Yet Another Triple Store Benchmark? Practical Experiences with Real-World Data

Martin Voigt, Annett Mitschick, and Jonas Schulz

Dresden University of Technology, Institute for Software and Multimedia Technology,
01062 Dresden, Germany
{martin.voigt, annett.mitschick, jonas.schulz}@tu-dresden.de

Abstract. Although quite a number of RDF triple store benchmarks have already been conducted and published, it appears to be not that easy to find the right storage solution for your particular Semantic Web project. A basic reason is the lack of comprehensive performance tests with real-world data. Confronted with this problem, we setup and ran our own tests with a selection of four up-to-date triple store implementations – and came to interesting findings. In this paper, we briefly present the benchmark setup including the store configuration, the datasets, and the test queries. Based on a set of metrics, our results demonstrate the importance of real-world datasets in identifying anomalies or differences in reasoning. Finally, we must state that it is indeed difficult to give a general recommendation as no store wins in every field.

Keywords: RDF triple stores, benchmark, real-world datasets, reasoning, multi-user

1 Introduction

The last months inevitably reveal the advance of Semantic Web technologies in organizing and finding information, e. g., through the advent of *schema.org* or *Google Knowledge Graph* [1]. This is especially fostered by the widespread W3C standards like RDF(S), OWL, and SPARQL to allow for publishing and consuming Linked (Open) Data. As their benefits become more and more clear, also vendors in the publishing sector, e. g., *moresophy* [2], are applying these technologies to facilitate the semantic tagging and searching of media assets within their archives. An important and critical issue when developing large-scale Semantic Web applications is the right choice of an appropriate storage solution for RDF-based data. Comprehensive benchmarking results may help to estimate the applicability of state-of-the-art triple stores for ones own project.

Although, a number of performance reports already exist, we soon discovered that available results are of limited significance for our particular purposes. The goal of our research project *Topic/S* [3] is to provide a topic-based ranking and search of texts, images, and videos delivered by press media agencies. Therefore, we rely on the information automatically extracted from the media assets using NLP algorithms, but also consume other datasets to broaden the knowledge

graph of our archive, e. g., with information about people or organizations from New York Times [4] or YAGO2 [5], to improve the search. Thus, we heavily depend on a high-performance RDF storage solution – on the one hand for the extracted semantic data, and on the other hand for simulating public SPARQL endpoints for the required third party datasets (for the reason of continuous availability and serviceability).

A review of the existing work in the area of RDF store benchmarking [6] exposes that the results are interesting but not quite helpful for our use case due to varied reasons. A prominent reason is the lack of comprehensive tests on real-world datasets (non-synthetic). According to [7] automatically generated datasets used for benchmarking differ from real datasets, like *DBpedia* or *WordNet*, regarding "structuredness", and inevitably lead to different benchmarking results. BSBM [8, 9] is the most advanced benchmark available with regard to data size, parameters, or number of RDF stores. Unfortunately, the results are building on a generated dataset apart from our media archive domain. Further, the last test does not address SPARQL 1.1 [10]. Another project to be mentioned is *SP²Bench* [11] which dealt with the use of a broad range of possible SPARQL constructs and their combination – without reasoning. As the benchmark was carried out in 2008 on a generated dataset the results were also not beneficial for us. In contrast to that, the *FedBench* suite [12] focused on query federation on real-world datasets. However, the benchmark does not address RDFS reasoning nor SPARQL 1.1. Further, current RDFS stores like Virtuoso are not tested. A very comprehensive and most up-to-date survey on existing RDF stores is given in [13]. The results of this survey, carried out in the context the *Europeana* project [13], were build upon previous studies which the authors extended by their own benchmark using the (real-world) Europeana dataset (bibliographic metadata, approx. 380 million triples). Even though the results are the most up-to-date available (March 2011), the performance of the stores are maybe improved meanwhile. Moreover, the tests also did not consider RDFS reasoning, SPARQL 1.1, and heavy load (multiple queries in parallel).

