database, and it was used in the experiment. A shell script has been written, which iterated over the files containing segmented data and measured the time of loading each segment into the database.

In the case of Neo4j, the data loading task was done using a Java program, which communicated with the embedded database using the JDBC. At each run, the program retrieved from the database an index of the nodes' keys that were previously created for senders, receivers, channels and locations. Next, it scanned the input CSV file to filter new vertices that needed to be created and inserted them into the database. Finally, the program created all of the required edges. Reading the nodes' keys from the memory was done for speeding up the process of filtering new nodes from the input files, although reading from the database index before loading each 1M sample took a significant amount of time – the bigger the index was, the longer it took (see Figure 2).

With Titan the loading procedure was similar to the Neo4j. The difference was in managing the set of nodes' keys–as the preliminary experiments showed, the time to read from Titan's index was too long. That is why in the experiment we stored the inserted nodes' keys in text files that could be quickly read into the memory. The overhead on the reading keys from text files was smaller than in the case of Neo4j.

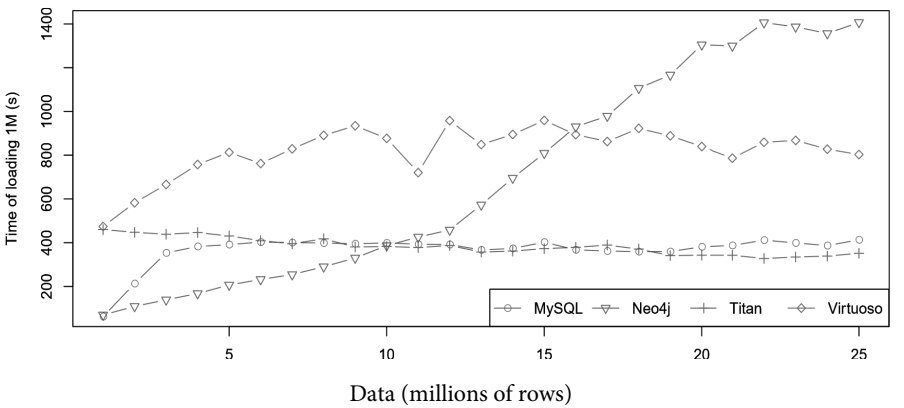

Figure 2. Loading times of sample data into MySQL, Neo4j, Titan and Virtuoso in batches of 1 million connections

For Virtuoso, the input files were prepared in the Turtle format (http://www.w3.org/TR/turtle/), using a custom script in Python. The identifiers of entities were converted into URIs to meet the requirements of this format. The execution time of the conversion script was negligible compared to loading time – ca. 4 minutes to generate all intermediate files. The script splits data into separate files of one million connections for benchmarking purposes, with an average file size of 280MB. Unfortunately, a significant amount of time was necessary to load the data into the Virtuoso server. The first file was loaded in 470 seconds (almost 8 minutes), but then each next loading took longer.

A comparison of all loading times is presented in Figure 2. All solutions but Titan were characterized by the growing time of loading a single file, each containing 1 million connections. In most cases the growth stopped at some moment, and only Neo4J seems to be problematic. This phenomenon can be explained by the indexing mechanism. Each connection has information on the sender and receiver, and before inserting a new record, the existence of entities has to be checked; hence some optimization described above.

4.2. Basic queries results

MySQL proved to be relatively fast and reliable, when it comes to basic queries. Due to the use of indexes, the execution of the *QB03* query was almost immediate.

Although basic queries seem to be simple, the query execution times of both Neo4j and Titan were not impressive, except for the QB03, where the number of nodes to check was relatively small. The reason for such results can

be the data model that was not optimised for these types of queries – both the senders and receivers are modelled in the same way, and the distinction can be made only upon checking, if there exists a given type of relation to a connection node. Using custom indexes could possibly improve the query times.

In the case of Virtuoso, the queries in the first group were executed in a relatively short time. It is more or less proportional to the number of objects that have to be counted.

The combined results showing query execution time are presented in Table 1 and Figure 3.

