Benchmarking RDF Metadata Representations: Reification, Singleton Property and RDF*

Fabrizio Orlandi(✉) ⓘ, Damien Graux ⓘ, Declan O'Sullivan ⓘ

ADAPT SFI Research Centre, Trinity College Dublin, Dublin, Ireland

Abstract—**RDF reification is a data modelling solution for writing RDF statements about RDF statements. A set of approaches exist in the literature to express statement-level metadata, or "reified" statements, in RDF. They have primarily been designed to attach additional contextual information to individual triples, such as provenance, spatio-temporal validity, or certainty. Practically, different methods have been developed to present such metadata while complying with the RDF standard and syntax. However, when effectively stored in triplestores, these various solutions produce diverse results in terms of querying performance, storage efficiency and usability. In this article, we analyse these metrics across three popular reification approaches: Standard Reification, Singleton Property and the recent RDF*. We publish our benchmark evaluation resources so that the community could expand and compare the study on different datasets, reification approaches and storage implementations.**

I. INTRODUCTION

The ability to express n-ary relations in RDF has always interested the Semantic Web research community since the creation of the first RDF specifications [1].[1] This feature, often called "reification" or "metadata representation" or "statement-level metadata", allows practitioners to express statements about statements. Which is useful, for instance, when attaching provenance [2], versioning [3] and ownership of facts on a statement-level and directly into a dataset [4].

As a consequence, different RDF metadata representation approaches exist: from standard reification to named graphs to singleton property. However, these strategies lead, in practice, to different data models, representations and number of RDF triples generated [5] (*e.g.* standard reification requires the use of four additional triples for each reified statement). Making it difficult to choose the most appropriate solution for the use case at hand. Moreover, in order to help users generating and maintaining their RDF statements of statements, Hartig *et al.* recently introduced the RDF* & SPARQL* syntax [6], [7]. This solution has rapidly gained attention and popularity in the last couple of years, mainly due to its simplicity in expressing statement-level metadata and a wider adoption by triplestore vendors.

For these reasons, there is a need for a benchmark that allows users to compare different reification approaches and different triplestore implementations. In this paper, we introduce REF, the RDF REiFication benchmark. It provides resources, namely datasets and queries, that can be used to test the performance of different metadata representation solutions, including the recent RDF*. Performance can be measured especially in terms of query execution time, usability, storage size, adherence to standards and support by existing implementations.

Finally, in addition the benchmark description, we describe some experiments leveraging REF for the evaluation of three different approaches (reification, singleton property, RDF*) in terms of database size and query execution time. The evaluation shows a clear difference between the approaches and demonstrates the usefulness of the benchmark.

We made all the resources easily accessible online. RDF/RDF* data dumps and SPARQL/SPARQL* queries, along with instructions, are all published on Zenodo, with a persistent URI:[2]

`<https://doi.org/10.5281/zenodo.3894745>`

The rest of the paper is organised as follows. Section II introduces the different reification approaches considered by REF. Section III and IV describe the REF benchmark resources in terms of data and queries respectively. Then, Section V provides details on our experiments, before concluding the paper with related work and conclusions.

II. EXPRESSING STATEMENTS OF STATEMENTS IN RDF

By construction, RDF statements [8] can only represent 'flat' information under the form of triples `subject predicate object`. It implies that both facts and their metadata cannot be represented at the same level, meaning that "Tolkien wrote the Lord of the Rings according to Wikipedia" could be represented as:

```
:Tolkien :wrote :LOTR .
:Fact    :says  :Wiki .
```

Obviously, these two RDF triples are not so far related together as they do not share a common identifier. During the past two decades, several methods have been developed in order to be able to express meta-facts about already existing triples. For example, some complex architecture can be designed to have the data split into silos: some for *real*-data and others about *meta*-data; such a paradigm implies to deal with several

---

Send correspondence to F. Orlandi, E-mail: orlandif@tcd.ie

[1]"Reifying RDF (properly), and N3", Tim Berners-Lee, 2005: https://www.w3.org/DesignIssues/Reify.html

[2]While more details about the REF Benchmark and the queries are also available on GitHub: https://github.com/dgraux/RDFStarObservatory/tree/master/testSuits/REF-Benchmark

RDF named graphs and led to the design of quad-stores where an additional piece of information carries the context (`subj pred obj context`).

In this study, we restrict our scope to methods that directly deal with triples *i.e.* to methods that set up strategies to add the metadata as triples within the knowledge graph. Until now, three approaches have been mainly described: *standard reification*, *singleton property* and *RDF**.

