Semantic Web 1 (0) 1–20
IOS Press

Towards Fully-fledged Archiving for RDF Datasets

Olivier Pelgrin [a,*], Luis Galárraga [b] and Katja Hose [a]
[a] *Department of Computer Science, Aalborg University, Denmark*
E-mails: olivier@cs.aau.dk, khose@cs.aau.dk
[b] *Inria, France*
E-mail: luis.galarraga@inria.fr

Abstract. The dynamicity of RDF data has motivated the development of solutions for archiving, i.e., the task of storing and querying previous versions of an RDF dataset. Querying the history of a dataset finds applications in data maintenance and analytics. Notwithstanding the value of RDF archiving, the state of the art in this field is under-developed: (i) most existing systems are neither scalable nor easy to use, (ii) there is no standard way to query RDF archives, and (iii) solutions do not exploit the evolution patterns of real RDF data. On these grounds, this paper surveys the existing works in RDF archiving in order to characterize the gap between the state of the art and a fully-fledged solution. It also provides *RDFev*, a framework to study the dynamicity of RDF data. We use RDFev to study the evolution of YAGO, DBpedia, and Wikidata, three dynamic and prominent datasets on the Semantic Web. These insights set the ground for the sketch of a fully-fledged archiving solution for RDF data.

Keywords: RDF Archiving, Knowledge Bases

1. Introduction

The amount of RDF data has steadily grown since the conception of the Semantic Web in 2001 [17], as more and more organizations opt for RDF [69] as the format to publish and manage semantic data [42, 44]. For example, by July 2009 the Linked Open Data (LOD) cloud counted a few more than 90 RDF datasets adding up to almost 6.7B triples [18]. By 2020, these numbers have catapulted to 1200+ datasets[1] and at least 28B triples[2], although estimates based on LOD-Stats [25] suggest more than 10K datasets and 150B+ triples if we consider the datasets with errors omitted by the LOD Cloud [62]. This boom does not only owe credit to the increasing number of data providers and availability of Open Government Data [1, 4, 10], but also to the constant evolution of the datasets in the LOD cloud. This phenomenon is especially true for community-driven initiatives such as DBpedia [13], YAGO [77], or Wikidata [26], and also applies to automatically ever-growing projects such as NELL [21].

Storing and querying the entire edition history of an RDF dataset, a task we call *RDF archiving*, has plenty of applications for data producers. For instance, RDF archives can serve as a backend for fine-grained version control in collaborative projects [9, 12, 32, 36, 52, 72]. They also allow data providers to study the evolution of the data [29] and track errors for debugging purposes. Likewise, they can be of use to RDF streaming applications that rely on a structured history of the data [20, 43]. But archives are also of great value for consumer applications such as data analytics, e.g.,

*Corresponding author. E-mail: olivier@cs.aau.dk.
[1] https://lod-cloud.net/
[2] http://lod-a-lot.lod.labs.vu.nl/

mining correction patterns [64, 65] or historical trend analysis [45].

For all the aforementioned reasons, a significant body of literature has started to tackle the problem of RDF archiving. The current state of the art ranges from systems to store and query RDF archives [3, 11, 12, 23, 34, 36, 59, 67, 72, 78, 81], to benchmarks to evaluate such engines [29, 51], as well as temporal extensions for SPARQL [16, 30, 35, 66]. Diverse in architecture and aim, all these works respond to particular use cases. Examples are solutions such as R&Wbase [72], R43ples [36], and Quit Store [12] that provide data maintainers with distributed version control management in the spirit of Git. Conversely, other works [34, 66] target data consumers who need to answer time-aware queries such as "obtain the list of house members who sponsored a bill from 2008". In this case the metadata associated to the actual triples is used to answer domain-specific requirements.

Despite this plethora of work, there is currently no available fully-fledged solution for the management of large and dynamic RDF datasets. This situation originates from multiple factors such as (i) the performance and functionality limitations of RDF engines to handle metadata, (ii) the absence of a standard for querying RDF archives, and (iii) a disregard of the actual evolution of real RDF data. This paper elaborates on factors (i) and (ii) through a survey of the state of the art that sheds light on what aspects have not yet been explored. Factor (iii) is addressed by means of a framework to study the evolution of RDF data applied to three large and ever-changing RDF datasets, namely DBpedia, YAGO, and Wikidata. The idea is to identify the most challenging settings and derive a set of design lessons for fully-fledged RDF archive management. We therefore summarize our contributions as follows:

1. *RDFev*, a metric-based framework to analyze the evolution of RDF datasets;
2. A study of the evolution of DBpedia, YAGO, and Wikidata using *RDFev*;
3. A detailed survey of existing work on RDF archive management systems and SPARQL temporal extensions;
4. An evaluation of Ostrich [78] on the history of DBpedia, YAGO, and Wikidata. This was the only system that could be tested on the experimental datasets;
5. The sketch of a fully-fledged RDF archiving system that can satisfy the needs not addressed in the literature, as well as a discussion about the challenges in the design and implementation of such a system.

$\langle s,p,o\rangle$, $\langle s,p,o,\rho\rangle$	triple and 4-tuple: subject, predicate, object, graph revision
G	an RDF graph
g	a graph label
G_i	the i-th version or revision of graph G
$A=\{G_0,G_1,\dots\}$	an RDF graph archive
$u=\{u^+,u^-\}$	an update or changeset with sets of added and deleted triples.
$u_{i,j}=\{u^+_{i,j},u^-_{i,j}\}$	the changeset between graph revisions i and j ($j>i$)
$rv(\rho)$	revision number of graph revision ρ
$ts(\rho)$	commit time of graph revision ρ
$l(\rho)$, $l(G)$	labels of graph revision ρ and graph G

Table 1

Notation related to RDF Graphs.

This paper is organized as follows. In Section 2 we introduce preliminary concepts. Then, Section 3 presents *RDFev*, addressing contribution (1). Contribution (2) is elaborated in Section 4. In the light of the evolution of real-world RDF data, we then survey the strengths and weaknesses of the different state-of-the-art solutions in Section 5 (contribution 3). Section 6 addresses contribution (4). The insights from the previous sections are then used to drive the sketch of an optimal RDF archiving system in Section 7, which addresses contribution (5). Section 8 concludes the paper.

2. Preliminaries

This section introduces the basic concepts in RDF archive storage and querying, and proposes some formalizations for the design of RDF archives.

2.1. RDF Graphs

We define an *RDF graph* G as a set of triples $t=\langle s,p,o\rangle$, where $s\in\mathcal{I}\cup\mathcal{B}$, $p\in\mathcal{I}$, and $o\in\mathcal{I}\cup\mathcal{L}\cup\mathcal{B}$ are the subject, predicate, and object of t, respectively. Here, $\mathcal{I},\mathcal{L}$, and $\mathcal{B}$ are sets of IRIs (entity identifiers), literal values (e.g., strings, integers, dates), and blank nodes (anonymous entities) [69]. The notion of a *graph* is based on the fact that G can be modeled as a directed labeled graph where the predicate p of a triple denotes a directed labeled edge from node s to node o. The RDF W3C standard [69] defines a *named graph* as an RDF graph that has been associated to a label $l(G)=$

$\langle s, p, o, \rho, \zeta \rangle$	a 5-tuple subject, predicate, object, graph revision, and dataset revision
$D = \{G^0, G^1, \dots\}$	an RDF dataset
$\mathcal{A} = \{D_0, D_1, \dots\}$	an RDF dataset archive
D_j	the j-th version or revision of dataset D
G_i^k	the i-th revision of the k-th graph in a dataset archive
$\hat{u} = \{\hat{u}^+, \hat{u}^-\}$	a graph changeset with sets of added and deleted graphs
$U = \{\hat{u}, u^0, u^1, \dots\}$	a dataset update or changeset consisting of a graph changeset $\hat{u}$ and changesets u^i associated to graphs G^i
U^+, U^-	the addition/deletion changes of U: $U^+ = \{\hat{u}^+, u^{0+}, u^{1+}, \dots\}$, $U^- = \{\hat{u}^-, u^{0-}, u^{1-}, \dots\}$
$U_{i,j}$	the dataset changeset between dataset revisions i and j ($j > i$)
$rv(\zeta)$	revision number of dataset revision ζ
$ts(\zeta)$	commit time of dataset revision ζ
$\Upsilon(\cdot)$	the set of terms (IRIs, literals, and blank nodes) present in a graph G, dataset D, changeset u, and dataset changeset U.

Table 2

Notation related to RDF Datasets.

$g \in \mathcal{I} \cup \mathcal{B}$. The function $l(\cdot)$ returns the associated label of an RDF graph, if any. Table 1 provides the relevant notation related to RDF graphs.

2.2. RDF Graph Archives

Intuitively, an *RDF graph archive* is a temporally-ordered collection of all the states an RDF graph has gone through since its creation. More formally, a graph archive $A = \{G_s, G_{s+i}, \dots, G_{s+n-1}\}$ is an ordered set of RDF graphs, where each G_i is a *revision* or *version* with *revision number* $i \in \mathcal{N}$, and G_s ($s \geqslant 0$) is the graph archive's *initial revision*. A non-initial revision G_i ($i > s$) is obtained by applying an *update* or *changeset* $u_i = \langle u_i^+, u_i^- \rangle$ to revision G_{i-1}. The sets u_i^+, u_i^- consist of triples that should be added and deleted respectively to and from revision G_{i-1} such that $u_i^+ \cap u_i^- = \emptyset$. In other words, $G_i = u_i(G_{i-1}) = (G_{i-1} \cup u_i^+) \setminus u_i^-$. Figure 1 provides a toy RDF graph archive A that models the evolution of the information about the country members of the United Nations (UN) and their diplomatic relationships (*:dr*). The archive stores triples such as ⟨ *:USA*, *a*, *:Country* ⟩ or ⟨ *:USA*, *:dr*, *:Cuba* ⟩, and consists of two revisions $\{G_0, G_1\}$. G_1 is obtained by applying update u_1 to the initial revision G_0. We extend the notion of changesets to arbitrary pairs of revisions i, j with $i < j$, and denote by $u_{i,j} = \langle u_{i,j}^+, u_{i,j}^- \rangle$ the changeset such that $G_j = u_{i,j}(G_i)$.

We remark that a graph archive can also be modeled as a collection of 4-tuples $\langle s, p, o, \rho \rangle$, where $\rho \in \mathcal{I}$ is the RDF identifier of revision $i = rv(\rho)$ and $rv \subset \mathcal{I} \times \mathcal{N}$ is a function that maps revision identifiers to natural numbers. We also define the function $ts \subset \mathcal{I} \times \mathcal{N}$ that associates a revision identifier ρ to its commit time, i.e., the timestamp of application of changeset u_i. Some solutions for RDF archiving [12, 34, 36, 59, 72, 78] implement this logical model in different ways and to different extents. For example, R43ples [36], R&WBase [72] and Quit Store [12] store changesets and/or their associated metadata in additional named graphs using PROV-O [24]. In contrast, x-RDF-3X [59] stores the temporal metadata in special indexes that optimize for concurrent updates at the expense of temporal consistency, i.e., revision numbers may not always be in concordance with the timestamps.

2.3. RDF Dataset Archives

In contrast to an RDF graph archive, an *RDF dataset* is a set $D = \{G^0, G^1, \dots, G^m\}$ of named graphs. Differently from revisions in a graph archive, we use the notation G^k for the k-th graph in a dataset, whereas G_i^k denotes the i-th revision of G^k. The notation related to RDF datasets is detailed in Table 2. Each graph $G^k \in D$ has a label $l(G^k) = g^k \in \mathcal{I} \cup \mathcal{B}$. The exception to this rule is G^0, known as the *default graph* [69], which is unlabeled.

