Federating Queries in SPARQL1.1: Syntax, Semantics and Evaluation

Carlos Buil-Aranda[a,b], Marcelo Arenas[b], Oscar Corcho[a], Axel Polleres[c]

[a]*Ontology Engineering Group, Facultad de Informática, UPM, Spain*
[b]*Department of Computer Science, PUC Chile*
[c]*Siemens AG Österreich, Siemensstraße 90, 1210 Vienna, Austria*

Abstract

Given the sustained growth that we are experiencing in the number of SPARQL endpoints available, the need to be able to send federated SPARQL queries across these has also grown. To address this use case, the W3C SPARQL working group is defining a federation extension for SPARQL 1.1 which allows for combining graph patterns that can be evaluated over several endpoints within a single query. In this paper, we describe the syntax of that extension and formalize its semantics. Additionally, we describe how a query evaluation system can be implemented for that federation extension, describing some static optimization techniques and reusing a query engine used for data-intensive science, so as to deal with large amounts of intermediate and final results. Finally we carry out a series of experiments that show that our optimizations speed up the federated query evaluation process.

Recent years have witnessed a large and constant growth in the amount of RDF data available on the Web, exposed by means of Linked Data-enabled dereferenceable URIs in various formats (such as RDF/XML, Turtle, RDFa, etc.) and – of particular interest for the present paper – by SPARQL endpoints. Several non-exhaustive, and sometimes out-of-date or not continuously maintained, lists of SPARQL endpoints or data catalogs are available in different formats like CKAN[1], The Data Hub[2], the W3C wiki[3], etc. Most of these datasets are interlinked, as depicted graphically in the well-known Linked Open Data Cloud diagram[4], which allows navigating through them and facilitates building complex queries by combining data from different, sometimes heterogeneous and often physically distributed datasets.

SPARQL endpoints are RESTful services that accept queries over HTTP written in the SPARQL query language [1, 2] adhering to the SPARQL protocol [3], as defined by the respective W3C recommendation documents. However, the current SPARQL recommendation has an important limitation in terms of defining and executing queries that span across distributed datasets, since it hides the physical distribution of data across endpoints, and has normally been used for querying isolated endpoints. Hence users willing to federate queries across a number of SPARQL endpoints have been forced to create ad-hoc extensions of the query language and protocol, to include additional information about data sources in the configuration of their SPARQL endpoint servers [4, 5, 6] or to devise engineering solutions where data from remote endpoints is copied into the endpoint being queried. Given the need to address these types of queries, the SPARQL working group has proposed a query federation extension for the upcoming SPARQL 1.1 language [7] which is now under discussion in order to generate a new W3C recommendation in the coming months.[5]

☆This work has been performed in the context of the ADMIRE project (EU FP7 ICT-215024), and was supported by the Science Foundation Ireland project Lion-2 (Grant No. SFI/08/CE/I1380), as well as the Net2 project (FP7 Marie Curie IRSES 247601). We would like to thank, among many others, the OGSA-DAI team, specially to Ally Hume, for their advice in the development of the data workflows. Marc-Alexandre Nolin for his help with the bio2rdf queries and Jorge Pérez for his advice in theorem proving.

Email addresses: `cbuil@ing.puc.cl` (Carlos Buil-Aranda), `marenas@ing.puc.cl` (Marcelo Arenas), `ocorcho@fi.upm.es` (Oscar Corcho), `axel.polleres@siemens.com` (Axel Polleres)

[1]http://ckan.org/
[2]http://thedatahub.org/
[3]http://www.w3.org/wiki/SparqlEndpoints
[4]http://lod-cloud.net

[5]It is expected that SPARQL1.1 will be released in June 2012 for

The federated query extension of SPARQL 1.1 includes the new SERVICE operator which can also be used in conjunction with another new operator in the main SPARQL 1.1 query document: BINDINGS.

Firstly, the SERVICE operator allows for specifying, inside a SPARQL query, a SPARQL query endpoint to which a portion of the query will be delegated. This query endpoint may be known at the time of building the query, and hence the SERVICE operator will already specify the IRI of the SPARQL endpoint where it will be executed; or may be a variable that gets bound at query execution time after executing an initial SPARQL query fragment in one of the aforementioned RDF-enabled data catalogs, so that potential SPARQL endpoints that can answer the rest of the query can be obtained and used.

Secondly, the BINDINGS operator allows transferring results that are used to constrain a query, and which may come for instance from constraints specified in user interfaces that then transform these into SPARQL queries or – particularly, this may be used when implementing federated queries through scripting – from previous executions of other queries.

In this paper, we propose a syntax and a formalization of the semantics of these federation extensions of SPARQL 1.1 and define the constraints that have to be considered in their use in order to be able to provide pragmatic implementations of query evaluators. To this end, we define notions of service-boundedness and service-safeness, which ensure that the SERVICE operator can be safely evaluated.

We implement the static optimizations proposed in [8], using the notion of well-designed patterns, which prove to be effective in the optimization of queries that contain the OPTIONAL operator, the most costly operator in SPARQL [8, 9]. This also has important implications in the number of tuples being transferred and joined in federated queries, and hence our implementation benefits from this. Other works have analyzed adaptive query processing [10, 11] which optimize SPARQL queries by adapting them depending on the specific conditions of the query/execution environment.

As a result of our work, we have not only formalized the semantics of the SPARQL 1.1 federated query extension, but we have also implemented a system that supports these extensions and makes use of the discussed optimizations. This system, SPARQL-DQP (which stands for SPARQL Distributed Query Processing), is built on top of the OGSA-DAI and OGSA-DQP infrastructures [12, 13] that allow dealing with large amounts of data in distributed settings, supporting for example an indirect access mode that is normally used in the development of data-intensive workflows. In summary, the main contributions of this paper are:

- A formalization of the semantics of the federation extension of SPARQL 1.1, based on the current SPARQL semantics.
- A definition of service-boundedness and service-safeness conditions so as to ensure a pragmatic evaluation of these queries.
- A set of static optimizations for these queries, in the presence of OPTIONAL operators.
- An implementation suited to deal with large-scale RDF datasets distributed over federated query endpoints.

Organization of the paper. In Section 1, we describe the syntax and semantics of the SPARQL 1.1 federation extension. In Section 2, we introduce the notions of service-boundedness and service-safeness, which ensures that the SERVICE operator can be safely evaluated. In Section 3, we present some optimization techniques for the evaluation of the SPARQL 1.1 federated query extension. Finally, in Section 4 and 5, we present our implementation as well as an experimental evaluation of it.

1. Syntax and Semantics of SPARQL including the SPARQL 1.1 Federated Query

In this section, we give an algebraic formalization of SPARQL 1.1 including the SPARQL 1.1 Federated Query. We restrict ourselves to SPARQL over simple RDF, that is, we disregard higher entailment regimes (see [14]) such as RDFS or OWL. Our starting point is the existing formalization of SPARQL described in [8], to which we add the operators SERVICE proposed in [7] and BINDINGS proposed in [2].

We introduce first the necessary notions about RDF (taken mainly from [8]). Assume there are pairwise disjoint infinite sets I, B, and L (IRIs [15], Blank nodes, and Literals, respectively). Then a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*, where s is called the *subject*, p the *predicate* and o the *object*. An *RDF graph* is a set of RDF triples.

Moreover, assume the existence of an infinite set V of variables disjoint from the above sets, and leave UNBOUND to be a reserved symbol that does not belong to any of the previously mentioned sets.

1.1. Syntax

The official syntax of SPARQL [1] considers operators `OPTIONAL`, `UNION`, `FILTER`, `GRAPH`, `SELECT` and concatenation via a point symbol (`.`), to construct graph pattern expressions. Operators `SERVICE` is introduced in the SPARQL 1.1 Federated Query extension and `BINDINGS` is introduced in the main SPARQL 1.1 query document, the former for allowing users to direct a portion of a query to a particular SPARQL endpoint, and the latter for transferring results that are used to constrain a query. The syntax of the language also considers `{ }` to group patterns and some implicit rules of precedence and association. In order to avoid ambiguities in the parsing, we follow the approach proposed in [8], and we first present the syntax of SPARQL graph patterns in a more traditional algebraic formalism, using operators AND (`.`), UNION (`UNION`), OPT (`OPTIONAL`), FILTER (`FILTER`), GRAPH (`GRAPH`) and SERVICE (`SERVICE`), then we introduce the syntax of BINDINGS queries, which use the BINDINGS operator (`BINDINGS`), and we conclude by defining the syntax of SELECT queries, which use the SELECT operator (`SELECT`). More precisely, a SPARQL graph pattern expression is defined recursively as follows:

(1) A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (a triple pattern).
(2) If P_1 and P_2 are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$ are graph patterns.
(3) If P is a graph pattern and R is a *SPARQL built-in* condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern.
(4) If P is a graph pattern and $a \in (I \cup V)$, then $(\text{GRAPH } a\ P)$ is a graph pattern.
(5) If P is a graph pattern and $a \in (I \cup V)$, then $(\text{SERVICE } a\ P)$ is a graph pattern.

As we will see below, despite the similarity between the syntaxes of GRAPH and SERVICE operators, they behave semantically quite differently.

For the exposition of this paper, we leave out further more complex graph patterns from SPARQL 1.1 including aggregates, property paths, and subselects, but only mention one additional feature which is particularly relevant for federated queries, namely, BINDINGS queries. A SPARQL BINDINGS query is defined as follows:

(6) If P is a graph pattern, $\vec{W} \in V^n$ is a nonempty sequence of pairwise distinct variables of length $n > 0$ and $\{\vec{A_1}, \ldots, \vec{A_k}\}$ is a nonempty set of sequences $\vec{A_i} \in (I \cup L \cup \{\text{UNBOUND}\})^n$, then $(P \text{ BINDINGS } \vec{W}\ \{\vec{A_1}, \ldots, \vec{A_k}\})$ is a BINDINGS query.

Finally, a SPARQL SELECT query is defined as:

(7) If P is either a graph pattern or a BINDINGS query, and W is a set of variables, then $(\text{SELECT } W\ P)$ is a SELECT query.

It is important to notice that the rules (1)–(4) above were introduced in [8], while we formalize in the rules (5)–(7) the federation extension of SPARQL proposed in [7].

We used the notion of built-in conditions for the FILTER operator above. A SPARQL built-in condition is constructed using elements of the set $(I \cup L \cup V)$ and constants, logical connectives ($\neg$, $\wedge$, $\vee$), the binary equality predicate ($=$) as well as unary predicates like *bound*, *isBlank*, *isIRI*, and *isLiteral*.[6] That is: (1) if $?X, ?Y \in V$ and $c \in (I \cup L)$, then $bound(?X)$, $isBlank(?X)$, $isIRI(?X)$, $isLiteral(?X)$, $?X = c$ and $?X = ?Y$ are built-in conditions, and (2) if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions.

Let P be either a graph pattern or a BINDINGS query or a SELECT query. 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.

1.2. Semantics

To define the semantics of SPARQL queries, we need to introduce some extra terminology from [8]. A mapping μ from V to $(I \cup B \cup L)$ is a partial function $\mu : V \to (I \cup B \cup L)$. Abusing notation, for a triple pattern t, we denote by $\mu(t)$ the pattern obtained by replacing the variables in t according to μ. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. We sometimes write down concrete mappings in square brackets, for instance, $\mu = [?X \to a, ?Y \to b]$ is the mapping with $dom(\mu) = \{?X, ?Y\}$ such that, $\mu(?X) = a$ and $\mu(?Y) = b$. Two mappings μ_1 and μ_2 are compatible, denoted by

[6]For simplicity, we omit here other features such as comparison operators ('<', '>','≤','≥'), data type conversion and string functions, see [1, Section 11.3] for details. It should be noted that the results of the paper can be easily extended to the other built-in predicates in SPARQL

$\mu_1 \sim \mu_2$, when for all $?X \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping. Intuitively, μ_1 and μ_2 are compatible if μ_1 can be extended with μ_2 to obtain a new mapping, and vice versa [8]. We will use the symbol $\mu_\emptyset$ to represent the mapping with empty domain (which is compatible with any other mapping).

Let Ω_1 and Ω_2 be sets of mappings.[7] Then the join of, the union of, the difference between and the left outer-join between Ω_1 and Ω_2 are defined as follows [8]:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \\
&\qquad \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\}, \\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}, \\
\Omega_1 \smallsetminus \Omega_2 &= \{\mu \in \Omega_1 \mid \forall \mu' \in \Omega_2 : \mu \nsim \mu'\}, \\
\Omega_1 ⟕ \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2).
\end{aligned}$$

Next we use these operators to give semantics to graph pattern expressions, BINDINGS queries and SELECT queries. More specifically, we define this semantics in terms of an evaluation function $[\![\cdot]\!]_G^{DS}$, which takes as input any of these types of queries and returns a set of mappings, depending on the *active dataset DS* and the *active graph G* within *DS*.

Here, we use the notion of a dataset from SPARQL, i.e. a *dataset* $DS = \{(def, G), (g_1, G_1), \ldots (g_k, G_k)\}$, with $k \geq 0$ is a set of pairs of symbols and graphs associated with those symbols, where the default graph G is identified by the special symbol $def \notin I$ and the remaining so-called "named" graphs (G_i) are identified by IRIs ($g_i \in I$). Without loss of generality (there are other ways to define the dataset such as via explicit FROM and FROM NAMED clauses), we assume that any query is evaluated over a fixed dataset DS and that any SPARQL endpoint that is identified by an IRI $c \in I$ evaluates its queries against its own fixed dataset $DS_c = \{(def, G_c), (g_{c,1}, G_{c,1}), \ldots (g_{c,k_c}, G_{c,k_c})\}$. That is, we assume given a partial function ep from the set I of IRIs such that for every $c \in I$, if $ep(c)$ is defined, then $ep(c) = DS_c$ is the dataset associated with the endpoint accessible via IRI c. Moreover, we assume (i) a function $graph(g, DS)$ which – given a dataset $DS = \{(def, G), (g_1, G_1), \ldots (g_k, G_k)\}$ and a graph name $g \in \{def, g_1, \ldots g_k\}$ – returns the graph corresponding to symbol g within DS, and (ii) a function $names(DS)$ which given a dataset DS as before returns the set of names $\{g_1, \ldots g_k\}$.

