January 2020

When is the Peak Performance Reached? An Analysis of RDF Triple Stores

Hashim KHAN [a,1] Manzoor ALI [a,2] Axel-Cyrille NGONGA NGOMO [a,3] and Muhammad SALEEM [b,4]

[a] *DICE Group, Department of Computer Science, Paderborn University*
[b] *AKSW, University of Leipzig, Germany*

Abstract. With significant growth in RDF datasets, application developers demand online availability of these datasets to meet the end users' expectations. Various interfaces are available for querying RDF data using SPARQL query language. Studies show that SPARQL endpoints may provide high query runtime performance at the cost of low availability. For example, it has been observed that only 32.2% of public endpoints have a monthly uptime of 99–100%. One possible reason for this low availability is the high workload experienced by these SPARQL endpoints. As complete query execution is performed at server side (i.e., SPARQL endpoint), this high query processing workload may result in performance degradation or even a service shutdown. We performed extensive experiments to show the query processing capabilities of well-known triple stores by using their SPARQL endpoints. In particular, we stressed these triple stores with multiple parallel requests from different querying agents. Our experiments revealed the maximum query processing capabilities of these triple stores after which point they lead to service shutdowns. We hope this analysis will help triple store developers to design workload-aware RDF engines to improve the availability of their public endpoints with high throughput.

Keywords. Triple Store, Throughput, Queries-per-Second, Availability

1. Introduction

One of the basic requirements of many semantic web applications is the ability to access and query live linked data. The term "live queryable" linked data demands that the data should be queryable via online SPARQL interfaces (without first downloading the entire knowledge graph) and processed locally to retrieve the desired information [1]. It is one of the most important demands for the successful deployment of many linked data-based applications. Various interfaces such as SPARQL endpoints and Triple Pattern Fragments (TPF) provide live SPARQL querying [1].

SPARQL endpoints offer a public interface to execute SPARQL queries over the underlying RDF datasets. In this interface, the client sends a complete SPARQL query to the server (i.e., SPARQL endpoint). The server executes the query and returns the final

[1]E-mail: hashim.khan@uni-paderborn.de
[2]E-mail: manzoor@campus.uni-paderborn.de
[3]E-mail: axel.ngonga@upb.de
[4]E-mail: saleem@informatik.uni-leipzig.de

results. The server is responsible for the execution of a complete query while the client is idle most of the time [2]. This model of query processing generally leads to better runtime performance due to the optimization techniques used in the server. Furthermore, the network overhead is low, as the complete query processing task is performed at one end. However, many of the SPARQL endpoints suffer from low availability rates [1,3]. According to the SPARQLES [4][5] current statistics,[6] only 176 (i.e. 20.71%) were found available out of a total 557 public endpoints. One potential reason for this low availability could be service shutdowns due to the high workload experienced by these SPARQL endpoints and the complex and expressive nature of SPARQL queries, which may require large processing time and resources. For example, the well-known public endpoints such as DBpedia[7] and Wikidata[8] receive more than 100K queries per day [5]. The RDF data storage and SPARQL query execution is performed by the backend triple store. For example, the DBpedia SPARQL endpoint is powered by the Virtuoso [6] triple store. The Wikidata endpoint works on top of the BlazeGraph[9] triple store. Every RDF query processing engine has a certain peak performance point when exposed to multiple parallel querying users. Exceeding the user workload beyond the maximum query processing capability of an engine would generally lead to performance degradation or even a service shutdown. The peak performance points of RDF triple stores depend upon multiple factors, including parallel query processing capabilities, the type of hardware resources being allocated, the efficiency of the underlying query planner, and the type of workload experienced. Multiple studies [7,8,9,10,11,12] have compared the performance of different triple stores; however, little attention has been paid to assessing parallel query processing capabilities of these triple stores [13]. To the best of our knowledge, no studies have reported the peak performance points under parallel loads of the state-of-the-art triple stores. We fill this gap by conducting extensive experiments and report the peak performance points of the triple stores with varying multiple parallel querying clients.

Our contributions are as follows:

- We performed experiments to show the maximum query processing capabilities of some well-known triple stores, with respect to the number of querying agents they can support, by using their SPARQL endpoints. In particular, we stressed these triple stores with multiple parallel requests from different numbers of querying agents.
- Beyond their peak performance points, we further stressed the selected triple stores towards launching a DoS attack.

The rest of the paper is organised as follows: In section 2, we provide a summary of the different evaluations related to RDF triple stores. Section 3 explains the evaluation setup and the evaluation results are presented in Section 4. Section 5 explains the availability of resources and their reusability and section 6 concludes this work. The complete data to reproduce the presented results is available from https://github.com/dice-group/RDF-Triplestores-Evaluation.

[5]SPARQLES Monitoring: https://sparqles.ai.wu.ac.at/availability

[6]Data taken on 31st of March, 2021 at 11:30 (CET)

[7]http://dbpedia.org/sparql

[8]https://query.wikidata.org/

[9]https://blazegraph.com/

2. Related Work

The focus of this section is to show the details of the experiments performed to evaluate the state-of-the-art triple stores. The main aim is to highlight the lack of research into the stress tolerance of different triple stores for their peak performance capabilities.