Most of the existing solutions for RDF archiving can handle the history of a single graph. However, scenarios such as data warehousing [33, 38, 39, 47–50, 56, 57] may require to keep track of the common evolution of an RDF dataset, for example, by storing the addition and removal timestamps of the different RDF graphs in the dataset. Analogously to the definition of graph archives, we define a *dataset archive* $\mathcal{A} = \{D_0, D_1, \dots, D_{l-1}\}$ as a temporally ordered collection of RDF datasets. The j-th revision of $\mathcal{A}$ ($j > 1$) can be obtained by applying a *dataset update* $U_j = \{\hat{u}_j, u_j^0, u_j^1, \dots u_j^m\}$ to revision $D_{j-1} = \{G_{j-1}^0, G_{j-1}^1, \dots G_{j-1}^m\}$. U_j consists of an update per graph plus a special changeset $\hat{u}_j = \langle \hat{u}_j^+, \hat{u}_j^- \rangle$ that we call the *graph changeset* ($\hat{u}_j^+ \cap \hat{u}_j^- = \emptyset$). The sets $\hat{u}_j^+, \hat{u}_j^-$ store the labels of the graphs that should be added and deleted in revision j respectively. If a graph G^k is in $\hat{u}_j^-$ (i.e., it is scheduled for removal), then G^k

G_0	u_1	$G_1 = u_1(G_0)$
$\langle$:*USA*, *a*, :*Country*$\rangle$ $\langle$:*Cuba*, *a*, :*Country*$\rangle$ $\langle$:*USA*, :*dr*, :*Cuba*$\rangle$	$u_1^+ = \{\langle$:*France*, *a*, :*Country*$\rangle\}$ $u_1^- = \{\langle$:*USA*, :*dr*, :*Cuba*$\rangle\}$	$\langle$:*USA*, *a*, :*Country*$\rangle$ $\langle$:*Cuba*, *a*, :*Country*$\rangle$ $\langle$:*France*, : *a*, :*Country*$\rangle$

Fig. 1. Two revisions G_0, G_1 and a changeset u_1 of an RDF graph archive A

as well as its corresponding changeset $u_j^k \in U_j$ must be empty. It follows that we can obtain revision D_j by (i) applying the individual changesets $u_j^k(G_{j-1}^k)$ for each $0 \leqslant k \leqslant m$, (ii) removing the graphs in $\hat{u}_j^-$, and (iii) adding the graphs in $\hat{u}_j^+$.

Figure 2 illustrates an example of a dataset archive with two revisions D_0 and D_1. D_0 is a dataset with graphs $\{G_0^0, G_0^1\}$ both at local revision 0. The dataset update U_1 generates a new global dataset revision D_1. U_1 consists of three changesets: u_1^0 that modifies the default graph G^0, u_1^1 that leaves G^1 untouched, and the graph update $\hat{u}_2$ that adds graph G^2 to the dataset and initializes it at revision $s = 1$ (denoted by G_1^2).

As proposed by some RDF engines [31, 76], we define the master graph $G^M \in D$ (with label M) as the RDF graph that stores the metadata about all the graphs in an RDF dataset D. If we associate the creation of a graph G^k with label g^k to a triple of the form $\langle g^k, rdf{:}type, \eta{:}Graph\rangle$ in G^M for some namespace η, then we can model a dataset archive as a set of 5-tuples $\langle s, p, o, \rho, \zeta\rangle$. Here, $\rho \in \mathcal{I}$ is the RDF identifier of the local revision of the triple in an RDF graph with label $g = l(\rho)$ (Table 2). Conversely, $\zeta \in \mathcal{I}$ identifies a (global) dataset revision $j = rv(\zeta)$. Likewise, we overload the function $ts(\zeta)$ (defined originally in Table 1) so that it returns the timestamp associated to the dataset revision identifier ζ. Last, we notice that the addition of a non-empty graph to a dataset archive generates two revisions: one for creating the graph, and one for populating it. A similar logic applies to graph deletion.

2.4. SPARQL

SPARQL 1.1 is the W3C standard language to query RDF data [74]. For the sake of brevity, we do not provide a rigorous definition of the syntax and semantics of SPARQL queries; instead we briefly introduce the syntax of a subset of SELECT queries and refer the reader to the official specification [74]. SPARQL is a graph-based language whose building blocks are triple patterns. A *triple pattern* $\hat{t}$ is a triple $\langle \hat{s}, \hat{p}, \hat{o}\rangle \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$, where $\mathcal{V}$ is a set of variables such that $(\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}) \cap \mathcal{V} = \emptyset$ (variables are always prefixed with ? or \$). A *basic graph pattern* (abbreviated BGP) $\hat{G}$ is the conjunction of a set of triple patterns $\{\ \hat{t}_1\ .\ \hat{t}_2\ \ldots\ .\ \hat{t}_m\ \}$, e.g.,

{ *?s a :Person . ?s :nationality :France* }

When no named graph is specified, the SPARQL standard assumes that the BGP is matched against the *default graph* in the RDF dataset. Otherwise, for matches against specific graphs, SPARQL supports the syntax *GRAPH* $\bar{g}$ $\{\hat{G}\}$, where $\bar{g} \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{V}$. In this paper we call this, a *named BGP* denoted by $\hat{G}_{\bar{g}}$. A SPARQL select query Q on an RDF dataset has the basic form "SELECT V (FROM NAMED $\bar{g}_1$ FROM NAMED $\bar{g}_2 \ldots$) WHERE $\{\hat{G}'\ \hat{G}''\ \ldots\ \hat{G}_{\bar{g}_1}\ \hat{G}_{\bar{g}_2} \ldots\}$, with projection variables $V \subset \mathcal{V}$. SPARQL supports named BGPs $\hat{G}_{\bar{g}}$ with variables $\bar{g} \in \mathcal{V}$. In some implementations [31, 76] the bindings for those variables originate from the master graph G^M. The BGPs in the expression can contain FILTER conditions, be surrounded by OPTIONAL clauses, and be combined by means of UNION clauses.

2.5. Queries on Archives

Queries on graph/dataset archives may combine results coming from different revisions in the history of the data collection in order to answer an information need. The literature defines five types of queries on RDF archives [29, 78]. We illustrate them by means of our example graph archive from Figure 1.

- **Version Materialization.** VM queries are standard queries run against a single revision, such as *what was the list of countries according to the UN at revision j?*
- **Delta Materialization.** DM queries are standard queries defined on a changeset $u_j = \langle u_j^+, u_j^- \rangle$, e.g., *which countries were added to the list at revision j?*
- **Version.** V queries ask for the revisions where a particular query yields results. An example of a V

D_0	U_1	$D_1 = U_1(D_0)$
$G_0^0 = \{\langle :USA, a, :Country\rangle,$ $\langle :Cuba, a, :Country\rangle,$ $\langle :USA, :dr, :Cuba\rangle\}$	$\hat{u}_1 = \{\hat{u}_1^+ = \{G^2\},\ \hat{u}_1^- = \emptyset\}$ $u_1^0 = \{u_1^{0+} = \{\langle :France, a, :Country\rangle\},$ $u_1^{0-} = \{\langle :USA, :dr, :Cuba\rangle\}\}$	$G_1^0 = \{\langle :USA, a, :Country\rangle$ $\langle :Cuba, a, :Country\rangle$ $\langle :France, a, :Country\rangle\}$
$G_0^1 = \{\langle x{:}JFK, a, x{:}Airport\rangle\}$	$u_1^1 = \{\emptyset, \emptyset\}$	$G_1^1 = \{\langle x{:}JFK, a, x{:}Airport\rangle\}$ $G_1^2 = \emptyset$

Fig. 2. A dataset archive $\mathcal{A}$ with two revisions D_0, D_1. The first revision contains two graphs, the default graph G^0 and G^1. The dataset update $\hat{U}_1$ (i) modifies G^0, (ii) leaves G^1 untouched, and (iii) and creates a new graph G^2, all with local revision 1.

query is: *in which revisions j did USA and Cuba have diplomatic relationships?*

- **Cross-version.** CV queries result from the combination (e.g., via joins, unions, aggregations, differences, etc.) of the information from multiple revisions, e.g., *which of the current countries was not in the original list of UN members?*
- **Cross-delta.** CD queries result from the combination of the information from multiple sets of changes, e.g., *what are the revisions j with the largest number of UN member adhesions?*

Existing solutions differ in the types of queries they support. For example, Ostrich [78] provides native support for queries of types VM, DM, and V on single triple patterns, and can handle multiple triple patterns via integration with external query engines. Dydra [11], in contrast, has native support for all types of queries on BGPs of any size. Even though our examples use the revision number $rv(\rho)$ to identify a revision, some solutions may directly use the revision identifier ρ or the revision's commit time $ts(\rho)$. This depends on the system's data model.

3. Framework for the Evolution of RDF Data

This section proposes *RDFev*, a framework to understand the evolution of RDF data. The framework consists of a set of metrics and a software tool to calculate those metrics throughout the history of the data. The metrics quantify the changes between two revisions of an RDF graph or dataset and can be categorized into two families: metrics for low-level changes, and metrics for high-level changes. Existing benchmarks, such as BEAR [29], focus on low-level changes, that is, additions and deletions of triples. This, however, may be of limited use to data maintainers, who may need to know the semantics of those changes, for instance, to understand whether additions are creating new entities or editing existing ones. On these grounds, we propose to quantify changes at the level of entities and object values, which we call high-level.

RDFev takes each version of an RDF dataset as an RDF dump in N-triples format (our implementation does not support multi-graph datasets and quads for the time being). The files must be provided in chronological order. *RDFev* then computes the different metrics for each consecutive pair of revisions. The tool is implemented in C++ and Python and uses the RocksBD[3] key-value store as storage and indexing backend. All metrics are originally defined for RDF graphs in the state of the art [29], and have been ported to RDF datasets in this paper. *RDFev*'s source code is available at our project website[4].

3.1. Low-level Changes

Low-level changes are changes at the triple level. Indicators for low-level changes focus on additions and deletions of triples and vocabulary elements. The vocabulary $\Upsilon(D) \subset \mathcal{I} \cup \mathcal{L} \cup \mathcal{B}$ of an RDF dataset D is the set of all the terms occurring in triples of the dataset. Tracking changes in the number of triples rather than in the raw size of the RDF dumps is more informative for data analytics, as the latter option is sensitive to the serialization format. Moreover an increase in the vocabulary of a dataset can provide hints about the nature of the changes and the novelty of the data incorporated in a new revision. All metrics are defined by Fernández et al. [29] for pairs of revisions i, j with $j > i$.

[3] http://rocksdb.org
[4] https://relweb.cs.aau.dk/rdfev

Change ratio. The authors of BEAR [29] define the change ratio between two revisions i and j of an RDF graph G as

$$\delta_{i,j}(G) = \frac{|u^+_{i,j}| + |u^-_{i,j}|}{|G_i \cup G_j|}. \quad (1)$$

$\delta_{i,j}$ compares the size of the changes between two revisions w.r.t. the revisions' joint size. Large values for $\delta_{i,j}$ denote important changes in between the revisions. For a more fine-grained analysis, Fernández et al. [29] also proposes the insertion and deletion ratios:

$$\delta^+_{i,j} = \frac{|u^+_{i,j}|}{|G_i|} \quad (2) \qquad \delta^-_{i,j} = \frac{|u^-_{i,j}|}{|G_i|}. \quad (3)$$

We now adapt these metrics for RDF datasets. For this purpose, we define the size of a dataset D as $sz(D) = \sum_{G \in D} |G|$ and the size of a dataset changeset U as $sz(U) = sz(U^+) + sz(U^-)$ with $sz(U^+) = \sum_{u \in U} |u^+|$ and $sz^-(U) = \sum_{u \in U} |u^-|$. With these definitions, the previous formulas can be ported to RDF datasets as follows:

$$\delta_{i,j}(D) = \frac{sz(U)}{\sum_{G \in D_i \cap D_j} |G_i \cup G_j| + \sum_{G \in D_i \triangle D_j} |G|} \quad (4)$$

$$\delta^+_{i,j}(D) = \frac{sz(U^+)}{sz(D_i)} \quad (5) \qquad \delta^-_{i,j}(D) = \frac{sz(U^-)}{sz(D_i)} \quad (6)$$

Here, $D_i \triangle D_j$ denotes the symmetric difference between the sets of RDF graphs in revisions i and j.

Vocabulary dynamicity. The vocabulary dynamicity for two revisions i and j of an RDF graph is defined as [29]:

$$\text{vdyn}_{i,j}(G) = \frac{|\Upsilon(u_{i,j})|}{|\Upsilon(G_i) \cup \Upsilon(G_j)|} \quad (7)$$

$\Upsilon(u_{i,j})$ is the set of vocabulary terms – IRIs, literals, or blank nodes – in the changeset $u_{i,j}$ (Table **??**). The literature also defines the vocabulary dynamicity for insertions (vdyn+$_{i,j}$) and deletions (vdyn-$_{i,j}$):

$$\text{vdyn+}_{i,j}(G) = \frac{|\Upsilon(u^+_{i,j})|}{|\Upsilon(G_i) \cup \Upsilon(G_j)|} \quad (8)$$

$$\text{vdyn-}_{i,j}(G) = \frac{|\Upsilon(u^-_{i,j})|}{|\Upsilon(G_i) \cup \Upsilon(G_j)|}. \quad (9)$$

The formulas are analogous for RDF datasets if we replace G by D and $u_{i,j}$ by $U_{i,j}$.

