FedUP: Querying Large-Scale Federations of SPARQL Endpoints

Julien Aimonier-Davat, Minh-Hoang Dang, Pascal Molli, Brice Nédelec, Hala Skaf-Molli

► **To cite this version:**

Julien Aimonier-Davat, Minh-Hoang Dang, Pascal Molli, Brice Nédelec, Hala Skaf-Molli. FedUP: Querying Large-Scale Federations of SPARQL Endpoints. The ACM Web Conference 2024 (WWW '24), May 2024, Singapore, Singapore. 10.1145/3589334.3645704 . hal-04538238

HAL Id: hal-04538238

https://hal.science/hal-04538238v1

Submitted on 9 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FedUP: Querying Large-Scale Federations of SPARQL Endpoints

Julien Aimonier-Davat
julien.aimonier-davat@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Minh-Hoang Dang
minh-hoang.dang@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Pascal Molli
pascal.molli@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Brice Nédelec
brice.nedelec@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Hala Skaf-Molli
hala.skaf@ls2n.fr
Nantes Université, CNRS, LS2N
F-44000, Nantes, France

Abstract

Processing SPARQL queries over large federations of SPARQL endpoints is crucial for keeping the Semantic Web decentralized. Despite the existence of hundreds of SPARQL endpoints, current federation engines only scale to dozens. One major issue comes from the current definition of the source selection problem, i.e., finding the minimal set of SPARQL endpoints to contact per triple pattern. Even if such a source selection is minimal, only a few combinations of sources may return results. Consequently, most of the query processing time is wasted evaluating combinations that return no results. In this paper, we introduce the concept of Result-Aware query plans. This concept ensures that every subquery of the query plan effectively contributes to the result of the query. To compute a Result-Aware query plan, we propose FedUP, a new federation engine able to produce Result-Aware query plans by tracking the provenance of query results. However, getting query results requires computing source selection, and computing source selection requires query results. To break this vicious cycle, FedUP computes results and provenances on tiny quotient summaries of federations at the cost of source selection accuracy. Experimental results on federated benchmarks demonstrate that FedUP outperforms state-of-the-art federation engines by orders of magnitude in the context of large-scale federations.

1 INTRODUCTION

Context and motivation Processing SPARQL queries over large federations of SPARQL endpoints is crucial for keeping the Semantic Web decentralized. Despite the existence of hundreds of SPARQL endpoints [14, 24], current federation engines [6, 20, 21, 23] only scale to dozens [9]. This is a severe issue for developing an effective, usable, and decentralized Semantic Web based on federation engines and federations of SPARQL endpoints.

Related work and problem Federated query processing has 3 conceptual steps [2]: (i) source selection and query decomposition, (ii) query optimization, and (iii) query execution.

One major issue comes from the current definition of the *source selection problem*, i.e., finding the minimal set of SPARQL endpoints to contact per triple pattern [20]. Even if such a source selection is minimal, only a few combinations of sources may return results. Consequently, most of the query processing time is wasted evaluating combinations that return no results. To illustrate, Figure 1 presents the query $q05$ of the FedShop benchmark [9], along with the optimal set of sources to contact per triple pattern [20]. First, triple patterns that share the same single data source are merged into exclusive groups [23], e.g., $tp7 - tp11$ are grouped together to be executed on $v3$.

```
SELECT DISTINCT ?product ?localProdLabel WHERE {
 ?lProd rdfs:label ?lProdLabel .                            #tp1 @ rs6, rs0
 ?lProd bsbm:productFeature ?lProdFeature .                 #tp2 @ rs6, rs0
 ?lProd bsbm:productPropertyNumeric1 ?simProperty1 .        #tp3 @ rs6, rs0
 ?lProd bsbm:productPropertyNumeric2 ?simProperty2 .        #tp4 @ rs6, rs0
 ?lProd owl:sameAs ?product .                               #tp5 @ rs6, rs0
 ?lProdFeature owl:sameAs ?prodFeature .                    #tp6 @ rs6, rs0
 ?lProdXYZ bsbm:productFeature ?lProdFeatureXYZ .           #tp7 @  v3,  v3
 ?lProdXYZ bsbm:productPropertyNumeric1 ?origProperty1 .    #tp8 @  v3,  v3
 ?lProdXYZ bsbm:productPropertyNumeric2 ?origProperty2 .    #tp9 @  v3,  v3
 ?lProdXYZ owl:sameAs bsbm:Product136030 .                  #tp10@  v3,  v3
 ?lProdFeatureXYZ owl:sameAs ?prodFeature .                 #tp11@  v3,  v3
 FILTER (...)} ORDER BY ?lProdLabel LIMIT 5
```

Figure 1: Cross Domain Query $q05$ of FedShop [9] along with its optimal source selection over a federation of 20 shops.

