# MOCHA2018: The Mighty Storage Challenge at ESWC 2018

Kleanthi Georgala[1], Mirko Spasić[2], Milos Jovanovik[2], Vassilis Papakonstantinou[3], Claus Stadler[1], Michael Röder[4,5], and Axel-Cyrille Ngonga Ngomo[4,5]

[1] AKSW Research Group, University of Leipzig, Goerdellering 9, Leipzig, Germany, 04109
{georgala,cstadler}@informatik.uni-leipzig.de
[2] OpenLink Software, London, United Kingdom
{mspasic,mjovanovik}@openlinksw.com
[3] Institute of Computer Science-FORTH, Greece
papv@ics.forth.gr
[4] Paderborn University, DICE Group, Warburger Str. 100, Pacerborn, Germany, D-33098
{michael.roeder,axel.ngonga}@uni-paderborn.de
[5] Institute for Applied Informatics (InfAI) e.V., AKSW Group, Hainstrae 11, Leipzig, Germany, D-04109

**Abstract.** The aim of the Mighty Storage Challenge (MOCHA) at ESWC 2018 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 (1) provide objective measures for how well current systems (including 2 commercial systems,

which have already expressed their desire to participate) perform on real tasks of industrial relevance and (2) detect bottlenecks of existing systems to further their development towards practical usage.

## 2 The MOCHA Challenge

### 2.1 Overview

MOCHA2018 took place in conjunction with the 15th European Semantic Web Conference (ESWC 2018), 3rd-7th June 2018, Heraklion, Crete, Greece, for the second consecutive year[6] As last year, the aim of the Mighty Storage Challenge 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 proposed challenge will test the systems against data derived from real applications and with realistic loads. An emphasis will be put on dealing with changing data in form of streams or updates. We designed the challenge to encompass the following tasks:

1. Sensor Streams Benchmark, that measured how well systems can ingest streams of RDF data.
2. Data Storage Benchmark, that measured how data stores perform with different types of queries.
3. Versioning RDF Data Benchmark, that measured how well versioning and archiving systems for Linked Data perform when they store multiple versions of large data sets. This task is introduced for the first time this year.
4. Faceted Browsing Benchmark, that measured for how well solutions support applications that need browsing through large data sets.

### 2.2 Tasks

**Task 1: Sensor Streams Benchmark** The aim of this task 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. For this task, we developed ODIN (StOrage and Data Insertion beNchmark), that was designed to test the abilities of tripe stores to store and retrieve streamed data. Our goal was to measure the performance of triple stores by storing and retrieving RDF data, considering two main choke points: (1) scalability (Data volume): Number of triples per stream and number of streams and (2) time complexity (Data velocity): Number of triples per second. 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.

[6] https://2018.eswc-conferences.org/

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 SPARQL INSERT queries. Each agent emulated a dataset that covered the duration of the benchmark. All agents operated in parallel and were independent of each other. The insertion of a triple was based on its generation timestamp. SPARQL SELECT queries were used to check when the system completed the processing of the particular triples. 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 time frames.

**Task 2: Data Storage Benchmark (DSB)** This task consists of an RDF benchmark that measures how data storage solutions perform with interactive read SPARQL queries. Running the queries is accompanied with high insert rate of the data (SPARQL INSERT queries), in order to mimic realistic application scenarios where READ and WRITE operations are bundled together. Typical bulk loading scenarios are supported. The queries and query mixes are designed to stress the system under test in different choke-point areas, while being credible and realistic. The benchmark is based on and extends the Social Network Benchmark (SNB), developed under the auspices of the Linked Data Benchmark Council (LDBC)[7]. LDBC introduced a new choke-point driven methodology for developing benchmark workloads, which combines user input with input from expert systems architects [2]. The dataset generator developed for the previously mentioned benchmark is modified in DSB in order to produce synthetic RDF datasets available in different sizes, but more realistic and more RDF-like. The structuredness of the dataset is in line with real-world RDF datasets unlike the LDBC Social Network Benchmark dataset, which is designed to be more generic and very well structured. For the second version of the DSB benchmark – the version used in this challenge – we introduced parallel execution of the benchmark queries, which simulates a real-world workload for the tested system. Also, it is possible to increase / decrease the speed of query issuing, i.e. to modify the speed of the social network activity simulation. More details about this benchmark can be found in [4] and [7].