Table 1: Benchmarking results for basic queries (execution time in seconds)

	MySQL	Neo4j	Titan	Virtuoso
QB01 LD01	1.07	26.281	329.84	0.63
LD02	5.98	65.530	1494.14	2.73
LD03	149.59	4369.543	FAIL	14.17
QB02 LD01	1.47	6.493	336.00	0.94
LD02	9.87	38.751	1500.95	5.08
LD03	114.68	330.104	FAIL	29.26
QB03 LD01	0.00	0.170	0.25	0.30
LD02	0.00	0.143	0.24	1.07
LD03	0.00	0.489	0.25	5.43

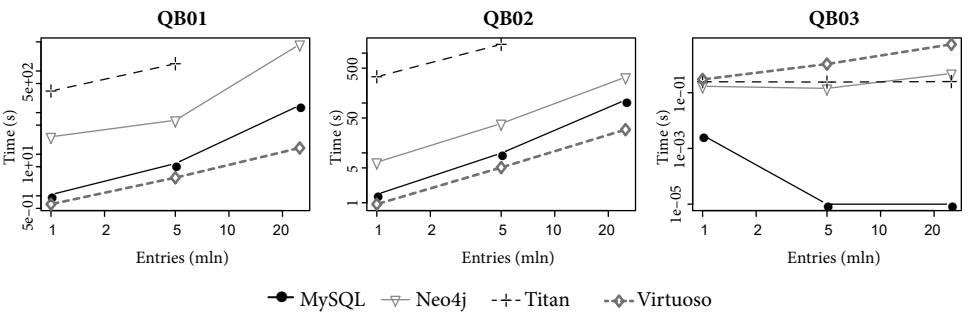

Figure 3. Time of basic queries execution for 1M, 5M and 25M samples

4.3. Aggregation queries results

The queries in the second group (QA–aggregation) required more advanced analysis on data to be performed. The combined results are presented in Table 2 and Figure 4.

Table 2. Benchmarking results for aggregation queries (execution time in seconds)

	MySQL	Neo4j	Titan	Virtuoso
QN01 LD01	8.13	17.760	956.64	11.51
LD02	70.46	281.965	FAIL	65.47
LD03	369.61	22111.838	FAIL	938.10
QN02 LD01	7.80	30.527	1076.93	6.61
LD02	78.82	120.437	FAIL	41.34
LD03	614.10	5865.503	FAIL	328.64
QN03 LD01	7.42	26.289	NA	10.69
LD02	85.28	173.958	NA	52.68
LD03	380.84	5841.766	NA	184.14
QN06 LD01	5.01	94.410	NA	27.30
LD02	25.68	672.518	NA	134.83
LD03	113.66	FAIL	NA	1096.03
QN07 LD01	26.52	0.082	0.60	0.00
LD02	327.65	0.539	0.69	0.00
LD03	1884.80	1.716	4.75	0.12

