Web Semantics: Science, Services and Agents on the World Wide Web 37–38 (2016) 184–206

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Triple Pattern Fragments: A low-cost knowledge graph interface for the Web

Ruben Verborgh [a], Miel Vander Sande [a], Olaf Hartig [b], Joachim Van Herwegen [a], Laurens De Vocht [a], Ben De Meester [a], Gerald Haesendonck [a], Pieter Colpaert [a]

[a] Ghent University – iMinds, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium

[b] Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany

ARTICLE INFO

Article history:
Received 31 March 2015
Received in revised form
31 January 2016
Accepted 6 March 2016
Available online 21 March 2016

Keywords:
Linked Data
Linked Data Fragments
Querying
SPARQL

ABSTRACT

Billions of Linked Data triples exist in thousands of RDF knowledge graphs on the Web, but few of those graphs can be queried live from Web applications. Only a limited number of knowledge graphs are available in a queryable interface, and existing interfaces can be expensive to host at high availability. To mitigate this shortage of live queryable Linked Data, we designed a low-cost Triple Pattern Fragments interface for servers, and a client-side algorithm that evaluates SPARQL queries against this interface. This article describes the Linked Data Fragments framework to analyze Web interfaces to Linked Data and uses this framework as a basis to define Triple Pattern Fragments. We describe client-side querying for single knowledge graphs and federations thereof. Our evaluation verifies that this technique reduces server load and increases caching effectiveness, which leads to lower costs to maintain high server availability. These benefits come at the expense of increased bandwidth and slower, but more stable query execution times. These results substantiate the claim that lightweight interfaces can lower the cost for knowledge publishers compared to more expressive endpoints, while enabling applications to query the publishers' data with the necessary reliability.

1. Introduction

Before the Linked Data initiative [1], the Semantic Web suffered from a *chicken-and-egg* situation: there were no applications because there was no data, and there was no data because no applications were using it. Fortunately, Tim Berners-Lee's credo "Raw data now" has caught on, and now more knowledge graphs exist as Linked Data than ever before [2]. Statistics from the LODstats project [3] indicate that, as of March 2015, there are over 88 billion Linked Data triples distributed over 9960 knowledge graphs.[1] Thus, the ball is now back in the Semantic Web's court: given this tremendous amount of data in various domains, we should be able to build the envisioned intelligent applications [4].

Unfortunately, the availability of live queryable knowledge graphs on the Web still appears to be low. With "live queryable", we mean Linked Data that can be queried without first downloading the entire knowledge graph. With "low availability", we mean the two-sided problem the Semantic Web is currently facing: (i) the majority of knowledge graphs is not published in queryable form [3] and (ii) knowledge graphs that *are* published in a public SPARQL endpoint suffer from frequent downtime [5]. This unavailability becomes all the more problematic if we consider queries over multiple, *distributed* knowledge graphs. It is therefore understandable that many publishers choose the safe option, avoiding the responsibility of hosting a SPARQL endpoint by offering a data dump instead. Yet, this does not bring us closer to the Semantic *Web* because such data dumps need to be downloaded and stored locally so that the actual querying can happen *offline*. Furthermore, their consumption is only possible on sufficiently powerful machines – not on mobile devices, whose popularity continues to increase – and requires a technical background to set up. A significant amount of Linked Data knowledge graphs is therefore not reliably queryable, nor easily accessible, on the Web.

If we want Semantic Web applications on top of live knowledge graphs to become a reality, we must reconsider our options regarding Web-scale publication of Linked Data. Between the two extremes of data dumps and SPARQL endpoints lies a whole spectrum of possible Web interfaces, which has remained largely unexplored. The challenge is to methodically analyze the benefits and drawbacks an interface brings for clients and servers. In

E-mail address: ruben.verborgh@ugent.be (R. Verborgh).

[1] http://stats.lod2.eu/.

http://dx.doi.org/10.1016/j.websem.2016.03.003
1570-8268/

particular, we aim for solutions with minimal server complexity (minimizing the *cost* for data publishers) while still enabling live querying (maximizing the *utility* for Semantic Web applications).