**Task 3: Versioning RDF Data Benchmark** The aim of this task was to test the ability of versioning systems to efficiently manage evolving Linked Data datasets and queries evaluated across multiple versions of these datasets. To do so, we developed the Semantic Publishing Versioning Benchmark (SPVB). SPVB acts like a Benchmark Generator, as it generates both the data and the queries needed to test the performance of the versioning systems. It is not tailored to any versioning strategy (the way that versions are stored) and can produce data of different sizes, that can be altered in order to create arbitrary numbers of versions using configurable insertion and deletion ratios. The data generator of

[7] http://www.ldbcouncil.org/

SPVB uses the data generator of Linked Data Benchmark Council[7] (LDBC) Semantic Publishing Benchmark[8] (SPB) as well as real DBpedia[9] data. The generated SPARQL queries are of different types (eight in total) and are partially based on a subset of the 25 query templates defined in the context of DBpedia SPARQL Benchmark[5] (DBPSB).

**Task 4: Faceted Browsing Benchmark** This goal of this task is to determine the performance and correctness characteristics of triple stores in faceted browsing scenarios. Faceted browsing thereby stands for a session-based (state-dependent) interactive method for query formulation over a multi-dimensional information space. For this purpose, we developed the *Hobbit Faceted Browsing Benchmark* which analyses SPARQL-based navigation through a dataset. The benchmark's data generator supports generation of datasets in the transportation domain. While the schema is fixed, the amount of instance data can be scaled up to virtually arbitrary sizes. A benchmark run consists of running a sequence of *scenarios*, whereas a scenario is comprised of a sequence of SPARQL that simulate related faceted browsing *interactions*, such as adding constraints or navigating to related set of resources. Conceptually, each interaction causes a *transition* from the current state of the faceted browsing system to a new one and thus implies changes in the solution space, i.e. the set of matching resources, facets, facet values and respective counts. The benchmark defines 11 scenarios with between 8 and 11 interactions, amounting to 172 queries in total. Every interaction of a scenario is thereby associated with one out of the 14 *choke-points* depicted in Figure 1. A choke-point thus represents a certain type of transition and is described in more detail in [6]. The overall scores computed by the benchmark are based on the performance and correctness characteristics at the level of choke-points.

## 3 Benchmarking Platform

All four tasks are carried out using the HOBBIT benchmarking platform[10]. This platform offers the execution of benchmarks to evaluate the performance of systems. For every task of the MOCHA challenge, a benchmark was 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.

[8] http://ldbcouncil.org/developer/spb
[9] http://wiki.dbpedia.org/
[10] HOBBIT project webpage: http://project-hobbit.eu/
HOBBIT benchmarking platform: master.project-hobbit.eu
HOBBIT platform source code: https://github.com/hobbit-project/platform

