

SPARUB: SPARQL UPDATE Benchmark

Damien Graux, Pierre Genevès, Nabil Layaïda

▶ **To cite this version:**

Damien Graux, Pierre Genevès, Nabil Layaïda. SPARUB: SPARQL UPDATE Benchmark. 2017. hal-01523496

HAL Id: hal-01523496
https://inria.hal.science/hal-01523496

Preprint submitted on 16 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SPARUB: SPARQL UPDATE Benchmark

Damien Graux, Pierre Genevès, and Nabil Layaïda

Inria, Cnrs, lig and Univ. Grenoble Alpes, France
{damien.graux,nabil.layaida}@inria.fr
pierre.geneves@cnrs.fr

Abstract. One aim of the rdf data model, as standardized by the w3c, is to facilitate the evolution of data over time without requiring all the data consumers to be changed. To this end, one of the latest addition to the sparql standard query language is an update language for rdf graphs. The research on efficient and scalable sparql evaluation methods increasingly relies on standardized methodologies for benchmarking and comparing systems. However, current rdf benchmarks do not support graphs updates. We propose and share sparub: a benchmark for the sparql update language on rdf graphs. The aim of sparub is not to be yet another rdf benchmark. Instead it provides the mean to automatically extend and improve existing rdf benchmarks along a new dimension of data updates, while preserving their structure and query scenarios.

1 Introduction

The research on efficient and scalable sparql evaluation methods increasingly relies on standardized methodologies for benchmarking and comparing systems. Indeed, there are many benchmarks designed for evaluating rdf systems, for instance [3,8,18,13,5,21,2,23]. Some of them are particularly popular *e.g.* lubm [13] or watdiv [2] and are becoming *de facto* standards to compare new evaluators with the already existing ones: see for instance some evaluator descriptions (sparqlgx [11], rya [19]) or surveys such as [7]. This profusion of benchmarks even leads to studies that compare benchmarks [9,20].

However, data dynamicity –which is an important criterion that affects the way data management systems are built and should also be evaluated– is rarely considered in the literature. The spectrum of data dynamicity is wide with immutable datasets on one end and data streams on the other end. In the literature, most papers on benchmarks for query evaluation consider immutable datasets (see *e.g.* [3,8,18]) with the exception of [23] that considers the other extrema by focusing on streaming rdf data.

In many practical applications however, datasets are neither streams nor immutable, but rather exhibit a small level of dynamicity; *i.e.* a large amount of the dataset remains unchanged and a slight portion of the data evolves over time, to reflect the changes of the entities they describe. This is the setting that we consider in this paper, and for which we develop extended benchmarking

methodology. In this setting, more or less frequent data updates are required to correct, add, or delete small portions of the whole dataset.

Several approaches exist for reevaluating queries after such modifications. They can be roughly divided in two categories: (1) those that re-load from scratch the whole dataset taking into account the changes and (2) those that allow slight (incremental) modifications to update an already loaded dataset. The former category might lead to redundant computations. For the latter category, however, the fine-grained incremental propagation of data updates might lead to the existence of thresholds (in *e.g.* the size or the complexity of the updated data portion) from which re-loading the whole new dataset yields better performance when compared to incremental reevaluation. More generally, current benchmarking technologies provide only very limited insights on how RDF systems rank in the presence of data dynamicity.

Contribution. In order to extend standard benchmarking processes with the ability of evaluating how RDF systems react to data updates, we propose and share the SPARQL UPDATE Benchmark (SPARUB). The aim of SPARUB is not to be yet another RDF benchmark. Instead it provides the mean to automatically extend and improve existing RDF benchmarks along a new dimension of data updates, while preserving their structure and query scenarios. The SPARUB resource is openly available under the CeCILL license[1] from:

`https://github.com/tyrex-team/sparub`

Outline. The rest of this study is organized as follows. First, we briefly remind common Semantic Web standards, the context of our work and the main features introduced with SPARQL UPDATE which constitute the core of the benchmark process in Section 2. The detailed behaviour of SPARUB and the advisable metrics to look at are described in Section 3. We next present experiments with SPARUB on popular evaluators in Section 4. Finally, we review related work in Section 5 before concluding in Section 6.

2 Background

Datasets. The Resource Description Framework (RDF) is a language standardized by W3C to express structured information on the Web as graphs [15]. It models knowledge about arbitrary resources using Unique Resource Identifiers (URIs), Blank Nodes and Literals. RDF data is structured in sentences –or triples– written (s p o), each one having a subject s, a predicate p and an object o.

