MATTER: International Journal of Science and Technology
ISSN 2454-5880

Nacional et al., 2019

Volume 5 Issue 3, pp.01-14

Date of Publication: 15th November 2019

DOI- https://dx.doi.org/10.20319/mijst.2019.53.0114

This paper can be cited as: Nacional, T., Niinimaki, M., & Heikkurinen, M., (2019). RDF Databases – Case Study and Performance Evaluation. MATTER: International Journal of Science and Technology, 5(3), 01-14.

This work is licensed under the Creative Commons Attribution-Non Commercial 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

RDF DATABASES – CASE STUDY AND PERFORMANCE EVALUATION

Tony Nacional
School of Business and Technology Webster University Thailand, Bangkok, Thailand
nacionalm@webster.ac.th

Marko Niinimaki
School of Business and Technology Webster University Thailand, Bangkok, Thailand
niinimakim@webster.ac.th

Matti Heikkurinen
PROCESS, Ludwig-Maximilians-Universität, Munich, Germany
heikku@nm.ifi.lmu.de

Abstract

The Resource Description Framework (RDF) data presentation model and the SPARQL query language have been the core of the semantic web technologies since the early 2000's. In this article, we evaluate three RDF storage technologies. Our motivation is to find a storage solution that can be used to process "big data" RDF sets. Our method is based on measuring query response times with large samples (hundreds of thousands of RDF documents, millions of RDF statements). We find that all the proposed technologies provide much better performance than querying RDF data stored in files. However, with 300 000 documents, even with the fastest technology, an aggregation query still lasts more than 100 seconds in our environment. As a further performance improvement, we test the same data and queries with MongoDB, demonstrate its performance (10 seconds instead of 100) and scalability (up to 1000 000 documents). However,

despite its benefits we must note that because of its data presentation and query limitations, MongoDB probably cannot serve as a generic storage for all kinds of RDF documents.

Keywords

RDF, Database, noSQL, Benchmarking, Big Data, Query Performance

1. Introduction

The Resource Description Framework (RDF) was originally developed for describing resources on the Web. This is done by making statements about Web resources (pages) and things that can be identified on the Web, like products in on-line shops (W3C, 2014). Using RDF, one identifies things using Uniform Resource Identifiers, or URIs, and describes resources by issuing statements in terms of simple properties and property values.

An RDF statement is a triple of subject, predicate, and object. The statement asserts that some relationship, indicated by the predicate, holds between the things denoted by the subject and the object of the triple. As an example of a resource on the Web, we can have the following statement. The web page whose URI is "http://www.example.org/xyz.html" (subject) has a creator (predicate) that is N.N. (object). As an example of a thing outside of the Web, but referred to it by an URI, we can consider the following: A person, Magnus Carlsen, referred to by the URI "<http://www.wikidata.org/entity/Q106807>" (subject) has a date of birth (predicate) 30 November 1990 (object). Both subject and object can be blank nodes that represent unknown or undetermined values. Figure 1 (from (W3C, 2004)) shows an illustration of a set of RDF statements about a person as a directed graph. If the email address was unknown, it would be represented as a blank node illustrating that there is an email address but we do not know it.

The SPARQL language (W3C, 2008) was designed for querying RDF documents. Intuitively, we can see an SPARQL query as a template containing blank nodes. The query evaluation process matches the blank nodes with actual data (if possible). For instance in the query "`SELECT ?MCdb WHERE {<http://www.wikidata.org/entity/Q106807> <http://www.wikidata.org/prop/direct /P569> ?MCdb . }`" the blank node "?MCdb" will be matched with the actual date. P569 in the query is the property "date of birth".

Figure 1: *An RDF Graph from W3C RDF Primer*

Since RDF graphs express information as subject-predicate-object, there is a terse text format called N-triples (W3C, 2014). However, graphs are normally stored in an XML format, often in files. It is possible to use command line tools to issue SPARQL queries using such files as sources, but with large amounts of data this will become impractical. Some of the early attempts to deal with large data included an RDF query API that could be used persistent RDF graph data stored in a BerkeleyDB database (Miller, Seaborne, & Reggior, 2002). Later, "native" RDF databases, among them many commercial solutions have appeared (Faye & Curé, 2012). One of the early databases was Sesame (Broekstra, Kampman, & Van Harmelen, 2002) that later developed into a framework called RDF4J. In general, databases that store RDF data in the subject-predicate-object format are called triple-stores (Levandoski & Mokbel, 2009). Levandoski and Mokbel (Levandoski & Mokbel, 2009) discuss popular approaches for implementing a storage for a triple-store. They mention a triple-store schema, where each triple is stored in a three-column table in a relational database, and a property table model where RDF properties are stored as n-ary table columns.

