Live Storage and Querying of Versioned Datasets on the Web

Ruben Taelman, Miel Vander Sande,
Ruben Verborgh, and Erik Mannens

IDLab, Department of Electronics and Information Systems, Ghent University – imec, Belgium
{firstname.lastname}@ugent.be

Abstract. Linked Datasets often evolve over time for a variety of reasons. While typical scenarios rely on the latest version only, useful knowledge may still be contained within or between older versions, such as the historical information of biomedical patient data. In order to make this historical information cost-efficiently available on the Web, a low-cost interface is required for providing access to versioned datasets. For our demonstration, we set up a live Triple Pattern Fragments interface for a versioned dataset with queryable access. We explain different version query types of this interface, and how it communicates with a storage solution that can handle these queries efficiently.

Keywords: Linked Data, versioning, Triple Pattern Fragments, OSTRICH, SPARQL

1 Introduction

Linked Datasets often change over time [7], because the underlying information changes, or mistakes are corrected. Currently, most data publishers provide queryable access to only the latest version of their datasets. Other data publishers, such as DBpedia [1], provide data dumps of previous versions of their datasets. Historical information and its evolution might still contain a lot of useful knowledge, such as historical information of biomedical patient data, or for the analysis of concept drift. Because of this, we need queryable access to this data at different versions.

Querying on the Web is possible using the Triple Pattern Fragments (TPF) framework [8], which was introduced as an alternative to SPARQL endpoints [2] for publishing Linked Data at a low cost, while still enabling queryable access. The TPF approach limits server interfaces to triple pattern queries and moves the effort of full SPARQL query evaluation to the client. TPF interfaces are based on the REST principles and provide declarative hypermedia controls with the Hydra Core Vocabulary [5] using which clients can discover the triple pattern query controls. Furthermore, each response contains metadata to help client-side query evaluation, such as an estimated query result counts.

Querying versioned datasets is possible using different query atoms [4]. In this demo, we focus on the following realistic query atoms:

- *Version Materialization* (VM) queries *at* a single version.
- *Delta Materialization* (DM) queries differences *between* versions.
- *Version Queries* (VQ) annotate results *for* different versions.

Each query engine needs some form of storage solution behind it. Strategies for storing different dataset versions can be categorized in the following storage policies [3]:

1. *Independent copies* (IC) for storing fully materialized snapshots for each version.
2. *Change-based* (CB) will only store differences between consecutive versions.
3. *Timestamp-based* (TB) annotates each triple with the versions for which it exists.

In this paper, we demonstrate an extension of the TPF interface that has support for the three versioned query atoms, which is discussed in Section 2. We illustrate the feasibility of this interface by publishing a large versioned dataset publicly, using a new storage solution that is optimized for triple pattern queries for each of these query atoms, as shown in Section 3. Finally, in Section 4, we demonstrate example usage of the interface.

2 Interface

In this section, we discuss *Versioned Triple Pattern Fragments* (VTPF), a versioning feature for the TPF interface that supports VM, DM and VQ queries.

Support for these query atoms is possible by adding three new query forms to the interface. Each of these forms supports the basic triple patterns. The VM form adds a *version* parameter for selecting the version in which the triple pattern query should be evaluated. For the DM form, a *start* and *end* parameter is added for selecting the version range in between which changes should be queried given a triple pattern. Triples are then annotated as *addition* or *deletion*. Finally, for VQ queries, no additional parameter is added but triples matching a given triple pattern are annotated with versions.

These forms are made available both in HTML and RDF representations. An example of the HTML form for VM queries can be seen in Fig. 1. Listing 1.1 shows a hypermedia control for VM queries in RDF. We use the *version* vocabulary[1] for these hypermedia controls, metadata and result representation.

[1] http://w3id.org/version/ontology

```
<http://versioned.linkeddatafragments.org/bear#metadata> {
  <http://versioned.linkeddatafragments.org/bear#dataset> hydra:search [
    hydra:template "http://versioned.linkeddatafragments.org/bear?versionType=
     VersionMaterialized{&s,p,o,v}";
    hydra:variableRepresentation hydra:ExplicitRepresentation;
    hydra:mapping [ hydra:variable "s"; hydra:property rdf:subject        ],
                  [ hydra:variable "p"; hydra:property rdf:predicate      ],
                  [ hydra:variable "o"; hydra:property rdf:object         ],
                  [ hydra:variable "v"; hydra:property ver:relatedVersion ]
  ].
}
```

Listing 1.1: The VM query RDF form using the Hydra Core Vocabulary.

