WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

BEAR: Benchmarking the Efficiency of RDF Archiving

Javier D. Fernández
Jürgen Umbrich
Axel Polleres

Arbeitspapiere zum Tätigkeitsfeld
Informationsverarbeitung, Informationswirtschaft und Prozessmanagement
Working Papers on Information Systems, Information Business and Operations

Nr./No. 02/2015

Herausgeber / Editor:
Department für Informationsverarbeitung und Prozessmanagement
Wirtschaftsuniversität Wien · Welthandelsplatz 1 · 1020 Wien
Department of Information Systems and Operations · Vienna University of Economics and Business · Welthandelsplatz 1 · 1020 Vienna

BEAR: Benchmarking the Efficiency of RDF Archiving

Javier D. Fernández, Jürgen Umbrich, Axel Polleres

Vienna University of Economics and Business, Vienna, Austria
{javier.fernandez,juergen.umbrich,axel.polleres}@wu.ac.at

Abstract. There is an emerging demand on techniques addressing the problem of efficiently archiving and (temporal) querying different versions of evolving semantic Web data. While systems archiving and/or temporal querying are still in their early days, we consider this a good time to discuss benchmarks for evaluating storage space efficiency for archives, retrieval functionality they serve, and the performance of various retrieval operations. To this end, we provide a blueprint on benchmarking archives of semantic data by defining a concise set of operators that cover the major aspects of querying of and interacting with such archives. Next, we introduce *BEAR*, which instantiates this blueprint to serve a concrete set of queries on the basis of real-world evolving data. Finally, we perform an empirical evaluation of current archiving techniques that is meant to serve as a first baseline of future developments on querying archives of evolving RDF data.

1 Introduction

Nowadays, RDF data is ubiquitous. In less than a decade, and thanks to active projects such as the Linked Open Data (LOD)[3] effort or schema.org, researchers and practitioners have built a continuously growing interconnected Web of Data. In parallel, a novel generation of semantically enhanced applications leverage this infrastructure to build services which can answer questions not possible before (thanks to the availability of SPARQL [12] which enables structure queries over this data). As previously reported by [25, 13], the published data is continuously undergoing changes (on a data and schema level). These changes naturally happen without a centralized monitoring nor pre-defined policy, following the scale-free nature of the Web. Applications and businesses leveraging the availability of certain data over time, and seeking to track data changes or conduct studies on the evolution of data, thus need to build their own infrastructures to preserve and query data over time. Moreover, at the schema level, evolving vocabularies complicate re-use as inconsistencies may be introduced between data relying on a previous version of the ontology, which can lead to failure of a system, such as an inference tool.

Thus, archiving policies of Linked Open Data (LOD) collections emerge as a novel –and open– challenge aimed at assuring quality and traceability of Semantic Web data over time. While sharing the same overall objectives with

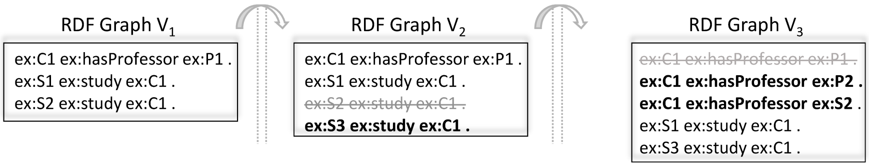

RDF Graph V_1
ex:C1 ex:hasProfessor ex:P1 .
ex:S1 ex:study ex:C1 .
ex:S2 ex:study ex:C1 .
RDF Graph V_2
ex:C1 ex:hasProfessor ex:P1 .
ex:S1 ex:study ex:C1 .
~~ex:S2 ex:study ex:C1 .~~
ex:S3 ex:study ex:C1 .
RDF Graph V_3
~~ex:C1 ex:hasProfessor ex:P1 .~~
ex:C1 ex:hasProfessor ex:P2 .
ex:C1 ex:hasProfessor ex:S2 .
ex:S1 ex:study ex:C1 .
ex:S3 ex:study ex:C1 .

Fig. 1: Example of RDF graph versions.

traditional Web archives, such as the Internet Archive[1], the archives for the Web of Data should offer capabilities for time-traversing and structural queries.

Initial RDF archiving policies/strategies [7] mainly consider three design models, namely independent copies (IC), change-based (CB) and timestamp-based (TB) approaches. In IC, each version is managed as a different, isolated dataset. Conversely, a baseline version and differences (deltas) between consecutive versions are stored in CB. Finally, in TB each statement locally holds the validity of the version, whereas in a range or just denoting modifications (typically version numbers or timestamps of addition and deletion).[2]

In general, these strategies are not implemented at large scale, and existing archiving infrastructures do not support structured and time-traversing queries in demand: for instance, knowing if a dataset, or a particular entity has changed goes beyond the current expressiveness of SPARQL itself, but, notably, also beyond the capability of most temporal extensions of SPARQL [8].

This paper anticipates the development of definitive solutions for archiving and querying Semantic Web data, and tackles the problem of evaluating the efficiency of the required retrieval demands. To the best of our knowledge, no work has been proposed to systematically benchmarking RDF archives. Existing RDF versioning and archiving solutions focus so far on providing feasible proposals for partial coverage of possible use case demands. Somewhat related, but not covering the specifics of (temporal) querying over archives, existing RDF/SPARQL benchmarks focus on static [1, 4, 21], federated [15] or streaming data [5] in centralized or distributed repositories. Thus, they obviate the particularities of RDF archiving, where querying entities across time is a crucial aspect.