In this paper, we evaluate the performance and scalability of three different RDF storage solutions. One of the is based on BerkeleyDB and two others are "native RDF" commercial products. The other one of the products is intended for enterprise data integration and the other

one is seen as more generic. The generic one is based on Sesame/RDF4J. In our measurement section, we call these products "Integration" and "Sesame based", respectively.

In related research, Arenas et al (Arenas, Gutierrez, & Pérez, 2009) present the semantics of RDF and the complexity of evaluating SPARQL expressions. Morsey at al (Morsey, Lehmann, Auer, & Ngomo, 2009) present DBPedia datasets and queries that can be used for query performance analysis. Unfortunately, the tool is no longer available, but an earlier benchmark by Becker (Becker, 2008) used similar data and five queries such as (i) query all information about a specific subject (ii) "two degrees of separation" (iii) unconstrained query about specific types (iv-v) combining web and GPS information in two cities. Other RDF query benchmark tools are discussed by Schmidt et al. (Schmidt, Schallhorn, Lausen, & Pinkel, 2009), and they additionally develop their own benchmark. Vicknair et al. (Vicknair, et al., 2010) compare features of a relational database management system with a graph database that is not based on RDF. Lindemann et al (Lindemann, Schmidt, Schrader, & Keune, 2009) emphasize the benefits of using RDF in representation of medical data. Our dataset has earlier been used in a XML database performance evaluation (Niinimaki, Heikkurinen, & Schmidt, 2019) but converting the data into an RDF form will allow us combine the data with rigorous ontologies (see (Noy, Rubin, & Musen, 2004). The main contribution of our paper is to test RDF databases with data from medical articles and compare the retrieval times with retrieval times of similar data in other types of databases. This research is a part of our long-term project where we build tools for accessing data from large data sets (Niinimaki & Thanisch, 2019), (Niinimaki & Niemi, 2009).

Our sample RDF documents, hardware and software environments and methods of measurement are introduced in Section 2. The performance results are presented in Section 3. Additionally, we discuss the performance of RDF access with two other technologies, XML databases and MongoDB in Section 4. Finally, Section 5 contains a summary, notes about non-RDF graph databases, and items for future research.

2. The Environment and Data

We have built and executed our query performance benchmarking in a relatively typical higher end Linux environment. The hardware is a 24-core Xeon server (E5-2620 v2 @ 2.10 GHz) with 32 GB memory running the Debian 8 distribution of the Linux operating system. Our native RDF databases are Java-based, and the Java version in the computer is 1.8.0_66.

Our data set consists of hundreds of thousands of XML documents downloaded from the U.S. National Institute of Health's PubMed collection of medical articles (Steinbrook, 2005). The articles (without images) are available in compressed files at ftp://ftp.ncbi.nlm.nih.gov/pub/pmc. The size of the compressed files is currently about 50 GB, and the uncompressed size about 140 GB. At the time of the writing, the files contained 2.1 million articles and thus the average size of an XML file was 67 kilobytes. The earliest article in the collection is from 1610[1], but almost 90% of the articles are from 2000 or later. Most of the articles contain both the metadata and the textual contents in the JATS (Journal Article Tag Suite) XML format. For details about JATS, see (Donohoe, Sherman, & Mistry, 2015). The XML documents were converted into an RDF format using a JATS-to-RDF stylesheet.

We have used four document sample sets for our measurements: a set of 100 000, 200 000, 300 000, and one million documents. The number of RDF statements in these sets is 7.6 million, 17.5 million, 27.6 million and 124 million, respectively.

The queries that we tested with the sets are as follows:

- Q1: Print the publication date (actually only the publication year is recorded as a date) of each article. The corresponding SPARQL expression is `select ?a ?y { ?a <http://purl.org/dc/elements/1.1/date> ?y }`
- Q2: Print article information if the article contains the word "genitalia" anywhere. `select ?s ?p ?o WHERE { ?s ?p ?o. FILTER (regex(?o,'genitalia')) }`
- Q3: Print top 100 articles that have been cited by other articles, and how many times they have been cited. `select ?c (count(?c) AS ?total) { ?a <http://purl.org/dc/terms/references> ?c } group by ?c order by ?total limit 100`

For curious readers, the most frequent year of publication was 2016, the word "genitalia" appeared in 3634 articles (of 1 million), and the most cited article was "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". The 1610 article is "The Whole Aphorismes of Great Hippocrates". Some documents contain years earlier than 1610 in the publication information, but they seem to be mistakes and years based on the Islamic calendar.