Growth ratio. The grow ratio is the ratio between the number of triples in two revisions i, j. It is calculated as follows for graphs and datasets:

$$\Gamma_{i,j}(G) = \frac{|G_j|}{|G_i|} \quad (10) \qquad \Gamma_{i,j}(D) = \frac{sz(D_j)}{sz(D_i)}. \quad (11)$$

3.2. High-level Changes

A high-level change confers semantics to a changeset. For example, if an update consists of the addition of triples about an unseen subject, we can interpret the triples as the addition of an entity to the dataset. High-level changes provide deeper insights about the development of an RDF dataset than low-level changes. In addition, they can be domain-dependent. Some approaches [63, 71] have proposed vocabularies to describe changesets in RDF data as high-level changes. Since our approach is oblivious to the domain of the data, we propose a set of metrics on domain-agnostic high-level changes.

Entity changes. RDF datasets describe real-world entities s by means of triples $\langle s, p, o \rangle$. Hence, an entity is a subject for the sake of this analysis. We define the metric *entity changes* between revisions i, j in an RDF graph as:

$$ec_{i,j}(G) = |\sigma_{i,j}(G)| = |\sigma^+_{i,j}(G) \cup \sigma^-_{i,j}(G)| \quad (12)$$

In the formula, $\sigma^+_{i,j}$ is the set of added entities, i.e., the subjects present in $\Upsilon(G_j)$ but not in $\Upsilon(G_i)$ (analogously the set of deleted entities $\sigma^-_{i,j}$ is defined by swapping the roles of i and j). This metric can easily be adapted to an RDF dataset D if we define $ec(G)$ (with no subscripts) as the number of different subjects in a graph G. It follows that,

$$ec_{i,j}(D) = \sum_{G \in D_i \cap D_j} ec_{i,j}(G) + \sum_{G \in D_i \triangle D_j} ec(G). \quad (13)$$

We also propose the *triple-to-entity-change score*, that is, the average number of triples that constitute a single entity change. It can be calculated as follows for RDF graphs:

$$ect_{i,j}(G) = \frac{|\langle s, p, o \rangle \in u^+_{i,j} \cup u^-_{i,j} : s \in \sigma_{i,j}(G)|}{ec_{i,j}(G)} \quad (14)$$

Low-level changes	Change ratio Insertion and deletion ratios Vocabulary dynamicity Growth ratio
High-level changes	Entity changes Triple-to-entity-change score Object updates Orphan object additions and deletions

Table 3

RDFev's metrics

We port this metric to RDF datasets by first defining $\mathbf{U}^{+} = \bigcup_{u \in U^{+}} u$ and $\mathbf{U}^{-} = \bigcup_{u \in U^{-}} u$ and plugging them into the formula for $ect_{i,j}$:

$$ect_{i,j}(D) = \frac{|\langle s, p, o\rangle \in \mathbf{U}^{+}_{i,j} \cup \mathbf{U}^{-}_{i,j} : s \in \sigma_{i,j}(D)|}{ec_{i,j}(D)} \quad (15)$$

Object Updates and Orphan Object Additions/Deletions. An object update in a changeset $u_{i,j}$ is defined by the deletion of a triple $\langle s, p, o\rangle$ and the addition of a triple $\langle s, p, o'\rangle$ with $o \neq o'$. Once a triple in a changeset has been assigned to a high-level change, the triple is *consumed* and cannot be assigned to any other high-level change. We define orphan object additions and deletions respectively as those triples $\langle s^{+}, p^{+}, o^{+}\rangle \in u^{+}_{i,j}$ and $\langle s^{-}, p^{-}, o^{-}\rangle \in u^{-}_{i,j}$ that have not been consumed by any of the previous high-level changes. The dataset counterparts of these metrics for two revisions i, j can be calculated by summing the values for each of the graphs in $D_i \cap D_j$.

Table 3 summarizes all the metrics defined by RDFev.

4. Evolution Analysis of RDF Datasets

Having introduced *RDFev*, we use it to conduct an analysis of the revision history of three large and publicly available RDF knowledge bases, namely YAGO, DBpedia, and Wikidata. The analysis resorts to the metrics defined in Sections 3.1 and 3.2 for every pair of consecutive revisions.

4.1. Data

We chose the YAGO [77], DBpedia [13], and Wikidata [26] knowledge bases for our analysis, because of their large size, dynamicity, and central role in the Linked Open Data initiative. We build an RDF graph archive by considering each release of the knowledge base as a revision. None of the datasets is provided as a monolithic file, instead they are divided into *themes*. These are subsets of triples of the same nature, e.g., triples with literal objects extracted with certain extraction methods. We thus focus on the most popular themes. For DBpedia we use the *mapping-based objects* and *mapping-based literals* themes, which are available from version 3.5 (2015) onwards. Additionally, we include the *instance-types* theme as well as the *ontology*. For YAGO, we use the knowledge base's core, namely, the themes *facts*, *meta facts*, *literal facts*, *date facts*, and *labels* available from version 2 (v.1.0 was not published in RDF). As for Wikidata, we use the *simple-statements* of the RDF Exports [2] in the period from 2014-05 to 2016-08. These dumps provide a clean subset of the dataset useful for applications that rely mainly on Wikidata's encyclopedic knowledge. All datasets are available for download in the RDFev's website https://relweb.cs.aau.dk/rdfev. Table 4 maps revision numbers to releases for the sake of conciseness in the evolution analysis.

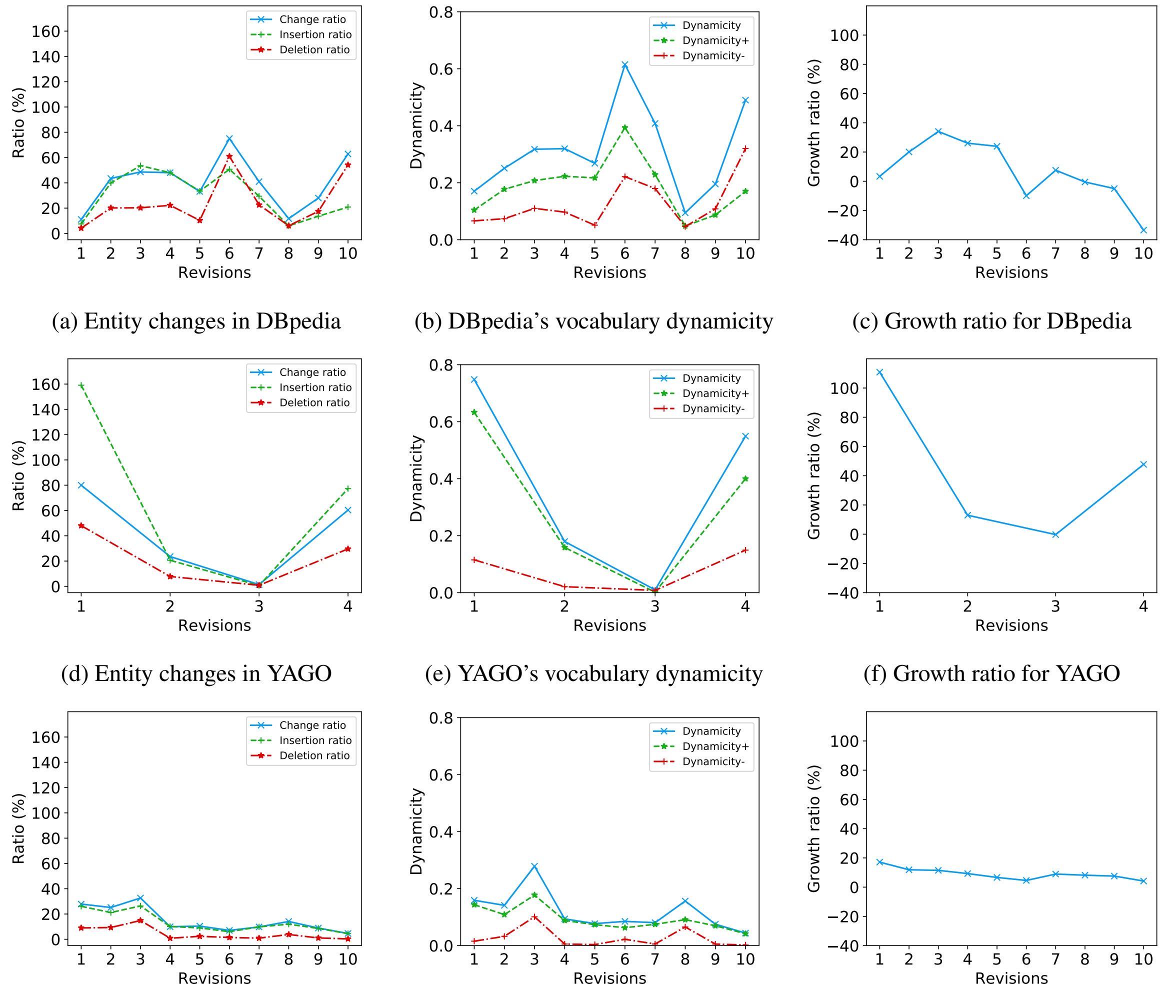
Revision	DBpedia	YAGO	Wikidata
0	3.5	2s	2014-05-26
1	3.5.1	3.0.0	2014-08-04
2	3.6	3.0.1	2014-11-10
3	3.7	3.0.2	2015-02-23
4	3.8	3.1	2015-06-01
5	3.9		2015-08-17
6	2015-04		2015-10-12
7	2015-10		2015-12-28
8	2016-04		2016-03-28
9	2016-10		2016-06-21
10	2019-08		2016-08-01

Table 4

Datasets revision mapping

4.2. Low-level Evolution Analysis

Change ratio. Figures 3a, 3d and 3g depict the evolution of the change, insertion, and deletion ratios for our experimental datasets. Up to the release 3.9 (rev. 5), DBpedia exhibits a steady growth with significantly more insertions than deletions. Minor releases such as 3.5.1 (rev. 1) are indeed minor in terms of low-level changes. Release 2015-04 (rev. 6) is an inflexion point



(a) Entity changes in DBpedia (b) DBpedia's vocabulary dynamicity (c) Growth ratio for DBpedia

(d) Entity changes in YAGO (e) YAGO's vocabulary dynamicity (f) Growth ratio for YAGO

(g) Entity changes in Wikidata (h) Wikidata's vocabulary dynamicity (i) Growth ratio for Wikidata

Fig. 3. Change ratio, vocabulary dynamicity, and growth ratio

not only in terms of naming scheme (see Table 4): the deletion rate exceeds the insertion rate and subsequent revisions exhibit a tight difference between the rates. This suggests a major design shift in the construction of DBpedia from revision 6.

As for YAGO, the evolution reflects a different release cycle. There is a clear distinction between major releases (3.0.0 and 3.1, i.e., rev. 1 and 4) and minor releases (3.0.1 and 3.0.2, i.e., rev. 2 and 3). The magnitude of the changes in major releases is significantly higher for YAGO than for any DBpedia release. Minor versions seem to be mostly focused on corrections, with a low number of changes.

Contrary to the other datasets, Wikidata shows a slowly decreasing change ratio that fluctuates between 5% (rev. 10) and 33% (rev. 3) within the studied period of 2 years.

Vocabulary dynamicity. As shown in Figures 3b, 3e, and 3h, the vocabulary dynamicity is, not surprisingly, correlated with the change ratio. Nevertheless, the vocabulary dynamicity between releases 3.9 and 2015-14 (rev. 5 and 6) in DBpedia did not decrease. This suggests that DBpedia 2015-04 contained more entities, but fewer – presumably noisy – triples about those entities. The major releases of YAGO (rev. 1 and 4) show a notably higher vocabulary dynamicity than the minor releases. As for Wikidata, slight spikes in dynamicity can be observed at revisions 4 and 9, however this metric remains relatively low in Wikidata compared to the others bases.

Growth ratio. Figures 3c, 3f, and 3i depict the growth ratio of our experimental datasets. In all cases, this metric is mainly positive with low values for minor revisions. As pointed out by the change ratio, the 2015-04 release in DBpedia is remarkable as the dataset shrank and was succeeded by more conservative growth ratios. This may suggest that recent DBpedia releases are more curated. We observe that YAGO's growth ratio is significantly larger for major versions. This is especially true for the 3.0.0 (rev. 1) release that doubled the size of the knowledge base.