In order to fill this gap, we provide foundations for the problem of benchmarking the efficiency of policies and strategies to archive and query evolving semantic Web data. Our main **contributions** are: (i) Based on an analysis of current RDF archiving proposals (Section 2), we provide a theoretical blueprint categorizing the type of retrieval demands and query operators involved (Section 3); (ii) we present *BEAR*, a concrete benchmark that makes use of real-world data snapshots extracted from the Dynamic Linked Data Observatory [13] (Section 4). We describe queries with varying complexity, covering a broad range of archiving use cases; finally, (iii) we have implemented current RDF archiving policies/strategies and evaluate them using *BEAR* to both set a baseline and illustrate our proposal (Section 5).

[1] http://archive.org.

[2] In the course of this paper, whenever we talk about time/validity, we talk about transaction time [10], i.e. time of retrieval of the snapshot of dataset to be archived.

Type / Focus	Materialisation	Structured Queries	
		Single time	**Cross time**
Version	Version Materialisation *-get snapshot at time t_i*	Single-version structured queries *-lectures given by certain teacher at time t_i*	Cross-version structured queries *-subjects who have played the role of student and teacher of the same course*
Delta	Delta Materialisation *-get delta at time t_i*	Single-delta structured queries *-students leaving a course between two consecutive snapshots, i.e. between t_{i-1}, t_i*	Cross-delta structured queries *-evolution of added/deleted students across versions*

Table 1: Classification and examples of retrieval needs.

2 Preliminaries

We briefly summarise the necessary findings of our previous work in which we surveyed current archiving techniques for dynamic Linked Open Data [7]. The use case is depicted in Figure 1, which shows an RDF archive with three versions (a formal definition on archives and versions is provided in Section 3): the original version V_1 models two students *ex:S1* and *ex:S2* of a course *ex:C1*, whose professor is *ex:P1*. In the second version, the *ex:S2* student disappeared in favour of a new student, *ex:S3*. In the last version, the former professor *ex:P1* leaves the course to a new pair of professors: a new professor *ex:P2* and the former student *ex:S2* who reappears now playing the role of a professor.

2.1 Retrieval Functionality

Given the relative novelty of archiving and querying evolving semantic Web data, retrieval needs are neither fully described nor broadly implemented in practical implementations (described below). First categorizations [7, 22] are compiled in Table 1. This classification distinguishes six different types of retrieval needs, mainly regarding the query type (materialisation or structured queries) and the main focus (version/delta) of the query.

Version materialisation is a basic demand in which a full version is retrieved. In fact, this is the most common feature provided by revision control systems and other large scale archives, such as current Web archiving that mostly dereferences URLs across a given time point.[3]

Single-version structured queries are queries which are performed on one specific version. One could expect to exploit current state-of-the-art query resolution in RDF management systems, with the additional difficulty of maintaining and switching between all versions.

Cross-version structured queries, also called time-traversal queries, add a novel complexity since these queries must be satisfied across different versions.

Delta materialisation retrieves the differences (deltas) between two or more given versions. This functionality is largely related to RDF authoring and other operations from revision control systems (merge, conflict resolution, etc.). There exist several approaches to compute the deltas between two RDF versions; *low-level* deltas [27] at the level of triples, distinguishing between added (Δ^+) and deleted (Δ^-) triples, or *high-level* deltas [19] which are human-readable explanations (*e.g.* deltas can state that a class has been renamed, and this affects all

[3] See the Internet Archive effort, http://archive.org/web/.

the instances). *High-level* deltas are more descriptive and can be more concise, but this is at the cost of relying on an underlying semantics (such as RDFS or OWL), and they are more complex to detect and manage [28].

Likewise, the **single-delta structured queries** and **cross-delta structured queries** are the counterparts of the aforementioned version-focused queries, but must be satisfied on change instances of the dataset. For instance one could retrieve *students leaving a course between two versions*, or *the evolution in the number of added/deleted students or teachers across several version.*

2.2 Archiving Policies and Retrieval Processing

Main research efforts addressing the challenge of RDF archiving fall in one of the following three storage strategies [7]: *independent copies (IC)*, *change-based (CB)* and *timestamp-based (TB)* approaches.

Independent Copies (IC) [14, 18] is a basic policy that manages each version as a different, isolated dataset. It is, however, expected that IC faces scalability problems as static information is duplicated across the versions. Besides simple retrieval operations such as version materialisation, other operations require non-negligible processing efforts. A potential retrieval mediator should be placed on top of the versions, with the challenging tasks of i) computing deltas at query time to satisfy delta-focused queries, ii) loading/accessing the appropriate version/s and solve the structured queries, and iii) performing both previous tasks for the particular case of structured queries dealing with deltas.

Change-based approach (CB) [27, 6, 29] partially addresses the previous scalability issue by computing and storing the differences (deltas) between versions. For the sake of clarity and simplicity, in this paper we focus on low-level deltas (added or deleted triples). As stated, complementary works tackle high-level delta management [19, 17] but they focus on materialisation retrieval [28].

A query mediator for this policy manages a materialised version and the subsequent deltas. Thus, CB requires additional computational costs for delta propagation which affects version-focused retrieving operations. Different alternatives have been proposed such as computing reverse deltas (storing a materialisation of the current versions and computing the changes with respect to this) or providing full version materialisation in some intermediate steps [6, 22], at the cost of augmenting space overheads.

Timestamp-based approach (TB) can be seen as a particular case of time modelling in RDF [24, 23, 11, 30], where each triple is annotated with its temporal validity. Likewise, in RDF archiving, each triple locally holds the timestamp of the version. In order to save space avoiding repetitions, practical proposals annotate the triples only when they are added or deleted. That is, the triples are augmented by two different fields: the created and deleted (if present) timestamps [16, 26, 9]. This latter constitutes the most practical approaches, which manage versions/deltas under named/virtual graphs, so that the retrieval mediator can rely on existing solutions providing named/virtual graphs. Except for delta materialisation, all retrieval demands can be satisfied with some extra efforts given that i) version materialisation requires to rebuild the delta similarly to CB, and ii) structured queries may need to skip irrelevant triples [16].