3. Measurements and Results

We timed the execution time of each query with the standard Linux "time" command. Each measurement was repeated several (usually 10) times. Other than the usual operating system tools, there were no programs executing in the computer during the measurements.

RDF frameworks and tools can evaluate queries even when the RDF source is plain files. With large amounts of data this, however, becomes impractical. For example, evaluating query 1 (list the publication year of each article) when each article is stored in an RDF file takes about 2.8 seconds per file, making this method unusable in general. This is mainly because the RDF tool needs to parse each file before evaluating the query. When using a database, the data has been already parsed and organized. All the databases that we have measured (below) perform much better than a file-based approach.

We have tested the performance with datasets containing RDF documents and loading them to the database products that we tested. With Python RDFlib, we first wrote a Python program that reads the RDF contents from files and stores it in a BerkeleyDB using its API. The queries are then executed using Python programs that build a graph from the database contents and then evaluate the query. Practically this means that each query evaluation is with a "cold start" since the data is always read before the query is evaluated. We can test the database opening time by "query 0" that just opens the database and exits.

The data integration oriented commercial RDF runs only as a client/server application. To imitate a "cold start", we stop and start the database server after each query. The database startup time is shown as "query 0" result. For completeness, for this product, we have included numbers of "warm start". These were measured by starting the database, running the queries 10 times in sequence and calculation the averages of query times. As an interesting detail (acknowledged by the software developers, too), the complex "most cited articles" query was faster after a cold start.

The Sesame based commercial RDF database has no limitations in terms of number of statements, and it has a command line tool that opens the database and then executes the query. A "query 0" is used to test the database opening time similarly to Python RDFlib.

The number of RDF statements in each of our datasets is as follows:

100k 7 581 887

200k 17 551 293

300k 27 652 173

1M 124,319,278

Results (query times in seconds) for each of the database products are shown in Table 1, and an illustration of the results in Fig 2.

Table 1: *Query Times in Seconds*

Python RDFlib with BerkeleyDB:	Q0	Q1	Q2	Q3
100k	23.8	55.1	2349.1	60.4
200k	53.5	102.4	5357.3	284.0
300k	95.2	175.4	8182.7	539.2

Integration, cold start				
100k	6.1	5.6	19.4	3.9
200k	7.1	7.1	40.0	11.5
300k	8.0	8.5	59.6	24.6
1M	9.6	25.2	221.0	58.4

Integration, warm start				
100k		3.8	18.1	3.9
200k		5.6	36.6	11.5
300k		6.8	57.6	24.6
1M		22.2	220.4	58.

Sesame-based				
100k	6.1	6.7	32.8	10.6
200k	6.5	7.5	66.7	43.4
300k	6.8	8.4	103.9	70.7
1M	7.3	14.8	507.6	-