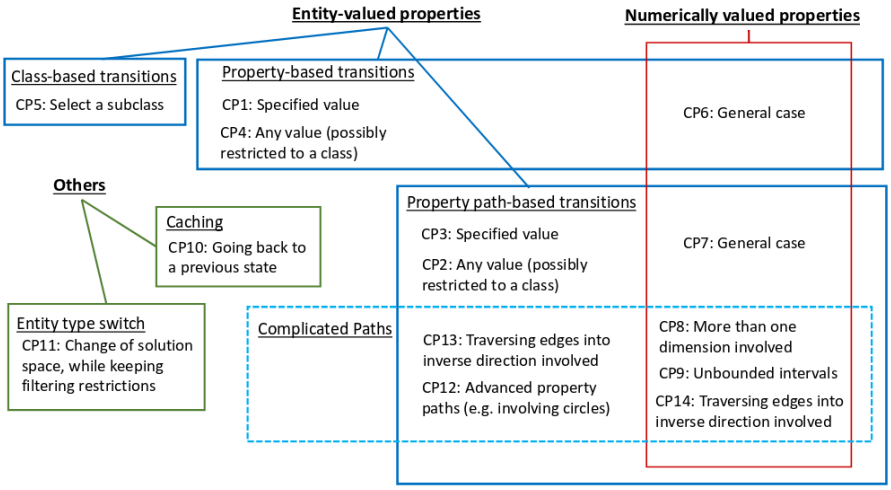



Fig. 1: Overview of the choke-points defined in the faceted browsing benchmark

## 4 The Challenge

### 4.1 Overview

The MOCHA2018 challenge ran on 18th May, 2018 and its results were presented during the ESWC 2018 closing ceremony. Five system participated in all tasks:

- Virtuoso v8.0, developed by OpenLink Software.[11] It is a modern enterprise-grade solution for data access, integration, and relational database management.
- Baseline Virtuoso Open Source, developed also by OpenLink Software and is the open source version of Virtuoso. [12]
- Blazegraph [13] that is an ultra-scalable, high-performance graph database with support for the Blueprints and RDF/SPARQL APIs.
- Graph DB Free 8.5 is a family of highly-efficient, robust and scalable RDF databases. It streamlines the load and use of linked data cloud datasets GraphDB Free implements the RDF4J [14]
- Apache Jena Fuseki 3.6.0 is a SPARQL server. It provides the SPARQL 1.1 protocols for query and update as well as the SPARQL Graph Store protocol. [15]

We had two additional systems participating for task 3, OSTRICH developed by IDLab, Department of Electronics and Information Systems, Ghent University imec and R43ples [3].

[11] https://virtuoso.openlinksw.com/
[12] https://github.com/openlink/virtuoso-opensource
[13] https://www.blazegraph.com/
[14] http://rdf4j.org/about/
[15] https://jena.apache.org/documentation/fuseki2/

### 4.2 Results & Discussion

**Task 1: Sensor Streams Benchmark**

*KPIs* ODIN's KPIs are divided into two categories: Correctness and Efficiency. Correctness is measured by calculating Recall, Precision and F-Measure. First, the INSERT queries created by each data generator will be send into a triple store by bulk load. After a stream of INSERT queries is performed against the triple store, a SELECT query will be 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. 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 will computed Macro and Micro Average Precision, Recall and F-measure to measure the overall performance of the system. Efficiency is measured by using the following metrics: (1) Triples-Per-Second that measures the triples per second as a fraction of the total number of triples that were inserted during a stream. This is 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. (2) Average Delay of Tasks as the task delay between the time stamp that the SELECT query (task) was sent to the system and the time stamp that the results were send to HOBBIT's storage for evaluation. We report both the average delay of each task and the average delay of task for the whole experiment.

*Experiment set-up* For the MOCHA2018 experiments, all parameters were set to their default values, except from the following:

- **Population of generated data** $= 10,000$
- **Number of data generators - agents** $= 4$
- **Number of insert queries per stream** $= 20$

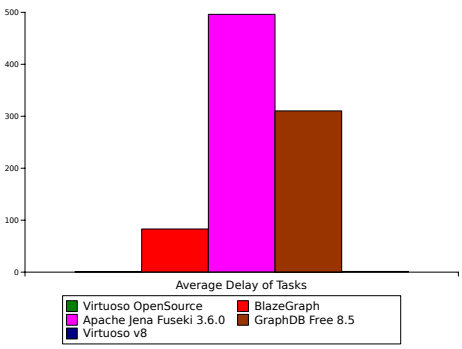
*Results for Task 1* Figure 2 illustrates the correctness KPIs for our baseline systems and Virtuoso Commercial 8.0 under the MOCHA2018 configuration. Beginning with the system that had the worse overall performance, Blazegraph, we observe that all KPIs, apart from Micro-Average-Precision, receive the lowest values compared to the other systems. The high value obtained in Micro-Average-Precision indicates that Blazegraph is able to retrieve correct results for the set of SELECT queries that it answered, but its low Macro-Average-Precision value shows that the set of SELECT queries that it managed to retrieve results for was exceptionally low. To continue with, we observe a very similar behavior between Apache Jena Fuseki 3.6.0 and GraphDB Free 8.5. Both systems achieve a high performance in Micro and Macro-Average-Recall, showing that they are able to retrieve all expected results in most SELECT queries. In contrast, their

