Towards Equivalences for Federated SPARQL Queries

Carlos Buil-Aranda[1]* and Axel Polleres[2]**

[1] Department of Computer Science, Pontificia Universidad Católica, Chile
cbuil@ing.puc.cl
[2] Vienna University of Economy and Business (WU)
{*axel.polleres*}@wu.ac.at

Abstract The most common way for exposing RDF data on the Web is by means of SPARQL endpoints. These endpoints are Web services that implement the SPARQL protocol and then allow end users and applications to query just the RDF data they want. However the servers hosting the SPARQL endpoints restrict the access to the data by limiting the amount of results returned by user queries or the amount of queries per time and client that can be issued. For addressing this problem we analysed different alternatives that shall allow to obtain complete query result sets from the SPARQL endpoints by rewriting the original query using SPARQL1.1's federated query extension. We show that some of the commonly used SPARQL query patterns for this task provide unsound results while other patterns are more suitable for this task. We provide equivalent query patterns that help users in obtaining complete result sets circumventing the limitations imposed by servers.

1 Introduction

The Linked Open Data initiative promotes the publication and linkage of RDF data on the Web. Under this initiative many organisations (either public or private) exposed their data using the RDF data model and also provided links to other RDF datasets. Currently there are about 450 RDF datasets [2] available on the Web exposing billions of RDF statements. The most common way for accessing these RDF data is by means of SPARQL endpoints. These endpoints are Web services that implement the SPARQL protocol and then allow end users and applications to query just the RDF data they want. A user can send an SPARQL query to this Web service by accessing the endpoint's Web interface and then obtain back the results to the SPARQL query. However the servers hosting the SPARQL endpoints restrict the access to the data by limiting the server resources used in each received query. This physical resources limitation most commonly results in a limitation of the size of result set returned to the end users (normally a 10.000 result limit for each query) but it can also generate errors in the same query execution (normally time outs). These imposed limitations are necessary for avoiding the server to use too many resources to answer complex SPARQL queries.

However, this result set size limitation prevents users from obtaining complete answers to their SPARQL queries. The result set size limitation is particularly relevant

* Supported by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004 and CONICYT/FONDECYT project 3130617.
** Supported by the Vienna Science and Technology Fund (WWTF) project ICT12-015.

when a user wants to federate SPARQL queries to a number of SPARQL endpoints. Imagine a query to DBpedia asking for authors born in Madrid for next querying the Spanish National Library endpoint for more data about these authors. The first query to DBpedia would return more than the DBpedia's result set size limit resulting in an incomplete result set, failing next to join with the other endpoint query results.

Using different combinations of SPARQL operators it is possible to overcome the server's result size limitation and obtain complete result sets. The most common operator that can be used for that purpose is the VALUES operator from the new SPARQL 1.1 Recommendation [3]. This operator performs a join using the results from a query to the default local graph with the remote query results. In this way the local results are submitted along the remote query and joined remotely. However this operator is still not widely deployed [2] and other options have to be considered. This paper presents a study of several combinations of SPARQL operators and query patterns that avoid the server's result set limitation, allowing users to obtain sound and complete answers to their queries. We show that using the UNION operator for constraining remote queries and next doing the union of their results may return unsound results. We also present a subset of UNION SPARQL queries that return sound and complete results. As summary, using the SPARQL query pattern combinations we present in this paper it is possible to avoid the remote server result size limitation obtaining sound and complete results for our SPARQL queries. Still, an evaluation should be conducted for knowing what characteristics of each server may affect these equivalent SPARQL query patterns.

2 Syntax and Semantics

We first describe the basics of the SPARQL syntax and semantics of SPARQL we use within this paper. For the complete syntax and semantics of SPARQL we refer the reader to the Appendix available at http://www.polleres.net/publications/buil-poll-2014AMW.pdf.

Syntax. The official syntax of SPARQL1.1 [3] considers operators `OPTIONAL`, `UNION`, `FILTER`, `SELECT` and concatenation via a point symbol (`.`), to construct graph pattern expressions. The syntax of the language also considers `{ }` to group patterns, as well as keywords (new in SPARQL 1.1) `SERVICE` to delegate parts of a query to remote endpoints, and `VALUES` to define sets of bindings of variable bindings. We use the approach proposed in [5] for defining the SPARQL 1.0 syntax operators and the syntax proposed in [1] for the SPARQL 1.1 VALUES and SERVICE operators. More precisely we use the following syntax for the VALUES operator:

– VALUES $\boldsymbol{W}\boldsymbol{A}$ is a graph pattern where $\boldsymbol{W} = [?X_1, \ldots, ?X_n]$ is a sequence of pairwise distinct variables, and

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1}, \ \ldots, \ a_{1,n} \\ a_{2,1}, \ \ldots, \ a_{2,n} \\ \vdots \\ a_{m,1}, \ldots, \ a_{m,n} \end{bmatrix}$$

is a matrix of values where $a_{i,j} \in (I \cup L \cup \{\text{UNBOUND}\})$.