Then, the objective of the optimizer is to generate an execution plan that minimizes the number of intermediate results and the communication costs. Thanks to heuristics and/or statistics, it can decide a particular join order and physical operators. In order to avoid huge data transfer of general predicates such as the `sameAs` predicate in $tp5$ and $tp6$, the query optimizer may decide to use a BoundJoin [23].

Finally, during query execution, a physical query plan for $q05$ that is based on the relevant sources per triple pattern and

©2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore, https://doi.org/10.1145/3589334.3645704.

Table 1: Execution times of $q05$ using FedShop's reference, a state-of-the-art federation engine, and our proposal FedUP.

	20 shops	200 shops
FedShop's reference RSA [9]	$50ms$	$1.5s$
CostFed [21]	$2.45s$	$> 1h$
Our proposal (FedUP)	**244ms**	**12.4s**

BoundJoin operators will check every 64 combinations of sources, while only 2 combinations effectively return results:

(1) $[rs6, rs6, rs6, rs6, rs6, rs6, v3, v3, v3, v3, v3]$;

(2) $[rs0, rs0, rs0, rs0, rs0, rs0, v3, v3, v3, v3, v3]$.

As the number of useless combinations increases with federation size, state-of-the-art federation engines suffer from serious performance issues as reported in Table 1. While FedShop empirically demonstrates that an engine could evaluate $q05$ in less than $2s$ (RSA), CostFed, the best federation engine on FedShop, needs $3s$ to finish evaluating $q05$ with 20 sources and more than one hour with 200 sources[1]. Our proposal, FedUP, processes $q05$ in $12.4s$, even on the federation comprising 200 endpoints.

Approach and contributions In this paper, we introduce the concept of Result-Aware query plans. It ensures that every subquery of the query plan effectively contributes to the results of the query. We propose FedUP, a new federation engine that builds such plans by tracking the provenance of query results. However, getting query results requires computing source selection, while computing source selection requires query results. To break this vicious cycle, FedUP computes results and provenances on tiny quotient summaries [4] of federations, but at the cost of source selection accuracy. The contributions of this paper are the following:

- We define and formalize the concept of Result-Aware query plans. Any federation query optimizer can safely use such a query plan. The overall idea is to normalize the logical plan and prune subexpressions that do not contribute to the final results of the query.
- We describe an algorithm that computes Result-Aware query plans while avoiding the combinatorial explosion of normalizing logical plans. The proof of its correctness is detailed in the appendix.
- We present the computation of quotient summaries and detail the effective derivation of Result-Aware query plans through the execution of our algorithm on these summaries.
- We evaluate FedUP on LargeRDFBench [22] and FedShop [9]. Our experiments empirically demonstrate that: (i) FedUP is on par with state-of-the-art federation engines [21, 23] on small federations. (ii) FedUP drastically outperforms state-of-the-art federation engines on large federations of SPARQL endpoints.

[1] We stopped the execution after 1 hour.

This paper is organized as follows: Section 2 presents the background and motivations. Section 3 defines the Result-Aware source selection problem and presents our solution to this problem. Section 4 presents our experimental results conducted on federations of SPARQL endpoints. Section 5 reviews related work about federation engines. Section 6 concludes and outlines future work.

2 BACKGROUND AND MOTIVATIONS

We assume that the reader is familiar with the concepts of RDF and core SPARQL [12, 17], i.e., triple patterns (tp), basic graph patterns (BGP), `AND`, `UNION`, `FILTER`, and `OPTIONAL` graph patterns.

Definition 1 (SPARQL Federation [7, 13]). A SPARQL federation F is a set of federation members (G, I_{sparql}) where G is an RDF graph and I_{sparql} is a SPARQL endpoint interface to access G.

Definition 2 (Federated Query Evaluation [7]). The evaluation $[\![Q]\!]_F$ of a federated query Q over a federation F is a set of solutions mappings defined as $[\![Q]\!]_F = [\![Q]\!]_{G_{union}}$ where $G_{union} = \bigcup_{(G, I) \in F} G$.

Example 1 (Query $S6$ over Federation F_1). Consider the federation F_1 in Figure 2 and the query $S6$ from FedBench [22]. $S6$ returns the names of artists located near the Federal Republic of Germany:

```
SELECT * WHERE {
    ?artist foaf:name ?name .                                    #tp1 @ D1, D3
    ?artist foaf:based_near ?location .                          #tp2 @ D1, D3
    ?location geo:parentFeature ?germany .                       #tp3 @ D2, D4
    ?germany geo:name "Federal Republic of Germany" . } #tp4 @ D2, D4
```

The evaluation of $S6$ over F_1 returns 2 solutions mappings:

	?artist	?name	?location	?germany
μ_1	http://D1/Scorpions	Scorpions	http://D2/Hanover	http://D2/Germany
μ_2	http://D1/Kraftwerk	Kraftwerk	http://D4/Berlin	http://D4/Germany

To retrieve such results, one major challenge for federation engines consists in solving the *source selection problem*, i.e., finding the minimal set of federation members to contact for each triple pattern of the query.

