Benchmarking Virtuoso 8 at the Mighty Storage Challenge 2018: Challenge Results

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

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

Abstract. Following the success of Virtuoso at last year's Mighty Storage Challenge - MOCHA 2017, we decided to participate once again and test the latest Virtuoso version against the new tasks which comprise the MOCHA 2018 challenge. The aim of the challenge is to test the performance of solutions for SPARQL processing in aspects relevant for modern applications: ingesting data, answering queries on large datasets and serving as backend for applications driven by Linked Data. The challenge tests the systems against data derived from real applications and with realistic loads, with an emphasis on dealing with changing data in the form of streams or updates. 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 the final challenge results from MOCHA 2018 for Virtuoso v8.0, compared to the other participating systems. Based on these results, Virtuoso v8.0 was declared as the overall winner of MOCHA 2018.

Keywords: Virtuoso, Mighty Storage Challenge, MOCHA, Benchmarks, Data Storage, Linked Data, RDF, SPARQL

1 Introduction

Last year's Mighty Storage Challenge, MOCHA 2017, was quite successful for our team and Virtuoso – we won the overall challenge [6, 11]. Building on that, we decided to participate in this year's challenge as well, in all four challenge tasks: (i) RDF data ingestion, (ii) data storage, (iii) versioning and (iv) browsing. The Mighty Storage Challenge 2018[4] aims to provide objective measures for how well current systems perform on real tasks of industrial relevance, and also help detect bottlenecks of existing systems to further their development towards practical usage. This arises from the need for devising systems that achieve acceptable performance on real datasets and real loads, as a subject of central importance for the practical applicability of Semantic Web technologies.

[4] `https://project-hobbit.eu/challenges/mighty-storage-challenge2018/`

2 Virtuoso Universal Server

Virtuoso Universal Server[5] 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 [2, 3]. Recently, the product has been revised to take advantage of column-wise compressed storage and vectored execution [1].

The largest Virtuoso applications are in the RDF and Linked Data domains, where terabytes of RDF triples are in use – a size which does not fit into main memory. The space efficiency of column-wise compression was the biggest incentive for the column store transition of Virtuoso [1]. This transition also made Virtuoso a competitive option for relational analytics. Combining a schemaless data model with analytics performance is an attractive feature for data integration in scenarios with high schema volatility. Virtuoso has a shared cluster capability for scaling-out, an approach mostly used for large RDF deployments.

A more detailed description of Virtuoso's triple storage, the compression implementation and the translation of SPARQL queries into SQL queries, is available in our paper from MOCHA 2017 [11].

3 Evaluation

Here we present the official challenge results for Virtuoso v8.0 for all MOCHA 2018 tasks, based on the challenge data and benchmark parameters specified by the organizers. Additionally, we give a brief comparison of Virtuoso v8.0 with the other participating systems.

The results presented in this section are publicly available at the official HOBBIT platform[6], where the challenge and all its tasks took place. The platform allows execution of different benchmarks in order to evaluate the performance of different systems. A specific benchmark was implemented and used for each task of the MOCHA challenge. These benchmarks share a common API which eases the work of the challenge participants. For the benchmarking of the participant systems, a server cluster was used. Each of the systems could use up to three

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

[6] `https://master.project-hobbit.eu/`

servers of the cluster, each of them having 256 GB RAM and 32 CPU cores. This enabled benchmarking of both monolithic and distributed solutions.

The Virtuoso v8.0 configuration parameters which we used for the challenge are available at GitHub[7].

Compared to MOCHA 2017, the tasks of MOCHA 2018 were significantly more demanding and the datasets were larger, which lead to a tougher playground. But, this tougher playground is also a better representation of the real-world applications over large amounts of RDF and Linked Data, which the benchmarks and the challenge aim to test.

3.1 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 a system can 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, SPARQL SELECT queries check the triples that are actually inserted and test the system's ingestion performance and storage abilities [4, 5].

Table 1. ODIN Configuration.