3 Evaluation of RDF Archives: Challenges and Guidelines

Previous considerations on RDF archiving policies and retrieval functionality set the basis of future directions on evaluating the efficiency of RDF archives. The design of a benchmark for RDF archives should meet three requirements:

- First, the benchmark should be **archiving-policy agnostic** both in the dataset design/generation and the selected set of queries to do a fair comparison of different archiving policies.
- Early benchmarks should mainly focus on simpler queries against an increasing number of snapshots and introduce complex querying once the policies and systems are better understood.
- While new retrieval features must be incorporated to benchmark archives, one should consider lessons learnt in previous works on RDF data management systems [1].

Besides these particular considerations, in general we briefly recall here the four most important criteria when designing a domain-specific benchmark [10]: Relevancy (to measure the performance when performing typical operations of the problem domain, i.e. archiving retrieval features), portability (easy to implement on different systems and architectures, i.e. RDF archiving policies), scalability (apply to small and large computer configurations, which should be extended in our case also to data size and number of versions), and simplicity.

In the following, we formalize features and challenges on the design of the benchmark data and queries. Most of these features will be instantiated in the next section to provide a concrete experimental testbed.

3.1 Dataset Configuration

We first provide semantics for RDF archives and adapt the notion of *temporal RDF graphs* by Gutierrez et al. [11]. In this paper, we make a syntatic-sugar modification to put the focus on version labels instead of temporal labels. Note, that time labels are a more general concept that could lead to time-specific operators (intersect, overlaps, etc.), which is complementary –and not mandatory– to RDF archives.

Let $\mathcal{N}$ be a set of version labels in which a total order is defined.

Definition 1 (RDF Archive). *A version-annotated triple is an RDF triple* (s, p, o) *with a label* $i \in \mathcal{N}$ *representing the version in which this triple holds, denoted by the notation* $(s, p, o) : [i]$. *An RDF archive graph* $\mathcal{A}$ *is a set of version-annotated triples.*

Definition 2 (RDF Version). *An RDF version of an RDF archive* $\mathcal{A}$ *at snapshot* i *is the RDF graph* $\mathcal{A}(i) = \{(s, p, o) | (s, p, o) : [i] \in \mathcal{A}\}$. *We use the notation* V_i *to refer to the RDF version* $\mathcal{A}(i)$.

As basis for comparing different archiving policies, we introduce four main features to describe the dataset configuration, namely *data dynamicity*, *data static core*, *total version-oblivious triples* and *RDF vocabulary*. The main objective is to precisely describe the important features of the benchmark data; although these blueprint could serve in the process of automatic generation of synthetic benchmark data, this is not addressed in this paper.

Data dynamicity. This feature measures the number of changes between versions, considering these differences at the level of triples (low-level deltas [29]). Thus, data dynamicity is mainly described by the *change ratio* and the *data growth* between versions, which we formally define as follows:

Definition 3 (Version change ratio). *Given two versions V_i and V_j, with $i < j$, let $\Delta^+_{i,j}$ and $\Delta^-_{i,j}$ two sets respectively denoting the triples added and deleted between these versions, i.e. $\Delta^+_{i,j} = V_j \setminus V_i$ and $\Delta^-_{i,j} = V_i \setminus V_j$. The change ratio between two versions denoted by $\delta_{i,j}$, is defined by*

$$\delta_{i,j} = \frac{|\Delta^+_{i,j} \cup \Delta^-_{i,j}|}{|V_i \cup V_j|}.$$

In turn, the insertion $\delta^+_{i,j} = \frac{|\Delta^+_{i,j}|}{|V_i|}$ and deletion $\delta^-_{i,j} = \frac{|\Delta^-_{i,j}|}{|V_i|}$ ratios provide further details on the proportion of inserted and add triples.

Definition 4 (Version data growth). *Given two versions V_i and V_j, having $|V_i|$ and $|V_j|$ different triples respectively, the data growth of V_j with respect to V_i, denoted by, $growth(V_i, V_j)$, is defined by*

$$growth(V_i, V_j) = \frac{|V_i|+|V_j|}{|V_i \cup V_j|}.$$

In archiving evaluations, one should provide details on three related aspects, $\delta_{i,j}$, $\delta^+_{i,j}$ and $\delta^-_{i,j}$, as well as the complementary version data growth, for all pairs of consecutive versions. Note that most archiving policies are affected by the frequency and also the type of changes. For instance, the space requirements for the IC policy increases with the static and the added information ($\delta^+_{i,j}$), but decreases with the number of deletions ($\delta^-_{i,j}$). In contrast, timestamp-based approaches store all changes, hence it is affected by the general dynamicity ($\delta_{i,j}$).

Data static core. The static core of an archive contains the triples that are available in all versions, formally defined as follows:

Definition 5 (RDF archive static core). *For an RDF archive $\mathcal{A}$, the static core $\mathcal{C}_\mathcal{A} = \{(s,p,o) | \forall i \in \mathcal{N}, (s,p,o) : [i] \in \mathcal{A}\}$*

This feature is particularly important for those archiving policies that, whether implicitly or explicitly, represent such static core. In a change-based approach, the static core is not represented explicitly, but it inherently conforms the triples that are not duplicated in the versions, which is an advantage against other policies such as IC. It is worth mentioning that the static core can be easily computed taking the first version and applying all the subsequent deletions.