A. Standard Reification [8]

The *reification* method has been proposed together with the RDF primer standardised by the W3C [8] and it relies on the fact that RDF allows the representation of resources which do **not** have URIs: *i.e.* the *blank nodes*. Practically, the idea is to "step back" from the way information is expressed in RDF by using blank nodes to refer to high-level `statements` and then to split the information into pieces according to the role each part of a statement has, such as its `subject`, its `predicate`, its `object` and its `metadata`. Reconsidering our "Lord of the Rings" example, we could reify it as follows:

```
_:x rdf:type      rdf:Statement .
_:x rdf:subject   :Tolkien .
_:x rdf:predicate :wrote .
_:x rdf:object    :LOTR .
_:x :says         :Wiki .
```

This representation projects all the information on the same level and creates blocks of triples sharing the same blank node allowing therefore engines to process them together. However, such a representation creates a large number of triples, resulting in larger graphs posing scalability challenges. And in addition, relying exclusively on blank nodes makes the identification tasks harder since the SPARQL engines will have to "dig" down to reach the object of the `rdf:subject` predicate in order to know what entity is referred by a blank node `_:x`, resulting in more complex processes.

B. Singleton Property [5]

In 2014, Nguyen *et al.* proposed an alternative approach [5] which would mainly rely on the predicates instead of the subjects (as for the standard reification). In a nutshell, the *singleton property* method amends the original predicate of a statement with a unique predicate, in order to carry meta-information in additional triples later. For instance, using the singleton property, our running example becomes:

```
:Tolkien :wrote#1                     :LOTR  .
:wrote#1 rdf:singletonPropertyOf :wrote .
:wrote#1 :says                       :Wiki  .
```

Here, unlike with the reification method which splits the facts by roles, we obtain with the singleton property a set of triples where the original `:wrote` predicate is extended in the original triple into `:wrote#1`. This new predicate is then used as a subject to append additional facts. As a consequence of this approach, the number of additional triples needed to represent a fact and its metadata is reduced, as compared to standard reification. Moreover, the general "shape" of the triples remains the same as the method slightly amends the original predicate, leaving the rest of the triples as intuitively expected. As a downside, this method generates a high number of unique predicates, equal to the number of reified statements. This might create problems with standard indexing strategies adopted by triplestores.

C. RDF* [7]

While both standard reification and singleton property stay within the scope of the RDF standard, Hartig & Thomson suggested to extend the standard's syntax in order to allow RDF graph nesting [6]. Practically, their extension –RDF*– offers a way of considering RDF graphs either as subjects or objects of RDF triples. The syntax provides therefore the possibility of recursively embedding graphs into graphs. The various levels of knowledge provide to RDF*-graphs a structure where hypernodes can carry the lower-level information and higher-edges encode metadata. Typically, our Tolkien-based example is encoded using only **one** RDF* statement:

```
<< :Tolkien :wrote :LOTR >> :says :Wiki .
```

Practically, the subject, included between `<<` and `>>`, is itself an RDF sub-graph for the base information and this sub-graph is completed by a provenance statement ":`says :Wiki`". Notwithstanding, the use of a new syntax extension requires a specific implementation of RDF engines and, therefore, limits the adoption of this approach.

III. DATASET DESCRIPTION

In order to provide a benchmark resource for testing the different reification methods we selected a real and already existing dataset. The dataset used for the experiments in this paper is the Biomedical Knowledge Repository (BKR) dataset [10] by the U.S. National Library of Medicine. It has been selected in order to make our results comparable (at least partially) with previous work by Nguyen *et al.* [5]. In their work, the authors compare their proposed approach, Singleton Property (SP), against Standard Reification.

Dataset	BKR-R	BKR-S	BKR-*
Triples (x10^6)	175.6	100.9	61.0

TABLE I: Size of the datasets in millions of triples.

Metric	BKR-R	BKR-S	BKR-*
N. Distinct "?s"	39,919,982	35,661,664	27,402,244
N. Distinct "?p"	6	33,630,338	674
N. Distinct "?o"	8,035,300	5,999,412	8,032,090
N. Distinct "?d"	—	—	1
N. Distinct "?e"	—	—	3,968,595
N. Distinct Classes		2,032,679	

TABLE II: Datasets statistics, where the indicated variables correspond to the SPARQL triple patterns {?s ?p ?o.} for BKR-R & BKR-S and {⟨⟨?s ?p ?o⟩⟩ ?d ?e.} for BKR-*.
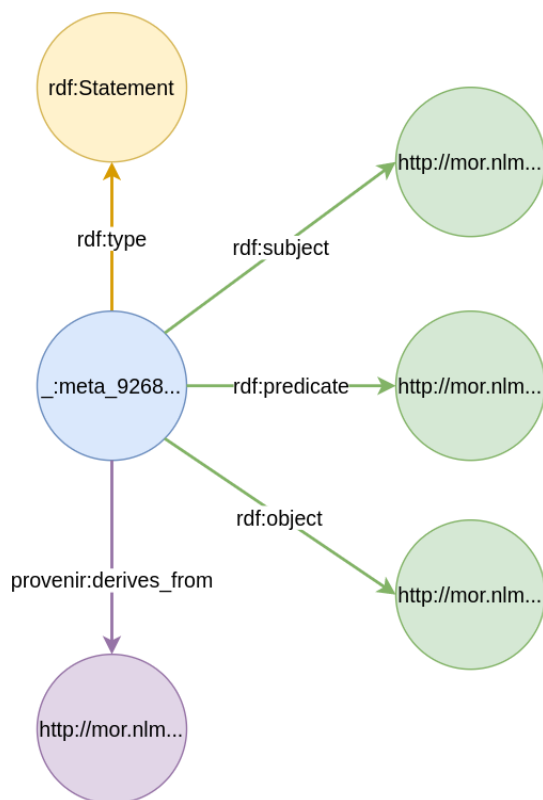
Fig. 1: Example of a typical RDF molecule contained in the BKR dataset, represented using standard RDF reification.

