Web Semantics: Science, Services and Agents on the World Wide Web 31 (2015) 1–26

Contents lists available at ScienceDirect

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

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

On the formulation of performant SPARQL queries

Antonis Loizou[a,*], Renzo Angles[b], Paul Groth[a]

[a] Department of Computer Science, VU University of Amsterdam, The Netherlands

[b] Department of Computer Science, Universidad de Talca, Chile

ARTICLE INFO

Article history:
Received 25 February 2014
Received in revised form
20 June 2014
Accepted 7 November 2014
Available online 15 November 2014

Keywords:
SPARQL
Heuristics
Optimisation
RDF store
Data integration
Biomedical data

ABSTRACT

The combination of the flexibility of RDF and the expressiveness of SPARQL provides a powerful mechanism to model, integrate and query data. However, these properties also mean that it is nontrivial to write performant SPARQL queries. Indeed, it is quite easy to create queries that tax even the most optimised triple stores. Currently, application developers have little concrete guidance on how to write "good" queries. The goal of this paper is to begin to bridge this gap. It describes 5 heuristics that can be applied to create optimised queries. The heuristics are informed by formal results in the literature on the semantics and complexity of evaluating SPARQL queries, which ensures that queries following these rules can be optimised effectively by an underlying RDF store. Moreover, we empirically verify the efficacy of the heuristics using a set of openly available datasets and corresponding SPARQL queries developed by a large pharmacology data integration project. The experimental results show improvements in performance across six state-of-the-art RDF stores.

1. Introduction

Since the release of the Resource Description Framework (RDF) as a W3C Recommendation in 1999 [1,2], the amount of data published in various RDF serialisations has been rapidly increasing. Sindice[1] currently indexes 15+ billion triples [3,4]. The Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch[2] provides a striking visualisation of the diversity of domains that this data covers. The query language SPARQL [5] and SPARQL 1.1 update [6] is the W3C Recommendation for querying RDF data.

The flexibility in terms of both data structures and vocabularies make RDF and Linked Open Data attractive from a data provider perspective, but poses significant challenges in formulating correct, complex and performant SPARQL queries [7].[3,4] Application developers need to be familiar with various data schemas, cardinalities, and query evaluation characteristics in order to write effective SPARQL queries [8].

The contribution of this paper is a set of heuristics that can be used to formulate complex, but performant SPARQL queries to be evaluated against a number of RDF datasets. The heuristics are grounded in our experience in developing the OpenPHACTS[5] Platform [9]—a platform to facilitate the integration of large pharmaceutical datasets. The efficiency of the SPARQL query templates obtained by applying these heuristics is evaluated on a number of widely used RDF stores and contrasted to that of baseline queries.

The rest of the paper is organised as follows. Section 2 gives the context and motivation for this work, while Section 3 introduces the SPARQL syntax and semantics. Section 4 presents the five heuristics. An empirical comparison of the performance of SPARQL queries optimised using the defined heuristics is provided in Section 5. Section 6 discusses the inherent difficulties in providing paginated RDF views and how these can be addressed through some of the heuristics defined in this paper. A brief overview of related work is provided in Section 7. Finally, we provide concluding remarks in Section 8.

2. Motivation and context

The work presented in this paper was carried out in the context of the OpenPHACTS project [10], a collaboration of research

* Corresponding author.
E-mail addresses: a.loizou@vu.nl (A. Loizou), rangles@utalca.cl (R. Angles), p.t.groth@vu.nl (P. Groth).

1 http://sindice.com.

2 http://lod-cloud.net/.

3 'YarcData: Tuning SPARQL Queries for Performance', see: http://yarcdata.com/blog/?p=312.

4 'YarcData: Dont use a hammer to screw in a nail: Alternatives to REGEX in SPARQL', see: http://yarcdata.com/blog/?p=811.

5 http://www.openphacts.org.

http://dx.doi.org/10.1016/j.websem.2014.11.003
1570-8268/© 2014 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-SA license (http://creativecommons.org/licenses/by-nc-sa/3.0/).

institutions and major pharmaceutical companies. The key goal of the project is to support a variety of common tasks in drug discovery through a technology platform that integrates pharmacological and other biomedical research data using Semantic Web technologies. In order to achieve this goal, the platform must tackle the problem of public domain data integration in the pharmacology space and provide efficient access to the resulting integrated data. The development of the OpenPHACTS platform is driven by a set of concrete research questions presented in [11]. Its architecture is described in [9].

In the context of OpenPHACTS, the decision was made to avoid pushing the burden of performant query formulation to developers, but instead to provide them with an API driven by parameterised SPARQL queries [12]. This in turn created a need to formulate a set of performant query templates that are instantiated through API requests. The information that should be returned by such queries is first defined by domain experts, and subsequently the API developers are tasked with formulating a performant query template to satisfy the requirements given. The work presented in this paper stems largely from our experiences in hand-crafting such query templates. The heuristics rely on established SPARQL algebra equivalences, but in some cases also require extensive data statistics. We argue that the large overhead associated with the latter is justified by the associated increase in query performance and the need to repeatedly evaluate queries obtained from the same template.