Total version-oblivious triples. This computes the total number of different triples in an RDF archive independently of the timestamp. Formally speaking:

Definition 6 (RDF archive version-oblivious triples). *For an RDF archive $\mathcal{A}$, the version-oblivious triples $\mathcal{O}_{\mathcal{A}} = \{(s, p, o) | \exists i \in \mathcal{N}, (s, p, o) : [i] \in \mathcal{A}\}$*

This feature serves two main purposes. First, it points to the diverse set of triples managed by the archive. Note that an archive could be composed of few triples that are frequently added or deleted. This could be the case of data denoting the presence or absence of certain information, e.g. a particular case of RDF streaming. Then, the total version-oblivious triples are in fact the set of triples annotated by temporal RDF [11] and other representation policies based on annotation (such as AnQL [30]). Note that these follow a merge strategy so that different annotations for the same triple are merged in an annotation set (which results often in an interval or a set of intervals).

RDF vocabulary. In general, we cover under this feature the main aspects regarding the different subjects (S_A), predicates (P_A), and objects (O_A) in the RDF archive $\mathcal{A}$. Namely, we put the focus on the RDF *vocabulary per version* and the *vocabulary set dynamicity*, defined as follows:

Definition 7 (RDF vocabulary per version). *For an RDF archive $\mathcal{A}$, the vocabulary per version is the set of subjects (S_{V_i}), predicates (P_{V_i}) and objects (O_{V_i}) for each version V_i in $\mathcal{A}$.*

Definition 8 (vocabulary set dynamicity). *The dynamicity of a vocabulary set K, being K one of $\{S, P, O\}$, over two versions V_i and V_j, with $i < j$, denoted by $vdyn(K, V_i, V_j)$ is defined by*

$$vdyn(K, V_i, V_j) = \frac{|(K_{V_i} \backslash K_{V_j}) \cup (K_{V_j} \backslash K_{V_i})|}{|K_{V_i} \cup K_{V_j}|}$$

Likewise, $vdyn^{+}(K, V_i, V_j) = \frac{|K_{V_j} \backslash K_{V_i}|}{|K_{V_i} \cup K_{V_j}|}$ and $vdyn^{-}(K, V_i, V_j) = \frac{|K_{V_i} \backslash K_{V_j}|}{|K_{V_i} \cup K_{V_j}|}$ define the vocabulary set dynamicity for instertions and deletions respectively.

Vocabulary information is important since many RDF management systems use dictionaries to efficiently manage the RDF graphs. Thus, for RDF archiving, concrete figures must be provided on the evolution of the these sets, with special attention to their cardinality and dynamicity, given that particular policies could take advantage of shared RDF dictionaries between versions.

3.2 Design of benchmark queries

As stated, there is neither a standard language to query RDF archives, nor an agreed way for the more general problem of querying temporal graphs. Nonetheless, most of the proposals (such as T-SPARQL [8] and SPARQL-ST [20]) are based on SPARQL modifications.

In general, previous experiences on SPARQL benchmarking show that benchmark queries should report on the query type, result size, graph pattern shape and query atom selectivity. For query federation, selected sources must be drawn. Conversely, for RDF archiving, one should put the focus on query dynamicity, without forgetting the strong impact played by query selectivity in most RDF triple stores and query planning strategies.

We briefly recall here the definition of query cardinality and selectivity, adapting the definition in [2, 1] to RDF archives. Given a SPARQL query Q, being this always a *BGP* hereinafter, the evaluation of Q over a general RDF graph $\mathcal{G}$ results in a bag of solution mappings $[[Q]]_G$, where Ω denotes its underlying set. The function $card_{[[Q]]_G}$ maps each mapping $\mu \in \Omega$ to its cardinality in $[[Q]]_G$. Then, for comparison purposes, we introduce three main features, namely *archive-driven result cardinality and selectivity*, *version-driven result cardinality and selectivity* and *version-driven result dynamicity*, defined as follows.

Definition 9 (Archive-driven result cardinality and selectivity). *The archive-driven result cardinality of Q over the RDF archive $\mathcal{A}$, is defined by*

$$CARD(Q, \mathcal{A}) = \textstyle\sum_{\mu \in \Omega} card_{[[Q]]_A}(\mu).$$

In turn, the archive-driven query selectivity accounts how selective is the query, and it is defined by $SEL(Q, \mathcal{A}) = |\Omega|/|\mathcal{A}|$.

Definition 10 (Version-driven result cardinality and selectivity). *The version-driven result cardinality of Q over a version V_i, is defined by*

$$CARD(Q, V_i) = \textstyle\sum_{\mu \in \Omega_i} card_{[[Q]]_{V_i}}(\mu),$$

where Ω_i denotes the underlying set of the bag $[[Q]]_{V_i}$. Then, the version-driven query selectivity is defined by $SEL(Q, V_i) = |\Omega_i|/|V_i|$.

Definition 11 (Version-driven result dynamicity). *The version-driven result dynamicity of Q over two versions V_i and V_j, with $i < j$, denoted by $dyn(Q, V_i, V_j)$ is defined by*

$$dyn(Q, V_i, V_j) = \tfrac{|(\Omega_i \setminus \Omega_j) \cup (\Omega_j \setminus \Omega_i)|}{|\Omega_i \cup \Omega_j|}$$

Likewise, we define the version-driven result insertion $dyn^{+}(Q, V_i, V_j) = \frac{|\Omega_j \setminus \Omega_i|}{|\Omega_i \cup \Omega_j|}$ and deletion $dyn^{-}(Q, V_i, V_j) = \frac{|\Omega_i \setminus \Omega_j|}{|\Omega_i \cup \Omega_j|}$ dynamicity.

The archive-driven result cardinality is reported as a feature directly inherited from traditional SPARQL querying, as it disregards the versions and evaluates the query over the set of triples present in the RDF archive. Although this feature could be only of peripheral interest, the knowledge of this feature can help in the interpretation of version-agnostic retrieval purposes (e.g. ASK queries).