Parameter	**Value**
Duration of the benchmark	600000
Name of mimicking algorithm	TRANSPORT_DATA
Name of mimicking algorithm output folder	output_data/
Number of data generators - agents	4
Number of insert queries per stream	20
Number of task generators - agents	1
Population of generated data	10000
Seed for mimicking algorithm	100

Results: Our system, Virtuoso v8.0 Commercial Edition, was tested against ODIN as part of the MOCHA challenge. The task organizers specified the benchmark parameters for the challenge and their values are shown in Table 1, while the achieved KPIs for our system are presented in the Table 2.

The task has three KPIs:

[7] `https://github.com/hobbit-project/DataStorageBenchmark/blob/master/system/virtuoso.ini.template`

- **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 that the SELECT query has been generated and sent to the System Adapter and the time that the results are received by the Evaluation Storage.
- **Correctness**: A recall, precision and F-measure of each SELECT query by comparing the retrieved results and the expected ones obtained from an instance of the Jena TDB storage solution.

Table 2. ODIN KPIs for Virtuoso v8.0.

KPI	**Value**
Average Delay of Tasks (in seconds)	1.3398
Average Triples-Per-Second	11.1909
Macro-Average-F-Measure	0.8587
Macro-Average-Precision	0.8047
Macro-Average-Recall	0.9206
Maximum Triples-Per-Second	25.0410
Micro-Average-F-Measure	0.8945
Micro-Average-Precision	0.8442
Micro-Average-Recall	0.9511

We can divide the KPIs into two categories: Efficiency (Average Delay of Tasks, Average Triples-Per-Second and Maximum Triples-Per-Second) and Correctness (Macro/Micro Average F-Measure/Precision/Recall). In terms of efficiency, the Average Triples-Per-Second KPI is not relevant, as it mainly depends from the benchmark parameters, and the values achieved by all tested systems are very similar. The Average Delay of Tasks KPI shows the efficiency of the system executing SPARQL SELECT queries, i.e. the average execution time of SELECT queries, while the Maximum Triples-Per-Second KPI indicates how fast can tripes be received by the system without any loss.

Our system takes about a second to process a SELECT query on average. Virtuoso Opensource (VOS) is very similar, while all other participating systems show worse results by two orders of magnitude (83-496s). Our achieved value for Maximum Triples-Per-Second is 25, while most of the systems are around 5. Blazegraph achieves the highest value here, but its average recall and f-measure values are exceptionally low. Figures 1 and 2 showcase these results and identify the overall winner of the task. Our system and VOS achieve very similar results in terms of correctness, while Virtuoso v8.0 is undoubtedly better in terms of

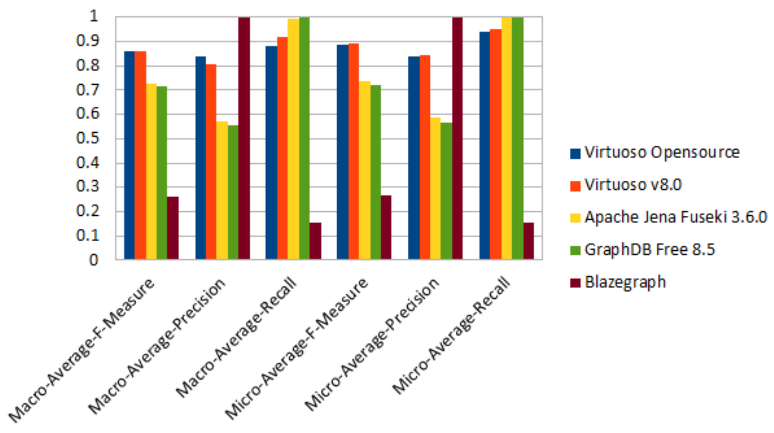

Fig. 1. Task 1 - Correctness KPIs

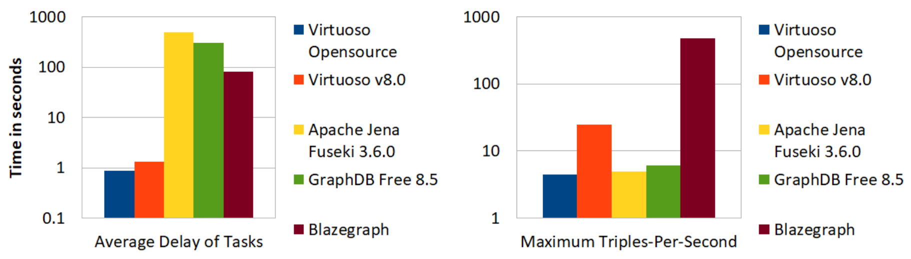

Fig. 2. Task 1 - Efficiency KPIs

efficiency. The organizers confirmed this by declaring our system as the winner in this task.

