IGUANA: A Generic Framework for Benchmarking the Read-Write Performance of Triple Stores

Felix Conrads[1], Jens Lehmann[2], Muhammad Saleem[1]
, Mohamed Morsey[3], and Axel-Cyrille Ngonga Ngomo[1,4]

[1] University of Leipzig, AKSW, Germany
email: {lastname}@informatik.uni-leipzig.de
[2] University of Bonn and Fraunhofer IAIS
email: jens.lehmann@cs.uni-bonn.de, jens.lehmann@iais.fraunhofer.de
[3] System and Network Engineering Group, University of Amsterdam
email: m.morsey@uva.nl
[4] Department of Computer Science, University of Paderborn

Type: Benchmarking paper
Permanent URL: http://github.com/AKSW/Iguana
Permanent URL for results: https://figshare.com/collections/Iguana_-_Benchmark_2016/3767501/1
Website and documentation: http://iguana-benchmark.eu/

Abstract. The performance of triples stores is crucial for applications driven by RDF. Several benchmarks have been proposed that assess the performance of triple stores. However, no integrated benchmark-independent execution framework for these benchmarks has yet been provided. We propose a novel SPARQL benchmark execution framework called IGUANA. Our framework complements benchmarks by providing an execution environment which can measure the performance of triple stores during data loading, data updates as well as under different loads and parallel requests. Moreover, it allows a uniform comparison of results on different benchmarks. We execute the FEASIBLE and DBPSB benchmarks using the IGUANA framework and measure the performance of popular triple stores under updates and parallel user requests. We compare our results [5] with state-of-the-art benchmarking results and show that our benchmark execution framework can unveil new insights pertaining to the performance of triple stores.

Keywords: Benchmarking, Triple Stores, SPARQL, RDF, Log Analysis

1 Introduction

The size of the Linked Open Data cloud has grown considerably over the last decade. We are now faced with a compendium of more than 10,000 data sets and more than 150 billion triples.[6] These data sets cover domains as diverse as geography, media and life sciences. Many Linked Data applications that consume and manipulate data rely on

[5] See https://doi.org/10.6084/m9.figshare.c.3767501.v1
[6] http://lodstats.aksw.org

triple stores for persisting data, which hold one or more of these data sets [10,7,11]. It is thus evident that the performance of triple stores plays a vital role for the deployment and use of Linked-Data-driven applications. This leads to the need for robust benchmarking, which is (1) able to pinpoint the strengths and weaknesses of the triple store under test. This in turn allows, (2) the evaluation of the suitability of a specific triple store to the application under which it is to operate, and the proposal of the best candidate triple stores for that application. In addition, benchmarking triple stores can also help (3) identifying the best running conditions for each triple store (e.g., the best memory configuration) as well as (4) providing developers with insights pertaining to how to improve their frameworks.

While many benchmarks (e.g., [1,2,5,7,14,11]) have resulted from these considerations, the comparability of benchmarking results remains problematic as each benchmark usually provides its own execution environment, thus making the results across different evaluations difficult if not impossible to compare. For example, while some of the benchmarks above provide dedicated execution scripts, these cannot always be ported easily to other benchmarks. In this replication and benchmark paper, we address this gap by proposing a novel SPARQL benchmark execution framework called IGUANA (Integrated Suite for Benchmarking SPARQL). IGUANA is a *benchmarking suite* that takes a benchmark, a dataset and possible updates as input. The suite is able to test the behavior of triple stores in a holistic manner, i.e., it can test for load times as well as concurrent query execution and data updates with different user configurations. The suite is thus *complementary to benchmarks* (which most commonly provide data or queries) and can execute both synthetic benchmarks and benchmarks based on real data and real queries to give more complete insights into the behavior of a triple store. Note that thanks to its flexible configuration of benchmark execution, the suite allows the assessment of endpoints which serve multiple agents (users, software systems, etc.) of different types (e.g., query and update) concurrently (e.g., the DBpedia endpoint with approximately 860k queries per day). The methodology implemented by IGUANA follows the four key requirements for domain-specific benchmarks that are postulated in the Benchmark Handbook [4], i.e., it is

1. *relevant*, as it allows the testing of typical operations within the specific domain,
2. *portable* as it can be executed on different platforms and using different benchmarks and datasets,
3. *scalable* as it is possible to run benchmarks on both small and large data sets with variable rates of updates and concurrent users and
4. *understandable* as it returns results using standard measures that have been used across literature for more than a decade.

Our contributions are as follows:

- We present the first (to the best of our knowledge) integrated and extensible benchmarks execution suite for SPARQL that can uniformly execute state-of-the-art triple store benchmarks under realistic loads such as concurrent requests and updates.
- We provide the first (to the best of our knowledge) realistic evaluation of triple stores with concurrent queries and updates. We evaluate commonly used triple stores under real loads from DBpedia Live and Semantic Web Dog Food (SWDF) and present novel insights pertaining to their behavior.

– As an example showcase, we integrate FEASIBLE [11] and DBPSB [6] real SPARQL benchmarks generators and evaluate state-of-the-art triple stores on four datasets.
– Our results show that while the triple stores we evaluated seem to scale up well to concurrent queries and updates, they are all affected significantly by the size of the datasets.

This paper is organized as follows. We begin by presenting the core of IGUANA. We then present an evaluation of state-of-the-art triple stores on standard server hardware under different loads. Thereafter, we give an overview of the state of the art in benchmarking triple store. Finally, we detail future work and conclude. The code can be found at `https://github.com/AKSW/IGUANA` Links to all information pertaining to IGUANA (including its source code, GPLv3) can be found at `http://iguana-benchmark.eu`. A guide on how to get started is at `http://iguana-benchmark.eu/gettingstarted.html`.