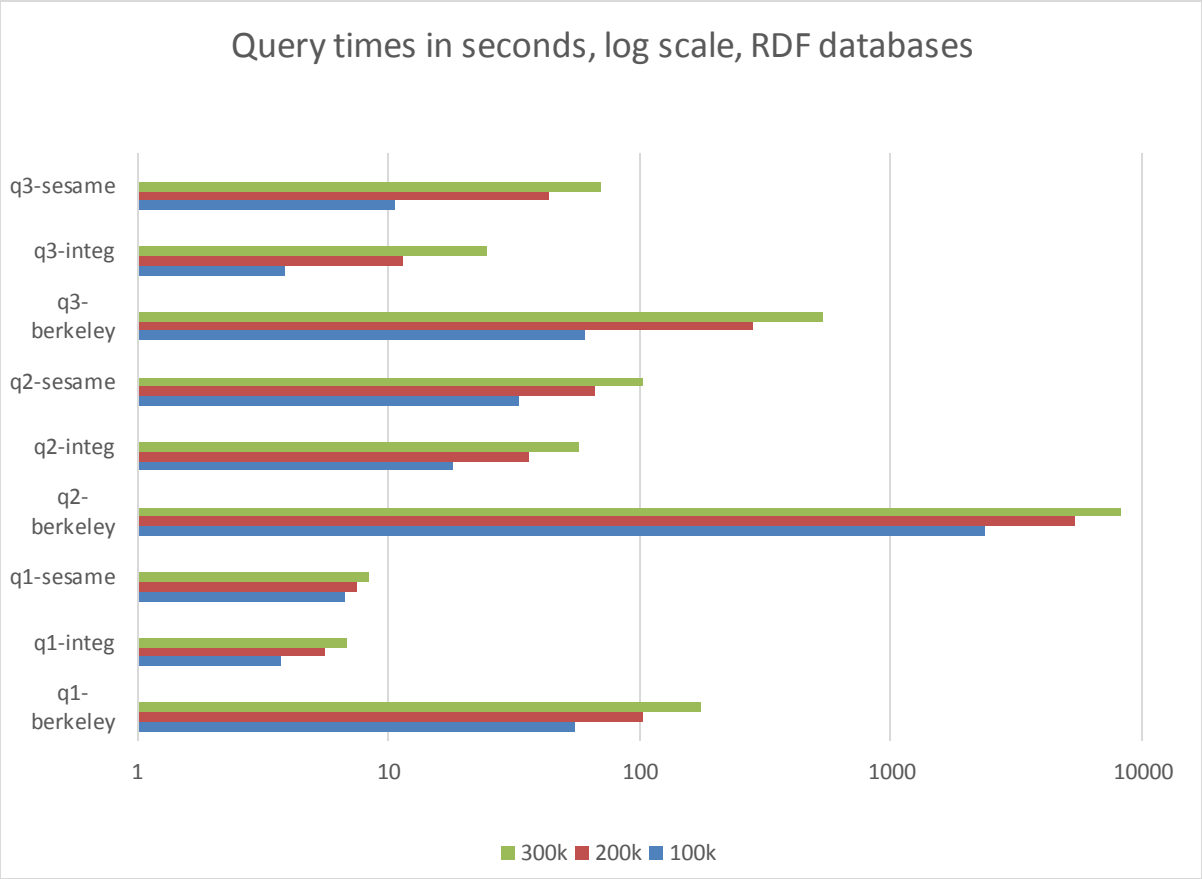

Figure 2: *Query Times in Seconds, Logarithmic Scale*

For completeness, we present the standard deviations of the 300k measurements for Q2 below.

Python RDFlib	Integration, warm start	Sesame
229.6	1.32	8.93

We can see that the text matching query (Q2) took most time with all the databases. This is probably because RDF databases are very seldom optimized for textual search. The native RDF

databases were (not surprisingly) generally faster than Python with a BerkeleyDB back-end. Both commercial RDF databases were able to process queries even with data of 1 million documents. However, with that amount of data, query Q3 was too hard for the Sesame-based database: the database query engine failed after 1 hour 14 minutes due to heap memory problem (we had allocated 8 GB maximum heap). In the era of "big data", a collection of one million RDF documents (or 124 million triples) is not exceptionally large – Oracle has tested an RDF storage with 475.6 billion triples (Oracle, 2016). In order to manage larger amounts of data in our environment, we shall compare the triple-store technology with other approaches in the next section.

4. Comparison with Other Solutions

In our earlier paper (Niinimaki, Heikkurinen, & Schmidt, Performance of XML databases, 2019), we studied the query performance of XML databases with the same source documents (medical articles) and same queries (using the XPath query language) as in this study. We measured the performance of an XML enabled relational database and a native XML database (query 3 refused to run on the native XML database). A summary of the results combined with the results of this study are shown in Table 2 and illustrated in Figure 3.

Table 2: *Query Times (in seconds) with XML and RDF based Databases*

100k	RDBMS-XML	Native XML	Python-RDF	Integration	Sesame-based
Q1	10.5	5.1	55.1	5.6	6.7
Q2	145.8	17.2	2349.1	19.4	32.8
Q3	45.5		60.4	3.9	10.6

200k	RDBMS-XML	Native XML	Python-RDF	Integration	Sesame-based
Q1	53	18.3	102.4	7.1	7.5
Q2	889	74.6	5357.3	40.0	66.7
Q3	262		284	11.5	43.4

300k	RDBMS-XML	Native XML	Python-RDF	Integration	Sesame-based
Q1	91.9	33.1	175.4	8.5	8.4

