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. Virtuoso, by OpenLink Software, 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 a short overview of Virtuoso with a focus on RDF triple storage and SPARQL query execution. Furthermore, we showcase the final results of the MOCHA 2017 challenge and its tasks, along with a comparison between the performance of our system and the other participating systems.

Keywords: Virtuoso, Social Network Benchmark, Mighty Storage Challenge, 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 are being 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. Its goal is to provide objective measures for how well current systems perform on real tasks of industrial relevance and detect bottlenecks of existing systems to further their development towards practical usage. The challenge was accepted and presented

[4] https://project-hobbit.eu/
[5] https://project-hobbit.eu/challenges/mighty-storage-challenge/

in the Extended Semantic Web Conference (ESWC)[6] in 2017, held in Portoroz, Slovenia. Even though four tasks were initially planned, MOCHA 2017 consisted of three tasks in the end. OpenLink Software[7], with our RDF Quad Store – *Virtuoso 8.0 Commercial Edition (beta release)* – participated in all of three:

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

In Section 2, we will briefly present our system, Virtuoso, putting our focus on its quad storage, represented as a relational table, and its translation engine for converting SPARQL queries to SQL. We will describe all preparatory actions requested by the challenge organizers which the system had to fulfill in Section 3. After that, in Section 4, we will present the evaluation results of our system achieved during the challenge, along with a comparison with the outcomes of the other participants. Finally, Section 5 concludes the paper, and contains guidelines for future work and further improvement of our system.

2 Virtuoso Universal Server

Virtuoso[8] is a modern enterprise-grade solution for data access, integration, and relational database management. It is a database engine hybrid that combines the functionality of a traditional relational database management system (RDBMS), object-relational database (ORDBMS), virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. 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, i.e. as a multi-protocol server providing ODBC and JDBC access to relational data stored either within Virtuoso itself or any combination of external relational databases. Besides catering to 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 schemaless 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.

[6] `https://2017.eswc-conferences.org/`
[7] `https://www.openlinksw.com/`
[8] `https://virtuoso.openlinksw.com/`

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.

2.2 Compression

The compression is implemented at two levels. First, within each database page, Virtuoso stores distinct values only once and eliminates common prefixes of strings. Without key compression, there are 75 bytes per triple with a billion-triple LUBM[9] 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 [4].

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].

2.3 Translation of SPARQL Queries to SQL

Internally, SPARQL queries are translated into SQL at the time of query parsing. If all triples are in one table, the translation is straightforward. In the next paragraph, we give a couple of simple SPARQL queries, and their simplified SQL translations.

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 [4]. For example, if a SPARQL query asks for first and last names of 10 people, as shown in the example on Figure 1, its SQL translation will be similar to the query given at Figure 2. The functions `__i2idn`, `__bft` and `__ro2sq` are used for translation of RDF IRIs to the internal datatypes mentioned in the Subsection 2.1, and vice versa.

A SPARQL *union* becomes an SQL *union* (Figures 3 and 4) and *optional* becomes a left outer join (Figures 5 and 6), while SPARQL *group by*, *having*,

[9] `http://swat.cse.lehigh.edu/projects/lubm/`

```
SELECT ?first ?last
FROM <http://project-hobbit.eu/task2>
WHERE {
  ?person a snvoc:Person .
  ?person snvoc:firstName ?first .
  ?person snvoc:lastName ?last .
}
LIMIT 10
```

Fig. 1: SPARQL Query 1

```
SELECT __ro2sq (t1.O) AS first, __ro2sq (t2.O) AS last
FROM RDF_QUAD AS t0
     INNER JOIN RDF_QUAD AS t1
     ON ( t0.S = t1.S )
     INNER JOIN RDF_QUAD AS t2
     ON ( t0.S = t2.S AND t1.S = t2.S )
WHERE
  t0.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
  AND
  t0.P = __i2idn ( __bft ( 'rdf:type', 1))
  AND
  t0.O = __i2idn ( __bft ( 'snvoc:Person', 1))
  AND
  t1.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
  AND
  t1.P = __i2idn ( __bft ( 'snvoc:firstName', 1))
  AND
  t2.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
  AND
  t2.P = __i2idn ( __bft ( 'snvoc:lastName', 1))
LIMIT 10
```

