Evaluation of a Representative Selection of SPARQL Query Engines using Wikidata

An Ngoc Lam[1][0000−0003−4929−054X], Brian Elvesæter[1][0000−0001−7304−4950], and
Francisco Martin-Recuerda[1][0000−0002−7146−692X]

SINTEF AS, Forskningsveien 1, 0373 Oslo, Norway
{an.lam,brian.elvesater,francisco.martin-recuerda}@sintef.no

Abstract. In this paper, we present an evaluation of the performance of five representative RDF triplestores, including GraphDB, Jena Fuseki, Neptune, RDFox, and Stardog, and one experimental SPARQL query engine, QLever. We compare importing time, loading time, and exporting time using a complete version of the knowledge graph Wikidata, and we also evaluate query performances using 328 queries defined by Wikidata users. To put this evaluation into context with respect to previous evaluations, we also analyze the query performances of these systems using a prominent synthetic benchmark: SP^{2}Bench. We observed that most of the systems we considered for the evaluation were able to complete the execution of almost all the queries defined by Wikidata users before the timeout we established. We noticed, however, that the time needed by most systems to import and export Wikidata might be longer than required in some industrial and academic projects, where information is represented, enriched, and stored using different representation means.

Keywords: RDF Triplestores · Knowledge Graphs · SPARQL Benchmarks · SP^{2}Bench · Wikidata.

1 Introduction

Wikidata [34] is a collaboratively edited multilingual knowledge graph hosted by the Wikimedia Foundation. Wikidata is becoming a prominent software artifact in academia and industry, and offers a broad collection of terms and definitions that can improve data understandability, integration, and exchange. Wikidata is stored as an RDF [35] graph that can be queried with the SPARQL language [36]. With more than 16 billion triples and 100 million defined terms, as the size of the knowledge graph continuously increases, it might be challenging for state-of-the-art triplestores to import, export, and query Wikidata. This is also acknowledged by the Wikimedia Foundation, which is looking for alternatives to replace Blazegraph [11], an open-source triplestore no longer in development [39].

To investigate how efficiently RDF triplestores can handle Wikidata, in this paper, we present an evaluation of the performance of five representative RDF triplestores, including Ontotext GraphDB [20], Apache Jena Fuseki [6], Amazon

Neptune [2], OST RDFox [24], and Stardog [31], and one experimental SPARQL query engine - QLever [9]. We compare importing, loading, and exporting time using a complete version of Wikidata, and we also evaluate query performances using 328 queries defined by Wikidata users.

Due to budget limitations, we limited our evaluation to only six representative tools. However, we tried to ensure a diverse selection. For instance, GraphDB, Neptune, RDFox, and Stardog are commercial applications, whereas QLever and Jena Fuseki are not. Neptune is based on Blazegraph, and it is available as a native cloud-based service. RDFox is an in-memory triplestore, while the others are persistent. QLever and Jena Fuseki are in the pool of tools considered by Wikimedia Foundation to replace Blazegraph. So we thought both could represent a good baseline for our study. After some deliberation, we also decided to conduct an evaluation of the six triplestores using the synthetic benchmark SP^2Bench [28], which has strong theoretical foundations and a main focus on query optimization, also very relevant in our analysis of query performances of SPARQL engines using Wikidata.

In our evaluation using a full version of Wikidata and large datasets generated by SP^2Bench, we study the query execution plan with query profiling information to understand better the difference in the performance of the triplestores on the same query. The ultimate goal is to obtain a thorough understanding of the impact of SPARQL features on the performance and confirm common best practices in the design of SPARQL queries. We also consider import and export time. As service-oriented and decentralized architecture has become a popular design for software infrastructure in recent years, importing and exporting performance may be critical to enable efficient data transformation and exchange, especially for big data applications (e.g., big data pipelines or Machine Learning pipelines). Therefore, it is important to optimize importing and exporting functionality to avoid bottlenecks in the execution of such applications.

Despite not considering the evaluation of the concurrent execution of SPARQL queries, we observed that most of the evaluated systems could complete the execution of almost all queries defined by Wikidata users before the timeout. However, most of them required a longer time to import and export a full Wikidata version than expected in many industrial and academic projects, where information is represented, enriched, and stored using a diverse selection of applications offering different representation means.

To help interested readers to dive into the details of this evaluation, all scripts, data, and results have been uploaded to an open repository [15]. The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 describes the evaluation setup. Section 4 provides a detailed discussion of the evaluation results using SP^2Bench and Wikidata. Finally, Section 5 concludes the paper and presents future work directions.

2   Related Work

To better relate our evaluation using Wikidata with previous evaluation papers, we reviewed existing benchmarks based on both synthetic and real-world datasets. Benchmarks that use synthetic datasets include *LUBM* (Lehigh University Benchmark) (2005) [16], *UOBM* (University Ontology Benchmark) (2006) [18], *BSBM* (Berlin SPARQL Benchmark) (2009) [10], SP^2Bench (SPARQL Performance Benchmark) (2009) [28], *Bowlognabench* (2011) [12], *WatDiv* (Waterloo SPARQL Diversity Test Suite) (2014) [1], *LDBC-SNB* (Linked Data Benchmark Council - Social Network Benchmark) (2015) [13], *TrainBench* (2018) [33] and *OWL2Bench* (2020) [29]. The sizes of the synthetic datasets used in the referenced papers ranged from 1M to 100M triples, and the numbers of SPARQL queries executed were between 12 and 29, with the exception of *WatDiv* that used a set of 12500 generated queries.

Amongst benchmarks that are based on real-world datasets or queries from real-world logs, we reviewed *DBPSB* (DBpedia SPARQL Benchmark) (2011) [19], *FishMark* (2012) [8], *BioBenchmark* (2014) [40], *FEASIBLE* (2015) [26], *WGPB* (Wikidata Graph Pattern Benchmark) (2019) [17] and *WDBench* (2022) [5]. The datasets for these benchmarks varied in size from 14M up to 8B triples, and the numbers of SPARQL queries defined were between 22 and 175, except for *WGPB* and *WDBench* that have 850 and more than 2000 queries respectively.