low values in Micro and Macro-Average-Precision indicate that both systems tend to include large sets of irrelevant answers to the query. Finally the two remaining systems, OpenLink Virtuoso and Virtuoso Commercial 8.0 share a similar behavior among the results: they both achieve high Mirco and Macro-Average-Recall and Precision, which shows their superior ability to ingest and retrieve triples with high accuracy.

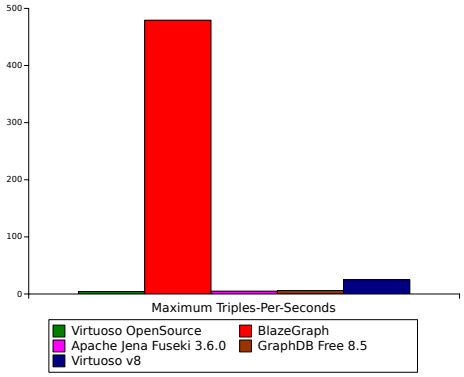
Regarding their maximum Triples-Per-Second KPI, Figure 1b shows that Blazegraph achieves the highest value among the other systems. This observation shows that Blazegraph is able to retrieve correct results for a large amount of triples that was inserted in a very short period of time. However, based on Figure 2, since both Micro and Macro-Average-Recall values are low, we can assume that this situation does not occur very often while using Blazegraph as a triple storage solution.

Regarding the efficiency KPIs, Figure 1a shows that both OpenLink Virtuoso and Virtuoso Commercial 8.0 require on average a minute amount of time to process the SELECT queries and retrieve correct results. For Apache Jena Fuseki 3.6.0 and GraphDB Free 8.5, the ability to retrieve high Recall performance comes at the cost of efficiency, since both systems have quite high average delay of tasks. Finally, Blazegraph has a low response time compared to the last two systems, but not insignificant as OpenLink Virtuoso and Virtuoso Commercial 8.0.

To conclude, both OpenLink Virtuoso and Virtuoso Commercial 8.0 had received the same Macro-Average-F-Measure value (approx. 0.86), so in order to announce the winner for Task 1 of MOCHA2018, we consider the second KPI in order: Macro-Average-Recall. OpenLink Virtuoso and Virtuoso Commercial 8.0 received 0.88 and 0.92 resp. This clearly indicates that Virtuoso Commercial 8.0 is the winner of Task 1 of MOCHA2018.