3.2 Task 2 - Data Storage

This task uses the Data Storage Benchmark (DSB) and its goal is to measure how data storage solutions perform with interactive read SPARQL queries, accompanied with a high insert data rate via SPARQL UPDATE queries. This approach mimics realistic application scenarios where read and write operations are bundled together. It also tests systems for their bulk load capabilities [7, 12].

Results: The benchmark parameters for the task are shown in Table 3, and the achieved KPIs for our system are presented in Table 4.

The main KPIs of the task are:

– **Bulk Loading Time**: The total time in milliseconds needed for the initial bulk loading of the dataset.

Table 3. DSB Configuration.

Parameter	**Value**
Scale Factor	30
Time Compression Ratio	0.5
Warm-up Percent	20
Enable/Disable Query Type	01100111011010
Number of Operations	15000
Enable Sequential Tasks	False
Seed	100

- **Average Query Execution Times per Query Type**: The execution time is measured for every single query, and for each query type the average query execution time is calculated.
- **Query Failures**: The number of returned results that are not as expected, obtained from the triple store used as a gold standard.

Table 4. DSB KPIs for Virtuoso v8.0.

KPI	**Value**	**KPI**	**Value**
Average Query Execution Time	64.9332	Average S5 Execution Time	20.0040
Average Q02 Execution Time	344.6176	Average S6 Execution Time	30.6064
Average Q03 Execution Time	329.1667	Average S7 Execution Time	43.5408
Average Q06 Execution Time	314.3750	Average Update2 Execution Time	27.6351
Average Q07 Execution Time	1943.5962	Average Update3 Execution Time	23.7248
Average Q08 Execution Time	45.9784	Average Update4 Execution Time	25.5625
Average Q10 Execution Time	1761.2794	Average Update5 Execution Time	22.6334
Average Q11 Execution Time	112.3680	Average Update6 Execution Time	30.2792
Average Q13 Execution Time	336.1515	Average Update7 Execution Time	41.9760
Average S1 Execution Time	84.2704	Average Update8 Execution Time	38.2564
Average S2 Execution Time	78.9440	Loading Time (in ms)	3301449
Average S3 Execution Time	25.5088	Query Failures	17
Average S4 Execution Time	26.0392	Throughput (queries/s)	17.6797

In this task, the organizers wanted to stress the scalability of the system by specifying a large dataset with over 1.4 billion triples. Virtuoso justified its

