Evaluating Query and Storage Strategies for RDF Archives

Javier D. Fernández
Vienna University of Economics and Business, Vienna, Austria
javier.fernandez@wu.ac.at

Jürgen Umbrich
Vienna University of Economics and Business, Vienna, Austria
juergen.umbrich@wu.ac.at

Axel Polleres
Vienna University of Economics and Business, Vienna, Austria
axel.polleres@wu.ac.at

Magnus Knuth
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
magnus.knuth@hpi.de

ABSTRACT
There is an emerging demand on efficiently archiving and (temporal) querying different versions of evolving semantic Web data. As novel archiving systems are starting to address this challenge, foundations/standards for benchmarking RDF archives are needed to evaluate its storage space efficiency and the performance of different retrieval operations. To this end, we provide theoretical foundations on the design of data and queries to evaluate emerging RDF archiving systems. Then, we instantiate these foundations along a concrete set of queries on the basis of a real-world evolving dataset. Finally, we perform an empirical evaluation of various current archiving techniques and querying strategies on this data. Our work comprises – to the best of our knowledge – the first benchmark for querying evolving RDF data archives.

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 [18] which enables structured queries over this data). As previously reported [30, 19], this 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.

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 traditional Web archives, such as the Internet Archive,[1] archives for the Web of Data should additionally offer capabilities for time-traversing structured queries. Recently, initial works on RDF archiving policies/strategies [13, 34] are starting to offer such time-based capabilities, such as knowing whether a dataset or a particular entity has changed, which is neither natively supported by SPARQL nor by any of the existing temporal extensions of SPARQL [29, 14, 25, 36].

This paper discusses the emerging problem of evaluating the efficiency of the required retrieval demands in RDF archives. To the best of our knowledge, no work has been proposed to systematically benchmark RDF archives. The recent HOBBIT[2] H2020 EU project on benchmarking Big Linked Data is starting to face similar challenges. 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, 27], federated [22] or streaming data [8] in centralized or distributed repositories: they do not cover the particularities of RDF archiving, where querying entity changes across time is a crucial aspect.

In order to fill this gap, our main **contributions** are: (i) Based on an analysis of current RDF archiving proposals (Section 2), we provide theoretical foundations on the design of benchmark data and queries for RDF archiving systems (Section 3); (ii) we present a prototypical BEnchmark of RDF ARchives (referred to as *BEAR*), which makes use of real-world data from the Dynamic Linked Data Observatory [19] (Section 4). We describe queries with varying complexity, covering a broad range of archiving use cases; then, (iii) we implement RDF archiving systems on different RDF stores and archiving strategies, and evaluate them using BEAR to set an (extensible) baseline and illustrate our foundations (Section 5). Finally, we conclude and point out future work (Section 6).

2. PRELIMINARIES
We briefly summarise the necessary findings of our previous survey on current archiving techniques for dynamic Linked Open Data

[1] http://archive.org/.
[2] http://project-hobbit.eu/.

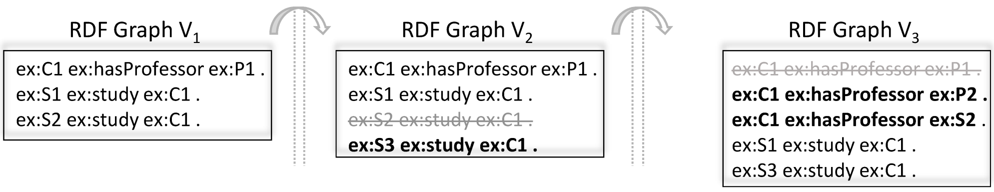


Figure 1: Example of RDF graph versions.

Type / Focus	Materialisation	Structured Queries: Single time	Structured Queries: 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 *-largest variation of students in the history of the archive*

Table 1: Classification and examples of retrieval needs.