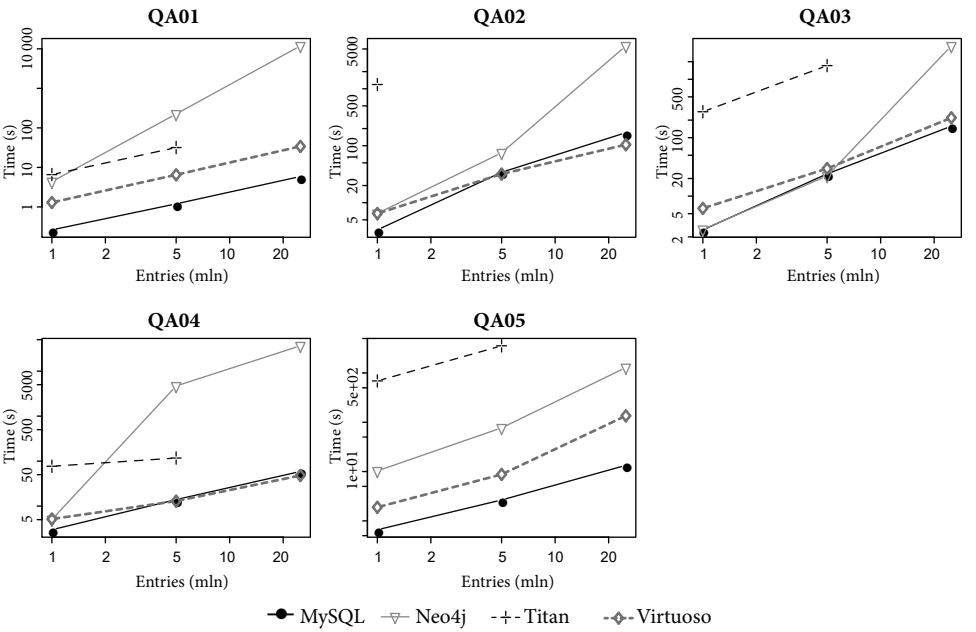

Figure 4. Time of aggregation queries execution for 1M, 5M and 25M samples

As relational databases were designed to support aggregation operations, all queries from the QA group were no challenge for MySQL. It was the fastest tool for most of the queries.

A worse situation was in the case of graph databases. The query times were relatively long for both Neo4j and Titan. The reason for this was that the query characteristics that involved value aggregations came from different nodes. The need of visiting a large number of nodes at each query resulted in poor results. In some cases for Titan even the 5M sample was too big to complete a query.

In many cases the performance of Virtuoso was comparable to MySQL. The execution time depended on a number of entities that needed to be aggregated, e.g. a small number of services in QA01 (20) and a big number of receivers in QA03 (several million).

4.4. Network queries results

The queries in the third group (*QN* – network) are the most interesting in the context of social graphs. The combined results are presented in Table 3 and Figure 5.

Table 3. Benchmarking results for network queries (execution time in seconds)

	MySQL	Neo4j	Titan	Virtuoso
QA01 LD01	0.26	4.367	6.57	1.30
LD02	1.20	224.987	31.78	6.51
LD03	5.89	11870.261	FAIL	34.04
QA02 LD01	3.33	6.493	1187.82	6.46
LD02	34.17	75.070	FAIL	31.83
LD03	169.60	5662.514	FAIL	104.54
QA03 LD01	2.64	2.684	277.96	6.17
LD02	24.02	22.147	1712.72	29.62
LD03	157.34	3730.040	FAIL	219.23
QA04 LD01	2.98	5.041	76.94	5.16
LD02	13.96	4689.591	117.11	12.94
LD03	58.87	36625.362	FAIL	47.95
QA05 LD01	0.66	10.324	689.16	1.89
LD02	2.65	76.940	3557.22	8.77
LD03	13.38	1286.945	FAIL	135.24