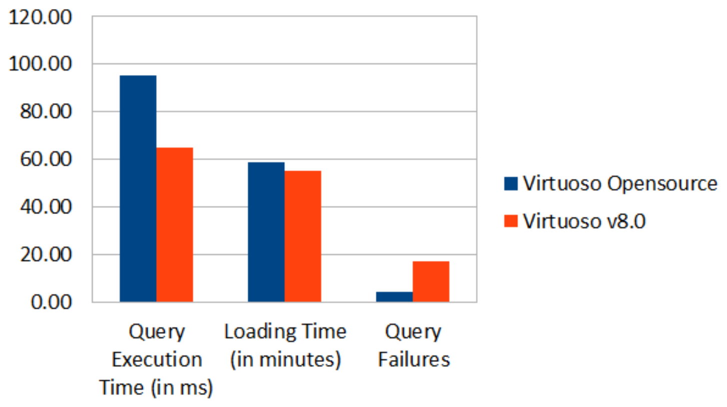

Fig. 3. Task 2 - Main KPIs

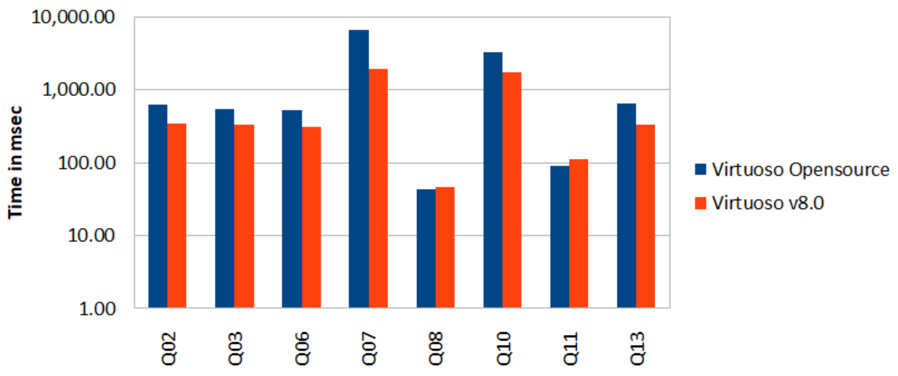

Fig. 4. Task 2 - Average Complex Query Execution Time per Type

dominance with a huge victory in this task, showing why it was well known to be a scalable system. Unfortunately, Blazegraph, GraphDB, and Jena were not able to even load the dataset in the maximum experiment time of 3 hours, thus the only comparable system here was VOS. The comparison of these two systems is given in the Figures 3, 4 and 5. In the domain of efficiency, Virtuoso v8.0 is 32% faster than VOS regarding average query execution times, and around 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 latest version of Virtuoso.

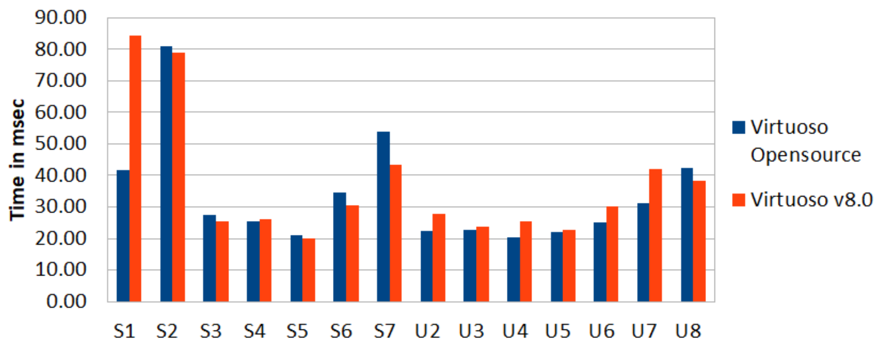

Fig. 5. Task 2 - Average Short and Update Query Execution Time per Type

3.3 Task 3 - Versioning RDF Data

The aim of this task is to test the ability of versioning systems to efficiently manage evolving datasets, where triples are added or deleted, and queries evaluated across the multiple versions of said datasets. It uses the Versioning Benchmark (VB) [9, 8].

Table 5. Versioning Benchmark Configuration.

Parameter	**Value**
Generated Data Form	IC/CS/IC+CS
Initial Version Size (in triples)	200000
Number of Versions	5
Version Deletion Ratio (%)	10
Version Insertion Ratio (%)	15
A seed for data generation (%)	100

Results: Table 5 shows the benchmark configuration and Table 6 shows the results achieved by Virtuoso v8.0 for the versioning task. The evaluation is based on the following performance KPIs:

– **Query Failures**: The number of queries whose returned results are not those that were expected.
– **Throughput** (in queries per second): The execution rate per second for all queries.
– **Initial Version Ingestion Speed** (in triples per second): The total triples that can be loaded per second for the dataset's initial version.

- **Applied Changes Speed** (in triples per second): The average number of changes that can be stored by the benchmarked system per second after the loading of all new versions.
- **Average Query Execution Time** (in ms): The average execution time, in milliseconds, for each one of the eight versioning query types.

Table 6. Versioning Benchmark KPIs for Virtuoso v8.0.