[7]As in [8], for the exposition in this paper, we consider a set-based semantics, whereas the semantics of [1] considers duplicate solutions, i.e., multisets of mappings.

The evaluation of a graph pattern P over a dataset DS with active graph G, denoted by $[\![P]\!]_G^{DS}$, is defined recursively as shown in Figure 1. In this figure, the definition of the semantics of the FILTER operator is based on the definition of the notion of satisfaction of a built-in condition by a mapping. More precisely, given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if: [8]

- R is $bound(?X)$ and $?X \in dom(\mu)$;
- R is $isBlank(?X)$, $?X \in dom(\mu)$ and $\mu(?X) \in B$;
- R is $isIRI(?X)$, $?X \in dom(\mu)$ and $\mu(?X) \in I$;
- R is $isLiteral(?X)$, $?X \in dom(\mu)$ and $\mu(?X) \in L$;
- R is $?X = c$, $?X \in dom(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X \in dom(\mu)$, $?Y \in dom(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is $(\neg R_1)$, and it is not the case that $\mu \models R_1$;
- 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 $\mu \models R_2$.

Moreover, the semantics of BINDINGS queries is defined as follows. Given a sequence $\vec{W} = [?X_1, \ldots, ?X_n]$ of pairwise distinct variables, where $n \geq 1$, and a sequence $\vec{A} = [a_1, \ldots, a_n]$ of values from $(I \cup L \cup \{\text{UNBOUND}\})$, let $\mu_{\vec{W} \mapsto \vec{A}}$ be a mapping with domain $\{?X_i \mid i \in \{1, \ldots, n\}$ and $a_i \in (I \cup L)\}$ and such that $\mu_{\vec{W} \mapsto \vec{A}}(?X_i) = a_i$ for every $?X_i \in dom(\mu_{\vec{W} \mapsto \vec{A}})$. Then

(8) If $P = (P_1 \text{ BINDINGS } \vec{W}\ \{\vec{A}_1, \ldots, \vec{A}_k\})$ is a BINDINGS query:

$$[\![P]\!]_G^{DS} \quad = \quad [\![P_1]\!]_G^{DS} \bowtie \{\mu_{\vec{W} \mapsto \vec{A}_1}, \ldots, \mu_{\vec{W} \mapsto \vec{A}_k}\}.$$

Finally, the semantics of SELECT queries is defined as follows. Given a mapping $\mu : V \to (I \cup B \cup L)$ and a set of variables $W \subseteq V$, the restriction of μ to W, denoted by $\mu_{|_W}$, is a mapping such that $dom(\mu_{|_W}) = (dom(\mu) \cap W)$ and $\mu_{|_W}(?X) = \mu(?X)$ for every $?X \in (dom(\mu) \cap W)$. Then

(9) If $P = (\text{SELECT } W\ P_1)$ is a SELECT query, then:

$$[\![P]\!]_G^{DS} \quad = \quad \{\mu_{|_W} \mid \mu \in [\![P_1]\!]_G^{DS}\}.$$

[8]For the sake of presentation, we use here the two-valued semantics for built-in conditions from [8], instead of the three-valued semantics including errors used in [1]. It should be noticed that the results of the paper can be easily extended to this three-valued semantics.

(1) If P is a triple pattern t, then $[\![P]\!]^{DS}_G = \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in G\}$.
(2) If P is (P_1 AND P_2), then $[\![P]\!]^{DS}_G = [\![P_1]\!]^{DS}_G \bowtie [\![P_2]\!]^{DS}_G$.
(3) If P is (P_1 OPT P_2), then $[\![P]\!]^{DS}_G = [\![P_1]\!]^{DS}_G ⟕ [\![P_2]\!]^{DS}_G$.
(4) If P is (P_1 UNION P_2), then $[\![P]\!]^{DS}_G = [\![P_1]\!]^{DS}_G \cup [\![P_2]\!]^{DS}_G$.
(5) If P is (GRAPH c P_1) with $c \in I \cup V$, then

$$[\![P]\!]^{DS}_G = \begin{cases} [\![P_1]\!]^{DS}_{graph(c,DS)} & \text{if } c \in names(DS) \\ \{\mu_\emptyset\} & \text{if } c \in I \setminus names(DS) \\ \{\mu \cup \mu_c \mid \exists g \in names(DS):\ \mu_c = [c \to g],\ \mu \in [\![P_1]\!]^{DS}_{graph(g,DS)} \text{ and } \mu_c \sim \mu\} & \text{if } c \in V \end{cases}$$

(6) If P is (SERVICE c P_1) with $c \in I \cup V$, then

$$[\![P]\!]^{DS}_G = \begin{cases} [\![P_1]\!]^{ep(c)}_{graph(def,ep(c))} & \text{if } c \in dom(ep) \\ \{\mu_\emptyset\} & \text{if } c \in I \setminus dom(ep) \\ \{\mu \cup \mu_c \mid \exists s \in dom(ep):\ \mu_c = [c \to s],\ \mu \in [\![P_1]\!]^{ep(s)}_{graph(def,ep(s))} \text{ and } \mu_c \sim \mu\} & \text{if } c \in V \end{cases}$$

(7) If P is (P_1 FILTER R), then $[\![P]\!]^{DS}_G = \{\mu \in [\![P_1]\!]^{DS}_G \mid \mu \models R\}$.

Figure 1: Definition of $[\![P]\!]^{DS}_G$ for a graph pattern P.

It is important to notice that the rules (1)–(5) and (7) in Figure 1 and the previous rule (9) were introduced in [8], while we propose in the rules (6) and (8) a semantics for the operators SERVICE and BINDINGS introduced in [7]. Intuitively, if $c \in I$ is the IRI of a SPARQL endpoint, then the idea behind the definition of (SERVICE c P_1) is to evaluate query P_1 in the SPARQL endpoint specified by c. On the other hand, if $c \in I$ is not the IRI of a SPARQL endpoint, then (SERVICE c P_1) leaves all the variables in P_1 unbound, as this query cannot be evaluated in this case. This idea is formalized by making $\mu_\emptyset$ the only mapping in the evaluation of (SERVICE c P_1) if $c \notin dom(ep)$. In the same way, (SERVICE $?X$ P_1) is defined by considering that variable $?X$ is used to store IRIs of SPARQL endpoints. That is, (SERVICE $?X$ P_1) is defined by assigning to $?X$ all the values s in the domain of function ep (in this way, $?X$ is also used to store the IRIs from where the values of the variables in P_1 are coming from). Finally, the idea behind the definition of (P_1 BINDINGS $\vec{W}$ $\{\vec{A}_1, \ldots, \vec{A}_k\}$) is to constrain the values of the variables in $\vec{W}$ to the values specified in $\vec{A}_1$, …, $\vec{A}_k$.

The goal of the rules (6) and (8) is to define in an unambiguous way what the result of evaluating an expression containing the operators SERVICE and BINDINGS should be. As such, these rules should not be considered as a straightforward basis for an implementation of the language. In fact, a direct implementation of the rule (6), that defines the semantics of a pattern of the form (SERVICE $?X$ P_1), would involve evaluating a particular query in every possible SPARQL endpoint, which is obviously infeasible in practice. In the next section, we face this issue and, in particular, we introduce a syntactic condition on SPARQL queries that ensures that a pattern of the form (SERVICE $?X$ P_1) can be evaluated by only considering a finite set of SPARQL endpoints, whose IRIs are actually taken from the RDF graph where the query is being evaluated.

2. On Evaluating the SERVICE Operator

As we pointed out in the previous section, the evaluation of a pattern of the form (SERVICE $?X$ P) is infeasible unless the variable $?X$ is bound to a finite set of IRIs. This notion of *boundedness* is one of the most significant and unclear concepts in the SPARQL federation extension. In fact, since agreement on such a boundedness notion could not yet be found, the current version of the specification of this extension [7] does not specify a formalization of the semantics of queries of the form (SERVICE $?X$ P). Here, we provide a formalization of this concept, and we study the complexity issues associated with it.

2.1. The notion of boundedness

Assume that G is an RDF graph that uses triples of the form (a, service_address, b) to indicate that a SPARQL endpoint with name a is located at the IRI b. Moreover, let P be the following SPARQL query:

$$\Big(\text{SELECT } \{?X, ?N, ?E\} \quad \Big((?X, \text{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \text{email}, ?E))\Big)\Big).$$

Query P is used to compute the list of names and email addresses that can be retrieved from the SPARQL endpoints stored in an RDF graph. In fact, if $\mu \in [\![P]\!]^{DS}_{G}$, then $\mu(?X)$ is the name of a SPARQL endpoint stored in G, $\mu(?N)$ is the name of a person stored in that SPARQL endpoint and $\mu(?E)$ is the email address of that person. It is important to notice that there is a simple strategy that ensures that query P can be evaluated in practice: first compute $[\![(?X, \text{service\_address}, ?Y)]\!]^{DS}_{G}$, and then for every μ in this set, compute $[\![(\text{SERVICE}\ a\ (?N, \text{email}, ?E))]\!]^{DS}_{G}$ with $a = \mu(?Y)$. More generally, SPARQL pattern (SERVICE $?Y$ $(?N, \text{email}, ?E)$) can be evaluated over DS in this case as only a finite set of values from the domain of G need to be considered as the possible values of $?Y$. This idea naturally gives rise to the following notion of boundedness for the variables of a SPARQL query. In the definition of this notion, $dom(G)$ refers to the domain of a graph G, that is, the set of elements from $(I \cup B \cup L)$ that are mentioned in G; $dom(DS)$ refers to the union of the domains of all graphs in the dataset DS; and finally, $dom(P)$ refers to the set of elements from $(I \cup L)$ that are mentioned in P.

Definition 1 (Boundedness). *Let P be a SPARQL query and $?X \in var(P)$. Then $?X$ is bound in P if one of the following conditions holds:*

- *P is either a graph pattern or a BINDINGS query, and for every dataset DS, every RDF graph G in DS and every $\mu \in [\![P]\!]^{DS}_{G}$: $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$.*
- *P is a SELECT query* (SELECT W P_1) *and $?X$ is bound in P_1.*

In the evaluation of a graph pattern (GRAPH $?X$ P) over a dataset DS, variable $?X$ necessarily takes a value from $names(DS)$. Thus, the GRAPH operator makes such a variable $?X$ to be bound. Given that the values in $names(DS)$ are not necessarily mentioned in the dataset DS, the previous definition first imposes the condition that $?X \in dom(\mu)$, and then not only considers the case $\mu(?X) \in dom(DS)$ but also the case $\mu(?X) \in names(DS)$. In the same way, the BINDINGS operator can make a variable $?X$ in a query P to be bound by assigning to it a fixed set of values. Given that these values are not necessarily mentioned in the dataset DS where P is being evaluated, the previous definition also considers the case $\mu(?X) \in dom(P)$. As an example of the above definition, we note that variable $?Y$ is bound in the graph pattern

$$P_1 = ((?X, \text{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \text{email}, ?E))),$$

as for every dataset DS, every RDF graph G in DS and every mapping $\mu \in [\![P_1]\!]^{DS}_{G}$, we know that $?Y \in dom(\mu)$ and $\mu(?Y) \in dom(DS)$. Moreover, we also have that variable $?Y$ is bound in (SELECT $\{?X, ?N, ?E\}$ P_1) as $?Y$ is bound in graph pattern P_1.

A natural way to ensure that a SPARQL query P can be evaluated in practice is by imposing the restriction that for every sub-pattern (SERVICE $?X$ P_1) of P, it holds that $?X$ is bound in P. However, in the following theorem we show that such a condition is undecidable and, thus, a SPARQL query engine would not be able to check it in order to ensure that a query can be evaluated.

Theorem 1. *The problem of verifying, given a SPARQL query P and a variable $?X \in var(P)$, whether $?X$ is bound in P is undecidable.*

Proof: The satisfiability problem for relational algebra is the problem of verifying, giving a relational expression φ, whether there exists a (finite) database instance I such that the set of answers of φ over I is not empty. Given that this problem is undecidable [16], it is possible to prove from the results in [17] the following result about the complexity of the satisfiability problem for SPARQL. A graph pattern P is said to be satisfiable if there exists a dataset DS and RDF graph G in DS such that $[\![P]\!]^{DS}_{G} \neq \emptyset$.

Claim 1. *The problem of verifying, given a graph pattern P, whether P is satisfiable is undecidable.*

Next we show that the complement of the previous problem can be reduced to the problem of verifying, given a graph pattern P and a variable $?X \in var(P)$, whether $?X$ is bound in P, from which we conclude that the theorem holds. Let P be a graph pattern and $?X$, $?Y$, $?Z$ be variables that are not mentioned in P. Then define a graph pattern Q as:

$$Q = ((?X, ?Y, ?Z) \text{ UNION } P).$$

It is easy to see that variable $?X$ is bound in Q if and only if graph pattern P is not satisfiable, which was to be shown. □

The fact that the notion of boundedness is undecidable prevents one from using it as a restriction over the variables in SPARQL queries. To overcome this limitation, we introduce here a syntactic condition that ensures that a variable is bound in a pattern and that can be efficiently verified.