[13]. The use case is depicted in Figure 1, showing an evolving RDF graph with three versions V_1, V_2 and V_3 : the initial version V_1 models two students *ex:S1* and *ex:S2* of a course *ex:C1*, whose professor is *ex:P1*. In V_2, the *ex:S2* student disappeared in favour of a new student, *ex:S3*. Finally, the former professor *ex:P1* leaves the course to a new professor *ex:P2*, and the former student *ex:S2* reappears also as 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 [13, 28] 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.).

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.

2.2 Archiving Policies and Retrieval Process

Main efforts addressing the challenge of RDF archiving fall in one of the following three storage strategies [13]: *independent copies (IC)*, *change-based (CB)* and *timestamp-based (TB)* approaches.

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

Independent Copies (IC) [20, 24] 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 case of structured queries dealing with deltas.

Change-based approach (CB) [33, 10, 35] partially addresses the previous scalability issue by computing and storing the differences (deltas) between versions. For the sake of simplicity, in this paper we focus on low-level deltas (added or deleted triples).

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 [10, 28], at the cost of augmenting space overheads.

Timestamp-based approach (TB) [6, 34, 23, 31] can be seen as a particular case of time modelling in RDF [17, 36], 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 follow two main strategies. On the one hand, compression techniques can be used to minimize the space overheads, e.g. using self-indexes [6] or delta compression in B+Trees [34]. On the other hand, triples can be time-annotated only when they are added or deleted (if present) [23, 31, 15, 34]. In these practical approaches, versions/deltas are often managed 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 [23].

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:

- The benchmark should be **archiving-policy agnostic** both in the dataset design/generation and the selection 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 recommendations on benchmarking RDF data management systems [1].

Although many benchmarks are defined for RDF stores [4, 1] (see the Linked Data Benchmark Council project [5] for a general overview) and related areas such as relational databases (e.g. the well-known TPC[4]) and graph databases [9], to the best of our knowledge, none of them are designed to address these particular considerations in RDF archiving. Nonetheless, most of the well-established benchmarks share important and general principles. We briefly recall here the four most important criteria when designing a domain-specific benchmark [16], which are also considered in our approach: 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 (to evaluate a set of easy-to-understand and extensible retrieval features).

We next formalize the most important features to characterize data and queries to evaluate RDF archives. These 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. [17]. 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*.

[4]http://www.tpc.org/

Data dynamicity. This feature measures the number of changes between versions, considering these differences at the level of triples. Thus, it is mainly described by the *change ratio* and the *data growth* between versions:

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*

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

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, IC policy duplicates the static information between two consecutive versions V_i and V_j, whereas the size of V_j increases with the added information ($\delta^+_{i,j}$) and decreases with the number of deletions ($\delta^-_{i,j}$), given that the latter are not represented. In contrast, CB and TB approaches store all changes, hence they are affected by the general dynamicity ($\delta_{i,j}$).

Data static core. It measures the triples that are available in all versions:

Definition 5 (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 (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 [17] and other representations based on annotation (e.g. AnQL [36]), where different annotations for the same triple are merged in an annotation set (often resulting 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 delta* 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 the RDF archive $\mathcal{A}$.*

Definition 8 (RDF VOCABULARY PER DELTA). *For an RDF archive $\mathcal{A}$, the vocabulary per delta is the set of subjects ($S_{\Delta^+_{i,j}}$ and $S_{\Delta^-_{i,j}}$), predicates ($P_{\Delta^+_{i,j}}$ and $P_{\Delta^-_{i,j}}$) and objects ($O_{\Delta^+_{i,j}}$ and $O_{\Delta^-_{i,j}}$) for all consecutive V_i and V_j in $\mathcal{A}$.*

Definition 9 (RDF 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*

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

Likewise, the vocabulary set dynamicity for insertions and deletions is defined by $vdyn^+(K, V_i, V_j) = \frac{|K_{V_j} \setminus K_{V_i}|}{|K_{V_i} \cup K_{V_j}|}$ and $vdyn^-(K, V_i, V_j) = \frac{|K_{V_i} \setminus K_{V_j}|}{|K_{V_i} \cup K_{V_j}|}$ respectively.