KPI	Value	KPI	Value
Applied Changes (changes/s)	19831.2070	QT5, Avg. Exec. Time (ms)	7186.0000
Initial Ingestion (triples/s)	43793.0820	QT6, Avg. Exec. Time (ms)	165.7500
QT1, Avg. Exec. Time (ms)	16861.0000	QT7, Avg. Exec. Time (ms)	17.0000
QT2, Avg. Exec. Time (ms)	147.6667	QT8, Avg. Exec. Time (ms)	155.6389
QT3, Avg. Exec. Time (ms)	11944.0000	Queries Failed	2
QT4, Avg. Exec. Time (ms)	142.3889	Throughput (queries/s)	1.2068

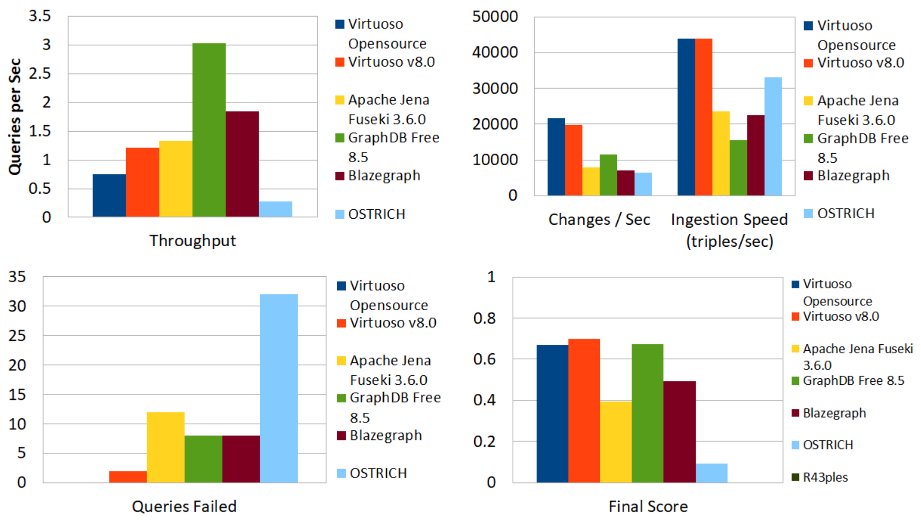

Fig. 6. Task 3 - KPIs and Final Score

Apart from the same five systems which participated in the previous tasks, two additional ones (specialized in storing different versions of datasets) took part at Task 3. These two system, as well as some other versioning systems are not mature enough to handle very large datasets, so the organizers decided to decrease the dataset size and give them the opportunity to compete. Relatively small datasets result in comparable throughput achieved by our and the other systems (Figure 6). Judging by the previous tasks, we would expect for our system to have better throughput compared the rest of the participants if the datasets were larger. Still, Virtuoso v8.0 acted better in the initial loading, applied changes per second and correctness (Figure 6). Therefore, the announcement of the winner here was not straightforward, and the organizers combined the results of the four most important KPIs with their assigned weights (Throughput - 0.4, Queries Failed - 0.3, Initial Version Ingestion Speed - 0.15 and Applied Changes Speed - 0.15). With this, they calculated the final scores which range from 0 to 1. The final scores for all participant systems are shown on Figure 6, with Virtuoso v8.0 providing the best performance in the task.

3.4 Task 4 - Faceted Browsing

This task uses the Faceted Browsing Benchmark, which tests 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 [10].

Table 7. Faceted Browsing Configuration.

Parameter	Value
Cells per LatLon	200
Delay Chance	0.2
Final Trip Time	977616000000
Initial Trip Time	0
Maximum Route Length	50
Minimum Route Length	10
Number of Connections	230000
Number of Routes	3000

Parameter	Value
Number of Stops	3000
Preconfiguration	mocha2018
Quick Test Run	false
Random Seed	111
Region Size X	2000
Region Size Y	2000
Route Choice Power	1.3

Results: The benchmark parameters for the task are given in Table 7, while Table 8 shows the Virtuoso v8.0 results. The evaluation is based on the following performance KPIs:

- **Correctness**: The conformance of SPARQL query result sets with precomputed reference results in terms of precision, recall and F-Measure.

– **Performance**: Query-per-second rate.

Table 8. Faceted Browsing Main KPIs for Virtuoso v8.0.