Definition 2 (Strong boundedness). *Let P be a SPARQL query. Then the set of strongly bound variables in P, denoted by $\mathrm{SB}(P)$, is recursively defined as follows:*

- *if $P = t$, where t is a triple pattern, then $\mathrm{SB}(P) = \mathit{var}(t)$;*
- *if $P = (P_1 \text{ AND } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \mathrm{SB}(P_2)$;*
- *if $P = (P_1 \text{ UNION } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cap \mathrm{SB}(P_2)$;*
- *if $P = (P_1 \text{ OPT } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$;*
- *if $P = (P_1 \text{ FILTER } R)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$;*
- *if $P = (\text{GRAPH } c\ P_1)$, with $c \in I \cup V$, then*

$$\mathrm{SB}(P) = \begin{cases} \emptyset & c \in I, \\ \mathrm{SB}(P_1) \cup \{c\} & c \in V; \end{cases}$$

- *if $P = (\text{SERVICE } c\ P_1)$, with $c \in I \cup V$, then $\mathrm{SB}(P) = \emptyset$;*
- *if $P = (P_1 \text{ BINDINGS } \vec{W}\ \{\vec{A_1}, \ldots, \vec{A_n}\})$, then*

$$\begin{aligned}\mathrm{SB}(P) = {} & \mathrm{SB}(P_1) \cup \\ & \{?X \mid ?X \textit{ is included in } \vec{W} \textit{ and for} \\ & \textit{every } i \in \{1, \ldots, n\} : ?X \in \mathit{dom}(\mu_{\vec{W} \mapsto \vec{A_i}})\};\end{aligned}$$

- *if $P = (\text{SELECT } W\ P_1)$, then $\mathrm{SB}(P) = (W \cap \mathrm{SB}(P_1))$.*

The previous definition recursively collects from a SPARQL query P a set of variables that are guaranteed to be bound in P. For example, if P is a triple pattern t, then $\mathrm{SB}(P) = \mathit{var}(t)$ as one knows that for every variable $?X \in \mathit{var}(t)$, every dataset DS and every RDF graph G in DS, if $\mu \in [\![t]\!]_G^{DS}$, then $?X \in \mathit{dom}(\mu)$ and $\mu(?X) \in \mathit{dom}(G)$ (which is a subset of $\mathit{dom}(DS)$). In the same way, if $P = (P_1 \text{ AND } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \mathrm{SB}(P_2)$ as one knows that if $?X$ is bound in P_1 or in P_2, then $?X$ is bound in P. As a final example, notice that if $P = (P_1 \text{ BINDINGS } \vec{W}\ \{\vec{A_1}, \ldots, \vec{A_n}\})$ and $?X$ is a variable mentioned in $\vec{W}$ such that $?X \in \mathit{dom}(\mu_{\vec{W} \mapsto \vec{A_i}})$ for every $i \in \{1, \ldots, n\}$, then $?X \in \mathrm{SB}(P)$. In this case, one knows that $?X$ is bound in P since $[\![P]\!]_G^{DS} = [\![P_1]\!]_G^{DS} \bowtie \{\mu_{\vec{W} \mapsto \vec{A_1}}, \ldots, \mu_{\vec{W} \mapsto \vec{A_n}}\}$ and $?X$ is in the domain of each one of the mappings $\mu_{\vec{W} \mapsto \vec{A_i}}$, which implies that $\mu(?X) \in \mathit{dom}(P)$ for every $\mu \in [\![P]\!]_G^{DS}$. In the following proposition, we formally show that our intuition about $\mathrm{SB}(P)$ is correct, in the sense that every variable in this set is bound in P (the proof of this proposition can be found in Appendix Appendix A).

Proposition 1. *For every SPARQL query P and variable $?X \in \mathit{var}(P)$, if $?X \in \mathrm{SB}(P)$, then $?X$ is bound in P.*

Given a SPARQL query P and a variable $?X \in \mathit{var}(P)$, it can be efficiently verified whether $?X$ is strongly bound in P. Thus, a natural and efficiently verifiable way to ensure that a SPARQL query P can be evaluated in practice is by imposing the restriction that for every sub-pattern $(\text{SERVICE } ?X\ P_1)$ of P, it holds that $?X$ is strongly bound in P. However, this notion still needs to be modified in order to be useful in practice, as shown by the following examples.

Example 1. *Assume first that P_1 is the following graph pattern:*

$$\begin{aligned} P_1 = \Big[& (?X, \mathit{service\_description}, ?Z) \text{ UNION} \\ & \Big((?X, \mathit{service\_address}, ?Y) \text{ AND} \\ & \qquad (\text{SERVICE } ?Y\ (?N, \mathit{email}, ?E)) \Big) \Big]. \end{aligned}$$

That is, either $?X$ and $?Z$ store the name of a SPARQL endpoint and a description of its functionalities, or $?X$ and $?Y$ store the name of a SPARQL endpoint and the IRI where it is located (together with a list of names and email addresses retrieved from that location). Variable $?Y$ is neither bound nor strongly bound in P_1. However, there is a simple strategy that ensures that P_1 can be evaluated over a dataset DS and an RDF graph G in DS: first compute $[\![(?X, \mathit{service\_description}, ?Z)]\!]_G^{DS}$, then compute $[\![(?X, \mathit{service\_address}, ?Y)]\!]_G^{DS}$, and finally for every μ in the set $[\![(?X, \mathit{service\_address}, ?Y)]\!]_G^{DS}$, compute $[\![(\text{SERVICE } a\ (?N, \mathit{email}, ?E))]\!]_G^{DS}$ with $a = \mu(?Y)$. In fact, the reason why P_1 can be evaluated in this case is that $?Y$ is bound (and strongly bound) in the sub-pattern $((?X, \mathit{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \mathit{email}, ?E)))$ of P_1.

As a second example, assume that DS is a dataset and G is an RDF graph in DS that uses triples of the form $(a_1, \mathit{related\_with}, a_2)$ to indicate that the SPARQL *endpoints located at the IRIs a_1 and a_2 store related data. Moreover, assume that P_2 is the following graph pattern:*

$$P_2 = \Big[(?U_1, \mathit{related\_with}, ?U_2) \text{ AND } \Big(\text{SERVICE } ?U_1 \Big((?N, \mathit{email}, ?E) \text{ OPT } (\text{SERVICE } ?U_2\ (?N, \mathit{phone}, ?F))\Big)\Big)\Big].$$

When this query is evaluated over the dataset DS and the RDF graph G in DS, it returns for every tuple $(a_1, \mathit{related\_with}, a_2)$ in G, the list of names and email addresses that can be retrieved from the SPARQL endpoint located at a_1, together with the phone number for each person in this list for which this data can be retrieved from the SPARQL endpoint located at a_2 (recall that graph pattern (SERVICE $?U_2$ $(?N, \mathit{phone}, ?F)$) *is nested inside the first SERVICE operator in P_2). To evaluate this query over an RDF graph, first it is necessary to determine the possible values for variable $?U_1$, and then to submit the query* $((?N, \mathit{email}, ?E)$ OPT (SERVICE $?U_2$ $(?N, \mathit{phone}, ?F)))$ *to each one of the endpoints located at the IRIs stored in $?U_1$. In this case, variable $?U_2$ is bound (and also strongly bound) in P_2. However, this variable is not bound in the graph pattern* $((?N, \mathit{email}, ?E)$ OPT (SERVICE $?U_2$ $(?N, \mathit{phone}, ?F)))$, *which has to be evaluated in some of the SPARQL endpoints stored in the RDF graph where P_2 is being evaluated, something that is infeasible in practice. It is important to notice that the difficulties in evaluating P_2 are caused by the nesting of* SERVICE *operators (more precisely, by the fact that P_2 has a sub-pattern of the form* (SERVICE $?X_1$ Q_1), *where Q_1 has in turn a sub-pattern of the form* (SERVICE $?X_2$ Q_2) *such that $?X_2$ is bound in P_2 but not in Q_1).* □

In the following section, we use the concept of strongly boundedness to define a notion that ensures that a SPARQL query containing the SERVICE operator can be evaluated in practice, and which takes into consideration the ideas presented in the above examples.

2.2. *The notion of service-safeness: Considering sub-patterns and nested* SERVICE *operators*

The goal of this section is to provide a condition that ensures that a SPARQL query containing the SERVICE operator can be safely evaluated in practice. To this end, we first need to introduce some terminology. Given a SPARQL query P, define $\mathcal{T}(P)$ as the *parse* tree of P. In this tree, every node corresponds to a sub-pattern of P. An example of a parse tree of a pattern Q is shown in Figure 2. In this figure, $u_1, u_2, u_3, u_4, u_5, u_6$ are the identifiers of the nodes of the tree, which are labeled with the sub-patterns of Q. It is important to notice that in this tree we do not make any distinction between the different operators in SPARQL, we just use the child relation to store the structure of the sub-patterns of a SPARQL query.

Tree $\mathcal{T}(P)$ is used to define the notion of service-boundedness, which extends the concept of boundedness, introduced in the previous section, to consider variables that are bound inside sub-patterns and nested SERVICE operators. It should be noticed that these two features were identified in the previous section as important for the definition of a notion of boundedness (see Example 1).

Definition 3 (Service-boundedness). *A SPARQL query P is service-bound if for every node u of $\mathcal{T}(P)$ with label* (SERVICE $?X$ P_1), *it holds that:*

(1) there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $?X$ is bound in P_2;
(2) P_1 is service-bound.

For example, query Q in Figure 2 is service-bound. In fact, condition (1) of Definition 3 is satisfied as u_5 is the only node in $\mathcal{T}(Q)$ having as label a SERVICE graph pattern, in this case (SERVICE $?X$ $(?Y, a, ?Z)$), and for the node u_3, it holds that: u_3 is an ancestor of u_5 in $\mathcal{T}(P)$, the label of u_3 is $P = ((?X, b, c)$ AND (SERVICE $?X$ $(?Y, a, ?Z)))$ and $?X$ is bound in P. Moreover, condition (2) of Definition 3 is satisfied as the sub-pattern $(?Y, a, ?Z)$ of the label of u_5 is also service-bound.

The notion of service-boundedness captures our intuition about the condition that a SPARQL query containing the SERVICE operator should satisfy. Unfortunately, the following theorem shows that such a condition is undecidable and, thus, a SPARQL query engine would not be able to check it in order to ensure that a query can be evaluated.

Theorem 2. *The problem of verifying, given a SPARQL query P, whether P is service-bound is undecidable.*

Proof: As in the proof of Theorem 1, we use the undecidability of the satisfiability problem for SPARQL to show that the theorem holds. Let P be a SPARQL graph pattern and $?X$, $?Y$, $?Z$, $?U$, $?V$, $?W$ be variables that are not mentioned in P, and assume that P does not mention the operator SERVICE (recall that the satisfiability problem is already undecidable for the fragment

$$\begin{tikzcd}
 & u_1 : ((?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } (\text{SERVICE } ?X\ (?Y, a, ?Z)))) \arrow[dl] \arrow[dr] & & \\
u_2 : (?Y, a, ?Z) & & u_3 : ((?X, b, c) \text{ AND } (\text{SERVICE } ?X\ (?Y, a, ?Z))) \arrow[dl] \arrow[dr] & \\
 & u_4 : (?X, b, c) & & u_5 : (\text{SERVICE } ?X\ (?Y, a, ?Z)) \arrow[d] \\
 & & & u_6 : (?Y, a, ?Z)
\end{tikzcd}$$

Figure 2: Parse tree $\mathcal{T}(Q)$ for the graph pattern $Q = ((?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } (\text{SERVICE } ?X\ (?Y, a, ?Z))))$.

of SPARQL consisting of the operators AND, UNION, OPT and FILTER). Then define a SPARQL query Q as:

$$Q = \Big(\big((?X, ?Y, ?Z) \text{ UNION } P\big) \text{ AND } \big(\text{SERVICE } ?X\ (?U, ?V, ?W)\big)\Big).$$

Next we show that Q is service-bound if and only if P is not satisfiable.

($\Leftarrow$) If P is not satisfiable, then Q is equivalent to the pattern:

$$Q' = ((?X, ?Y, ?Z) \text{ AND } (\text{SERVICE } ?X\ (?U, ?V, ?W))),$$

which is service-bound since variable $?X$ is bound in Q'.

($\Rightarrow$) Assume that P is satisfiable. Then given that variable $?X$ is not mentioned in P, we have that $?X$ is not bound in the graph pattern $((?X, ?Y, ?Z) \text{ UNION } P)$. Thus, given that $?X$ is neither bound in (SERVICE $?X$ $(?U, ?V, ?W)$), we deduce that query Q is not service-bound since $?X$ is not a bound variable in Q.

Therefore, we have shown that the complement of the satisfiability problem for SPARQL can be reduced to the problem of verifying, given a SPARQL query P, whether P is service-bound. From this we conclude that the theorem holds. □

As for the case of the notion of boundedness, the fact that the notion of service-boundedness is undecidable prevents one from using it as a restriction over the variables used in SERVICE calls. To overcome this limitation, in the definition of service-boundedness, we replace the restriction that the variables used in SERVICE calls are bound by the decidable restriction that they are strongly bound. In this way, we obtain a syntactic condition over SPARQL patterns that ensures that they are service-bound, and which can be efficiently verified.

Definition 4 (Service-safeness). *A SPARQL query P is service-safe if for every node u of $\mathcal{T}(P)$ with label* (SERVICE $?X$ P_1)*, it holds that:*

(1) there exists a node v of $\mathcal{T}(P)$ with label P_2 such that v is an ancestor of u in $\mathcal{T}(P)$ and $?X \in \text{SB}(P_2)$;
(2) P_1 is service-safe.

As a corollary of Proposition 1, we obtain the following proposition.

Proposition 2. *If a SPARQL query P is service-safe, then P is service-bound.*

Prior to starting with the most technological part of this article, we describe to the reader an algorithm for SERVICE safeness checking. Our system uses a bottom-up algorithm over the parse tree $\mathcal{T}(Q)$ of a SPARQL query Q for validating the service-safeness condition. This procedure traverses the parse tree $\mathcal{T}(Q)$ twice for ensuring that Q can be correctly evaluated. In the first traversal, for each node identifier u of $\mathcal{T}(Q)$, the algorithm computes the set of strongly bound variables for the label P of u. For example, in the parse tree shown in Figure 2, the variable $?X$ is identified as the only strongly bound variable for the label of the node with identifier u_3. In the second traversal, the bottom-up algorithm uses these sets of strongly bound variables to check two conditions for every node identifier u of $\mathcal{T}(Q)$ with label of the form (SERVICE $?X$ P): whether there exists a node v of $\mathcal{T}(Q)$ with label P' such that v is an ancestor of u in $\mathcal{T}(Q)$ and $?X$ is strongly bound in P', and whether P is itself service-safe. If these two conditions are fulfilled, then the algorithm returns *true* to indicate that Q is service-safe. Otherwise, the procedure returns *false*.