As stated, result cardinality and query selectivity are main influencing factors for the query performance, and should be considered in the benchmark design and also known for the result analysis. In RDF archiving, both processes require particular care, given that the results of a query can highly vary in different versions. Knowing the version-driven result cardinality and selectivity helps to interpret the behaviour and performance of a query across the archive. For instance, selecting only queries with the same cardinality and selectivity across all version should guarantee that the index performance is always the same and as such, potential retrieval time differences can be attributed to the archiving policy. Finally, the version-driven result dynamicity does not just focus on the number of results, but how these are distributed in the archive *timeline*.

In the following, we introduce six different, foundational query atoms to cover the broad spectrum of emerging retrieval demands in RDF archiving. Rather than providing a complex catalog, our main aim is to reflect the basic atoms allowing to gain specific knowledge on RDF archiving, without harming neither the combination of them in order to serve more complex queries, nor the concrete implementation in existing languages/archiving policies.

Version materialisation, $Mat(Q, V_i)$***:*** it provides the SPARQL query resolution of the query Q at the given version V_i. Formally, $Mat(Q, V_i) = [[Q]]_{V_i}$.

Delta materialisation, $Diff(Q, V_i, V_j)$***:*** it provides the different results of the query Q between the given V_i and V_j versions. Formally, let us consider that the output is a pair of mapping sets, corresponding to the results that are present in V_i but not in V_j, that is $(\Omega_i \setminus \Omega_j)$, and viceversa, i.e. $(\Omega_j \setminus \Omega_i)$.

A particular case of delta materialisation is to retrieve all the differences between V_i and V_j, which corresponds to the aforementioned $\Delta^+_{i,j}$ and $\Delta^-_{i,j}$.

Version Query, $Ver(Q)$***:*** it provides the results of the query Q *annotated* with the version label in which each of them holds. In other words, it facilitates the $[[Q]]_{V_i}$ solution for those V_i that contribute with results.

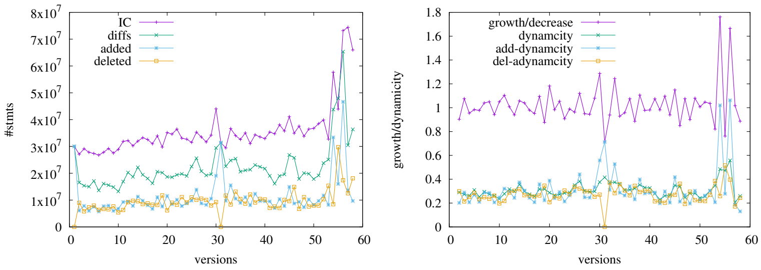
Change checking, $Change(V_i, V_j)$***:*** it answers with a boolean value to state if there is any change between the given V_i and V_j versions. Acknowledging that this operation could be seen as a particular case of delta materialisation (with delta results not null), change checking could be implemented in very different ways depending on the archiving policy, being this of particular importance for some processes such as data monitoring and data synchronization.

Cross-version join, $join(Q_1, V_i, Q_2, V_j)$***:*** it serves the join between the results of Q_1 in V_i, and Q_2 in V_j. Intuitively, it is similar to $Mat(Q_1, V_i) \bowtie Mat(Q_2, V_j)$.

Change materialisation, $Change(Q)$***:*** it provides those consecutive versions in which the given query Q produces different results. In other words, it reports the points in which the query is evaluated differently. Formally, $Change(Q)$ reports the labels i, j (referring to the versions V_i and V_j) $\Leftrightarrow Diff(Q, V_i, V_j) \neq \emptyset, i < j, !\exists k \in \mathcal{N} / i < k < j$.

versions	$\|V_0\|$	$\|V_{57}\|$	$\overline{growth}$	$\overline{\delta}$	$\overline{\delta^-}$	$\overline{\delta^+}$	$\mathcal{C}_\mathcal{A}$	$\mathcal{O}_\mathcal{A}$
58	30m	66m	101%	31%	32%	27%	3.5m	376m

Table 2: Dataset configuration



(a) Number of statements

(b) Relative growth and dynamicity

Fig. 2: Dataset description.

4 BEAR: Benchmark Description

This section presents our BEAR benchmark, which instantiates our blueprints in a real-world scenario. We first detail the benchmark data configuration and the query set covering basic retrieval needs. Next section evaluates the BEAR benchmark on state-of-the-art archiving policies. All the benchmark data and queries, together with the implementation of the policies and additional results are available at the BEAR repository[4].

4.1 Benchmark Data

Although evolving RDF data is ubiquitous, few works systematically provide and maintain a clear and large corpus of RDF versions across time. As such, we build our RDF archive on the data hosted by the Dynamic Linked Data Observatory[5], monitoring more than 650 different domains (containing around 96k RDF documents) across time and serving weekly crawls of these domains. BEAR benchmark data are composed of the first 58 weekly snapshots, i.e. 58 versions, from this corpus.

Each original week consists of triples annotated with their RDF document provenance, in N-Quads format. In this paper we focus on archiving of a single RDF graph, so that we remove the context information and manage the resultant set of unique triples, disregarding duplicates. The benchmark extension to multiple graph archiving can be seen as a future work, as can easily make use of the proposed blueprints.

Dataset configuration. In order to describe our benchmark dataset, we compute and report the data configuration features (presented in our blueprints in

[4] https://github.com/webdata/BEAR
[5] http://swse.deri.org/dyldo/

QUERY SET	lookup position	$\overline{CARD}$	$\overline{dyn}$	#queries
Q_L^S-ϵ=0.2	subject	6.7	0.46	50
Q_L^P-ϵ=0.6	predicate	178.66	0.09	6
Q_L^O-ϵ=0.1	object	2.18	0.92	50
Q_H^S-ϵ=0.1	subject	55.22	0.78	50
Q_H^P-ϵ=0.6	predicate	845.3	0.12	10
Q_H^O-ϵ=0.6	object	55.62	0.64	50

Table 3: Overview of benchmark queries

Section 3) that are relevant for our purposes. Table 2 lists basic statistics of our dataset, further detailed in Figure 2, which shows the figures per version.

