MOCHA2017: The Mighty Storage Challenge at ESWC 2017

Kleanthi Georgala[1], Mirko Spasić[2], Milos Jovanovik[2], Henning Petzka[3], Michael Röder[1,4], and Axel-Cyrille Ngonga Ngomo[1,4]

[1] AKSW Research Group, University of Leipzig, Augustusplatz 10, Leipzig, Germany, 04109
georgala@informatik.uni-leipzig.de
[2] OpenLink Software, United Kingdom
{mspasic, mjovanovik}@openlinksw.com
[3] Fraunhofer Institute IAIS, Schloss Birlinghoven, Sankt Augustin Germany, 53757
henning.petzka@iais.fraunhofer.de
[4] Paderborn University, DICE Group, Pohlweg 51, Pacerborn, Germany, D-33098
axel.ngonga@upb.de

Abstract. The aim of the Mighty Storage Challenge (MOCHA) at ESWC 2017 was to test the performance of solutions for SPARQL processing in aspects that are relevant for modern applications. These include ingesting data, answering queries on large datasets and serving as backend for applications driven by Linked Data. The challenge tested the systems against data derived from real applications and with realistic loads. An emphasis was put on dealing with data in form of streams or updates.

1 Introduction

Triple stores and similar solutions are the backbone of most applications based on Linked Data. Hence, devising systems that achieve an acceptable performance on real datasets and real loads is of central importance for the practical applicability of Semantic Web technologies. This need is emphasized further by the constant growth of the Linked Data Web in velocity and volume [1], which increases the need for storage solutions to ingest and store large streams of data, perform queries on this data efficiently and enable high performance in tasks such as interactive querying scenarios, the analysis of industry 4.0 data and faceted browsing through large-scale RDF datasets.

The lack of comparable results on the performance on storage solutions for the variety of tasks which demand time-efficient storage solutions was the main motivation behind this challenge. Our main aims while designing the challenge were to

- provide comparable performance scores 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.

2 The MOCHA Challenge

2.1 Overview

The MOCHA challenge was carried out within the Extended Semantic Web Conference (ESWC) 2017, which took place from May 28th, 2016 to June 1st, 2017 in Portoroz, Slovenia. Given the goals aforementioned, we designed the challenge to encompass the following tasks:

1. Ingestion of RDF data streams;
2. RDF data storage and
3. Browsing RDF data.

2.2 Tasks

Task 1: RDF Data Ingestion The aim of task 1 was to measure the performance of SPARQL query processing systems when faced with streams of data from industrial machinery in terms of efficiency and completeness. Our benchmark ODIN (StOrage and Data Insertion beNchmark) was designed to test the abilities of tripe stores to store and retrieve streamed data. The experimental setup was hence as follows: We used a mimicking algorithm to generate RDF data similar to real data. We increased the size and velocity of RDF data used in our benchmarks to evaluate how well a given storage solution was able to deal with streamed RDF data derived from the mimicking approach aforemetionen. The data was generated from one or multiple resources in parallel and was inserted using SPARQL INSERT queries. SPARQL SELECT queries were used to check when the system completed the processing of the particular triples.

The input data for this task consists of data derived from mimicking algorithms trained on real industrial datasets. Each training dataset included RDF triples generated within a predefined period of time (e.g., a production cycle). Each event (e.g., each sensor measurement or tweet) had a timestamp that indicates when it was generated. The datasets differed in size regarding the number of triples per second. During the test, data was generated using data agents (in form of distributed threads). An agent is a data generator who is responsible for inserting its assigned set of triples into a triple store, using a SPARQL INSERT query. Each agent emulated a dataset that covered the duration of the benchmark. All agents operated in parallel and were independent of each other. As a result, the storage solution benchmarked had to support concurrent inserts. The insertion of a triple was based on its generation timestamp. To emulate the ingestion of streaming RDF triples produced within large time periods within a shorter time frame, we used a time dilatation factor that allowed rescaling data inserts to shorter timeframes. Our benchmark hence allows for testing the performance of the ingestion in terms of precision and recall by deploying datasets that vary in volume (size of triples and timestamps), and used different dilatation values, various number of agents and different size of update queries.