3. Optimizing the Evaluation of the OPTIONAL Operator in SPARQL Federated Queries

If a SPARQL query Q including the SERVICE operator has to be evaluated in a SPARQL endpoint A, then some of the sub-queries of Q may have to be evaluated in some external SPARQL endpoints. Thus, the problem of optimizing the evaluation of Q in A, and, in particular, the problem of reordering Q in A to optimize this evaluation, becomes particularly relevant in this scenario, as in some cases one cannot rely on the optimizers of the external SPARQL endpoints. Motivated by this, we present in this section some optimization techniques that extend the techniques presented in [8] to the case of SPARQL queries using the SERVICE operator, and which can be applied to a considerable number of SPARQL federated queries.

3.1. Optimization via well-designed patterns

In [8, 9], the authors study the complexity of evaluating a pattern in the fragment of SPARQL consisting of the operators AND, UNION, OPT and FILTER. One of the conclusions of these papers is that the main source of complexity in SPARQL comes from the use of the OPT operator. In fact, it is proved in [8] that the complexity of the problem of verifying, given a mapping μ, a SPARQL pattern P, a dataset DS and an RDF graph G in DS, whether $\mu \in [\![P]\!]_G^{DS}$ is PSPACE-complete, and it is proved in [9] that this bound remains the same if only the OPT operator is allowed in SPARQL patterns. In light of these results, in [8] a fragment was introduced of SPARQL that forbids a special form of interaction between variables appearing in optional parts, which rarely occurs in practice. The patterns in this fragment, which are called well-designed patterns [8], can be evaluated more efficiently and are suitable for reordering and optimization. In this section, we extend the definition of the notion of being well-designed to the case of SPARQL patterns using the SERVICE operator, and prove that the reordering rules proposed in [8], for optimizing the evaluation of well-designed patterns, also hold in this extension. The use of these rules allows to reduce the number of tuples being transferred and joined in federated queries, and hence our implementation benefits from this as shown in Section 4.

Let P be a graph pattern constructed by using the operators AND, OPT, FILTER and SERVICE, and assume that P satisfies the safety condition that for every sub-pattern $(P_1$ FILTER $R)$ of P, it holds that $var(R) \subseteq var(P_1)$. Then, by following the terminology introduced in [8], we say that P is well-designed if for every sub-pattern $P' = (P_1$ OPT $P_2)$ of P and for every variable $?X$ occurring in P: If $?X$ occurs both inside P_2 and outside P', then it also occurs in P_1. All the graph patterns given in the previous sections are well-designed. On the other hand, the following pattern is not well-designed:

$$P = \Big((?X, \text{nickname}, ?Y) \text{ AND } (\text{SERVICE } c \; ((?X, \text{email}, ?U) \text{ OPT } (?Y, \text{email}, ?V)))\Big)$$

as for the sub-pattern $P' = (P_1$ OPT $P_2)$ of P with $P_1 = (?X, \text{email}, ?U)$ and $P_2 = (?Y, \text{email}, ?V))$, we have that $?Y$ occurs in P_2 and outside P' in the triple pattern $(?X, \text{nickname}, ?Y)$, but it does not occur in P_1. Given an RDF graph G, graph pattern P retrieves from G a list of people with their nicknames, and retrieves from the SPARQL endpoint located at the IRI c the email addresses of these people and, optionally, the email addresses associated to their nicknames. What is unnatural about this graph pattern is the fact that $(?Y, \text{email}, ?V)$ is giving optional information for $(?X, \text{nickname}, ?Y)$, but in P appears as giving optional information for $(?X, \text{name}, ?U)$. In fact, it could happen that some of the results retrieved by using the triple pattern $(?X, \text{nickname}, ?Y)$ are not included in the final answer of P, as the value of variable $?Y$ in these intermediate results could be incompatible with the values for this variable retrieved by using the triple pattern $(?Y, \text{email}, ?V)$. To overcome this limitation, one should use instead the following well-designed SPARQL graph pattern:

$$\Big[\Big((?X, \text{nickname}, ?Y) \text{ AND } (\text{SERVICE } c \; (?X, \text{email}, ?U))\Big) \text{ OPT } (\text{SERVICE } c \; (?Y, \text{email}, ?V))\Big]$$

In the following proposition, we show that well-designed patterns including the SERVICE operator are suitable for reordering and, thus, for optimization.

Proposition 3. *Let P be a well-designed pattern and P′ a pattern obtained from P by using one of the following reordering rules:*

$$((P_1 \text{ OPT } P_2) \text{ FILTER } R) \longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2),$$

$$(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) \longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3),$$

$$((P_1 \text{ OPT } P_2) \text{ AND } P_3) \longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2).$$

Then P' is a well-designed pattern equivalent to P.

The proof of this proposition is a simple extension of the proof of Proposition 4.10 in [8].

In our federated SPARQL query engine (SPARQL-DQP), we have implemented the rewriting rules shown in Proposition 3 with a bottom up algorithm for checking the condition of being well-designed. In the following section, we describe the details of the implementation of these algorithms and the architecture of SPARQL-DQP.

4. Implementation of SPARQL-DQP and Well-Designed Patterns Optimization

In this section, we describe the implementation details of the SPARQL-DQP system and, in particular, we describe how we implemented the optimization techniques for well-designed SPARQL graph patterns (which are presented in Section 3).

We base our implementation on the use of Web Service-based access to data sources. WS-based access is a widely used technology in the data intensive scientific workflow community, and several systems for accessing large amounts of data already use this approach in their implementation. Some of these data workflow systems are presented in [18, 12, 19]. These systems have been successfully used in a variety of data intensive scenarios like analyzing data from the Southern California Earthquake Center [20], data from biological domains like post genomic research [21], analysis of proteins and peptides from tandem mass spectrometry data [22], cancer research [23], meteorological phenomena [24] or used in the German grid platform [25]. In these scenarios, the systems accessed and processed petabytes of data, and we are convinced that the approach they use is the most suitable for managing the large amounts of data present in the LOD cloud.

We will provide some background on WS-based access to data sources, before describing in more detail our implementation. But first we will briefly introduce the reader to the state of the art of distributed query systems.

4.1. Introduction to Data Integration and Query Federation

There are several approaches for integrating heterogeneous data sources. In [26], the author provides an initial classification of different architectures for this purpose. One of the architectures is a mediator-wrapper architecture [27] which provides an integrated view of the data that resides in multiple databases. A schema for the integrated view is available from the mediator, and queries can be made against that schema. This schema can be generated in two different ways, using a Global as View (GAV) [28] or a Local as View (LAV) [29] approach. One example of a mediator system based on the GAV approach is Garlic [30]. Besides of Garlic, other mediator systems that pioneered the work on distributed query processing and data integration were the TSIMISS project [31] and the Information Manifold [32], among others.

Another type of architecture for accessing distributed data are query federation systems. Federated architectures provide a framework in which several databases can join in a federation. As members of the federation, each database extends its schema to incorporate subsets of the data held in the other member databases. In most cases, a virtualized approach is supported for this approach [33]. In [34], a general architecture[9] is presented with the following components: query parser, query rewriter, query optimizer, plan refinement component and query execution engine.

We base our approach on extending a query federation system (OGSA-DQP [13]) built on top of a data workflow system (OGSA-DAI [12]) targeted at dealing with large amounts of data in e-Science applications [35, 36], as we mentioned before. In the next subsection we describe the architecture of the extended system and the specific characteristics for dealing with large amounts of data: data streaming and process parallelization.

4.2. OGSA-DAI and OGSA-DQP

OGSA-DAI[10] is a framework that allows access, transformation, integration and delivery of distributed data resources. The data resources supported by OGSA-DAI are relational databases, XML databases and file systems. These features are collectively enabled through the use of data workflows which are executed within the OGSA-DAI framework. The components of the data workflows are activities: well-defined functional units (data goes in, the data is operated on, data comes out), and can be viewed as equivalent to programming language methods. One key characteristic of the architecture is that data is streamed between activities so these data can be consumed by the next activity on the workflow as soon as it is outputted. The other key feature of the workflow execution engine is that all

[9]In fact, this architecture is generic to all kind of query processing systems, not only distributed query processors

[10]http://www.ogsadai.org.uk/

activities within a data workflow are executed in parallel: data streams go through activities in a pipeline-like way (as soon as a data unit is processed by an activity this data unit is buffered or sent to the next activity in the pipeline), and each activity operates on a different portion of a data stream at the same time.

The distributed query processor (DQP) [13] is a set of activities within the OGSA-DAI framework that execute SQL queries on a set of distributed relational databases managed by OGSA-DAI. OGSA-DQP receives as input an SQL query addressed to a set of distributed databases. It parses the query identifying to which of the databases in the federation these queries are addressed, and creates a data workflow using the OGSA-DAI activities. This data workflow is executed within the OGSA-DAI workflow execution engine and results are sent back to the client.

The deployment of OGSA-DAI/DQP can be done in several Web application servers, depending on how much we want to distribute the processing and how many remote datasets we want to access. The standard configuration is of an OGSA-DAI instance running in a Web server for each data source we want to access, but other configurations are available. For instance, it could be possible to configure OGSA-DAI with a single server which would be in charge of accessing all datasets in the federation. Figure 3 shows a possible deployment configuration of OGSA-DAI. In that Figure, there is a main node (HQResource) which is in charge of coordinating the federation of data sources. At startup, this node gathers information about the existing data sources that are wrapped at the remote OGSA-DAI. In the Figure there are two other OGSA-DAI nodes which expose the remote data. In this example we expose an RDF database and two SPARQL endpoints. SPARQL endpoints are managed in a slightly different manner than the other data resources (SQL and RDF databases): they can be loaded dynamically in the remote data nodes without previously configuring them. The processing of a distributed SPARQL query is presented in the next section.

4.3. SPARQL-DQP implementation

From a high level point of view, SPARQL-DQP can be defined as an extension of OGSA-DQP that considers an additional query language: SPARQL. The design of SPARQL-DQP follows the idea of adding a new type of data source (RDF data sources) to the standard data sources managed by OGSA-DAI, and extending the parsers, planners, operators and optimizers that are handled by OGSA-DQP in order to handle the SPARQL query language.

We extend OGSA-DQP to accept, optimize and distribute SPARQL queries across different data nodes. SPARQL-DQP reads the SPARQL query, it creates a basic logical query plan (LQP), optimizes it, next it selects in which nodes is going to be executed that query plan, and finally it executes the query plan using the workflow engine. For that, a new coordinator of the distributed query processor is needed (HQResource in Figure 3). This coordinator extends the original OGSA-DQP coordinator in such a way that accepts SPARQL queries. Also, other components are extended or developed like the new OGSA-DQP's data dictionary that contains information about the federation nodes, the SPARQL query parser and the SPARQL LQP builder plus its optimizer. At initialization time the SPARQL-DQP resource checks the availability of the data nodes in which the federation will be executed, and obtains their characteristics which are stored in a data dictionary. These characteristics are information about ad-hoc functions implemented by the remote RDF resource, data node information (like security information, connection information and data node address) and table metadata (currently only the RDF repository name, to be extended with statistics about the data in the datasets). This information is used to build the SPARQL LQP and to configure the federation.

The SPARQL LQP Builder takes the abstract syntax tree generated by the SPARQL parser and produces a logical query plan. The logical query plan follows the semantics defined by the SPARQL-WG[11] in the SPARQL 1.1 Federated Query extension specification [7], which is also formalized in Section 1.2. The query plan produced represents the SPARQL query using a mix of operators and activities coming from the existing ones in OGSA-DQP and the newly added SPARQL operators (like the SPARQL union, filters, the specific SPARQL optimizations, scans, etc.).

Next, the OGSA-DQP chain of optimizers is applied, and we add rewriting rules based on well-designed pattern based optimizations. Besides, safeness rules have to be checked as we described in Section 3.1, since some SQL optimizers can only be applied to safe SPARQL patterns.

In the final stage of the query processing, the generated remote requests and local sub-workflows are executed and the results collected, and returned by the activity.

[11]http://www.w3.org/2009/sparql/wiki/

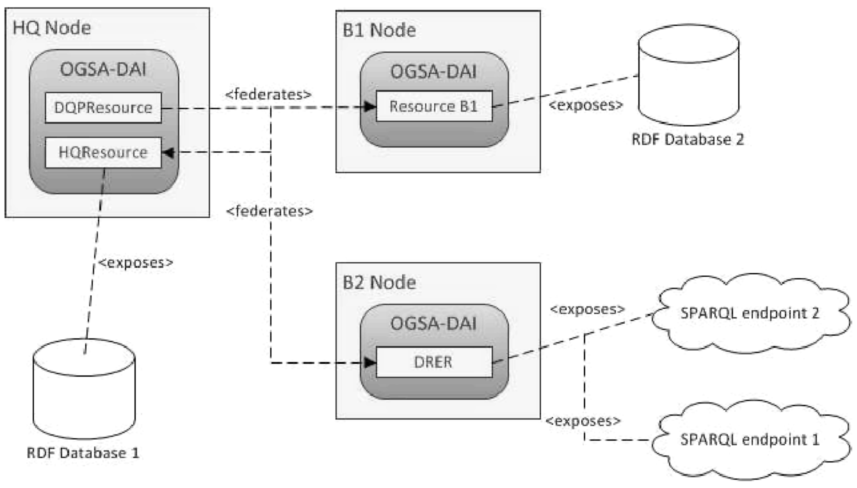

Figure 3: Deployment of OGSA-DAI

4.4. Other federated SPARQL querying processing systems

In this section, we briefly describe similar systems that provide some support for SPARQL query federation. Some of the existing engines supporting the SPARQL 1.1 Federated Query extension are ARQ[12], RDF-Query[13], Rasqal RDF query Library[14] and ANAPSID [10] among others. There are also other systems which implement a distributed query processing system for SPARQL like DARQ [4], Networked Graphs [5], SPLENDID [37], FedX 1.1 [6], the system by Ladwig et al. [38] and SemWIQ [39], but they do not follow the official SPARQL 1.1 Federation specification. Another system that supports distributed RDF querying is presented in [40]. However, we do not consider it here as it uses the query language SeRQL instead of SPARQL.