As can be seen, data growth behaviour (dynamicity) can be identified at a glance: although the number of statement in the last version ($|V_{57}|$) is more than double the initial size ($|V_0|$), the mean version data growth ($\overline{growth}$) between versions is almost marginal (101%). A closer look to Figure 2 allows to identify that the latest versions are highly contributing to this increase. Similarly, the version change ratios point to the concrete adds and delete operations. Thus, one can see that a mean of 31% of the data change between two versions and that each new version deletes a mean of 27% of the previous triples, and adds 32%. Nonetheless, Figure 2 (b) points to particular corner cases (in spite of a common stability), such as V_{31} in which no deletes are present, as well as it highlights the noticeable dynamicity in the last versions.

Conversely, the number of version-oblivious triples, $376m$, points to a relatively low number of different triples in all the history if we compare this against the number of versions and the size of each version. This fact is in line with the aforementioned add and delete dynamicity values around 30%. The same reasoning applies for the remarkably small static core, $\mathcal{O}_{\mathcal{A}} = 3.5m$.

4.2 Benchmark Queries

The challenging task for every benchmark is to provide meaningful and comprehensive queries which allow to test a wide set of features. As suggested in the previous blueprints (Section 3), we decided to start our RDF archiving benchmark by sampling atomic lookup queries Q, in the form (SVV), (VPV), and (VVO), as this could constitute the basis for further and more complex queries.

In order to provide comparable results, we consider entirely dynamic queries, meaning that the results always differ between consecutive versions. In other words, for each of our selected queries Q, and all the versions V_i and V_j ($i < j$), we assure that $dyn(Q, V_i, V_j) > 0$. To do so, we first extract subjects, predicates and objects that appear in all $\mathit{\Delta}_{i,j}$.

Then, we follow the blueprint suggestions and try to minimise the influence of the result cardinality on the query performance. For this purpose, we sample queries which return, for all versions, result sets of similar size, that is, $CARD(Q, V_i) \approx CARD(Q, V_j)$ for all queries and versions. We introduce here the notation of a ϵ-stable query, that is, a query for which the min and max result cardinality over all versions do not vary by more than a factor of $1 \pm \epsilon$ from

RAW DATA(GZIP)	DIFF (GZIP)	**IC**	**CB**	**TB**
23GB	14GB	225GB	196GB	353GB

Table 4: Space requirements for the different policies.

the mean cardinality, i.e., $\max_{\forall i \in \mathcal{N}} CARD(Q, V_i) \leq (1+\epsilon) \cdot \frac{\sum_{\forall i \in \mathcal{N}} CARD(Q,V_i)}{|\mathcal{N}|}$ and $\min_{\forall i \in \mathcal{N}} CARD(Q, V_i) \geq (1-\epsilon) \cdot \frac{\sum_{\forall i \in \mathcal{N}} CARD(Q,V_i)}{|\mathcal{N}|}$.

Thus, the previous selected dynamic queries are effectively run over each version in order to collect the result cardinality. Next, we split subject, objects and predicate queries producing low (Q_L^S, Q_L^P, Q_L^O) and high (Q_H^S, Q_H^P, Q_H^O) cardinalities. Finally, we filter these sets to sample at most 50 subject, predicate and object queries which can be considered ϵ-stable for a given ϵ. Table 3 shows the selected query sets with their epsilon value, mean cardinality and mean dynamicity. Although, in general, one could expect to have queries with a low ϵ (i.e. cardinalities are equivalent between versions), we test higher ϵ values in objects and predicates in order to have queries with higher cardinalities. Note that even with this relaxed restriction, the number of predicate queries that fulfil the requirements is just 6 and 10 for low and high cardinalities respectively.

5 Implementation and Evaluation

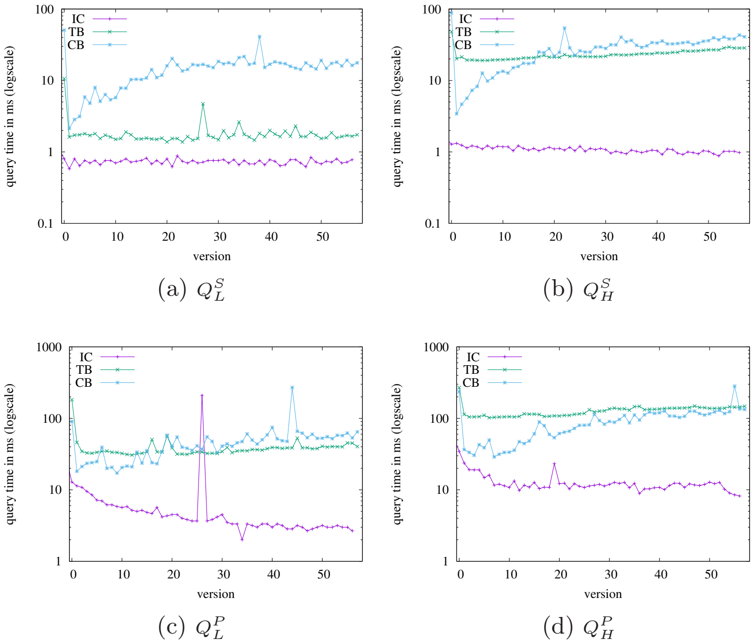
In this section we present our evaluation on the performance of state-of-the-art archiving policies with respect to the retrieval functionality defined in BEAR. To do so, we have developed a proof-of-concept scenario and implemented the IC, CB and TB policies (and their mediators) using Jena‘s TDB store.