Problem 1 (BGP-Aware Source Selection [20]). Let Q be a SPARQL query and F a federation, find the minimal set of federation members $R(tp) \subseteq F$ for each triple pattern $tp \in Q$ where $\forall (G, I) \in R(tp), \exists \mu \in [\![Q]\!]_F$ such that $\mu(tp) \in G$.

The source selection result can be represented as a FedQPL expression [7, 8]. FedQPL is a language to represent logical query plans over heterogeneous federations.

Definition 3 (FedQPL expression [7, 8]). A FedQPL expression is an expression φ that can be constructed from the following grammar, in which *req*, *filter*, *mj*, *mu*, and *leftjoin* are terminal symbols, tp is a triple pattern, f is a federation member, R is a SPARQL filter condition, and Φ is a non-empty set of FedQPL expressions.

$$\varphi ::= req_f^{tp} \mid filter^R(\varphi) \mid mu\ \Phi \mid mj\ \Phi \mid leftjoin(\varphi, \varphi)$$

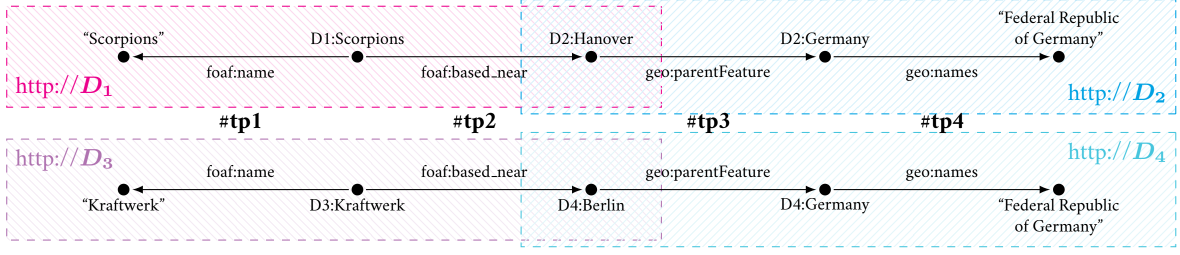
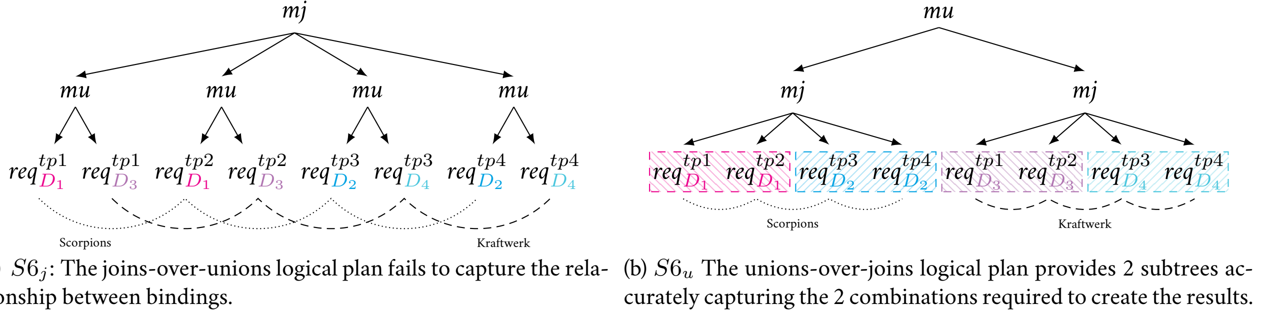


Figure 2: Federation F_1 with 4 endpoints storing information about Scorpions and Kraftwerk, 2 bands from Germany.



(a) $S6_j$: The joins-over-unions logical plan fails to capture the relationship between bindings.

(b) $S6_u$ The unions-over-joins logical plan provides 2 subtrees accurately capturing the 2 combinations required to create the results.

Figure 3: Logical plans for Query S6 over the federation F expressed in FedQPL Language.

Definition 4 (FedQPL semantics [7, 8]). Let φ be a FedQPL expression, the solutions mappings obtained with φ, denoted by $sols(\varphi)$, is a set of solutions mappings that is defined recursively as follows:

(1) If φ is of the form req_f^{tp}, then $sols(\varphi) = [\![tp]\!]_f$

(2) If φ is of the form $filter^R(\varphi')$, then $sols(\varphi') = \{\mu \mid \mu \in sols(\varphi') \wedge \mu \vDash R\}$

(3) If φ is of the form $mj\ \Phi$ where $\Phi = \{\varphi_1, \cdots, \varphi_n\}$, then $sols(\varphi) = sols(\varphi_1) \bowtie \cdots \bowtie sols(\varphi_n)$

(4) If φ is of the form $mu\ \Phi$ where $\Phi = \{\varphi_1, \cdots, \varphi_n\}$, then $sols(\varphi) = sols(\varphi_1) \cup \cdots \cup sols(\varphi_n)$