4.3. High-level Evolution Analysis

4.3.1. Entity changes

Figures 4a, 4d, and 4g illustrate the evolution of the entity changes, additions, and deletions for DBpedia, YAGO and Wikidata. We also show the number of triples used to define these high-level changes (labeled as *affected triples*). We observe a stable behavior for these metrics in DBpedia except for the minor release 3.5.1 (rev. 1). Entity changes in Wikidata also display a monotonic behavior, even though the deletion rate tends to decrease from rev. 4. In YAGO, the number of entity changes peaks for the major revisions (rev. 1 and 4), and is one order of magnitude larger than for minor revisions. The minor release 3.0.2 (rev. 3) shows the lowest number of additions, whereas deletions remain stable w.r.t release 3.0.1 (rev. 2). This suggests that these two minor revisions focused on improving the information extraction process, which removed a large number of noisy entities.

Figure 4b shows the triple-to-entity-change score in DBpedia. Before the 2015-14 release, this metric fluctuates between 2 and 12 triples without any apparent pattern. Conversely subsequent releases show a decline, which suggests a change in the extraction strategies for the descriptions of entities. The same cannot be said about YAGO and Wikidata (Figures 4e and 4h), where values for this metric are significantly lower than for DBpedia, and remain almost constant. This suggests that minor releases in YAGO improved the strategy to extract entities, but did not change much the amount of extracted triples per entity.

4.3.2. Object Updates and Orphan Object Additions/Deletions

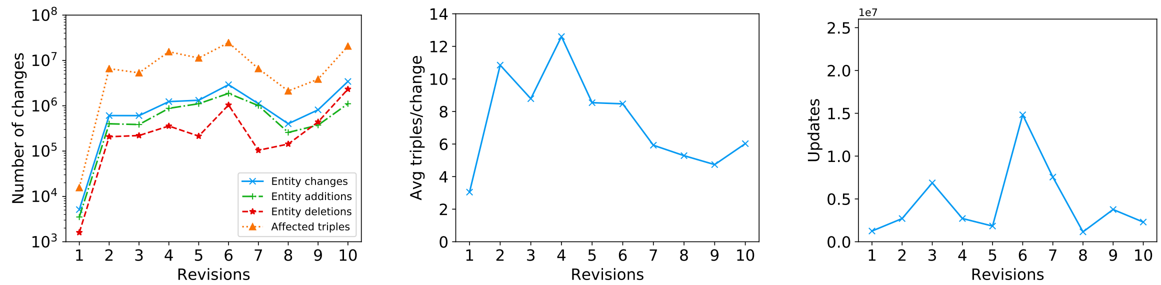
We present the evolution of the number of object updates for our experimental datasets in Figures 4c, 4f, and 4i. For DBpedia, the curve is consistent with the change ratio (Figure 3a). In addition to a drop in size, the 2015-04 release also shows the highest number of object updates, which corroborates the presence of a drastic redesign of the dataset.

The results for YAGO are depicted in Figure 4f, where we see larger numbers of object updates compared to major releases in DBpedia. This is consistent with the previous results that show that YAGO goes through bigger changes between releases. The same trends are observed for the number of orphan object additions and deletions in Figures 5a and 5b. Compared to the other two datasets, Wikidata's number of object updates, shown in Figure 4i, is much lower and constant throughout the stream of revisions.

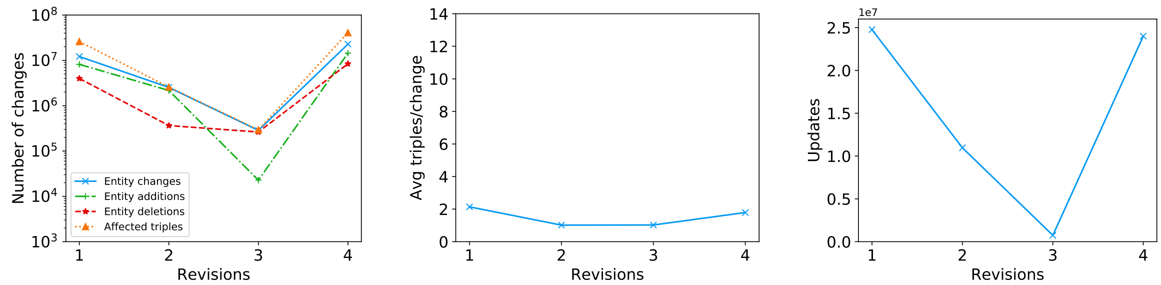
Finally, we remark that in YAGO and DBpedia, object updates are 4.8 and 1.8 times more frequent than orphan additions and deletions. This entails that the bulk of editions in these knowledge bases aims at updating existing object values. This behavior contrasts with Wikidata, where orphan object updates are 3.7 times more common than proper object updates. As depicted in Figure 5c, Wikidata exhibits many more orphan object updates than the other knowledge bases. Moreover, orphan object additions are 19 times more common than orphan object deletions.

4.4. Conclusion

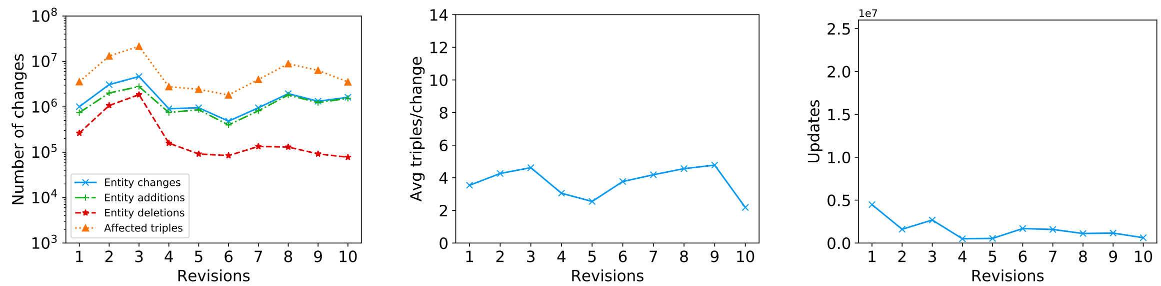
In this section we have conducted a study of the evolution of three large RDF knowledge bases using our proposed framework *RDFev*, which resorts to a domain-agnostic analysis from two perspectives: At the low-level it studies the dynamics of triples and vocabulary terms across different versions of an RDF dataset, whereas at the high-level it measures how those low-level changes translate into updates to the entities described in the experimental datasets. All in all, we have identified different patterns of evolution. On the one hand, Wikidata exhibits a stable release cycle in the studied period, as our metrics did not exhibit big fluctuations from release to release. On the other hand, YAGO and DBpedia have a release cycle that distinguishes between minor and major releases. Major releases are characterized by a large number of updates in the knowledge base and may not necessarily increase its size. Conversely, minor releases incur in at least one order of magnitude fewer changes than major releases and seem to focus on improving the quality of the knowledge base, for instance, by being more conservative in the number of triple and entity additions.



(a) Entity changes for DBpedia (b) Triple-to-entity changes in DBpedia (c) Dbpedia object updates

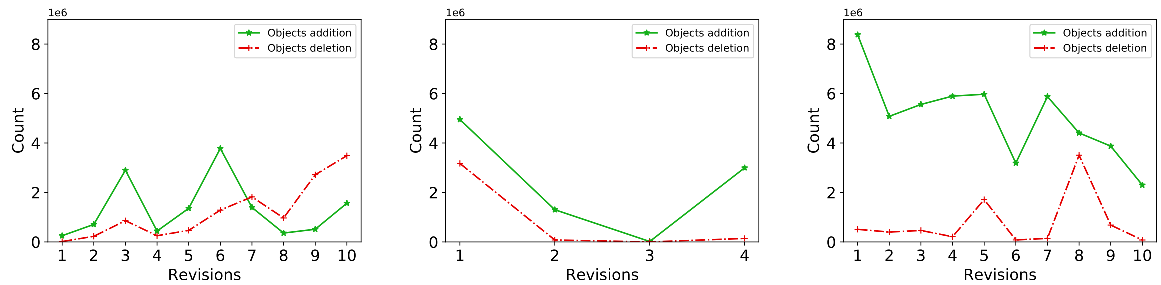


(d) Entity changes for YAGO (e) Triple-to-entity changes in YAGO (f) YAGO object updates



(g) Entity changes for Wikidata (h) Triple-to-entity changes in Wikidata (i) Wikidata object updates

Fig. 4. Entity changes and object updates



(a) Dbpedia (b) YAGO (c) Wikidata

Fig. 5. Orphan object additions and deletions

Unlike YAGO, DBpedia has shown decreases in size across releases. We argue that an effective solution for large-scale RDF archiving should be able to adapt to different patterns of evolution.

5. Survey of RDF Archiving Solutions

We structure this section in three parts. Section 5.1 surveys the existing engines for RDF archiving and discusses their strengths and weaknesses. Section 5.2 presents the languages and SPARQL extensions to express queries on RDF archives. Finally, Section 5.3 introduces various endeavors on analysis and benchmarking of RDF archives.

5.1. RDF Archiving Systems

There are plenty of systems to store and query the history of an RDF dataset. Except for a few approaches [11, 12, 36, 81], most available systems support archiving of a single RDF graph. Ostrich [78], for instance, manages quads of the form $\langle s, p, o, rv(\rho)\rangle$. Other solutions do not support revision numbers and use the ρ-component $\rho \in \mathcal{I}$ to model temporal metadata such as insertion, deletion, and validity timestamps for triples [34]. In this paper we make a distinction between insertion/deletion timestamps for triples and validity intervals. While the former are unlikely to change, the latter are subject to modifications because they constitute domain information, e.g., the validity of a marriage statement. This is why the general data model introduced in Section 2 only associates revision numbers and commit timestamps to the fourth component ρ, whereas other types of metadata are still attached to the graph label $g = l(\rho)$. We summarize the architectural spectrum of RDF archiving systems in Table 5 where we characterize the state-of-the-art approaches according to the following criteria:

- **Storage paradigm.** The storage paradigm is probably the most important feature as it shapes the system's architecture. We identify three main paradigms in the literature [29], namely independent copies (IC), change-based (CB), and timestamp-based (TB). Some systems [78] may fall within multiple categories, whereas Quit Store [12] implements a fragment-based (FB) paradigm.
- **Data model.** It can be quads or 5-tuples with different semantics for the fourth and fifth component.
- **Full BGPs.** This feature determines whether the system supports BGPs with a single triple pattern or full BGPs with an unbounded number of triple patterns and filter conditions.
- **Query types.** This criterion lists the types of queries on RDF archives (see Section 2.5) natively supported by the solution.
- **Branch & tags.** It defines whether the system supports branching and tagging as in classical version control systems.
- **Multi-graph.** This feature determines if the system supports archiving of the history of multi-graph RDF datasets.
- **Concurrent updates.** This criterion determines whether the system supports concurrent updates. This is defined regardless of whether conflict management is done manually or automatically.
- **Source available.** We also specify whether the system's source code is available for download and is usable, that is, whether it can be compiled and run in modern platforms.

In the following, we discuss further details of the state-of-the-art systems, grouped by their storage paradigms.

5.1.1. Independent Copies Systems

In an IC-like approach, each revision D_i of a dataset archive $\mathcal{A} = \{D_1, D_2, \ldots, D_n\}$ is fully stored as an independent RDF dataset. IC approaches shine at the execution of VM and CV queries as they do not incur any materialization cost for such types of queries. Conversely, IC systems are inefficient in terms of disk usage. For this reason they have mainly been proposed for small datasets or schema version control [61, 81]. SemVersion [81], for instance, is a system that offers similar functionalities as classical version control systems (e.g., CVS or SVN), with support for multiple RDF graphs and branching. Logically, SemVersion supports 5-tuples of the form $\langle s, p, o, l(\rho), rv(\rho)\rangle$, in other words, revision numbers are local to each RDF graph. This makes it difficult to track the addition or deletion of named graphs in the history of the dataset. Lastly, SemVersion provides an HTTP interface to submit updates either as RDF graphs or as changesets. Despite this flexibility, new revisions are always stored as independent copies. This makes its disk-space consumption prohibitive for large datasets like the ones studied in this paper.