(a) Average Delay of tasks of all systems for Task 1




(b) Maximum Triples-per-Second of all systems for Task 1

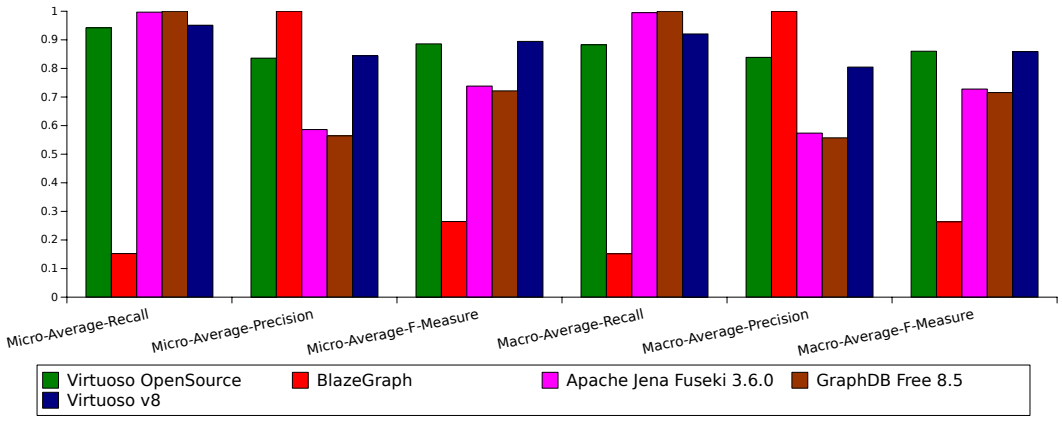



Fig. 2: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure of all systems for Task 1

### Task 2: Data Storage Benchmark

*KPIs* The key performance indicators for the Data Storage benchmark are rather simple and cover both efficiency and correctness:

- **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 in milliseconds.
- **Average Task Execution Time Per Query Type:** The average SPARQL query execution time per query type in milliseconds.
- **Query failures:** The number of SPARQL SELECT queries whose result set is different (in any aspect) from the result set obtained from the triple store used as a gold standard.
- **Throughput:** The average number of tasks (queries) executed per second.

*Experiment set-up* The benchmark had a defined maximum time for the experiment of 3 hours. The DSB parameters used in the challenge were the following:

- **Number of operations** = 15000
- **Scale factor** = 30
- **Seed** = 100
- **Time compression ratio (TCR)** = 0.5
- **Sequential Tasks** = false
- **Warm-up Percent** = 20

The scale factor parameter defines the size of dataset. Its value of 30 means 1.4 billion triple dataset. After bulk loading it, there were 15000 SPARQL queries

(INSERT and SELECT) executed against the system under test. One fifth of them was used for warm-up, while the rest of the queries were evaluated. The value 0.5 of TCR parameter implies about 17 queries per second.

*Results for Task 2* Unfortunately, out of the five systems which participated in the task, only two managed to complete the experiment in the requested time. Blazegraph, GraphDB and Jena Fuseki exhibited a timeout during the bulk loading phase, while the achieved KPIs for Virtuoso v8.0 and Virtuoso Open Source (VOS) are given below.

Based on the results from the main KPIs (Figure 3), the winning system for the task was Virtuoso 8.0 Commercial Edition by OpenLink Software. In the domain of efficiency, it is 32% faster than VOS regarding average query execution times, and also 6% faster in data loading. In the domain of correctness, Virtuoso v8.0 made 17 query failures compared to the 4 made by VOS; however, having in mind the fact that VOS was used as a golden standard for calculating the expected query results, this KPI is biased, and should not be considered as a weakness of the commercial edition of Virtuoso.

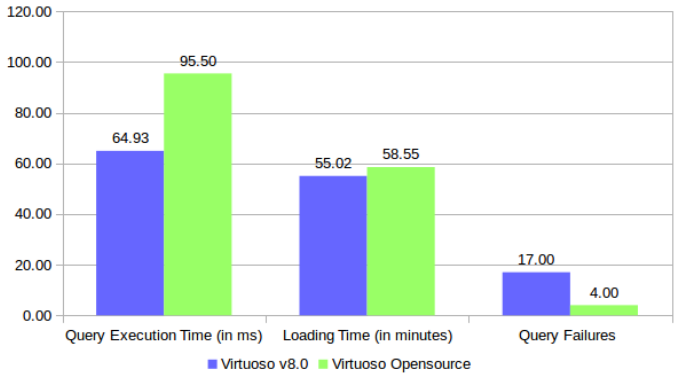



Fig. 3: Task 2: Main KPIs

The rest of the KPIs (Figure 4) show where the most important advantage comes from. For the complex query types, Virtuoso v8.0 is much faster than its open source counterpart, while the differences in the short look-ups and updates (SPARQL INSERT queries) are negligible.

