Some Thoughts on OWL-Empowered SPARQL Query Optimization

V. Papakonstantinou[1], G. Flouris[1], I. Fundulaki[1], and A. Gubichev[2]

[1] Institute of Computer Science-FORTH, Greece
[2] TU Munich, Germany

Abstract. The discovery of optimal or close to optimal query plans for SPARQL queries is a difficult and challenging problem for query optimisers of RDF engines. Despite the growing volume of work on optimising SPARQL query answering, using heuristics or data statistics (such as cardinality estimations) there is little effort on the use of OWL constructs for query optimisation. OWL axioms can be the basis for the development of *schema-aware* optimisation techniques that will allow significant improvements in the performance of RDF query engines when used in tandem with data statistics or other heuristics. The aim of this paper is to show the potential of this idea, by discussing a diverse set of cases that depict how schema information can assist SPARQL query optimisers.

1 Introduction

The Linked Data paradigm, which is now the prominent enabler for sharing huge volumes of data using Semantic Web technologies, has created novel challenges for non-relational data management technologies such as RDF and graph database systems. Semantics of Linked Data are expressed in terms of the RDF Schema Language (RDFS) and the OWL Web Ontology Language. RDFS and OWL vocabularies are used from nearly all data sources in the LOD cloud. Moreover, according to a recent study[3], 36.49% of LOD use various OWL fragments, so it becomes critical to optimize RDF engines by considering OWL features.

Commercial RDF engines implement RDFS and OWL rules by performing *forward* or *backward* reasoning. Regardless of the way of reasoning, they basically store RDF data in a large *triple table* and consequently the evaluation of SPARQL queries boils down to performing a query with a large number of costly *self-joins*. To evaluate such difficult SPARQL queries a number of prototypes have been proposed. Many of these approaches propose a mapping of regular schema-conforming part of the RDF dataset into a set of relational tables [1, 2, 4], and rely on the optimization techniques of the underlying DBMSs for query evaluation. Other approaches [6, 7, 9, 13] propose main-memory resident extensive indexes for RDF triples. In either case, information residing in OWL schemas is rarely taken into account as in [3, 5, 8, 11], so it is our belief that an *an OWL schema-aware SPARQL query optimizer* could complement

[3] `http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/`

those approaches since many datasets (especially in the LOD Cloud) come with such good quality schemas.

In this paper we discuss how schema information expressed in terms of OWL ontologies can be used to perform interesting, possibly complex, optimizations in order to improve SPARQL query execution plans, and, consequently, the performance of the RDF engines. Such optimizations can be employed in a complementary fashion to traditional ones to further improve query planners' performance. Our intention in this work *is not* to provide full solutions, but to present the *potential* of the idea (fully described in [10]) by discussing some possible types of optimizations (Section 2) that can be performed. Many more may exist.

2 Schema based Optimization Techniques

2.1 Constraint Violation

An RDF engine could be able at compile time to take advantage of class and property constraints as expressed in an OWL schema; these include *equivalence* (`owl:equivalentClass`, `owl:equivalentProperty`) and *disjointness* (`owl:disjointWith`, `owl:propertyDisjointWith`) of classes and properties as well as constraints relevant to the property's *domain* and *range* (`rdfs:domain` and `rdfs:range` resp.). For instance, a query looking for an instance of two disjoint classes (`owl:disjointWith` construct) is certain to return no answers, so it should be answerable in constant time, without having the query engine evaluate it. This kind of information is important for RDF engines that follow either a *forward* or *backward* reasoning approach for computing the inferred knowledge.

2.2 Selectivity Estimation

Cardinality Constraints: OWL allows defining cardinality restrictions through the *min* (`owl:minCardinality`), *max* (`owl:maxCardinality`) and *exact* (`owl:cardinality`) cardinality constraints for object and datatype properties, which state how many instances of said property a resource can have. These schema-level constraints can be used to guide the optimizer into selecting a possibly efficient join ordering without resorting to statistics [3, 5]. To do so, triple patterns that refer to more selective properties (e.g. *functional* properties, `owl:FunctionalProperty` could be pushed down in the plan to reduce intermediate results).
Complex Class Expressions: Selectivity of triple patterns in a SPARQL query can be estimated through OWL constructs that define classes through *set operations*, such as *intersection* (`owl:intersectionOf`) and *union* (`owl:unionOf`). For example, consider a query that requests instances ?`x` of a class <`C`>, the latter defined as an intersection of classes <`C1`> and <`C2`>, in conjunction with triple patterns that relate instances ?`y` and ?`z` of the intersected classes, with triple patterns with predicates <`P1`> and <`P2`>. The class <`C`>, being more selective, should be considered first in a bushy plan with two sub-trees (around ?`x` and ?`y`, respectively) being joined with a hash join. Without the knowledge of schema constraints, the query optimizer would put the three triple patterns with `rdf:type` predicate at the end, since those usually match a lot of triples [12]. An analogous line of thought can be followed for the `owl:unionOf` construct.