2 The IGUANA Framework

This section describes IGUANA. We begin by presenting the components of the framework. We then present the main necessary and optional parameters through which it can be configured. Finally, we give an overview of the core functionality of the framework.

2.1 Overview

Figure 1 shows the core components of the IGUANA framework. The input for the framework is a *configuration file* (short: config file), which contains (1) the configuration parameters (see Section 2.2), (2) instructions that orchestrate how queries are to be processed and issued as well as (3) a specification of the benchmarking process and (4) the external data sources to be used during this process. A representation of the parsed config file is stored internally in a *configuration object* (short: config object). If the config object points to a query log, then the *analyzer processor* analyzes this query log file and generates benchmark queries (e.g., using the FEASIBLE [11] approach). IGUANA also supports the benchmark queries being directly provided to the framework. The *dataset generator process* creates a fraction (e.g., 10% of DBpedia) of dataset, thus enabling it to test the scalability of the triple stores with varying sizes of the same dataset. Note that this generator is an interface which can be used to integrate data generators (e.g., the DBPSB [7] generator) that can create datasets of varying size. The *warmup processor* allows the execution of a set of test queries before the start of the actual stress testing. The *testcase processor* then performs the benchmarking by means of stress tests according to the parameters specified in the config file. Finally, the *result processor* generates the results which can be emailed by the *email processor*. In the following, we describe the core components of IGUANA in more detail.

2.2 Input Parameters

The IGUANA framework (see Figure 1 for an overview) requires the following input parameters:

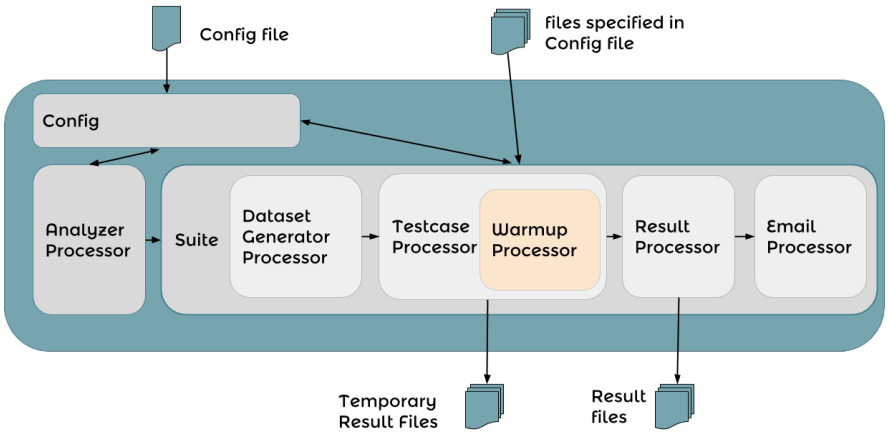


Fig. 1: Overview of the IGUANA benchmarking components.

1. An *input dataset* (necessary). This is the dataset upon which the SPARQL queries are to be executed. The dataset is part of the input because the framework also measures the time necessary for triple stores to load data.
2. *A set of change sets* (optional). The change sets are triples that are added or deleted from the triple store at runtime. In real applications, it is common to write and read from a triple store while queries are executed. This behavior is emulated by means of the change sets contained in this portion of the input.
3. *Benchmark queries to use* (necessary). This is the set of queries that are to be executed to assess the performance of the triple store to benchmark. Note that we support both query templates (see [7]) and query sets (see, e.g., [11]) as provided by most of the existing benchmarks.
4. *Number and type of workers* (necessary). IGUANA supports two main types of workers: Update workers perform SPARQL INSERT queries to inject new triples into a triple store. Query workers can perform SELECT, ASK, DESCRIBE and CONSTRUCT queries to gather information from the triple store. The workers are parametrized by the frequency at which they carry out queries. This frequency can either be static (e.g., every 500 ms) or abide by a statistical distribution such as a Gaussian (e.g., mean = 500 ms, standard deviation = 100 ms).
5. *Amount of data to load into the triple store* (optional). With this parameter, benchmarks on a fraction of a dataset (e.g., 10% of DBpedia) are made possible.
6. *Warmup parameters* (optional). An *optional set of warm up queries*, i.e., queries used by the systems that are to be benchmarked to fill the triple store caches, as well as an *optional warm up time* can be set.

Note that IGUANA provides means to define pre- and post-shell scripts for triple stores. This enables the suite to use bulk loading scripts provided by some triple stores as these scripts are often significantly more time-efficient than loading data via `INSERT` queries. However, the bulk loading strategies of triples stores are not standardized and

their execution thus had to be moved to pre-processing scripts. The post-processing scripts allow to clear triple stores (e.g., to delete global indexes) as requested.

2.3 Anatomy of a Test Case

Once IGUANA has been parametrized, the benchmarking can begin. IGUANA's benchmarking approach is built around the concept of *test cases*, which are basically runs of benchmarks. In essence, the core simply implements the methods and interfaces necessary to execute these test cases. First, the core *pre-processes* all the data necessary to carry out a given test case. The pre-processing begins with the gathering the endpoints which should be tested on particular datasets, the details about the configuration of the benchmark data as well the properties of the test case. Thereafter, a *reference connection* is created if necessary. This connection is used during the configuration of query templates into queries for the benchmark. The reference connection links the benchmark to an auxiliary data source that contains the same data as the triple store to benchmark. Note that this connection should not point to the triple store we aim to benchmark as the queries sent through the auxiliary connection could falsify the results at runtime. As an example, if we aim to benchmark a triple store containing DBpedia data, we can set `http://dbpedia.org/sparql` as the auxiliary connection. Note that the reference connection is only needed for the purpose of generating queries from query templates. Consequently, if IGUANA is provided with queries (not templates), it does not require this connection. IGUANA completes this first step by setting up all the data necessary to carry out the test case at hand. For example, it converts query patterns into complete SPARQL queries with the help of the auxiliary connection.