The dataset is a biomedical knowledge graph containing over 30 million semantic statements extracted from PubMed[3] abstracts in addition to the Unified Medical Language System[4] (UMLS). Originally published in 2010 [10] [11], it has been used by Nguyen *et al.* in 2014 to evaluate query efficiency of Singleton Property. We collected the original BKR dataset from [5] via the Internet Archive[5], as the dataset was not available at the original URL indicated in the paper.

The original data is represented in RDF using the Singleton model, serialised in N-Triples format. We converted it to the Standard Reification model and also into RDF*, using the available "RDFstarTools" developed in Java by O. Hartig *et al.*[6]. As a result of the different data models, the BKR dataset modelled using reification (BKR-R) has 175.6 million triples in total; the one based on singleton property (BKR-S) has 100.9 million triples; the RDF* dataset (BKR-*) amounts to 61.0 million total triples (Table I).

The BKR dataset includes integrated biomedical data from a variety of sources such as biomedical literature and terminological knowledge sources such as the Unified Medical Language System (UMLS). However the structure is quite simple and can be illustrated as in Figure 1. Each biomedical

statement is reified to express the source where it has been derived from, hence tracking provenance of statements using `provenir:derived_from`. In Table II we provide some statistics on the three BKR datasets provided in our benchmark. As we can see from the figures, the Singleton Property model leads to a large amount of distinct properties. Which can be an issue for triplestores whose indexes are not optimised for a large number of predicates. At the same time, all the predicates in a KG using standard Reification end up as objects of the `rdf:predicate` property. As regards RDF*, we note that in BKR the only predicate adding extra contextual information to the statements is `provenir:derives_from`. This is used to add provenance information to each "regular" binary statement in the BKR graph.

IV. CONSIDERED QUERIES

We use three representative series (A, B and F) of provenance queries in this REF benchmark, to evaluate the query performance on the three datasets (BKR-R, BKR-S and BKR-*). Series A is obtained from the experiments conducted in [10] by Sahoo *et al.* All 4 queries of set A include one block of provenance-specific triple patterns related to one data triple only. Therefore, Nguyen *et al.* [5] created series B with longer queries, where the lengths of actual data triple patterns (excluding the metadata level) range from 1 to 3. Although the length of these triple patterns looks small, their corresponding SPARQL query patterns are relatively long, *i.e.* up to 21 SPARQL triple patterns for BKR-R using reification. Considering that some of the originally designed queries did not work correctly on the BKR datasets[7] we decided to recreate them and add a few new ones (series F) to our benchmark. For more details on series A and B queries we refer to [5]. All the queries have been included in the Zenodo archive of our REF benchmark, linked in Section I.[8]

Our *series F* query patterns have been designed with additional focus on SPARQL* query patterns and include a combination of both short and long queries. They have been designed by our research group considering the use-case of representing (and querying) provenance (or contextual/temporal information) at a statement-level. They represent realistic queries that one could perform on BKR, or a similar dataset, when retrieving statement-level provenance information and/or comparing provenance metadata for different statements (triples). Moreover, the queries have been designed to test the different internal representations and indexes that triplestores might have for RDF* or similar data models. For instance, some queries (*e.g.* F-Q3) are computationally expensive and some require a comparison (filter) between many graph patterns (*e.g.* F-Q5). In Figure 2 the five "F" queries are depicted following

---

[3]https://pubmed.ncbi.nlm.nih.gov/

[4]https://www.nlm.nih.gov/research/umls/index.html

[5]https://web.archive.org/web/20190622010440/http://wiki.knoesis.org/index.php/Singleton_Property

[6]https://github.com/RDFstar/RDFstarTools

[7]Since June 2019 they are not available online anymore, only accessible via the Internet Archive: https://web.archive.org/web/20190622010440/http://wiki.knoesis.org/index.php/Singleton_Property. Moreover, the original queries used named graphs, which were not included in the data dumps obtained via the Internet Archive.