The study conducted by the Wikimedia Foundation to replace Blazegraph is the closest work we have been able to identify so far. This study provided a detailed analysis of relevant features of triplestores according to Wikimedia Foundation. This study, however, only considered open-source triplestores and did not include execution times for importing, loading, exporting, or querying Wikidata [37]. Another related study [14] discussed the possibility of hosting a full version of Wikidata, and it measured the import time of popular triplestores, including Jena Fuseki, QLever, and Stardog. This study, however, did not discuss export time or query performances. *WGPB* and *WDBench* rely on a substantially reduced version of full Wikidata, and they have very specific objectives. *WGPB* defines a large set of SPARQL basic graph patterns exhibiting a variety of increasingly complex join patterns for testing the benefits of worst-case optimal join algorithms. The design goal behind *WDBench* was to create an evaluation environment able to test not only graph databases supporting RDF data model and SPARQL query language. The authors of *WDBench* created a collection of more than 2000 SPARQL queries distributed in four different categories. These queries were selected from real Wikidata query logs. Due to budget constraints, we did not include the queries defined by *WDBench* in our study. Still, we would like to include them in an extended version of this evaluation and compare the results with the queries we selected.

3   Evaluation Setup

In this section we present the details of the evaluation by describing the operational setup.

Triplestores To ensure that our limited selection of triple stores is representative and diverse, the following triplestores were evaluated: (1) Jena Fuseki 4.4.0 with Jena TDB2 RDF store, (2) Amazon Neptune Engine 1.0.5.1, (3) GraphDB Enterprise Edition 9.10.0, (4) RDFox 5.4, (5) QLever (commit version 742213facfcc80af11dade9a971fa6b09770f9ca), and (6) Stardog 7.8.0. In this selection: there are commercial and non-commercial (Jena Fuseki and QLever) applications; there is one triplestore distributed as native cloud-service (Neptune); and there is one in-memory triplestore (RDFox). All triplestores support SPARQL 1.1 syntax and provide querying services via SPARQL endpoints.

Datasets We aim to evaluate the scalability and performance of the SPARQL query engines using large datasets. For SP^{2}Bench, we generated four different datasets with 125M, 250M, 500M, and 1B triples. For Wikidata, we used the full version `latest-all.nt.gz` (downloaded on 2021-11-19). Table 1 shows the general statistics of these datasets.

Table 1. Statistics of the datasets: number of distinct Triples, Sub[jects], Pred[idcates], Obj[ects], Class[es], Ind[ividuals], Obj[ect] Prop[erties] and Data Prop[erties].

Benchmark	Triples	Sub	Pred	Obj	Class	Ind	Obj Prop	Data Prop
	125M	22.4M	78	59.5M	19	22.4M	64	21
SP^{2}Bench	250M	45.9M	78	120.8M	19	46.2M	64	21
	500M	94M	78	244.9M	19	94.1M	64	21
	1B	190.3M	78	493M	19	190.5M	64	21
Wikidata	16.3B	1.78B	42.92K	2.93B	1.2K	1.77K	17.1K	27K

SPARQL Queries SP^{2}Bench comes with a set of 14 SELECT and 3 ASK queries which were designed to cover several relevant SPARQL constructs and operators as well as to provide diverse execution characteristics in terms of difficulty and result size [28]. For Wikidata, the set of 356 SPARQL query examples defined by Wikidata users [38] was selected. Some of these queries use proprietary service extensions deployed for the Wikidata Query Service. We modified the queries to not use these service extensions and discarded some queries that are not compliant with SPARQL 1.1 specification or use proprietary built-in functions not supported by the evaluated triplestores. As a result, a set of 328 queries is used for the evaluation.

Table 2. Coverage (%) of SPARQL features for each benchmark.

Benchmark	distinct	filter	optional	union	limit	order	bound	offset
SP^{2}Bench	35.29	58.82	17.65	17.65	5.88	11.76	11.76	5.88
Wikidata	33.14	30.84	31.12	5.19	14.7	48.7	2.59	0
	DateFnc	**SetFnc**	**NumFnc**	**StringFnc**	**TermFnc**	**exists**	**notexists**	**in**
SP^{2}Bench	0	0	0	0	0	0	0	0
Wikidata	8.65	27.67	2.59	9.8	16.14	0.58	3.75	1.44
	groupby	**bind**	**values**	**minus**	**coalesce**	**if**	**having**	**PropPath**
SP^{2}Bench	0	0	0	0	0	0	0	0
Wikidata	29.11	10.66	8.65	5.19	0.86	3.17	1.44	35.73

Table 2 presents an analysis of the queries from the two benchmarks regarding the SPARQL features and operators. These features may have a correlation

with the execution time of the queries [25,27]. Hence, they need to be taken into consideration when designing a SPARQL query benchmark. It can be observed that Wikidata queries provide a broader coverage of SPARQL features and operators than SP^{2}Bench. Wikidata queries have more advanced features [36] such as Property Path, or built-in functions such as Dates and Times (DateFnc), Set (SetFnc), Strings (StringFnc), and RDF Terms (TermFnc) functions.

Amazon Web Services (AWS) Infrastructure This evaluation was conducted on the AWS cloud. We used Amazon Elastic Compute Cloud (EC2) instances with Elastic Blob Store (EBS) volumes. Specifically, `r5` instances with memory configurations of 128GB, 256GB, and 512GB were selected. This corresponds to instances of type `r5.4xlarge`, `r5.8xlarge`, and `r5.16xlarge` [3]. This choice also matched the on-demand `r5` instances available for the fully managed Neptune triplestore [4]. As RDFox is an in-memory triplestore, it needs additional memory to load a full version of Wikidata. None of the available `r5` instances offers enough memory for RDFox. Therefore, `x1` instances which offer up to 1,952 GB of memory were selected instead. In particular, `x1.32xlarge` [3] was employed to evaluate RDFox using Wikidata. Each EC2 instance was set up with a separate EBS `gp3` volume for data storage with the performance of 3,000 IOPS and 125 MB/s throughput.