The evolution (cardinality and dynamicity) of the vocabulary is specially relevant in RDF archiving, since traditional RDF management systems use dictionaries (mappings between terms and integer IDs) to efficiently manage RDF graphs. Finally, whereas additional graph-based features (e.g. in-out-degree, clustering coefficient, presence of cliques, etc.) are interesting and complementary to our work, our proposed properties are feasible (efficient to compute and analyse) and grounded in state-of-the-art of archiving policies.

3.2 Design of Benchmark Queries

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 [14] and SPARQL-ST [25]) are based on SPARQL modifications.

In this scenario, previous experiences on benchmarking SPARQL resolution in RDF stores show that benchmark queries should report on the query type, result size, graph pattern shape, and query atom selectivity [26]. Conversely, for RDF archiving, one should put the focus on data dynamicity, without forgetting the strong impact played by query selectivity in most RDF triple stores and query planning strategies [1].

Let us briefly recall and adapt definitions of query cardinality and selectivity [2, 1] to RDF archives. Given a SPARQL query Q, where we restrict to SPARQL Basic Graph Patterns (*BGPs*[5]) hereafter, 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 10 (ARCHIVE-DRIVEN RESULT CARDINALITY). *The archive-driven result cardinality of Q over the RDF archive $\mathcal{A}$, is defined by*

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

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 11 (VERSION-DRIVEN RESULT CARDINALITY). *The version-driven result cardinality of Q over a version V_i, is defined by*

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

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 12 (VERSION-DRIVEN RESULT DYNAMICITY). *The version-driven result dynamicity of the query Q over two versions V_i and V_j, with $i < j$, denoted by $dyn(Q, V_i, V_j)$ is defined by*

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

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

[5] Sets of triple patterns, potentially including a FILTER condition, in which all triple patterns must match.

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 five foundational query atoms to cover the broad spectrum of emerging retrieval demands in RDF archiving. Rather than providing a complete catalog, our main aim is to reflect basic retrieval features on RDF archives, which can be combined to serve more complex queries. We elaborate these atoms on the basis of related literature, with especial attention to the needs of the well-established *Memento Framework* [7], which can provide access to prior states of RDF resources using datetime negotiation in HTTP.

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

Within the Memento Framework, this operation is needed to provide mementos (URI-M) that encapsulate a prior state of the original resource (URI-R).

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.

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. Formally, $Change(Q)$ reports the labels i, j (referring to the versions V_i and V_j) $\Leftrightarrow Diff(Q, V_i, V_j) \neq \emptyset, i < j, !\exists k \in \mathcal{N} / i < k < j$.

Within the Memento Framework, change materialisation is needed to provide timemap information to compile the list of all mementos (URI-T) for the original resource, i.e. the basis of datetime negotiation handled by the timegate (URI-G).

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

Table 2: Dataset configuration

These query features can be instantiated in domain-specific query languages (e.g. DIACHRON QL [21]) and existing extensions of SPARQL: the technical report extending our paper [11] includes an instantiation of these five queries in AnQL [36], as well as a discussion of how these AnQL queries could be evaluated over off-the-shelf RDF stores using "pure" SPARQL. However, since such an approach would typically render rather inefficient SPARQL queries, in the following sections, we focus on tailored implementations using optimized storage techniques to serve these features.

4. BEAR: A TEST SUITE FOR RDF ARCHIVING

This section presents BEAR, a prototypical (and extensible) test suite to demonstrate the new capabilities in benchmarking the efficiency of RDF archives using our foundations, and to highlight current challenges and potential improvements in RDF archiving. We first detail the dataset description and the query set covering basic retrieval needs. Next section evaluates BEAR on different archiving systems. The complete test suite (data corpus, queries, archiving system source codes, evaluation and additional results) are available at the BEAR repository[6].