Task 2: Data Storage The goal of task 2 was to measure how data storage solutions perform with interactive, simple, read, SPARQL queries as well as complex ones. We used our Data Storage Benchmark (DSB)[5] for this purpose. DSB runs simple and complex SPARQL SELECT queries, 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. The queries were designed to stress the system under test in different choke-point areas, while being credible and realistic.

The dataset used with DSB is an RDF dataset derived from the LDBC Social Network Benchmark (SNB) dataset[6], but modified so that its characteristics further match real-world RDF datasets [4]. These modifications were considered necessary due to the high structuredness of the SNB RDF dataset, which makes it more similar to RDB dataset, as opposed to real-world RDF datasets, such as DBpedia. Our DSB dataset was pre-generated in several different scale factors (sizes), and split in two parts: the dataset that should be loaded by the system under test, and a set of update streams containing update queries.

The benchmark was started by loading the dataset into the data storage solution under test, after which the benchmark queries were executed. The execution format was defined as a *query mix* which mimics the activities of a real-world online social network, e.g. there were more executions of short lookup queries than complex queries, each complex query was followed by one or more short lookups, etc. The current version of DSB supports sequential execution of the queries, which allows the storage system to use all available resources for the current query. As a final step, the results from the executed queries were evaluated against expected results. The main KPIs of this task were bulk loading time, average task execution time, average task execution time per query type, number of incorrect answers, and throughput.

Task 3: Faceted Browsing The task on faceted browsing checked existing solutions for their capabilities of enabling faceted browsing through large-scale RDF datasets. Faceted browsing stands for a session-based (state-dependent) interactive method for query formulation over a multi-dimensional information space, where it is the efficient transition from one state to the next that determines the users experience. The goal was to measure the performance relative to a number of types of transitions, and thereby analysing a system's efficiency in navigating through large datasets. We made use of the Benchmark on Faceted Browsing[7] on the HOBBIT platform[8] to carry out the testing of systems.

As the underlying dataset, a transport dataset of linked connections was used. The transport dataset was provided by a data generator PoDiGG[9][5] containing train connections between stations on an artificially created map. For the inte-

[5] https://github.com/hobbit-project/DataStorageBenchmark
[6] http://www.ldbcouncil.org/benchmarks/snb
[7] https://github.com/hobbit-project/faceted-benchmark
[8] master.project-hobbit.eu/
[9] https://github.com/PoDiGG/podigg-lc

gration of delays into the dataset the Transport Disruption Ontology[10][2] was used, which models possible events that can disrupt the schedule of transport plans. The dataset had to be loaded into the database of a participating system at the beginning of the benchmark.

A participating system was subsequently required to answer a sequence of SPARQL queries, which simulate browsing scenarios through the underlying dataset. The browsing scenarios were motivated by natural navigation behaviour of a user (such as a data scientist) through the data, as well as to check participating systems on a list of 14 choke points defined by certain types of transitions. The queries involved temporal (time slices), spatial (different map views) and structural (ontology related) aspects. In addition to these so-called instance retrievals, the benchmark included facet counts. Facet counts are SPARQL COUNT queries for retrieving the number of instances behind a certain facet selection, i.e. the number of instances that would remain after applying a certain additional filter restriction. This resulted in a total workload of 173 SPARQL queries divided up into 11 browsing scenarios.

3 Benchmarking Platform

All three tasks are carried out using the HOBBIT benchmarking platform[11]. This platform offers the execution of benchmarks to evaluate the performance of systems. For every task of the MOCHA challenge, a benchmark has been implemented. The benchmarks are sharing a common API which eases the work of the challenge participants.

For the benchmarking of the participant systems a server cluster has been used. Each of these systems could use up to three servers of this cluster each of them having 256 GB RAM and 32 cores. This enabled the benchmarking of monolythical as well as distributed solutions.