Q2	1731.6	133.7	8182.7	59.6	103.9
Q3	523.1		539.2	24.6	70.7

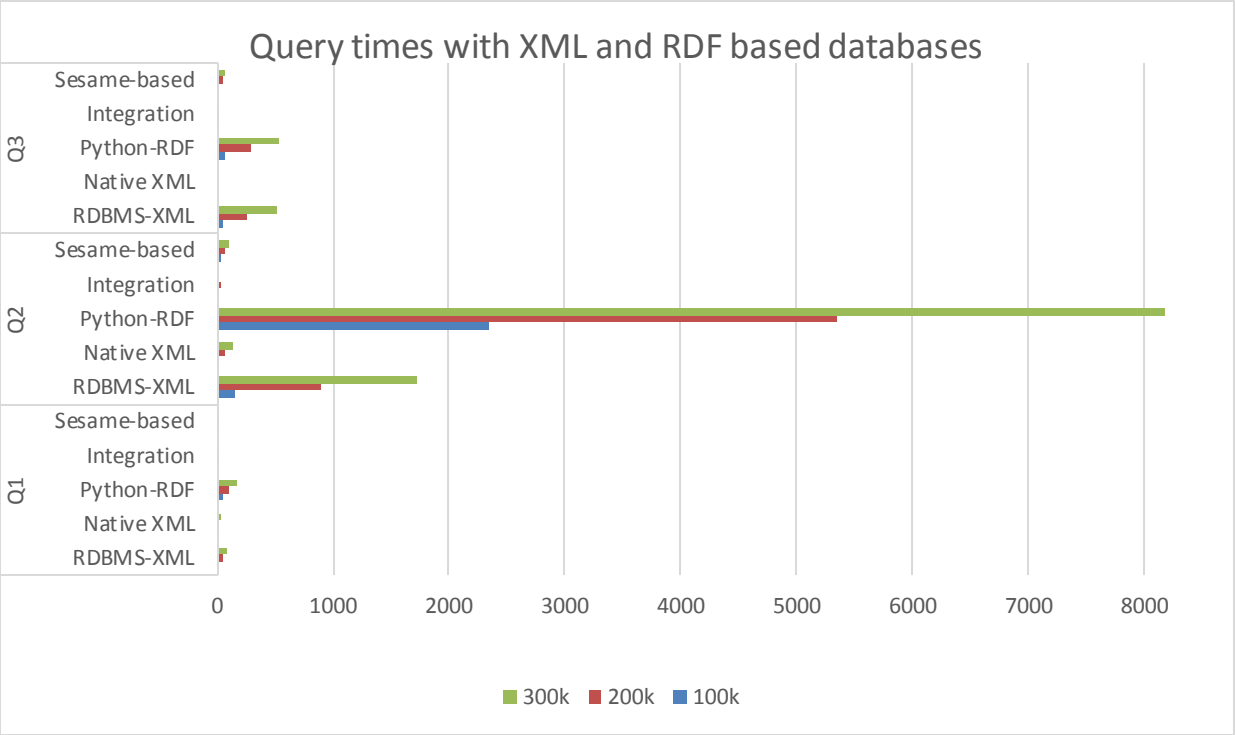

Figure 3: *Query times with XML and RDF Storage Solutions*

We can see that the native RDF databases most often provided better performance than their alternatives. However, in our article (Niinimaki, Heikkurinen, & Schmidt, 2019; Oracle, 2016), in addition to XML databases we tested MongoDB, a popular "noSQL" database. For the tests, we converted our documents into the JSON format used by MongoDB (for details, see (Banker, 2011)) and rewrote the queries using MongoDB's query language. We found the performance generally very good, especially with aggregation queries. In Table 3 we summarize the results, adding a further test with 1 million documents. Figure 4 illustrates the results. It must be noted that MongoDB's query language is quite different from SPARQL and therefore this approach cannot be used for all RDF storage/query needs. For details about the expressiveness of MongoDB's query language, see Botoeva et al. (Botoeva, Calvanese, Cogrel, & Xiao, 2018).