Query Language. In the Semantic Web, querying RDF data is mainly realized using the SPARQL Protocol and RDF Query Language *i.e.* SPARQL which became a standard thanks to the RDF data access working group (DAWG) [12]. The SPARQL standard language has been studied under various forms and fragments,

[1] CeCILL details at `http://www.cecill.info/index.en.html`

for instance the Basic Graph Pattern fragment (a.k.a. BGP) which strictly focuses on conjunctions of triple patterns *i.e.* a triple where the elements (subject, predicate, object) are either variable or value such as IRI.

Graph store. In the rest of this paper, we will use "store" (instead of "RDF store" or "Graph store") to designate a mutable container of RDF graphs managed by a single service. A store contains an unnamed slot which helds the default graph and if necessary additional slots named for desambiguation. As a consequence, operations must specify the target graph or the default one will be considered for instance using the `FROM` SPARQL keyword.

Updates. The W3C extended the SPARQL syntax [12] – which is the RDF query language [15] – with a new recommendation dealing with updates [22]. This extension provides new keywords to add of delete triples from a database using the classic SPARQL syntax. Such an evolution allows to build more realistic application for the Semantic Web. Thereby, instead of manually (or externally) modify the datasets, this extension offers a way to perform these changes internally.

Introduced in 2013 by the W3C, SPARQL UPDATE[2] [22] – also designated by "SPARUL" – is an update language for RDF graphs using a SPARQL-like syntax. It is poised to become the standard method to handle modifications in RDF graphs by providing a set of keywords where each corresponds to an operation on graphs *e.g.* inserting triples in a graph store, deleting a whole graph store... Indeed, this fragment extension is part of the SPARQL 1.1 release of the standard.

More precisely, SPARQL UPDATE allows two categories of operations on RDF graphs *i.e.* a set of operations directly deals with graphs (management) where the other one treates triples of a designated graph (specific updates). From a high perspective, the fragment provides the following primitives: `create`, `drop`, `copy`, `move` and `add` which repsectively allow to create or delete a graph, to duplicate data from on graph to an other, to move (or rename) one and to append a graph to an other one. To deal with updates directly in an already existing graph, the standard provides these primitives: `insert data` and `delete data` are able to insert or delete specific triples that should be explicitly given; `load` allows to add new data coming from a file into a designated graph; `clear` empties a graph and `delete/insert` allows to delete or insert data into a graph according to solutions extracted from an other graph like the classic behavior of a `construct` in the SPARQL core which returns triples that are solutions of a list of conditions in the `where` clauses.

3 SPARUB: the SPARQL UPDATE Benchmark

3.1 General Architecture

In this Section, we present the challenges that RDF updates impose on the design of a SPARQL benchmark followed by the practical details of our standardized process of benchmark generation.

[2] `https://www.w3.org/TR/sparql11-update/`

SPARUL instruction	Corresponding sub-step
CREATE	A E
INSERT DATA	C
DELETE DATA	C
INSERT/DELETE	D
CLEAR	E F G H
LOAD	B E F
DROP	H
COPY	C G
MOVE	G
ADD	F G

Table 1: Required SPARUL keywords to complete a SPARUB sub-step.

Challenges. First of all, in order to test the various possible scenarios allowed by the standard, the benchmark should maintain the same dataset structure as the initial one. As a consequence, the creation of the multiple RDF sub-datasets constitutes a key point of the "realism" of the tool and requires knowledge about the graph structure.

In parallel, the update benchmark has to respect the initially benchmarked SPARQL fragment, *e.g.* it should only involved conjunctive queries if the given queries are focusing on. This step will therefore imply to be able to look at the shapes that are considered originally.

Finally, SPARUB should build concise sketch of benchmark divided into small steps so users can rank evaluators on each sub-step.

Concept. The main idea of SPARUB is to allow an extension of all the existing RDF/SPARQL benchmarks *i.e.* keeping the already run test suite and giving new scenarios involving dynamicity of data. To do that, SPARUB takes as inputs an RDF dataset (typically with the N-Triples format [1] to be easily parsed) and an optional list of SPARQL queries. It then returns a set of RDF files and a testing scenario divided into several sub steps. Technically, SPARUB starts by analyzing the various SPARQL queries in order to list the used SPARQL keywords and thus having an idea of the benchmarked fragment. It then splits the input RDF file into pieces according (1) to statistics of the initial dataset and (2) to the various sub-fragment that can be extracted from the general tested one.