(5) If φ is of the form $leftjoin(\varphi_1, \varphi_2)$, then $sols(\varphi) = sols(\varphi_1) ⟕ sols(\varphi_2)$

Example 2 (Joins-over-unions logical plans). The minimal source selection of the query $S6$ over the federation F_1 is $R(tp1) = \{D_1, D_3\}$, $R(tp2) = \{D_1, D_3\}$, $R(tp3) = \{D_2, D_4\}$, and $R(tp4) = \{D_2, D_4\}$. By default, such a source selection is represented as a joins-over-unions FedQPL expression as depicted in Figure 3a:

$$S6_j = mj\ \{mu\ \{req_{D_1}^{tp1}, req_{D_3}^{tp1}\}, mu\ \{req_{D_1}^{tp2}, req_{D_3}^{tp2}\}, mu\ \{req_{D_3}^{tp3}, req_{D_2}^{tp3}\}, mu\ \{req_{D_2}^{tp4}, req_{D_4}^{tp4}\}\}$$

To evaluate $S6_j$, federation engines generate at least as many `SERVICE` queries as the number of *req* in $S6_j$. Gathering *req* in exclusive groups constitutes a major performance improvement as it lowers the number of `SERVICE` queries, pushing more computation on SPARQL endpoints. However, federation engines cannot apply such an optimization on $S6_j$.

The main issue with the current definition of source selection is that important information is missing. Based on the results of $S6$, the evaluation of the query $S6$ only requires two series of joins: $\{tp1 \rightarrow D_1, tp2 \rightarrow D_1, tp3 \rightarrow D_2, tp4 \rightarrow D_2\}$ for μ_1, and $\{tp1 \rightarrow D_3, tp2 \rightarrow D_3, tp3 \rightarrow D_4, tp4 \rightarrow D_4\}$ for μ_2. However, with unions (*mu*) under joins (*mj*), this information is hidden from the query optimizer, preventing it from considering other options, and potentially finding better plans. All existing federation engines generate such joins-over-unions plans [7]. By design, they remain blind to many optimizations that their counterpart, unions-over-joins, can perform.

Example 3 (Unions-over-joins logical plans). Using the results of $S6$ over F_1, an alternative to $S6_j$ is the unions-over-joins FedQPL expression $S6_u$ depicted in Figure 3b:

$$S6_u = mu\ \{mj\ \{req_{D_1}^{tp1}, req_{D_1}^{tp2}, req_{D_2}^{tp3}, req_{D_2}^{tp4}\}, mj\ \{req_{D_3}^{tp1}, req_{D_3}^{tp2}, req_{D_4}^{tp3}, req_{D_4}^{tp4}\}\}$$

Using $S6_u$, federation engines only evaluate joins that actually contribute to the final results of $S6$. Moreover, $S6_u$ allows federation engines to identify more exclusive groups than $S6_j$. For example, triple patterns $tp1$ and $tp2$ are grouped together, as well as $tp3$ and $tp4$:

$$S6'_u = mu\ \{mj\ \{req_{D_1}^{tp1,tp2}, req_{D_2}^{tp3,tp4}\}, mj\ \{req_{D_3}^{tp1,tp2}, req_{D_4}^{tp3,tp4}\}\}$$

In summary, the current source selection definition hides important information about which sources should be combined to find results. With just a set of relevant sources per triple pattern, it is impossible to know which combinations of sources contribute to the final query results. Consequently, many valuable query plans such as $S6_u$ remain invisible to the query optimizer. This problem is at the origin of the poor performance of current federation engines on the FedShop benchmark [9]. Solving this problem requires defining a new kind of source selection able to reveal the relevant combination of sources.

3 FEDUP: A RESULT-AWARE QUERY ENGINE

In this section, we introduce our approach to build a Result-Aware federation engine. We consider a federation F and a core SPARQL query Q composed of BGP patterns with `UNION`, `FILTER`, `OPTIONAL`. In the following, we rely on set-based semantics of SPARQL [17].

As stated in the previous section, the existing source selection does not reveal which combinations of relevant sources effectively produce results. Without this information, some classes of query plans cannot be explored, such as unions-over-joins query plans. A first step to generate a unions-over-joins query plan is to rewrite φ using equivalence rules.