[8]All the queries are also available on our GitHub repository at: https://github.com/dgraux/RDFStarObservatory/tree/master/testSuits/REF-Benchmark/BKR
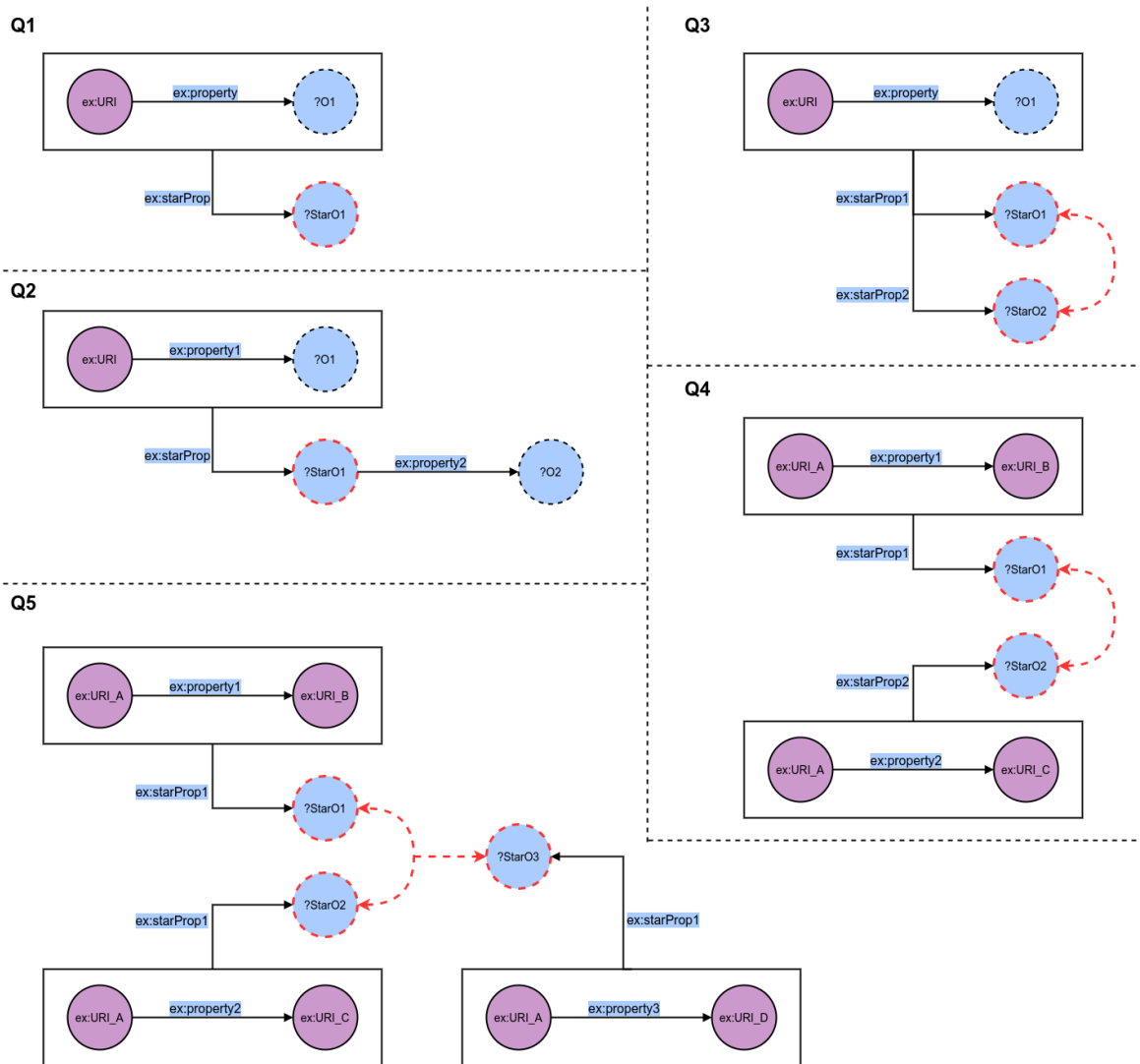
Fig. 2: Structure of our proposed benchmark queries (series F), in addition to the queries (series A and B) proposed by Nguyen *et al.* [5] (using QueryVOWL notation [9]).

the QueryVOWL notation. The figure shows the query patterns in their SPARQL* version only. The red dashed line indicates where a SPARQL "FILTER" clause is applied, either with a "regex" filter on ?StarO1 (for Q1 & Q2), or with a comparison filter (*i.e.*, comparing ?StarO1 with ?StarO2 in Q3 and Q4 and additionally with ?StarO3 in Q5).

Only very recently, a community of researchers and practitioners has started an RDF* Community Group as part of the W3C *RDF dev* community group.[9] The REF benchmark and the series of queries proposed in this article are already covering some of the use cases proposed by this community: mainly the use cases related to natural representation of "property graph" data in RDF and annotation of triples with contextual metadata. The use cases and documents of this

community group would be very relevant for driving the design of the proposed benchmark, however, at present they are still being defined. In the near future, we plan to align our efforts with the RDF* group and extend the benchmark in order to cover the use cases collected by the community.