Sub-steps. The whole benchmark process is actually divided into several sub-steps. Indeed, each one focuses on a specific set of SPARQL UPDATE functionalities and thus on a specific scenario. Moreover, SPARUB always uses the given dataset to generate its subsets, it thus guarantees that the tests are done considering the same graph structure than the initial graph and therefore allows to extend correctly the already obtained results. Specifically SPARUB generates in the following order:

A) The initialization which creates the needed graphs for the benchmark without loading any triples in;

B) A reference run of the initial benchmark to know how the benchmarked system reacts and to have reference times. This step is composed of the dataset load and of the evaluation of the given queries (to SPARUB) if any;
C) The sub-step focuses on updating triples in already existing graphs. It thus deals with insertions and deletions of pieces of data which are explicit *i.e.* the sets of updates are already known and not extracted from specific patterns. Several cases are considered: from inserting only one triple to several thousands. In addition, this sub-step also offers to compare performances to insert k triples one by one and k triples in one time;
D) This step is an extension of the previous one since it mainly consists of moving blocks of data according to specific properties. Indeed, it focuses on deleting all the triples related to the most common predicate and also plans to move all the triples related to the most common couple *predicate-object*;
E) In this scenario, SPARUB focuses on comparing updating strategies *i.e.* it gives clues to answer "when is it better to re-load everything instead of inserting the new triples with an `insert data`?". To do that, a reference time is recorded to load k triples, then several size are considered: first SPARUB offers to load 99% of the k with an insertion of the 1% left (a 99L-1I), then a 90%-load with a 10%-insertion (90L-10I) followed by a 75L-25I, a 50L-50I and finally a 20L-80I. These variations –with increasing size of the updated part– allow to observe the potential linearity of the updates;
F) [*Optional.*] –The creation of this sub-step depends on the existence of specific SPARQL queries as arguments.– If any, SPARUB provides specific scenarios for each given SPARQL query to measure the impact of updates to evaluate it. Actually, (1) it starts analyzing the query to generate several subsets of various size of triples which might be parts of the solutions, (2) it considers the performance of the SPARQL query –computed in the sub-step B–, (3) for each generated sub-step it inserts the potential solutions into the graph and re-evaluates the query;
G) This penultimate sub-step acts on graphs instead of triples and has thereby a higher perspective. The graph manipulations in the SPARQL UPDATE fragment mainly consist of copying, moving and adding graphs. One of the scenario is to compare the needed times between these instructions *e.g.* we consider two graphs G_1, G_2 and G_3 which are respectively non-empty, empty and empty, first we copy G_1 into G_2 which leads to two indentical graphs, then we move G_2 into G_3 (there are still two identical graphs and an empty) and finally we add G_3 to G_2;
H) Finally a cleaning step which empties and then drops the various graphs used during the benchmark is realized to reset everything.

We present in Table 1 a mapping between the existing keywords of the standard and the sub-steps where they are required. A quick-look gives an idea of the fragment coverage of an evaluator.

In Table 2, we present typical examples of the SPARUB generated SPARQL queries for each sub-step. For instance, we can see that the initialization (sub-step A) only deals with graph creation. This representation allows to notice

Sub-step	Typical Generated Queries:
A	`CREATE graph <name>`
B	`LOAD ./path/dataset INTO GRAPH <name>`

C	`COPY GRAPH <name1> TO GRAPH <name2>`
	`INSERT DATA { GRAPH <name> { <s> <p> <o> . }}`
	`INSERT DATA { GRAPH <name> { <s> <p> <o> . }}`

D	`WITH <name> DELETE ?s <pred> ?o WHERE {?s <pred> ?o .}`

E	`CREATE GRAPH <name>`
	`LOAD ./dataset-k% INTO GRAPH <name>`
	`LOAD ./dataset-(100-k)% INTO GRAPH <name>`
	`CLEAR GRAPH <name>`

[*optional*] F	`LOAD ./possible-solutions INTO GRAPH <name>`

G	`COPY GRAPH <ref> TO GRAPH <full>`
	`MOVE GRAPH <full> TO GRAPH <empty>`
	`ADD GRAPH <empty> TO GRAPH <full>`

H	`CLEAR GRAPH <name>` `DROP GRAPH <name>`

Table 2: Typical queries for each sub-step.

that several sub-steps are thus independant *e.g.* the sub-steps D & G have no SPARQL UPDATE keywords in common; it implies that SPARUB can be seen as a juxtaposition of several independent scenarios.