Definition 5 (Equivalence rules [7, 17])**.** Let φ_1, φ_2, and φ_3 be FedQPL expressions that are valid for F. It holds that[2]:

(R1) $\textit{join}(\varphi_1, \varphi_2) \overset{F}{\equiv} \textit{join}(\varphi_2, \varphi_1)$;
(R2) $\textit{union}(\varphi_1, \varphi_2) \overset{F}{\equiv} \textit{union}(\varphi_2, \varphi_1)$;
(R3) $\textit{union}(\varphi_1, \varphi_1) \overset{F}{\equiv} \varphi_1$;
(R4) $\textit{join}(\varphi_1, \textit{join}(\varphi_2, \varphi_3)) \overset{F}{\equiv} \textit{join}(\textit{join}(\varphi_1, \varphi_2), \varphi_3)$;
(R5) $\textit{union}(\varphi_1, \textit{union}(\varphi_2, \varphi_3)) \overset{F}{\equiv} \textit{union}(\textit{union}(\varphi_1, \varphi_2), \varphi_3)$;
(R6) $\textit{join}(\varphi_1, \textit{union}(\varphi_2, \varphi_3)) \overset{F}{\equiv} \textit{union}(\textit{join}(\varphi_1, \varphi_2), \textit{join}(\varphi_1, \varphi_3))$;
(R7) $\textit{leftjoin}(\textit{union}(\varphi_1, \varphi_2), \varphi_3) \overset{F}{\equiv} \textit{union}(\textit{leftjoin}(\varphi_1, \varphi_3), \textit{leftjoin}(\varphi_2, \varphi_3))$.

To illustrate, by applying the equivalence rules $[R1 - R7]$ to $S6_j$ of Example 2, we generate a unions-over-joins query $S6'_j$:

$$
\begin{aligned}
S6'_j = \textit{mu}\ \{&\textit{mj}\ \{\textit{req}_{D_1}^{tp1}, \textit{req}_{D_1}^{tp2}, \textit{req}_{D_2}^{tp3}, \textit{req}_{D_2}^{tp4}\},\\
&\ldots \times 14\\
&\textit{mj}\ \{\textit{req}_{D_3}^{tp1}, \textit{req}_{D_3}^{tp2}, \textit{req}_{D_4}^{tp3}, \textit{req}_{D_4}^{tp4}\}\}
\end{aligned}
$$

However, only the first and last subexpressions contribute to the results. When we remove the 14 useless subexpressions that return empty results, we obtain the unions-over-joins query plan $S6_u$ of Example 3 with only subexpressions contributing to the final results of S_6. Intuitively, we define the *Result-Aware source selection problem* as the problem of finding a unions-over-joins expression where every subexpression, i.e., expression under the root multi-union operator, returns results. In that regard, both $S6'_j$ and $S6_u$ are unions-over-joins plans but $S6_u$ is Result-Aware while $S6'_j$ is not.

Problem 2 (Result-Aware source selection)**.** Given a SPARQL query Q and a federation F, find a FedQPL expression φ such that φ is normalized (e.g., using [R1-R7] equivalences), and φ is Result-Aware:

$$\forall \varphi' \subseteq \varphi, \exists \mu' \in \textit{sols}(\varphi') \text{ such that } \mu' \subseteq \mu \text{ where } \mu \in [\![Q]\!]_F$$

If we only consider SPARQL queries based on conjunctive and disjunctive queries, the normal form of φ follows a unions-over-joins grammar $S_{\cup(\bowtie)}$ defined as $S_{\cup(\bowtie)} = \textit{mu}\ \{(\textit{mj}\ \{(\textit{req}_D^{tp})^+\})^+\}$.

[2] $\textit{leftjoin}(\varphi_1, \textit{union}(\varphi_2, \varphi_3)) \overset{F}{\equiv} \textit{union}(\textit{leftjoin}(\varphi_1, \varphi_2), \textit{leftjoin}(\varphi_1, \varphi_3))$ does not hold. See counter example in Appendix A.1.

In the presence of `OPTIONAL` clauses, we extend the grammar of the unions-over-joins class $S_{\cup(\bowtie)}$ to include the *leftjoin* operator:

$$
\begin{cases}
a_u ::= & a_j \mid \textit{mu}\ \Phi_u\\
a_j ::= & a_b \mid \textit{mj}\ \Phi_b \mid \textit{leftjoin}(a_j, a_u)\\
a_b ::= & \textit{req}_D^{tp}
\end{cases}
$$

As a Result-Aware source selection is normalized, then pruned of its useless subexpressions, some subexpressions may appear several times in φ. We assume that the factorization of such duplicated subexpressions is a task that falls to query optimizers.

3.1 Providing Result-Aware Source Selection

The normalize-then-prune algorithm to implement Result-Aware source selection introduces combinatorial explosions as illustrated by $S6'_j$. To alleviate this issue, we propose an algorithm that iteratively builds relevant subexpressions instead of pruning useless ones.

Algorithm 1: Result-Aware source selection $\mathcal{A}$ for a SPARQL query Q over a federation F.