The importance of linked data and knowledge graphs has motivated the development of several RDF triple stores. Ali et al. [14] categorized a total of 116 triple stores: each employs different data storage and querying processing mechanisms. Consequently, various triple store benchmarks also have been developed. Saleem et al. [15] provide an analysis of 10 triple store benchmarks, each employing a different evaluation setup and experiments. Table 1 shows the list of the triple stores evaluated and details of the experiments conducted in these state-of-the-art triple stores benchmarks.

The performance metrics used by state-of-the-art triple store benchmarks to compare triple stores can be divided into four main categories.

- **Processing Related Metrics.** The metrics included in this category are related to the query processing capabilities of the triple stores. In this category, the Queries per Second (QpS), Queries Mix per Hour (QMpH), and Processing Overhead (PO) are the key metrics used in the state-of-the-art benchmarks.

- **Storage Related Metrics.** The metrics included in this category are related to the data storage and indexing techniques used in the triple stores. In this category, the data Loading Time (LT), the Storage Space (SS) required, and the Size of generated Indexes (IS) are the key metrics used in the state-of-the-art benchmarks.

- **Result Set Related Metrics.** The metrics included in this category are related to the result sets of the query execution over underlying triple stores. In this category, Result Set Completeness (RCm) and Correctness (RCr) are the key metrics used in the state-of-the-art benchmarks.

- **Additional Metrics.** This category includes additional metrics pertaining to the use of Multiple parallel Clients (MC) to assess the parallel querying capabilities of the triple store, and the Dataset Updates (DU).

The *MC* is the central metric related to our study, which is clearly missing in the majority of benchmark evaluations. Some basic evaluation is shown in BSBM and BioBench by multiple parallel querying clients. However, they did not report the peak performance points of tested triple stores.

Apart from the evaluations conducted in triple store benchmarks, additional performance evaluations can be found in the literature as well. Voigt et al. [20] evaluated triple stores for data loading time, query runtimes, and result set completeness. They aimed to test the systems for some specific type of queries like `SELECT` (with or without `UNION`, `REGEX`, `FILTER` or `sub-queries`). For the multi-client scenario, they measured the avg. query performance, as well as how many queries could be executed within a 10-minute time slot. In addition, some experiments related to memory requirements were conducted. Conrads et al. [13] presented a generic framework for benchmarking the read/write performance of triple stores in the presence of multiple querying agents. They evaluated three triple stores (Virtuoso, Fuseki and Blazegraph) for QpH and QMpH for

Table 1. Details of Benchmarks and type of experiments performed

Benchmarks	**Triple Stores**	**Experimental Details**
DBPSB[16]	Virtuoso Sesame Jena-TDB BigOWLIM	QpS and QMpH of all mentioned triple stores were evaluated. These triple stores were loaded with real-world DBpedia dataset and one querying agent (user) at a time was used.
FEASIBLE[7]	Virtuoso OWLIM-SE Jena-TDB (Fuseki) Sesame	QpS, QmpH and performance metrics relating to result set correctness and completeness were evaluated. Two datasets, i.e., real-world DBpedia and synthetic WatDiv, were used. Only one querying agent was used at a time.
WatDiv[8]	4Store RDF-3X MonetDB Virtuoso	Query execution time for synthetic datasets of different sizes was measured for only one querying user at a time. The experiments aim to compare triple stores by using synthetically generated data.
FishMark[17]	Virtuoso Quest	QpS for all the selected triple stores was evaluated against one querying user at a time. A synthetic dataset was used in this benchmark.
Bowlogna[18]	RDF-3X 4Store Virtuoso Diplodocus	Performance metrics relating to storage, i.e., RDF data loading time and the index size of all triple stores were evaluated. A synthetic dataset relating to the university data was used.
TrainBench[19]	RDF4J Jena-TDB	All mentioned triple stores were loaded with synthetic datasets of different sizes. After that, they were evaluated for result size completeness and correctness.
BioBench [12]	OWLIM-SE Virtuoso Bigdata Mulgara 4Store	Performance metrics relating to load time, storage space, and result sets were evaluated for single and multi users. However, the triple stores were not evaluated for query processing.
BSBM [11]	Sesame Jena-TDB Jena-SDB Virtuoso	The mentioned triple stores were loaded with synthetic datasets of different sizes and were evaluated for QpS, QMpH, and some other metrics related to data storage and result sets.
SP2Bench[10]	Sesame Virtuoso ARQ Redland	The selected triple stores were evaluated for processing overhead, storage and result set related performance metrics. Synthetic datasets were used and only one querying user was used.

different dataset sizes (DBpedia and SWDF[10]). Rohloff et al. [21] evaluated some triple store technologies, such as MySQL,[11] DAML DB[12] and BigOWLIM[13] (currently called GraphDB), in combination with RDF4J[14] and Jena,[15] as query frameworks for data loading time and query response time, by changing the dataset sizes. Stegmaier et al. [21] performed an evaluation on some of the RDF database technologies, including RDF4J,[16]