Table 3: *Query times with MongoDB*

	100k	200k	300k	1M
Q1	24.5	55.1	84.9	1062

Q2	1.2	4.2	4.3	14.5
Q3	5.2	5.2	10.5	823.9

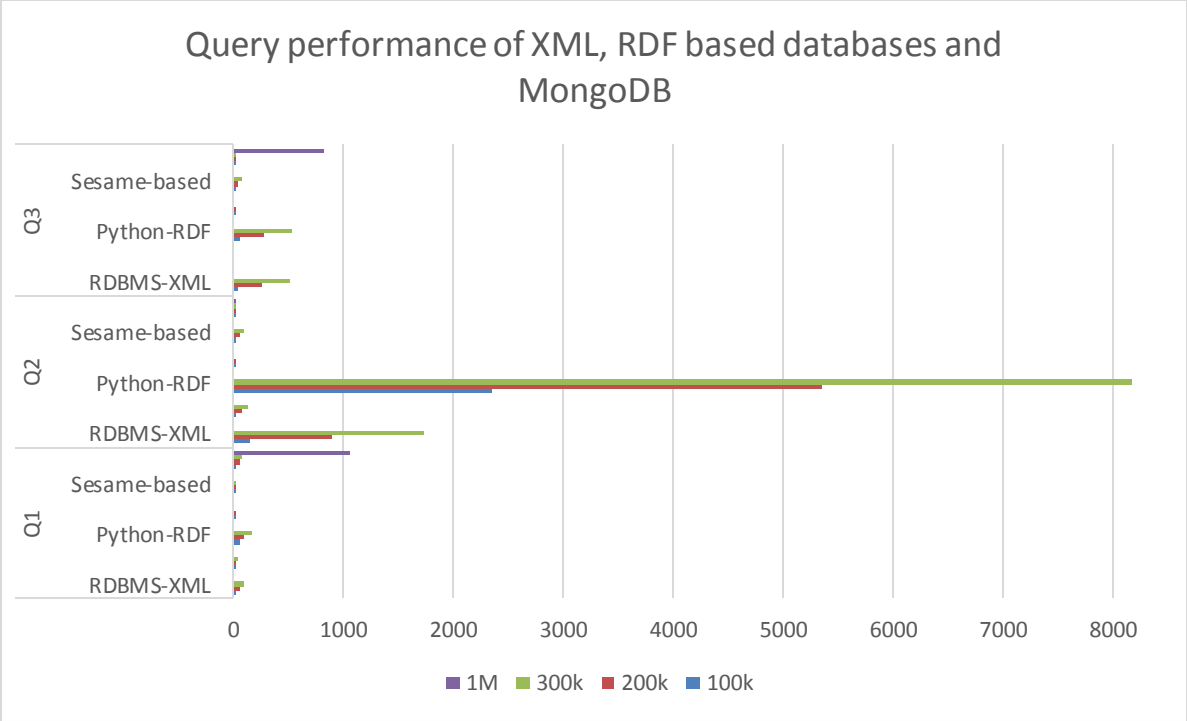

Figure 4: *Query times with XML, RDF based databases and MongoDB*

5. Summary and Discussion

In this paper, we have compared the query performance of RDF database packages. The test data is based on XML files of medical articles, converted into an RDF XML form. The documents originate from the U.S. National Institute of Health's PubMed collection. The queries represent "typical" tasks in an information system containing a database, namely:

Q1 List the publication year of all the documents in the database.

Q2 List the document ID's of all documents containing word "genitalia" anywhere in the document.

Q3 Find the article that is most cited by other articles in the collection.

Our test environment was a relatively high-end Linux server (a 24-core Xeon server with 32 GB memory). We tested three methods of storing RDF data persistently: a Python library with

a BerkeyleyDB back-end and two native, commercial RDF database products. All the methods were significantly faster than querying RDF files, but the native databases were faster than the Python library. Additionally, we demonstrated that MongoDB can work efficiently as a storage of some RDF-style data if the RDF structures are converted to JSON, and the SPARQL queries are converted to MongoDB's query language.

It's worth noticing that though an RDF graph is a graph, there are "graph databases" that are not meant for storing only RDF data. Notably, Neo4j presents their graph database simply as a database that exposes a graph data model (Robinson, Webber, & Eifrem, 2015). The graph data model in this case is "labeled property graph". Labeled property graphs contain nodes and relationships (arcs between nodes); nodes contain properties (key-value pairs); nodes can be labeled with one or more labels; relationships are named and directed and have a start and end node; and relationships can contain properties. Moreover, GraphQL, promoted by Facebook (Hartig & Pérez, 2017) is a design of a query language and a query processor API that follows some graph-style principles.

The limitations of this research are mainly related to our query model: we run simple queries without parallelization and in a single computer and database node.