3.2 Advisable Metrics

In order to help user to rank the evaluators, SPARUB also provides a set of metrics that might be looked at. First of all, it is recommended to save the needed times of each query.

Since the SPARQL UPDATE fragment is an extension of the SPARQL core language, popular evaluators focus on the main fragment (see Section 5). For this reason, the first interesting element is the percentage of the language which is supported by. That is why SPARUB is divided into several sub-steps (see the previous sub-section for details), indeed, they can be considered independently and thus a selection of the possible sub-step can be done using the Table 1.

Moreover, we recommend to pay attention to the way memory or disks are used during the benchmark. Actually, it is possible that each update creates a new structure to keep track of the history of modification which is a feature not required by the standard and thus gaps between stress insertions (see sub-step C) and a bulk update might be observed for instance.

Finally, some sub-steps aims at providing specific behaviors of the benchmarked evaluators. Indeed, with sub-steps E and F, *i.e.* with sub-steps where a reference time (from sub-step B) is involved, in addition to time considerations, we also encourage to look at ratios T_{Update}/T_{Ref}.

Benchmark		Dataset			Queries		SPARUB	
Name	Parameter	Triple Nb	Size	Time(s)	Query Nb	Time(s)	Size	Time(s)
WatDiv	1	110063	15M	0.7	20	4.7	9.6M	0.5
WatDiv	20	2192105	294M	10.9	20	5.1	60M	10.5
WatDiv	100	10992378	1.5G	56.6	20	7.0	273M	54.9
SP^{2}Bench	100000	100080	11M	0.4	17	0	8M	0.5
SP^{2}Bench	2000000	2000167	214M	8.3	17	0	49M	10.2
SP^{2}Bench	10000000	10000460	1.1G	49.2	17	0	222M	50.5

Table 3: Using SPARUB with state-of-the-art benchmarks.

4 Experimental Validation

In this Section, we now provide practical information about SPARUB. The experiments presented are twofold. First, we start by showing the orthogonal aspect of SPARUB since it can generate update scenarios from any RDF/SPARQL benchmark. Second, we review how some popular state-of-the-art SPARQL evaluators are dealing with RDF updates using some SPARUB sub-steps.

4.1 Examples with popular benchmarks

Since SPARUB should be used as an extension of already existing benchmarks –it takes a RDF dataset and an optional list of SPARQL queries–, we briefly present how it behaves taking as sources some popular benchmarks of the literature.

Selected Benchmarks. For the purpose of this study, we select two benchmarks from the state-of-the-art:

- WatDiv [2] which has been proposed in 2014 by the university of Waterloo.
- SP^{2}Bench [21] which focuses on a DBLP scenario.

These benchmarks both have deterministic dataset generators giving RDF triples according to the N-Triples format [1]. Nonetheless, their sets of SPARQL queries are obtained differently: SP^{2}Bench comes with a set a predefined ones while WatDiv needs to generate its set according to (1) query patterns and (2) statistics on the dataset it has just created. We can also notice that the considered SPARQL fragments differ a lot between these two benchmarks *i.e.* SP^{2}Bench deals with an extension (with *e.g.* `union`, `optional` or also `ask`) of the BGP fragment which is the focus of WatDiv.

Observations. In Table 3, we present the obtained results to generate state-of-the-art benchmarks and their SPARUB extensions. These results were computed through a Docker image [17] we designed[3] on a four processor machine. The use of Docker is especially important for reproducibility reasons, allowing users to

[3] The Dockerfile is also openly available from our repository.

deploy our benchmark in the same environment. In particular, for each benchmark scenario (various scenarios are generated for each benchmark depending on the scaling factor given to the dataset generator), the Table shows:

- the needed parameters that should be given to the generators.
- information about the generated datasets *e.g.* the number of RDF triples, the size of the dataset and the time needed to complete this process.
- information dealing with the SPARQL queries such as their number and the time needed to generate them if any.
- the time SPARUB needs to generate its scenarios taking as inputs the freshly generated dataset and all its queries.
- the size of the SPARUB generated files.