4.1 Dataset Description

We build our RDF archive on the data hosted by the Dynamic Linked Data Observatory[7], monitoring more than 650 different domains across time and serving weekly crawls of these domains. BEAR 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 triples, disregarding duplicates. The extension to multiple graph archiving can be seen as a future work.

We report the data configuration features (cf. Section 3) that are relevant for our purposes. Table 2 lists basic statistics of our dataset[8]. 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%).

Conversely, the number of version-oblivious triples ($\mathcal{O}_\mathcal{A}$), $376m$, 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 $\overline{\delta}$ dynamicity values, stating that a mean of 31% of the data change between two versions. The same reasoning applies for the remarkably small static core ($\mathcal{C}_\mathcal{A}$), $3.5m$.

4.2 Test Queries

BEAR provides triple pattern queries Q to test each of the five atomic operations defined in our foundations (Section 3). Note that, although BEAR queries do not cover the full spectrum of SPARQL

[6]https://github.com/webdata/BEAR.
[7]http://swse.deri.org/dyldo/.
[8]Details on the vocabulary evolution can be found in our BEAR repository.

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

queries, triple patterns (i) constitute the basis for more complex queries, (ii) are the main operation served by lightweight clients such as the Linked Data Fragments [32] proposal, and (iii) they are the required operation to retrieve prior states of a resource in the Memento Framework. For simplicity, we focus on atomic lookup queries Q in the form (S??), (?P?), and (??O), which can then be extended to the rest of triple patterns (SP?), (S?O), (?PO), and (SPO)[9].

As for the generation of queries, we randomly select such triple patterns from the 58 versions of the Dynamic Linked Data Observatory. 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 $\Delta_{i,j}$.

Then, we follow the foundations 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 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. 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. EVALUATION OF RDF ARCHIVING SYSTEMS

We illustrate the use of our foundations to evaluate RDF archiving systems. To do so, we built two RDF archiving systems using the Jena's TDB store[10] (referred to as Jena hereinafter) and HDT [12], considering different state-of-the-art archiving policies (IC, CB and TB). Then, we use our prototypical BEAR to evaluate the influence of the concrete store and policy.

[9]The triple pattern (???) retrieves all the information, so no sampling technique is required.

[10]https://jena.apache.org/documentation/tdb/.

RAW DATA (gzip)	DIFF DATA (gzip)	JENA TDB IC	JENA TDB CB	JENA TDB TB	HDT IC	HDT CB
23 GB	14 GB	225 GB	196 GB	353 GB	46 GB	26 GB

Table 4: Space of the different archiving systems and policies.

Note that we considered these particular open RDF stores given that they are (i) easy to extend in order to implement the suggested archiving strategies, (ii) representative in the community and (iii) useful for potential archiving adopters. Jena is widely used in the community and can be considered as the de-facto standard implementation of most W3C efforts in RDF querying (SPARQL) and reasoning. In turn, HDT is a compressed store that considerably reduces space requirements of state-of-the-art stores (e.g. Virtuoso), hence it perfectly fits space efficiency requirements for archives. Furthermore, HDT is the underlying store of potential archiving adopters such as the crawling system LOD Laundromat[11], which generates new versions in each crawling process[12].

We implemented the IC, CB and TB policies in Jena (referred to as Jena-IC, Jena-CB and Jena-TB) as follows. 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 [31, 15] and indexed all deltas using named graph in one single TDB instance. We follow the same strategy to develop the IC and CB strategies in HDT [12] (referred to as HDT-IC, HDT-CB), which provides a compressed representation and indexing of RDF. The TB policy cannot be implemented as current HDT implementations[13] do not support quads, hence triples cannot be annotated with the version.