4 The Challenge

4.1 Overview

The MOCHA2017 challenge ran on 22nd May, 2017 and its results were presented during the ESWC 2017 closing ceremony. Three system participated in all tasks:

- *Virtuoso Open-Source Edition 7.2*[12], developed by OpenLink Software, that served as the baseline system for all MOCHA2017 tasks (*MOCHA Baseline*),
- *QUAD*[13], developed by Ontos, and
- *Virtuoso Commercial Edition 8.0 (beta)*[14], developed by OpenLink Software.

[10] https://transportdisruption.github.io/

[11] HOBBIT project webpage: http://project-hobbit.eu/
HOBBIT benchmarking platform: master.project-hobbit.eu
HOBBIT platform source code: https://github.com/hobbit-project/platform

[12] https://virtuoso.openlinksw.com/

[13] http://ontos.com/

[14] https://virtuoso.openlinksw.com/

4.2 Results & Discussion

Task 1: RDF Data Ingestion

KPIs Our evaluation consists of three KPIs:

– Recall, Precision and F-Measure: The INSERT queries created by each data generator were sent into a triple store by bulk load. After a stream of INSERT queries was performed against the triple store, a SELECT query was conducted by the corresponding data generator. In Information Retrieval, Recall and Precision were used as relevance measurements and were defined in terms of retrieved results and relevant results for a single query. Recall is the fraction of relevant documents that were successfully retrieved and precision is the fraction of the retrieved documents that are relevant to a query. F-measure is the harmonic mean of Recall and Precision. For our set of experiments, the relevant results for each SELECT query were created prior to the system benchmarking by inserting and querying an instance of the Jena TDB storage solution.
Additionally, we computed:

$$Macro{-}Average{-}Precision = \frac{\sum_{i=1}^{\lambda} Precision_i}{\lambda} \tag{1}$$

$$Macro{-}Average{-}Recall = \frac{\sum_{i=1}^{\lambda} Recall_i}{\lambda} \tag{2}$$

where λ is the number of SELECT queries performed against the storage solution during the execution of the benchmark and Micro and Macro-Average Recall, Precision and F-measure of the whole benchmark. The aforementioned measurements $Precision_i$ and $Recall_i$ are the precision and recall of the i-th SELECT query. We also calculated $Macro{-}Average{-}F{-}measure$ as the harmonic mean of Equations 1 and 2.

$$Micro{-}Average{-}Precision = \frac{\sum_{i=1}^{\lambda} |\{relevant\ results_i\} \cap \{retrieved\ results_i\}|}{\sum_{i=1}^{\lambda} |\{retrieved\ results_i\}|} \tag{3}$$

$$Micro{-}Average{-}Recall = \frac{\sum_{i=1}^{\lambda} |\{relevant\ results_i\} \cap \{retrieved\ results_i\}|}{\sum_{i=1}^{\lambda} |\{relevant\ results_i\}|} \tag{4}$$

where the $\{relevant\ results_i\}$ and $\{retrieved\ results_i\}$ are the relevant and the retrieved results of the i-th SELECT query resp. We also calculated $Micro{-}Average{-}F{-}measure$ as the harmonic mean of Equations 3 and 4.
We have to mention that misclassifications between the expected and received results does not necessarily mean that the triple stores are prone to

misclassify results or to have a bad performance, but that there are missmatches for results sets between Jena TDB and the storage solution.
- Triples per second: at the end of each stream and once the corresponding SELECT query was performed against the system, we measured the triples per second as a fraction of the total number of triples that were inserted during that stream. This was divided by the total time needed for those triples to be inserted (begin point of SELECT query - begin point of the first INSERT query of the stream). We provided the maximum value of the triples per second of the whole benchmark. The maximum triples per second value was calculated as the triples per second value of the last stream with Recall value equal to 1.
- Average answer time: we reported the average answer 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. The first aforementioned time stamp was generated by the benchmark when the SELECT query was sent to the system and the second time stamp was generated by the platform when the results of the corresponding SELECT query were sent to the storage.