Configuration Details We followed the recommended memory configuration for Stardog [30] and GraphDB [21], and applied it to all triplestores. We used the default settings for other configurations. In the case of RDFox, this implies that we use a persistence mode that stores incremental changes in a file [23]. RDFox can also be set up to run purely in-memory. According to the vendor, this would result in much lower import and export times than the ones presented in the paper. Similarly, RDFox offers the possibility to store datastores as binary files [22], which might significantly reduce loading times. These claims could not be verified before this study was submitted.

The evaluation was carried out simultaneously with one triplestore running on one instance. For SP^{2}Bench, `r5.8xlarge` was used to deploy all triplestores. For Wikidata, we ran the evaluation on `r5.4xlarge`, `r5.8xlarge`, and `r5.16xlarge`, except RDFox that was deployed only on `x1.32xlarge`. Due to some differences in the hardware configuration of `r5` and `x1`, we performed a sensitivity analysis of their performance. The result of this analysis is discussed in Section 4.2.

To avoid the impact of network latency, the triplestores (i.e., SPARQL server) and the evaluation scripts (i.e., SPARQL client) were deployed on the same machine. Neptune is provided as database-as-a-service in the cloud. Thus, the SPARQL client needs to run on a separate machine. To estimate the effect of network latency, we set up a test with GraphDB where the SPARQL client was running on a separate machine. This analysis helped us to adjust and make the results for Neptune comparable with the others. Detail about this analysis is discussed in Section 4.2.

The evaluation is comprised of the following stages:

1. **Data Import**. All datasets were imported into the selected triplestores.
2. **System Restart and Warm-up**. The triplestores were restarted, and the evaluated dataset was loaded again (if needed). Then, one test query was executed to warm up the triplestores.
3. **Hot-run**. Each query was executed ten times. We set the query timeout to 30 minutes for the SP^{2}Bench and 5 minutes for the Wikidata.

The following metrics are recorded in this evaluation:

- **Import Time**. The time required to import the dataset for the first time. This step involves building indexes and persisting the datasets to storage.
- **(Re)Load Time**. The time required to (re)load the dataset after importing.
- **Export Time**. The time required to write imported data to an external file.
- **Query Execution Time**: The time needed to finish one query execution.
- **Success Indicators**. The numbers of success, error, and timeout queries.
- **Global Performance**. We follow the proposal in [28] to compute both well-known arithmetic mean and geometric mean (the nth root of the product over n number) of the execution times. Accordingly, the failed queries (e.g., timeout, error) were penalized with the double of timeout value. Arithmetic mean is used as an indicator of a high success and failure ratio (i.e., a smaller value indicates a higher ratio of success queries) while geometric mean is used to evaluate the overall performance over success queries (i.e, a smaller value as an indicator of shorter execution time for the success queries).

Although we ran our experiments on a cloud-based framework, it is worth mentioning that the executions of each run are remarkably consistent. Specifically, the standard deviation of the ten runs of 95% of the Wikidata queries is less than one millisecond.

4   Discussion of the Evaluation Results

In this section, we discuss the evaluation results using SP^{2}Bench and Wikidata. Additional supplementary information and all experimental results, including runtimes for individual queries on each engine tested, can be found online at [15].

4.1   Evaluation results using SP^{2}Bench

Import Time Table 3 includes the import time of SP^{2}Bench datasets. For all triplestores, the import time increases proportionally to the size of the dataset. Jena Fuseki showed poor import performance even though the `tdb2.xloader` [7] - a multi-threading bulk loader for very large datasets - was employed.

RDFox is the fastest when importing the datasets, even though it was configured using persistence mode. RDFox also exhibited similar loading times. As discussed in Section 3, RDFox importing and loading times might be reduced using a different configuration. The other triplestores show similar importing times and very fast loading times. For instance, they were able to restart and reload the synthetic dataset with 1B triples in less than a minute.

Table 3. Global performance of the triplestores on SP^{2}Bench. To compute the mean, Timeout and Error queries were penalized with 3600 seconds (1 hour).

Triple stores	125M			250M			500M			1B		
	Imp Time	Arith Mean	Geo Mean	Imp Time	Arith Mean	Geo Mean	Imp Time	Arith Mean	Geo Mean	Imp Time	Arith Mean	Geo Mean
QLever	17m	1694.23	3.96	35m	1694.23	4.10	1h9m	1694.24	4.24	2h20m	1702.58	**5.74**
Fuseki	33m	1089.78	14.04	1h6m	1324.24	23.18	2h15m	1370.08	29.28	5h40m	1816.33	45.03
Neptune	18m	898.35	7.11	37m	974.22	9.89	1h16m	1323.48	13.83	2h22m	1781.18	26.40
RDFox	**3m**	1061.38	**1.75**	**5m**	1065.10	**2.42**	**9m**	1074.52	**2.91**	**18m**	1528.20	6.98
Stardog	18m	862.86	2.76	37m	878.88	3.62	1h17m	**917.33**	5.54	2h33m	1378.62	9.10
GraphDB	17m	**728.62**	3.97	36m	**766.16**	5.00	1h11m	995.00	7.55	2h23m	**1248.67**	11.75

Table 4. Success indicators (S[uc]C[ess], T[ime]O[ut], ERR[or]) on SP^{2}Bench.