5.1 RDF Storage Space Results

Table 4 shows the required on-disk space for the raw data of the corpus, the GNU diff of such data, and the space required by the Jena and HDT[14] archiving systems under the different policies. Raw gzipped data take roughly 23GB disk space, while storing the diffs information results just requires 14GB. A comparison of these figures against the size of the different systems and policies allows the description of their inherent overheads: the IC policy indexing in Jena requires roughly ten times more space than the raw data, mainly due to the data decompression and the built-in Jena indexes. In contrast, the compact HDT indexes in the IC policy just double the size of the gzipped raw data, serving the required retrieval operations in such compressed space. In turn, Jena-CB reduces the space needs w.r.t Jena-IC (13% less). In HDT, the CB policy produces stronger reductions (43% less than HDT-IC), being roughly the double of the gzipped diff data. Finally, TB policy in Jena reports the highest size as it requires 57% and 80% more space than Jena-IC and Jena-CB respectively. Note that, although CB and TB policies manage the same delta sets, TB uses a unique Jena instance and stores named graph for the triples, so additional "context" indexes are required.

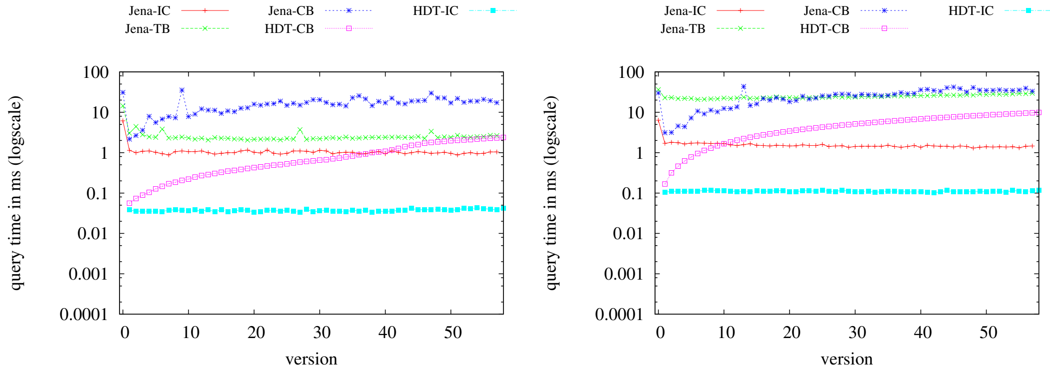
These initial results confirm current RDF archiving scalability

[11]http://lodlaundromat.org/

[12]LOD Laundromat only serves the last crawled version of a dataset.

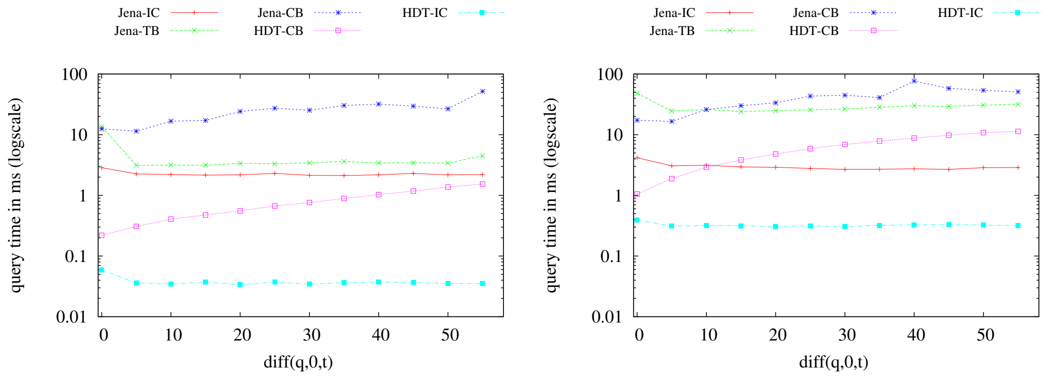
[13]We use the HDT C++ libraries at http://www.rdfhdt.org/.

[14]We include the space overheads of the provided HDT indexes to solve all lookups.



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

Figure 2: Query times for Mat queries.



(a) Q_L^O (b) Q_H^O

Figure 3: Query times for $Diff$ queries with increasing intervals.

problems at large scale, where specific RDF compression techniques such as HDT emerges as an ideal solution [13]. Note that Jena-IC requires almost 5 times the size of HDT-IC, whereas Jena-CB takes more than 7 times the space required by HDT-CB.