Experiment set-up ODIN require a set of parameters to be executed, that are independent of the triple store. For MOCHA, all three systems were benchmarked using the same values. Each triple store was allowed to communicate with the HOBBIT [3] platform for at most $25\,mins$. The required parameters and their corresponding values for MOCHA are:

- **Duration of the benchmark**: It determines the time interval of the streamed data. Value for MOCHA2017 = $600,000\,ms$.
- **Name of mimicking algorithm output folder**: The relative path of the output dataset folder. Value for MOCHA2017 = *output_data/*.
- **Number of insert queries per stream**: This value is responsible for determining the number of INSERT SPARQL queries after which a SELECT query is performed. Value for MOCHA2017 = 100.
- **Population of generated data**: This value determines the number of events generated by a mimicking algorithm for one Data Generator. Value for MOCHA2017 = $10,000$.
- **Number of data generators - agents**: The number of indepedent Data Generators that send INSERT SPARQL queries to the triple store. Value for MOCHA2017 = 4.
- **Name of mimicking algorithm**: The name of the mimicking algorithm to be invoked to generate data. Value for MOCHA2017 = *TRANSPORT_DATA*.
- **Seed for mimicking algorithm**: The seed value for a mimicking algorithm. Value for MOCHA2017 = 100.
- **Number of task generators - agents**: The number of indepedent Task Generators that send SELECT SPARQL queries to the triple store. Value for MOCHA2017 = 1.

Results for Task 1 By observing Figure 1, we notice that *Virtuoso Commercial 8.0* has by far the best performance compared to the other two systems in

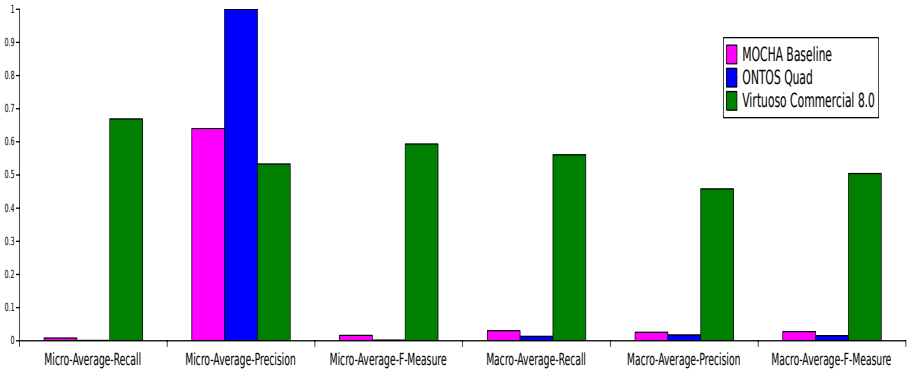

Fig. 1. Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure of *MOCHA Baseline*, *ONTOS Quad* and *Virtuoso Commercial 8.0* for MOCHA2017.

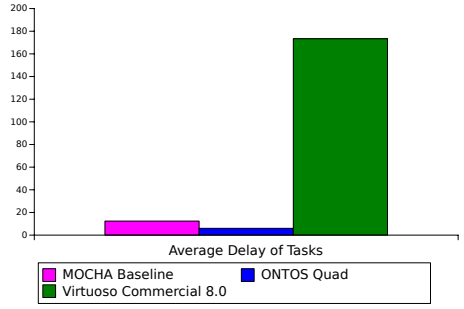

Fig. 2. Average Delay of tasks of *MOCHA Baseline*, *ONTOS Quad* and *Virtuoso Commercial 8.0* for MOCHA2017.