In this article, we present and extend our ongoing work on *Linked Data Fragments* [6], a framework to analyze Linked Data publication interfaces on the Web, and *Triple Pattern Fragments* [7], a low-cost interface to triples. Novel contributions include in particular:

- an extended formalization (Sections 4 and 5) that details the response format and its considerations (Section 5.3);
- a detailed discussion of Triple Pattern Fragments publication and their relationship to existing interfaces (Sections 4.3 and 5.4);
- an extension of the query execution algorithm toward other SPARQL constructs (Section 6.3) and toward a federation of interfaces (Section 6.4);
- additional experimental results that
  - measure queries on the real-world knowledge graph DBpedia, revealing that the type of queries has a stronger influence than knowledge graph size (Section 7.2);
  - assess the impact of different serialization formats, which reveals a limited gain for specialized binary formats, likely due to the small page size (Section 7.3);
  - extend the application from one knowledge graph to multiple knowledge graphs, where we find that our proposed solution performs well for certain types of queries with regard to precision, recall, and/or execution time, but less so for other types (Section 7.4).

The remainder of this paper is structured as follows. Section 2 derives the research questions and hypotheses underlying this work, based on quantifiable characteristics for Web APIs. In Section 3, we describe existing solutions and highlight their advantages and disadvantages, focusing especially on the potential for live query execution. Section 4 introduces the Linked Data Fragments conceptual framework, followed by the definition of the Triple Pattern Fragments interface in Section 5. Section 6 details a client-side query algorithm for basic graph patterns, and extends it toward both general SPARQL queries and federations of knowledge graphs. We present an experimental evaluation in Section 7. Finally, Section 8 concludes the article with lessons learned and starting points for further research.

2. Problem statement

For querying knowledge graphs on the Web, there exist interfaces with powerful query capabilities (e.g., SPARQL endpoints) and interfaces with low server-side CPU cost (e.g., data dumps). The task of query evaluation currently happens either fully on the server side, or fully on the client side. However, there is a lack of options that balance these and other trade-offs in different ways. In this article, we aim to define and analyze an interface that distributes the load of query evaluation between a client and a server. To this end, we first define different characteristics relevant to Web Application Programming Interfaces (Web APIs) in Section 2.1. Using these characteristics, we then formulate a research question and associated hypotheses in Section 2.2.

2.1. Characteristics for Web APIs

A crucial architectural choice is the definition of the interface, which in turn reflects on characteristics such as *performance*, *cost*, *cache reuse*, and *bandwidth*. Each of these can be considered from the perspective of either servers in general, or clients performing a specific task [8].

Performance Performance is the rate at which tasks can be completed. For the server, performance can be measured as the number of requests it can handle per time interval, i.e., the inverse of the average request processing time. However, we need to take into account that one API might offer more granular requests than another. Therefore, to solve the same task, a client might require a different number of requests on different APIs.

Cost Cost is the amount of resources a request consumes. For the server, the resources typically involve CPU, RAM, and IO consumption. From the client perspective, the cost consists of processing one or multiple server responses into the desired result for a given task.

Cache reuse Cache reuse is the ratio of items that are requested multiple times from the cache versus the number of items it stores. The server might offer responses that can be reused by multiple clients, which can then subsequently be served from a cache instead, saving on server cost.

Bandwidth The bandwidth for the client is the product of the number of retrieved responses and the average response size (ignoring the relatively small request size). This is the same for the server, except that a portion of the responses might be cached and thus involve cache bandwidth instead of server bandwidth.

Given a task T a client needs to complete, two Web APIs I and I' might exhibit different behavior. For instance, a client might be able to complete T using a single large operation o with server-side cost c against I, whereas n smaller operations $o'_1, \ldots, o'_n$ with costs $c'_1, \ldots, c'_n$ might be needed in the case of I'. We assume here that the cost c'_i of each individual smaller operation o'_i is less than the cost of the large operation, c, but the total server cost for I' is $\sum_1^n c'_i$, which may be greater than c. If, however, some of these n smaller operations are cacheable, multiple clients executing tasks similar to T could reuse already generated responses from a cache, lowering the total number of requests – and thus the cost – for the server. Which of these factors dominates depends on the number of clients, the cost per request, the cache reuse ratio, the similarity of tasks, and other factors. In general, if costs increase to a certain level, a server might become fully occupied and unable to fulfill new incoming requests, and hence start a period of unavailability. Due to its impact on availability, it is thus important to examine how choosing a specific interface influences the cost for the server.

Lastly, we introduce an important practical characteristic. When designing an API, we need to consider the restrictions the interface places on clients. For this reason, we also assess the overhead for clients, which we express as follows:

Efficiency Efficiency for the client is the fraction of data retrieved from the server during the execution of a task over the amount of data required to execute that task.

2.2. Balancing trade-offs for publishing and querying Linked Data on the Web

Our goal is to enable reliable applications on top of knowledge graphs on the Web. This requires Linked Data that is (i) available for a high percentage of time (ii) in a queryable form. Given that the interface is the aspect of cost we have most control over, and that the interface directly determines queryability, we define our main research question as follows.