We note (Table 3) that the SPARUB generation time and the dataset generation time are almost the same for instance ≈10 seconds for WatDiv 20 or ≈50 seconds for SP^{2}Bench with 10000000 triples. This observation is explicated by two reasons: (1) SPARUB needs to read several times the input dataset to generate specific scenarios such as sub-step D and (2) it also splits the input dataset into parts which have the size of k% of this initial dataset to set up sub-step E for example. That is why the size of SPARUB generated files increases with the size of the initial dataset: the larger is the initial dataset, the larger will be the updates.

4.2 Examples with SPARQL evaluators

In this Section, we briefly present experimental results obtained with some popular state-of-the-art systems claiming that they are supporting the SPARQL UPDATE fragment.

Reproducibility. We share (in the SPARUB repository) the testing scripts and the installation processes we used in this Section. Moreover, to allow easy deployments, we provide also a Dockerfile to generate the **exact** same image as us.

Selected Evaluators. The first matter was to find systems claiming their support of the SPARQL fragment extension dedicated to updates. For our experiments, we selected the following systems:

1. Virtuoso which is a general purpose relational / federated database and applications platform introduced in [10]. It allows to load RDF data and then query these datasets using SPARQL through a relational context *i.e.* the RDF datasets are stores in the Virtuoso table warehouse and SPARQL queries can be given using the SQL command line tool.
2. 4store which is a native RDF solution introduced in [14]. It has an index to translate URIs to identifiers, which allows a space-efficient representation of triples. For each predicate it uses two indexes (subject to object and object to subject) for optimizing query evaluation.

Required Operation	4store	Virtuoso	Jena
Reference Load (sub-step B)	1.187 s	0.983 s	1.269 s
Inserting one triple	0.08 s	0.010 s	0.07 s
Inserting 20 triples	0.08 s	0.020 s	0.13 s
Inserting 500 triples	0.1 s	0.032 s	0.54 s
Inserting 10% of the base	Failure	Failure	Failure

Table 4: Inserting Triples with `insert data` (*i.e.* sub-step C) to Watdiv 1.

3. Apache Jena [16] which is a free and open source Java framework for building semantic web and Linked Data applications. It comes with several APIs to evaluate SPARQL queries and store RDF datasets.

For these experiments, we remain in a single-node context on the same machine that the one used previously to test SPARUB against various popular state-of-the-art benchmarks.

Observations. Since SPARUB generates several dozens of SPARQL queries to test all the possible scenarios and since none of the tested systems were able to cover the whole standard extension, we present here an excerpt of the results restricted to successful sub-steps. Indeed, even simple scenarios such as the sub-step D seems complicated for some stores (*e.g.* 4store) which were not able to parse queries of the kind: "`DELETE ?s <friend> ?o WHERE{?s <friend> ?o.}`".

In Table 4, we present the performance of the three tested evaluators to deal with `insert data` (and `delete data` which gives exactly the same results) considering the WatDiv 1 RDF dataset. This scenario corresponds to the sub-steps B & C of SPARUB *i.e.* inserting one triple, then twenty triples, followed by 500 ones and finally inserting about 10% of the dataset size. The queries all have the following forms: "`insert data where{s1 p1 o1 . s2 p2 o2 . [...]}`". Focusing on this scenario offers us the possibility to test all the update operations provided by Virtuoso[4] *i.e.* `insert data`, `delete data`, `load` and `clear`.

For the three evaluators, we notice (see Table 4) that inserting triples is almost immediate: for example less than one second to compute the update. However, none of them was able to insert the 10%-update *i.e.* in our case to insert about 10'000 triples. These failures pinpoint that these evaluators do not consider that an update can be larger than several hundreds of triples.

Summary. The main lesson of our experiments is that only a few evaluators are able to deal with updates. Indeed, they mainly focus on optimizing the SPARQL SELECT fragment in the case of static datasets. Second, we also found that even the SPARQL UPDATE compliant systems were not able to cover the whole standard extension and they often need the users to adapt the SPARQL queries manually because they do not respect exactly the standard. Third, a

[4] Virtuoso SPARQL 1.1 coverage: `https://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtTipsAndTricksSPARQL11Update`

silver lining remains since the systems' performance with some sub-steps were very fast *i.e.* sub-second response times.