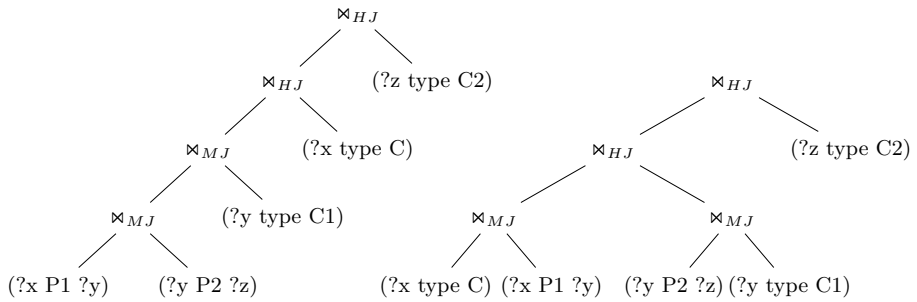

Fig. 1: Optimal plan, considers $C = C1$ `owl:intersectionOf` $C2$ (right) and suboptimal ignores the rule (left)

Class and Property Hierarchies: Hierarchies of classes and properties (through `rdfs:subClassOf` and `rdfs:subPropertyOf`) can also improve selectivity estimation. In this case, the triple patterns that request for instances of classes found lower in a class hierarchy should be considered first in a query plan (depending on the form of the query), when deciding join ordering.

2.3 Advanced Optimizations

In this section we present a set of cases where schema information can help the query engine determine the optimal plan in a more sophisticated way.

Inference: In backward reasoning systems, the inferred knowledge obtained through OWL reasoning rules is computed at query time. Is some cases, the same information may be obtained in various ways. For example, assume that we have a long hierarchy where <$\texttt{B}_\texttt{i}$> is a subclass (`rdfs:subClassOf`) of <$\texttt{B}_{\texttt{i}+1}$>, $i = 1, \ldots, n$. Consider also that the domain (`rdfs:domain`) of property <P> is class <A> and *all its values* (`owl:allValuesFrom`) come from root class <$\texttt{B}_{\texttt{n}+1}$>. In a query that asks for instances ?v of class <$\texttt{B}_{\texttt{n}+1}$> that are also values of property <P>, there are two ways to obtain instances ?v: one through the `owl:allValuesFrom` (*cls-avf* axiom[4]), and another through the transitivity of `rdfs:subClassOf` (*cax-sco* axiom[4]). For large n, class <$\texttt{B}_{\texttt{n}+1}$> is positioned high in the hierarchy, so the engine should use the `owl:allValuesFrom` construct to obtain the values for ?v. The alternative may be better if the two classes are sufficiently "close" in the hierarchy, especially given the fact that subsumption-related inference is the most optimized type (due to its widespread use).

Star Query Transformation: Schema information can also be used by the query optimizer to rewrite SPARQL queries to equivalent ones that have a form for which already known optimization techniques are easily applicable. For example, when a triple pattern, involving a *symmetric property* (`owl:SymmetricProperty`), "breaks" a star-shaped query pattern (subject values of remaining triple

[4] `https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules`

patterns appear as an object value), a schema-aware optimizer, should rewrite this query into its equivalent one, where all triple pattern's subject values are the same, according to the semantics of `owl:SymmetricProperty`.

3 Conclusions

We advocated on the use of OWL schema information for improving SPARQL query planning, and described some optimizations that can be employed in this direction. Our proposal is meant to be complementary to well-known optimizations (e.g., based on statistics) for query planning, and is most appropriate for datasets and benchmarks that use a rich schema structure (e.g., UOBM). In the future, we plan to work further on understanding the different possible optimizations and potential trade-offs, so that they can be implemented on top of an RDF store in order to quantify the achieved speed-up.

Acknowledgements. This work was partially funded by the EU projects LDBC (FP7 GA No. 317548) and HOBBIT (H2020 GA No. 688227).

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. VLDBJ 18(2) (2009)
2. Bornea, M.A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., Bhattacharjee, B.: Building an efficient rdf store over a relational database. In: SIGMOD. pp. 121–132. ACM (2013)
3. Bursztyn, D., Goasdou, F., Manolescu, I.: Optimizing Reformulation-based Query Answering in RDF. In: EDBT (2015)
4. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An efficient SQL-based RDF querying scheme. In: VLDB (2005)
5. Damian Bursztyn, Franois Goasdou, I.M.: Efficient Query Answering in DL-Lite through FOL Reformulation . In: DL (2015)
6. Erling, O., Mikhailov, I.: RDF support in the virtuoso DBMS. In: Networked Knowledge-Networked Media (2009)
7. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: A federated repository for querying graph structured data from the Web. In: ISWC (2007)
8. Kollia, I., Glimm, B.: Optimizing sparql query answering over owl ontologies. In: JAIR (2013)
9. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDBJ 19(1), 91–113 (2010)
10. Papakonstantinou, V., Fundulaki, I., Flouris, G., Alexiev, V.: Benchmark Design for Reasoning. Tech. Rep. D4.4.2, LDBC Council (2014)
11. Rodriguez-Muro, M., Rezk, M.: Web Semantics: Science, Services and Agents on the World Wide Web, chap. Efficient SPARQL-to-SQL with R2RML mappings. Elsevier (2015)
12. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.: Heuristics-based query optimisation for SPARQL. In: EDBT (2012)
13. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for Semantic Web data management. PVLDB 1(1) (2008)