After the pre-processing stage, IGUANA tests every given dataset described in the config file. To this end, our framework runs the test cases in the order stated in the config file. For example, if the user declares a *pre-shell script*, IGUANA will firstly execute this script, therewith enabling users to configure the triple store to benchmark at will before the beginning of the benchmark. Typically (and in our evaluation as well), these scripts are used to stop and start the current triple store as well as copy a dataset dump to triple store. Note that IGUANA measures how long the pre-shell script takes to be executed. Hence, our framework can measure how long a framework needs for bulk loading data. For the sake of completeness, IGUANA provides the possibility to benchmark the upload via `SPARQL INSERT` queries. While this test is not recommended when loading a large amount of data, as most frameworks provide bulk loading scripts, IGUANA will check for the existence of upload tests, the corresponding dataset and upload it to the triple store that is currently being tested by means of `INSERT` queries.

IGUANA then starts the *warm-up phase* (if the user defined one), during which a set of user-defined SPARQL queries and updates are sent to the triple store for a predefined period of time (default = 20 min). After the warm-up phase has been completed, the stress test begins (see Section 3). The completion of the stress test also marks the completion of the test case. The core gathers the results and adds them to the results which were gathered in previous steps of the test case (if any). If the user defined a *post-shell script* (e.g., to free resources on the server, backup data, clean out a dump, send a message marking the end of the stress test) the core will executes it and proceed

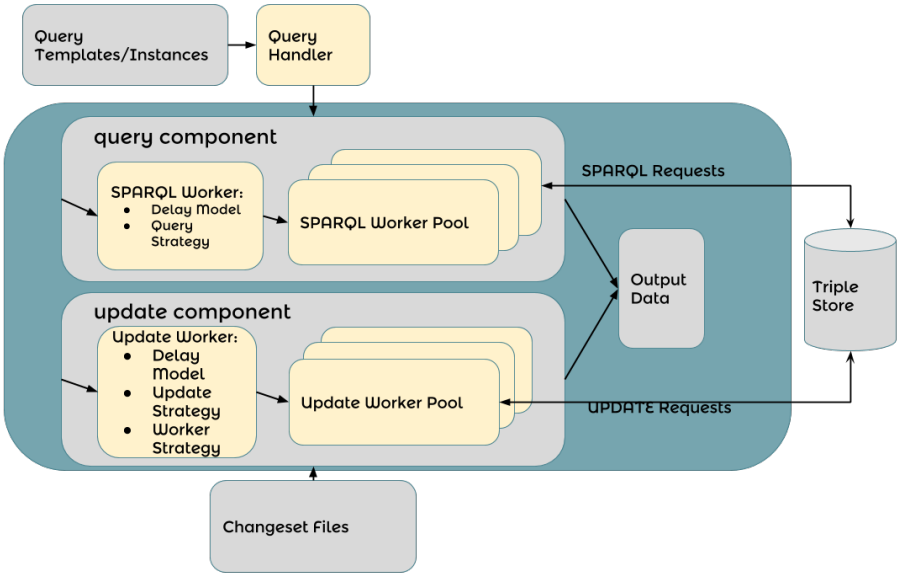


Fig. 2: Overview of the IGUANA Stresstest

to the next testcase. Once every dataset has been tested with every triple store and every test case, IGUANA saves the final results and exits.

3 Anatomy of a Stress Test

The main objective of this strategy is to simulate the operation of a live triple store which is updated continuously, while many users send queries at the same time. IGUANA's default stress test aims to simulate such real workloads of triple stores. Hence, it implements a situation in which several users belonging concurrently access the triple store. In general, there are two types of users querying the triple store: The *first type of user* queries the triple store using SELECT, ASK, DESCRIBE or CONSTRUCT queries while the *second type* updates the same triple store by sending INSERT queries. Consequently, our default stress test consists of two main components, namely the *query component*, and the *update component* as shown in Figure 2. The query component issues all queries from the first type of users to the triple store while the update component inserts and deletes triples from the test triple store.

3.1 Query Component

To initialize the query component, the stress test needs query templates or queries that are to be posed to the system. Given that having static queries is a special case of having query templates, we describe how the system deals with query templates. Query

templates are modelled as parameterized SPARQL queries (i.e., ASK, SELECT, CONSTRUCT or DESCRIBE queries) which can contain several template variables. Each template variable abides by the syntax %%v[0-9]*%%. For example a template can look as follows:

```
PREFIX dct:<http://purl.org/dc/terms/>
SELECT ?s ?p FROM <http://dbpedia.org>
WHERE
{   ?s ?p %%v1%% .
    %%v1%% dct:creator %%v2%% }
LIMIT 10
```

The template variables serve as slots to be replaced by resources, literals or blank nodes that lead to a query that can be executed on the data contained in the triple store to benchmark. The large number of valid queries that can be generated from such a template attenuates, if not circumvents, the effect of naïve caching in the triple stores to evaluate. The maximum number of instances of each template generated out of the data is set by the user. Note here that it can happen that the user requires more instances than query solutions available in the triple store, in which case IGUANA selects all solutions to generate query instances. Instead of querying the triple store to benchmark for relevant replacements of the variables, the *query component* uses the *reference connection* (see Section 2.3) provided by the user to obtain valid replacements for the variables. In the example, it will try to get instances for %%v1%% and %%v2%%. To achieve this goal, the query generator transforms the input query to the following:

```
    PREFIX dct:<http://purl.org/dc/terms/>
    SELECT ?v1 ?v2 FROM <http://dbpedia.org>
    WHERE
    {?s ?p ?v1 .
    ?v1 dct:creator ?v2}
    LIMIT K
```

where K is the number of queries per template set by the user.