In this paper we present the latest results of *"yet another"* benchmark of current RDF stores – but with the following *unique features*: loading and querying real-world datasets, testing RDFS reasoning and SPARQL 1.1 queries, conducting multiple queries in parallel, and recording the memory requirements.

Thus, the paper is organized as follows: In the next section, we briefly summarize the benchmark setting, including the selected RDF stores, the datasets and queries. In Section 3, we shortly introduce the metrics, present and discuss the results of our benchmark. A conclusion and outlook on future work is given in Section 4.

2 Benchmark Setup

In this section, we briefly introduce which RDF stores we have selected and how we set them up in our benchmark. In the second part, we present the four real-world datasets and the queries we utilized for our evaluation.

2.1 RDF Triple Stores

Although, there are more RDF stores available, we focused on Apache Jena [14], BigData RDF Database [15], OWLIM Lite [16], and OpenLink Virtuoso [17] within our benchmark due to the restrictions of our project setup: freely available, allows to handle up to 100 million triples, supports RDFS reasoning as well as SPARQL 1.1, and is build for the Java runtime environment.

The Apache Jena projects comes up with a bunch of sub-systems. For our purpose we needed a fast triple store as well as a SPARQL endpoint, thus, we relied on the Fuseki server with the version 0.2.3 which includes TDB 0.9.0 – a high performance RDF store. We used Fuseki with the default configuration. The RDFS reasoning is applied using an assembler description file.

BigData®is a high-performance RDF database which includes the NanoSparqlServer as SPARQL endpoint. We used the version 1.2.0 which comprises SPARQL UPDATE functionality. We employed the default setting except for setting up the RDFS reasoning within the RWStore.properties file.

Ontotext distributes three OWLIM editions. We deployed OWLIM-Lite 5.0.5 which builds on top of Sesame 2.6.5 and is freely available. Further, it is designed for datasets up to 100 million triples which fits our requirements.

Virtuoso provides an open source edition of their RDF store which includes a SPARQL endpoint. We installed version 6.1.5 und used the default setting. Inference is done only on runtime while querying.

Our benchmarks were conducted within a Ubuntu Linux 12.04 64bit virtual machine on a Intel Xeon CPU X5660 2.80GHz with 4 cores, 16GB RAM and 120GB virtual hard drive. The stores ran within Java 1.7.0 64bit runtime environment. If an application server was required, e. g., for OWLIM-Lite, we used Apache Tomcat 7.0.28.

2.2 Datasets and Queries

Especially as we want to reuse existing semantic datasets within the Topic/S project [3], e. g., to link named entities to DBpedia or YAGO2 [5], we chose to test the stores with real-world data. Furthermore, we could check if some of the stores had problems in loading or querying the data. To allow for a better comparability we transformed all sets to the N-Triple format what means that every row contains a single RDF triple. Therefore, for all but YAGO2 Core we used the current version of TopBraid composer. For YAGO2 we used the RDF2RDF tool [18]. The New York Times dataset [4] is originally distributed over four files which were merged. Table 1 gives an overview of the used sets which illustrates their difference of the general size but also of the schema and the number of instances.

We defined 15 queries per dataset using the SPARQL 1.1 query language. To compare the performance between stores and datasets at once, we created six general queries, e. g., counting the number of distinct subjects or the occurrence of the properties. The further nine queries are dataset-dependent and designed for real-world scenarios in Topics, for instance to retrieve the names of persons

	NY Times	Jamendo	Movie DB	YAGO2 Core
Size (in MByte)	56,2	151,0	891,6	5427,2
Triples (in Mio)	0,35	1,05	6,15	35,43
Instances (in k)	13,2	290,4	665,4	2648,4
Classes	19	21	53	292861
Properties	69	47	222	93

Table 1. Overview of the benchmark datasets

living in a dedicated time. Four of them are SELECTs with rising complexity, e. g., by using UNION, regex filters, or subqueries. Query 11 is used to investigate the DESCRIBE performance. In query 12 and 13 we especially considered the performance of RDFS inference. The last two UPDATE queries delete and insert triples. The complete list of queries can be found at [19].