For the IC policy, we index each version in an independent TDB instance. Likewise, for the CB policy, we create an index for each added and deleted statements, again for each version and using an independent TDB store. Last, for the TB policy, we followed the approach of [26, 9] and indexed all deltas using named graph in one single TDB instance.

5.1 Space Results

Table 4 shows the required on-disk space for the raw data, the GNU diff and the different policies. As can be seen, raw gzipped data of all 58 versions takes roughly 23G disk space, while storing the diffs information requires 14G. A comparison of these figures against the size of the different policies allows to describe their inherent overheads. Thus, the IC policy indexing (in TDB) requires roughly ten times more disk space than the raw data, mainly due to the data decompression and the built-in TDB indexes. In turn, CB occupies 14 times the diff data but, as expected, it reduces the space needs w.r.t IC (15% less). Finally, TB reports the highest size as it requires 57% and 80% more space than IC and CB respectively. Note that, although CB and TB policies manages the same delta sets, TB uses a unique TDB instance and stores named graph for the triples, so additional “context” indexes are required.

These initial results confirm current RDF archiving scalability problems at large scale, where specific RDF compression emerges as an ideal solution [7].



(a) Q_L^S (b) Q_H^S

(c) Q_L^P (d) Q_H^P

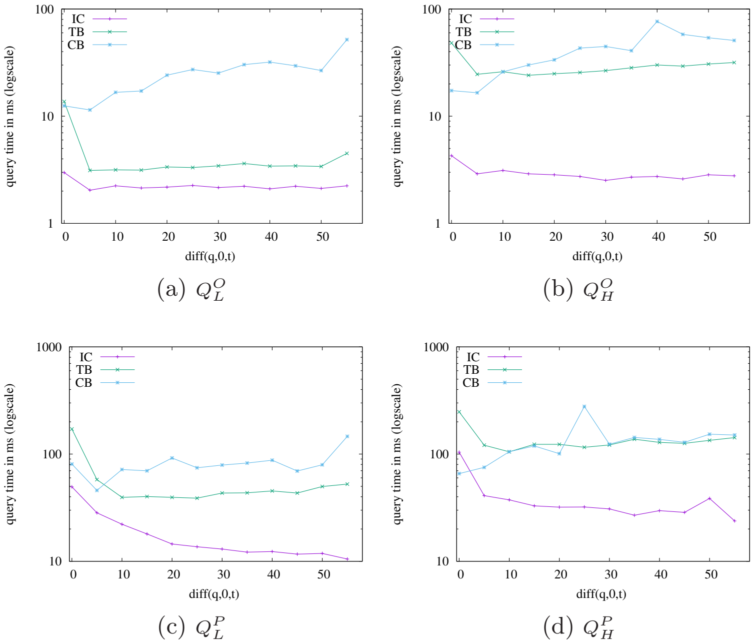
Fig. 3: Query times for subject/object Mat queries.

5.2 Retrieval Performance

Next, we evaluate the overall retrieval performance of the archiving policies. From our blueprints, we chose three exemplary query operations: i) version materialisation, ii) delta materialisation and iii) version queries, and we apply the selected BEAR queries (cf. Section 4.2) as the target query in each case.

In general, our evaluation confirmed our assumptions and assessments about the characteristics of the policies (cf. Section 2). The IC and TB policies show in general a very constant behaviour for all our tests, while the retrieval time of the CB policy decreases if more deltas have to be queried. The main difference between IC and TB is the slightly higher retrieval time of TB due to the larger index size. Next, we present and discuss selected plots for each query operation.

Version materialisation. We measure and compute, for each version, the average query time over all queries in the BEAR query set. The results for the version materialisation queries show very similar patterns for the subject and object query sets. As such, we present in Figure 3 only the results for the subject and predicate queries. We can observe in all plots that the IC policy provides the best and most constant retrieval times. The TB policy has to query more indexed data than the IC policy and as such shows higher time than the IC. The CB policy shows a clear trend that the query performance decreases if we query a higher version since more deltas have to be queried and the adds and delete



(a) Q_L^O

(b) Q_H^O

(c) Q_L^P

(d) Q_H^P

Fig. 4: Query times for $Diff$ queries with increasing intervals.

information processed. Also interestingly, CB and TB have similar performances if the queries have higher cardinalities and higher versions are queried.

Delta materialisation queries. We performed two different experiments. First, we perform diffs for all two consecutive versions, i.e., $diff(Q, V_i, V_{i+1})$ for i in [0,57]. Then, we perform diffs between the initial version and increasing intervals of 5 versions, i.e., $diff(Q, V_0, V_i)$ for i in $\{5, 10, 15, \cdots, 55, 57\}$. For the sake of clarity and space, we omit here the plots for the former evaluation since the results are similar but much more appreciable in the second evaluation. Figure 4 shows again the plots for some selected query sets. Conversely, we observe the expected constant retrieval performance of the IC policy which always needs to query only two version to compute the delta in-memory. We can see that the query time decreases for the CB policy if the intervals of the deltas are increasing, given that more deltas have to be inspected. In turn TB behaves similar than the Mat case given that it always inspects the full TDB instance.

Version queries. Finally, we report the results for the version queries, summarised in Table 5. We report the average query time over each $ver(Q)$ query per BEAR query set. As can be seen, the TB policy outperforms IC and CB policies in contrast to the previous Mat and $Diff$ experiments. This can be explained since both, IC and CB, require to query each version, while TB requires only one query over the full store and then splits the results by version.