Fig. 1: VM query HTML form and results.

3 Storage

In this section, we give a high-level overview of OSTRICH, which is a new storage solution that is used as a back-end for our VTPF instance.

Previous work [4, 6] has shown that the complexity for different versioning query atoms depends on the storage policy. VTPF supports the VM, DM and VQ query atoms, where each of them can be disabled in case the cost for publishing would be too high, when for example a storage policy is in place that is inefficient for certain query atoms. For our demonstration, we enable the three query atoms, because OSTRICH is a hybrid IC–CB–TB approach, which enables efficient VM, DM and VQ querying. OSTRICH supports these query atoms for simple triple pattern queries, which is the only type of query that is required for a back-end TPF datasource. Furthermore, the TPF approach requires each fragment to contain metadata about the estimated number of matching triples for each triple pattern. For this, OSTRICH supports (approximate) count queries for all query atoms for triple patterns.

For our demonstration, we used the first ten versions of the RDF archive provided by the BEAR benchmark [4]. Each version in this dataset is a weekly snapshot from the Dynamic Linked Data Observatory[2]. When combined, these ten versions are approximately 45GB large in N-Triples format, and 3GB when gzipped. OSTRICH requires less than 6GB for storing these versions, which is significantly less than the N-Triples format, i.e., it only requires 13% of the storage space. Even though OSTRICH requires twice as much storage space for this dataset when compared to gzip, it still provides queryable access, which is not possible with gzip. On average, OSTRICH evaluates triple pattern queries for any of the versions in approximately 1ms, which is sufficiently fast for a Web interface back-end system.

4 Demonstration Overview

We published the first ten versions of the BEAR dataset using a VTPF interface with an OSTRICH back-end at `http://versioned.linkeddatafragments.org/bear`. This enables

[2] `http://swse.deri.org/dyldo/`

queryable access to this dataset *at*, *between* and *for* different versions using triple patterns. Visiting the interface using a webbrowser allows humans to consume the data as shown in Fig. 1. Machines can also consume the data by setting an RDF serialization format in the accept header[3].

We list several example queries about "WebDeveloper1" that can be performed using the interface:

- Information in version 6:
  `http://versioned.linkeddatafragments.org/bear?object=%22WebDeveloper1%22&versionType=VersionMaterialized&version=6`
- Changes between version 6 and 9:
  `http://versioned.linkeddatafragments.org/bear?object=%22WebDeveloper1%22&versionType=DeltaMaterialized&versionStart=6&versionEnd=9`
- Version query for all information:
  `http://versioned.linkeddatafragments.org/bear?object=%22WebDeveloper1%22&versionType=Version`

With this demonstration, we show the feasibility of the VTPF interface for exposing versioned Linked Datasets on the Web, with queryable triple pattern access. In future work, we will extend the TPF client [8] with support for this interface, so that it is able to consume data using the version query atoms for more complex queries, using an appropriate query language.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A nucleus for a Web of open data. In: The semantic web, pp. 722–735. Springer (2007)
2. Feigenbaum, L., Todd Williams, G., Grant Clark, K., Torres, E.: SPARQL 1.1 protocol. Rec., W3C (Mar 2013), `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`
3. Fernández, J.D., Polleres, A., Umbrich, J.: Towards efficient archiving of dynamic Linked Open Data. In: Proceedings of DIACHRON (2015)
4. Fernández, J.D., Umbrich, J., Polleres, A., Knuth, M.: Evaluating query and storage strategies for RDF archives. In: Proceedings of the 12th International Conference on Semantic Systems (2016)
5. Lanthaler, M., Gütl, C.: Hydra: A vocabulary for hypermedia-driven Web APIs. In: Proceedings of the 6th Workshop on Linked Data on the Web (May 2013)
6. Taelman, R., Verborgh, R., Mannens, E.: Exposing RDF archives using Triple Pattern Fragments. In: Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management: Posters and Demos (Nov 2016)
7. Umbrich, J., Decker, S., Hausenblas, M., Polleres, A., Hogan, A.: Towards dataset dynamics: Change frequency of Linked Open Data sources. 3rd International Workshop on Linked Data on the Web (LDOW) (2010)
8. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. Journal of Web Semantics 37–38 (Mar 2016)

[3] Request data in TriG using curl: `curl -H "Accept: application/trig" "http://versioned.linkeddatafragments.org/bear"`