There are several interesting directions for future research. Scalability, especially when processing queries in parallel is essential in modern database systems (Agrawal, El Abbadi, Das, & Elmore, 2011). We are currently researching clustered database solutions for RDF. On the other hand, large clusters use a lot of energy. Energy efficient query processing in clusters is another area of our research.

Acknowledgements: The research activities described on this paper have received support from the PROCESS project (https://www.process-project.eu/), that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777533.

References

Agrawal, D., El Abbadi, A., Das, S., & Elmore, A. J. (2011). Database scalability, elasticity, and autonomy in the cloud. International Conference on Database Systems for Advanced Applications (pp. 2-15). Springer. https://doi.org/10.1007/978-3-642-20149-3_2

Arenas, M., Gutierrez, C., & Pérez, J. (2009). Foundations of RDF databases. Reasoning Web International Summer School (pp. 158-204). Heidelberg: Springer. https://doi.org/10.1007/978-3-642-03754-2_4

Banker, K. (2011). MongoDB in action. Manning Publications.

Becker, C. (2008). RDF Store Benchmarks with DBpedia. Berlin: Freie Universitat Berlin.

Botoeva, E., Calvanese, D., Cogrel, B., & Xiao, G. (2018). Expressivity and complexity of MongoDB queries. 21st International Conference on Database Theory. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. https://doi.org/10.3233/IA-190023

Broekstra, J., Kampman, A., & Van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF schema. Proc. 1st International semantic web conference (pp. 54-68). Sardinia: Springer. https://doi.org/10.1007/3-540-48005-6_7

Donohoe, P., Sherman, J., & Mistry, A. (2015). The Long Road to JATS. Journal Article Tag Suite Conference (JATS-Con) Proceedings 2015.

Faye, D. C., & Curé, O. B. (2012). A survey of RDF storage approaches. Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées, 15, (pp 11-35).

Hartig, O., & Pérez, J. (2017). An initial analysis of Facebook's GraphQL language. AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web. Montevideo.

Levandoski, J. J., & Mokbel, M. F. (2009). RDF data-centric storage. 2009 IEEE International Conference on Web Services (pp. 911-918). IEEE. https://doi.org/10.1109/ICWS.2009.49

Miller, L., Seaborne, A., & Reggior, A. (2002). Three implementations of SquishQL, a simple RDF query language. Proc. International Semantic Web Conference. Heidelberg, Germany. https://doi.org/10.1007/3-540-48005-6_36

Morsey, M., Lehmann, J., Auer, S., & Ngomo, A. (2009). DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. Proc. International semantic web conference 2011, (pp. 1-24). Springer

Niinimaki, M., & Niemi, T. (2009). An ETL process for OLAP using RDF/OWL ontologies. Journal of Data Semantics, XIII, 97-119. https://doi.org/10.1007/978-3-642-03098-7_4

Niinimaki, M., & Thanisch, P. (2019). Dataspace Management for Large Data Sets. In P. Vasant, I. Litvinchev, & Marmolejo-Saucedo. J., Innovative Computing Trends and Applications (pp. 13-21). Springer. https://doi.org/10.1007/978-3-030-03898-4_2

Niinimaki, M., Heikkurinen, M., & Schmidt, J. (2019). Performance of XML databases., forthcoming.

Oracle. (2016). Oracle Spatial and Graph: Benchmarking a Trillion Edges RDF Graph. Oracle.

Robinson, I., Webber, J., & Eifrem, E. (2015). Graph Databases (2nd ed.). Sebastopol, CA: O'Reilly.

Schmidt, M., Schallhorn, T., Lausen, G., & Pinkel, C. (2009). SP2Bench: A SPARQL performance benchmark. IEEE International Conference on Data Engineering, 42. https://doi.org/10.1109/ICDE.2009.28

Steinbrook, R. (2005, April). Public Access to NIH-Funded Research. New England Journal of Medicine(352), 1739-1741. https://doi.org/10.1056/NEJMp058088

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. Proceedings of the 48th annual Southeast regional conference. ACM. https://doi.org/10.1145/1900008.1900067

W3C. (2004). RDF Primer - W3C Recommendation.

W3C. (2008). SPARQL Query Language for RDF, W3C Recommendation.

W3C. (2014). RDF 1.1 N-Triples, A line-based syntax for an RDF graph. Retrieved from https://www.w3.org/TR/n-triples/