1 **Function** $\mathcal{A}(Q, F)$: ▷ Root of the logical plan
2   **return** $\textit{mu}\ \mathcal{A}'(Q, F)$
3 **Function** $\mathcal{A}'(Q, F)$: ▷ Explores every graph pattern Q
4     $\Phi_o \leftarrow \varnothing$
5   **if** Q is a triple pattern tp **then**
6         $\Phi_o \leftarrow \Phi_o \cup \{\textit{req}_f^{tp} \mid f \in F\}$
7   **else if** Q is (P_1 `AND` P_2) **then** ▷ $P_1 \bowtie P_2$
8         $\Phi_1, \Phi_2 \leftarrow \mathcal{A}'(P_1, F), \mathcal{A}'(P_2, F)$
9         $\Phi_o \leftarrow \Phi_o \cup \{\textit{mj}\ \{\varphi_1, \varphi_2\} \mid \varphi_1 \in \Phi_1 \wedge \varphi_2 \in \Phi_2\}$
10   **else if** Q is (P_1 `UNION` P_2) **then** ▷ $P_1 \cup P_2$
11         $\Phi_1, \Phi_2 \leftarrow \mathcal{A}'(P_1, F), \mathcal{A}'(P_2, F)$
12         $\Phi_o \leftarrow \Phi_o \cup \{\varphi \mid \varphi \in \Phi_1 \vee \varphi \in \Phi_2\}$
13   **else if** Q is (P_1 `OPTIONAL` P_2) **then** ▷ $P_1 ⟕ P_2$
14         $\Phi_1, \Phi_2 \leftarrow \mathcal{A}'(P_1, F), \mathcal{A}'(P_2, F)$
15   **for** $\varphi_1 \in \Phi_1$ **do**
16             $\Phi_{\textit{join}}^{\varphi_1} \leftarrow \{\varphi_2 \mid \varphi_2 \in \Phi_2 \wedge \textit{sols}(\textit{mj}\{\varphi_1, \varphi_2\}) \neq \varnothing\}$
17   **if** $\Phi_{\textit{join}}^{\varphi_1} = \varnothing$ **then** $\Phi_o \leftarrow \Phi_o \cup \{\varphi_1\}$
18   **else** $\Phi_o \leftarrow \Phi_o \cup \{\textit{leftjoin}(\varphi_1, \textit{mu}\ \Phi_{\textit{join}}^{\varphi_1})\}$
19   **else if** Q is (P `FILTER` R) **then**
20         $\Phi \leftarrow \mathcal{A}'(P, F)$
21         $\Phi_o \leftarrow \Phi_o \cup \{\textit{filter}^R(\varphi) \mid \varphi \in \Phi\}$
22   **return** $\{\varphi \mid \varphi \in \Phi_o \wedge \textit{sols}(\varphi) \neq \varnothing\}$

Algorithm 1 builds a Result-Aware source selection plan for a query Q over a federation F based on a set of recursive rules. The algorithm is designed with 2 main ideas:

1. Build a unions-over-joins logical plan;
2. Keep only expressions that contribute to the final results of the query.

To reach this objective, the algorithm evaluates the query on the federation, extracts the provenance of solutions mappings, and produces a Result-Aware FedQPL expression. To ensure that every produced φ expression is based on results, it relies on $\textit{sols}(\varphi)$ as a function that returns the mappings resulting in the evaluation of the expression φ over F. Proofs of correctness, completeness, and result-awareness are available in Appendix A.2.

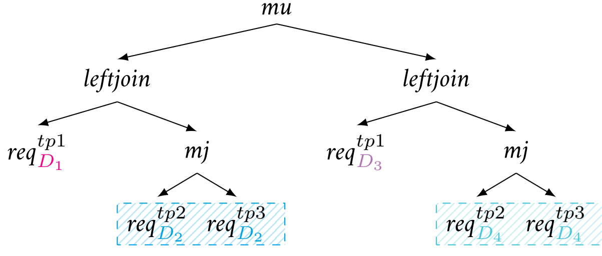


Figure 4: Logical plan for Query $S7$ with OPTIONAL.

We illustrate this algorithm on Query $S6$ of Example 2 over the federation F_1 of Figure 2. First, the algorithm merges all upcoming subexpressions with a multi-union at Line 1. Then, it enters Line 7 with ($tp1$ AND ($tp2$ AND ($tp3$ AND $tp4$))). Line 3 states that evaluating $tp3$ and $tp4$ both returns $\{req^{tp}_{D_2}, req^{tp}_{D_4}\}$. Then, Line 22 checks that their intersections return mappings. Here, $mj\ \{req^{tp3}_{D_2}, req^{tp4}_{D_2}\}$ and $mj\ \{req^{tp3}_{D_4}, req^{tp4}_{D_4}\}$ indeed return mappings, but most importantly:

$$mj\ \{req^{tp3}_{D_2}, req^{tp4}_{D_4}\} = \varnothing$$
$$mj\ \{req^{tp3}_{D_4}, req^{tp4}_{D_2}\} = \varnothing$$

Only the former expressions are kept, the latter ones are discarded. After applying the joining rule for every triple pattern and simplifying nested multi-join expressions, we obtain Figure 3b's expected plan:

$$mu\ \{mj\ \{req^{tp1}_{D_1}, req^{tp2}_{D_1}, req^{tp3}_{D_2}, req^{tp4}_{D_2}\},$$
$$mj\ \{req^{tp1}_{D_3}, req^{tp2}_{D_3}, req^{tp3}_{D_4}, req^{tp4}_{D_4}\}\}$$

The 4 exclusive groups are easily identified: tp1.tp2 at D_1 and D_3, tp3.tp4 at D_2 and D_4.