This query is now sent to the reference connection. The results are stored in a table of key-value pairs for each of the query templates. With this approach, we ensure that the instantiations of the template that we generate return (non-empty) results. In a final step, the variables in the query templates are replaced with all key-value pairs in the table and the results are stored in one reference file per query. This whole process is carried out once during the complete test case. The approach ensures that all triple stores that are to be compared are confronted with exactly the same queries.

Now that the queries to be sent to the triple store are available, a *pool of query workers* is created. For the given test case, the workers are assigned a seed number. This number is used to seed a pseudo-random number generator. This generator then computes the index of the query template as well as the index of the instantiation of the said query template that is to be used. Note that as the seeds are generated for each stress test, all triple stores are confronted with the same query load. Each worker sends the query that the generator selected, waits for results and sends the next query after a preset

delay according to a delay strategy (constant delay, Gaussian delay, etc.) specified by the user. Each of the workers sends queries to the endpoint until the *benchmark runtime* (which is shared across all workers) has elapsed.

3.2 Update Component

The update component relies on a predefined set of triple additions and deletions that are to be carried out during the stress test. The update component relies on a pool of a pre-defined number of *update workers* and an overall update strategy. The *update strategy* defines whether the update workers are to (1) first carry out every insert and then every deletion, (2) execute first every deletion and then every insert, (3) insert then delete in an alternating fashion or (4) alternatively first delete then insert. Each worker of the update component is also assigned a *worker strategy*, which defines whether it is only to carry out additions, deletions or both. Moreover, a *delay model* is assigned to each worker, which determines how long the worker is to wait between two update queries. The user can set a fixed time or assign a variable time with the seed value s. In case of a variable time, a pool of numbers between $s - \sqrt{s}$ and $s + \sqrt{s}$ is created. Every time a new update is needed the corresponding worker will draw a random time out of the interval to wait until it generates the next update. Again, all random values are pseudo-random and thus the same across all experiments.

Once the stress test has been completed, the results are saved. Through the whole stress test, the runtime of each query as well as the number of failed and successful queries for each and every template is saved for every user. The framework also computes (1) the number of queries per time, (2) the number of queries per second for every user and (3) the mean and sums over all users for all measurements.

4 Evaluation

The motivation behind our evaluation was to check whether the IGUANA approach reveals new insights across several benchmarks. In this section, we describe how we went about addressing this question. We begin by presenting our experimental setup and subsequently present the results of our evaluation in detail.

4.1 Experimental Setup

All experiments were performed on a desktop machine with an Intel i7-3770 CPU with 3.4 GHz, 32 GB RAM, 4 TB HDD running Ubuntu 14.04 and Java 1.7. The benchmark program and the test triple stores were executed on the same machine to avoid any network delay. We used the following criteria to select triple stores for the benchmark: (1) The triple store had to be able to load and process the DBpedia dataset which currently has 391,020,690 triples.[7] (2) The triple store had to be able deal with the characteristics of DBpedia, e.g., its high number of properties (this rules out stores such as 4Store,

[7] The W3C wiki at `http://www.w3.org/wiki/LargeTripleStores` lists triple stores that are used commonly and the number of triples they can store.

which is optimised for a low number of properties). (3) The triple store had to have no benchmarking restrictions or the maintainers had to approve the inclusion of their system in the benchmark, and the publication of the results to the public. After selecting the candidate systems and contacting the maintainers for approval when required, the following systems were included in the benchmark: OpenLink Virtuoso,[8] Blazegraph®,[9] and Apache Jena TDB.[10]

For all stores, we selected the standard configuration except for an adjustment of their memory limits. We chose this configuration because it is the configuration most commonly used by lay users. Still, we are aware that the configurations can be tuned and that the results we present are thus to be taken with the corresponding grain of salt.

The configuration of each triple store was as follows:

1. *Virtuoso* Open-Source Edition version 7.0.0: We set the following memory-related parameters: NumberOfBuffers = 1360000, MaxDirtyBuffers = 1000000.
2. *Blazegraph* Version 1.5.3, with Jetty as HTTP interface: We set the Java heap size to 16GB.
3. *Jena TDB* Version 2.3.0 with Fuseki2 as HTTP interface: We also set the Java heap size to 16GB.

We used the DBpedia Live dataset for the experiments on DBpedia.[11] For Semantic Web Dog Food[12] we computed the difference between the provided dump[13] and a dump generated by the SPARQL endpoint[14] from March 1st, 2016. This difference was separated into three files which contained (1) the triples in both the dump and the endpoint, (2) the triples available only in the dump and (3) the triples present only in the endpoint. The triples in (2) were split into 5 files, which were used to delete data from the triple store. File (3) was split into 140 files, which were used to add data to the triple store. An overview of the data sets can be found in Table 1.

We configured IGUANA as follows: The warm-up phase was set to 20 minutes. The hot run phase was set to 60 minutes. The number of workers that update the system was varied between 0 and 1, while the number of workers that query the system was set to 1, 4, or 16. As queries, we used 250 real queries benchmark generated by FEASIBLE [11] and the 20 query templates of DBPSBv2 [7]. We selected these benchmark generation frameworks because they generate benchmarks from the real query logs, thus allowing us to test the triple stores under a more realistic evaluation environment.

4.2 Results and Discussion

The aim of our evaluation was to show how IGUANA can be used to address the following research questions:

Q1: Baseline: How do triple stores scale for static data sets of different sizes?