3 Benchmark Results

In the following, we give a brief introduction to the metrics and the execution plan of our benchmark tests. The actual results of the tests are presented in Section 3.2.

3.1 Benchmark Metrics and Execution

In our benchmark we propose several metrics to capture different evaluation aspects for the stores. They are tested with all four datasets which are different in size and structure (c.f. Sect. 2.2). Furthermore, all the metrics are evaluated with and without RDFS reasoning (except the multi-client performance tests which were conducted exclusively with reasoning).

1. *Loading time*: At first, we measured the loading time of each dataset with the stores three times and calculated the average.
2. *Memory requirement*: Further, we measured the memory consumption of each store after the datasets were loaded.
3. *Per-query type performance*: Our main focus was to compare the query performance of the stores. We mainly distinguish between the generic, the dataset-specific, and the UPDATE queries. For our report we calculate the average but also extract the min and max values.
4. *Success rate*: We also investigate the success rate of each query. We define a query to be successful if it delivers the expected results without any error or timeout.
5. *Multi-client performance*: For the multi-client scenario we measured the average query performance as well as how many queries could be executed within a 10 minutes time slot.

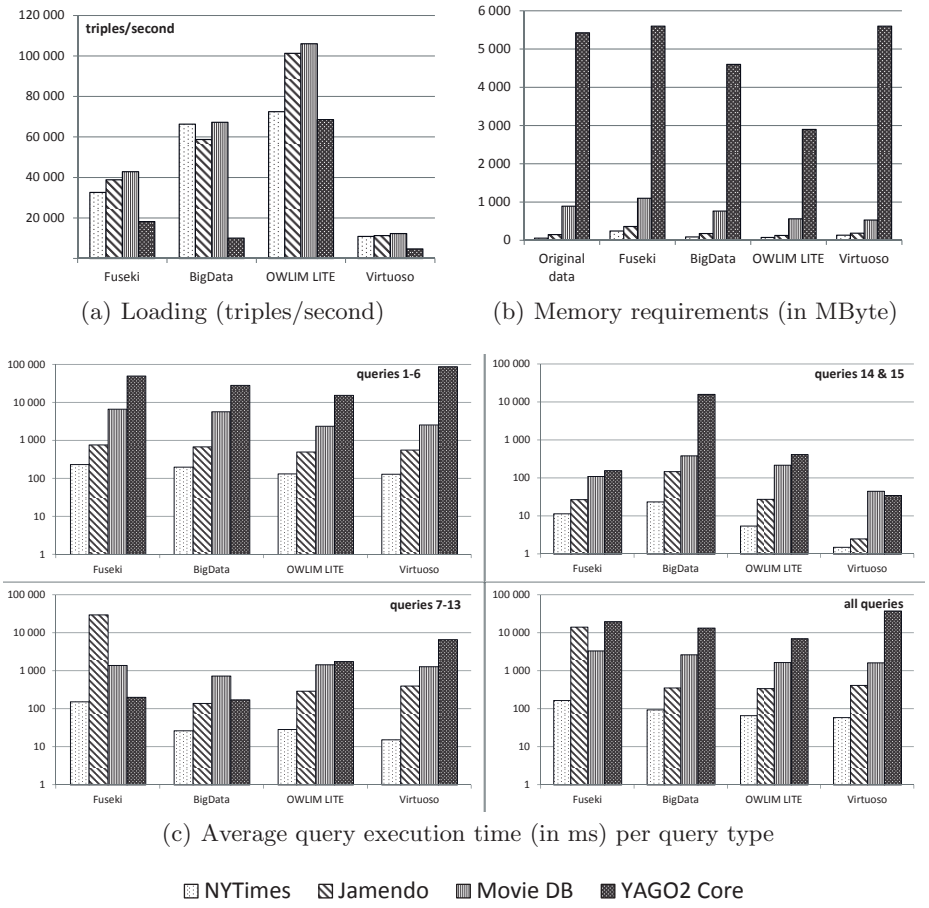
For the execution of the query benchmark we loaded the same dataset into all installed and pre-configured stores. Further, we wrote a simple Java client (test driver), which is available at our website [19], to rise automation and comparability. It requests a store with all 15 queries 20 times in a round robin manner. Having all values, we compute the average for each query. After all four sets were evaluated, we enabled RDFS reasoning for the stores, loaded the data, and did the same request using the test driver. Besides theses metrics, we evaluate how the stores scale in a multi-client scenario. Therefore, we loaded the NY Times dataset in every store with enabled RDFS reasoning. We selected four queries – two generic and to dataset-specific – which had approximately the same average execution time and called them using our test driver in a randomized round robin mechanism.