Fig. 2: SQL Translation of SPARQL Query 1

order by and aggregate functions are translated to their SQL corresponding counterparts. Figure 4 shows an optimization trick: both members of a *union* have *limit* clauses, as well as the main *select*, providing that both parts of the query will not find more than 10 results.

```
SELECT ?name
FROM <http://project-hobbit.eu/task2>
WHERE {
  { ?person snvoc:firstName ?name . }
  UNION
  { ?person snvoc:lastName ?name . }
}
LIMIT 10
```

Fig. 3: SPARQL Query 2

```
SELECT __ro2sq ( tmp.name ) AS name
FROM (
    SELECT tmp1.O AS name
    FROM RDF_QUAD AS tmp1
    WHERE
      tmp1.G = __i2idn (  __bft ( 'http://project-hobbit.eu/task2', 1))
      AND
      tmp1.P = __i2idn (  __bft ( 'snvoc:firstName', 1))
    LIMIT 10
    UNION ALL
    SELECT tmp2.O AS name
    FROM RDF_QUAD AS tmp2
    WHERE
      tmp2.G = __i2idn (  __bft ( 'http://project-hobbit.eu/task2', 1))
      AND
      tmp2.P = __i2idn (  __bft ( 'snvoc:lastName', 1))
    LIMIT 10
) AS tmp
LIMIT 10
```

Fig. 4: SQL Translation of SPARQL Query 2

In conclusion, 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, i.e. do data sampling.

```
SELECT ?first ?last
FROM <http://project-hobbit.eu/task2>
WHERE {
  ?person a snvoc:Person .
  OPTIONAL { ?person snvoc:firstName ?first } .
  OPTIONAL { ?person snvoc:lastName ?last } .
}
LIMIT 10
```

Fig. 5: SPARQL Query 3

```
SELECT __ro2sq ( tmp3.first ) AS first, __ro2sq ( tmp5.last ) AS last
FROM RDF_QUAD AS tmp1
  LEFT OUTER JOIN (
    SELECT tmp2.S AS person,
           tmp2.O AS first
    FROM RDF_QUAD AS tmp2
    WHERE
      tmp2.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
      AND
      tmp2.P = __i2idn ( __bft ( 'snvoc:firstName', 1))
  ) AS tmp3
  ON ( tmp1.S = tmp3.person )
  LEFT OUTER JOIN (
    SELECT tmp4.S AS person,
           tmp4.O AS last
    FROM RDF_QUAD AS tmp4
    WHERE
      tmp4.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
      AND
      tmp4.P = __i2idn ( __bft ( 'snvoc:lastName', 1))
  ) AS tmp5
  ON ( tmp1.S = tmp5.person )
WHERE
  tmp1.G = __i2idn ( __bft ( 'http://project-hobbit.eu/task2', 1))
  AND
  tmp1.P = __i2idn ( __bft ( 'rdf:type', 1))
  AND
  tmp1.O = __i2idn ( __bft ( 'snvoc:Person', 1))
LIMIT 10
```

Fig. 6: SQL Translation of SPARQL Query 3

3 Challenge Prerequisites for Participation

In order to be a part of the challenge, the organizers proposed a set of requirements that participants had to conform to. The participants had to provide:

- A storage system that processes SPARQL INSERT queries
- A storage solution that can process SPARQL SELECT queries
- A solution as a Docker image that abides by the technical specifications, i.e. the MOCHA API

Virtuoso has build-in SPARQL support, so we only had to pack it as a Docker image and develop a System Adapter, a component of the HOBBIT platform[10] which implements the requested API and enables communication between the benchmark and the Virtuoso instance. We developed an instance of the System Adapter for the commercial version of Virtuoso 8.0, which shares the same Docker container with it. Its code is publicly available on GitHub[11].