	Storage paradigm	Data model	Full BGPs	Queries	Branch & tags	Multi-graph	Concurrent Updates	Source available
Dydra [11]	TB	5-tuples	+	all	-	+	-	-
Ostrich [78]	IC/CB/TB	quads	+[d]	VM, DM, V	-	-	-	+
QuitStore [12]	FB	5-tuples	+	all	+	+	+	+
RDF-TX [34]	TB	quads	+	all	-	-	-	-
R43ples [36]	CB	5-tuples[c]	+	all	+	+	+	+[a]
R&WBase [72]	CB	quads	+	all	+	-	+	+
RBDMS [46]	CB	quads	+	all	+	-	+	-
SemVersion [81]	IC	5-tuples[c]	-	VM, DM	+	-	+	-
Stardog [3]	CB	5-tuples	+	all	tags	+	-	-
v-RDFCSA [23]	TB	quads	-	VM, DM, V	-	-	-	-
x-RDF-3X [59]	TB	quads	+	VM, V	-	-	-	+[b]

[a] It needs modifications to have the console client running and working [b] Old source code [c] Graph local revisions
[d] Full BGP support is possible via integration with the Comunica query engine

Table 5

Existing RDF archiving systems

5.1.2. *Change-based Systems*

Solutions based on the CB paradigm store a subset $\hat{\mathcal{A}} \subset \mathcal{A}$ of the revisions of a dataset archive as independent copies or *snapshots*. On the contrary, all the intermediate revisions D_j ($p < j < q$) between two snapshots D_p and D_q, are stored as deltas or changesets U_j. The sequence of revisions stored as changesets between two snapshots is called a *delta chain*. CB systems are convenient for DM and CD queries. Besides, they are obviously considerably more storage-efficient than IC solutions. Their weakness lies in the high materialization cost for VM and CV queries, particularly for long delta chains.

R&WBase [72] is an archiving system that provides Git-like distributed version control with support for merging, branching, tagging, and concurrent updates with manual conflict resolution on top of a classical SPARQL endpoint. R&WBase supports all types of archive queries on full BGPs. The system uses the PROV-Ontology (PROV-O) [24] to model the metadata (e.g., timestamps, parent branches) about the updates of a single RDF graph. An update u_i generates two new named graphs G_g^{i+}, G_g^{i-} containing the added and deleted triples at revision i. Revisions can be materialized by processing the delta chain back to the initial snapshot, and they can be referenced via aliases called *virtual named graphs*. In the same spirit, tags and branches are implemented as aliases of a particular revision. R&WBase has inspired the design of R43ples [36]. Unlike the former, R43ples can version multiple graphs, although revision numbers are not defined at the dataset level, i.e., each graph manages its own history. Moreover, the system extends SPARQL with the clause *REVISION* j ($j \in \mathcal{N}$) used in conjunction with the GRAPH clause to match a BGP against a specific revision of a graph. Last, the approach presented by Dong-hyuk et al. [46] relies on an RDBMS to store snapshots and deltas of an RDF graph archive with support for branching and tagging. Its major drawback is the lack of support for SPARQL queries: while it supports all the types of queries introduced in Section 2.5, they must be formulated in SQL, which can be very tedious for complex queries.

Stardog [3] is a commercial RDF data store with support for dataset snapshots, tags, and full SPARQL support. Unlike R43ples, Stardog keeps track of the global history of a dataset, hence its logical model consists of 5-tuples of the form $\langle s, p, o, l(\rho), \zeta \rangle$ (i.e., metadata is stored at the dataset level). While the details of Stardog's internal architecture are not public, the documentation[5] suggests a CB paradigm with a relational database backend.

5.1.3. *Timestamp-based Systems*

TB solutions store triples with their temporal metadata, such as domain temporal validity intervals or insertion/deletion timestamps. Like in CB solutions, revisions must be materialized at a high cost for VM and CV queries. V queries are usually better supported, whereas the efficiency of materializing deltas depends on the system's indexing strategies.

[5] https://github.com/stardog-union/stardog-examples/tree/d7ac8b562ecd0346306a266d9cc28063fde7edf2/examples/cli/versioning

x-RDF-3X [59] is a system based on the RDF-3X [58] engine. Logically x-RDF-3X supports quads of the form $\langle s, p, o, \rho \rangle$ where $\rho \in \mathcal{I}$ is associated to all the revisions where the triple was present as well as to all addition and deletion timestamps. The system is a fully-fledged query engine optimized for highly concurrent updates with support for snapshot isolation in transactions. However, x-RDF-3X does not support versioning for multiple graphs, neither branching nor tagging.

Dydra [11] is a TB archiving system that supports archiving of multi-graph datasets. Logically, Dydra stores 5-tuples of the form $\langle s, p, o, l(\rho), \zeta \rangle$, that is, revision metadata lies at the dataset level. In its physical design, Dydra indexes quads $\langle s, p, o, l(\rho) \rangle$ and associates them to visibility maps and creation/deletion timestamps that determine the revisions and points in time where the quad was present. The system relies on six indexes – *gspo*, *gpos*, *gosp*, *spog*, *posg*, and *ospg* implemented as B+ trees – to support arbitrary SPARQL queries ($g = l(\rho)$ is the graph label). Moreover, Dydra extends the query language with the clause *REVISION* x, where x can be a variable or a constant. This clause instructs the query engine to match a BGP against the contents of the data sources bound to x, namely a single database revision ζ, or a dataset changeset $U_{j,k}$. A revision can be identified by its IRI ζ, its revision number $rv(\zeta)$ or by a timestamp τ'. The latter case matches the revision ζ with the largest timestamp $\tau = ts(\zeta)$ such that $\tau \leqslant \tau'$. Alas, Dydra's source is not available for download and use.

RDF-TX [34] supports single RDF graphs and uses a multiversion B-tree (MVBT) to index triples and their time metadata (insertion and deletion timestamps). An MVBT is actually a forest where each tree indexes the triples that were inserted within a time interval. RDF-TX implements an efficient compression scheme for MVBTs, and proposes SPARQL-T, a SPARQL extension that adds a fourth component $\hat{g}$ to BGPs. This component can match only time objects τ of type timestamp or time interval. The attributes of such objects can be queried via built-in functions, e.g., $year(\tau)$. While RDF-TX offers interval semantics at the query level, it stores only timestamps.

v-RDFCSA [23] is a lightweight and storage-efficient TB approach that relies on suffix-array encoding [19] for efficient storage with basic retrieval capabilities (much in the spirit of HDT [27]). Each triple is associated to a bitsequence of length equals the number of revisions in the archive. That is, v-RDFCSA logically stores quads of the form $\langle s, p, o, rv(\rho) \rangle$. Its query functionalities are limited since it supports only VM, DM, and V queries on single triple patterns.

5.1.4. Hybrid and Fragment-based Systems

Some approaches can combine the strengths of the different storage paradigms. One example is Ostrich [78], which borrows inspirations from IC, CB, and TB systems. Logically, Ostrich supports quads of the form $\langle s, p, o, rv(\rho) \rangle$. Physically, it stores snapshots of an RDF graph using HDT [27] as serialization format. Delta chains are stored as B+ trees timestamped with revision numbers in a TB-fashion. These delta chains are redundant, i.e., each revision in the chain is stored as a changeset containing the changes w.r.t. the latest snapshot – and not the previous revision as proposed by Dong-hyuk et al. [46]. Ostrich alleviates the cost of redundancy using compression. All these design features make Ostrich query and space efficient, however its functionalities are limited. Its current implementation does not support more than one (initial) snapshot and a single delta chain, i.e., all revisions except for revision 0 are stored as changesets of the form $u_{0,i}$. Multi-graph archiving as well as branching/tagging are not possible. Moreover, the system's querying capabilities are restricted to VM, DM, and V queries on single triple patterns. Support for full BGPs is possible via integration with the Comunica query engine[6].

Like R43ples [36], Quit Store [12] provides collaborative Git-like version control for multi-graph RDF datasets, and uses PROV-O for metadata management. Unlike R43ples, Quit Store provides a global view of the evolution of a dataset, i.e., each commit to a graph generates a new dataset revision. The latest revision is always materialized in an in-memory quad store. QuitStore is implemented in Python with RDFlib and provides full support for SPARQL 1.1. The dataset history (RDF graphs, commit tree, etc.) is physically stored in text files (i.e. N-quads files resp. N-triples files in the latest implementation) and is accessible via a SPARQL endpoint on a set of virtual graphs. However, the system only stores snapshots of the modified files in the spirit of fragment-based storage. Quit Store is tailored for collaborative construction of RDF datasets, but its high memory requirements make it unsuitable as an archiving backend. As discussed in Section 7, fully-fledged RDF archiving can provide a backend for this type of applications.

[6]https://github.com/rdfostrich/comunica-actor-init-sparql-ostrich

5.2. Languages to Query RDF Archives

Multiple research endeavors have proposed alternatives to succinctly formulate queries on RDF archives. The BEAR benchmark [29] uses AnQL to express the query types described in Section 2.5. AnQL [82] is a SPARQL extension based on quad patterns $\langle \hat{s}, \hat{p}, \hat{o}, \hat{g} \rangle$. AnQL is more general than SPARQL-T (proposed by RDF-TX [34]) because the $\hat{g}$-component can be bound to any term $u \in \mathcal{I} \cup \mathcal{L}$ (not only time objects). For instance, a DM query asking for the countries added at revision 1 to our example RDF dataset from Figure 1 could be written as follows:

```
SELECT * WHERE {
  { (?x a :Country): [1] } MINUS
  { (?x a :Country): [0] }
}
```

T-SPARQL [35] is a SPARQL extension inspired by the query language TSQL2 [75]. T-SPARQL allows for the annotation of groups of triple patterns with constraints on temporal validity and commit time, i.e., it supports both time-intervals and timestamps as time objects. T-SPARQL defines several comparison operators between time objects, namely *equality*, *precedes*, *overlaps*, *meets*, and *contains*. Similar extensions [16, 66] also offer support for geo-spatial data.

SPARQ-LTL [30] is a SPARQL extension that makes two assumptions, namely that (i) triples are annotated with revision numbers, and (ii) revisions are accessible as named graphs. When no revision is specified, BGPs are iteratively matched against *every* revision. A set of clauses on BGPs can instruct the SPARQL engine to match a BGP against other revisions at each iteration. For instance the clause *PAST* in the expression *PAST*{ q } MINUS { q } with $q = \langle ?x\ a\ :Country \rangle$ will bind variable ?x to all the countries that were ever deleted from the RDF dataset, even if they were later added.

5.3. Benchmarks and Tools for RDF Archives

BEAR [29] is the state-of-the-art benchmark for RDF archive solutions. The benchmark provides three real-world RDF graphs (called BEAR-A, BEAR-B, and BEAR-C) with their corresponding history, as well as a set of VM, DM, and V queries on those histories. In addition, BEAR allows system designers to compare their solutions with baseline systems based on different storage strategies (IC, CB, TB, and hybrids TB/CB, IC/CB) and platforms (Jena TDB and HDT). Despite its multiple functionalities and its dominant position in the domain, BEAR has some limitations: (i) It assumes single-graph RDF datasets; (ii) it does not support CV and CD queries, moreover VM, DM, and V queries are defined on single triple patterns; and (iii) it cannot simulate datasets of arbitrary size and query workloads.

EvoGen [51] tackles the latter limitation by extending the Lehigh University Benchmark (LUBM) [37] to a setting where both the schema and the data evolve. Users can not only control the size and frequency of that evolution, but can also define customized query workloads. EvoGen supports all the types of queries on archives presented in Section 2.5 on multiple triple patterns.

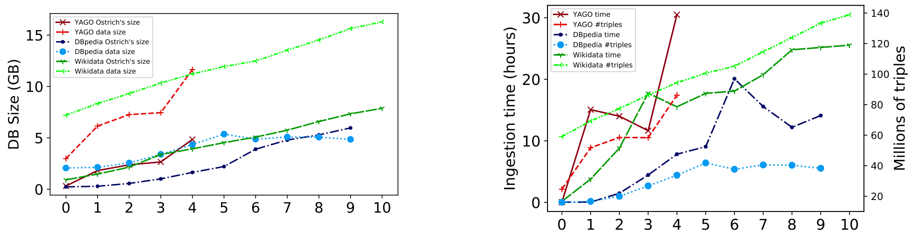
A recent approach [79] proposes to use FCA (Formal Concept Analysis) and several data fusion techniques to produce summaries of the evolution of entities across different revisions of an RDF archive. A summary can, for instance, describe groups of subjects with common properties that change over time. Such summaries are of great interest for data maintainers as they convey edition patterns in RDF data through time.

6. Evaluation of the Related Work

In this section, we conduct an evaluation of the state-of-the-art RDF archiving engines. We first provide a global analysis of the systems' functionalities in Section 6.1. Section 6.2 then provides a performance evaluation of Ostrich (the only testable solution) on our experimental RDF archives from Table 4. This evaluation is complementary to the Ostrich's evaluation on BEAR (available in [78]), as it shows the performance of the system in three real-world large RDF datasets.