Example 4 (Optional Query $S7$ over Federation F_1). To illustrate Result-Aware query plan in the presence of OPTIONAL, we consider the query $S7$:

```
SELECT * WHERE {
  ?artist foaf:based_near ?location .                        #tp1
  OPTIONAL {
    ?location geo:parentFeature ?germany .                   #tp2
    ?germany geo:name "Federal Republic of Germany" }} #tp3
```

For such a query $S7$, and as depicted in Figure 4, Algorithm 1 produces a multi-union of two left joins:

$$S7_u = mu\ \{leftjoin(req^{tp1}_{D_1}, mu\ \{mj\ \{req^{tp2}_{D_2}, req^{tp3}_{D_2}\}\}),$$
$$leftjoin(req^{tp1}_{D_3}, mu\ \{mj\ \{req^{tp2}_{D_4}, req^{tp3}_{D_4}\}\})\}$$

Line 16 ensures that expressions representing the OPTIONAL clauses are Result-Aware. The evaluation of $S7_u$ on F_1 returns the expected results.

3.2 FedUP on Summaries

FedUP introduces a vicious cycle: its source selection requires query results, and query results require computing source selection. To tackle this issue, we execute Algorithm 1 and $sols(\varphi)$ on a tiny quotient summary [4, 5] of the federation.

Definition 6 (Quotient RDF summary [5]). Given an RDF graph G and an RDF node equivalence relation ψ, the summary of G by ψ, which is an RDF graph denoted $\psi(G)$, is the quotient of G by ψ.

Quotient summaries have many interesting properties that are relevant in the context of the source selection problem. First, queries that have answers on F also have answers on $\psi(F)$, enabling FedUP to ensure complete results. Abusing notation, $\psi(F)$ is the quotient summary of F, i.e., the federation obtained by replacing all RDF graphs G in F by the quotient summary of G. Second, quotient summaries are RDF graphs. The source selection algorithm is the same whether it is executed over the federation or a quotient summary of the federation. Finally, quotient summaries preserve edges in graphs, increasing source selection accuracy compared to other summaries [20, 21].

Definition 7 (Summary representativeness [5]). Given a SPARQL query Q, a federation F, and an RDF node equivalence relation ψ, if $[\![Q]\!]_F \neq \varnothing$ then we have $[\![\psi(Q)]\!]_{\psi(F)} \neq \varnothing$.

FedUP uses ψ_h as the RDF node equivalence relation to summarize SPARQL federations. It is defined as follows:

$$\psi_h(node) = \begin{cases} authority(node) & \text{if } node \text{ is an IRI} \\ \text{“any”} & \text{if } node \text{ is a literal} \end{cases}$$

ψ_h is based on the HiBISCuS summary [20], and replaces IRIs by their authority and literals by "any". It can can be expressed as a simple CONSTRUCT SPARQL query:

```
CONSTRUCT { ?ps ?p ?po } WHERE {
  ?s ?p ?o FILTER ISIRI( ?s )
  BIND(URI(REPLACE(STR( ?s ), "^(https?://?.*?)/.*", "$1")) AS ?ps )
  BIND(IF(ISIRI( ?o ),
       URI(REPLACE(STR( ?o ),
                   "^(https?://?.*?)/.*", "$1")),"any") AS ?po )}
```

To illustrate, the quotient summary of the federation F_1 by ψ_h is a federation $\psi_h(F_1)$ comprising 8 quads:

http://D1	foaf:based_near	http://D2	http://D1
http://D1	foaf:name	any	http://D1
http://D2	geo:parentFeature	http://D2	http://D2
http://D2	geo:names	any	http://D2
http://D3	foaf:based_near	http://D4	http://D3
http://D3	foaf:name	any	http://D3
http://D4	geo:parentFeature	http://D4	http://D4
http://D4	geo:names	any	http://D4

On this simple example, both F_1 and $\psi_h(F_1)$ have the same size. However, in practice, ψ_h generates summaries that are orders of magnitude smaller than original federations as shown in Table 2. Although very compact, experimental results demonstrate that quotient summaries generated by ψ_h allows FedUP to find efficient query plans. The intuition behind ψ_h is that authorities alone allow federation engines to identify the endpoints hosting the triples.