After this component initializes itself, it starts receiving data from the Data Generator, i.e. the files representing the benchmark dataset. When all files are accepted, indicated by a signal from the Data Generator (the other part of the platform that is in charge for creating the dataset for the benchmark), the System Adapter starts loading the dataset into the Virtuoso instance. Upon completion, it sends a signal to the other components indicating it is ready to start answering the SPARQL queries, which are then sent by the Task Generator, a component which creates the tasks, i.e. the SELECT and INSERT queries. All

[10] `http://master.project-hobbit.eu/`
[11] `https://github.com/hobbit-project/DataStorageBenchmark`

accepted queries are then executed against our system, and their answers are sent to the Evaluation Storage, for validation against the expected answers and for measuring the achieved efficiency of the system.

4 Evaluation

In this section, we present the official results of the challenge for all its tasks.

4.1 Task 1: RDF Data Ingestion

The aim of this task is to measure the performance of SPARQL query processing systems when faced with streams of data from industrial machinery in terms of efficiency and completeness. This benchmark, called ODIN (StOrage and Data Insertion beNchmark), increases the size and velocity of RDF data used in order to evaluate how well can a system store streaming RDF data obtained from the industry. The data is generated from one or multiple resources in parallel and is inserted using SPARQL INSERT queries. At some points in time, the SPARQL SELECT queries check the triples that are actually inserted [9].

This task has three main KPIs:

- Triples per Second: For each stream, a fraction of the total number of triples that were inserted during that stream divided by the total time needed for those triples to be inserted.
- Average Answer Time: A delay between the time stamp that the SELECT query has been executed and the time stamp that the results are send to the Evaluation Storage.
- Correctness: A recall of each SELECT query by comparing the expected and retrieved results.

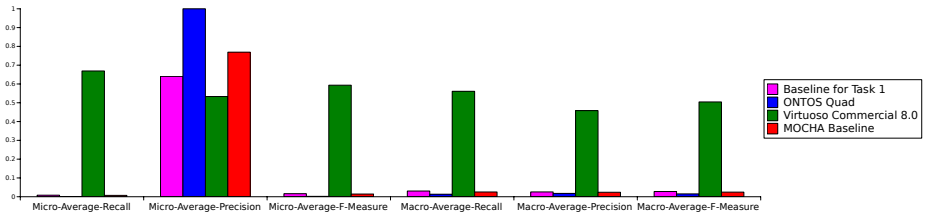


Fig. 7: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Task 1 of MOCHA 2017.

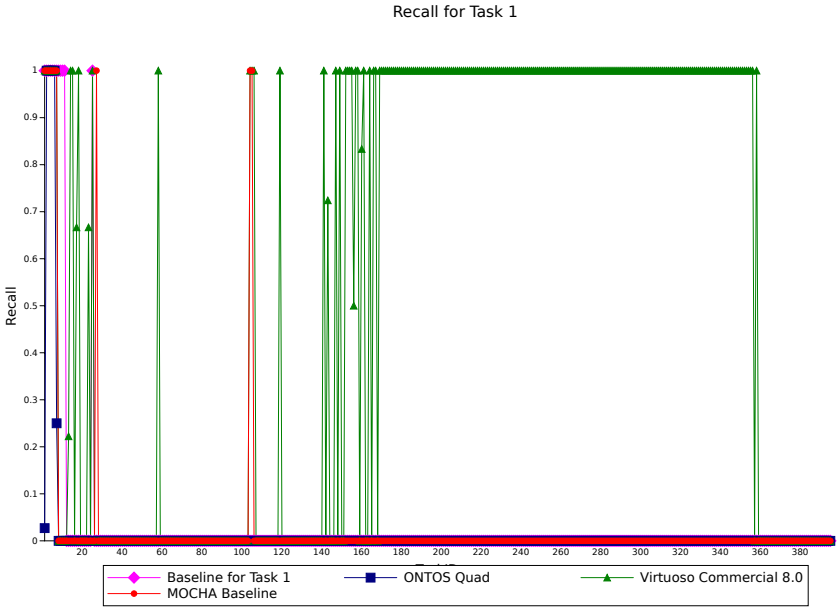


Fig. 8: Recall for Task 1 of MOCHA 2017.