### Task 3: Versioning RDF Data Benchmark

*KPIs* SPVB evaluates the correctness and performance of the system under test through the following *Key Performance Indicators (KPIs)*:

– **Query failures**: The number of queries that failed to execute. Failure refers to the fact that the system under test return a result set ($RS_{sys}$) that is not

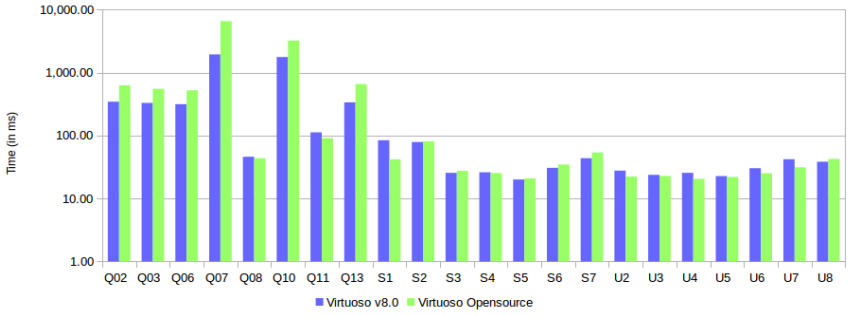



Fig. 4: Task 2: Average Query Execution Time per Query Type

equal to the expected one ($RS_{exp}$). This means that i) $RS_{sys}$ has equal size to $RS_{exp}$ and ii) every row in $RS_{sys}$ has one matching row in $RS_{exp}$, and vice versa (a row is only matched once). If the size of the result set is larger than 50.000 rows, for time saving, only condition i) is checked.

- **Initial version ingestion speed** (triples/second): the total triples of the initial version that can be loaded per second. We distinguish this from the ingestion speed of the other versions because the loading of the initial version greatly differs in relation to the loading of the following ones, where underlying processes such as, computing deltas, reconstructing versions, storing duplicate information between versions etc., may take place.
- **Applied changes speed** (changes/second): tries to quantify the overhead of such underlying processes that take place when a set of changes is applied to a previous version. To do so, this KPI measures the average number of changes that could be stored by the benchmarked systems per second after the loading of all new versions.
- **Storage space cost** (MB): This KPI measures the total storage space required to store all versions measured in MB.
- **Average Query Execution Time** (ms): The average execution time, in milliseconds for each one of the eight versioning query types.
- **Throughput** (queries/second): The execution rate per second for all queries.

*Experiment set-up* Since SPVB gives the ability to the systems under test to retrieve the data of each version as Independent Copies (IC), Change-Sets (CS) or both as IC and CS, three sub-tasks defined in the context of the challenge which only differed in terms of the "generated data form" configuration parameter. So, each participant was able to submit his/her system in the correct sub-task according to the implemented versioning strategy. In particular all the systems benchmarked using the following common parameters:

- **A seed for data generation**: 100
- **Initial version size** (in triples): 200000
- **Number of versions**: 5

– **Version Deletion Ratio** (%): 10
– **Version Insertion Ratio** (%): 15

The experiment timeout was set to 1 hour for all systems.

*Results for Task 3* All the systems except of the R43ples one, were managed to be tested. The latest, did not manage to load the data and answer all the queries in the defined timeout of 1 hour.

In order to be able to decide who is the winner for the Task 3 of the challenge, we combined the results of the four most important KPIs and calculated a *final score* that ranges from 0 to 1. Note here that due to reliability issues mentioned earlier, the "Storage space cost" KPI excluded from the final score formula. So, to compute the final score, we assigned weights to those KPIs, whose sum equals to 1. The four KPIs (in order of importance) along with the assigned weights are shown in Table 2.

| **Order** | **KPI** | **Weight** |
|---|---|---|
| 1 | Throughput | 0.4 |
| 2 | Queries failed | 0.3 |
| 3 | Initial version ingestion speed | 0.15 |
| 4 | Applied changes speed | 0.15 |

Table 2: Weights for the four most important KPIs

Next, we applied feature scaling[16] to normalize the results of the *Throughput*, *Initial version ingestion speed* and *Applied changes speed* by using the following formula:

$x' = \frac{x - min(x)}{max(x) - min(x)}$, where $x$ is the original value and $x'$ is the normalized value.