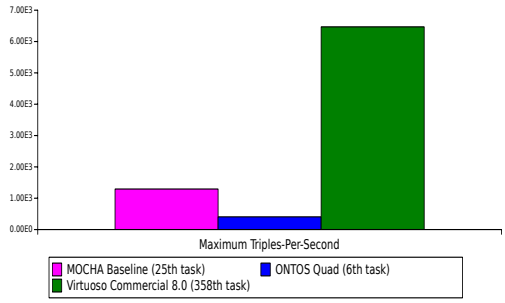

Fig. 3. Maximum Triples-per-Second of *MOCHA Baseline*, *ONTOS Quad* and *Virtuoso Commercial 8.0* for MOCHA2017.

terms of Macro and Micro-Average Precision, Recall and F-measure. *Virtuoso Commercial 8.0* was able to store and retrieve more triples through out the whole benchmark. However, the maximum performance value was achieved for Micro-Average Recall = 0.67, which indicates that the miss classifications between the Jena TDB and *Virtuoso Commercial 8.0* were still high on average. Additionally, since the Micro-Average values were higher compared to the Macro-Average values, we can conclude by stating that *Virtuoso Commercial 8.0* was able to retrieve more relevant triples to a SELECT query, for tasks with higher quantity of expected results.

Furthermore, we also notice that the Micro-Average Precision of *ONTOS Quad* is higher that the other systems. The Micro-Average values are calculated only when there are non-zero received results for a task. *ONTOS Quad* was able to retrieve results for the first 7 tasks and for the remaining 388 tasks, the system

returned 0 received results or included 0 relevant results in its received result set, we notice that *Virtuoso Commercial 8.0* is the only system that was able to retrieve non-zero results for the majority of the SELECT queries.

In terms of maximum Triples-per-Second, based on Figure 3, we notice that *Virtuoso Commercial 8.0* was able to achieve the highest maximum TPS at the latest task possible. It receives the last recall value of 1 at task 358 (out of 395), whereas the other systems have issues with recall at much earlier stages of the benchmark. Especially for the *ONTOS Quad* system, we see that its recall drops significantly after the 6th SELECT query.

Also, we need to mention that *ONTOS Quad* and *Virtuoso Commercial 8.0* were not able to perform all select queries within 25 *mins*. *ONTOS Quad* was not able to send results to the evaluation storage throughout the whole benchmark, whereas *Virtuoso Commercial 8.0* was not able to execute SELECT queries after 358 tasks, which is one of the reasons why its recall drops to 0.

Finally, we present the task delay for each task for all systems in Figure 2. We notice that all systems have a relatively low task average delay over the set of SELECT queries. Whereas *Virtuoso Commercial 8.0* has a monotonically ascending task delay function, that drops to 0 after the 358th task, since the system is no longer available because it exceeded the maximum allowed time to process queries.

Task 2: Data Storage

KPIs. The main KPIs of this task are:

- **Bulk Loading Time:** The total time in milliseconds needed for the initial bulk loading of the dataset.
- **Average Task Execution Time:** The average SPARQL query execution time.
- **Average Task Execution Time Per Query Type:** The average SPARQL query execution time per query type.
- **Number of Incorrect Answers:** 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.
- **Throughput:** The average number of tasks executed per second.

Experiment set-up. The Data Storage Benchmark has parameters which need to be set in order to execute the benchmark for this task. These parameters are independent of the triple store which is evaluated. The required parameters are:

- **Number of operations:** This parameter represents the total number of SPARQL queries that should be executed against the tested system. This number includes all query types: simple SELECT queries, complex SELECT queries and INSERT queries. The ratio between them, e.g. the number of queries per query type, has been specified in a query mix in such a way that each query type has the same impact on the overall score of the benchmark.

This means that the simpler and faster queries are present much more frequently than the complex and slower ones. The value of this parameter for MOCHA was 15,000 operations.

– **Scale factor:** The DSB can be executed using different sizes of the dataset, i.e. with different scale factors. The scale factor for MOCHA was 1, i.e. the smallest DSB dataset.