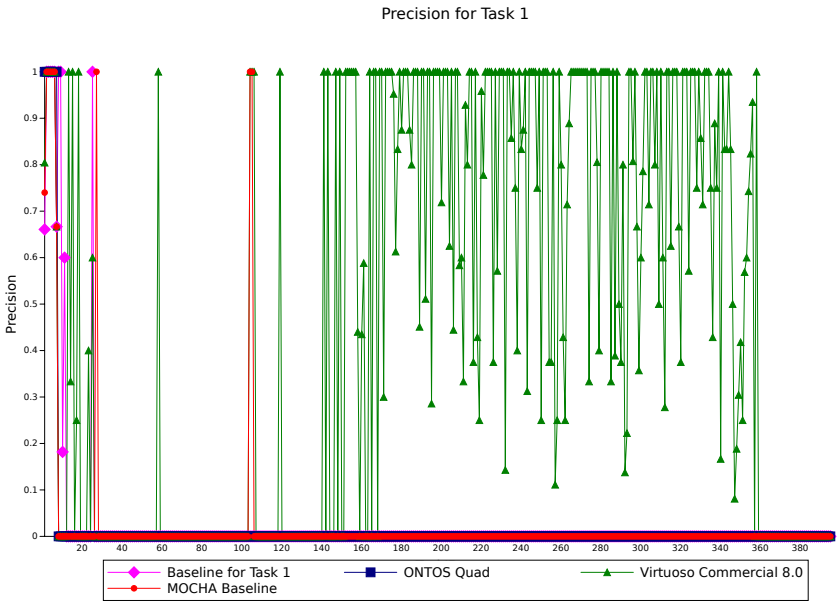


Fig. 9: Precision for Task 1 of MOCHA 2017.

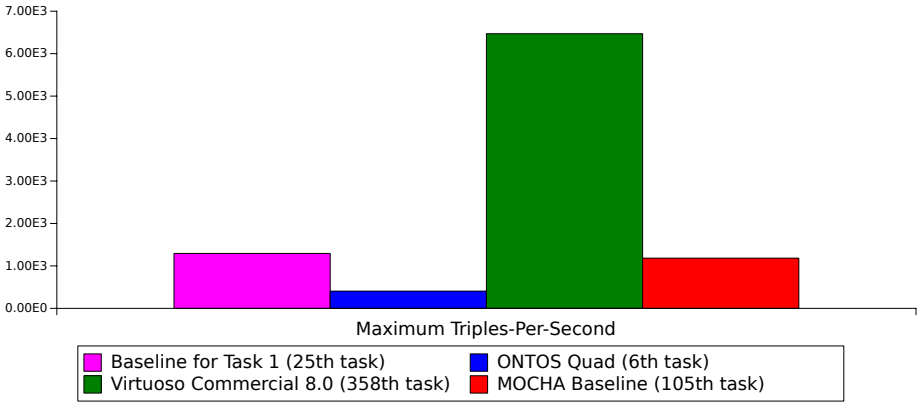


Fig. 10: Maximum Triples-per-Second for Task 1 of MOCHA 2017.

Results: Here, we give the official results of our system, *Virtuoso 8.0 Commercial Edition (beta release)*, against ODIN, achieved during the challenge and published by the organizers of the challenge. The task organizers at MOCHA 2017 specified the benchmark parameters for the actual challenge run, in order to achieve the desired size and velocity of the RDF data. The values of the parameters were: number of insert queries per stream = 100, population of generated data = 10,000, number of data generators - agents = 4.

Our system, *Virtuoso Commercial 8.0*, had by far the best performance compared to the other systems, in terms of Macro and Micro-Average Precision, Recall, and F-measure (Figure 7). By observing Figures 8 and 9, it is obvious that our system was able to store and retrieve much more triples throughout the whole benchmark, than the other systems. In terms of maximum Triples-per-Second, based on Figure 10, our system has just confirmed its convincing overall victory in this task, with a one order of magnitude better score. This results were announced by the organizers – our system had a best overall performance in terms of data ingestion and retrieval.

4.2 Task 2: Data Storage

The goal of this task is to measure how data storage solutions perform with interactive, simple, read, SPARQL queries as well as complex ones, accompanied with a high insert data rate via SPARQL UPDATE queries, in order to mimic real use-cases where READ and WRITE operations are bundled together. This task also tests systems for their bulk load capabilities [10].