5 Related Work

Parallely to the development of SPARQL evaluators and RDF stores, several RDF/SPARQL benchmarks have been published in order to rank the various solutions and thus help users in system selection. Indeed, comparative experiments are relevant ways to rank available SPARQL evaluators. For instance, by loading the same RDF datasets on each system and by querying them after, one can decide which system is the fastest.

Based on that idea, researchers have developed standardized and reproducible RDF/SPARQL benchmarks. These benchmarks are usually made of two parts: first the datasets and second a list of queries which should be evaluated on these datasets; sometimes a testing scenario is also presented, for example, it provides a defined order to execute the queries and suggests that some should be tested several times. Since, the most important concept of them is to offer reproducibility to the community, datasets can often be generated – in a deterministic way – and the list of queries is either pre-defined or generated. It even exists tools to generate RDF "fake" data from an initial dataset that share the same structure *e.g.* GRR [6].

Moreover, these benchmarks are often specialized to test particular fragments of the SPARQL grammar. First of all, they almost always focus on `SPARQL SELECT` queries. In addition, we also notice that the BGP fragment constitutes the common base which is always tested in the set of queries and sometimes additional SPARQL keywords such as `OPTIONAL` or `UNION` are part of the patterns. Nonetheless, specific benchmarks dedicated to peculiar SPARQL extensions have also been developed, for instance the SRBench [23] which focuses on streaming RDF systems.

Practically, the RDF/SPARQL benchmarks usually rank SPARQL evaluators according to the temporal performance of the tested systems. Indeed, they often recommend to pay attention to needed times to load dataset and then to execute each query. Sometimes they also consider the disk footprint of the system. Finally, they may also provide *mixed metrics* where various measurements are aggregated using for instance averages after several computations of the test suite.

In the last fifteen years, a large number of benchmarks have been created and deployed. We provide here a non-exhaustive list of popular benchmarks:

- LUBM [13] is a benchmark proposed in 2005 by the Lehigh University. It focuses on the BGP fragment with 14 SPARQL queries which should be evaluated on generated datasets.
- WatDiv [2] is a more recent benchmark proposed in 2014 by the university of Waterloo. It provides a deterministic RDF data generator. It then also provides sets of SPARQL which should be generated according to 20 query-shapes and the previously generated dataset. These shapes are divided into

4 types: centralized, stared, linear and "snow flake". It strictly focuses on the BGP fragment.
- SP^{2}Bench [21] is settled in the DBLP scenario and comprises both a data generator for creating arbitrarily large DBLP-like documents and a set of carefully designed benchmark queries. It also tests a large fragment of SPARQL with `FILTER`, `OPTIONAL`, `UNION`, the solution modifiers and also three `SPARQL ASK` queries.
- BolowgnaBench [8] provides a framework for evaluating the performance of RDF systems on a real-world context derived from the Bologna process; it strains systems using both analytic and temporal queries; and it models real academic information needs. In terms of SPARQL fragment, it focuses on testing BGPs and also provides queries with `SELECT`-aggregators such as `COUNT` which are part of the standard since the 1.1 version.
- BSBM [5] has been designed to compare performance of native RDF stores with the performance of SPARQL-to-SQL rewriters across architectures. It provides a "query mix" which tests the same SPARQL fragment as SP^{2}Bench excepted the `ASK` but instead it tests also the negation and the `CONSTRUCT`.
- DBPSB [18] – DBPedia SPARQL Benchmark – is a general SPARQL benchmark procedure, which uses the DBpedia [4] knowledge base. The benchmark is based on query-log mining, clustering and SPARQL feature analysis. In contrast to other benchmarks, it performs measurements on actually posed queries against existing RDF data.
- RBench [20] is an application-specific framework to generate RDF benchmarks: it takes an RDF dataset as a template, and generates a set of synthetic datasets with similar characteristics including graph structure and literal labels. RBench then analyzes several features from the given RDF dataset, and uses them to reconstruct a new benchmark graph. A flexible query load generation process is then proposed according to the design of RBench.

More generally, it even exists federated projects to develop such benchmarks. For instance, Linked Data Benchmark Council [3] is an european project that aims to develop industry-strength benchmarks for graph and RDF data management systems.

In this study, we did not create an other RDF/SPARQL benchmark dedicated to a new variation of a SPARQL fragment dealing with `select` queries. Instead, we tried to benefit from all the already existing research by adding the possibility of benchmarking a new dimension which is orthogonal to what is usually looked at *i.e.* the RDF dynamicity.