V. EXPERIMENTS

In this section we report on some experiments conducted to compare three RDF reification approaches including standard reification, singleton property and RDF*. These experiments have been performed on the proposed REF benchmark to show its usefulness. They are not meant as an exhaustive and comprehensive set of experiments, as this would be beyond the scope of this paper. In this regard, for completeness, several triplestores would need to be tested using the proposed benchmark and the same experimental setup. However, even using one triplestore, we show that there is a considerable difference between the three approaches and data representations for

---

[9]In November 2020, the RDF* Community Group started collecting specifications, test cases and use cases for RDF* available at: https://w3c.github.io/rdf-star/
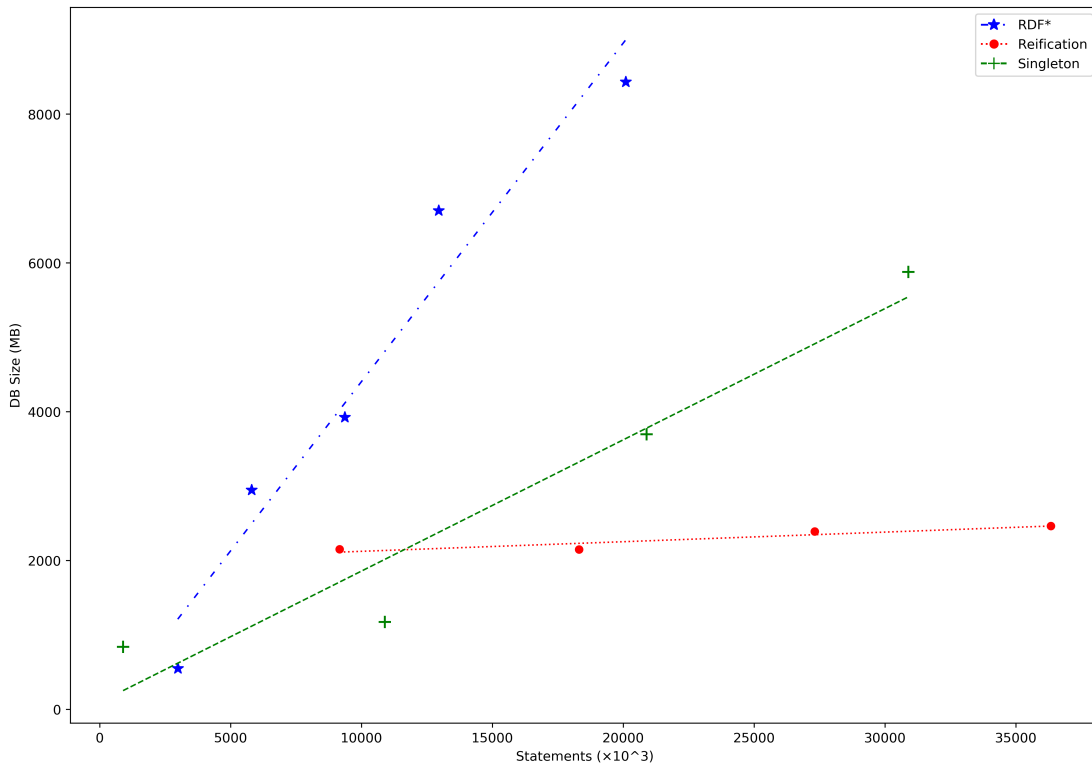
Fig. 3: Growth of the database, in MB, for the three reification methods with increasing number of triples inserted.

statement-level metadata, especially in terms of storage size and query efficiency. Our experiments are based on three main quantitative criteria: number of triples (described in Sec. III), storage size and query execution time.

For reproducibility, the datasets and the queries used in the experiments are the same as the REF benchmark published on Zenodo, with a persistent URI[10] (see Section I). As for the experiments setup, the three datasets were loaded on a Stardog triplestore running on a single node VM with a 4-cores CPU and 32GB of main memory (8GB of JVM memory and 20GB of direct memory allocated for Stardog version 7.3). We have chosen Stardog for these first experiments as it was one of the first triplestores with native support for RDF*. Stardog has been running on a basic Debian VM with no extra services running in parallel. More details on the SPARQL queries are reported in Section IV.

A. Storage Size

While in terms of number of triples the three RDF metadata representations present obvious differences (Section III). Singleton Property shows approximately 40% reduction in the number of triples, compared to Reification. RDF* shows almost a 70% reduction in the number of triples compared to Reification. However, in terms of database size, including indexes and different data structures, the comparisons could be much different. For this reason, we performed an experiment

---

[10]More details about the REF Benchmark and the queries are also available on GitHub: https://github.com/dgraux/RDFStarObservatory/tree/master/testSuits/REF-Benchmark

by measuring the size of the database, in megabytes, with an increasing number of triples being added to each of the three KGs.

The results are reported in Figure 3. The considerable differences shown in this chart are due to the different compression strategies and indexes adopted by Stardog, which are probably optimised more for the standard reification case. It is clear that, with Stardog, approaches such as singleton and RDF* cause the database size to increase at a much faster rate than reification.