[8] http://virtuoso.openlinksw.com
[9] http://www.blazegraph.com
[10] http://jena.apache.org/documentation/tdb
[11] The dump can be found here https://doi.org/10.6084/m9.figshare.4954598.v1
[12] http://semanticweb.org/
[13] http://data.semanticweb.org/dumps/
[14] http://data.semanticweb.org/sparql

Table 1: Overview of the data sets used in our experiments

	SWDF	DBpedia 10%	DBpedia 50%	DBpedia 100%
No. of triples	307,787	40,234,659	197,951,941	391,020,690
No. of classes	149	715	752	778
No. of properties	301	27,337	47,310	61,707
No. of subjects	32,111	2,100,802	12,637,791	26,557,064

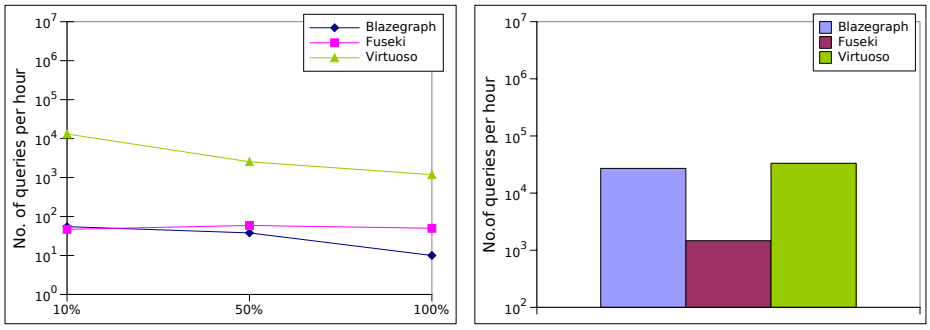
Q2: How do triple stores scale under parallel load?
Q3: How do triple stores scale under updates?
Q4: How do triple stores scale under parallel load and updates?
Q5: Are some benchmarks more demanding than others?

The configuration as well as the complete results are available for download.[15] To test the scalability of the selected triple stores on static data sets of different sizes, we created two additional subsets of DBpedia, namely DBpedia 10% and DBpedia 50% (see Table 1). The partitioning was carried out by truncating the dump. Note IGUANA also support the dataset slicing introduced in DBPSB and DAW [12]. The goal of the partitioning was to check how the triple stores perform with the increasing size of the datasets.

Figure 3 shows the performance of the selected triple stores in terms of the number of queries executed per hour for the different sizes of the DBpedia and SWDF benchmarks generated by FEASIBLE framework (Q1). As expected, the performance of Virtuoso decreases by *80.62%* while going from DBpedia 10% to DBpedia 50% and decreases further by *53.26%* while going from DBpedia 50% to DBpedia 100%. Similarly, the performance of Blazegraph decreases by *30.91%* while going from 10% to 50% and decreases by *73.68%* while going from 50% to 100%. Surprisingly, the performance of Fuseki is not greatly affected by the size of the data set, which seems to suggest that the store scales better. However in reality, the reason for this behavior is seen in the absolute number of queries that Fuseki can answer per hour. The triple store is unable to complete one query mix (i.e., 250 queries) within the time set, while Virtuoso achieves more than 40 and is thus more than 100 times faster. Fuseki's behavior is however superior to that of BlazeGraph, which has the same performance issues with DBpedia 10% and whose performance decreases further with the dataset size. The performance of the triple stores being greatly influenced by the dataset size is further confirmed by the results on the small control dataset SWDF, where Blazegraph performs as well as Virtuoso. This clearly answers Q1: while triple stores can work well with small datasets, their performance is significantly affected when it comes to dealing with large datasets. Devising scalable triple stores is an important research direction to be considered in the future.

Figure 4 shows the effect of parallel query users on the performance of the selected triple stores on the FEASIBLE queries (Q2). As an overall performance evaluation, the number of queries executed per hour increases with the number of simultaneous querying users. This is simply due to all triple stores making use of parallel answer threads.

[15] See `https://doi.org/10.6084/m9.figshare.c.3767501.v1`. Note that due to space restrictions, we cannot present all results in detail. Instead, we focus on the highlights of our findings.



(a) DBpedia (b) SWDF

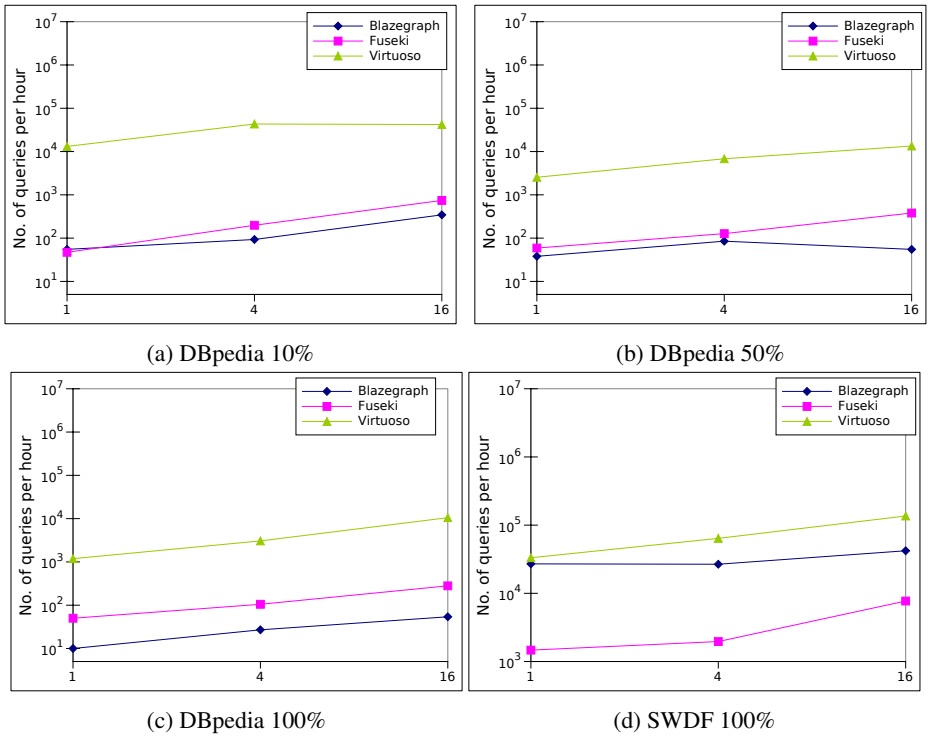
Fig. 3: Scalability test of triple stores based on single querying user and no updates using FEASIBLE. The y-axis is in logarithmic scale.