Regarding the *Queries failed* KPI, since the lower is the result, the better the system performs, the aforementioned formula applied on the percentage of succeeded queries and not to the number of queries that failed to be executed.

Having all the results normalized in the range of $[0 - 1]$ and the weights of each KPI, we computed the final scores as the sum of the weighted normalized results. As shown in Figure 5 Virtuoso v8.0 was the system that performed better.

[16] Bring all results into the range of [0,1]

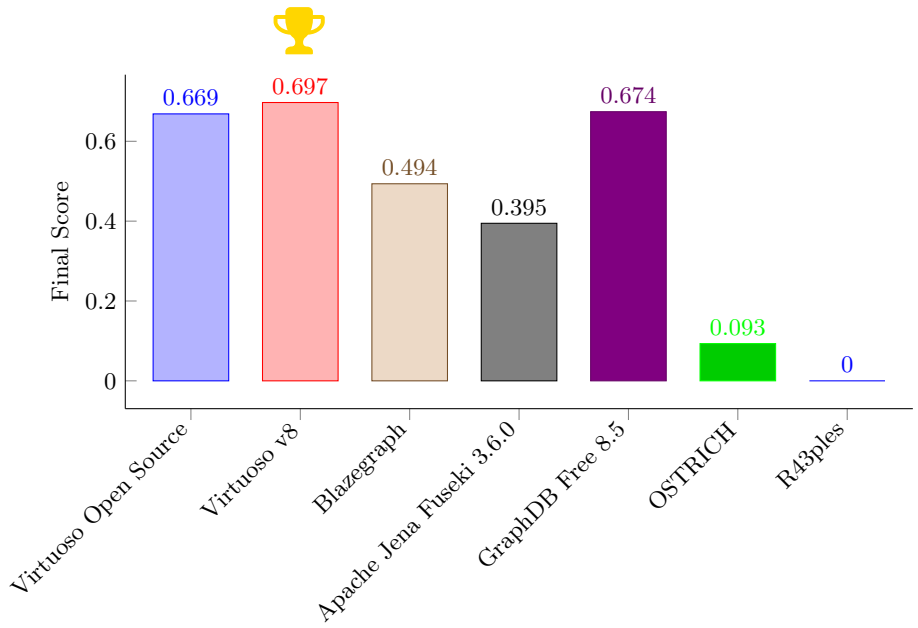



Fig. 5: Final scores for all systems

**Task 4: Faceted Browsing Benchmark**

*KPIs* The benchmark tracks the conventional indicators for correctness and efficiency, namely precision, recall, F-Measure and query-per-second rate. These measurements are recorded for the individual choke-points as well as the overall benchmark run. For ranking the systems, we were only interested in the latter:

- **Correctness** The conformance of SPARQL query result sets with pre-computed reference results.
- **Performance** The average number of faceted browsing queries per second.

*Experiment set-up* The dataset and SPARQL queries generated by the faceted browsing benchmark for the challenge are fully determined by the following settings:

- **Random Seed** = 111
- **Time range** = 0 - 977616000000
- **Number of stops / routes / connections** = 3000 / 3000 / 230000
- **Route length range** = 10 - 50
- **Region size / Cell Density** = 2000 x 2000 / 200
- **Delay Change / Route Choice Power** = 0.02 / 1.3

*Results for Task 4* Figure 6 shows the performance results of the faceted browsing benchmark. All participating systems were capable of successfully loading the data and executing the queries. Error values are obtained as follows: For COUNT-based queries, it is the difference in the counts of the actual and reference result. For all other (SELECT) queries, the size of the symmetric difference of the RDF terms mentioned in the reference and actual result sets is taken. While no differences were observer during testing the benchmark with Virtuoso Open Source and Apache Jena Fuseki 3.6.0, surprisingly, very minor ones were observed in the challenge runs. Yet, as all systems nonetheless achieved an f-measure score of >99% we did not discriminate by correctness and ranked them by performance only, depicted in Figure 6. The winner of the faceted browsing challenge is Virtuoso Open Source with an average rate of 2.3 query executions per second.