[10]Semantic Web Dog Food
[11]http://www.mysql.com/
[12]http://www.daml.org/2001/09/damldb/
[13]https://www.ontotext.com/products/graphdb/
[14]http://www.openrdf.org/
[15]https://jena.apache.org/
[16]http://www.openrdf.org/

AllegroGraph [22], and Jena-SDB[17] for their query execution time by using the SP2 [10] benchmark. Cudré-Mauroux et al. [23] empirically evaluated the NoSQL databases for RDF. Their evaluation is based on a comparison of several NoSQL stores, along with a native triple store, i.e., 4Store [24] for RDF processing. Furthermore, Verborgh et al. [25] evaluate their query engine named `Triple Pattern Fragments (TPF)` based on performance metrics *Number of Timeouts*, *Query Execution Time*, and *Network Usage*. Similarly, Minier et al. in S_AG_E [26] perform evaluations based on avg. workload completion time for 50 clients and compare their system with brTPF [27], TPF [25] and Virtuoso [6]. Finally, Azzam et al. [28] compare `SMART-KG` with TPF, Virtuoso and S_AG_E by using performance metrics *Number of Timeouts*, *Execution Time* and *Resource Consumption*.

However, to the best of our knowledge, none of these additional evaluations tested the performance of triple stores for their maximum throughput during parallel querying workload.

3. Evaluation Setup

In this section, we explain the evaluation setup used in the experiments. In general, any evaluation related to RDF systems comprises an RDF dataset, a collection of SPARQL queries, and a set of performance metrics. Here, we present key features of each of these components that are important to consider for fair evaluation. Many of these features come from state-of-the-art research contributions mentioned in [7,15,8].

Benchmarks. Benchmarks for the evaluation of triple stores can either be synthetic or real-data [15]. The synthetic-data benchmarks make use of a data generator to generate synthetic data. Queries can be generated by using query templates on the underlying data. Synthetic-data benchmarks are useful in testing the scalability of triple stores with varying dataset sizes. However, they often fail to reflect characteristics of the real-world queries posted by users of the datasets in practice [15,5,29]. On the other hand, real-data benchmarks contain both data and queries, selected from real-world RDF datasets and their corresponding query logs. Such benchmarks more closely reflect the real-world deployment of triple stores. However, analysis of real-world queries [5,29] show that they are quite simple in terms of the structural features (number of triple patterns, types of joins, projections, etc.) and data-driven features (result sizes, selectivity, etc.) of SPARQL queries [15,8]. Keeping in mind the pros and cons of both types of benchmarks, we consider both real-world as well as synthetic benchmarks in our evaluation:

- **FEASIBLE[7]**: is a real-data benchmark generator, which generates benchmarks by using the real-world query logs of RDF datasets. We used the same benchmark (analyzed in [15]) that was generated by the FEASIBLE framework. This benchmark is based on the DBpedia3.5.1 dataset.[18] The dataset contains a total of 232M (English version) triples, 18,425k distinct subjects, 39,672 predicates, and 65,184k objects. The benchmark includes a total of 50 real queries selected from the DBpedia3.5.1 SPARQL endpoint log. These queries cover most of the required structural and data-driven features of the SPARQL queries [15]. Furthermore, it is the

[17]https://jena.apache.org/documentation/sdb/
[18]DBpedia3.5.1: dbpedia.org

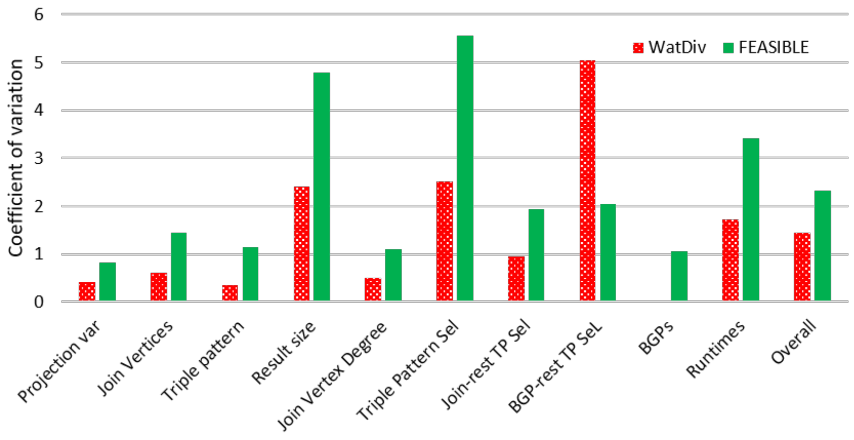


Figure 1. Diversity scores across different SPARQL query features of the benchmarks.

Table 2. Coverage of SPARQL clauses and join vertex types used in the benchmarks in percentages. SPARQL clauses: `DIST[INCT]`, `FILT[ER]`, `REG[EX]`, `OPT[IONAL]`, `UN[ION]`, `LIM[IT]`, `ORD[ER BY]`. Join vertex types: Star, Path, Sink, Hyb[rid], N[o] J[oin].