When transitioning from 1 to 4 parallel query workers on DBpedia 10%, the performance of Virtuoso increases by *222.75%*. Blazegraph is *69.09%* better and Fuseki is *3.2 times faster*. The transition from 4 to 16 parallel users leads to a further improvement of Virtuoso by *220.5%* while Blazegraph improves by *527.27%* and Fuseki by *1480.85%*.

On DBpedia 50%, we also see an increase of performance when comparing the behavior of the systems with 1 and 4 users. Only Blazegraph's performance decreases by *35.29%* when it is confronted with 16 users. The other systems keep on improving. On DBpedia 100%, the performance of Virtuoso increases by *157.02%*, Blazegraph improves by *170%* and Fuseki is *1.1 times faster* with 4 parallel users. The improvements are even larger when moving from 4 to 16 users, where Virtuoso is *784.19%* faster while Blazegraph and Fuseki improve by *440%* resp. *460%*.

Our control dataset SWDF confirms the increase in performance of all systems with more users but also points to an upper bound for the performance of the systems. For example, Virtuoso can answer *92.39%* more queries when transitioning from 1 to 4 users. However, Blazegraph's performance remains quasi constant (*-1.23%*) while Fuseki improves by *33.54%*. Virtuoso improves further when comparing its behavior with 4 and 16 users (*309.37%*) while Blazegraph increases only moderately (*55.62%*). Fuseki profits the most of the 16 parallel users on a relative scale (*+424.56%*). This clearly answers Q2.

Figures 5 shows some of the most interesting results of this work as they display (to the best of knowledge) the first results of systems with parallel queries and updates (1 query user, 1 update user). For FEASIBLE, the triple stores are barely affected by the update workers in most cases (Q3). On DBpedia 10%, the performance of Virtuoso decreases by *0.24 %*, Fuseki's increases by *29.79%*, and Blazegraph's remains constant. On DBpedia 50%, the performance of Virtuoso increases by *2.59 %*, Fuseki's remains constant, and Blazegraph's improves by *7.89 %* with single worker updates. On DBpedia 100%, small losses (Virtuoso = *-6.98 %*, Fuseki = *-16%*, Blazegraph = *-10 %*) can be monitored. A similar picture can be derived from the results on SWDF (Virtuoso



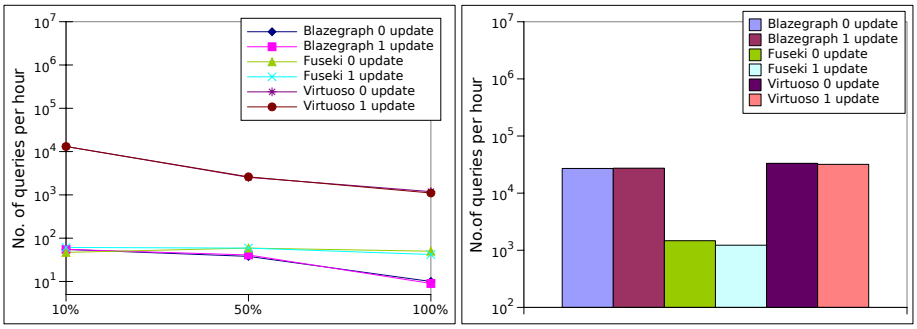
(a) DBpedia 10% (b) DBpedia 50%

(c) DBpedia 100% (d) SWDF 100%

Fig. 4: Effect of parallel requests using the FEASIBLE queries. The x-axis shows the number of simultaneous querying users

= *-4.02%*), Fuseki = *-16.6%*, Blazegraph = *-1.14 %*). The results suggest that a single query and update worker duo does not significantly affect the overall performance of triple stores when faced with FEASIBLE queries.

We were hence interested to know how triple stores scale under parallel load and updates (Q4). Figure 6 shows that the performance of all triple stores only decreases slightly on DBpedia 10% with parallel loads and updates, as compared to only parallel loads and no updates (ref. Figure 4). Here, no system has more than 20% performance loss (Virtuoso = *-2.32%*, Blazegraph = constant, Fuseki = *3.03%* with 4 parallel users; Virtuoso = *-0.42%*, Blazegraph = *-16.23%*, Fuseki = *-6.59%* with 16 parallel users; a similar picture). On DBpedia 50%, more drastic performance changes occur, with Blazegraph's performance decreasing by *61.18%* with 4 parallel users and decreasing further by *20%* with 16 parallel users. In combination with Figure 7, IGUANA allows for the first unified comparison of benchmark results (FEASIBLE vs. DBPSBv2). The clear decrease in performance of Fuseki on DBPSBv2 (more than 1 order of magnitude, see Figure 7) demonstrates that (1) FEASIBLE pushes most of the systems closer to the edge than DBPSBv2, leading to the systems not being able to carry out a lot of queries (Q5) and (2) Virtuoso clearly scales up to heavy load better than the other solutions.