5.2 Retrieval Performance

From our foundations, 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 Q in each case. In general, our evaluation confirmed our assumptions about the characteristics of the policies (cf. Section 2), but also pointed out differences between the stores (Jena and HDT). Next, we present and discuss selected plots for each query operation.

Version materialisation. We measure and compute, for each version, the average query time[15] over all queries in the BEAR query set. The results for the version materialisation queries show very similar patterns for the subject, predicate and object query sets. Figure 2 reports results for the subject queries (with low and high cardinality), which are required in the Memento framework (predicate and object results are available in the BEAR repository).

First, we can observe in all plots that the HDT archiving system generally outperforms Jena. In both systems, the IC policy provides the best and most constant retrieval time. In contrast, 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 information processed. The TB policy in Jena performs worse than the IC, as TB has to query more indexed data than the IC policy. In turn, CB and TB have similar performances if queries have higher cardinalities, as shown in Figure 2 (b).

Delta materialisation queries. We performed diffs between the initial version and increasing intervals of 5 versions, i.e., $diff(Q, V_0, V_i)$ for i in $\{5, 10, 15, \cdots, 55, 57\}$. Figure 3 shows again the plots for selected query sets, in this case the diff of objects lookups. 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. The query time increases for the CB policy if the intervals of the deltas are increasing, given that more deltas have to be inspected.

Interestingly, HDT outperforms Jena under the same policy (IC or CB), i.e. HDT implements the policy more efficiently. However, an IC policy in Jena can be faster than a CB policy in HDT. In turn the TB policy in Jena behaves similar than the Mat case given that it always inspects the full store.

Version queries. Table 5 reports the average query time over each $ver(Q)$ query per BEAR query set. As can be seen, HDT archiving system clearly outperforms Jena in all scenarios, taking advantages of its efficient indexing. Nonetheless, the policies plays an important role: HDT-IC outperforms HDT-CB (as expected once versions have to be materialized in CB) except for predicates lookups

[15] All reported (elapsed) times in the evaluation are the average of three independent executions deployed in an Intel Xeon E5-2650v2 @ 2.6 GHz (32 cores), RAM 256 GB, Debian 7.9.

Query set	Jena TDB IC	CB	TB	HDT IC	CB
Q_L^S	55.70	122.44	27.32	**2.03**	2.73
Q_H^S	71.62	144.12	61.46	**6.57**	10.19
Q_L^P	304.67	412.17	237.83	128.20	**31.28**
Q_H^P	733.80	523.90	362.20	354.60	**104.57**
Q_L^O	40.54	91.92	22.78	**1.65**	2.89
Q_H^O	78.18	136.30	73.54	**8.37**	13.23

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

queries (Q_L^P and Q_H^P). The explanation of such behaviour is that predicate lookups in HDT have a logarithmic cost in the number of different predicates in the dataset, which is smaller in CB. In Jena the TB policy outperforms IC and CB policies in contrast to the previous Mat and $Diff$ experiments. This can be explained since IC and CB require to query each version, while TB only requires a query over the full store and then splits the results by version.

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. To this end, we have provided foundations to guide future evaluation of RDF archives. First, we formalized dynamic notions of archives, allowing to effectively describe the data corpus. Then, we described the main retrieval facilities involved in RDF archiving, and have provided guidelines on the selection of relevant and comparable queries. We instantiate these foundations in a prototypical benchmark, BEAR, serving a clean, well-described data corpus and a basic, but extensible, query testbed. Finally, we have implemented state-of-the-art archiving policies (using independent copies, differential representation and timestamps) in two stores (Jena TDB and HDT) and ran BEAR over them. Results clearly confirm challenges (in terms of scalability) and strengths of current archiving approaches, guiding future developments. We currently focus on exploiting the presented blueprints to build a customizable generator of evolving synthetic RDF data which can preserve user-defined characteristics while scaling to any dataset size and number of versions. We also work on providing more complex and meaningful time-based queries by inspecting query logs and real user needs.