Triplestore	125M			250M			500M			1B		
	SC	TO	ERR	SC	TO	ERR	SC	TO	ERR	SC	TO	ERR
QLever	9	0	8	9	0	8	9	0	8	9	0	8
Jena Fuseki	12	5	0	11	6	0	11	6	0	9	8	0
Amazon Neptune	13	4	0	13	4	0	11	6	0	9	8	0
RDFox	12	5	0	12	5	0	12	5	0	10	7	0
Stardog	13	4	0	13	4	0	13	4	0	11	6	0
GraphDB	15	2	0	15	2	0	14	3	0	13	4	0

Query Execution Time Table 3 and 4 present success indicators and average execution time for the four SP^{2}Bench datasets. QLever is the only triplestore that had errors and no timeout. Seven queries could not be executed due to unsupported syntax or functions (e.g., ASK query, combined conditions in FILTER) and one "OutOfMemory" (OOM) error.

According to Table 4, the size of the dataset and results has a significant effect on the performance of all triplestores, in particular as the dataset grows from 125M to 1B triples. Neptune, as mentioned earlier, may suffer from network latency, especially for queries with large results because the server and client were deployed on different machines. For SP^{2}Bench, the timeout was set to 30 minutes which is relatively long enough for transferring big data between Amazon machines. In fact, in 8 timeout cases, Neptune failed to finish the execution of the queries and returned no result. Therefore, network latency is not the main issue for these timeout queries. Moreover, compared to the average execution time of the 17 SP^{2}Bench queries which is about 15 minutes, network latency may be considered as an insignificant factor. A thorough evaluation of network latency will be discussed in Section 4.2 using the Wikidata benchmark where there are a lot of queries executed in less than 100 milliseconds.

Regarding the global performance, the arithmetic means of GraphDB were superior to the others since it had a higher number of success queries. However, RDFox had better performance over successful queries, so its geometric means were the smallest; timeouts were limited to queries that introduce equi-joins using FILTER statements. In all cases, Stardog was always in the top two. It had more success queries than RDFox and executed difficult queries slightly faster than GraphDB. Jena Fuseki delivered the poorest performance while Neptune had mixed results on query execution. QLever was very fast on success queries, but it offered limited support for queries with complex SPARQL constructs.

Moreover, QLever automatically puts a limit up to 100.000 results for all queries. Therefore, its reported execution times may not be comparable, especially for queries with large results.

Analysis of Query Execution Plan A SPARQL query can be represented as a Basic Graph Pattern (BGP) which is a set of Triple Patterns (TPs) specified in the query [25]. Typically, the result of a BGP is obtained by joining the results of the TPs. Therefore, selective TPs that have smaller result sizes are usually executed first in order to minimize the number of intermediate results and therefore reduce the cost related to joining operations [17]. For the same purpose, filters are also moved closer to the part of the BGP where they apply. The execution order of the TPs in the BGP is the query execution plan. Typically, the query execution plan needs to be decided before the SPARQL engine executes the query. In order to do this, the triplestore requires precise estimation of the result size of each TP, which is done through building and updating different types of indexes [17]. In general, different triplestores may employ different data structures to implement their indexes and use different algorithms to optimize their query execution plan, therefore resulting in varying performance. In this study, except for QLever which does not provide the method to get the query plan, we investigated the execution plans of the other triplestores in order to gain a better understanding of their performances on the benchmarks.

In addition to the execution plan, RDFox, Neptune, and especially, Stardog also include very comprehensive profiling reports with the actual result sizes and the execution time of each TP. However, GraphDB provides only the estimation of the result sizes while Jena Fuseki produces only the complete results of each TP. Therefore, for these two triplestores, it was difficult to diagnose performance problems or identify expensive operations for the difficult queries.

Query 2 is one of the most difficult queries of the SP^{2}Bench where many triplestores timed out. It has a bushy BGP (i.e., single node linked to a multitude of other nodes) with 10 TPs, and the result size of this query grows with database size. Therefore, the execution time might be linear to the dataset size. According to the statistics provided by Stardog query profiler, the most expensive operation for this query is post-processing data (i.e., converting the results into the data structure that will be sent to the client). Similarly, Amazon Neptune also spent most of the execution time for this query on post-processing data. This observation illustrates the effect of large result sizes and large strings on the querying performance of SPARQL engines.

Query 3 (a, b, c) has just two TPs, one of which is of the form (?,?,?), and one FILTER with "equal to" operator. To avoid evaluating the TP (?,?,?) which may result in matching the whole dataset, all triplestores embedded the filter expression into this TP and transformed it into (?,p,?) form.

Query 4 is the most challenging query of the SP^{2}Bench benchmark. The result of the query is expected to be quadratic in the number of "journal" individuals in the dataset. To deal with this query, the author in [28] suggested that the query engines embed the FILTER expression into the computation of TPs (i.e., the same approach done in Query 3), which may help to reduce the

intermediate results earlier. However, it is more challenging for all triplestores to embed the "greater than" operator in this query than the "equal to" operator. As a result, all of the triplestores failed to complete the query due to timeout.

Query 5a and 12a also test optimizations for embedding FILTER expression. The expression in these queries is the "equal to" comparison between two variables while Query 3 filters a variable with a constant value. GraphDB is the only triplestore handling Query 5a in 30 minutes. After rewriting this query by explicitly embedding the filtering expression, the triplestores can execute the query before timeout. However, as the dataset increased to 1B, timeout still occurred for the others. Query 12a replaces the SELECT construct of Query 5a with the ASK, which has a positive effect on query performances in all triplestores with the exception of Qlever and RDFox. This might indicate that collecting the results of Query 5a also has a significant impact on performances.

Query 6, 7, and 8 test another different optimization approach related to reusing TP results. These queries have several TPs repeated multiple times. Thus, intermediate results of those TPs can be reused to save cost for matching those triples. From the execution plan, it is unclear whether the triplestores implement this optimization approach or the same TPs were executed again.