Query set	**IC**	**TB**	**CB**
Q_L^S	55.70	27.32	122.44
Q_H^S	71.62	61.46	144.12
Q_L^P	304.67	237.83	412.17
Q_H^P	733.80	362.20	523.90
Q_L^O	40.54	22.78	91.92
Q_H^O	78.18	73.54	136.30

Table 5: Average query time (in ms) for ver(Q) queries

6 Conclusions

RDF archiving is still in an early stage of research. Novel solutions have to face the additional challenge of comparing the performance against other archiving policies or storage schemes, as there is not a standard way of defining neither a specific data corpus for RDF archiving nor relevant retrieval functionalities.

This paper tackles these shortcomings and provide a blueprint to guide future benchmarking of RDF archives. First, we formalize dynamic notions of archives, allowing to effectively describe the data corpus. Then, we describe the main retrieval facilities involved in RDF archiving, and we provide guidelines on the selection of relevant and comparable queries. We instantiate these blueprints in a concise benchmark called BEAR, serving a clean, well-described benchmark corpus, and a query testbed composed of basic, but well-addressed, retrieval queries. Finally, we implement and evaluate three state-of-the-art archiving policies. Results clearly point weakness (specially in scalability) and strengths of current archiving policies, guiding future developments.

We currently focus on exploiting the presented blueprints as basis to generate diverse synthetic benchmark data. Furthermore, we work on novel archiving representations to minimize its representation through compression techniques.

Acknowledgments

The research work of Javier D. Fernández is funded by Austrian Science Fund (FWF): M1720-G11.

References

1. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified Stress Testing of RDF data management systems. In *Proc. of ISWC*, pages 197–212, 2014.
2. M. Arenas, C. Gutierrez, and J. Pérez. On the semantics of sparql. *Semantic Web Information Management*, pages 281–307, 2009.
3. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst*, 5:1–22, 2009.
4. C. Bizer and A. Schultz. The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst*, 5(2):1–24, 2009.

5. D. Dell'Aglio, J. Calbimonte, M. Balduini, O. Corcho, and E. Della Valle. On Correctness in RDF Stream Processor Benchmarking. In *Proc. of ISWC*, pages 326–342, 2013.
6. I. Dong-Hyuk, L. Sang-Won, and K. Hyoung-Joo. A Version Management Framework for RDF Triple Stores. *Int. J. Softw. Eng. Know.*, 22(1):85–106, 2012.
7. J. D. Fernández, A. Polleres, and J. Umbrich. Towards Efficient Archiving of Dynamic Linked Open Data. In *Proc. of DIACHRON*, 2015.
8. F. Grandi. T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *Proc. of ADBIS*, pages 21–30. 2010.
9. M. Graube, S. Hensel, and L. Urbas. R43ples: Revisions for triples. In *Proc. of LDQ*, volume CEUR-WS 1215, paper 3, 2014.
10. J. Gray. *Benchmark handbook: for database and transaction processing systems.* Morgan Kaufmann Publishers Inc., 1992.
11. C. Gutierrez, C. Hurtado, and A. Vaisman. Introducing Time into RDF. *IEEE T. Knowl. Data En.*, 19(2):207–218, 2007.
12. S. Harris and A. Seaborne. *SPARQL 1.1 Query Language.* W3C Recom. 2013.
13. T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing Linked Data Dynamics. In *Proc. of ESWC*, pages 213–227. 2013.
14. M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology versioning and change detection on the web. In *Proc. of EKAW*, pages 197–212. 2002.
15. G. Montoya, M. Vidal, O. Corcho, E. Ruckhaus, and C. Buil Aranda. Benchmarking Federated SPARQL Query Engines: Are Existing Testbeds Enough? In *Proc. of ISWC*, pages 313–324, 2012.
16. T. Neumann and G. Weikum. x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. *Proc. of VLDB Endowment*, 3(1-2):256–263, 2010.
17. N. F. Noy and M. A. Musen. Promptdiff: A Fixed-Point Algorithm for Comparing Ontology Versions. In *Proc. of IAAI*, pages 744–750. 2002.
18. N. F. Noy and M. A. Musen. Ontology Versioning in an Ontology Management Framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.
19. V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. High-level Change Detection in RDF(S) KBs. *ACM Trans. Database Syst.*, 38(1), 2013.
20. M. Perry, P. Jain, and A. P. Sheth. SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries. *Geospatial Semantics and the Semantic Web*, 12:61–86, 2011.
21. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: a SPARQL performance benchmark. In *Proc. of ICDE*, pages 222–233, 2009.
22. K. Stefanidis, I. Chrysakis, and G. Flouris. On Designing Archiving Policies for Evolving RDF Datasets on the Web. In *Proc. of ER*, pages 43–56. 2014.
23. J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *Proc. of ESWC*, pages 308–322. 2009.
24. O. Udrea, D. R. Recupero, and V. S. Annotated RDF. *ACM Transactions on Computational Logic*, 11(2):1–41, 2010.
25. J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, and S. Decker. Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In *Proc. of LDOW*, 2010.
26. M. Vander Sander, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, and R. Van de Walle. R&Wbase: Git for triples. In *Proc. of LDOW*, 2013.
27. M. Volkel, W. Winkler, Y. Sure, S. Kruk, and M. Synak. Semversion: A versioning system for RDF and ontologies. In *Proc. of ESWC*, 2005.

28. F. Zablith, G. Antoniou, M. d'Aquin, G. Flouris, H. Kondylakis, E. Motta, D. Plexousakis, and M. Sabou. Ontology evolution: a process-centric survey. *The Knowledge Engineering Review*, 30(01):45–75, 2015.
29. D. Zeginis, Y. Tzitzikas, and V. Christophides. On Computing Deltas of RDF/S Knowledge Bases. *ACM Transactions on the Web (TWEB)*, 5(3):14, 2011.
30. A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. *Journal of Web Semantics*, 12:72–95, 2012.