(a) DBpedia (b) SWDF

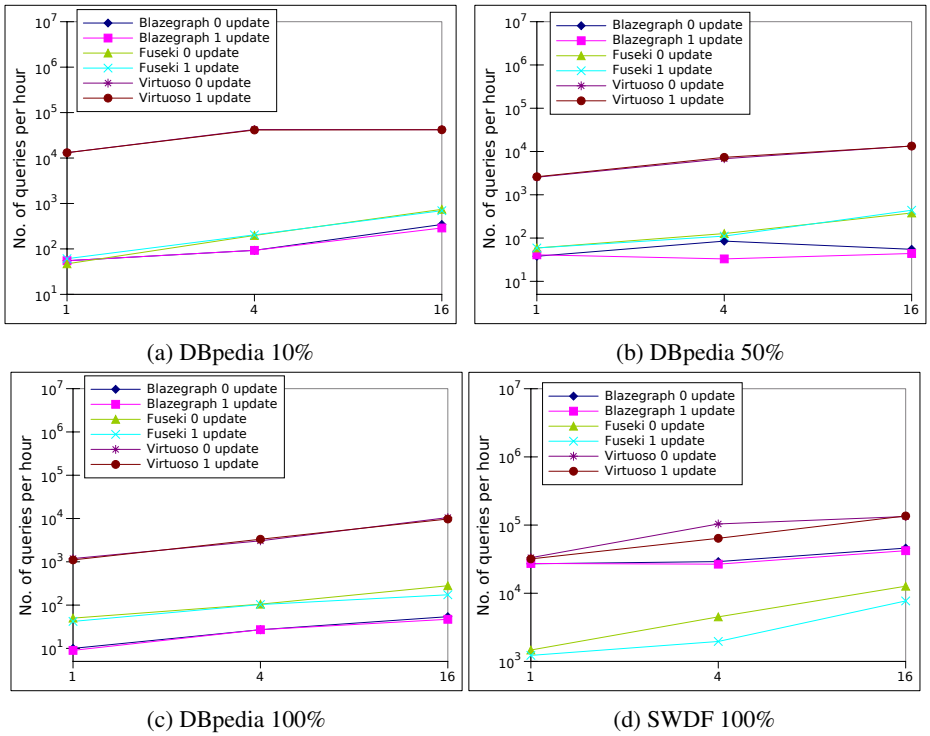
Fig. 5: Scalability test of triple stores based on a single querying user and a single update user for FEASIBLE. The y-axis is in logarithmic scale.

Overall, these results suggest that all systems can deal well with parallel updates and queries. However, their performance is significantly affected by the dataset sizes. Virtuoso is clearly the fastest system in all experiments while Fuseki is most commonly faster than Blazegraph. The systems all implement efficient parallel query handling and can thus be used for multiple concurrent requests. While Fuseki seems to scale up well with the number of concurrent users, its performance with 16 users still remains significantly poorer than Virtuoso's.

5 Related work

Several RDF benchmarks were developed over recent years. The Lehigh University Benchmark (LUBM) [5] is a synthetic benchmark that aims to test the triple stores and reasoner for their reasoning capabilities. The synthetic data is about universities, their departments, the professors, etc. SP^{2}Bench [14] is a synthetic benchmark for testing the query processing capabilities of triple stores. The synthetic data is based on the DBLP [16] bibliographic database. The Berlin SPARQL Benchmark (BSBM) [2] is a synthetic triple stores benchmark based on an e-commerce use case in which a set of products is provided by a set of vendors and consumers post reviews regarding those products. Since v3.1 it uses synthetic updates. It tries to mimic a real user operation, i.e., it orders the queries in a sequence to resemble the real operation sequence performed by a human user. The SRBench [16] is a RDF benchmark designed for the benchmarking of streaming RDF/SPARQL engines. The streaming data arrives as a continuous stream at a high rate. It uses real RDF data sets and 17 synthetic queries. The main advantage of that benchmark is that it addresses the various features of SPARQL 1.1 and reasoning. In [8], the authors propose a synthetic benchmark based on `Last.fm`, which can benchmark systems w.r.t. various SPARQL 1.1, e.g., property paths, and subqueries. [15] proposes a SPARQL benchmark based on electronic publishing scenario. It uses 8

[16] `http://www.informatik.uni-trier.de/~ley/db/`



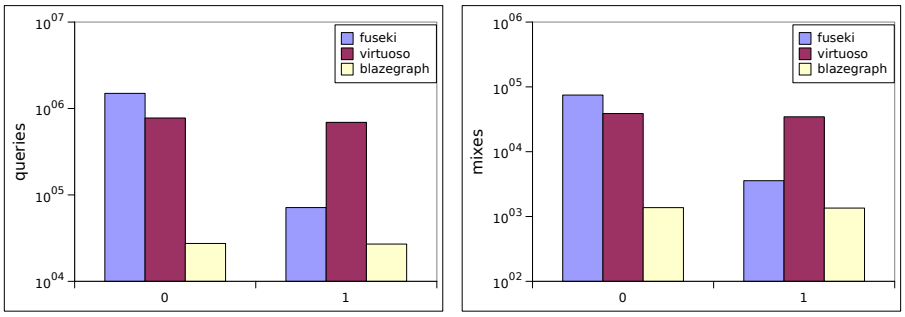
(a) DBpedia 10% (b) DBpedia 50%

(c) DBpedia 100% (d) SWDF 100%

Fig. 6: Effect of parallel requests with and without updates, X-axis shows the number of simultaneous querying users

different data set sizes, and 19 queries covering various SPARQL constructs. Although the data used is real, the queries are still synthetic. *The Waterloo SPARQL Diversity Test Suite* (WatDiv) [1] provides synthetic data and query generator to generate large numbers of queries from a total of 125 queries templates. The queries cover both simple and complex categories with a varying number of features such as result set sizes, total number of query triple patterns, join vertices and mean join vertices degree. However, this benchmark is restricted to conjunctive SELECT queries (single BGPs).