The main KPIs of this task are:

- Bulk Loading Time: The total time in milliseconds needed for the initial bulk loading of the dataset.
- Throughput: The average number of tasks executed per second.

– Correctness: The number of SPARQL SELECT queries whose result set is different from the result set obtained from the triple store used as a gold standard.

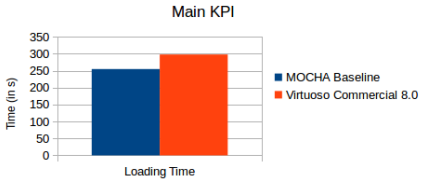


Fig. 11: Loading Time for Task 2 of MOCHA 2017.

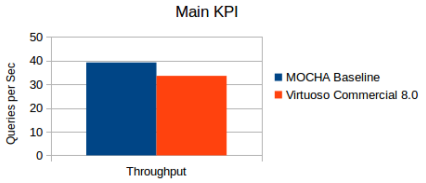


Fig. 12: Throughput for Task 2 of MOCHA 2017.

Fig. 13: Long Queries for Task 2 of MOCHA 2017.

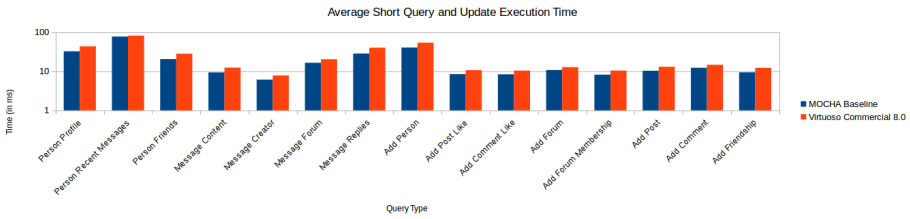


Fig. 14: Short Queries and Updates for Task 2 of MOCHA 2017.

Results: Based on the results from the KPIs, shown in Figures 11, 12, 13 and 14, the winning system for the task was *Virtuoso 7.2 Open-Source Edition* by OpenLink Software, that was used as a baseline system for all tasks in the challenge. Our system, *Virtuoso 8.0 Commercial Edition*, was slightly slower, while the third system was not able to finish the experiment in the requested time, i.e. it exhibited a timeout, thus its scores are not present at the figures.

4.3 Task 4: Browsing

The task on faceted browsing checks existing solutions for their capabilities of enabling faceted browsing through large-scale RDF datasets, that is, it analyses their efficiency in navigating through large datasets, where the navigation is driven by intelligent iterative restrictions. The goal of the task is to measure the performance relative to dataset characteristics, such as overall size and graph characteristics [11].

The evaluation is based on the following performance KPIs:

- Throughput: The time required by the system is measured for the two tasks – facet count and instance retrieval – separately. The results are returned in a score function computing number of returned queries per second.
- Correctness: The facet counts are being checked for correctness. For each facet count, the distance of the returned count to the correct count in terms of absolute and relative value is recorded. For each instance retrieval the benchmark collects the true positives, the false positives and false negatives to compute an overall precision, recall and F1-score.

Results: Similar to the first task, the only two systems that managed to finish the task within the requested time slot are shown on the Figures 15, 16, 17a and 17b, representing the main KPIs of the Faceted Browsing Benchmark. Based on that, the organizers announced a tie between our system and the baseline system. The Open-Source edition of Virtuoso was slightly faster, but the Commercial edition performed better on the correctness of the facet counts queries.

4.4 Overall Winner

As a summary, our system had a significant victory in Task 1; the baseline system was slightly better in Task 2; and there was a tie in Task 4. Based on the results in each task separately and the overall results, during the closing ceremony of the ESWC 2017 conference, the challenge organizers declared our system as the overall winner of the MOCHA 2017.