Overall, when evaluating the execution plans, we observed that the triplestores did not pass several optimization tests designed by the authors of the benchmark. Despite being a synthetic benchmark with only 17 queries where some tests may not be practical (e.g., Query 4) or biased towards some specific constructs (e.g., FILTER), SP^{2}Bench proved to be very useful to test common query optimization techniques and to collect useful insights of the triplestores selected for this evaluation. Next, we will present the evaluation results using a completed version of Wikidata and 328 queries defined by its user community.

4.2   Evaluation results using Wikidata

Import and Export Time The Wikidata dump used in this evaluation is available as a 112GB gzip file (738GB as unzip file). QLever is the only triplestore that has no support for gzip format. However, due to errors during importing, only Stardog, GraphDB, and RDFox managed to load the gzip file. Jena TDB2, in particular, was not able to import Wikidata due to 1319 URI syntax errors (e.g., special characters not allowed by Jena RIOT - the Jena syntax validator). After fixing these errors by replacing the special characters with their HTML numeric codes, we used this "clean" version to import into Jena Fuseki. Jena Fuseki also suffered from OOM error and succeeded in importing Wikidata only on `r5.16xlarge` machine (512GB RAM).

Table 5 presents the performance for importing and exporting Wikidata. RD-Fox was much faster than the others. This result is consistent with the figures reported in SP^{2}Bench where all triplestores are evaluated using the same machine configuration. However, as the triplestores were restarted, RDFox required around 3.75 hours (40% faster than its initial import time) to reload the data while the others took only a few minutes. As discussed in Section 3, RDFox importing and loading times might be reduced using a different configuration.

Table 5. Import and Export performance of the triplestores on Wikidata.

Triplestore	VM	Import Time	(Re)Load Time	Export Time	Persisted Storage
QLever	r5.4xlarge	1d 17h 2m	11m	n/a	871 GB
Jena Fuseki (TDB2)	r5.16xlarge	3d 15h 27m	<1m	Timeout	1.52 TB
Stardog	r5.4xlarge	2d 1h 9m	<1m	Error	862 GB
Amazon Neptune	r5.4xlarge	3d 1h 50m	7m	Error	3.98 TB
GraphDB	r5.4xlarge	1d 8h 13m	10m	Timeout	1.11 TB
RDFox	x1.32xlarge	0d 6h 25m	3h42m	5h28m	202 GB

As QLever, Jena Fuseki, and Amazon Neptune used the unzip data and may not require any decompress operation during importing, their performance is expected to be slower on gzip Wikidata. Also, we observed that QLever reported a much larger number of loaded triples (21.5B triples). QLever may have a different way of building the statistics on the imported data.

To measure the export time, we set a timeout of 4 days for the triplestores. Except for QLever which has no support for data exporting, the other triplestores provide native functions to export the data. However, RDFox is the only triplestore that succeeded in exporting Wikidata within the timeout. Stardog did not show any progress or runtime output while Amazon Neptune encountered an error after exporting 503M statements in 1.5 hours. GraphDB took 28 days and 8 hours to export Wikidata. Due to cost constraints, we did not continue the exporting process for the others after 4 days. Based on this figure, it is obvious that exporting is not a prioritized feature of most triplestores.

Query Execution Time Table 6 presents the success indicators and mean execution time of the triplestores on the Wikidata benchmark. QLever reported the most errors (67% of all queries). Nearly all are syntax errors due to limited support for the SPARQL 1.1. Furthermore, there are 5 queries where QLever returns only the first 100.000 results. They were also classified as errors. QLever also has several OOM errors that were resolved on more powerful machines.

Jena Fuseki is the second triplestore with the most errors. It had 13 query syntax errors. Particularly, it does not allow using an existing variable name for the AS operator (e.g., `(SAMPLE(?dob) AS ?dob)`). Jena Fuseki also suffered from memory issues. This triplestore either crashed or froze and produced no output while executing the other 8 error queries. Amazon Neptune and Stardog accounted for one error which was reported as an internal failure exception.

If we look at the performance of each individual triplestore on the three different `r5` configurations, GraphDB and Stardog were more robust with small variances among the machines. They had approximately the same query execution time and number of timeouts on the three machines. They may be optimized to work efficiently even on machines with less physical resources. In contrast, Jena Fuseki and Amazon Neptune performed better on more powerful machines as they had more success queries on those machines.

When comparing the execution time of all triplestores, RDFox and GraphDB are the top two triplestores with the lowest arithmetic mean followed by Amazon Neptune. Jena Fuseki and QLever are the slowest triplestores due to a lot of

Table 6. Global performance of the triplestores on Wikidata benchmark. To compute the mean, Timeout and Error queries were penalized with 600 seconds (10 minutes). The table also contains the estimated mean (in brackets) for RDFox on `r5` machine and for Amazon Neptune r5.16x (r5.16xlarge VM) with network latency deducted.

Triplestore	VM	SC	TO	ERR	Arithmetic Mean	Geometric Mean
Qlever r5.4x	r5.4xlarge	106	0	222	404.87	12.33
Qlever r5.8x	r5.8xlarge	107	0	221	403.05	11.91
Qlever r5.16x	r5.16xlarge	108	0	220	401.23	11.73
Jena Fuseki r5.4x	r5.4xlarge	224	83	21	192.30	1.43
Jena Fuseki r5.8x	r5.8xlarge	231	76	21	180.21	1.29
Jena Fuseki r5.16x	r5.16xlarge	250	57	21	148.57	1.20
Amazon Neptune r5.4x	r5.4xlarge	309	18	1	39.29	0.34
Amazon Neptune r5.8x	r5.8xlarge	310	17	1	36.26	0.31
Amazon Neptune r5.16x	r5.16xlarge	312	15	1	31.65 (31.59)	0.28 (0.27)
Stardog r5.4x	r5.4xlarge	307	20	1	43.46	0.19
Stardog r5.8x	r5.8xlarge	308	19	1	42.01	0.16
Stardog r5.16x	r5.16xlarge	308	19	1	41.77	0.18
GraphDB r5.4x	r5.4xlarge	321	7	0	15.62	0.08
GraphDB r5.8x	r5.8xlarge	322	6	0	14.48	0.08
GraphDB r5.16x	r5.16xlarge	321	7	0	15.67	0.07
RDFox x1.32x	x1.32xlarge	324	4	0	12.11 (9.23)	0.04 (0.016)