For the exposition of this paper, we leave out more complex SPARQL patterns such as GRAPH patterns as well as SPARQL 1.1 [3] patterns including, aggregates, and property paths. We highlight that solution modifiers are important for paginating results from the SPARQL endpoints and thus obtaining complete results. We mark the analysis of these modifiers as future work. We use the notion FILTER expressions as defined in [5]. We also use unary predicates like *bound*, *isBlank*, and the binary equality predicate '$=$', which herein we consider as synonym to *sameTerm*$(\cdot,\cdot)$ from [3].

Let P be a graph pattern; in what follows, we use $var(P)$ to denote the set of variables occurring in P. In particular, if t is a triple pattern, then $var(t)$ denotes the set of variables occurring in the components of t. Similarly, for a built-in condition R, we use $var(R)$ to denote the set of variables occurring in R.

Semantics. We use the SPARQL 1.0 semantics and the terminology defined in [5] and for the SPARQL 1.1 operators we use the definitions in [1]. We use letters B, I, L for denoting the sets of blank nodes, IRIs and RDF literals. As usual, we use $dom(\mu)$ for denoting the variables bound within – i.e. the domain of – a SPARQL solution mapping μ. We remind the reader of the most relevant definitions in Fig. 2 ; the evaluation of a FILTER expression R wrt. solution mapping μ, also defined in [5], is also relevant here:

- R is *bound*$(?X)$ and $?X \in dom(\mu)$;
- R is *isBlank*$(?X)$, $?X \in dom(\mu)$ and $\mu(?X) \in B$;
- R is $?X = c$, $?X \in dom(\mu)$ and $sameTerm(\mu(?X), c)$;
- R is $?X = ?Y$, $?X \in dom(\mu)$, $?Y \in dom(\mu)$ and $sameTerm(\mu(?X), \mu(?Y))$;
- R is $(R_1 \vee R_2)$, and $\mu \models R_1$ or $\mu \models R_2$;
- R is $(R_1 \wedge R_2)$, $\mu \models R_1$ and $\mu \models R_2$.

Here, B denotes the set of blank nodes, I the set of IRIs' and L the set of literals.

(1) If P is (P_1 FILTER R), then $[\![P]\!]_G = \{\mu \in [\![P_1]\!]_G \mid \mu \models R\}$.
(2) If $P =$ VALUES $\boldsymbol{W}$ $\boldsymbol{A}$ then

$$[\![P]\!]_G = \{\mu_j | 1 \leq j \leq m, dom(\mu_j) = \{?X_i \in \boldsymbol{W} \mid a_{i,j} \neq \text{UNBOUND}\}, \mu_j(?X_i) = a_{i,j}\}$$

Figure 1. Definition of $[\![P]\!]_G$ for a graph pattern P using FILTER and VALUES operators.

3 SPARQL endpoint query completeness alternatives

In this section we outline potential alternatives for obtaining complete results from queries to a remote SPARQL endpoints using SERVICE patterns. These alternatives are the use of VALUES , as well as combinations of UNION and FILTERs.

3.1 Naive approaches for replacing operators in federated queries

The VALUES operator can be used for restricting a remote SPARQL query that is sent using the SERVICE operator as illustrated in the following example: Let

$P = P_1$ AND (SERVICE c P_2) be a SPARQL pattern where $P_1 = (?X, a, ?Y)$ and $P_2 = (?Y, d, ?Z)$. If we pre-evaluate the solution bindings for P_1, written $[\![P_1]\!]_G$, the SERVICE operator could then be equivalently evaluated by sending pattern

$$P_2^{\text{VALUES}_{P_1}} = P_2 \,\text{VALUES}\, \boldsymbol{W}\boldsymbol{A}$$

where $\boldsymbol{W} = [var(P_1) \cap var(P_2)]$ and $\boldsymbol{A} = [\![P_1]\!]_G$ to endpoint c, i.e. if G_c is the default graph of service c

$$[\![P]\!]_G = [\![P1]\!]_G \bowtie [\![P_2^{\text{VALUES}_{P_1}}]\!]_{G_c}$$

However, the problem with this approach is that the VALUES operator is not yet widely deployed in existing endpoints [2] and other operators have to be used in order to simulate the desired behaviour. One alternative is the use of the UNION operator. In the previous SPARQL pattern P a commonly used alternative is to execute first pattern P_1 and use its results to create a large UNION query that will be submitted to the SERVICE address. In this way the results from one query are "injected" into the remote SPARQL endpoint in each branch of the UNION . I.e., let P as above, the idea would be to first evaluate P_1 and then for $[\![P_1]\!]_G = \{\mu_1, \ldots, \mu_n\}$ replace P_2 by

$$P_2^{\text{UNION}_{P_1}} = \{(\mu_1(P_2))\,\text{UNION}\ldots\text{UNION}(\mu_n(P_2))\}$$

Here, when writing $\mu(P)$ we mean the pattern obtained from P by replacing all $v \in dom(\mu) \cap vars(P)$ with $\mu(v)$.