5 Conclusion and Future Work

This paper should be considered as an extended participant paper of MOCHA 2017, a challenge included in the Challenges Track of ESWC 2017, intended to

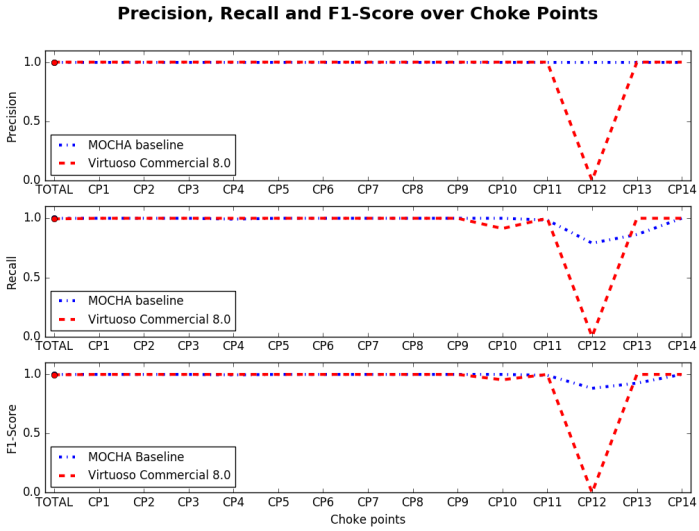


Fig. 15: Instance Retrieval: Correctness for Task 4 of MOCHA 2017.

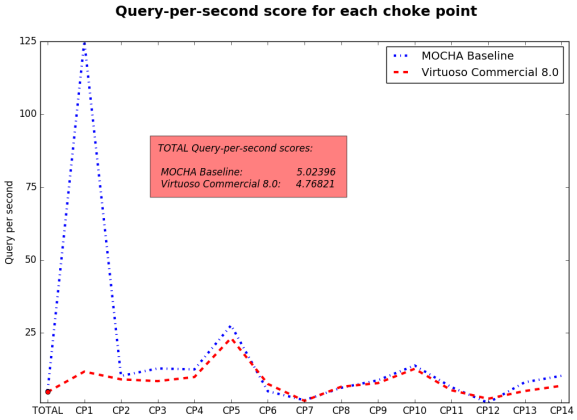
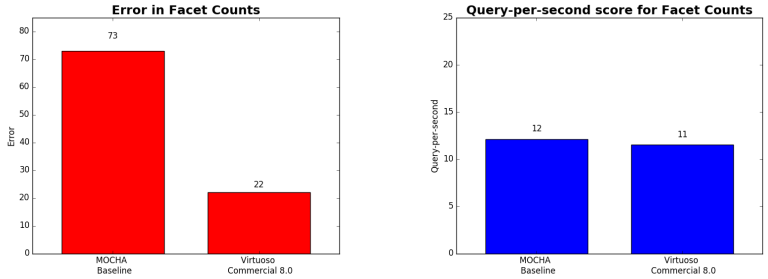


Fig. 16: Instance Retrieval: Speed for Task 4 of MOCHA 2017.

test RDF storage and SPARQL systems for the following tasks: RDF Data Ingestion, Data Storage and Browsing. Thus, a short overview of Virtuoso Universal Server has been presented, with a focus on its RDF storage engine and the internal SPARQL to SQL translation. The evaluation part of the paper contains the official measurements from the challenge and its tasks. This section represents an excellent guideline as to where our Virtuoso optimizer should be improved.



(a) Correctness (b) Speed

Fig. 17: Facet Counts for Task 4 of MOCHA 2017.

As future work, a further evaluation has been planned against newer versions of the challenge benchmarks. For example, in Task 2, real-world workloads will be used, 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.

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-Pérez 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
9. Georgala K., Data Extraction Benchmark for Sensor Data. https://project-hobbit.eu/wp-content/uploads/2017/06/D3.1.1_First_Version_of_the_Data_Extraction_Benchmark_for_Sensor_Data.pdf
10. Jovanovik M., Spasić M., First Version of the Data Storage Benchmark. https://project-hobbit.eu/wp-content/uploads/2017/06/D5.1.1_First_version_of_the_Data_Storage_Benchmark.pdf
11. Petzka H., First Version of the Faceted Browsing Benchmark. https://project-hobbit.eu/wp-content/uploads/2017/06/D6.2.1_First_Version_FacetedBrowsing.pdf