6.1. Functionality Analysis

As depicted in Table 5, existing RDF archiving solutions differ greatly in design and functionality. The first works [22, 59, 81] offered mostly storage of old revisions and support for basic VM queries. Consequently, subsequent efforts focused on extending the query capabilities and allowing for concurrent updates as in standard version control systems [12, 36, 46, 72]. Such solutions are attractive for data maintainers in collaborative projects, however they still lack scalability, e.g., they cannot handle large datasets and changesets, besides conflict management is still delegated to users. More recent works [23, 78] have therefore focused on improving storage and querying perfor-



(a) Disk-space usage

(b) Ingestion time

Fig. 6. Ostrich's performance on multiple revisions of DBpedia and YAGO

mance, alas, at the expense of features. For example, Ostrich [78] is limited to a single snapshot and delta chain. In addition to the limitations in functionality, Table 5 shows that most of the existing systems are not available because their source code is not published. While this still leaves us with Ostrich [78], Quit Store [12], R&WBase [72], R43ples [36] and x-RDF-3X as testable solutions, only [78] was able to run on our experimental datasets. To carry out a fair comparison with the other systems, we tried Quit Store in the *persistence* mode, which ingests the data graphs into main memory at startup – allowing us to measure ingestion times. Unfortunately, the system crashes for all our experimental datasets[7]. We also tested Quit Store in its default lazy loading mode, which loads the data into main memory at query time. This option throws a Python *MemoryError* for our experimental queries. In regards to R43ples, we had to modify its source code to handle large files[8]. Despite this change, the system could not ingest a single revision of DBpedia after four days of execution. R&WBase, on the other hand, accepts updates only through a SPARQL endpoint, which cannot handle the millions of update statements required to ingest the changesets. Finally, x-RDF-3X's source code does not compile out of the box in modern platforms, and even after successful compilation, it is unable to ingest one DBpedia changeset.

6.2. Performance Analysis

We evaluate the performance of Ostrich on our experimental datasets in terms of storage space, ingestion time – the time to generate a new revision from an input changeset – and query response time. The changesets were computed with RDFev from the different versions of DBpedia, YAGO, and Wikidata (Table 4). All the experiments were run on a server with a 4-core CPU (Intel Xeon E5-2680 v3@2.50GHz) and 64 GB of RAM.

Storage space. Figure 6a shows the amount of storage space (in GB) used by Ostrich for the selected revisions of our experimental datasets. We provide the raw sizes of the RDF dumps of each revision for reference. Storing each version of YAGO separately requires 36 GB, while Ostrich uses only 4.84 GB. For DBpedia compression goes from 39 GB to 5.96 GB. As for Wikidata, it takes 131 GB to stores the raw files, but only 7.88 GB with Ostrich. This yields a compression rate of 87% for YAGO, 84% for DBpedia and 94% for Wikidata. This space efficiency is the result of using HDT [27] for snapshot storage, as well as compression for the delta chains.

Ingestion time. Figure 6b shows Ostrich's ingestion times. We also provide the number of triples of each revision as reference. The results suggest that this measure depends both on the changeset size, and the length of the delta chain. However, the latter factor becomes more prominent as the length of the delta chain increases. For example, we can observe that Ostrich requires ∼22 hours to ingest revision 9 of DBpedia (2.43M added and 2.46M deleted triples) while it takes only ∼14 hours to ingest revision 5 (12.85M added and 5.95M deleted triples). This confirms the trends observed in [78] where ingestion time increases linearly with the number of revisions. This is explained by the fact that Ostrich stores the i-th revision of an archive as a changeset of the form $u_{0,i}$. In consequence, Ostrich's changesets are constructed from the triples in all previous revisions, and can only grow in size. This fact makes it unsuitable for very long histories.

Query runtime. We run Ostrich on 100 randomly generated VM, V, and DM queries on our experimental

[7] The Python interpreter reports a *UnboundLocalError*.

[8] The code creates an array that exceeds the maximal array size in Java.

	DBpedia			YAGO			Wikidata		
Triple Patterns	**VM**	**V**	**DM**	**VM**	**V**	**DM**	**VM**	**V**	**DM**
? p ?	92.81(0)	118.64(0)	91.78(0)	2.9(3)	- (5)	1.82(3)	281.41(0)	302.26(0)	303.73(0)
? <top p> o	112.81(0)	283.59(0)	130.88(0)	35.08(2)	69.65(4)	137.16(4)	347.06(0)	499.77(0)	285.02(0)
? p o	96.74(0)	92.04(0)	91.93(0)	2.38(4)	2.35(4)	2.35(2)	282.12(0)	281.63(0)	281.45(0)
s p ?	94.99(0)	91.67(0)	92.81(0)	2.42(2)	2.36(3)	2.41(1)	284.74(0)	281.4(0)	281.2(0)

Table 6

Ostrich's Query Performance in seconds

datasets. Ostrich does not support queries on full BGPs natively, thus the queries consisted of single triple patterns of the most common forms, namely ⟨ *?*, *p*, *?* ⟩, ⟨ *s*, *p*, *?* ⟩, and ⟨ *?*, *p*, *o* ⟩ in equal numbers. We also considered queries ⟨ *?*, *<top p>*, *o* ⟩, where *<top p>* corresponds to the top 5 most common predicates in the dataset. Revision numbers for all queries were also randomly generated. Table 6 shows Ostrich's average runtime in seconds for the different types of queries. We set a timeout of 1 hour for each query, and show the number of timeouts in parentheses next to the runtime, which excludes queries that timed out. We observe that Ostrich is roughly one order of magnitude faster on YAGO than on DBpedia and Wikidata. To further understand the factors that impact Ostrich's runtime, we computed the Spearman correlation score between Ostrich's query runtime and a set of features relevant to query execution. These features include the length of the delta chain, the average size of the relevant changesets, the size of the initial revision, the average number of deleted and added triples in the changesets, and the number of query results. The results show that the most correlated features are the length of the delta chain, the standard deviation of the changeset size, and the average number of deleted triples. This suggests that Ostrich's runtime performance will degrade as the history of the archive grows and that massive deletions actually aggravate that phenomenon. Finally, we observe some timeouts in YAGO in contrast to DBpedia and Wikidata. We believe this is mainly caused by the sizes of the changesets, which are on average of 3.72GB for YAGO, versus 2.09GB for DBpedia and 1.86GB for Wikidata. YAGO's changesets at revisions 1 and 4 are very large as shown in Section 4.

7. Towards Fully-fledged RDF Archiving

We now build upon the findings from previous sections to derive a set of lessons towards the design of a scalable fully-fledged solution for archiving of large RDF datasets. We structure this section in two parts. Section 7.1 discusses the most important functionalities that such a solution may offer, whereas Section 7.2 discusses the algorithmic and design challenges of providing those functionalities.

7.1. Functionalities

Global and local history. Our survey in Section 5.1 shows that R43ples [36] and Quit Store [12] are the only available solutions that support both archiving of the local and joint (global) history of multiple RDF graphs. We argue that such a feature is vital for proper RDF archiving: It is not only of great value for distributed version control in collaborative projects, but can also be useful for the users and maintainers of data warehouses. Conversely, existing solutions are strictly focused on distributed version control and their Git-based architectures make them unsuitable to archive the releases of large datasets such as YAGO, DBpedia, or Wikidata as explained in Section 6. From an engineering and algorithmic perspective, this implies to redesign RDF solutions to work with 5-tuples instead of triples. We discuss the technical challenges of such requirement in Section 7.2.

Temporal domain-specific vs. revision metadata. Systems and language extensions for queries with time constraints [34, 35], treat both domain-specific metadata (e.g., triple validity intervals) and revision-related annotations (e.g., revision numbers) in the same way. We highlight, however, that revision metadata is immutable and should therefore be logically placed at a different level. In this line of thought we propose to associate revision metadata for graphs and datasets, e.g., commit time, revision numbers, or branching & tagging information, to the local and global revision identifiers ρ and ζ, whereas depending on the application, domain-specific time objects could be modeled either as statements about the revisions or as statements about the graph labels $g = l(\rho)$. The former alternative enforces the same temporal domain-specific metadata to all the triples added in a changeset, whereas the latter option makes sense if all the triples with the same graph label are supposed to share the same domain-specific information – which can still be edited by another changeset on the master graph. We depict both alternatives in Figure 7. We remark that such associations are only defined at the logical level.

Provenance. Revision metadata is part of the history of a triple within a dataset. Instead, its complete history is given by its workflow provenance. The W3C offers the PROV-O ontology [24] to model the history of a triple from its sources to its current state in an RDF dataset. Pretty much like temporal domain-specific metadata, provenance metadata can be logically linked to either the (local or global) revision identifiers or to the graph labels (Figure 7). This depends on whether we want to define provenance for changesets because the triples added to an RDF graph may have different provenance workflows. A hybrid approach could associate a default provenance history to a graph and use the revision identifiers to override or extend that default history for new triples. Moreover, the global revision identifier ζ provides an additional level of metadata that allow us to model the provenance of a dataset changeset.

Concurrent updates & modularity. We can group the existing state-of-the-art solutions in three categories regarding their support for concurrent updates, namely (i) solutions with limited or no support for concurrent updates [11, 23, 34, 67, 78], (ii) solutions inspired by version control systems such as Git [12, 36, 46, 72, 81], and (iii) approaches with full support for highly concurrent updates [59]. Git-like solutions are particularly interesting for collaborative efforts such as DBpedia, because it is feasible to delegate users the task of conflict management. Conversely, fully automatically constructed KBs such as NELL [21] or data-intensive (e.g., streaming) applications may need the features of solutions such as x-RDF-3X [59]. Consequently, we propose a modular design that separates the concurrency layer from the storage backend. Such a middleware could take care of enforcing a consistency model for concurrent commits either automatically or via user-based conflict management. The layer could also manage the additional metadata for features such as branching and tagging. In that light, collaborative version control systems for RDF [12, 36, 72] become an application of fully-fledged RDF archiving.

Formats for publication and querying. A fully functional archiving solution should support the most popular RDF serialization formats for data ingestion and dumping. For metadata enhanced RDF, this should include support for N-quads, singleton properties, and RDF-star. Among those, RDF-star [40] is the only one that can natively support multiple levels of metadata (still in a very verbose fashion). For example RDF-star could serialize the tuple $\langle :USA, :dr, :Cuba, \rho, \zeta \rangle$ with graph label (*:gl*) $l(\rho) = :UN$ and global timestamp (*:ts*) $ts(\zeta) =$*2020-07-09* as follows:

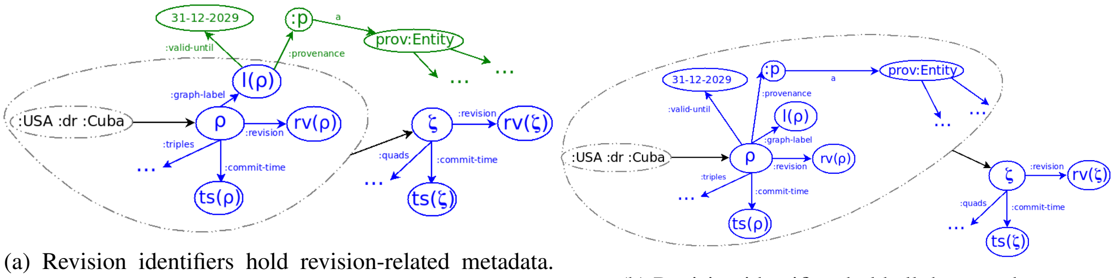
<<<*:USA :dr :Cuba*> *:gl :UN*> *:ts* "*2020-07-09*"^^*xsd:date*>

The authors of [40] propose this serialization as part of the Turtle-star format. Moreover, they propose SPARQL-star that allows for nested triple patterns. While SPARQL-star enables the definition of metadata constraints at different levels, a fully archive-compliant language could offer further syntactic sugar such as the clauses *REVISION* [11, 36] or *DELTA* to bind the variables of a BGP to the data in particular revisions or deltas. We propose to build such an archive-compliant language upon SPARQL-star.

Support for different types of archive queries. Most of the studied archiving systems can answer all the query types defined in the literature of RDF archives [29, 78]. That said, more complex queries such as CD and CV queries, or queries on full BGPs are sometimes supported via query middlewares and external libraries [12, 78]. We endorse this design philosophy because it eases modularity. Existing applications in mining archives [45, 64, 65] already benefit from support for V, VM, and DM queries on single triple patterns. By guaranteeing scalable runtime for such queries, we can indirectly improve the runtime of more complex queries. Further optimizations can be achieved by proper query planning.

7.2. Challenges

Trade-offs on storage, query runtime, and ingestion time. RDF archiving differs from standard RDF management in an even more imperative need for scalability, in particular storage efficiency. As shown by Taelman et al. [78], the existing storage paradigms shine at different types of queries. Hence, supporting arbitrary queries while being storage-efficient requires the best from the IC, CB, FB, and TB philosophies. A hybrid approach, however, will inevitably be more



(a) Revision identifiers hold revision-related metadata. Provenance as well as temporal domain-specific metadata are in green.