errors and timeouts. Regarding geometric mean, RDFox is also the fastest with a value of 0.04 which is 50% smaller than the second best, GraphDB. Stardog is in the third place. Its geometric mean is around 0.18 which is 35% faster than Amazon Neptune. This insight can also be noticed from Figure 1 that compares the best performance (i.e., on `r5.16xlarge` machines) of the triplestores for the top 50 easy and difficult queries (excluding error queries). Accordingly, for easy queries, it can be clearly identified the order of the triplestores where RDFox is the fastest and Amazon Neptune is the slowest. However, there is a mixed result for difficult queries.
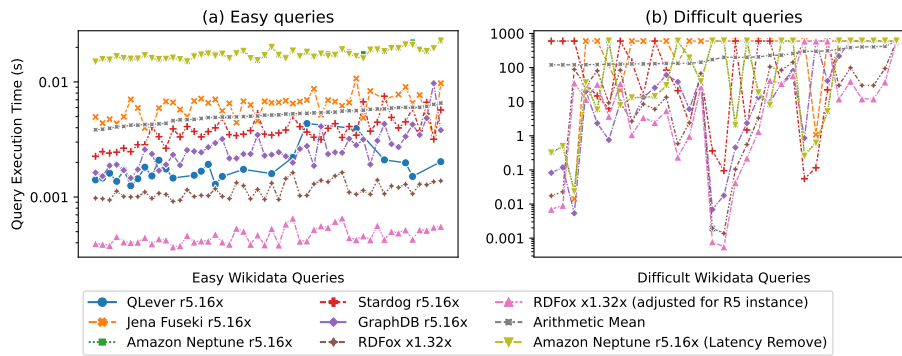

Fig. 1. Query execution time of the top 50 easy and difficult Wikidata queries on average (excluding error queries). The queries (x axis) are ordered by the arithmetic mean of the execution time of all triplestores.

Network Latency Analysis In order to estimate network latency which can be incorporated into the execution time of Amazon Neptune, especially for those queries with large results, a separate experiment was conducted. Specifically, the same setting was deployed for GraphDB on two `r5.4xlarge` machines. On average, the latency amounted to 100 milliseconds. In Figure 1(a) and Table 6, we also have the execution time of Amazon Neptune adjusted by removing the latency for each individual query. As the latency is just a few milliseconds for easy queries, the adjusted figure is not different from the original value. Amazon Neptune is still the slowest on the top 50 easy queries.

Sensitivity Analysis of R5 and X1 Instances To compare the performance of the triplestores on `x1` and `r5` instances, we performed a sensitivity analysis that: (1) evaluates the performance of GraphDB with Wikidata on `x1e.4xlarge` and `r5.4xlarge` machines, (2) evaluates the performance of RD-Fox with SP²Bench on `x1e.8xlarge` and `r5.8xlarge` machines. On average, GraphDB had approximately 50% performance degradation on `x1e.4xlarge` machine. Similarly, the performance of RDFox on SP²Bench decreased about 59.75% on `x1e.8xlarge` machine. Thus, it is expected that RDFox may have better performance on `r5` instances for Wikidata queries if a suitable `r5` machine with sufficient RAM memory is available. To ensure the results of this analysis are reflected in the evaluation, in Table 6, we also provide the adjusted means for RDFox assuming the queries would be executed 59.75% on average faster.

Analysis of Query Execution Plan To better understand the evaluation results discussed earlier, several queries are studied in more detail. In particular, the following selection criteria were applied:

1. Queries not executed by most of the triple stores due to timeout.
2. Queries with large variation in execution times (i.e., timeout for some triple-stores and executed in few seconds by the rest).
3. Queries where the numbers of results are not consistent (i.e., some triple-stores returned different numbers of results for the same query).

Due to the page limitation, only a summary of the analysis is presented in this section. Firstly, for timeout queries, the most expensive operations are related to processing a large number of results, including both intermediate and final results. Therefore, to minimize overhead related to handling the results, highly selective triple patterns are usually prioritized to be executed earlier in order to reduce the scanning space for later triple patterns of the query. To do so, the query optimizer needs to have a good estimation of the outputs for each TP in the query. For queries with simple SPARQL constructs, most triplestores can manage to create an optimal execution plan. However, as the query employs complex constructs or features (e.g., nested SELECT query, built-in functions, property path, etc.), estimating a good execution plan becomes challenging. Based on the analysis of the query execution plans, the following observations can be considered when designing the queries for the evaluated triplestores:

– For queries with complicated patterns, most triplestores tend to keep the execution plan the same as the original order described in the query. In this

case, it is recommended to rewrite the query using simple constructs. For example, the property path with arbitrary length can be rewritten explicitly with a sequence of TPs and UNION. For example, after we rewrite the property path of **Query 234** and **Query 235** with the sequence of three patterns explicitly, Stardog can produce a better plan and result in 2.6x and 4.2x faster respectively. If the query cannot be changed, the TPs need to be re-ordered manually to help the triplestores to optimize the execution plan. This can be done by repeating the execution with different orders [32].

- FILTER operations are usually moved earlier in the execution plan in order to reduce the number of intermediate results. However, filters with complex conditions (e.g., string and date-time functions) may slow down the execution, especially for a large number of results (e.g., **Query 192**, **Query 327**, **Query 343** and **Query 350**). If possible, such condition should be rewritten in an equivalent form without using FILTER or applied later in the execution plan when there are fewer intermediate results.
- The TP of the form (?,?,?) should be considered carefully as it may result in a bottleneck in the execution of most triplestores (e.g., **Query 45**).