	Distributions of SPARQL Clauses							Distr. of Join Vertex Type				
Benchmark	`DIST`	`FILT`	`REG`	`OPT`	`UN`	`LIM`	`ORD`	Star	Path	Sink	Hyb.	N.J.
Watdiv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.0	64.0	26.0	20.0	0.0
FEASIBLE	56.0	58.0	22.0	28.0	40.0	42.0	32.0	58.0	18.0	36.0	16.0	30.0

most diverse benchmark in comparison to other triple store benchmarks included in [15].

- **WatDiv[8]**: is a synthetic benchmark generator. Again, we used the same benchmark analyzed in [15] that was generated by WatDiv generators having 108M triples, usually called 100M WatDiv dataset. Similarly, for more diverse evaluation and to test the scalability of the triple stores w.r.t. varying dataset sizes, we considered two more datasets generated by the same benchmark having 10M and one billion triples. The total number of query templates used in benchmarks is 50, including 20 basic testing query templates and 30 extensions to basic testing. The basic testing consists of queries in four categories, namely, linear queries (L), star queries (S), snowflake-shaped queries (F) and complex queries (C) [8].

The coefficient of variation, which shows diversity scores [15] across different SPARQL query features, is shown in Fig. 1. The coverage of different SPARQL clauses and join vertex types is shown in Table 2. Further detailed analysis of the datasets as well as queries about the selected benchmarks can be found in [15]. Please note that these are the two most diverse benchmarks according to the benchmarks analysis conducted in [15].

Performance Metric. Since we are measuring the throughput of triple stores, we use Queries per Second (QpS) as the main performance indicator.

Triple Stores. We selected triple stores to be included in the evaluation based on the following criteria: (1) the triple stores should be available for free, therefore we excluded commercial triple stores, (2) they should be able to load and process both the selected datasets and the corresponding queries, (3) they should offer SPARQL HTTP endpoints, (4) they should support the SPARQL features included in the FEASIBLE benchmark, therefore triple stores which only support BGP[19] queries are excluded, and (5) they should have no benchmarking restrictions, e.g., the maintainers had to approve the inclusion of their system results in the publication to the public.

Based on the above criteria we have considered the following triple stores in our evaluation[20]:

1. **Virtuoso**[21] is flexible enough to configure most of its parameters through config file. We used Virtuoso version 7.2.6 with `NumberOfBuffers=680000` and `MaxDirtyBufferes=500000`, which is recommended settings for `8 GB` of free system memory. The parameter `MaxClientConnections` in the HTTP Server section, is set according to the number of querying users (i.e., one connection per querying user) for all the individual experiments.
   The `ThreadsPerQuery = 32` is set according to the number of CPU cores.

2. **Jena TDB**[22] Version 3.13.1 with **Fuseki** as HTTP interface with Java heap size set to `8g`. The documentation[23] about parallelism shows that Jena's query mechanism is itself multi-threaded, and it supports parallel querying by default.

3. **Blazegraph**[24] Version 2.1.4, with Jetty as HTTP interface and Java heap size set to `8g`. Through its configuration file, we changed the `QueryThreadPoolSize=32` and `ReadOnly=True`. All other parameters were kept default.

4. **Ontotext GraphDB**[25] with `Java heap=8g`.

5. **Parliament** [30] with `MIN_MEM=1g` and `MAX_MEM=8g` of Java heap and jetty as HTTP interface.

In some cases, we also contacted the maintainers of the systems to get the recommended and comparable settings.

Benchmark Execution. All the experiments were performed using the benchmark execution framework Iguana V2.1.2 [13], which is particularly developed to measure the read/write performance of RDF triple stores in the presence of multiple querying agents. As recommended by the maintainer, we set the query time-out to 10 minutes per query, and each experiment was performed in a stress test with 60 minutes run time. All the

[19]https://www.w3.org/TR/rdf-sparql-query/#BasicGraphPatterns
[20]We tried our best to test the selected triple stores with matching configuration settings.
[21]Virtuoso:https://virtuoso.openlinksw.com/
[22]Jena TDB: https://jena.apache.org/documentation/tdb/
[23]https://jena.apache.org/documentation/notes/concurrency-howto.html
[24]Blazegraph: https://blazegraph.com/
[25]GraphDB: http://graphdb.ontotext.com/

experiments were performed for 1, 2, 4, 8, 16, 32, 64 and 128 concurrently executing clients. Before starting the evaluation, we bulk loaded each of the datasets into the triple stores. During each run of the experiment, the triple stores contained only the dataset upon which the benchmarking was being carried out. Moreover, we tested the selected triple stores up to 128 concurrent clients to ensure the service unavailability.

Hardware. All experiments were performed on a machine with two Intel Xeon E5-2620 v4 CPUs having each 8 physical cores and 16 logical cores, 256GB RAM, 11TB HDD and running Ubuntu 20.04.2 LTS.