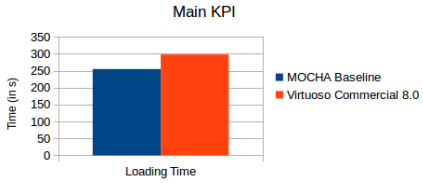

Fig. 4. Loading Time

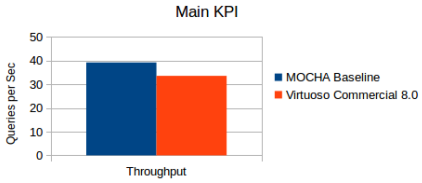

Fig. 5. Throughput

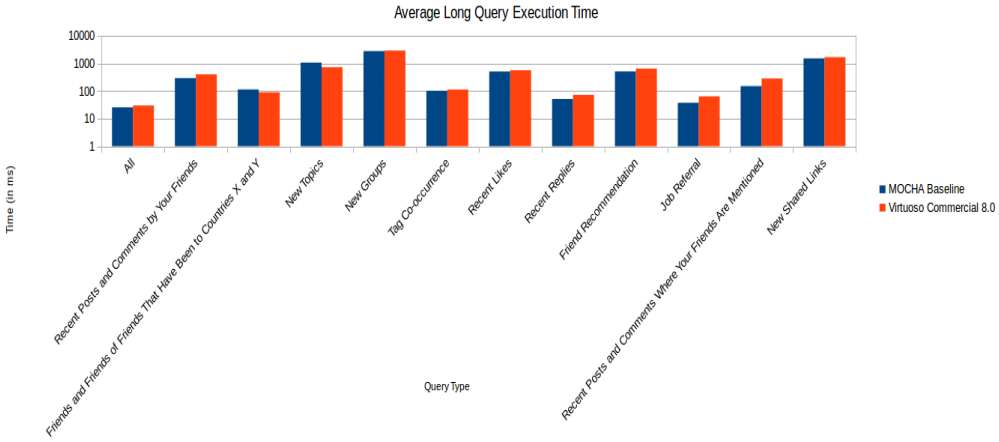

Fig. 6. Long Queries

Results for Task 2 Three systems applied and were submitted for Task 2: Virtuoso 7.2 Open-Source Edition by OpenLink Software, Virtuoso 8.0 Commercial Edition (beta release) by OpenLink Software, and QUAD by Ontos. Unfortunately, QUAD was not able to finish the experiment in the requested time (30 minutes), i.e. it exhibited a timeout.

Based on the results from the KPIs, shown in Figures 4, 5, 6 and 7, the winning system for the task was Virtuoso 7.2 Open-Source Edition by OpenLink Software.

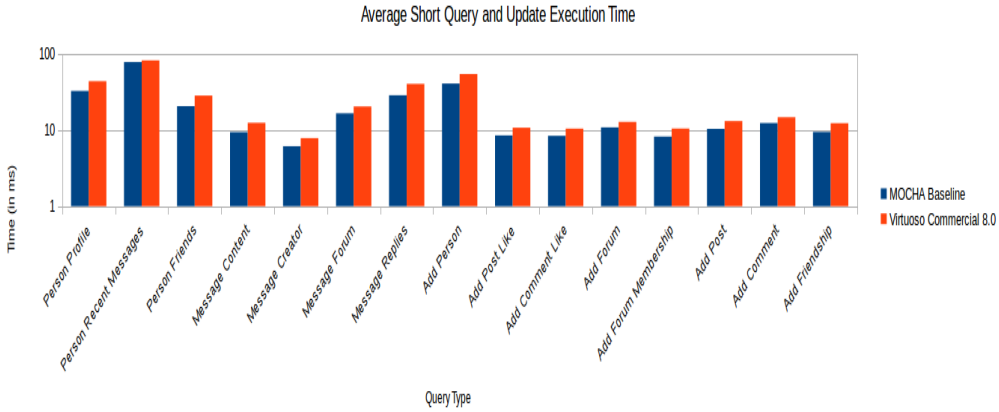

Fig. 7. Short Queries and Updates

Task 3: Faceted Browsing

KPIs. For the evaluation, the received results from the participating system were compared with the expected ones. Results were returned in form of several key performance indicators:

The performance on instance retrievals was measured by a query-per-second score, by precision, recall and F1-score. Next to results for the full workload, the values were recorded for each of the 14 choke point individually. The list of choke points reads as follows:

1. Find all instances which (additional to satisfying all restrictions defined by the state within the browsing scenario) have a certain property value
2. Find all instances which (additionally) realize a certain property path with any value
3. Find all instances which (additionally) have a certain value at the end of a property path
4. Find all instances which (additionally) have a property value lying in a certain class
5. For a selected class that a property value should belong to, select a subclass
6. Find all instances that (additionally) have numerical data lying within a certain interval behind a directly related property
7. Similar to 6, but now the numerical data is indirectly related to the instances via a property path
8. Choke points 6 and 7 under the assumption that bounds have been chosen for more than one dimension of numerical data
9. Choke points 6,7,8 when intervals are unbounded and only an upper or lower bound is chosen
10. Go back to the instances of a previous step by unselecting previously chosen facets

11. Change the solution space to instances in a different class while keeping the current filter selections (Entity-type switch)
12. Choke points 3 and 4 with advanced property paths involved
13. Choke points 1 through 4 where the property path involves traversing edges in the inverse direction
14. Additional numerical data restrictions at the end of a property path where the property path involves traversing edges in the inverse direction

For facet counts, we measured the accuracy of participating systems in form of the deviation from the correct and expected count results. Additionally, we computed the query-per-second score for the corresponding COUNT queries.

Experiment set-up The Faceted Browsing Benchmark required only one parameter which needed to be set in order to execute the benchmark for this task. This parameter consisted of a random seed, whose change alters the SPARQL queries of the browsing scenarios. The dataset was fixed and comprised about 1 million triples.

Results for Task 3 Three systems were submitted for Task 3: Virtuoso 7.2 Open-Source Edition by OpenLink Software which served as the MOCHA baseline, Virtuoso 8.0 Commercial Edition (beta release) by OpenLink Software, and QUAD by Ontos. Unfortunately, QUAD was not able to finish the experiment in the requested time (30 minutes), i.e. it exhibited a timeout. In Figure 8 and Figure 9, we display the results on instance retrievals. We see that both systems experienced problems on choke point number 12, which corresponds to filtering for the realisation of a certain property path (i.e., the task is to find all instances that, additionally to satisfying all restrictions defined by the state within the browsing scenario, realize a certain property path), and where the property path is of a rather complicated form. For example, complicated paths include those containing circles, or property paths where multiple entries need to be avoided.

Consider now the query-per-second score in Figure 9 for instance retrievals. Aside from the peculiar spike at choke point 2, the performance of both the open and the commercial version of Virtuoso are very similar with a slight advantage for the open source version. Interestingly, the query-per-second score of both system is the lowest for choke points 6 -8, which all correspond to selections of numerical data at the end of a property or property path.

In Figure 10 and 11 we see the performance on count queries. Again, we see the slight advantage in the query-per-second score of the open source Virtuoso version serving as the MOCHA baseline. On the other hand, the commercial version of Virtuoso made less errors in returning the correct counts.

Overall, this resulted in a tie on this task, as both systems had very similar results with both having their slight advantage on one task or the other.

5 Conclusion

The goal of MOCHA2017 was to test the performace of storage solutions in terms of data ingestion, query answering and faceted browsing againist large