For queries with large variation in execution times, the following observations about the evaluated triplestores are recorded:

- Due to its in-memory solution, RDFox tends to perform better on scanning indexes and joining results, especially for large results or complicated operations such as string functions (e.g., **Query 13**, **Query 233**, **Query 237** and **Query 350**).
- Jena Fuseki had very poor performance on scanning and joining large results compared with the others. It also has very simple optimization algorithms. Mostly, it does not change the order of triple patterns, which results in inefficient execution plans and timeouts (e.g., **Query 45**, **Query 84** and **Query 286**). Therefore, the triple patterns need to be re-ordered manually in order to improve the performance of this triplestore.
- Stardog may have issues with queries having many OPTIONAL constructs. The triplestore tends to produce exponential intermediate results when matching such triple patterns (e.g., **Query 176** and **Query 326**) while the others produced much fewer results, and therefore resulting in timeout.
- For queries with sequence paths, Amazon Neptune tends to prioritize triple patterns with such property path syntax. This strategy may result in an exponential increase in the intermediate results if those patterns are not the most selective (e.g., **Query 84** and **Query 286**).
- For queries with UNION construct, RDFox tends to keep the triple patterns inside UNION unchanged while GraphDB tends to expand this construct by moving the JOIN operation inside each of the operands of UNION. Stardog and Amazon Neptune are more flexible in estimating the optimal plan for the UNION pattern. Therefore, if the optimal plan can be anticipated, it is recommended to rewrite the UNION patterns, especially for GraphDB and RDFox (e.g., **Query 184** and **Query 345**).

The third category of queries selected for further investigation is the one where the numbers of results are not consistent. Accordingly, we identified 10 queries where there is one triplestore that disagreed with the majority of other triplestores. As we only captured and compared the number of results, it is not sufficient to determine whether a triplestore is correct or not. However, if most of the triplestores report the same number of results, this might be a reasonable indication in terms of correctness. Based on the analysis of the execution plan, the following issues from the triplestores were identified:

- Amazon Neptune reported different results when executing a few queries with REGEX expressions. For instance, the triplestore returned no result after applying some filters with complex regex patterns (e.g., **Query 93**, **Query 133** and **Query 327**).
- Stardog reported different numbers of results for a few SELECT nested queries (e.g., **Query 82** and **Query 195**). Additionally, in **Query 284** which has a FILTER operator on date-time values, Stardog returned six more results than the others.
- GraphDB also had issues with a few nested SELECT queries. It returned no result for **Query 109** and **Query 319**. Additionally, in **Query 178** and **Query 233**, the triplestore returned much fewer results than the others.

5   Conclusions and Future Work

To the best of our knowledge, this study presents one of the most detailed analyses of the performances of a representative selection of the state-of-the-art triplestores using a complete version of the knowledge graph Wikidata. In this section, we conclude the paper with a summary of some of the most relevant observations produced by this evaluation.

With respect to the evaluation setup used in this study, all selected triplestores were tested on Amazon EC2 `r5` or `x1` instances. Despite some initial concerns about the reliability of the evaluation results, the execution times of each run were remarkably consistent. Amazon EC2 instances are also required to test Neptune, the only native cloud-based service in our evaluation. Some specific execution requirements posed by Neptune and RDFox difficult a fair comparative analysis. Sensitivity analyses were conducted to adjust the evaluation results for Neptune and RDFox and make them comparable with the others. While the impact of network latency in the Neptune client-server configuration seems to be small, the differences in performances between `r5` and `x1` instances seem to be significant (approximately 50% to 60% performance degrade).

SP^{2}Bench proved to be a great choice to test scalability and common query optimization techniques, which helps us to collect useful insights of the triplestores selected for this evaluation. RDFox was the fastest triplestore importing the synthetic datasets generated for this study and it has better performance over success queries. Regarding the global query performances, GraphDB was superior to the others, followed very close by Stardog. After analyzing query execution plans and query profiling information, we observed that the triplestores

did not pass several optimization tests designed by the authors of SP^{2}Bench. It is worth mentioning that Stardog provides a comprehensive query profiling service, which establishes a reference for other triplestores.

SP^{2}Bench also has some limitations. It only provides 17 SPARQL queries offering limited coverage of some SPARQL constructs and features. Moreover, some of these queries and the synthetic datasets do not seem to be practical in real use case applications. Our evaluation employing a complete version of Wikidata with 328 queries defined by its users seems to overcome these limitations. This evaluation helps us to stress the triplestores and identify relevant insights. Importing Wikidata, and especially, exporting Wikidata was challenging for all triplestores, where RDFox was significantly more efficient. RDFox was also the only triplestore that managed to export Wikidata, and it completed this operation in a few hours. Loading Wikidata, however, was done much faster by the other triplestores, although a different configuration for RDFox might reduce loading time significantly.

Importing Wikidata was also difficult because of syntax errors reported by some rigorous parsers such as the ones implemented in RDFox and Jena Fuseki. It seems that the complete dumps published by Wikidata might not strictly follow W3C recommendations. For instance, it was possible to find values not formatted according to these recommendations. The same problem arises with some queries published by Wikidata users. Some of these queries use, for example, proprietary service extensions deployed by the Wikidata Query Service team.

In terms of query performances, RDFox reported the best overall performances followed by GraphDB. It is remarkable how consistent GraphDB and Stardog were, in terms of query performances independent of the memory configuration of the machine. This indicates a careful optimization of the design of both triplestores in terms of memory consumption. With the exception of QLever and Jena Fuseki, most triplestores reported none or just one error in the execution of the queries. Few discrepancies in the number of results were identified in the case of Stardog, GraphDB, and Neptune. The cause of these discrepancies could not be explained in this study and it will require further investigation.

As for future work, we plan to evaluate a larger collection of relevant triplestores or extend the queries used in the Wikidata evaluation with the queries defined by the benchmark WDBench.