4. Results and Discussion

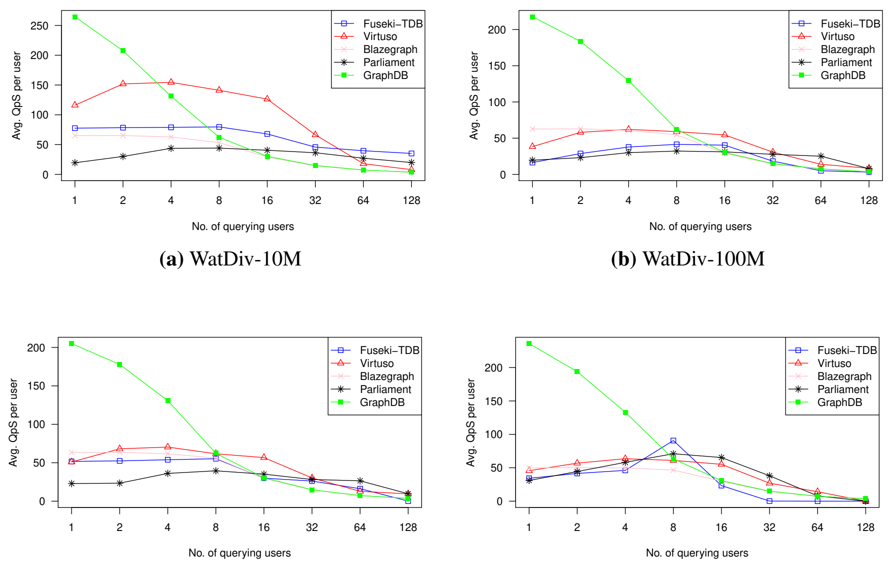
Fig. 2d shows the results when all the triple stores are loaded with the DBpedia3.5.1 dataset and FEASIBLE [7] benchmark queries were executed on them. Similarly, Fig. 2a, 2b, and 2c show the WatDiv benchmarking results when the triple stores are tested with 10 million, 100 million and one billion triples datasets, respectively. From these graphs, we want to look for the key findings pertaining to the following research questions: (1) Which triple store achieved the highest throughput in terms of QpS? (2) On avg., which triple store is performing the best? (3) What is the peak performance point of each of the selected triple stores and when is it achieved? (4) How do the triple stores scale to the increasing number of parallel querying agents? (5) At which point does the DoS occur? and (6) How do systems scale with the increasing dataset sizes? In the following, we discuss each of these key questions.[26]

Highest Throughput: Fig. 4a shows that GraphDB achieves the highest peak performance point, i.e., 231 on avg., as well as in the case of all the individual benchmarks (ref. Figure 2). Followed by GraphDB, Virtuoso achieves the second position by achieving maximum avg. throughput of 88 QpS, and in the case of WatDiv-10-Million benchmark, it has the highest individual QpS value as shown in Fig. 3. Then Fuseki-TDB, Blazegraph and Parliament achieve the 3rd, 4th and 5th position, respectively. Finally, Blazegraph achieves almost the same maximum QpS in all WatDiv benchmarks.

Average Throughput: The avg. throughput of the selected triple stores can be measured by calculating the area under the curve in the corresponding throughput graphs. The higher the area covered, the higher the avg. throughput. Fig. 4b shows that Virtuoso achieves the maximum avg. throughput of 3621. It is followed by Parliament having 3101, then Fuseki-TDB having 2727, followed by GraphDB with 2364 and then Blazegraph with 1775. From the Fig. 3 and 4b, it can be observed that the maximum peak performance in terms of QpS, does not necessarily mean that the same system will perform well in terms of avg. throughput.

Peak Performance Points: The results in Fig. 2 show that there is a peak performance point for each triple store. This peak point of any triple store differs for all the benchmarks, but is reached during the same number of querying agents. Once that point is

[26]Please note that the aim of this paper is to report the triple stores performances with different stress testing. The reason why one triple store performs better than others is out of the scope of this paper.



(a) WatDiv-10M **(b)** WatDiv-100M **(c)** WatDiv-1B **(d)** FEASIBLE (DBpedia)

Figure 2. Benchmark results on (a), (b), (c) and (d) for all the triple stores - For each benchmark, the x-axis shows the No. of querying users while y-axis shows the average Queries-per-Second per one user.

reached, further increase in the querying workload leads to gradual decrease in performance.

Parallel Scalability: It refers to how triple stores react to the increasing querying agents. A highly parallel scalable triple store's throughput would gradually increase with the increasing number of multiple querying agents. We can see from Fig. 5f, that this is not the case for the majority of selected triple stores, i.e., the peak performance point of these triple stores is reached quite early. In this regard, the parallel scalable triple store ranking is: Fuseki-TDB and Parliament are scalable up to eight querying agents, followed by Virtuoso with four, Blazegraph with two, and GraphDB with only one. It is worth mentioning here that Parliament achieve the least avg. peak performance but is scalable in terms of the maximum number of querying agents it supports.

Denial of Service (DoS): Our results show that the throughput of the selected triple stores almost reaches zero when exposed to 128 querying agents. This is the point at which triple stores almost stop responding.

