MOCHA 2017 as a Challenge for Virtuoso

Mirko Spasić[1,2] and Milos Jovanovik[1,3]

[1] OpenLink Software, United Kingdom
[2] Faculty of Mathematics, University of Belgrade, Serbia
[3] Faculty of Computer Science and Engineering,
Ss. Cyril and Methodius University in Skopje, Macedonia
`{mspasic,mjovanovik}@openlinksw.com`

Abstract. The Mighty Storage Challenge (MOCHA) aims to test the performance of solutions for SPARQL processing, in several aspects relevant for modern Linked Data applications. The Virtuoso Server by OpenLink is a modern enterprise-grade solution for data access, integration, and relational database management, which provides a scalable RDF Quad Store. In this paper, we present the initial evaluation results of running the Social Network Benchmark queries over the provided challenge datasets, as part of our application for the MOCHA challenge. These initial results will serve as a guideline for improvements in Virtuoso which will then be tested as part of the MOCHA challenge.

Keywords: Virtuoso, Social Network Benchmark, ESWC, Mighty Storage Challenge, MOCHA, Benchmarks, Data Storage, Linked Data, RDF, SPARQL

1 Introduction

Triple stores are the heart of a growing number of Linked Data applications. This uncovers a growing need for representative benchmarks which will fairly summarize their strengths and weaknesses [1], allowing stakeholders to choose between technologies from different vendors according to their needs and use-cases. The HOBBIT project[4] aims to push the development of Big Linked Data processing solutions by providing a family of industry-relevant benchmarks through a generic evaluation platform – the HOBBIT Platform [2]. In the scope of the project, several challenges will be organized, with the goal of reaching system providers, familiarizing them with the benchmarks of their interest, as well as the platform itself. The Mighty Storage Challenge (MOCHA 2017)[5] is one of these challenges: it aims to test the performance of systems capable of answering SPARQL SELECT queries and processing INSERT queries. It consists of four tasks and OpenLink Software[6], with their RDF Quad Store, would like to participate in the following three:

[4] `https://project-hobbit.eu/`
[5] `https://project-hobbit.eu/challenges/mighty-storage-challenge/`
[6] `https://www.openlinksw.com/`

– Task 1: RDF Data Ingestion,
– Task 2: Data Storage, and
– Task 4: Browsing.

2 Virtuoso Server

Virtuoso[7] is a modern enterprise-grade solution for data access, integration, and relational database management. It operates with SQL Tables and/or RDF based Property/Predicate Graphs. Virtuoso was initially developed as a row-wise transaction oriented RDBMS with SQL federation, e.g. as a multi-protocol server providing ODBC/JDBC access to relational data stored either within Virtuoso itself or any combination of external relational databases. Besides catering for SQL clients, Virtuoso has a built-in HTTP server providing a DAV repository, SOAP and WS* protocol end-points and dynamic web pages in a variety of scripting languages. It was subsequently re-targeted as an RDF graph store with built-in SPARQL and inference [4, 5]. Recently, the product has been revised to take advantage of column-wise compressed storage and vectored execution [6].

The largest Virtuoso applications are in the RDF domain, with terabytes of RDF triples which do not fit into main memory. The excellent space efficiency of column-wise compression was the greatest incentive for the column store transition [6]. Additionally, this also makes Virtuoso an option for relational analytics. Finally, combining a schema-less data model with analytics performance is attractive for data integration in places with high schema volatility. Virtuoso has a shared cluster capability for scale-out. This is mostly used for large RDF deployments.