We will now describe briefly each of these systems. ANAPSID implements two adaptive operators: the *agjoin* and the *adjoin* operators. The agjoin operator uses a hash join along with storing join tuples for speeding up join operators. The adjoin operator, hides delays coming from the data sources and perform dereferences for certain predicates.

The system by Ladwig et al. [38] implements a join operator called Symmetric Index Hash Join (SIHJoin), which combines queries to remote SPARQL endpoints with queries to local RDF data stores. When this situation happens, data retrieved from the local RDF dataset is stored in an index hash structure for faster access when performing a join with remote data. The authors also provide cost models and the use of non-blocking operators for joining data.

FedX also extends Sesame[15] and bases its optimizations in grouping joins that are directed to the same SPARQL endpoints and rule join optimizer using a heuristics-based cost estimation. FedX also reduces the number of intermediate joins by grouping sets of mappings in a single subquery (exclusive groups) and also bound joins, a technique that uses the results from one remote exclusive group to constrain the next grouped query using SPARQL UNION.

SPLENDID extends Sesame[16] adding a statistics-based join reordering system. SPLENDID bases its optimizations in join reordering rules based in a cost model described in [37]. The statistics are collected from VoID descriptions and allow to perform join reordering in an efficient manner.

SemWIQ is a mediator-wrapper based system, where heterogeneous data sources (available as CSV files, RDF datasets or relational databases) are accessed by a mediator through wrappers. Queries are expressed in SPARQL and consider OWL as the vocabulary for the RDF data. SemWIQ uses the Jena's SPARQL processor ARQ to generate query plans and it applies its own optimizers. These optimizers mainly consist in rules to move down filters or unary operators in the query plan, together with join reordering based on statistics. The system has a registry catalog that indicates where the sources to be queried are and the vocabulary to be used. Currently, the system does not handle SPARQL endpoints but this is being updated at the time of writing this paper.

[12]http://jena.sourceforge.net/ARQ/
[13]http://search.cpan.org/dist/RDF-Query/
[14]http://librdf.org/rasqal
[15]a framework for processing RDF data
[16]http://www.openrdf.org/

DARQ extends the Jena's SPARQL processor ARQ. This extension requires attaching a configuration file to the SPARQL query, with information about the SPARQL endpoints, vocabulary and statistics. DARQ applies logical and physical optimizations, focused on using rules for rewriting the original query before query planning (so as to merge basic graph patterns as soon as possible) and moving value constrains into subqueries to reduce the size of intermediate results. Other important drawback of DARQ is that it can only execute queries with bound predicates. Unfortunately, DARQ is no longer maintained.

Networked Graphs creates graphs for representing views, content or transformations from other RDF graphs, and allowing the composition of sets of graphs to be queried in an integrated manner. The implementation considers optimizations such as the application of distributed semi-join optimization algorithms.

5. Evaluation

The objective of our evaluation is to show that the architecture chosen (the extension of a well-known data workflow processing system) is more suitable for processing the large amounts of RDF data that are available in the Web, specially when remote SPARQL endpoints do not impose any kind of restriction over the amount of results returned. For that, we decided to run the experiments in an uncontrolled environment such as the Web. In this uncontrolled environment the behavior of endpoints and latencies can vary largely among executions, hence leading to evaluation results that are not clearly comparable across systems and replicable. In despite of that, these evaluation results provide some indications about the behaviors of these systems that will be important for characterizing each tool. We also run the evaluation in a controlled environment using synthetic data which will be distributed across several SPARQL endpoints. In this evaluation we will show the behavior of the optimization techniques proposed in Section 3.1, and how these optimization techniques reduce the amount of intermediate results of the SPARQL queries and thus how its use actually reduces the time needed to process queries when compared to non optimized approaches.

5.1. Note on other systems' evaluation

We compared our system with FedX 1.1 and FedX 1.1 using SERVICE, ARQ (2.8.8) and RDF::Query (2.908). We chose these systems because two of them are part of the official SPARQL implementations and FedX is based in Sesame using an endpoint virtualization approach. Also, FedX 1.1 adds statistical models for join reordering and the SERVICE keyword to its normal federated query processing engine. When FedX is not using SERVICE, it uses the predicates in the query for identifying the right dataset to which the queries should be directed which makes the system to query more SPARQL endpoints than the other systems. In order to provide a more fair comparison when querying synthetic data, we adapted the datasets described in the next section so the predicates in each SPARQL endpoint are not repeated across them. In this way FedX can uniquely identify each dataset by looking at the triple pattern predicates so a more fair comparison can be done. Thus, we compare twice to FedX: first we compare to FedX using the SERVICE operator and next to FedX using the virtualization of remote SPARQL endpoints. Regarding the query execution, ARQ differs with the other implementations of SPARQL 1.1, since it generates bind join queries like FedX, which also results in the generation of many SPARQL queries to the same remote endpoint and sometimes many connection errors from these servers. RDF::Query is the last system evaluated and the one that follows more closely the algorithms described in the official SPARQL 1.1 document.

In this evaluation we opted for a representative set of systems but without fully covering the state of the art in distributed SPARQL query processing systems. The aim of this section is to provide an overview of the most common system architectures to federate SPARQL queries, not to perform an exhaustive evaluation of the existing SPARQL query federation systems.

5.2. Query Selection

We reuse many of the queries proposed in Fedbench [41]. Fedbench proposes three sets of queries: a cross domain set of queries which distributes queries across widely used SPARQL endpoints such as DBpedia[17] and the LinkedMDB endpoint[18]; life sciences set of queries which evaluate how systems query one of the largest domain in the LOD cloud; finally Fedbench also proposes the use of the SP2Bench [42] evaluation benchmark, focused on evaluating the robustness and performance of RDF data stores.

However, the queries in the Fedbench evaluation framework do not take into account the SPARQL 1.1 Federated Query extension. That means that the queries

[17]http://dbpedia.org/sparql
[18]http://data.linkedmdb.org/sparql

do not contain the SERVICE keyword, instead, the query engines have to identify to which endpoint direct each part of those queries. We modified manually the queries adding the SERVICE keyword where necessary. Furthermore, Fedbench does not contain many SPARQL patterns that are common in most of the user queries like FILTER or OPTIONAL [43].

Looking carefully, the queries in the original Cross Domain query set did not contain any SPARQL pattern that used either OPTIONAL or the FILTER operators. From the total of queries submitted to DBpedia in a month the OPTIONAL operator is used in 39% of these queries and the FILTER operator is used in a 46% of them [43]. Thus, we decided to complement the query set with queries containing combinations of these missing patterns (FILTER and OPTIONAL).The life sciences domain queries contain a variety of SPARQL queries, including OPTIONAL, FILTER and UNION operators, thus, we decided not to add any new query to the existing ones. The SP2Bench queries are taken from the original benchmark targeted at measuring the performance of RDF databases, and thus, some adaptations have to be done if we want to use it within distributed SPARQL query processors. In this set of queries also some important query patterns are missing, and thus we added some queries to solve this problem.

For evaluating the previous systems in an uncontrolled environment like the Web of data, we run all described queries five times and we apply an arithmetic mean to the results of these five queries. In this way, we provide a more homogenized set of results that reflect better the systems' real performance. We also perform two warm-up queries to avoid initial delay of the systems configuration on their first run.