Acknowledgment

References

1. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In: International Semantic Web Conference. pp. 197–212. Springer (2014)
2. Amazon AWS: Amazon Neptune Official Website, https://aws.amazon.com/neptune/
3. Amazon Web Services: Amazon EC2 Instance Types - Memory Optimized. https://aws.amazon.com/ec2/instance-types/#Memory_Optimized, accessed: 2022-12-12
4. Amazon Web Services: Amazon Neptune Pricing. https://aws.amazon.com/neptune/pricing/, accessed: 2022-12-12
5. Angles, R., Aranda, C.B., Hogan, A., Rojas, C., Vrgoč, D.: WDBench: A Wikidata Graph Query Benchmark. In: International Semantic Web Conference. Springer (2022)
6. Apache Jena: Apache Jena Fuseki Documentation, https://jena.apache.org/documentation/fuseki2/
7. Apache Jena: Apache Jena TDB xloader. https://jena.apache.org/documentation/tdb/tdb-xloader.html, accessed: 2022-12-12
8. Bail, S., Alkiviadous, S., Parsia, B., Workman, D., Van Harmelen, M., Concalves, R., Garilao, C.: FishMark: A Linked Data Application Benchmark. CEUR (2012)
9. Bast, Hannah and Buchhold, Björn: QLever GitHub repository, https://github.com/ad-freiburg/qlever
10. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(2), 1–24 (2009)
11. Blazegraph: Blazegraph Official Website, https://blazegraph.com/
12. Demartini, G., Enchev, I., Wylot, M., Gapany, J., Cudré-Mauroux, P.: BowlognaBench—Benchmarking RDF Analytics. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 82–102. Springer (2011)
13. Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.D., Boncz, P.: The LDBC Social Network Benchmark: Interactive Workload. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 619–630 (2015)
14. Fahl, W., Holzheim, T., Westerinen, A., Lange, C., Decker, S.: Getting and hosting your own copy of Wikidata. In: Proceedings of the 3rd Wikidata Workshop 2022. CEUR-WS.org (2022), https://ceur-ws.org/Vol-3262/paper9.pdf
15. GitHub: Analysis and supplementary information for the paper, including queries, execution logs, query results and scripts. https://github.com/SINTEF-9012/rdf-triplestore-benchmark, accessed: 2023-03-13
16. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics **3**(2-3), 158–182 (2005)
17. Hogan, A., Riveros, C., Rojas, C., Soto, A.: A Worst-Case Optimal Join Algorithm for SPARQL. In: International Semantic Web Conference. pp. 258–275. Springer (2019)
18. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a Complete OWL Ontology Benchmark. In: European Semantic Web Conference. pp. 125–139. Springer (2006)
19. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.C.: DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In: International semantic web conference. pp. 454–469. Springer (2011)
20. Ontotext: GraphDB Official Website, https://graphdb.ontotext.com/

21. Ontotext: GraphDB Requirements. https://graphdb.ontotext.com/documentation/enterprise/requirements.html, accessed: 2022-12-12

22. OST: RDFox Documentation: Managing Data Stores. https://docs.oxfordsemantic.tech/5.4/data-stores.html#, accessed: 2022-12-12

23. OST: RDFox Documentation: Operations on Data Stores, persist-ds. https://docs.oxfordsemantic.tech/5.4/data-stores.html#persist-ds, accessed: 2022-12-12

24. Oxford Semantic Technologies: RDFox Official Website, https://www.oxfordsemantic.tech/product

25. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.C.N.: LSQ: The Linked SPARQL Queries Dataset. In: International Semantic Web Conference. pp. 261–269. Springer (2015)

26. Saleem, M., Mehmood, Q., Ngonga Ngomo, A.C.: FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework. In: International Semantic Web Conference. pp. 52–69. Springer (2015)

27. Saleem, M., Szárnyas, G., Conrads, F., Bukhari, S.A.C., Mehmood, Q., Ngonga Ngomo, A.C.: How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks. In: The World Wide Web Conference. pp. 1623–1633 (2019)

28. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^{2}Bench: A SPARQL Performance Benchmark. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 222–233. IEEE (2009)

29. Singh, G., Bhatia, S., Mutharaju, R.: OWL2Bench: A Benchmark for OWL 2 Reasoners. In: International Semantic Web Conference. pp. 81–96. Springer (2020)

30. Stardog: Stardog Capacity Planning. https://docs.stardog.com/operating-stardog/server-administration/capacity-planning, accessed: 2022-12-12

31. Stardog: Stardog Official Website, https://www.stardog.com/

32. Stardog: 7 Steps to Fast SPARQL Queries. https://www.stardog.com/blog/7-steps-to-fast-sparql-queries/ (May 2017), accessed: 2022-12-12

33. Szárnyas, G., Izsó, B., Ráth, I., Varró, D.: The Train Benchmark: cross-technology performance evaluation of continuous model queries. Software & Systems Modeling **17**(4), 1365–1393 (2018)

34. Vrandečić, D., Krötzsch, M.: Wikidata: A Free Collaborative Knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)

35. W3C: RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. https://www.w3.org/TR/rdf11-concepts/, accessed: 2022-12-12

36. W3C: SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-query/, accessed: 2022-12-12

37. WDQS Search Team: WDQS Backend Alternatives: The process, details and result. Tech. rep., Wikimedia Foundation (2022), https://www.wikidata.org/wiki/File:WDQS_Backend_Alternatives_working_paper.pdf

38. Wikidata: SPARQL query service/queries/examples. https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples, accessed: 2022-12-12

39. Wikidata: SPARQL query service/WDQS backend update. https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/WDQS_backend_update, accessed: 2022-12-12

40. Wu, H., Fujiwara, T., Yamamoto, Y., Bolleman, J., Yamaguchi, A.: BioBenchmark Toyama 2012: an evaluation of the performance of triple stores on biological data. Journal of Biomedical Semantics **5**(1), 1–11 (2014)