2.1 Triple Storage

The storage solution in Virtuoso is fairly conventional: a single table of four columns, named RDF_QUAD, holds one quad, i.e. a triple plus graph, per row. The columns are G for graph, P for predicate, S for subject and O for object. P, G and S are IRI IDs, for which Virtuoso has a custom data type, distinguishable at runtime from integer, even though internally this is a 32 or 64-bit integer. Since O is a primary key part, it is not desired to have long O values repeated in the index. Hence, Os of string type which are longer than 12 characters are assigned a unique ID and this ID is stored as the O of the quad table, while the mapping is stored in the RDF_IRI and RDF_PREFIX tables [4]. By default, and with the idea of faster execution, the table is represented as five covering indices, $PSOG$, $POSG$, SP, GS, and OP. In the first one, the quads are sorted primarily by predicate, then subject and object, and finally by graph. The structures of the other indices are analog to this one.

The compression is implemented at two levels. First, within each database page, Virtuoso stores distinct values only once and eliminates common prefixes

[7] `https://virtuoso.openlinksw.com/`

of strings. Without key compression, there are 75 bytes per triple with a billion-triple LUBM dataset (LUBM scale 8000). With compression, only 35 bytes per triple are present. Thus, when using 32-bit IRI IDs, key compression doubles the working set while sacrificing no random access performance. The benefits of compression are even better when using 64-bit IRI IDs. The second stage of compression involves applying "gzip" to database pages, which reduces their size to a third, even after key compression. This is expected, since indices are repetitive by nature, even if the repeating parts are shortened by key compression [4].

Internally, SPARQL queries are translated into SQL at the time of parsing the query. If all triples are in one table, the translation is straightforward, with *union* becoming an SQL *union* and *optional* becoming a left outer join [4]. All triple patterns from the SPARQL query should be translated to SQL as a self-join of the RDF_QUAD table, with conditions if there are common subjects, predicates and/or objects, e.g. a SPARQL query with n triple patterns will result with $n-1$ self-joins. Thus, the correct join order and join type decisions are difficult to make given only the table and column cardinalities for the RDF triple or quad table. Histograms for ranges of P, G, O, and S are also not useful [4]. The solution is to go look at the data itself when compiling the query, e.g. do data sampling.

3 Evaluation

3.1 Task 2: Data Storage

In this section, the initial evaluation of the Virtuoso triple store against the Social Network Benchmark (SNB) queries will be presented. The main reason why this evaluation should be considered as an initial one, is that the section contains only the query execution times, without review of the key performance indicator (KPI) of the task – throughput. The experiments were run on a dual Xeon(R) E5-2630 @ 2.33GHz machine, with 192GB RAM and 2 SSDs with 470GB, and consist of a single threaded execution for all queries, allowing the system to execute them as fast as possible. The aspect of multi-threaded runs will be taken into account later. The tested scale factors are 1 and 10. The link to the examples of concrete queries, together with training datasets, is available from the MOCHA challenge webpage[8]. The paper [7] describes the modifications of the dataset generator which are relevant for this task of the challenge. Table 1 contains the numbers of Persons and Posts in the datasets of different scale, along with the total numbers of triplets.

Table 2 shows the average execution times per query time, for the smallest dataset, and for the 10 times larger one. The execution times for Q1 are not available, since the current version of Virtuoso exhibits a problem when compiling the query. The slowest queries are $SelectQuery5$ and $SelectQuery9$, while the fastest ones are $SelectQuery8$ and $SelectQuery11$, which corresponds to the

[8] `http://hobbitdata.informatik.uni-leipzig.de/mighty-storage-challenge/`

Table 1. Dataset Sizes

Scale Factor	Persons	Posts	Triplets
1	9927	3.1M	47.4M
10	65678	29.2M	476.2M

Table 2. Average Query Execution Times (in milliseconds)

Query	SF1	SF10
SelectQuery1	-	-
SelectQuery2	44.67	95.77
SelectQuery3	43	331.38
SelectQuery4	21.39	442.4
SelectQuery5	1809.81	12417.87
SelectQuery6	75.41	87.48
SelectQuery7	1.75	138.26
SelectQuery8	8.24	19.5
SelectQuery9	7381.95	27613.17
SelectQuery10	62.36	362.92
SelectQuery11	3.43	13.8
SelectQuery12	131.02	163.53
SelectQuery13	23.01	90.34
SelectQuery14	368.26	2283.54