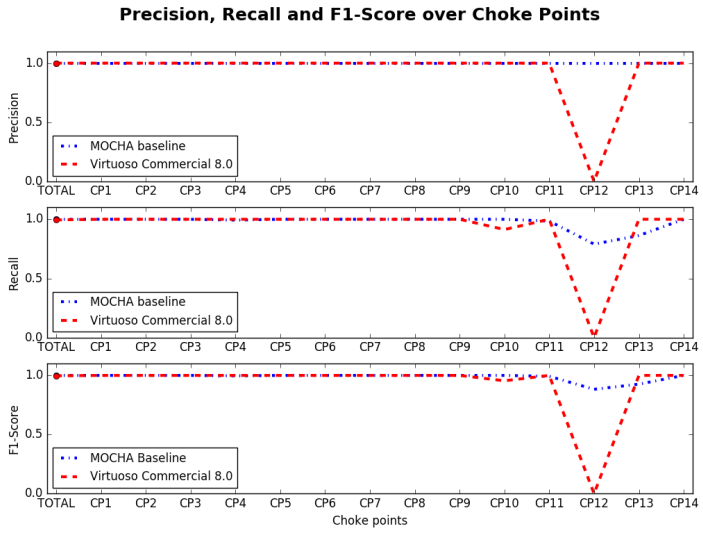

Fig. 8. Instance Retrieval - Accuracy

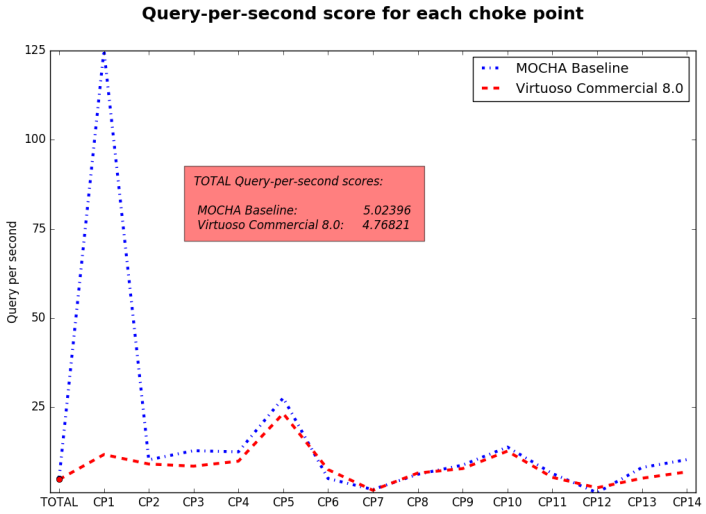

Fig. 9. Instance Retrieval - Query-per-second score

datasets. We benchmarked and evaluated three triple stores and presented a detailed overview and analysis of our experimental set-up, KPIs and results. Overall, our results suggest that while the scalability of triple stores is improving, a need for scalable distributed solutions remains. As part of our future work, we

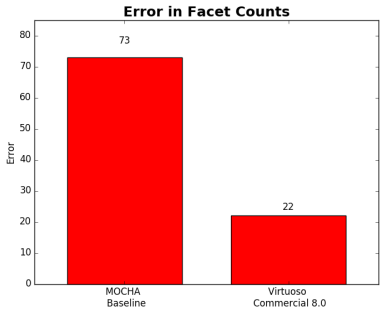

Fig. 10. Facet Counts - Accuracy

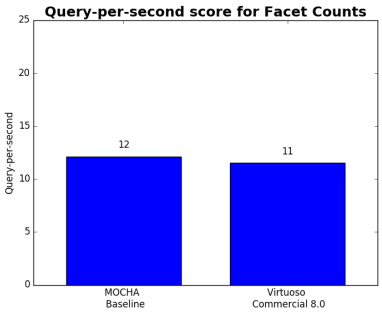

Fig. 11. Facet Counts - Query-per-second score

will benchmark more triple storage solutions by scaling over the volume and velocity of the RDF data and use a diverse number of datasets to test the scalability of our approaches.

References

1. Sören Auer, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. *Introduction to Linked Data and Its Lifecycle on the Web*, pages 1–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
2. David Corsar, Milan Markovic, Peter Edwards, and John D. Nelson. *The Transport Disruption Ontology*, pages 329–336. Springer International Publishing, Cham, 2015.
3. Axel-Cyrille Ngonga Ngomo, Alejandra García-Rojas, and Irini Fundulaki. HOBBIT: Holistic Benchmarking of Big Linked Data. *ERCIM News*, 2016(105), 2016.
4. Mirko Spasić, Milos Jovanovik, and Arnau Prat-Pérez. 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.
5. Ruben Taelman, Ruben Verborgh, Tom De Nies, and Erik Mannens. PoDiGG: A Public Transport RDF Dataset Generator. In *Proceedings of the 26th International Conference Companion on World Wide Web*, April 2017.