6 Conclusion

We present and share a new benchmark dedicated to the SPARQL UPDATE fragment which is a W3C extension of the SPARQL standard. To the best of our knowledge, it constitutes the first tool openly available which can rank the SPARQL evaluators focusing on updates. In addition, we pay attention not to provide a

complex standalone tool, indeed we decided to improve already existing benchmarks instead of building dedicated RDF graphs from scratch while only requiring a simple *bash* environment. In that sense, SPARUB[5] adds a new dimension to already existing benchmarks and thereby allows to improve the testing scenarios without invalidating anything.

In addition, the experiments we conducted show that the support of the RDF data dynamicity is still in a very early stage of development since: (1) only a few systems are partially supporting this fragment extension and (2) improvements could also be developed for these covered parts, in particular for supporting finer-grained updates. For these reasons, we believe that SPARUB will be helpful in measuring the progress in implementing the SPARQL UPDATE fragment.

References

1. RDF 1.1 N-Triples: A line-based syntax for an RDF graph (2014), http://www.w3.org/TR/n-triples/
2. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: ISWC. pp. 197–212. Springer (2014)
3. Angles, R., Boncz, P., Larriba-Pey, J., Fundulaki, I., Neumann, T., Erling, O., Neubauer, P., Martinez-Bazan, N., Kotsev, V., Toma, I.: The linked data benchmark council: a graph and RDF industry benchmarking effort. ACM SIGMOD Record 43(1), 27–31 (2014)
4. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a web of open data. Springer (2007)
5. Bizer, C., Schultz, A.: The berlin SPARQL benchmark. IJSWIS (2009)
6. Blum, D., Cohen, S.: Grr: generating random rdf. In: Extended Semantic Web Conference. pp. 16–30. Springer (2011)
7. Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., Keppmann, F.L., Miranker, D., Sequeda, J.F., Wylot, M.: NoSQL databases for RDF: An empirical evaluation. ISWC pp. 310–325 (2013)
8. 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)
9. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 145–156. ACM (2011)
10. Erling, O., Mikhailov, I.: Rdf support in the virtuoso dbms. In: Networked Knowledge-Networked Media, pp. 7–24. Springer (2009)
11. Graux, D., Jachiet, L., Genevès, P., Layaïda, N.: SPARQLGX: Efficient Distributed Evaluation of SPARQL with Apache Spark. To appear in ISWC (2016)
12. Group, W.S.W., et al.: SPARQL 1.1 overview (2013), http://www.w3.org/TR/sparql11-overview/
13. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Web Semantics (2005)

[5] Public repository: `https://github.com/tyrex-team/sparub`, it also contains demonstration scenarios (with already installed triplestores and considering popular benchmarks) that are easy to deploy using Docker [17].

14. Harris, S., Lamb, N., Shadbolt, N.: 4store: The design and implementation of a clustered RDF store. SSWS (2009)
15. Hayes, P., McBride, B.: RDF semantics. W3C recommendation 10 (2004), www.w3.org/TR/rdf-concepts/
16. Jena, A.: A free and open source java framework for building semantic web and linked data applications. Available online: jena. apache. org/(accessed on 28 April 2015) (2015)
17. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux Journal 2014(239), 2 (2014)
18. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL Benchmark – Performance assessment with real queries on real data. ISWC pp. 454–469 (2011)
19. Punnoose, R., Crainiceanu, A., Rapp, D.: RYA: a scalable RDF triple store for the clouds. In: International Workshop on Cloud Intelligence. p. 4. ACM (2012)
20. Qiao, S., Özsoyoğlu, Z.M.: Rbench: Application-specific RDF benchmarking. In: SIGMOD. pp. 1825–1838. ACM (2015)
21. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: a SPARQL performance benchmark. ICDE pp. 222–233 (2009)
22. Seaborne, A., Manjunath, G., Bizer, C., Breslin, J., Das, S., Davis, I., Harris, S., Idehen, K., Corby, O., Kjernsmo, K., et al.: Sparql/update: A language for updating rdf graphs. W3c member submission 15 (2008)
23. Zhang, Y., Duc, P., Corcho, O., Calbimonte, J.P.: Srbench: a streaming rdf/sparql benchmark. The Semantic Web–ISWC 2012 pp. 641–657 (2012)