Acknowledgements

Funded by Austrian Science Fund (FWF): M1720-G11, the PROPEL project (grant no. 849983) by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and the Austrian Research Promotion Agency (FFG), the ADEQUATe project (grant no. 849982) by Austrian Research Promotion Agency (FFG) and the German Government, Federal Ministry of Education and Research under the project number 03WKCJ4D.

7. 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] P. Boncz, I. Fundulaki, A. Gubichev, J. Larriba-Pey, and T. Neumann. The linked data benchmark council project. *Datenbank-Spektrum*, 13(2):121–129, 2013.

[6] A. Cerdeira-Pena, A. Farina, J. D. Fernández, and M. A. Martınez-Prieto. Self-indexing rdf archives. In *Proc. of DCC*, 2016.

[7] H. V. de Sompel, R. Sanderson, M. L. Nelson, L. Balakireva, H. Shankar, and S. Ainsworth. An HTTP-Based Versioning Mechanism for Linked Data. In *Proc. of LDOW*, 2010.

[8] 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.

[9] D. Dominguez-Sal, N. Martinez-Bazan, V. Muntes-Mulero, P. Baleta, and J. L. Larriba-Pey. A discussion on the design of graph database benchmarks. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, pages 25–40. Springer, 2010.

[10] 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.

[11] J. Fernández, J. Umbrich, A. Polleres, and M. Knuth. Query and Storage Strategies for RDF Archives. Technical report, 2015. Available at `http://git.io/v056F`.

[12] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange (HDT). *JWS*, 19:22–41, 2013.

[13] J. D. Fernández, A. Polleres, and J. Umbrich. Towards Efficient Archiving of Dynamic Linked Open Data. In *Proc. of DIACHRON*, 2015.

[14] F. Grandi. T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *Proc. of ADBIS*, pages 21–30. 2010.

[15] M. Graube, S. Hensel, and L. Urbas. R43ples: Revisions for triples. In *Proc. of LDQ*, volume CEUR-WS 1215, paper 3, 2014.

[16] J. Gray. *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann Publishers Inc., 1992.

[17] C. Gutierrez, C. Hurtado, and A. Vaisman. Introducing Time into RDF. *IEEE T. Knowl. Data En.*, 19(2):207–218, 2007.

[18] S. Harris and A. Seaborne. *SPARQL 1.1 Query Language*. W3C Recom. 2013.

[19] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing Linked Data Dynamics. In *Proc. of ESWC*, pages 213–227. 2013.

[20] 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.

[21] M. Meimaris, G. Papastefanatos, S. Viglas, Y. Stavrakas, and C. Pateritsas. A query language for multi-version data web archives. Technical report, Institute for the Management of Information Systems, Greece, 2015.

[22] 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.

[23] 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.

[24] N. F. Noy and M. A. Musen. Ontology Versioning in an Ontology Management Framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004.

[25] 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.

[26] M. Saleem, Q. Mehmood, and A.-C. N. Ngomo. FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework. In *Proc. of ISWC*, pages 52–69. 2015.

[27] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: a SPARQL performance benchmark. In *Proc. of ICDE*, pages 222–233, 2009.

[28] 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.

[29] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In *Proc. of ESWC*, pages 308–322. 2009.

[30] 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.

[31] 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.

[32] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying Datasets on the Web with High Availability. In *Proc. of ISWC*, pages 180–196, 2014.

[33] M. Volkel, W. Winkler, Y. Sure, S. Kruk, and M. Synak. Semversion: A versioning system for RDF and ontologies. In *Proc. of ESWC*, 2005.

[34] S. G. J. G. C. Zaniolo. Rdf-tx: A fast, user-friendly system for querying the history of rdf knowledge bases. In *Proc. of EDBT*, 2016.

[35] 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.

[36] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. *JWS*, 12:72–95, 2012.