KPI	**Value**
Total F-measure	1.0
Total precision	1.0
Total recall	1.0
Throughput (queries/s)	1.5755

The comparison of our system with the other participant systems for the task are shown on the Figure 7. As we can see, Virtuoso v8.0 did not achieve the best performance score for the task, and the task winner was VOS. One possible reason why our system showed lower performance compared to VOS and GraphDB should be sought in the small size of the dataset and the fact that our system was pre-configured for significantly larger deployments. Even though the training phase of this task used a smaller dataset, we still expected a larger dataset for the challenge run, thus configured Virtuoso v8.0 accordingly.

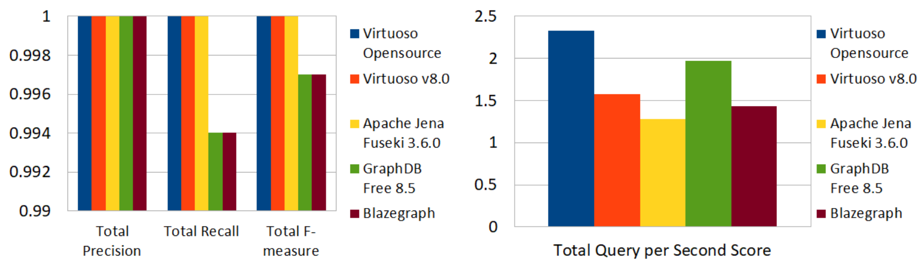

Fig. 7. Task 4 - Correctness (left) and Performance (right)

4 Conclusion and Future Work

In this paper, we provide an overview of the setup, participation and results of Virtuoso v8.0 at the Mighty Storage Challenge - MOCHA 2018, at the Extended Semantic Web Conference - ESWC 2018. Our system participated in all four tasks of the challenge: (i) RDF data ingestion, (ii) data storage, (iii) versioning and (iv) faceted browsing, winning the first three. Therefore, the challenge organizers declared Virtuoso v8.0 as a clear winner of MOCHA 2018.

As future work, a further Virtuoso evaluation has been planned, using other dataset sizes and especially larger datasets, stressing its scalability. We can already foresee improvements of the query optimizer, driven by the current evaluation. The comparison of our performance with the other participant systems also provides significant guidelines for the future development of Virtuoso.

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

References

1. Orri Erling. Virtuoso, a Hybrid RDBMS/Graph Column Store. *IEEE Data Eng. Bull.*, 35(1):3–8, 2012.
2. Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge-Networked Media*, pages 7–24. Springer, 2009.
3. Orri Erling and Ivan Mikhailov. Virtuoso: RDF support in a native RDBMS. In *Semantic Web Information Management*, pages 501–519. Springer, 2010.
4. Kleanthi Georgala. First Version of the Data Extraction Benchmark for Sensor Data, May 2017. Project HOBBIT Deliverable 3.1.1.
5. Kleanthi Georgala. Second Version of the Data Extraction Benchmark for Sensor Data, May 2018. Project HOBBIT Deliverable 3.1.2.
6. Kleanthi Georgala, Mirko Spasić, Milos Jovanovik, Henning Petzka, Michael Röder, and Axel-Cyrille Ngonga Ngomo. MOCHA2017: The Mighty Storage Challenge at ESWC 2017. In *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017*, pages 3–15. Springer, 2017.
7. Milos Jovanovik and Mirko Spasić. First Version of the Data Storage Benchmark, May 2017. Project HOBBIT Deliverable 5.1.1.
8. Vassilis Papakonstantinou, Irini Fundulaki, and Giorgos Flouris. Second Version of the Versioning Benchmark, May 2018. Project HOBBIT Deliverable 5.2.2.
9. Vassilis Papakonstantinou, Irini Fundulaki, Giannis Roussakis, Giorgos Flouris, and Kostas Stefanidis. First Version of the Versioning Benchmark, May 2017. Project HOBBIT Deliverable 5.2.1.
10. Henning Petzka. First Version of the Faceted Browsing Benchmark, May 2017. Project HOBBIT Deliverable 6.2.1.
11. Mirko Spasić and Milos Jovanovik. MOCHA 2017 as a Challenge for Virtuoso. In *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017*, pages 21–32. Springer, 2017.
12. Mirko Spasić and Milos Jovanovik. Second Version of the Data Storage Benchmark, May 2018. Project HOBBIT Deliverable 5.1.2.