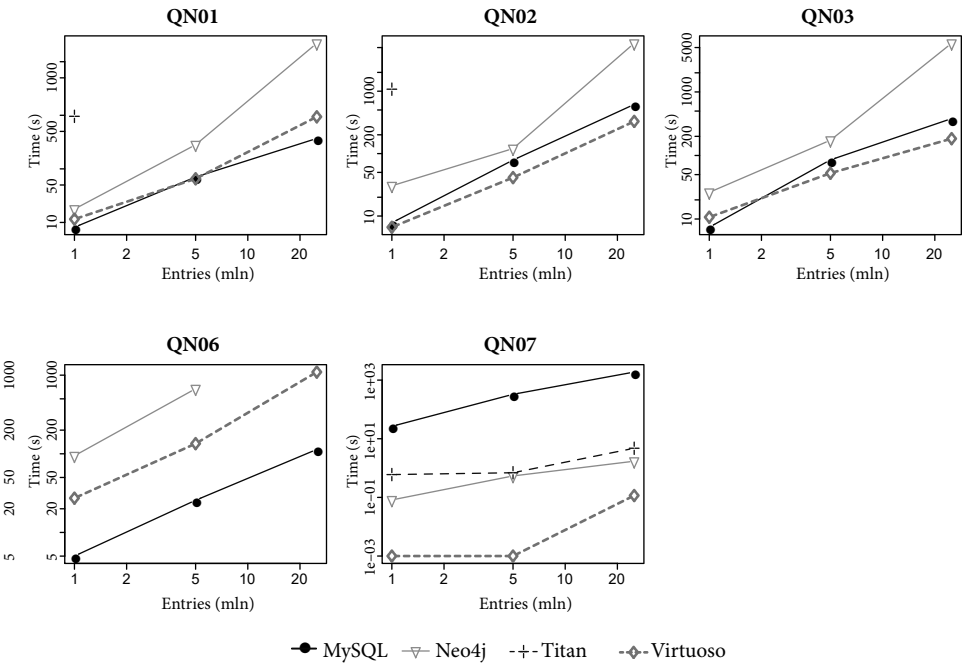

Figure 5. Time of network queries execution for 1M, 5M and 25M samples

The network queries took substantially longer to process by MySQL as the queries became more complicated and required many joins. Moreover, the execution time of the *QN07* query, which was intended to utilize the potential of graph databases, was incomparably longer than in the case of the alternative solutions.

Unexpectedly, graph databases proved to be not the best solution for the *QN* queries. They were doing quite well for the *QN07* query, which involved neighbourhood querying, but as the other queries involved also some aggregation tasks, both Neo4j's and Titan's performance was poor just as the *QA* queries were. Additionally, Titan did not complete all queries because of difficulties in formulating appropriate queries using the Gremlin query language.

As Virtuoso is optimized for operations on a graph, its performance for *QN07* – path traversal – was no surprise. The query with the longest duration to execute was *QN06* which actually required the execution of two sub queries (using UNION). *QN03* required the filtering of aggregated results, since there was also a sub query that first calculated all aggregations and then filtered them out.

5. Conclusions and future work

The MySQL database proved to be relatively fast and reliable, when it comes to basic and aggregation queries, but slightly slower while performing the networking queries. As relational databases were designed to support aggregation operations, all queries from that group were no challenge for MySQL. The networking queries group took substantially longer to process as queries became larger and more complicated. Moreover, the execution time of the *QN07* query, which was intended to utilize the potential of graph databases, was incomparably longer than in the case of alternative databases.

Although loading the data into the relational database like MySQL and executing most of the tested queries is relatively fast, the lack of graph representation is a major flaw of this solution. However, the good performance that may be achieved while performing the aggregation tasks and the stable time of loading the data predestines MySQL as a support for other, graph-enabled solutions. Future work should determine, if and to what extent this relational database can be used as a supportive tool in a solution that performs more efficiently while processing graphs.

The graph databases Neo4J and Titan perform quite well with data loading, provided that there is an in-memory index of nodes' internal identifiers. Their performance is also good in carrying out queries concerning single nodes and their neighbourhood. The poor performance of graph databases is observed in aggregation queries; hence many real-life scenarios cannot be supported. The reason for this is that aggregation queries often force the graph databases to read the whole graph into the memory, causing problems with memory allocation. Moreover, from the programming point of view, aggregation queries are sometimes hard to express in the graph query languages.

The loading of files into Virtuoso (triple store) took an unexpectedly long time–it was around 6 hours on a sample day. There is a risk that when data concerning a longer timeslot (more days) are added, the time will get even longer. On the other hand, the best results (overall) were achieved for queries related to the network (path traversal), which is a good message with regard to building and analysing a social graph.

As already mentioned, a graph database is not so effective for aggregation as it first requires grouping, and grouping on-the-fly requires a lot of memory or disk cache. One possible solution for this is a split of information among dedicated systems, i.e. detailed data kept in a database, aggregations in a data warehouse, and only graph-related data in graph databases (Neo4J) or triple stores (Virtuoso). Potential benefits depend on how much networked data will

be utilized. Another area of investigation is the performance of the databases with different data models. One example is compacting the graph, so that it represents directly connections between senders and receivers. Only aggregated data is provided for these connections, like total and maximal duration. It is sufficient enough to answer the benchmark queries and improves results significantly (in the case of *QN* queries network relations do not have to be created per query). Some information is lost, but when taking business case into account it does not affect the decisiveness. We expect to gain improvements with the use of data pre-processing and aggregation before loading the data to the graph database, which has already been confirmed by experiments carried out on the aggregated data model.

Future work should also focus on tuning the performance of graph databases with regard to concurrent data loading and database querying tasks. This can be achieved with the use of tools like LinkBench [Armstrong et al. 2013] along with a set of custom plug-ins. It is also crucial to find an appropriate balance between data aggregation and details of analysis. Therefore, further experiments with specific business cases are necessary.