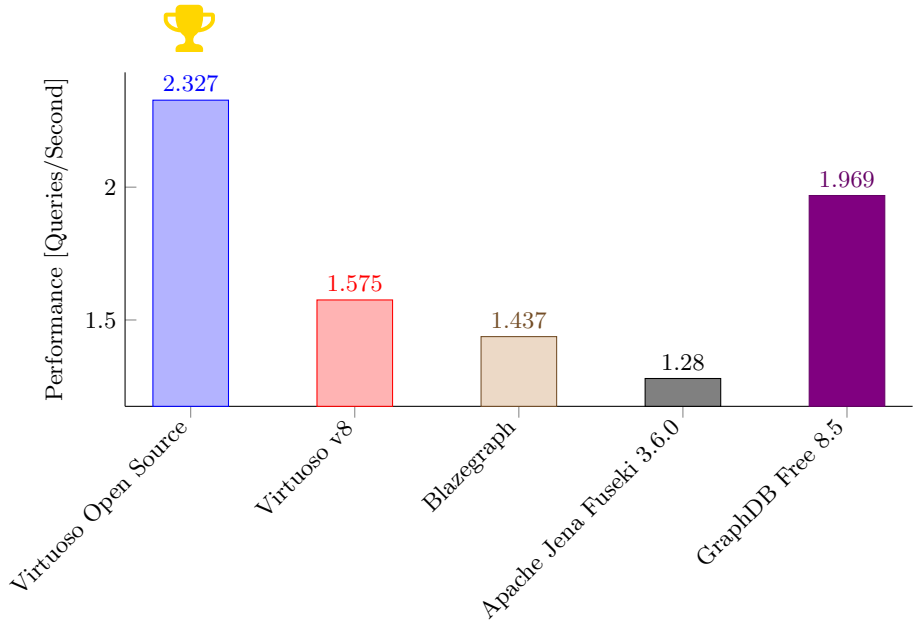



Fig. 6: Task 4 Faceted Browsing Benchmark Results

## 5 Conclusion

The goal of MOCHA2018 was to test the performance of storage solutions by measuring the systems performance in four different aspects. We benchmarked and evaluated six triple stores and presented a detailed overview and analysis of our experimental set-up, KPIs and results. Overall, our results suggest that the

clear winner of MOCHA2018 is Virtuoso v8.0. As part of our future work, we 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.

## Acknowledgements

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

## 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. Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The LDBC Social Network Benchmark: Interactive Workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 619–630. ACM, 2015.
3. Markus Graube, Stephan Hensel, and Leon Urbas. R43ples: Revisions for triples. *LDQ*, 2014.
4. Milos Jovanovik and Mirko Spasić. First Version of the Data Storage Benchmark, May 2017. Project HOBBIT Deliverable 5.1.1.
5. Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. DBpedia SPARQL Benchmark—Performance Assessment with Real Queries on Real Data. In *ISWC 2011*, 2011.
6. Henning Petzka, Claus Stadler, Georgios Katsimpras, Bastian Haarmann, and Jens Lehmann. Benchmarking faceted browsing capabilities of triplestores. In *13th International Conference on Semantic Systems (SEMANTiCS 2017), September 11-14 2017, Amsterdam, Netherlands*, 2017.
7. Mirko Spasić and Milos Jovanovik. Second Version of the Data Storage Benchmark, May 2018. Project HOBBIT Deliverable 5.1.2.