3.2 Experimental Results and Discussion

From our benchmark we gain several insights we will discuss in the following paragraphs. The selected Figures 1, 2 and 3 give a short summary of our results, whereas our website [19] provides more detailed information. Please mind, that Figures 1(c), 2(c) and 3 use logarithmic scaling.

Fig. 1 showcase the benchmark results with RFDS inference turned *off*. The first interesting finding is that OWLIM Lite performs best and Virtuoso worst to load all four datasets. BigData was fast but we identified a strange behaviour with the regex filters. The worked well for all datasets exempt from Jamendo. Here, all queries with a regex deliver no result. Further, query 3 and 13 on YAGO2 Core caused timeouts but BigDatas log didn't provide any insight to solve the problem. Next, Virtuoso is the store of your choice if you had many UPDATE transactions (query 14 and 15) to handle. If we compare all INSERT and DELETE queries of all datasets, it is approximately 4 times faster then second-placed Fuseki. With regard to all queries made, the performance of OWLIM Lite and Virtuoso is nearly the same. Only with the 35 million dataset Virtuoso fell back. Finally, we identified that Fuseki scales really bad with subquery requests on NY Times and Jamendo. Unfortunately, we could not identify a particular reason as the query complexity is quite the same as for the other two datasets.

Our findings regarding the abilities of the stores to support RDFS reasoning are displayed in Fig. 2. Here, Fig. 2(a) confirms that OWLIM Lite is approximately 80% faster than Virtuoso regarding the load performance. As Fuseki is almost as fast like without reasoning, we run into performance issues with BigData. For our technical setup, it was not possible to load YAGO2 Core into the store because the created temporary file exceeded the disk space. But all in all, the memory consumption of the store changes only slightly (Fig. 2(b)). The average execution times of the queries (c.f. Fig. 2(c)) show that Virtuoso is still the fastest on UPDATEs by far – approximately 8 times faster than the second-placed Fuseki. The performance of OWLIM and BigData decreases dramatically with this query type. Similar to the subquery problem of Fuseki we found that Virtuoso performs really bad with two generic queries (query 4 and 5) on the