B. Query Execution Time

Different RDF metadata representations affect not only the way queries are designed (with RDF*/SPARQL* being the most easy to use "syntactic sugar" for a knowledge engineer) but also the query execution time. This is because data can be represented and stored internally in a triplestore in different ways and SPARQL/SPARQL* engines might be able to use different strategies for query optimisation. This has been shown already in the past in the state-of-the-art [12], [13]. We propose the REF Benchmark also for evaluating this aspect of metadata representations, and for the first time we can use it to compare the performance of different RDF* implementations against "traditional" models.

For these experiments we use all the queries from sets A, B and F described in Section IV. Each query has been run on the respective BKR dataset multiples times. We recorded the time of the first ("cold") execution of a query, with empty cache, and the average time of the subsequent four ("warm")
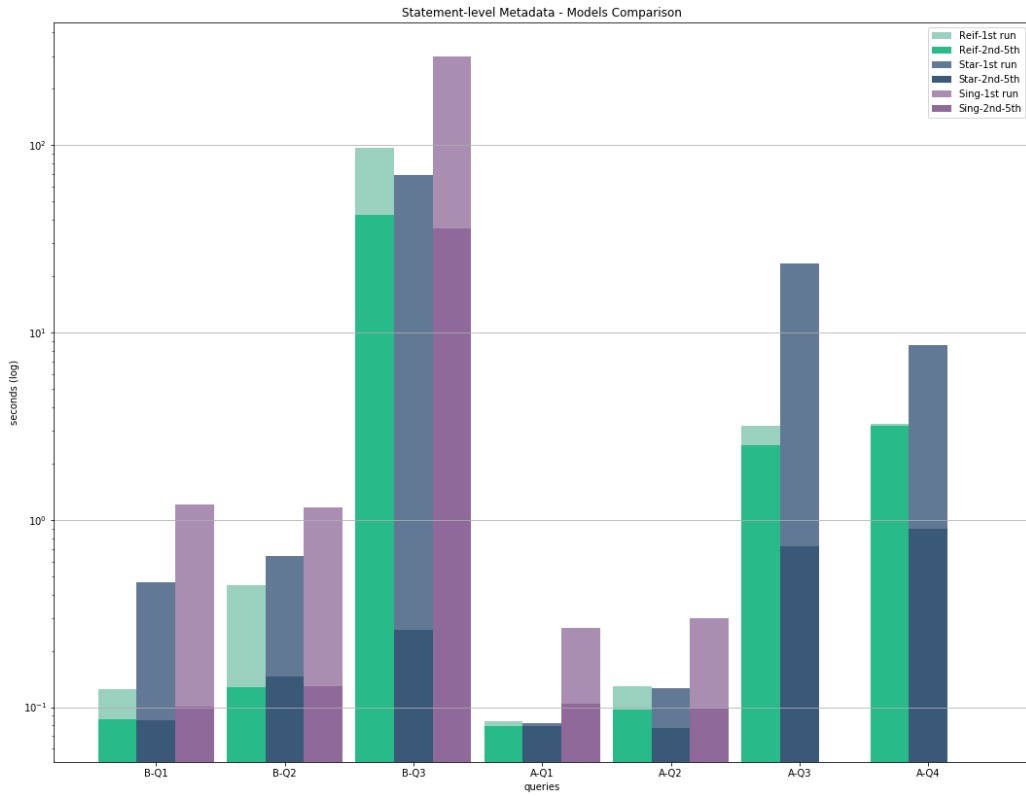
Fig. 4: Query execution time (in seconds) for all the models and the query series A & B by Nguyen *et al.* (log scale).
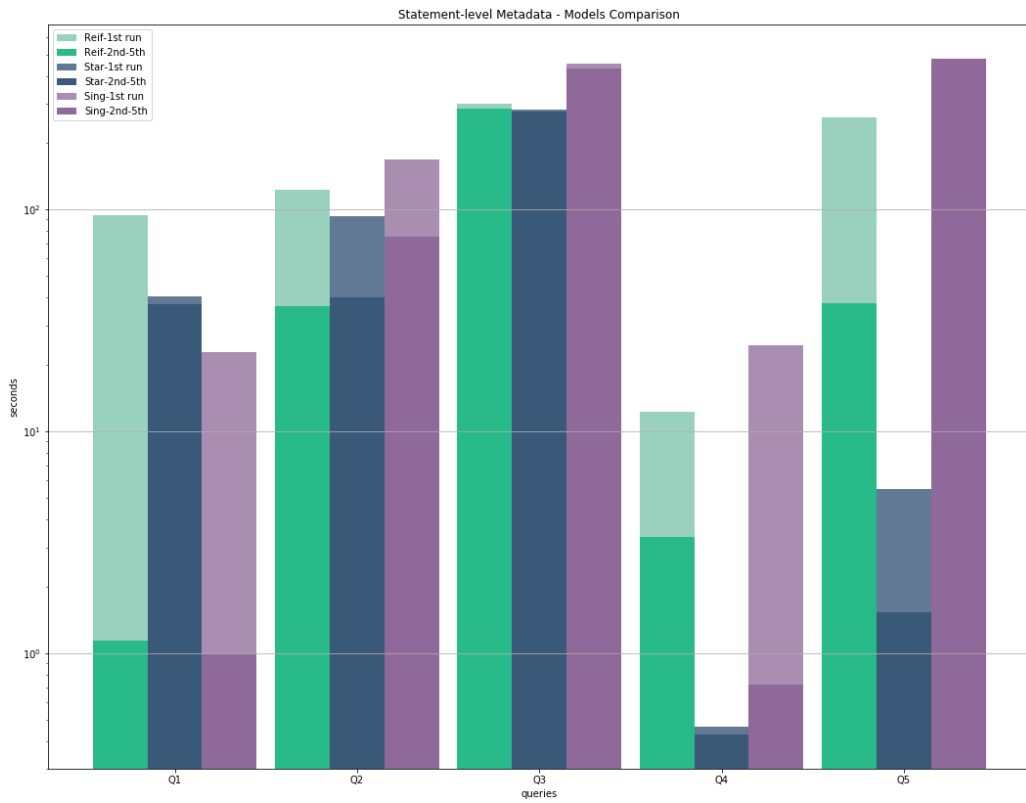