5.2.1. New queries used in the evaluation

We added three queries to the cross domain query set and five more queries to the SP2Bench set of queries. The new cross domain queries are query CDQ4b, CDQ8 and CDQ9 in Appendix C. In CDQ4b we added a new FILTER to the original query (cross domain query 4 in [41]), asking now for those actors that appear in any NY Times news, filtering for the film 'Tarzan'. Queries, CDQ8 and CDQ9 are completely new. In CDQ8 we query DBpedia and the El Viajero [44] SPARQL endpoint[19] for data about countries and existing travel books, we filter for countries with a surface greater than 20,000 km^2. In CDQ9 we query DBpedia for countries and optionally we get the existing travel books for these countries from the El Viajero endpoint, completing this information with the climate data at the CIA world factbook SPARQL endpoint[20]. Next we show CDQ9 to give the reader an idea of the type of queries that we are considering:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX imdb: <http://data.linkedmdb.org/resource/movie>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?title ?actor ?news ?director ?film WHERE {
 SERVICE <http://data.linkedmdb.org/sparql> {
  ?film dcterms:title ?title .
  ?film imdb:actor ?actor .
  ?film imdb:production_company
   <http://data.linkedmdb.org/resource/production_company/15>.
  ?actor owl:sameAs ?x .
 } OPTIONAL {
 SERVICE<http://api.talis.com/stores/nytimes/services/sparql>{
   ?y owl:sameAs ?x .
   ?y <http://data.nytimes.com/elements/topicPage> ?news
  }
 }
 FILTER (?title="Tarzan")
```

We also added five queries to the SP2Bench set of queries in Fedbench, which are an extension of the SP2Bench queries 7 and 8, that ask for proceedings or journals and their authors. We added an extra level of complexity first by adding OPTIONAL to those queries. SP2BQ7b asks for journals, optionally it obtains the authors' publications in a conference, and later it obtains also the authors' names. We modified SP2BQ8 to ask for papers in some collection of papers instead of asking for journal papers since SP2BQ7 already asks for that. SP2BQ8b asks for all people and optionally obtains all the papers these people published in a conference for later on joining the results with the people that published a paper in a collection. SP2BQ7c asks for all journals, obtaining their authors with an optional and filtering for the number of pages. Query SP2BQ8c is the most complex query since it queries 4 different SPARQL endpoints. In this query, we query for all papers in a conference, optionally obtaining the people who wrote them, next asking for those authors that also wrote a journal paper and also the paper which is in a paper collection. Query SP2BQ8d asks for all the papers in conference proceedings, optionally obtaining their authors and limiting the output data to those proceedings from the year 1950.

To the previous queries we add the queries in [45]. These queries follow the following path: first, querying GeneId endpoint we obtain Pubmed which we use to access the Pubmed endpoint (queries Q1 and Q2). In

[19]http://webenemasuno.linkeddata.es/sparql

[20]http://www4.wiwiss.fu-berlin.de/factbook/sparql

these queries, we retrieve information about genes and their references in the Pubmed dataset. From Pubmed we access the information in the National Library of Medicine's controlled vocabulary thesaurus (queries Q3 and Q4), stored at MeSH endpoint, so we have more complete information about such genes. Finally, to increase the data retrieved by our queries, we also access the HHPID endpoint (queries Q5, Q6 and Q7), which is the knowledge base for the HIV-1 protein. These queries can be found in [45].

5.3. Datasets description

For the cross domain queries mentioned in [41] we used the datasets available at the DBpedia, LinkedMDB, Geonames, the New York Times and El Viajero SPARQL endpoints. We did not download any data to a local server, instead we queried directly these endpoints. We did similarly for the life sciences queries, accessing the default SPARQL endpoints (which are Drugbank, Kegg and DBpedia). For the SP2Bench queries, we generated a dataset of 1.000.000 triples which we clustered into 5 different SPARQL endpoints in a local server. The local SPARQL endpoints were Journal (410.000 triples), InCollections (8.700 triples), InProceedings (400.000 triples), People (170.000 triples) and Masters (5.600 triples).

5.4. Results

Our evaluation was done on a Pentium Xeon with 4 cores and 8 GB of memory run by an Ubuntu 11.04. The data and the queries used in this evaluation can be found in `http://www.oeg-upm.net/SparqlDQP/jws`. The results of our evaluation are shown in Figures 4, 5 and 6 for the Fedbench sets of queries in Appendix C. The data for generating these charts can also be found in Appendix B. In that appendix Tables B.1, B.2 and B.3 present the results of the query executions. For the life sciences set of queries we refer to Table B.2 and also to the previous work present in [45]. We represent as 600,000ms those queries that need more than 10 minutes to be answered by the evaluated systems (SPARQL-DQP, SPARQL-DQP optimized, ARQ, RDF::Query and FedX 1.1 with SERVICE and without it). The results are presented in a logarithmic scale.

The results presented in Figure 4 show how the evaluated systems performed in the Cross domain set of queries. These queries show how the systems behave in a typical situation, in which users query some of the most common SPARQL endpoints. These endpoints, as commented before have been DBpedia, the NYTimes endpoint, LinkedMDB, Geonames, El Viajero and the CIA world factbook. These remote endpoints usually return between 10.000 and 2.00 results. This makes all systems answer queries in reasonable times.

One of the problems when querying remote SPARQL endpoints is the update rate of the data contained in the datasets. Sometimes, when querying these endpoints the data may have been updated and the queries used previously may not return the same results (or any) again. This is the situation in queries 5, 6 and 7, in which there are no results returned. In the evaluation of these queries, FedX 1.1 without using the SERVICE keyword is the fastest since it uses first an ask query to know there will be any result or not. Regarding the execution of the rest of the queries, all systems performed similarly, especially in the first four queries and in query 4b. For the same query 4b, all systems returned the same amount of results, except for both FedX versions: FedX 1.1 using SERVICE returns 84 results while the FedX version that virtualizes a list of SPARQL endpoints if they were a single one returns no results. In query 8, FedX using SERVICE gives an evaluation error due to the use of statistical-based pattern reordering (*"it is not supported filter reordering without statistics"*) and FedX without SERVICE returns no results. The difference in the amount of results between both FedX flavours is that they use a different approach for querying the remote SPARQL endpoints. While the FedX flavour that implements the SERVICE operator queries only the specified RDF datasets, the other FedX version virtualizes all the RDF datasets in its list and thus uses a different query evaluation strategy (FedX without SERVICE queries all SPARQL endpoints in its list retrieving as much data as possible). The other systems performed similarly but ARQ needed more time than the others, almost one order of magnitude. ARQ also inserted the FILTER expression in the SERVICE call, giving a different amount of results that the other systems implementing SPARQL 1.1 Fed. In the last query, SPARQL-DQP performs better than the others which either do not halt (RDF::Query and ARQ do not finish their processing) or give an error in the query execution (FedX with SERVICE: *"left join nor supported for cost optimization"*). We think that SPARQL-DQP performed better because of the architecture chosen. Some of the endpoints queried in CDQ9 returned 10,000 results, which is a significant increase comparing to the other endpoints queried (normally they returned 2,000 results). When the amount of data increased, our system performed better, as query CDQ9 showed. The amount of results returned for these queries was of 8.604 for

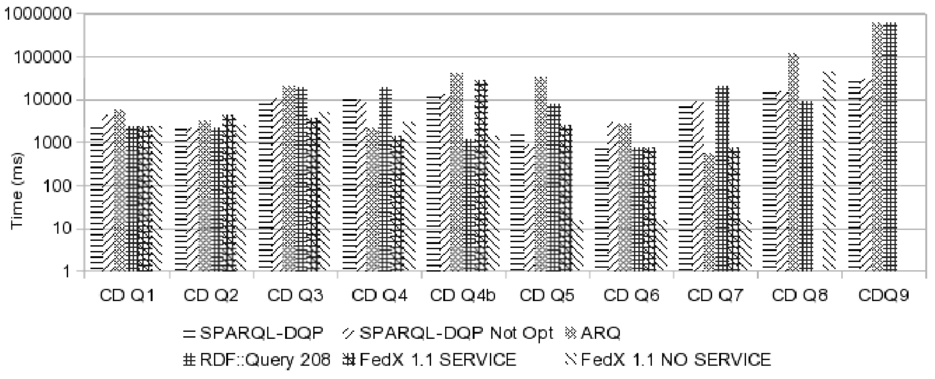

Figure 4: Cross Domain Query Results

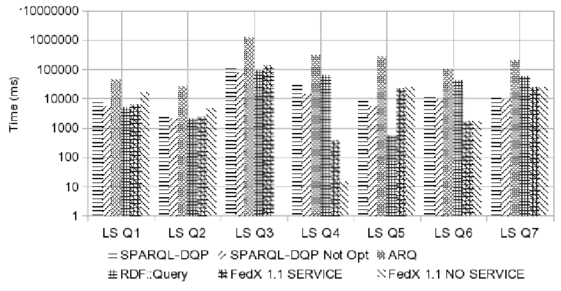

Figure 5: Life Science Query Results

the systems following the SPARQL 1.1 Federation document. The optimizations presented in Section 3.1 were applied in queries CDQ4b, CDQ8 and CDQ9 but they did not reduce the final result times. This is due to the fact that the amount of data transferred between the query operators was not significant enough.

Figure 5 shows the times needed for evaluating the Life Science domain queries. We did not add any extra query to the evaluation, since the query set already contains the most common patterns used in SPARQL queries, and a more complete evaluation in the life science domain can be found in [45]. As in the previous set of queries, the RDF datasets were updated and some queries (LSQ4 and LSQ5) did not return any result. In general, all systems behaved similarly in this set of queries. ARQ performed a bit worse, mainly due to the way it manages the connections with the remote SPARQL endpoints (ARQ generates a set of binding queries restricting some of the remote SPARQL queries which generates an overload over the remote endpoints, which was a common problem for all systems). The Life Science domain SPARQL endpoints usually reject queries from a host when too many connections are asked, which in the case of an intense evaluation may be a common problem. Life sciences servers behaved worse in our evaluation returning server errors, specially when the bound join query technique was used. Regarding SPARQL-DQP and the other systems, they performed similarly but when data increased in query LS7. In that situation, SPARQL-DQP worked better than other systems. The implemented optimizations (specially the implementation of the pattern reordering rules described in Section 3.1) are less noticeable when the amount of transferred data (and number of intermediate results) is lower, but there is no loosing of performance in the applications of the rules.

The results represented in Figure 6 show how the evaluated systems behaved with larger amounts of data. In this evaluation, the SPARQL endpoints do not have any result limit restriction, which is of key importance in the evaluation. The configuration of the endpoints is as follows: the People endpoint contains 82.685 persons with name, the InProceedings endpoint contains 65.863 in proceedings with author, the InCollections contains 615 papers in belonging to a collection of papers with author, and the Journal endpoint contains 83.706 journals with author. In total we used 1.000.000 triples distributed in the previous endpoints. From a results point of view, all the systems that implement the SPARQL 1.1 Federated Query extension returned the same amount of results in the first set of queries (SP2BQ1 to SP2BQ5). In the rest of the queries, again the systems implementing the SPARQL federation extension returned the same amount of results, while FedX (not using SERVICE) returned (when possible) different amounts due to FedX accesses the SPARQL endpoints virtualizing all of them rather than pointing each portion of the query to the dataset the user specifies. Regarding the times needed for executing the evaluation queries, all systems performed similarly in the first five queries, being SPARQL-DQP a bit worse than the others but better than RDF::Query which was the worse system in queries SP2BQ3, SP2BQ4 and SP2BQ5. In these queries, both versions of FedX performed bet-

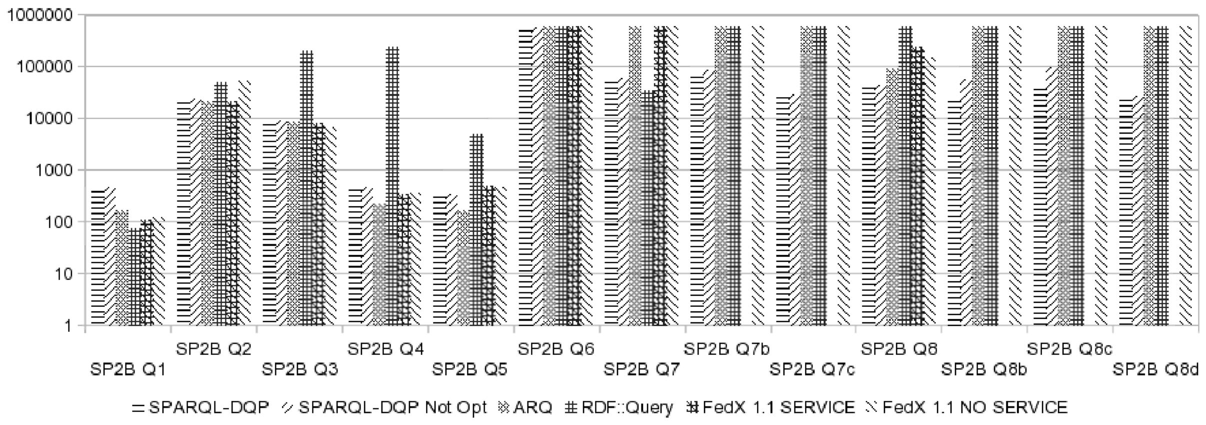

Figure 6: SP2Bench Query Results

ter than the other systems. We think that this is due to the use of its architecture design which parallelizes the execution of the queries and the use of the BIND JOIN technique. In SP2BQ6 none of the systems returned results in reasonable times, not because of the amount of data transferred but because of the time needed for the processing of these data. In the rest of the queries (SP2BQ7, SP2BQ7b, SP2BQ7c, SP2BQ8, SP2BQ8b, SP2BQ8c, SP2BQ8d) only SPARQL-DQP and SPARQL-DQP without optimizations return results in time. The reason for SPARQL-DQP return results in reasonable times is the selection of its base architecture. We extend an architecture designed for working in data intensive scenarios (like [24]), which is based on a data workflow system using a streaming model for transferring the data. In that architecture each query/data processing activity is executed concurrently, in either a remote node in the federation or in the main node in the configuration [13] and the data is also consumed as soon as it is generated.

Regarding the optimizations described in Section 3.1, it is possible to notice their effect specially in queries SP2BQ7b, SP2BQ8b, SP2BQ8c in which rule 2 is applied. Rule 1 is also applied in queries SP2BQ7c and SP2BQ8d, in which it can also be noticed a minor reduction of the execution times.

Looking at the evaluation performed as a whole, it is possible to observe three different sets of results from this evaluation (notice that in Figures 4, 5 and 6 results are represented in logarithmic scale and thus for more accurate results we refer the reader to Appendix B). The first set (standard Fedbench Life Sciences domain, Cross domain and SP2 queries) are those that are not optimized because the reordering rules in Section 3.1 are not applicable. The second query group represents the class of queries that can be optimized using our approach, but where the difference is not too relevant, because the less amount of transferred data (notice that the rules are applied but their execution time is negligible). In this query group we identify query 7 in the Life Sciences domain (LSQ7), Q4 in [45], queries CDQ4b, CDQ8 and CDQ9 in the cross domain query set and queries SP2BQ7c and SP2BQ8d in the SP2Bench evaluation. The last group of queries (queries SP2BQ7b, SP2BQ8c and SP2BQ8d of the SP2bench queries) shows a clear optimization when using the well-designed patterns rewriting rules. These optimizations are better noticed when looking at the result tables in Appendix B. In there query execution times of queries SP2B8b and SP2B8c are reduced in a 50% of its non optimized time. This is even more noticeable when normalizing the time results and using a geometric mean for representing these results, as described in [46]. In our previous work presented in [45] similar results were noticed, specially when always querying remote SPARQL endpoints, since the amount of time for transferring data from several nodes to another will be much higher. In query SP2BQ8b the reduction of intermediate results is done by joining first the SERVICE call to the endpoint containing the collection of scientific papers with the first solution mappings from the SERVICE call to the endpoint containing data about people. The amount of intermediate results is reduced significantly which is specially noted in the execution of the OPTIONAL part of the query, when optionally adding the solution mappings from a SERVICE call to the endpoint containing conference papers.

This evaluation complements the evaluation results from [45] in which we evaluated the system (SPARQL-DQP) and the rewriting rules in a similar way but focusing only in a life science domain. In that evaluation the same result patterns are observed, in which three sets of results are observed, all similar to the ones observed in this work. From that paper we highlight the usefulness of applying the rewriting rules described in Section 3.1: in query 6 in [45] the amount of transferred data varies from a join of $150,000 \times 10,000$ tuples to a join of $10,000 \times 23,841$ tuples (using Entrez, Pubmed and MeSH endpoints), which highly reduced the global processing time of the query.

Regarding the other systems, they all behaved similarly. ARQ and RDF::Query query evaluation times were similar giving the same results as SPARQL-DQP since they implement the same SPARQL specification. They did not return results in the same queries in the SP2B evaluation and performed similarly in the Life Science evaluation, noticing the same problem with the remote server overloads. FedX and FedX with SERVICE also performed similarly to the other systems, but in general FedX was faster than the other systems.

6. Conclusions

In this paper, we first proposed a formal syntax for the SPARQL 1.1 Federated Query extension, along with a formalization of its semantics. In this study, we identified the problems when evaluating the pattern SERVICE $?X$, which requires the variable $?X$ to be bound before the evaluation of the entire pattern. Thus, we proposed syntactic restrictions for assuring the boundedness of SERVICE $?X$, which allows us to safely execute such patterns. We also extended the well-designed patterns definition [8] with the SERVICE operator, which allows to reorder SPARQL queries. This last result is of key importance since it allows to reduce the amount of intermediate results in the query execution. We implemented all these notions in the SPARQL-DQP system and we evaluated it using an existing evaluation framework, which we extended for covering a broader range of common SPARQL queries.

The first conclusion we want to highlight is the importance of using a specific architecture for dealing with large amounts of data. As we have seen in the evaluation section, all systems were not able to process from query 5 in the SP2B query set onwards. All the systems needed more than 10 minutes to answer them while our system, SPARQL-DQP finished in reasonable times. This is due to the architecture chosen, in which the data transfer is done by using streams of data between OGSA-DAI nodes (data is consumed as soon as it is generated) and the data processing is done concurrently in each of these nodes. OGSA-DAI and OGSA-DQP architectures have been highly used in data intensive applications [13, 36] and certainly the Web of data is a data intensive scenario. Thus, the approach to follow should be one that deals with such amounts of data. We also highlight that this architecture is targeted at dealing with SPARQL endpoints with no result limitation, which is not the common case in the current endpoints available to common users. But if a more experienced users want to access unrestricted remote SPARQL endpoints, a more robust approach than the existing ones will be needed. Although the architecture is focused for dealing with large amounts of data, SPARQL-DQP does not perform badly when dealing with restricted SPARQL endpoints.

The next conclusion we want to highlight is the applicability of the rewriting rules presented in Section 3.1. As we have seen in the evaluation section, the application of these rules reduce the execution time when a well-designed pattern of the form $((P_1 \text{ OPT } P_2) \text{ AND } P_3)$ or $((P_1 \text{ FILTER } R) \text{ AND } P_2)$ is present in the SPARQL query. This situation is shown in query SP2BQ8c.rq, in which there are four remote SERVICE calls joined together with an OPTIONAL operator and two join operators. The amount of intermediate results in this query is highly reduced when the first rewriting pattern is applied at the beginning of the query execution. We also highlight that these rewriting rules can be applied to any well-designed pattern, and the cost of applying them is negligible in most of the scenarios. Also, to check whether a SPARQL query is well-designed and to check the safeness condition mentioned previously may produce some overhead, but it is negligible as well, specially when SPARQL queries scale out in the amount of returned results. Besides of this small proof of concept, we refer the reader to [45]. In that work the rewriting rules are more intensively used, and the results highlighted here can also be observed in that work.