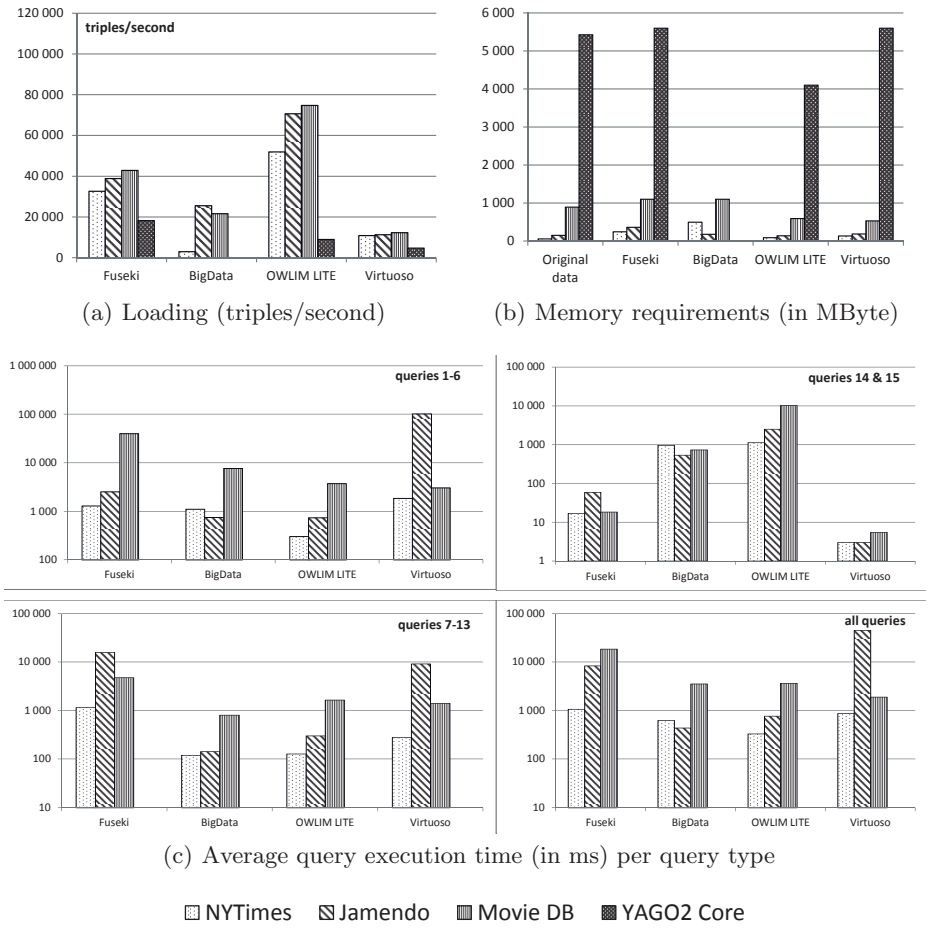


(a) Loading (triples/second)

(b) Memory requirements (in MByte)

(c) Average query execution time (in ms) per query type

Fig. 1. Overview of the results **without** RDFS reasoning: (a) describes the loading time of the stores and (b) shows the final memory consumption; (c) allows for comparing the performance per-query type



(a) Loading (triples/second)

(b) Memory requirements (in MByte)

(c) Average query execution time (in ms) per query type

Fig. 2. Overview of the results **with** RDFS reasoning: (a) describes the loading time of the stores and (b) shows the final memory consumption; (c) allows for comparing the performance per-query type. Here, YAGO2 is left out because of some shortcomings.

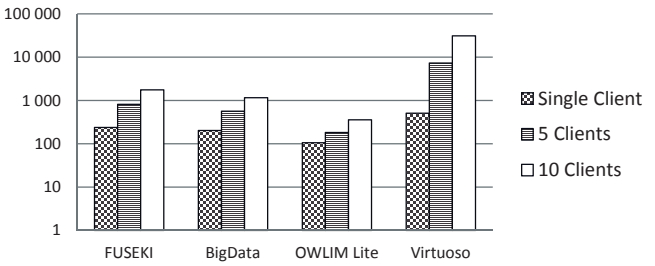


Fig. 3. Comparison on average query execution time (in ms) in our multi-client scenario using the NY Times dataset

Jamendo dataset. As the reason is not obvious it strengthens our finding that benchmarks on real-world datasets is important. In the end, we must state that the query performance decreases in general.

The bar chart in Fig. 3 illustrates the average query execution time in our multi-client setup. OWLIM Lite scales best with factor 2 from single to five as well as 5 to 10 clients. Fuseki and BigData are running shoulder on shoulder as both scale quite linear to the number of clients. For Virtuoso, the performance of query 2 does not depend on the number of clients. But it unfolds issues for some queries, e. g., query 9 is around 533 time slower with 10 clients compared to the single client scenario.