The DBpedia SPARQL Benchmark (DBPSB) [7] is a SPARQL benchmark that uses both real data, i.e., DBpedia, and real queries, i.e., the query log of the DBpedia endpoint, for benchmarking. An important feature of DBPSB is that it selects the queries based on their frequency, i.e., it does not only selects the queries that cover certain SPARQL features, but it also picks the query with the highest frequency among those queries of the query log. However, this benchmark does not consider key query features (i.e., number of join vertices, mean join vertices degree, mean triple pattern selectivities, the query result size and overall query runtimes) while selecting query templates. Previous works [1,3] have however pointed out that these query features greatly affect the triple stores performance and thus should be considered while designing SPARQL benchmarks. Some of the drawbacks of DBPSB are addressed by the



(a) Number of queries (b) QmpH

Fig. 7: Queries per Hour and Query Mixes Per Hour with 16 querying workers for the DBPSBv2 Benchmark on DBpedia 100%. The x-axis shows the numnber of update users. The y-axis is in logarithmic scale.

FEASIBLE benchmark [11], a real benchmark generation framework which can generate customized benchmarks out of query logs. The approach underlying the benchmark takes SPARQL features as well as SPARQL query types into consideration while deriving prototypical queries and can easily be ported to any query log. Finally FedBench [13] and LargeRDFBench[9] are benchmarks for federated SPARQL query processing. To the best of our knowledge, IGUANA is the first unified benchmark execution platform for SPARQL queries. This framework is orthogonal to the current state of art as it allows the execution of all of the benchmarks above and the comparison of their results with and without data updates and parallel requests. In addition, IGUANA is also able to execute federated SPARQL queries benchmarks.

6 Conclusions and Future Work

We presented IGUANA, an execution framework for SPARQL query benchmarks. We evaluated 3 triples stores on 4 datasets under 6 different settings using FEASIBLE and DBPSBv2. Our results unveiled that the triple stores perform well on small amounts of data and scale well with the number of reading users. Moreover, they are also able to deal well with 1 update user. However, systems such as Blazegraph struggle with large datasets such as DBpedia 100%. For the first time, we were able to compare two benchmarks within an identical environment and revealed that the FEASIBLE queries stress triple stores significantly more than the DBPSBv2 queries. Overall, we showed that IGUANA can be used for benchmarking triple stores in a variety of settings and that this flexibility gives new insights into the behavior of triple stores. In future works, we will extend of our framework to streaming RDF data. The sustainability of the framework will be ensured by making it one of the key assets of the HOBBIT association, which will emerge from the EU-funded project HOBBIT (`http://project-hobbit.eu`) and already has 9 funding members.

7 Acknowledgements

This work was supported by the project HOBBIT, which has received funding from the European Union's H2020 research and innovation action program (GA number 688227), and by the SAKE project (GA 01MD15006E) financed by the BMWI.

References

1. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of rdf data management systems. In *International Semantic Web Conference (ISWC)*. 2014.
2. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
3. O. Görlitz, M. Thimm, and S. Staab. Splodge: Systematic generation of sparql benchmark queries for linked open data. In *International Semantic Web Conference (ISWC)*. 2012.
4. J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann, 1991.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *J. Web Sem.*, 3(2-3):158–182, 2005.
6. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In *International Semantic Web Conference (ISWC)*, 2011.
7. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Usage-Centric Benchmarking of RDF Triple Stores. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.
8. M. Przyjaciel-Zablocki, A. Schätzle, T. Hornung, and I. Taxidou. Towards a sparql 1.1 feature benchmark on real-world social network data. In *Proceedings of the First International Workshop on Benchmarking RDF Systems*, 2013.
9. M. Saleem, A. Hasnain, and A.-C. N. Ngomo. Largerdfbench: a billion triples benchmark for sparql endpoint federation. *Web Semantics: Science, Services and Agents on the World Wide Web. Elsevier*, 2017.
10. M. Saleem, M. R. Kamdar, A. Iqbal, S. Sampath, H. F. Deus, and A.-C. Ngonga Ngomo. Big linked cancer data: Integrating linked tcga and pubmed. *JWS*, 2014.
11. M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo. FEASIBLE: A featured-based sparql benchmark generation framework. In *International Semantic Web Conference (ISWC)*, 2015.
12. M. Saleem, A.-C. Ngonga Ngomo, J. Xavier Parreira, H. Deus, and M. Hauswirth. DAW: Duplicate-aware federated query processing over the web of data. In *ISWC*, 2013.
13. M. Schmidt, O. GÃűrlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, editors, *International Semantic Web Conference (ISWC)*, volume 7031 of *Lecture Notes in Computer Science*, pages 585–600. Springer, 2011.
14. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: A SPARQL performance benchmark. In *International Conference on Data Engineering (ICDE)*, pages 222–233. IEEE, 2009.
15. T. Tarasova and M. Marx. Parlbench: a sparql benchmark for electronic publishing applications. In *The Semantic Web: ESWC 2013 Satellite Events*, pages 5–21. Springer, 2013.
16. Y. Zhang, M.-D. Pham, Ó. Corcho, and J.-P. Calbimonte. SRBench: A Streaming RDF/SPARQL Benchmark. In *International Semantic Web Conference (ISWC)*, pages 641–657, 2012.