Tools for federating queries across the Web of Data are being released frequently. These increase of tools for accessing distributed RDF datasets are only the first step towards a more important goal: to efficiently and effectively query the Web of Data. Currently there are thousands of datasets, and to select to which ones point the SPARQL queries is a complicated task, and part of the research of distributed SPARQL query processing should aim towards solving such great problem.

Focusing more in specific aspects of SPARQL query federation, one of the most common problems we had to deal with was the instability of network connections. Data transfer may be interrupted frequently thus making difficult to query the LOD cloud. We believe that one approach for solving these problems that we experienced is the implementation of adaptive query processing techniques [47] like the ones present in [10]. Also, exploration of the datasets dynamics is an important issue to deal with [48] since data changed during our evaluations. Focusing more in the theoretical aspects of this research work, an interesting contribution would be the analysis of the applicability of the well-designed patterns to SPARQL subqueries. Some work about this research topic has already been carried out [49, 50] but there is still space for improvement.

Acknowledgements

We would like to thank the anonymous reviewers and the editors for their feedback which helped to improve this paper. We would also like to thank the OGSA-DAI team (specially Ally Hume) for their support in extending OGSA-DAI and OGSA-DQP. C. Buil-Aranda and O. Corcho have been funded by the ADMIRE project FP7 ICT-215024, the myBigData Spanish project TIN2010-17060. M. Arenas was supported by FONDECYT grant 1110287. Carlos Buil-Aranda was also supported by the School of Engineering at Pontificia Universidad Católica de Chile.

References

[1] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF (January 2008).

[2] S. Harris, A. Seaborne, SPARQL 1.1 Query Language (January 2012).

[3] K. G. Clark, L. Feigenbaum, E. Torres, SPARQL protocol for RDF, w3C Recommendation, `http://www.w3.org/TR/rdf-sparql-protocol/` (January 2008).

[4] B. Quilitz, U. Leser, Querying distributed rdf data sources with SPARQL, in: ESWC, 2008, pp. 524–538.

[5] S. Schenk, S. Staab, Networked graphs: a declarative mechanism for SPARQL, rules, SPARQL views and rdf data integration on the web., in: WWW, 2008, pp. 585–594.

[6] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization techniques for federated query processing on linked data, in: Proceedings of the 10th International Semantic Web Conference (ISWC), 2011.

[7] E. Prud'hommeaux, C. Buil-Aranda, SPARQL 1.1 Federated Query (November 2011).

[8] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, TODS 34(3).

[9] M. Schmidt, M. Meier, G. Lausen, Foundations of SPARQL query optimization, in: ICDT, 2010, pp. 4–33.

[10] M. Acosta, M. E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, ANAPSID: An adaptive query processing engine for SPARQL endpoints, in: Proceedings of the 10th International Semantic Web Conference (ISWC), 2011.

[11] S. J. Lynden, I. Kojima, A. Matono, Y. Tanimura, ADERIS: An adaptive query processor for joining federated SPARQL endpoints, in: OTM Conferences (2), 2011, pp. 808–817.

[12] M. Jackson, M. Antonioletti, B. Dobrzelecki, N. Hong, Distributed data management with OGSA-DAI, in: S. Fiore, G. Aloisio (Eds.), Grid and Cloud Database Management, Springer Berlin Heidelberg, 2011, pp. 63–86.

[13] B. Dobrzelecki, A. Krause, A. C. Hume, A. Grant, M. Antonioletti, T. Y. Alemu, M. Atkinson, M. Jackson, E. Theocharopoulos, Integrating distributed data sources with OGSA-DAI DQP and views, Philosophical Transactions of the Royal Society A.

[14] B. Glimm, C. Ogbuji, SPARQL 1.1 Entailment Regimes (January 2012).

[15] M. Durst, M. Suignard, Rfc 3987, internationalized resource identifiers (IRIs) (2005).

[16] S. Abiteboul and R. Hull and V. Vianu., Foundations of Databases, Addison-Wesley, 1995.

[17] R. Angles, C. Gutierrez., The expressive power of SPARQL, in: International Semantic Web Conference, 2008, pp. 114–129.

[18] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, M. Papa, K. Vahi, Pegasus and the pulsar search: From metadata to execution on the grid, in: R. Wyrzykowski, J. Dongarra, M. Paprzycki, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, Springer, 2004.

[19] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Research (2006) 729–732.

[20] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T. H. Jordan, E. Deelman, G. Mehta, K. Vahi, D. Gunter, K. Beattie, C. Brooks, Metrics for heterogeneous scientific workflows: A case study of an earthquake science application, Int. J. High Perform. Comput. Appl. 25 (3) (2011) 274–285.

[21] P. Li, J. Castrillo, G. Velarde, I. Wassink, S. Soiland-Reyes, S. Owen, D. Withers, T. Oinn, M. Pocock, C. Goble, S. Oliver, D. Kell, Performing statistical analyses on quantitative data in taverna workflows: an example using r and maxdbrowse to identify differentially-expressed genes from microarray data, BMC Bioinformatics.

[22] A. Nagavaram, G. Agrawal, M. A. Freitas, K. H. Telu, G. Mehta, R. G. Mayani, E. Deelman, A cloud-based dynamic workflow for mass spectrometry data analysis, in: Proceedings of the 2011 IEEE Seventh International Conference on eScience, 2011, pp. 47–54.

[23] W. Tan, R. Madduri, A. Nenadic, S. Soiland-Reyes, D. Sulakhe, I. Foster, C. A. Goble, Cagrid workflow toolkit: A taverna based workflow tool for cancer grid,, BMC Bioinformatics.

[24] J. Bartoka, O. Habalab, P. Bednarc, M. Gazaka, L. Hluchb, Data mining and integration for predicting significant meteorological phenomena, in: International Conference on Computational Science, ICCS 2010.

[25] W. Buehler, O. Dulov, A. Garcia, T. Jejkal, F. Jrad, H. Marten, X. Mol, D. Nilsen, O. Schneider, Reference installation for the german grid initiative d-grid, Journal of Physics: Conference Series.

[26] R. Hull, Managing semantic heterogeneity in databases: a theoretical prospective, in: Proceedings of the sixteenth ACM Symposium on Principles of database systems, 1997, pp. 51–61.

[27] G. Wiederhold, Mediators in the architecture of future information systems, Computer 25 (1992) 38–49.

[28] A. Y. Levy, Answering queries using views: A survey, Tech. rep., VLDB Journal (2001).

[29] J. D. Ullman, Information integration using logical views, in: Proceedings of the 6th International Conference on Database Theory, Springer-Verlag, London, UK, 1997, pp. 19–40.

[30] M. T. Roth, M. Arya, L. Haas, M. Carey, W. Cody, R. Fagin, P. Schwarz, J. Thomas, E. Wimmers, The garlic project, SIGMOD Rec. 25 (2) (1996) 557–.

[31] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom, The tsimmis approach to mediation: Data models and languages, J. Intell. Inf. Syst. 8 (2) (1997) 117–132.
[32] A. Y. Levy, The information manifold approach to data integration, IEEE Intelligent Systems 13 (1998) 12–16.
[33] A. P. Sheth, J. A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, ACM Comput. Surv. 22 (3) (1990) 183–236.
[34] D. Kossmann, The state of the art in distributed query processing, ACM Computing Surveys 32 (2000) 2000.
[35] T. Blanke, M. Hedges, A data research infrastructure for the arts and humanities, in: S. C. Lin, E. Yen (Eds.), Managed Grids and Cloud Systems in the Asia-Pacific Research Community, Springer US, 2010, pp. 179–191.
[36] A. Shaon, A. Woolf, S. Crompton, R. Boczek, W. Rogets, M. Jackson, An open source linked data framework for publishing environmental data under the uk location strategy, in: Terra Cognita workshop in International Semantic Web Conference (ISWC2011).
[37] O. Grlitz, S. Staab, SPLENDID: SPARQL endpoint federation exploiting VOID descriptions, in: COLD 2011 - Consuming Linked Data Workshop.
[38] G. Ladwig, T. Tran, SIHJoin: querying remote and local linked data, in: Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Volume Part I, ESWC'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 139–153.
[39] A. Langegger, Virtual data integration on the web: novel methods for accessing heterogeneous and distributed data with rich semantics, in: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS '08, ACM, New York, NY, USA, 2008, pp. 559–562.
[40] H. Stuckenschmidt, R. Vdovjak, H. Geert-Jan, J. Broekstra., Index structures and algorithms for querying distributed RDF repositories, in: WWW, 2004, pp. 631–639.
[41] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, T. Tran, FedBench: a benchmark suite for federated semantic data query processing, in: Proceedings of the 10th international conference on The semantic web - Volume Part I, ISWC'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 585–600.
[42] M. Schmidt, T. Hornung, G. Lausen, C. Pinkel, SP2Bench: A SPARQL performance benchmark, in: ICDT, 2010, pp. 4–33.
[43] F. Picalausa, S. Vansummeren, What are real SPARQL queries like?, in: Proceedings of the International Workshop on Semantic Web Information Management, SWIM '11, ACM, New York, NY, USA, 2011, pp. 7:1–7:6.
[44] D. Garijo, B. Villazón, O. Corcho, A provenance-aware linked data application for trip management and organization, in: 7th International Conference on Semantic Systems (iSemantics), 2011.
[45] C. Buil-Aranda, M. Arenas, O. Corcho, Semantics and optimization of the SPARQL 1.1 federation extension, in: 8th Extended Semantic Web Conference (ESWC2011), 2011.
[46] P. J. Fleming, J. J. Wallace, How not to lie with statistics: the correct way to summarize benchmark results, Commun. ACM 29 (3) (1986) 218–221.
[47] A. Deshpande, Z. Ives, V. Raman, Adaptive query processing, Foundations and Trends in Databases 1 (1) (2007) 1–140.
[48] J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, S. Decker, Towards dataset dynamics: Change frequency of linked open data sources, in: LDOW, 2010.
[49] R. Angles, C. Gutierrez, SQL nested queries in SPARQL, in: Alberto Mendelzon Workshop (AMW), 2010.
[50] R. Angles, C. Gutierrez, Subqueries in SPARQL, in: Alberto Mendelzon Workshop (AMW), 2011.

Appendix A. Proof of Proposition 1

Let P be a SPARQL query and $?X \in var(P)$. Next we show that if $?X \in \mathrm{SB}(P)$, then $?X$ is bound in P.

The proof is by induction on the structure of P. If P is a triple pattern the proposition trivially holds. Now assume that the proposition holds for patterns P_1 and P_2 and consider the following cases:

- Assume that $P = (P_1 \text{ AND } P_2)$ and $?X \in \mathrm{SB}(P)$. Then we have that $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \mathrm{SB}(P_2)$ and, therefore, $?X \in \mathrm{SB}(P_1)$ or $?X \in \mathrm{SB}(P_2)$. Without loss of generality assume that $?X \in \mathrm{SB}(P_1)$. Now, let DS be a dataset, G an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$. Given that $\mu \in [\![P]\!]_G^{DS}$, we have that $\mu = \mu_1 \cup \mu_2$, where $\mu_1 \in [\![P_1]\!]_G^{DS}$ and $\mu_2 \in [\![P_2]\!]_G^{DS}$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \mathrm{SB}(P_1)$. Hence, given that $\mu_1 \in [\![P_1]\!]_G^{DS}$, we conclude that $?X \in dom(\mu_1)$ and $\mu_1(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Thus, given that $\mu(?X) = \mu_1(?X)$, $dom(\mu_1) \subseteq dom(\mu)$ and $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- Assume that $P = (P_1 \text{ UNION } P_2)$ and $?X \in \mathrm{SB}(P)$. Then we have that $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cap \mathrm{SB}(P_2)$ and, therefore, $?X \in \mathrm{SB}(P_1)$ and $?X \in \mathrm{SB}(P_2)$. Now, let DS be a dataset, G an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$. Given that $\mu \in [\![P]\!]_G^{DS}$, we have that $\mu \in [\![P_1]\!]_G^{DS}$ or $\mu \in [\![P_2]\!]_G^{DS}$. Assume without loss of generality that $\mu \in [\![P_1]\!]_G^{DS}$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \mathrm{SB}(P_1)$. Hence, given that $\mu \in [\![P_1]\!]_G^{DS}$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Thus, given that $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- Assume that $P = (P_1 \text{ OPT } P_2)$ and $?X \in \mathrm{SB}(P)$. Then we have that $\mathrm{SB}(P) = \mathrm{SB}(P_1)$ and, therefore, $?X \in \mathrm{SB}(P_1)$. Now, let DS be a dataset, G

an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$. Given that $\mu \in [\![P]\!]_G^{DS}$, we have that either $\mu = \mu_1 \cup \mu_2$, where $\mu_1 \in [\![P_1]\!]_G^{DS}$ and $\mu_2 \in [\![P_2]\!]_G^{DS}$, or $\mu \in [\![P_1]\!]_G^{DS}$ and μ is not compatible with any mapping in $[\![P_2]\!]_G^{DS}$.

- In the first case, given that $?X$ is bound in P_1 (since $?X \in \mathrm{SB}(P_1)$) and $\mu_1 \in [\![P_1]\!]_G^{DS}$, we have that $?X \in dom(\mu_1)$ and $\mu_1(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Thus, given that $\mu(?X) = \mu_1(?X)$, $dom(\mu_1) \subseteq dom(\mu)$ and $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.
- In the second case, given that $?X$ is bound in P_1 (since $?X \in \mathrm{SB}(P_1)$) and $\mu \in [\![P_1]\!]_G^{DS}$, we have that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Thus, given that $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- Assume that $P = (P_1 \text{ FILTER } R)$, where R is a built-in condition, and that $?X \in \mathrm{SB}(P)$. Then we have that $\mathrm{SB}(P) = \mathrm{SB}(P_1)$ and, therefore, $?X \in \mathrm{SB}(P_1)$. Now, let DS be a dataset, G an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$. Given that $\mu \in [\![P]\!]_G^{DS}$, we have that $\mu \in [\![P_1]\!]_G^{DS}$ and $\mu \models R$. By induction hypothesis, we have that $?X$ is bound in P_1 since $?X \in \mathrm{SB}(P_1)$. Hence, given that $\mu \in [\![P_1]\!]_G^{DS}$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Thus, given that $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- If $P = (\text{GRAPH } a\ P_1)$, where $a \in I$, then we have that $\mathrm{SB}(P) = \emptyset$, and we conclude that the property trivially holds. Thus, assume that $P = (\text{GRAPH } ?Y\ P_1)$ and $?X \in \mathrm{SB}(P)$, and let DS be a dataset, G an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, for which we consider two cases. Notice that in these cases, we assume that $DS = \{(def, G_0), (g_1, G_1), \ldots (g_k, G_k)\}$ with $k \geq 1$, as if we have that $DS = \{(def, G)\}$, then $[\![P]\!]_G^{DS} = \emptyset$, which contradicts the fact that $\mu \in [\![P]\!]_G^{DS}$.

  - Assume that $?X \neq ?Y$. Then we have that there exists $g \in names(DS)$ such that $\mu \in [\![P_1]\!]_{graph(g,DS)}^{DS}$. Moreover, we also have that $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \{?Y\}$, from which we conclude that $?X \in \mathrm{SB}(P_1)$. Thus, we have by induction hypothesis that $?X$ is bound in P_1. Therefore, given that $\mu \in [\![P_1]\!]_{graph(g,DS)}^{DS}$, we have that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P_1))$. Hence, given that $dom(P_1) = dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.
  - Assume that $?X = ?Y$. Then by definition of the semantics of the GRAPH operator, we have that there exists $g \in names(DS)$ such that $\mu(?X) = g$. Therefore, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- Assume that $P = (\text{SERVICE } c\ P_1)$, where $c \in I \cup V$. Then given $\mathrm{SB}(P) = \emptyset$, we conclude that the property trivially holds.

- Assume that $P = (P_1 \text{ BINDINGS } \vec{W}\ \{\vec{A_1}, \ldots, \vec{A_n}\})$ and $?X \in \mathrm{SB}(P)$. Then given that

$$\begin{aligned}\mathrm{SB}(P) \;=\; & \mathrm{SB}(P_1) \cup \\ & \{?Y \mid ?Y \text{ is included in } \vec{W} \text{ and for} \\ & \text{every } i \in \{1, \ldots, n\} : ?Y \in dom(\mu_{\vec{W} \mapsto \vec{A_i}})\},\end{aligned}$$

we conclude that either $?X \in \mathrm{SB}(P_1)$ or $?X$ is included in $\vec{W}$ and for every $i \in \{1, \ldots, n\}$, it holds that $?X \in dom(\mu_{\vec{W} \mapsto \vec{A_i}})$. Now, let DS be a dataset, G an RDF graph in DS and μ a mapping such that $\mu \in [\![P]\!]_G^{DS}$. In order to prove that $?X$ is bound in P, we have to demonstrate that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$. Given that $\mu \in [\![P]\!]_G^{DS}$, we have that there exist $\mu_1 \in [\![P_1]\!]_G^{DS}$ and $k \in \{1, \ldots, n\}$ such that $\mu = \mu_1 \cup \mu_{\vec{W} \mapsto \vec{A_k}}$. Next we consider two cases to prove that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$.

  - First, assume that $?X \in \mathrm{SB}(P_1)$. Then by induction hypothesis we have that $?X$ is bound in P_1. Thus, given that $\mu_1 \in [\![P_1]\!]_G^{DS}$, we conclude that $?X \in dom(\mu_1)$ and $\mu_1(?X) \in$

$(dom(DS) \cup names(DS) \cup dom(P_1))$. Therefore, given that $dom(\mu_1) \subseteq dom(\mu)$, $\mu(?X) = \mu_1(?X)$ and $dom(P_1) \subseteq dom(P)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

- Second, assume that $?X$ is included in $\vec{W}$ and for every $i \in \{1, \ldots, n\}$, it holds that $?X \in dom(\mu_{\vec{W} \mapsto \vec{A}_i})$. Then we have that $?X \in dom(\mu_{\vec{W} \mapsto \vec{A}_k})$, which implies that $\mu(?X) \in dom(P)$ (since $\mu = \mu_1 \cup \mu_{\vec{W} \mapsto \vec{A}_k}$, and if $a \in (I \cup L)$ is mentioned in $\vec{A}_k$, then a is in $dom(P)$). Thus, given that $dom(\mu_{\vec{W} \mapsto \vec{A}_k}) \subseteq dom(\mu)$ and $\mu(?X) = \mu_{\vec{W} \mapsto \vec{A}_k}(?X)$, we conclude that $?X \in dom(\mu)$ and $\mu(?X) \in (dom(DS) \cup names(DS) \cup dom(P))$, which was to be shown.

• Assume that $P = (\text{SELECT } W \ P_1)$ and $?X \in \text{SB}(P)$. Then we have that $\text{SB}(P) = (W \cap \text{SB}(P_1))$ and, therefore, $?X \in W$ and $?X \in \text{SB}(P_1)$. Thus, we conclude by induction hypothesis that $?X$ is bound in P_1. Therefore, we have by definition of boundedness that $?X$ is bound in P, which was to be shown.

Appendix B. Query Result Tables

	CD Q1	CD Q2	CD Q3	CD Q4	CD Q4b	CD Q5	CD Q6	CD Q7	CD Q8	CDQ9
SPARQL-DQP NO OPT	4489	2404,6	11580	10645	14725	925	3206,8	9966,6	15993,4	32345
SPARQL-DQP	2685,2	2429,8	9734,6	12076,4	13717,2	1866,2	872	8607,2	18270,6	31744
ARQ	5657,6	3139,6	20407,6	2147,2	40864	32739,4	2734,8	577,6	118788,4	600000
RDF::Query	2310,6	2119,6	19777	19893,8	1155,2	7854,2	783,2	21129,2	9007,3	600000
FedX SERVICE	2312,5	4329	3682,1	1401	28327,8	2571,1	784	783,6	0	0
FedX NO SERVICE	2258,8	2570,6	4997	3018,2	1388,4	15,6	15	15	43292,6	0

Table B.1: Results of Cross domain queries (Evaluation times in milliseconds)

	LS Q1	LS Q2	LS Q3	LS Q4	LS Q5	LS Q6	LS Q7
SPARQL-DQP NO OPT	9497,8	3000,4	132935,6	35558	10458	13586	12640,8
SPARQL-DQP	5667,6	2320,4	83830,4	14822,4	6011	10052	10425
ARQ	4689,7	3300	47204,9	13325,7	13391,4	5390,9	4479,9
RDF::Query	5100,2	2119,4	89275,4	61663,8	570,4	43575,2	58824,6
FedX SERVICE	6134,2	2391	141365,4	364,2	22011	1662,8	24652
FedX NO SERVICE	16427,4	4817,2	0	15	23858,6	1662,8	24652

Table B.2: Results of life sciences domain queries (Evaluation times in milliseconds)

	SP2B Q1	SP2B Q2	SP2B Q3	SP2B Q4	SP2B Q5	SP2B Q6	SP2B Q7	SP2B Q7b	SP2B Q7c	SP2B Q8	SP2B Q8b	SP2B Q8c	SP2B Q8d
SPARQL-DQP NO OPT	511	25275	9717,8	495,4	358,2	600000	62710,6	92543,6	30066,2	46535,6	61154	100220,4	29533
SPARQL-DQP	476,4	24386	9204	495,4	358,2	600000	60433	77217,6	30471,6	45867,2	25287,6	42877,6	27851,2
ARQ	77	48262,2	196549,8	233497,6	4935,8	600000	33768,8	600000	600000	600000	600000	600000	600000
RDF::Query	165,25	21232,75	8578,25	226,75	169	600000	600000	600000	600000	90751,6	600000	600000	600000
FedX SERVICE	106,2	20949	8296	351,4	490	600000	600000	0	0	239535,75	0	0	0
FedX NO SERVICE	125,8	54224	6699,6	375,8	466	600000	600000	600000	600000	600000	600000	600000	600000

Table B.3: Results of SP2B domain queries (Evaluation times in milliseconds)

Appendix C. New Queries used in the Evaluation

SP2BQ7b