Fig. 5: Query execution time (in seconds) for all the models and our proposed series F queries (log scale).

executions of the same query. The results are reported in Figure 4 for query sets A & B and in Figure 5 for query set F. Already with these initial experiments we can observe that there are significant differences between the three approaches. Singleton Property seems to have the worst performance of the three overall. Probably because of the high number of unique properties with long URIs that need to be generated with that method. Additionally, triplestores' indexes are usually optimised for a lower number of distinct predicates than distinct subjects or objects, and this is not the case with Singleton. Interestingly, RDF* shows a greater reduction of execution time after the first "cold" run, as compared to standard reification (*e.g.* see B series queries and A-Q3, A-Q4). For more complex queries, that present many more graph patterns in the case of std. reification, we can observe a clear difference in performance. In these cases (see *e.g.* A-Q3, A-Q4, B-Q3, F-Q4, F-Q5) RDF* outperforms standard reification. While the opposite happens with simpler queries like B-Q1 and F-Q1.

C. Discussion

While these experiments are not exhaustive, and are only based on one triplestore, they show that the proposed REF benchmark could be relevant for the research community for testing different metadata representations. The benchmark could also be used by triplestore vendors to improve their performance, especially for the newer RDF* solutions. Interestingly, by performing these experiments we also discovered different behaviour and internal representations between different triplestores implementing RDF*. We reported those issues in [14] and created a repository for observing and reporting on such issues[11].

VI. RELATED WORK

Various methods exist to express statements over a set of RDF triples. More technically, we could say that these methods are used to define n-ary relations (not only binary) in RDF and/or OWL [1]. Different solutions, or reification methods, have been designed by experts in the community. The most popular ones being (see Section II for more details): standard reification[12], singleton property [5], named graphs [15] (or quads). However, these strategies usually lead to an extensive amount of additional RDF triples generated [5], *e.g.* standard reification requires the use of four additional triples for each reified statement. In order to facilitate users in creating and managing their RDF statements of statements, Hartig *et al.* introduced the RDF* syntax [6], [7]. This syntactical extension of the RDF standard has been received with enthusiasm by the Semantic Web community[13] and is now implemented by most of the popular SPARQL engines[14].

Following the need of expressing RDF metadata statements and the different existing approaches to do so, researchers have performed extensive comparative analyses. In particular, Frey *et al.* [13] and Hernandez *et al.* [12] conducted detailed comparisons and studying different RDF-based metadata representations such as the ones discussed in our study (see *e.g.* Section II for more details on them) together with other ones like n-ary relations or quad-stores. They both ([12], [13]) design their own methodologies to compare engines and approaches, *e.g.* Hernandez [12] focuses on Wikidata. However, they did not consider RDF* in their discussions.

More generally, multiple benchmarks have been designed to report and compare specific facets of RDF and SPARQL. For example, Aluç *et al.* developed WatDiv to evaluate the behaviour of engines over the conjunctive fragment of SPARQL [16]. On a different note, Guo *et al.* review the OWL inferring capabilities of stores with LUBM [17]. Alternatively, the BEAR benchmark of Fernández *et al.* aims at comparing storage strategies for RDF archives [18]. Therefore, our proposed benchmark completes the set of available benchmarks for the community by covering the reification aspects of RDF.

Finally, these benchmarks allow the community to conduct extensive comparative analyses such as [19] for the distributed SPARQL engines, or [20] to compare SPARQL engines according to use-case criteria. Our benchmark offers, for instance, a method to extend the discussions which have already been started in [14] about RDF* implementations of commercial RDF stores.