(b) Revision identifiers hold all the metadata.

Fig. 7. Two logical models to handle metadata in RDF archives. The namespace *prov:* corresponds to the PROV-O namespace.

complex and introduce further parameters and trade-offs. The authors of Ostrich [78], for instance, chose to benefit faster version materialization via redundant deltas at the expense of larger ingestion times. Users requiring shorter ingestion times could, on the other hand, opt for non-redundant changesets, or lazy non-asynchronous ingestion (at the expense of data availability). We argue that the most crucial algorithmic challenge for a CB archiving solution is to decide when to store a revision as a snapshot or as a delta, which is tantamount to trading disk space for faster VM queries. This could be formulated as an (multi-objective) minimization problem whose objective might be a function of response time for triple patterns in VM, CV and V queries with constraints on available disk space and average ingestion time. When high concurrency is imperative, the objective function could also take query throughput into account. In the same vibe, a TB solution could trigger the construction of further indexes (e.g., new combinations of components, incremental indexes in the concurrent setting) based on a careful consideration of disk consumption and runtime gain. Such scenarios would not only require the conception of a novel cost model for query runtime in the archiving setting, but also the development of approximation algorithms for the underlying optimization problems, which are likely NP-Hard. Finally, since fresh data is likely to be queried more often than stale data, we believe that fetch time complexity[9] on the most recent(s) version(s) of the dataset should not depend on the size of the archive history. Hence, and depending on the host available main memory, an RDF archiving system could keep the latest revision(s) of a dataset (or parts of it) in main memory or in optimized disk-based stores for faster query response time (as done by QuitStore [12]). Hence, main memory consumption could also be part of the optimization objective.

[9]For single triple patterns on VM queries

Internal serialization. Archiving multi-graph datasets requires the serialization of 5-tuples, which complexifies the trade-offs between space (i.e., disk and main memory consumption) and runtime efficiency (i.e., response time, ingestion time). For example, dealing with more columns increases the number of possible index combinations. Also, it leads to more data redundancy, since a triple can be associated to multiple values for the fourth and fifth component. Classical solutions for metadata in RDF include reification [70], singleton properties [60], and named graphs [69]. Reification assigns each RDF statement (triple or quad) an identifier $t \in \mathcal{I}$ that can be then used to link the triple to its ρ and ζ components in the 5-tuples data model introduced in Section 2.3. While simple and fully compatible with the existing RDF standards, reification is well-known to incur serious performance issues for storage and query efficiency, e.g., it would quintuple the number of triple patterns in SPARQL queries. On those grounds, Nguyen et al. [60] proposes singleton properties to piggyback the metadata in the predicate component. In this strategy, predicates take the form $p\#m \in \mathcal{I}$ for some $m \in \mathcal{N}$ and every triple with p in the dataset. This scheme gives $p\#m$ the role of ρ in the aforementioned data model reducing the overhead of reification. However, singleton properties would still require an additional level of reification for the fifth component ζ. The same is true for a solution based on named graphs. A more recent solution is HDTQ [28], which extends HDT with support for quads. An additional extension could account for a fifth component. Systems such as Dydra [11] or v-RDFCSA [23] resort to bit vectors and visibility maps for triples and quads. We argue that vector and matrix representations may be suitable for scalable RDF archiving as they allow for good compression in the presence of high redundancy: If we assume a binary matrix from triples (rows) to graph revisions (columns) where a one denotes the presence of a triple in a revision, we would expect rows and columns to contain many contiguous ones – the logic is analogous for removed triples.

Accounting for evolution patterns. As our study in Section 4 shows, the evolution patterns of RDF archives can change throughout time leading even, to decreases in dataset size. With that in mind, we envision an adaptive data-oriented system that adjusts its parameters according to the archive's evolution for the sake of efficient resource comsumption. Parameter tuning could rely on the metrics proposed in Section 3. Nonetheless, these desiderata translate into some design and engineering considerations. For example, we saw in Section 6 that a large number of deletions can negatively impact Ostrich's query runtime, hence, such an event could trigger the construction of a complete snapshot of the dataset in order to speed-up VM queries (assuming the existence of a cost model for query runtime). In the same spirit and assuming some sort of dictionary encoding, an increase in the vocabulary dynamicity could increase the number of bits used to encode the identifiers of RDF terms in the dictionary. Those changes could be automatically carried out by the archiving engine, but could also be manually set up by the system administrator after an analysis with RDFev. A design philosophy that we envision to explore divides the history of each graph in the dataset in intervals such that each interval is associated to a block file. This file contains a full snapshot plus all the changesets in the interval. It follows that the application of a new changeset may update the latest block file or create a new one (old blocks could be merged into snapshots to save disk space). This action could be automatically executed by the engine or triggered by the system administrator. For instance, if the archive is the backend of a version control system, new branches may always trigger the creation of snapshots. This base architecture should be enhanced with additional indexes to speed up V queries and adapted compression for the dictionary and the triples.

Finally as we expect long dataset histories, it is vital for solutions to improve their ingestion time complexity, which should depend on the size of the changesets rather than on history size—contrary to what we observed in Section 6 for Ostrich. This constraint could be taken into account by the storage policy for the creation of storage structures such as deltas, snapshots, or indexes (e.g., by reducing the length of delta chains for redundant changesets). Nevertheless, very large changesets may still be challenging, specially in the concurrent scenario. This may justify the creation of temporary incremental (in-memory) indexes and data structures optimized for asynchronous batch updates as proposed in x-RDF-3X [59].

8. Conclusions

In this paper we have discussed the importance of RDF archiving for both maintainers and consumers of RDF data. Besides, we have discussed the importance of evolution patterns in the design of a fully-fledged RDF archiving solution. On these grounds, we have proposed a metric-based framework to characterize the evolution of RDF data, and we have applied our framework to study the history of three challenging RDF datasets, namely DBpedia, YAGO, and Wikidata. This study has allowed us to characterize the history of these datasets in terms of changes at the level of triples, vocabulary terms, and entities. It has also allowed us to identify design shifts in their release history. Those insights can be used to optimize the allocation of resources for archiving, for example, by triggering the creation of a new snapshot as a response to a large changeset.

In other matters, our survey and study of the existing solutions and benchmarks for RDF archiving has shown that only a few solutions are available for download and use, and that among those, only Ostrich can store the release history of very large RDF datasets. Nonetheless, its design still does not scale to long histories and does not exploit the data evolution patterns. R43ples [36], R&WBase [72], Quit Store [12], and x-RDF-3X [59] are also available, however they are still far from tackling the major challenges of this task, mainly because, they are conceived for collaborative version control, which is an application of RDF archiving in itself. Our survey also reveals that the state of the art lacks a standard to query RDF archives. We think that a promising solution is to use SPARQL-star combined with additional syntactic sugar as proposed by some approaches [11, 30, 36]

Finally, we have used all these observations to derive a set of design lessons in order to overcome the gap between the literature and a fully functional solution for large RDF archives. All in all, we believe that such a solution should (i) support global histories for RDF datasets, (i) resort to a modular architecture that decouples the storage from the application layers, (iii) handle provenance and domain-specific temporal metadata, (iv) implement a SPARQL extension to query archives, (v) use a metric-based approach to monitor the data evolution and adapt resource consumption accordingly, and (vi) provide a performance that does not depend on the length of the history. The major algorithmic challenges in the field lie in how to handle the inherent trade-offs between disk usage, ingestion time, and query runtime. With this detailed study and the derived guidelines, we aim at paving the way towards an ultimate solution for this problem. In this sense, we envision archiving solutions to not only serve as standalone single server systems but also as components of the RDF ecosystem on the Web in all its flavors covering federated [5, 55, 68, 73] and client-server architectures [6, 14, 15, 41, 53, 54, 80] as well as peer-to-peer [7, 8] solutions.

Acknowledgements

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B and the Poul Due Jensen Foundation.

References

[1] European Open Data Portal. https://data.europa.eu/data/. Accessed: 2020-06-09.

[2] RDF Exports from Wikidata. Available at tools.wmflabs.org/wikidata-exports/rdf/index.html.

[3] Stardog. http://stardog.com. Accessed: 2020-06-09.

[4] USA Data Gov. portal. https://www.data.gov/. Accessed: 2020-06-09.

[5] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011*, pages 18–34, 2011. doi:10.1007/978-3-642-25073-6_2.

[6] Christian Aebeloe, Ilkcan Keles, Gabriela Montoya, and Katja Hose. Star Pattern Fragments: Accessing Knowledge Graphs through Star Patterns. *CoRR*, abs/2002.09172, 2020. URL: https://arxiv.org/abs/2002.09172, arXiv:2002.09172.

[7] Christian Aebeloe, Gabriela Montoya, and Katja Hose. A Decentralized Architecture for Sharing and Querying Semantic Data. In *ESWC*, volume 11503, pages 3–18. Springer, 2019. doi:10.1007/978-3-030-21348-0_1.

[8] Christian Aebeloe, Gabriela Montoya, and Katja Hose. Decentralized Indexing over a Network of RDF Peers. In *ISWC*, volume 11778, pages 3–20. Springer, 2019. doi:10.1007/978-3-030-30793-6_1.

[9] Christian Aebeloe, Gabriela Montoya, and Katja Hose. ColChain: Collaborative Linked Data Networks. In *WWW '21: The Web Conference 2021*. ACM / IW3C2, 2021. to appear.

[10] Alex B. Andersen, Nurefsan Gür, Katja Hose, Kim Ahlstrøm Jakobsen, and Torben Bach Pedersen. Publishing Danish Agricultural Government Data as Semantic Web Data. In *Semantic Technology - 4th Joint International Conference, JIST 2014, Chiang Mai, Thailand, November 9-11, 2014. Revised Selected Papers*, pages 178–186. Springer, 2014. doi:10.1007/978-3-319-15615-6_13.

[11] James Anderson and Arto Bendiken. Transaction-Time Queries in Dydra. In *MEPDaW/LDQ@ESWC*, volume 1585 of *CEUR Workshop Proceedings*, pages 11–19. CEUR-WS.org, 2016.

[12] Natanael Arndt, Patrick Naumann, Norman Radtke, Michael Martin, and Edgard Marx. Decentralized collaborative knowledge management using git. *Journal of Web Semantics*, 54:29 – 47, 2019. Managing the Evolution and Preservation of the Data Web. URL: http://www.sciencedirect.com/science/article/pii/S1570826818300416, doi:https://doi.org/10.1016/j.websem.2018.08.002.

[13] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, pages 722–735. 2007. doi:10.1007/978-3-540-76298-0_52.

[14] Amr Azzam, Christian Aebeloe, Gabriela Montoya, Ilkcan Keles, Axel Polleres, and Katja Hose. WiseKG: Balanced Access to Web Knowledge Graphs. In *WWW '21: The Web Conference 2021*. ACM / IW3C2, 2021. to appear.

[15] Amr Azzam, Javier D. Fernández, Maribel Acosta, Martin Beno, and Axel Polleres. SMART-KG: Hybrid Shipping for SPARQL Querying on the Web. In *WWW '20: The Web Conference 2020*, pages 984–994. ACM / IW3C2, 2020. doi:10.1145/3366423.3380177.

[16] Konstantina Bereta, Panayiotis Smeros, and Manolis Koubarakis. Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In *Extended Semantic Web Conference*, pages 259–274, 2013. doi:10.1007/978-3-642-38288-8_18.

[17] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The Semantic Web. *Scientific American*, 284(5):28–37, 2001. doi:10.1007/978-0-387-30440-3_478.

[18] Christian Bizer. The Emerging Web of Linked Data. *IEEE Intelligent Systems*, 24(5):87–92, 2009. doi:10.1109/MIS.2009.102.

[19] Nieves R. Brisaboa, Ana Cerdeira-Pena, Antonio Fariña, and Gonzalo Navarro. A Compact RDF Store Using Suffix Arrays. In *String Processing and Information Retrieval*, pages 103–115, 2015. doi:10.1007/978-3-319-23826-5_11.

[20] Jörg Brunsmann. Archiving Pushed Inferences from Sensor Data Streams. In *Proceedings of the International Workshop on Semantic Sensor Web*, pages 38–46. INSTICC, 2010. doi:10.5220/0003116000380046.

[21] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, 2010. doi:10.5555/2898607.2898816.

[22] Steve Cassidy and James Ballantine. Version Control for RDF Triple Stores. *ICSOFT (ISDM/EHST/DC)*, 7:5–12, 2007. doi:10.1142/S0218194012500040.