Query	SF1	SF10
ShortQuery1	40.34	45.76
ShortQuery2	27.09	43.29
ShortQuery3	5.53	8.05
ShortQuery4	3.41	3.44
ShortQuery5	1.74	1.56
ShortQuery6	4.1	4.83
ShortQuery7	8.14	8.45
UpdateQuery1	47.02	361.85
UpdateQuery2	2.42	22.86
UpdateQuery3	2.35	21.03
UpdateQuery4	7.26	64.39
UpdateQuery5	2.82	24.54
UpdateQuery6	10.4	81.47
UpdateQuery7	11.44	93.39
UpdateQuery8	4.32	35.38

same results exhibited when running the original SNB against a Virtuoso relational database [8]. Table 2 shows several irregularity, as well, which will serve as good guidelines for further improvements on the Virtuoso optimizer, e.g. the slow executions of *SelectQuery*6 against the smallest dataset. This means that Virtuoso has a problem with finding an optimal query execution plan for these queries against the dataset in question, most probably due to the wrong estimate of cardinalities. The simplicity of short queries caused the expected, almost equal execution times (without noticeable increase) on both datasets.

4 Conclusion and Future Work

This paper should be considered as a part of the registration process of MOCHA 2017, a challenge included in the Challenges Track of ESWC 2017, specially for the following tasks: RDF Data Ingestion, Data Storage and Browsing. A short overview of the Virtuoso system has been presented, with a focus on its RDF Storage engine. The evaluation part of the paper contains the measurements from the initial experiment, the second task of MOCHA 2017 (Data Storage), running the SNB queries against the dataset in question of different sizes. That section represents an excellent guideline as to where the Virtuoso optimizer should be improved.

As future work, the further evaluation has been planned, using real-world workloads, consisting of specified query mixes, where reads and updates are bundled together, and queries are run concurrently. Virtuoso will be tested with more dataset sizes and especially larger datasets, stressing its scalability. For this purpose, the HOBBIT platform will be used. We foresee improvements of the query optimizer, driven by the current evaluation. After the challenge, the official results will be presented.

Acknowledgments. This work has been supported by the H2020 project HOBBIT (GA no. 688227).

References

1. Morsey M., Lehmann J., Auer S., Ngonga Ngomo AC.: DBpedia SPARQL Benchmark Performance Assessment with Real Queries on Real Data. In: Aroyo L. et al. (eds.) The Semantic Web ISWC 2011. ISWC 2011. Lecture Notes in Computer Science, vol 7031. Springer, Berlin, Heidelberg (2011)
2. Ngonga Ngomo AC., Röder M.: HOBBIT: Holistic Benchmarking for Big Linked Data. In: ERCIM News 2016 - 105 (2016)
3. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
4. Erling O., Mikhailov I.: RDF Support in the Virtuoso DBMS. In: Pellegrini T. et al. (eds.) Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems 2009., pp. 7–24. Springer Berlin Heidelberg (2009)
5. Erling O., Mikhailov I.: Virtuoso: RDF Support in a Native RDBMS. In: de Virgilio R. at al. (eds.) Semantic Web Information Management: A Model-Based Perspective 2010., pp. 501–519. Springer Berlin Heidelberg (2010)
6. Erling O.: Virtuoso, a Hybrid RDBMS/Graph Column Store. `https://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSArticleVirtuosoAHybridRDBMSGraphColumnStore`
7. Spasić M., Jovanovik M., Prat-Perez A.: An RDF Dataset Generator for the Social Network Benchmark with Real-World Coherence. In Proceedings of the Workshop on Benchmarking Linked Data (BLINK 2016), pages 18-25. (2016)
8. Kaufmann M., Social Network Benchmark Interactive Workload Full Disclosure Report. `http://ldbcouncil.org/sites/default/files/LDBC_SNB_I_20150427_SF300_virtuoso.pdf`