VII. CONCLUSION AND FUTURE WORK

In this paper we introduced **REF**, the first benchmark for RDF REiFication approaches. It consists of a set of equivalent SPARQL (and SPARQL*) queries and the corresponding RDF (and RDF*) datasets, for different metadata representations. These can be used by the community as a reference for testing the performance of different storage solutions. The data used in this release of the benchmark is derived from the Biomedical Knowledge Repository (BKR) project. We translated the original BKR dataset, which was available in RDF using the singleton property reification model, into standard RDF reification and RDF*. We performed some experiments using these resources to show the effectiveness of the benchmark in evaluating the performance of a triplestore, *e.g.* in terms of storage size and query execution time. Our experiment shows considerable differences between the three approaches even on one single SPARQL engine. In the future, we plan to add new data and queries to the benchmark and use it to compare different triplestores.

---

[11]https://github.com/dgraux/RDFStarObservatory

[12]https://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification

[13]The *W3C Workshop on Web Standardization for Graph Data* (2019) has set a direction to bridge the Semantic Web and Property Graph communities together indicating RDF* as a viable option. https://www.w3.org/Data/events/data-ws-2019/index.html. In addition, a W3C working group is currently establishing a *use cases and requirements* list for RDF* https://w3c.github.io/rdf-star/UCR/rdf-star-ucr.html.

[14]https://github.com/dgraux/RDFStarObservatory

REFERENCES

[1] P. Hayes, J. Carroll, C. Welty, M. Uschold, B. Vatant, F. Manola, I. Herman, and J. Lawrence, "Defining N-ary Relations on the Semantic Web," *W3C Working Group Note*, 2006. [Online]. Available: http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

[2] F. Orlandi and A. Passant, "Modelling provenance of DBpedia resources using Wikipedia contributions," *Journal of Web Semantics*, vol. 9, no. 2, pp. 149–164, 2011.

[3] M. Frommhold, R. N. Piris, N. Arndt, S. Tramp, N. Petersen, and M. Martin, "Towards versioning of arbitrary RDF data," in *Proceedings of the 12th International Conference on Semantic Systems*, 2016, pp. 33–40.

[4] J. Frey, K. Müller, S. Hellmann, E. Rahm, and M.-E. Vidal, "Evaluation of metadata representations in RDF stores," *Semantic Web*, vol. 10, no. 2, pp. 205–229, 2019.

[5] V. Nguyen, O. Bodenreider, and A. Sheth, "Don't like RDF reification? Making statements about statements using singleton property," in *Proceedings of the 23rd international conference on World wide web*, 2014, pp. 759–770.

[6] O. Hartig and B. Thompson, "Foundations of an alternative approach to reification in RDF," *arXiv preprint arXiv:1406.3399*, 2014.

[7] O. Hartig, "Foundations of RDF* and SPARQL* (an alternative approach to statement-level metadata in RDF)," in *11th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, 2017.

[8] F. Manola, E. Miller, B. McBride *et al.*, "RDF primer," *W3C recommendation*, vol. 10, no. 1-107, p. 6, 2004.

[9] F. Haag, S. Lohmann, S. Siek, and T. Ertl, "QueryVOWL: A visual query notation for linked data," in *Proceedings of ESWC 2015 Satellite Events*, ser. LNCS, vol. 9341. Springer, 2015, pp. 387–402.

[10] S. S. Sahoo, O. Bodenreider, P. Hitzler, A. Sheth, and K. Thirunarayan, "Provenance context entity (PaCE): Scalable provenance tracking for scientific RDF data," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[11] S. S. Sahoo, V. Nguyen, O. Bodenreider, P. Parikh, T. Minning, and A. P. Sheth, "A unified framework for managing provenance information in translational research," *BMC Bioinformatics*, 2011.

[12] D. Hernández, A. Hogan, and M. Krötzsch, "Reifying RDF: What works well with wikidata?" in *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015)*, vol. 1457. CEUR-WS, 2015, pp. 32–47. [Online]. Available: http://ceur-ws.org/Vol-1457/

[13] J. Frey, K. Müller, S. Hellmann, E. Rahm, and M. E. Vidal, "Evaluation of metadata representations in RDF stores," *Semantic Web*, vol. 10, no. 2, pp. 205–229, 2019.

[14] F. Orlandi, D. Graux, and D. O'Sullivan, "How many stars do you see in this constellation?" in *Proceedings of ESWC 2020 Satellite Events*, 2020.

[15] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler, "Named graphs," *Journal of Web Semantics*, vol. 3, no. 4, pp. 247–267, dec 2005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1570826805000235

[16] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, "Diversified stress testing of RDF data management systems," in *International Semantic Web Conference*. Springer, 2014, pp. 197–212.

[17] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Journal of Web Semantics*, vol. 3, no. 2-3, pp. 158–182, 2005.

[18] J. D. Fernández, J. Umbrich, A. Polleres, and M. Knuth, "Evaluating query and storage strategies for RDF archives," *Semantic Web*, vol. 10, no. 2, pp. 247–291, 2019.

[19] Z. Kaoudi and I. Manolescu, "RDF in the clouds: a survey," *The VLDB Journal*, vol. 24, no. 1, pp. 67–91, 2015.

[20] D. Graux, L. Jachiet, P. Geneves, and N. Layaïda, "A multi-criteria experimental ranking of distributed SPARQL evaluators," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 693–702.