[23] Ana Cerdeira-Pena, Antonio Fariña, Javier D. Fernández, and Miguel A. Martínez-Prieto. Self-Indexing RDF Archives. In *DCC*, pages 526–535. IEEE, 2016. doi:10.1109/DCC.2016.40.
[24] The World Wide Web Consortium. PROV-O: The PROV Ontology. http://www.w3.org/TR/prov-o, 2013.
[25] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. LODStats: The Data Web Census Dataset. In *Proceedings of 15th International Semantic Web Conference - Resources Track*, 2016. URL: http://svn.aksw.org/papers/2016/ISWC_LODStats_Resource_Description/public.pdf.
[26] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing Wikidata to the Linked Data Web. In *International Semantic Web Conference*, pages 50–65, 2014. doi:10.1007/978-3-319-11964-9_4.
[27] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary RDF Representation for Publication and Exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013. doi:10.1016/j.websem.2013.01.002.
[28] Javier D. Fernández, Miguel A. Martínez-Prieto, Axel Polleres, and Julian Reindorf. HDTQ: Managing RDF Datasets in Compressed Space. In *The Semantic Web*, pages 191–208, 2018. doi:10.1007/978-3-319-93417-4_13.
[29] Javier D Fernández, Jürgen Umbrich, Axel Polleres, and Magnus Knuth. Evaluating Query and Storage Strategies for RDF Archives. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 41–48, 2016. doi:10.3233/SW-180309.
[30] Valeria Fionda, Melisachew Wudage Chekol, and Giuseppe Pirrò. Gize: A Time Warp in the Web of Data. In *International Semantic Web Conference (Posters & Demos)*, volume 1690, 2016.
[31] The Apache Software Foundation. Apache Jena Semantic Web Framework. jena.apache.org.
[32] Johannes Frey, Marvin Hofer, Daniel Obraczka, Jens Lehmann, and Sebastian Hellmann. DBpedia FlexiFusion the Best of Wikipedia > Wikidata > Your Data. In *The Semantic Web – ISWC 2019*, pages 96–112, 2019. doi:10.1007/978-3-030-30796-7_7.
[33] Luis Galárraga, Kim Ahlstrøm, Katja Hose, and Torben Bach Pedersen. Answering Provenance-Aware Queries on RDF Data Cubes Under Memory Budgets. In *The Semantic Web – ISWC 2018*, pages 547–565, 2018. doi:10.1007/978-3-030-00671-6_32.
[34] Shi Gao, Jiaqi Gu, and Carlo Zaniolo. RDF-TX: A Fast, User-Friendly System for Querying the History of RDF Knowledge Bases. In *Proceedings of the Extended Semantic Web Conference*, pages 269–280, 2016. doi:10.1016/j.ic.2017.08.012.
[35] Fabio Grandi. T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *Local Proceedings of the Fourteenth East-European Conference on Advances in Databases and Information Systems*, pages 21–30, 2010. doi:10.1145/3180374.3181338.
[36] Markus Graube, Stephan Hensel, and Leon Urbas. R43ples: Revisions for Triples. In *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems (SEMANTiCS)*, 2014.
[37] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005. doi:10.1016/j.websem.2005.06.005.
[38] Nurefsan Gür, Jacob Nielsen, Katja Hose, and Torben Bach Pedersen. GeoSemOLAP: Geospatial OLAP on the Semantic Web Made Easy. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 213–217. ACM, 2017. doi:10.1145/3041021.3054731.
[39] Nurefsan Gür, Torben Bach Pedersen, Esteban Zimányi, and Katja Hose. A foundation for spatial data warehouses on the semantic web. *Semantic Web*, 9(5):557–587, 2018. doi:10.3233/SW-170281.
[40] Olaf Hartig. Foundations of RDF⋆ and SPARQL⋆: An Alternative Approach to Statement-Level Metadata in RDF. In *Proc. of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management*, AMW, 2017.
[41] Olaf Hartig and Carlos Buil Aranda. Bindings-restricted triple pattern fragments. In *On the Move to Meaningful Internet Systems: OTM 2016 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016*, pages 762–779, 2016. doi:10.1007/978-3-319-48472-3_48.
[42] Olaf Hartig, Katja Hose, and Juan F. Sequeda. Linked Data Management. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019. doi:10.1007/978-3-319-63962-8_76-1.
[43] Ian Horrocks, Thomas Hubauer, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Manolis Koubarakis, Ralf Möller, Konstantina Bereta, Christian Neuenstadt, Özgür Özçep, Mikhail Roshchin, Panayiotis Smeros, and Dmitriy Zheleznyakov. Addressing Streaming and Historical Data in OBDA Systems: Optique's Approach (Statement of Interest). In *Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data (Know@LOD)*, 2013.
[44] Katja Hose and Ralf Schenkel. RDF stores. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018. doi:10.1007/978-1-4614-8265-9_80676.
[45] Thomas Huet, Joanna Biega, and Fabian M. Suchanek. Mining History with Le Monde. In *Proceedings of the Workshop on Automated Knowledge Base Construction*, pages 49–54, 2013. doi:10.1145/2509558.2509567.
[46] Dong hyuk Im, Sang won Lee, and Hyoung joo kim. A Version Management Framework for RDF Triple Stores. *International Journal of Software Engineering and Knowledge Engineering*, 22, 04 2012. doi:10.1142/S0218194012500040.
[47] Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Towards Exploratory OLAP Over Linked Open Data - A Case Study. In *Enabling Real-Time Business Intelligence - International Workshops, BIRTE 2013*, volume 206, pages 114–132. Springer, 2014. doi:10.1007/978-3-662-46839-5_8.
[48] Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Processing Aggregate Queries in a Federation of SPARQL Endpoints. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, volume 9088, pages 269–285. Springer, 2015. doi:10.1007/978-3-319-18818-8_17.

[49] Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. Optimizing Aggregate SPARQL Queries Using Materialized RDF Views. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, pages 341–359, 2016. doi:10.1007/978-3-319-46523-4_21.

[50] Kim Ahlstrøm Jakobsen, Alex B. Andersen, Katja Hose, and Torben Bach Pedersen. Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries. In *International Workshop on Consuming Linked Data (COLD) co-located with 14th International Semantic Web Conference (ISWC 2105)*, volume 1426. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1426/paper-02.pdf.

[51] Marios Meimaris. EvoGen: A Generator for Synthetic Versioned RDF. In *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016*, volume 1558, 2016. URL: http://ceur-ws.org/Vol-1558/paper9.pdf.

[52] Steffen Metzger, Katja Hose, and Ralf Schenkel. Colledge: a vision of collaborative knowledge networks. In *Proceedings of the 2nd International Workshop on Semantic Search over the Web*, pages 1–8. ACM, 2012. doi:10.1145/2494068.2494069.

[53] Thomas Minier, Hala Skaf-Molli, and Pascal Molli. SaGe: Web Preemption for Public SPARQL Query Services. In *The World Wide Web Conference, WWW 2019*, pages 1268–1278. ACM, 2019. doi:10.1145/3308558.3313652.

[54] Gabriela Montoya, Christian Aebeloe, and Katja Hose. Towards Efficient Query Processing over Heterogeneous RDF Interfaces. In *Proceedings of the 2nd Workshop on Decentralizing the Semantic Web co-located with the 17th International Semantic Web Conference, DeSemWeb@ISWC*, volume 2165 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2165/paper4.pdf.

[55] Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. The Odyssey Approach for Optimizing Federated SPARQL Queries. In *International Semantic Web Conference*, volume 10587, pages 471–489. Springer, 2017. doi:10.1007/978-3-319-68288-4_28.

[56] Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, and Oscar Romero. SETL: A programmable semantic extract-transform-load framework for semantic data warehouses. *Inf. Syst.*, 68:17–43, 2017. doi:10.1016/j.is.2017.01.005.

[57] Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, Oscar Romero, and Amrit Bhattacharjee. SET_{LBI}: An Integrated Platform for Semantic Business Intelligence. In *Proceedings of The 2020 World Wide Web Conference*, 2020. doi:10.1145/3366424.3383533.

[58] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008. doi:10.14778/1453856.1453927.

[59] Thomas Neumann and Gerhard Weikum. x-rdf-3x: Fast querying, high update rates, and consistency for rdf databases. *Proceedings of the VLDB Endowment*, 3(1-2):256–263, 2010. doi:10.14778/1920841.1920877.

[60] Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 759–770, 2014. doi:10.1145/2566486.2567973.

[61] Natasha Noy and Mark Musen. Ontology Versioning in an Ontology Management Framework. *Intelligent Systems, IEEE*, 19:6–13, 08 2004. doi:10.1109/MIS.2004.33.

[62] George Papadakis, Konstantina Bereta, Themis Palpanas, and Manolis Koubarakis. Multi-core Meta-blocking for Big Linked Data. In *Proceedings of the 13th International Conference on Semantic Systems*, pages 33–40, 2017. doi:10.1145/3132218.3132230.

[63] Vicky Papavassiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. On Detecting High-level Changes in RDF/S KBs. In *International Semantic Web Conference (ISWC)*, pages 473–488, 2009. doi:10.1007/978-3-642-04930-9_30.

[64] Thomas Pellissier Tanon, Camille Bourgaux, and Fabian Suchanek. Learning How to Correct a Knowledge Base from the Edit History. In *The World Wide Web Conference*, pages 1465–1475, 2019. doi:10.1145/3308558.3313584.

[65] Thomas Pellissier Tanon and Fabian M. Suchanek. Querying the Edit History of Wikidata. In *Extended Semantic Web Conference*, 2019. doi:10.1007/978-3-030-32327-1_32.

[66] Matthew Perry, Prateek Jain, and Amit P. Sheth. SPARQL-ST: Extending SPARQL to Support Spatio-temporal Queries. In *Geospatial Semantics and the Semantic Web*, volume 12, pages 61–86, 2011. doi:10.1007/978-1-4419-9446-2_3.

[67] Maria Psaraki and Yannis Tzitzikas. CPOI: A Compact Method to Archive Versioned RDF Triple-Sets, 2019. arXiv:1902.04129.

[68] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, pages 524–538, 2008. doi:10.1007/978-3-540-68234-9_39.

[69] Yves Raimond and Guus Schreiber. RDF 1.1 primer. W3C recommendation, 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[70] Yves Raimond and Guus Schreiber. RDF 1.1 Semantics. W3C recommendation, 2014. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[71] Yannis Roussakis, Ioannis Chrysakis, Kostas Stefanidis, Giorgos Flouris, and Yannis Stavrakas. A Flexible Framework for Understanding the Dynamics of Evolving RDF Datasets. In *International Semantic Web Conference (ISWC)*, 2015. doi:10.1007/978-3-319-25007-6_29.

[72] Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle. R&Wbase: Git for triples. *Linked Data on the Web Workshop*, 2013.

[73] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *International Semantic Web Conference*, volume 7031, pages 601–616. Springer, 2011. doi:10.1007/978-3-642-25073-6_38.

[74] Andy Seaborne and Steven Harris. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[75] Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don S. Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna G. Kulkarni, T. Y. Cliff Leung, Nikos A. Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, and Suryanarayana M. Sripada. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, 1994. doi:10.1145/181550.181562.

[76] OpenLink Software. OpenLink Virtuoso. http://virtuoso.openlinksw.com.

[77] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Large Ontology from Wikipedia and Wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008. doi:10.1016/j.websem.2008.06.001.

[78] Ruben Taelman, Miel Vander Sande, and Ruben Verborgh. OSTRICH: Versioned Random-Access Triple Store. In *Companion of the The Web Conference 2018, WWW*, pages 127–130, 2018. doi:10.1145/3184558.3186960.

[79] Mayesha Tasnim, Diego Collarana, Damien Graux, Fabrizio Orlandi, and Maria-Esther Vidal. Summarizing Entity Temporal Evolution in Knowledge Graphs. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 961–965, 2019. doi:10.1145/3308560.3316521.

[80] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple Pattern Fragments: A low-cost knowledge graph interface for the Web. *J. Web Semant.*, 37-38:184–206, 2016. doi:10.1016/j.websem.2016.03.003.

[81] Max Volkel, Wolf Winkler, York Sure, Sebastian Ryszard Kruk, and Marcin Synak. SemVersion: A Versioning System for RDF and Ontologies. *Second European Semantic Web Conference*, 2005.

[82] Antoine Zimmermann, Nuno Lopes, Axel Polleres, and Umberto Straccia. A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. *Web Semantics*, 11:72–95, 2012. doi:10.1016/j.websem.2011.08.006.