Bibliography

Angles, R., Gutierrez, C., 2008, *Survey of Graph Database Models*, ACM Computing Surveys, vol. 40, iss. 1, s. 1–39.

Angles, R., Prat-Pérez, A., Dominguez-Sal, D., Larriba-Pey, J.-L., 2013, *Benchmarking Database Systems for Social Network Applications*, in: *First International Workshop on Graph Data Management Experiences and Systems*, New York, NY, s. 15:1–15:7.

Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M., 2013, *LinkBench: A Database Benchmark Based on the Facebook Social Graph*, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, New York, NY, s. 1185–1196.

Barmpis, K., Kolovos, D.S., 2012, *Comparative Analysis of Data Persistence Technologies for Large-scale Models*, in: *Proceedings of the 2012 Extreme Modeling Workshop*, Innsbruck, s. 33–38.

Bouillet, E., Kothari, R., Kumar, V., Mignet, L., Nathan, S., Ranganathan, A., Turaga, D.S., Udrea, O., Verscheure, O., 2012, *Processing 6 Billion CDRs/day: From Research to Production (Experience Report)*, in: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, Berlin, s. 264–267.

Cattell, R., 2011, *Scalable SQL and NoSQL Data Stores*, SIGMOD Rec., vol. 39, iss. 4, s. 12–27.

Ciglan, M., Averbuch, A., Hluchy, L., 2012, *Benchmarking Traversal Operations over Graph Databases*, 2013 IEEE 29th International Conference on Data Engineering Workshops, s. 186–189.

Deng, C., Qian, L., Xu, M., Du, Y., Luo, Z., Sun, S., 2012, *Federated Cloud-based Big Data Platform in Telecommunications*, in: *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, San Jose, CA, s. 44–48.

Dominguez-Sal, D., Urbon-Bayes, P., Gimenez-Vano, A, Gomez-Villamor, S., Martinez-Bazan, N., Larriba-Pey, J., 2010, *Survey of Graph Database Performance on the HPC Scalable Graph Analysis Benchmark*, WAIM 2010 Workshops, s. 37–48.

Erling, O., Mikhailov, I., 2010, *Virtuoso: RDF Support in a Native RDBMS*, in: Virgilio, R., de, Giunchiglia, F., Tanca, L. (eds.), *Semantic Web Information Management*, Springer, Berlin Heidelberg, s. 501–519.

Garulli, L., 2011, *GraphDB Benchmark part II* [https://zion-city.blogspot.com/2011/04/graphdb-benchmark-part-ii.html, accessed 1.12.2017].

Holzschuher, F., Peinl, R., 2013, *Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4j*, in: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, Genoa, s. 195–204.

Lehmann, J. et al., 2015, *DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia*, Semantic Web Journal, vol. 6, no. 2, s. 167–195.

Macko, P., Margo, D., Seltzer, M., 2013, *Performance Introspection of Graph Databases*, in: *Proceedings of the 6th International Systems and Storage Conference*, Haifa, s. 18:1–18:10.

Magnusson, J., Kvernvik, T., 2012, *Subscriber Classification within Telecom Networks Utilizing Big Data Technologies and Machine Learning*, in: *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, Beijing, s. 77–84.

Menon, A., 2012, *Big Data @ facebook*, in: *Proceedings of the 2012 Workshop on Management of Big Data Systems*, San Jose, CA, s. 31–32.

Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C., 2011, *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*, in: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.), *The Semantic Web – ISWC 2011*, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, vol. 7031, s. 454–469.

Paradies, M., 2012, *Challenges in the Design of a Graph Database Benchmark* [http://www.slideshare.net/graphdevroom/challenges-in-the-design-of-a-graphdatabase-benchmark, accessed 1.12.2007].

Rodriguez, M., 2012, *Titan Provides Real-time Big Graph Data*, [https://dzone.com/articles/titan-provides-real-time-big, accessed 1.12.2007].

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D., 2010, *A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective*, in: *Proceedings of ACM Southeast Regional Conference*, s. 42.