Scalability with Increasing Dataset: Finally, we want to measure the scalability of the selected triple stores in terms of varying datasets sizes as well as increasing querying agents. Fig. 5a, 5b, 5c, 5d and 5e show the corresponding results for each of the selected triple stores. We can clearly see that, in general, performance is decreased with the increase in the number of agents as well as the size of dataset. These results are as expected because increasing the workload or the dataset size will lead to more processing work to be performed by the triple stores to get the desired query results. However, a sub ques-

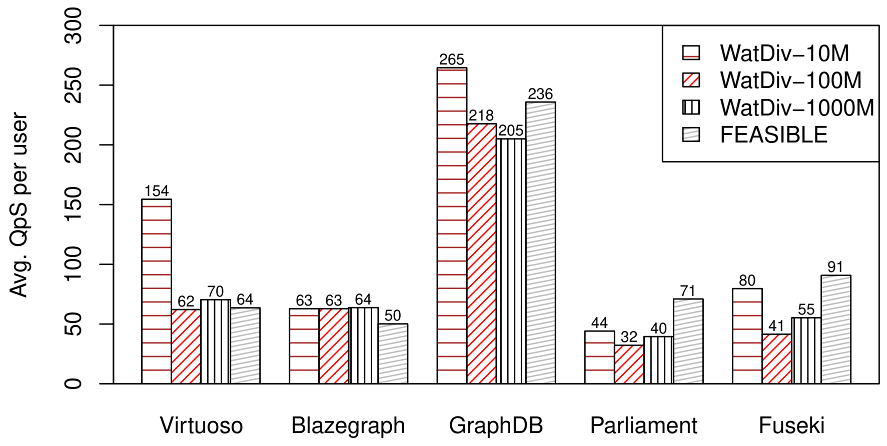
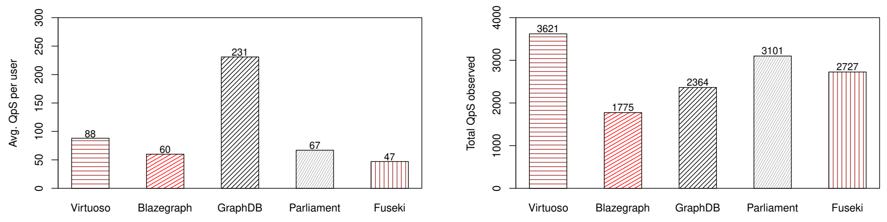


Figure 3. Peak throughput in terms of avg. Queries-per-Second per user of all given triple stores with different benchmarks.

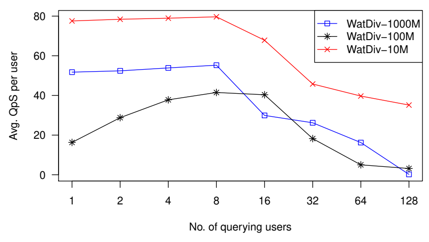


(a) Peak performance point **(b)** Total throughput

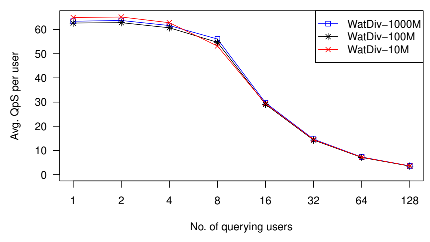
Figure 4. (a) shows the peak performance point in terms of the avg. QpS per user of all the benchmarks, while (b) shows the total throughput of the systems (triple stores) in terms of the area covered under the curve of avg. QpS of all benchmarks.

tion to be investigated is that which triple store scale better with increasing dataset size? Fig 5b shows that the throughput of Blazegraph is not much affected by increasing the size of the dataset. It is followed by GraphDB (ref. Figure 5e) with little effect on the varying dataset sizes. On the other hand, we can clearly see a short performance drop on the other three selected triple stores. In particular, the performance of Virtuoso is greatly affected by the dataset sizes. In conclusion, the results suggest that Blazegraph is the most scalable triple store to handle big data with smaller effect on the throughput.

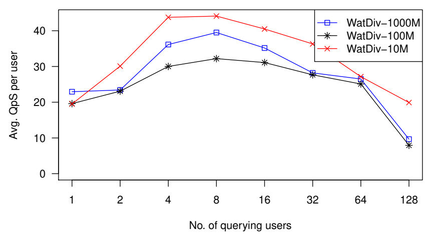
In summary, our results reveal the parallel query processing capabilities of selected triple stores. In particular, there exists a peak performance point for each of these triple stores which is generally reached with only a small number of multiple querying agents, as shown in Fig. 5f. Hence, these triple stores can easily lead to performance degradation or even a service shutdown when they are exposed to multiple querying users.