A large body of work has been carried out on defining formal semantics for RDF and SPARQL in order to analyse query complexity and provide upper and lower bounds for generic SPARQL constructs [13–21]. These approaches are mainly focused on exploiting the formal semantics of SPARQL in order to prove generic rewrite rules for SPARQL patterns that are used in order to evaluate equivalence or subsumption between (sets of) queries. While a more detailed overview of the various SPARQL formalisation and optimisation techniques is provided in the next section, we note here that while the findings of these studies are invaluable to better understand the complexity of evaluating SPARQL queries and provide solid foundations for designing RDF store query planners and optimisers, *the issue of query formulation is not addressed*.

In contrast, the work presented here provides a set of heuristics to be used in formulating performant SPARQL queries based on concrete application requirements and known dataset schemata. The goal is to identify patterns that can be used to formulate queries that can be effectively optimised by a wide range of RDF stores. To that end, we provide a comparison on the performance of six state-of-the-art RDF storage systems with respect to the various query formulation techniques in order to study their effectiveness and applicability. As the implementation details of each system, the indices that are available and the manner in which the indices are used vary greatly across the different systems, investigating why one may outperform another is considered outside the scope of this work. Instead, we report on the efficacy of the heuristics across the different systems, considering each RDF store as a black box.

In summary, the paper has four main contributions:

1. A mapping between formal results published in the literature and SPARQL syntax.
2. A set of heuristics through which performant SPARQL queries can be formulated based on application requirements.
3. Guidance for RDF store selection based on the formulated SPARQL queries.
4. A reference set of queries and openly available datasets.

3. Preliminaries

In this section, we introduce the SPARQL query language by following the syntax used in [14,13,22], which is better suited to do formal analysis than the syntax presented by the W3C specification. The terminology given here is adopted for the remainder of this paper.

3.1. RDF datasets

Assume there are pairwise disjoint infinite sets $\mathbf{I}$ (IRIs), $\mathbf{B}$ (Blank nodes), and $\mathbf{L}$ (Literals). An *RDF term* is an element in the set $\mathbf{T} = \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. Given any structure α, we denote by $\mathrm{iri}(\alpha)$, $\mathrm{blank}(\alpha)$, $\mathrm{literal}(\alpha)$ and $\mathrm{term}(\alpha)$ the set of IRIs, blank nodes, literals and terms occurring in α respectively.

A tuple $(v_1, v_2, v_3) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{T}$ is called an *RDF triple*, where v_1 is the *subject*, v_2 the *predicate*, and v_3 the *object*. An *RDF Graph* (just graph from now on) is a set of RDF triples. The *union* of graphs, $G_1 \cup G_2$, is the set theoretical union of their sets of triples.

An *RDF dataset* D is a set $\{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$ where each G_i is a graph and each u_j is an IRI. G_0 is called the *default graph* of D. Each pair $\langle u_i, G_i\rangle$ is called a *named graph*; define $\mathrm{name}(G_i)_D = u_i$ and $\mathrm{graph}(u_i)_D = G_i$. The set of IRIs $\{u_1, \ldots, u_n\}$ is denoted $\mathrm{names}(D)$. Every dataset satisfies that: (i) it always contains one default graph (which could be empty); (ii) there may be no named graphs; (iii) each $u_j \in \mathrm{names}(D)$ is distinct; and (iv) $\mathrm{blank}(G_i) \cap \mathrm{blank}(G_j) = \emptyset$ for $i \neq j$. Finally, the *active graph* of D is the graph G_i used for querying D.

3.2. The SPARQL query language

SPARQL syntax

Assume the existence of an infinite set $\mathbf{V}$ of variables disjoint from $\mathbf{T}$. We denote by $\mathrm{var}(\alpha)$ the set of variables occurring in the structure α.

A *SPARQL select query*[6] (or just *query* from now on) is a tuple $Q = (W, F, P)$ where W is a set of variables (the symbol $*$ can be used to express "all variables"), F is a set-possibly empty-of dataset clauses, and P is a graph pattern. Next we define each component.

A *dataset clause* is either an expression FROM u or FROM NAMED u where $u \in \mathbf{I}$. The set of dataset clauses is used to define the RDF dataset used by the query.

A *graph pattern* is defined recursively as follows:

- The expression () is a graph pattern called the empty graph pattern.
- A tuple from $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ is a graph pattern called a *triple pattern*.
- If P_1 and P_2 are graph patterns then (P_1 AND P_2), (P_1 UNION P_2), and (P_1 OPT P_2) are graph patterns.
- If P_1 is a graph pattern and $u \in \mathbf{I} \cup \mathbf{V}$ then (u GRAPH P_1) is a graph pattern.
- If P_1 is a graph pattern then (P_1 FILTER C) is a graph pattern, where C is a *filter constraint* which is defined recursively as follows: (i) If $?X, ?Y \in \mathbf{V}$ and $u \in \mathbf{I} \cup \mathbf{L}$, then $?X = u$ and $?X = ?Y$ are *atomic filter constraints*.[7] (ii) If C_1 and C_2 are filter constraints then $(\neg C_1)$, $(C_1 \wedge C_2)$ and $(C_1 \vee C_2)$ are filter constraints.

In this paper, we will assume that every query $Q = (W, F, P)$ satisfies the following conditions:

- If $?X \in \mathrm{var}(W)$ then $?X \in \mathrm{var}(P)$. (*Safe result condition.*)

[6] In this paper, we restrict our study to Select queries and we do not consider solution modifiers.

[7] For a complete list of atomic filter constraints see [6].