Low-Cost Queryable Linked Data through Triple Pattern Fragments

Ruben Verborgh[1], Olaf Hartig[2], Ben De Meester[1], Gerald Haesendonck[1], Laurens De Vocht[1], Miel Vander Sande[1], Richard Cyganiak[3], Pieter Colpaert[1], Erik Mannens[1], and Rik Van de Walle[1]

[1] Ghent University – iMinds, Belgium
{firstname.lastname}@ugent.be
[2] University of Waterloo, Canada
ohartig@uwaterloo.ca
[3] Digital Enterprise Research Institute, NUI Galway, Ireland
richard@cyganiak.de

Abstract. For publishers of Linked Open Data, providing queryable access to their dataset is costly. Those that offer a public SPARQL endpoint often have to sacrifice high availability; others merely provide non-queryable means of access such as data dumps. We have developed a client-side query execution approach for which servers only need to provide a lightweight triple-pattern-based interface, enabling queryable access at low cost. This paper describes the implementation of a client that can evaluate SPARQL queries over such triple pattern fragments of a Linked Data dataset. Graph patterns of SPARQL queries can be solved efficiently by using metadata in server responses. The demonstration consists of SPARQL client for triple pattern fragments that can run as a standalone application, browser application, or library.

Keywords: Linked Data, Linked Data Fragments, querying, availability, scalability, SPARQL

1 Introduction

An ever increasing amount of Linked Data is published on the Web, a large part of which is freely and publicly available. The true value of these datasets becomes apparent when users can execute arbitrary queries over them, to retrieve precisely those facts they are interested in. The SPARQL query language [3] allows to specify highly precise selections, but it is very costly for servers to offer a public SPARQL endpoint over a large dataset [6]. As a result, current public SPARQL endpoints, often hosted by institutions that cannot afford an expensive server setup, suffer from low availability rates [1]. An alternative for these institutions is to provide their data in a non-queryable form, for instance, by allowing consumers to download a data dump which they can use to set up their own private SPARQL endpoint. However, this prohibits live querying of the data, and is in turn rather expensive on the client side.

In this demo, we will show a low-cost server interface that offers access to a dataset through all of its triple patterns, together with a client that performs efficient execution of complex queries through this interface. This enables publishers to provide Linked Data in a queryable way at low cost. The demo complements our paper at the ISWC2014 Research Track [6], which profoundly explains the principles behind the technology and experimentally verifies its scalability. The present paper details the implementation and introduces the supporting prototype implementation of our SPARQL client of triple pattern fragments.

2 Related Work

We contrast our approach with the three categories of current HTTP interfaces to RDF, each of which comes with its own trade-offs regarding performance, bandwidth, and client/server processor usage and availability.

Public SPARQL endpoints The current de-facto way for providing queryable access to triples on the Web is the SPARQL protocol, which is supported by many triple stores such as Virtuoso, AllegroGraph, Sesame, and Jena TDB. Even though current SPARQL interfaces offer high performance, individual queries can consume a significant amount of server processor time and memory. Because each client requests unique, highly specific queries, regular HTTP caching is ineffective, since this can only optimize repeated identical requests. These factors contribute to the low availability of public SPARQL endpoints, which has been documented extensively [1]. This makes providing reliable public SPARQL endpoints an exceptionally difficult challenge, incomparable to hosting regular public HTTP servers.

Linked Data servers Perhaps the most well-known alternative interface to triples is described by the Linked Data principles. The principles require servers to publish documents with triples about specific entities, which the client can access through their entity-specific URI, a process which is called *dereferencing.* Each of these Linked Data documents should contain data that mention URIs of other entities, which can be dereferenced in turn. Several Linked Data querying techniques [4] use dereferencing to solve queries over the Web of Data. This process happens client-side, so the availability of servers is not impacted. However, execution times are high, and many queries cannot be solved (efficiently) [6].