Unfortunately, this method fails in relatively simple queries. Assume that P_1 = (?X, c, d), $c = I$ and P_2 = ((?Y, ?Z, ?T) UNION (?X, ?Y, b)) FILTER (?X = ?Y). With the local default graph $G_1 = \{(a, c, d)\}$ and the remote service' default graph $G_2 = \{(a, a, b), (e, c, d)\}$, we obtain: $[\![P_1]\!]_{G_1} = \{[?X \to a]\}$ whereas $[\![P_2]\!]_{G_2} = \{[?X \to a, ?Y \to a]\}$. However, if we proceed as suggested above, then $P_2^{\text{UNION}_{P_1}} = ((?Y, ?Z, ?T)$ UNION $(a, ?Y, b))$ FILTER $(a = ?Y)$ which yields an additional solution $[?Y \to a, ?Z \to a, ?T \to b]$ that was not admissible in the original P_2 but is also compatible with $\{[?X \to a]\}$. If we thus use this method in a federation scenario, we would obtain inconsistent results between the two queries if we use the result set from the first execution into the remote query execution.

Another problem are blank nodes. Assume for instance $P_1 = P_2$ =(?X, c, d) with $G_1 = \{(\_ : \texttt{b}, c, d)\}$ and $G_2 = \{(a, c, d)\}$. Here, as replacement would yield $P_2^{\text{UNION}_{P_1}} = (\_ : \texttt{b}, c, d)$ and since SPARQL engines treat blank nodes in patterns as variables, again a non-admissible solution would arise.

As a second alternative to UNIONs, one may consider is using FILTERs to inject the results of P_1 into P_2, namely,

$$P_2^{\text{FILTER}_{P_1}} = \{P_2 \text{ FILTER } \bigvee_{\mu \in [\![P_1]\!]_G} (\bigwedge_{v \in dom(\mu) \cap vars(P_2)} v = \mu(v))\}$$

This version does not run into the same issues as the before-mentioned solution using UNIONs. However, there are still problems here, as shown in the following example: assume now P_1 = (?X, b, c), $c = I$ and P_2 = ((?Y,d,e) UNION (?X,d,e)). With the local default graph $G_1 = \{(a, b, c)\}$ and the remote service' default graph $G_2 = \{(a, d, e)\}$, we obtain: $[\![P_1]\!]_{G_1} = \{[?X \to a]\}$ and $[\![P_2]\!]_{G_2} = \{[?X \to a], [?Y \to a]\}$; here, the second solution for $[\![P_2]\!]_{G_2}$, i.e. $\mu_2 = [?Y \to a]$ is compatible with the single solution for $[\![P_1]\!]_{G_1}$, i.e., $\mu_1 = [?Y \to a]$ yielding overall $\mu = [?X \to a, ?Y \to a]$. However, if we apply the method including FILTERs, then $P_2^{\mathrm{FILTER}_{P_1}}$ =\{ \{(?Y,d,e) UNION (?X,d,e)\} FILTER(?X = a)\} would not yield μ as a solution.

So, while the first alternatives may yield in inconsistent results the version using FILTERs seems to miss some results. In the next section we proceed to show refined versions of both alternatives, that solve these issues.

3.2 Two Equivalence theorems for SPARQL Federated Queries

As we have seen, some queries may return unexpected result mappings when substituting a variable for a specific value. This affects us when we use naively the UNION operator for dividing the remote query in several subqueries. Thus, we have to find a restricted class of SPARQL remote queries for which we obtain correct results. It turns out that one class of queries which avoid the above-mentioned problems is the class of *Strongly Bound* queries [1]: this class imposes the restriction to a SPARQL pattern that a variable $?X \in P$ will always be bound to a value, independent of the underlying data (we refer the reader to the Appendix for or a more detailed explanation of the *S*trongly Bound notion). Indeed, the source of the problem in the queries before were the query results containing the empty result mapping. That empty result mapping combined with other operators generate result sets different to the original one (besides of being unexpected) since the empty result mapping is not *null rejecting*. Let be P SPARQL a pattern; we use the definition of *Strongly Bound* variables in a pattern, denoted $\mathrm{SB}(P)$ as per [1] to assure that the variables used in both the SERVICE part of the query and in the part of the query that will restrict the SERVICE pattern do not contain that empty mapping:

Lemma 1. *Given a SPARQL pattern P with $v \in SB(P)$, let $\mu_e = [v \to e]$ for an $e \in I \cup L$, then $[\![\mu_e(P)]\!]_G \bowtie \mu_e = \{\mu \in [\![P]\!]_G | v \in dom(\mu) \wedge \mu(v) = e\}$.*

Lemma 1 essentially states that replacing strongly bound variables with IRIs or literals in a pattern will not yield additional results for P. In order to address the issue of blank nodes we adapt the definition of $P_2^{\mathrm{UNION}_{P_1}}$ from above as follows:

$$P_2^{\mathrm{UNION}'_{P_1}} = \{(\mu_1^B(P_2))\,\mathrm{UNION}\ldots\mathrm{UNION}(\mu_n^B(P_2))\}$$

where $\mu^B(P_2) = \{\mu(P_2)\,\mathrm{FILTER}(\neg(\bigvee_{v \in dom(\mu) \cap vars(P_2)} isBlank(\mu(v))))\}$, i.e. the problematic solutions containing blank nodes are filtered out.

Indeed, we confirm that UNION replacement with this modification works for remote patterns with only strongly bound variables; plus, it turns out that strongly bound remote queries also are evaluated correctly using the FILTER approach:

Theorem 1. *Let $P = P_1$ AND (SERVICE $c\, P_2$) such that $(vars(P_2) \cap vars(P_1)) \subseteq \mathrm{SB}(P_2)$, i.e. all variables that participate in a join are strongly bound in the pattern appearing on the service side, and let G_c be the default graph of service c and let $P_2^{\mathrm{UNION}_{P_1}}$ and $P_2^{\mathrm{FILTER}_{P_1}}$ defined as above , then (i) $[\![P]\!]_G = [\![P1]\!]_G \bowtie [\![P_2^{\mathrm{UNION}'_{P_1}}]\!]_{G_c}$, and (ii) $[\![P]\!]_G = [\![P1]\!]_G \bowtie [\![P_2^{\mathrm{FILTER}_{P_1}}]\!]_{G_c}$*

We note that, in case the local graph G does not contain any blank nodes Theorem 1(i) would also hold using the original replacement $P_2^{\text{UNION}_{P_1}}$. Moreover, it turns out that we can generalize the result in Theorem 1(ii) to also work in the general case with potentially unbound variables in the service pattern. To this end, we redefine $[\![P_2^{\text{FILTER}_{P_1}}]\!]_{G_c}$ as follows:

$$P_2^{\text{FILTER}'_{P_1}} = \{P_2 \text{ FILTER} \bigvee_{\mu \in [\![P_1]\!]_G} (\bigwedge_{v \in dom(\mu) \cap vars(P_2)} (v = \mu(v) \vee \neg bound(v)))\}$$

The trick to only filter for variable bound within P_2 fixes the problem from the example above, as stated in the following theorem.

Theorem 2. *Let $P = P_1$ AND (SERVICE c P_2) and G_c be the default graph of service c then*

$$[\![P]\!]_G = [\![P1]\!]_G \bowtie [\![P_2^{\text{FILTER}'_{P_1}}]\!]_{G_c}$$

For proofs we refer to the Appendix.

4 Conclusions

In this paper we have aimed at illustrating the problem that querying remote SPARQL endpoints with SERVICE patterns is a non-trivial task due to the limitations the server hosting these endpoints impose. The most important restriction is the result set size limitation that prevents users from obtaining complete results to their queries. We have also shown some initial results in terms of defining equivalent SPARQL query patterns involving the SERVICE operator that may help remedy this problem in practice. We believe that such equivalences are highly relevant for practical use cases involving queries on Linked data on existing SPARQL endpoints and deserve further investigation: Our next goal is to practically evaluate these and more equivalences for potentially restricted SERVICE queries on real endpoints to arrive at strategies for obtaining complete results on federated SPARQL queries over real endpoints.

References

1. C. Buil-Aranda, M. Arenas, O. Corcho, A. Polleres. Federating Queries in SPARQL 1.1: Syntax, Semantics and Evaluation. *J. Web Semantics*, 18(1), 2012.
2. C. Buil-Aranda, A. Hogan, J. Umbrich, P.-Y. Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In *ISWC2013*, LNCS 8219:277–293, 2013.
3. S. Harris, A. Seaborne. SPARQL 1.1 Query Language, W3C Rec., January 2012.
4. T. S. Jayram, P. G. Kolaitis, E. Vee. The containment problem for real conjunctive queries with inequalities. In PODS 2006, pages 80–89, 2006.
5. J. Pérez, M. Arenas, C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3), 2009.