```
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?article ?inproc ?person ?name
WHERE {
 SERVICE <http://localhost:3030/Journal/sparql> {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
 }
 OPTIONAL {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
   ?inproc rdf:type bench:/Inproceedings> .
   ?inproc dc:creator ?person .
  }
 }
 SERVICE <http://localhost:3030/People/sparql> {
  ?person foaf:name ?name .
 }
}
```

SP2BQ7c
```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?article ?inproc ?person ?name ?title ?pages
WHERE {
 SERVICE <http://localhost:3030/Journal/sparql> {
  ?article rdf:type <http://localhost/vocabulary/bench/Article> .
  ?article dc:creator ?person .
  ?article swrc:pages ?pages
 }
 OPTIONAL {
  SERVICE <http://localhost:3030/People/sparql> {
   ?person foaf:name ?name .
  }
 }
 FILTER (?pages = 200)
```

SP2BQ8b
```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX bench: <http://localhost/vocabulary/bench/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?person ?name ?incol ?inproc
WHERE {
 SERVICE <http://localhost:3030/People/sparql> {
  ?person foaf:name ?name
 }
 OPTIONAL {
  SERVICE <http://localhost:3030/InProceedings/sparql> {
   ?inproc rdf:type bench:Inproceedings .
   ?inproc dc:creator ?person .
  }
 }
 SERVICE <http://localhost:3030/InCol/sparql> {
  ?incol rdf:type bench:Incollection .
  ?incol dc:creator ?person .
 }
```

SP2BQ8c
```
PREFIX dcInproc: <http://purl.org/dc/elements/1.1/Inproc/>
PREFIX dcJournal: <http://purl.org/dc/elements/1.1/Journal/>
PREFIX dcIncol: <http://purl.org/dc/elements/1.1/Incol/>
PREFIX benchInproc: <http://localhost/vocabulary/bench/Inproc/>
PREFIX benchIncol: <http://localhost/vocabulary/bench/Incol/>
PREFIX benchJournal: <http://localhost/vocabulary/bench/Journal/>
PREFIX dctermsInproc: <http://purl.org/dc/terms/Inproc/>
PREFIX rdfInproc:  <http://www.example.org/rdf/type/rdfinproc#>
PREFIX rdfIncol:  <http://www.example.org/rdf/type/rdfincol#>
PREFIX rdfJournal:  <http://www.example.org/rdf/type/rdfjournal#>
PREFIX foafPeople: <http://xmlns.com/foaf/0.1/People/>
PREFIX foafInproc: <http://xmlns.com/foaf/0.1/Inpror
PREFIX swrcInproc: <http://inproc.swrc.ontoware.org/ontology#>
PREFIX rdfsInproc: <http://www.w3.org/2000/01/inproc-rdf-schema#>
SELECT DISTINCT ?person  ?incol ?inproc ?title ?article
WHERE {
 SERVICE <http://localhost:3030/Journal/sparql> {
   ?article rdfJournal:type benchJournal:Article .
   ?article dcJournal:creator ?person .
  }
 SERVICE <http://localhost:3030/InProceedings/sparql> {
 ?inproc rdfInproc:type benchInproc:Inproceedings .
      ?inproc dcInproc:creator ?person .
      ?inproc benchInproc:booktitle ?booktitle
      OPTIONAL {
        ?inproc benchInproc:abstract ?abstract
    }
 }
 OPTIONAL {
  SERVICE <http://localhost:3030/People/sparql> {
    ?person foafPeople:name ?name
   }
 }
 SERVICE <http://localhost:3030/InCol/sparql> {
   ?incol rdfIncol:type benchIncol:Incollection .
  ?incol dcIncol:creator ?person .
 }
}
```

SP2BQ8d

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT DISTINCT ?title ?name
WHERE {
 SERVICE <http://localhost:3030/InProceedings/sparql> {
  ?inproc rdf:type <http://localhost/vocabulary/bench/Inproceedings> .
  ?inproc dc:creator ?person .
  ?inproc dc:title ?title .
  ?inproc dcterms:issued ?yr
 }
 OPTIONAL {
  SERVICE <http://localhost:3030/People/sparql> {
   ?person foaf:name ?name
  }
 }
 FILTER (?yr="1950")
```

CDQ8

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX viajero: <http://webenemasuno.linkeddata.es/ontology/OPMO/>
SELECT ?Book ?Country ?Area
WHERE {
 SERVICE <http://dbpedia.org/sparql> {
  ?Country rdf:type <http://dbpedia.org/ontology/Country> .
  ?Country dbprop:areaKm ?Area
 }
 OPTIONAL {
  SERVICE <http://webenemasuno.linkeddata.es/sparql> {
   ?Book viajero:refersTo ?Country . }
 }
 FILTER(?Area < "20000"^^xsd:integer)
```

CDQ9

```
PREFIX rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX viajero: <http://webenemasuno.linkeddata.es/ontology/OPMO/>
PREFIX factbook: <http://www4.wiwiss.fu-berlin.de/factbook/ns#>

SELECT ?Book ?Country ?Area ?climate
WHERE {
 SERVICE <http://dbpedia.org/sparql> {
  ?Country rdf:type <http://dbpedia.org/ontology/Country> .
  ?Country dbprop:areaKm ?Area .
  ?Country rdfs:label ?countryLabel
 }
 OPTIONAL {
  SERVICE <http://webenemasuno.linkeddata.es/sparql> {
   ?Book viajero:refersTo ?Country .
  }
 }
 SERVICE <http://www4.wiwiss.fu-berlin.de/factbook/sparql> {
  ?CountryCIA rdfs:label ?countryLabel .
  ?CountryCIA rdf:type  <http://www4.wiwiss.fu-berlin.de/factbook/ns#Country> .
  ?CountryCIA factbook:climate ?climate
 }
```