In the end, we want to review the error rate in a qualitative way. First, we need to state that we faced issues regarding YAGO2 Core with RDFS inference turned *on* so that we did not measure any query performance. For instance, BigData produces a temporary file which exceeded the disk space of our virtual machine or Fuseki timed out for some of the queries. Second, our generic queries comprise the SPARQL *COUNT* statement so that we could easily compare the results. Within the RDFS inference scenario the benchmark was very surprising as for the queries 1, 2, 3, and 6 *every* triple store returned a different result. Further, OWLIM Lite was the only store which counts more triples for query 4 and 5. Thus, our urgent advice for your own project is to cross-check the results at random if you need to use RDFS reasoning. Third, another anomaly we faced was that BigData had problems with *regex* filters but only on Jamendo dataset. This underlines again the need to benchmark an RDF store with different real-world datasets.

4 Conclusion

In this paper we present *yet another triple store benchmark* as we did not find anyone with evaluation criteria like they are required for our research project: loading and querying real-world datasets, testing RDFS reasoning and SPARQL 1.1 queries, as well as conducting multiple queries in parallel. In the following, we discuss our four findings.

As first result on our work with four up-to-date store we need to state, that comprehensive tests with real-world data are necessary. Otherwise it is not possible to detect anomalies like we identified. Second, every tested store allows for RDFS inference. But be careful as the result set may differ from store to store. Third, SPARQL 1.1 is well implemented nowadays. But the performance on UPDATE queries is varying. Here, Virtuoso stands out. Finally, we could not recommend any triple store in general as no store could win on all fields. Thus, the selection strongly depends on your specific project requirements. For our work, we will rely on OWLIM Lite because we need one which is fast in reading the datasets and multi-client query processing.

As we had problems with YAGO2 Core and inference, especially with regard to our technical setup, we are evaluating the requirements for bigger datasets. Besides the core version we like to benchmark the stores with YAGO2 Full as well. Another future work is to evaluate the performance of OWL reasoning for some common constructs, e. g., cardinalities. Therefore, we need to identify suitable real-world datasets.

5 Acknowledgments

Work on this paper is partly funded within the Topic/S project by the European Social Fund / Free State of Saxony, contract no. 99457/2677.

References

1. Google Knowledge Graph: `http://www.google.com/insidesearch/features/search/knowledge.html`.
2. Moresophy: L4 suite `http://www.moresophy.com/l4_suite` (only in German).
3. Topic/S: Project website `http://www.topic-s.de/` (only in German).
4. New York Times Dataset: `http://data.nytimes.com/`.
5. YAGO2 Dataset: `http://www.mpi-inf.mpg.de/yago-naga/yago/index.html`.
6. W3C Wiki: `http://www.w3.org/wiki/RdfStoreBenchmarking`.
7. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In: Procs. of the Intern. Conf. on Management of Data, ACM (2011) 145–156
8. Bizer, C., Schultz, A.: The berlin sparql benchmark. In: International Journal on Semantic Web & Information Systems. Volume 5. (2009)
9. Berlin SPARQL Benchmark: `http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/`.
10. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (October 2010)
11. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: Sp2bench: A sparql performance benchmark. CoRR **abs/0806.4627** (2008)
12. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: The Semantic Web ISWC 2011. Volume 7031 of LNCS. Springer (2011) 585–600
13. Haslhofer, B., Roochi, E.M., Schandl, B., Zander, S.: Europeana rdf store report. Technical report, University of Vienna, Vienna (March 2011)

14. Apache Jena: http://jena.apache.org.
15. BigData RDF Database: http://www.systap.com/bigdata.htm.
16. OWLIM: http://www.ontotext.com/owlim.
17. OpenLink Virtuoso: http://virtuoso.openlinksw.com/.
18. RDF2RDF: http://www.l3s.de/~minack/rdf2rdf/.
19. Voigt, M., Mitschick, A., Schulz, J.: Yet another triple store benchmark? practical experiences with real-world data (website) http://mt.inf.tu-dresden.de/topics/bench.