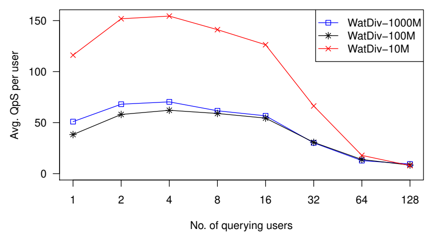
(a) Fuseki-TDB



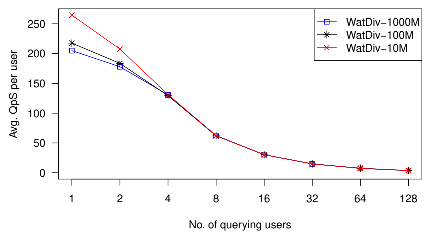
(b) Blazegraph



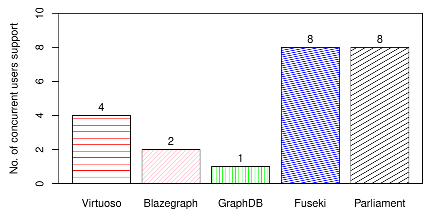
(c) Parliament



(d) Virtuoso



(e) GraphDB



(f) Users support

Figure 5. Benchmark results on WatDiv-10-Million (a), WatDiv-100-Million (b), WatDiv-One-Billion (c) and DBpedia (d); For each benchmark, x-axis shows the No. of querying users while y-axis shows the avg. QpS per user. (f) shows the No. of users that triple stores support concurrently with highest throughput.

5. Resource Availability and Reusability

The datasets and queries used in this work are based on state-of-the-art benchmarks [7,8]. The query execution was performed by using the IGUANA [13] benchmark execution framework. All data required to reproduce these experiments or conduct a new set of experiments are available from the aforementioned repository homepage. Since we used standard state-of-the-art benchmarks and a standard benchmark execution framework, new triple store developers can use the same setup to test their own triple stores and compare with the state of the art. Finally, we also provide the complete evaluation results to enable a more fine-grained analysis. The current queries used in the FEASIBLE-DBpedia benchmark were selected from the query log of the DBpedia version 3.5.1. However, new queries for other versions of DBpedia are now available from the LSQ [5] dataset, which can be directly consumed by the FEASIBLE benchmark generation framework. In the future, we will provide more FEASIBLE-DBpedia benchmarks for the newer versions of DBpedia from the same resource home page. This will ensure triple store testing for their scalability with respect to varying sizes of DBpedia.

6. Conclusion

State-of-the-art linked data querying interfaces face the problem of finding a reasonable solution for the trade-off between performance and availability of RDF triple stores. Serving requests with high efficiency, and at the same time ensuring high availability of the endpoints, is crucial for the success of the Semantic Web. We conducted experiments with the aim of facilitating the design of smart query processing interfaces that ensure both high performance and availability. In particular, we showed the peak performance points and the parallel query processing capabilities of selected triple stores. Furthermore, we showed the extreme workloads that lead to potential service shutdowns on these triple stores. Finally, we measured the effect of varying dataset sizes on the query runtime performances of the selected triple stores. In the future, we want to include more triple stores and measure the effect of the resources (allocated RAM memory in particular) on the performance.

Acknowledgments

This work has been supported by the project 3DFed (Grant no. 01QE2114B), KnowGraphs (no. 860801), and DAIKIRI (no. 01IS19085B). In addition, the Higher Education Commission (HEC) of Pakistan.

References

[1] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck and P. Colpaert, Triple pattern fragments: a low-cost knowledge graph interface for the web, *JOURNAL OF WEB SEMANTICS* **37-38** (2016), 184–206. http://dx.doi.org/10.1016/j.websem.2016.03.003.

[2] H. Khan, Towards More Intelligent SPARQL Querying Interfaces, in: *International Semantic Web Conference*, 2019. http://ceur-ws.org/Vol-2548/paper-12.pdf.

[3] C. Buil-Aranda, A. Hogan, J. Umbrich and P.-Y. Vandenbussche, SPARQL Web-Querying Infrastructure: Ready for Action?, in: *The Semantic Web – ISWC 2013*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 277–293. ISBN ISBN 978-3-642-41338-4.

[4] P.-Y. Vandenbussche, J. Umbrich, L. Matteis, A. Hogan and C. Buil-Aranda, SPARQLES: Monitoring public SPARQL endpoints, *Semantic Web* **8** (2017), 1–17. doi:10.3233/SW-170254.

[5] M. Saleem, M.I. Ali, A. Hogan, Q. Mehmood and A.N. Ngomo, LSQ: The Linked SPARQL Queries Dataset, in: *ISWC*, Springer, 2015, pp. 261–269.

[6] O. Erling and I. Mikhailov, *Virtuoso: RDF Support in a Native RDBMS*, in: *Semantic Web Information Management: A Model-Based Perspective*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 501–519.

[7] M. Saleem, Q. Mehmood and A.N. Ngomo, FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework, in: *ISWC*, Springer, 2015, pp. 52–69.

[8] G. Aluç, O. Hartig, M.T. Özsu and K. Daudjee, Diversified Stress Testing of RDF Data Management Systems, in: *The Semantic Web – ISWC 2014*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz and C. Goble, eds, 2014, pp. 197–212. ISBN ISBN 978-3-319-11964-9.

[9] Y. Guo, Z. Pan and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *J. Web Sem.* **3**(2–3) (2005), 158–182. doi:10.1016/j.websem.2005.06.005.