Other HTTP interfaces for triples Additionally, several other HTTP interfaces for triples have been designed. Strictly speaking, the most trivial HTTP interface is a data dump, which is a single-file representation of a dataset. The Linked Data Platform [5] is a read/write HTTP interface for Linked Data, scheduled to become a W3C recommendation. It details several concepts that extend beyond the Linked Data principles, such as containers and write access. However, the API has been designed primarily for consistent read/write access to Linked Data resources, not to enable reliable and/or efficient query execution. The interface we will discuss next offers low-cost publishing and client-side querying.

3 Linked Data Fragments and Triple Pattern Fragments

Linked Data Fragments [6] enable a uniform view on all possible HTTP interfaces for triples, and allow to define new interfaces with different trade-offs.

Definition 1. *A **Linked Data Fragment** (LDF) of a dataset is a resource consisting of those triples of this dataset that match a specific selector, together with their metadata and hypermedia controls to retrieve other Linked Data Fragments.*

We define a specific type of LDFs that require minimal effort to generate by a server, while still enabling efficient querying on the client side:

Definition 2. *A **triple pattern fragment** is a Linked Data Fragment with a triple pattern as selector, count metadata, and the controls to retrieve any other triple pattern fragment of the dataset. Each **page** of a triple pattern fragment contains a subset of the matching triples, together with all metadata and controls.*

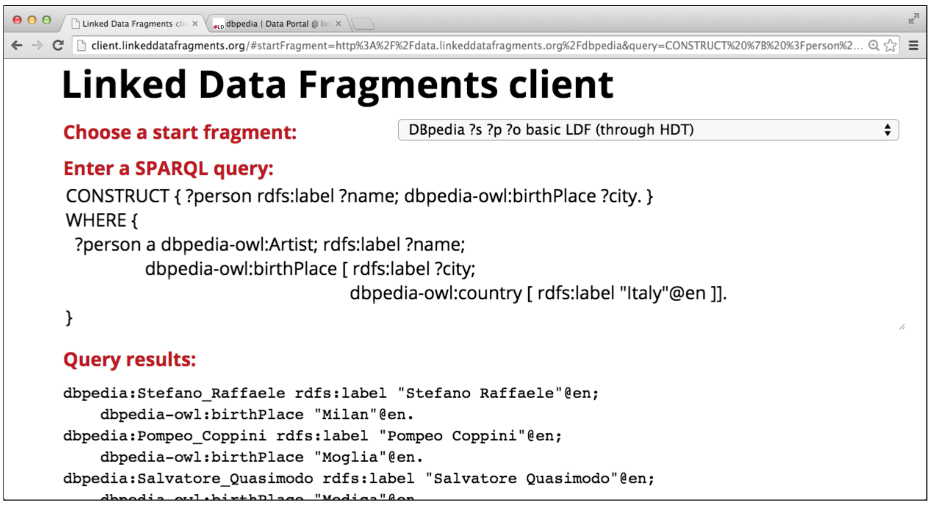
Triple pattern fragments can be generated easily, as triple-pattern selection is an indexed operation in the majority of triple stores. Furthermore, specialized formats such as the compressed RDF HDT (Header – Dictionary – Triples [2]) natively support fast triple-pattern extraction. This ensures low-cost servers.

Clients can then efficiently evaluate SPARQL queries over the remote dataset because each page contains an estimate of the total number of matching triples. This allows efficient asymmetric joins by first binding those triple patterns with the lowest number of matches. For basic graph patterns (BGPs), which are the main building blocks of SPARQL queries, the algorithm works as follows:

1. For each triple pattern tp_i in the BGP $B = \{tp_1, \ldots, tp_n\}$, fetch the first page ϕ_1^i of the triple pattern fragment f_i for tp_i, which contains an estimate cnt_i of the total number of matches for tp_i. Choose ϵ such that $cnt_\epsilon = \min(\{cnt_1, \ldots, cnt_n\})$. f_ϵ is then the optimal fragment to start with.
2. Fetch all remaining pages of the triple pattern fragment f_ϵ. For each triple t in the LDF, generate the solution mapping μ_t such that $\mu_t(tp_\epsilon) = t$. Compose the subpattern $B_t = \{tp \mid tp = \mu_t(tp_j) \wedge tp_j \in B\} \setminus \{t\}$. If $B_t \neq \emptyset$, find mappings Ω_{B_t} by recursively calling the algorithm for B_t. Else, $\Omega_{B_t} = \{\mu_\emptyset\}$ with $\mu_\emptyset$ the empty mapping.
3. Return all solution mappings $\mu \in \{\mu_t \cup \mu' \mid \mu' \in \Omega_{B_t}\}$.

4 Demo of Client-side Querying

The above recursive algorithm has been implemented by a dynamic pipeline of iterators [6]. At the deepest level, a client uses `TriplePatternIterator`s to retrieve pages of triple pattern fragments from the server, turning the triples on those pages into bindings. A basic graph pattern of a SPARQL query is evaluated by a `GraphPatternIterator`, which first discovers the triple pattern in this graph with the lowest number of matches by fetching the first page of the corresponding triple pattern fragment. Then, `TriplePatternIterator`s are recursively chained together in the optimal order, which is chosen dynamically based on the number of matches for each binding. More specific iterators enable other SPARQL features.


Linked Data Fragments client

client.linkeddatafragments.org/#startFragment=http%3A%2F%2Fdata.linkeddatafragments.org%2Fdbpedia&query=CONSTRUCT%20%7B%20%3Fperson%2…

Linked Data Fragments client

Choose a start fragment: DBpedia ?s ?p ?o basic LDF (through HDT)

Enter a SPARQL query:

```
CONSTRUCT { ?person rdfs:label ?name; dbpedia-owl:birthPlace ?city. }
WHERE {
  ?person a dbpedia-owl:Artist; rdfs:label ?name;
            dbpedia-owl:birthPlace [ rdfs:label ?city;
                                     dbpedia-owl:country [ rdfs:label "Italy"@en ]].
}
```

Query results:

```
dbpedia:Stefano_Raffaele rdfs:label "Stefano Raffaele"@en;
    dbpedia-owl:birthPlace "Milan"@en.
dbpedia:Pompeo_Coppini rdfs:label "Pompeo Coppini"@en;
    dbpedia-owl:birthPlace "Moglia"@en.
dbpedia:Salvatore_Quasimodo rdfs:label "Salvatore Quasimodo"@en;
```

Fig. 1. The demo shows how Linked Data Fragments clients, such as Web browsers, evaluate SPARQL queries over datasets offered as inexpensive triple pattern fragments. In the above example, a user searches for artists born in Italian cities.

This iterator-based approach has been implemented as a JavaScript application (Fig. 1), to allow its usage on different platforms (standalone, library, browser application). The source code of the client, and also of triple pattern fragment servers, is freely available at `https://github.com/LinkedDataFragments/`. The versatility and efficiency of client-side querying is demonstrated through the Web application `http://client.linkeddatafragments.org`, which allows users to execute arbitrary SPARQL queries over triple pattern fragments. That way, participants experience first-hand how low-cost Linked Data publishing solutions can still enable efficient, realtime query execution over datasets on the Web.

References

1. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQL Web-querying infrastructure: Ready for action? In: Proceedings of the 12th International Semantic Web Conference (Nov 2013)
2. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Journal of Web Semantics 19, 22–41 (Mar 2013)
3. Harris, S., Seaborne, A.: SPARQL 1.1 query language. Recommendation, W3C (Mar 2013), `http://www.w3.org/TR/sparql11-query/`
4. Hartig, O.: An overview on execution strategies for Linked Data queries. Datenbank-Spektrum 13(2), 89–99 (2013)
5. Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0. Working draft, W3C (Mar 2014), `http://www.w3.org/TR/2014/WD-ldp-20140311/`
6. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the Web with high availability. In: Proceedings of the 13th International Semantic Web Conference (Oct 2014)