[10] M. Schmidt, T. Hornung, M. Meier, C. Pinkel and G. Lausen, *SP2Bench: A SPARQL Performance Benchmark*, in: *Semantic Web Information Management: A Model-Based Perspective*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 371–393.

[11] C. Bizer and A. Schultz, The Berlin SPARQL Benchmark, *Int. J. Semantic Web Inf. Syst.* **5**(2) (2009), 1–24. doi:10.4018/jswis.2009040101.

[12] H. Wu et al., BioBenchmark Toyama 2012: An evaluation of the performance of triple stores on biological data, *J. Biomedical Semantics* **5** (2014), 32. doi:10.1186/2041-1480-5-32.

[13] F. Conrads, J. Lehmann, M. Saleem, M. Morsey and A.N. Ngomo, IGUANA: A Generic Framework for Benchmarking the Read-Write Performance of Triple Stores, in: *ISWC*, Springer, 2017, pp. 48–65.

[14] W. Ali, M. Saleem, B. Yao, A. Hogan and A.-C.N. Ngomo, A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs, 2021.

[15] M. Saleem, G. Szárnyas, F. Conrads, S.A.C. Bukhari, Q. Mehmood and A.-C. Ngonga Ngomo, How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks, in: *The World Wide Web Conference*, WWW '19, ACM, New York, NY, USA, 2019, pp. 1623–1633. ISBN ISBN 978-1-4503-6674-8. doi:10.1145/3308558.3313556.

[16] M. Morsey, J. Lehmann, S. Auer and A.N. Ngomo, DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data, in: *ISWC*, 2011, pp. 454–469.

[17] S. Bail, S. Alkiviadous, B. Parsia, D. Workman, M. van Harmelen, R.S. Gonçalves and C. Garilao, FishMark: A Linked Data Application Benchmark, in: *Proceedings of the Joint Workshop on Scalable and High-Performance Semantic Web Systems*, 2012, pp. 1–15. http://ceur-ws.org/Vol-943/SSWS_HPCSW2012_paper1.pdf.

[18] G. Demartini, I. Enchev, M. Wylot, J. Gapany and P. Cudré-Mauroux, BowlognaBench - Benchmarking RDF Analytics, in: *Data-Driven Process Discovery and Analysis SIMPDA*, Springer, 2011, pp. 82–102.

[19] G. Szárnyas, B. Izsó, I. Ráth and D. Varró, The Train Benchmark: cross-technology performance evaluation of continuous model queries, *Software and systems modeling* **17**(4) (2018), 1365—1393–. doi:10.1007/s10270-016-0571-8. https://europepmc.org/articles/PMC6132656.

[20] M. Voigt, A. Mitschick and J. Schulz, Yet Another Triple Store Benchmark? Practical Experiences with Real-World Data, in: *SDA*, 2012.

[21] F. Stegmaier, U. Gröbner, M. Döller and H. Kosch, Evaluation of Current RDF Database Solutions.

[22] P. Tajabor and T. Raafat, Challenges Over Two Semantic Repositories - OWLIM and AllegroGraph, *Indonesian Journal of Electrical Engineering and Computer Science* **2** (2016), 194. doi:10.11591/ijeecs.v2.i1.pp194-204.

[23] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, F.L. Keppmann, D. Miranker, J.F. Sequeda and M. Wylot, NoSQL Databases for RDF: An Empirical Evaluation, in: *The Semantic Web – ISWC 2013*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 310–325.

[24] S. Harris, N. Lamb, N. Shadbolt and G. Ltd, 4store: The design and implementation of a clustered RDF store, *Proc. SSWS* (2009).

[25] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck and P. Colpaert, Triple Pattern Fragments: A low-cost knowledge graph interface for the Web, *Journal of Web Semantics* **37-38** (2016), 184–206. doi:https://doi.org/10.1016/j.websem.2016.03.003. https://www.sciencedirect.com/science/article/pii/S1570826816000214.

[26] T. Minier, H. Skaf-Molli and P. Molli, SaGe: Web Preemption for Public SPARQL Query Services, in: *The World Wide Web Conference*, WWW '19, ACM, New York, NY, USA, 2019, pp. 1268–1278. ISBN ISBN 978-1-4503-6674-8. doi:10.1145/3308558.3313652.

[27] O. Hartig and C.B. Aranda, brTPF: Bindings-Restricted Triple Pattern Fragments (Extended Preprint), *CoRR* (2016). http://arxiv.org/abs/1608.08148.

[28] A. Azzam, J.D. Fernandez, M. Acosta, M. Beno and A. Polleres, SMART-KG: Hybrid Shipping for SPARQL Querying on the Web, in: *Proceedings of The 2020 World Wide Web Conference, WWW 2020. (To appear)*, 2020.

[29] M. Arias, J.D. Fernández, M.A. Martínez-Prieto and P. de la Fuente, An Empirical Study of Real-World SPARQL Queries, *CoRR* **abs/1103.5043** (2011). http://arxiv.org/abs/1103.5043.

[30] D. Kolas, I. Emmons and M. Dean, Efficient Linked-List RDF Indexing in Parliament.