To build Result-Aware source selection, on summaries, FedUP applies the same summary function ψ_h to triple patterns of the input query. As ψ_h projects all literals on one constant, most query filters cannot be properly evaluated and are removed. Our motivating query $S6$ remains identical, except for the literal ``Federal Republic of Germany''

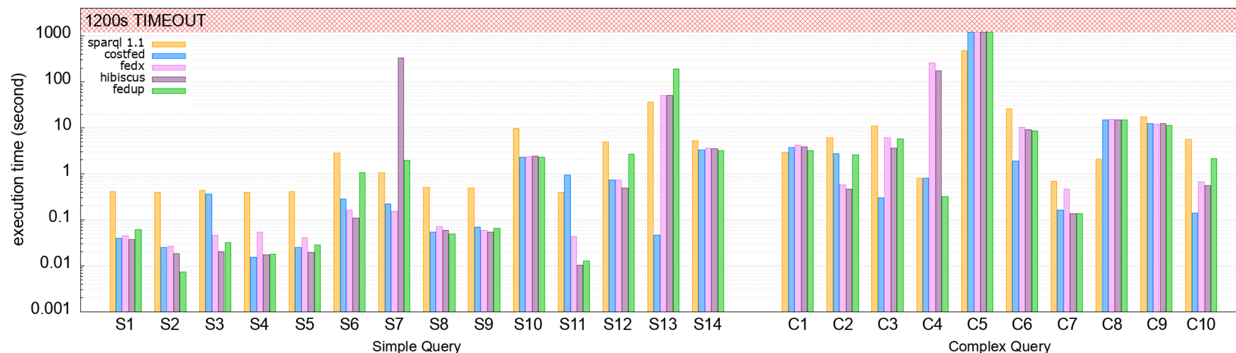


Figure 5: Query execution time of federation engines for simple and complex queries of LargeRDFBench.

Table 2: The size of the summaries of the federation engines.

	FedShop20	FedShop200	LargeRDFBench
Federation	5,167,810 quads	41,821,489 quads	1,004,491,996 quads
CostFed [21]	9MB	95MB	11MB
SemaGrow [6]	1,8MB	18MB	
HiBISCuS [20]	892KB	9MB	539KB
FedUP	**76KB (0.6K quads)**	**767KB (1K quads)**	**705KB (6K quads)**

that becomes `''any''`. As the Result-Aware property is now ensured on the summary and not on the original federation, some subexpressions of the query plan may return empty results on the federation. However, experimental results demonstrate that query plans generated using such summaries remain very efficient.

4 EXPERIMENTAL STUDY

This experimental study aims to empirically answer two questions:

(1) Does FedUP perform better than existing engines on LargeRDFBench [19] where the federation comprises a dozen of endpoints?
(2) Does FedUP perform better than existing engines when the size of the federation grows up to 200 endpoints?

To conduct the experimental study, we implemented FedUP on top of FedX [23], i.e., FedUP produces Result-Aware source selection plans that are optimized and executed by FedX. Similarly to many state-of-the-art federation engines [21, 23], FedUP performs an additional pruning step by performing `ASK` queries in the presence of general predicate with constants. The summary and `ASK` queries allow FedUP to build accurate logical plans even in large-scale federations. Code, configurations, queries, and datasets are available on the GitHub platform at `https://github.com/GDD-Nantes/fedup-experiments`.

4.1 Experimental Setup

We evaluated FedUP on 2 benchmarks:

(1) LargeRDFBench [19] is the most commonly used benchmark to evaluate the performance of federation engines [1, 6, 20, 21, 23]. The benchmark is explicitly designed to represent federated SPARQL queries on real-world datasets. In our experiments, the workload comprises 14 simple queries (S) and 10 complex queries (C). Each dataset is loaded into a separate endpoint, resulting in a total of 14 endpoints. These queries cover all types of core SPARQL operators such as `UNION`, `OPTIONAL`, and `FILTER`. However, LargeRDFBench cannot scale on the number of sources.
(2) FedShop [9] is a recent benchmark that enables scaling on the number of sources. It provides queries and datasets from 20 endpoints up to 200 endpoints. The queries cover all types of core SPARQL operators. The query workload consists of 10 instances for each of the 12 templates, resulting in 120 instances. Each instance is generated by replacing placeholders in the template with randomly selected values. Query templates are organized into 3 levels of source selection difficulties: Single-Domain (SD), Multi-Domain (MD), and Cross-Domain (CD). SD queries are restricted to a single source with no global join variables and bound triple pattern subjects. MD queries are assessed on multiple sources without global join variables and unbound subjects. CD queries must be broken down into subqueries, each evaluated on different sources, requiring global join variables.

To run FedUP, we computed the quotient summaries for LargeRDFBench and FedShop. Table 2 represents the size of the summaries for each federation engine. The summaries of FedUP remains very compact, although multiplying the size of the federation by 10 increases the size of FedUP's summary by a multiplicative factor of 10 in FedShop.

For the two experiments, all federation graphs are stored as named graphs in a single Virtuoso endpoint (Version 7.2.7.3234-pthreads).

To run our experiments, we used a local cloud instance with Ubuntu 20.04.4 LTS, a AMD EPYC 7513-Core processor with 16 vCPUs allocated to the VM, 1TB SSD, and 64GB of RAM. The Virtuoso endpoint hosting the data, as well as the federation engines, ran on the same machine.

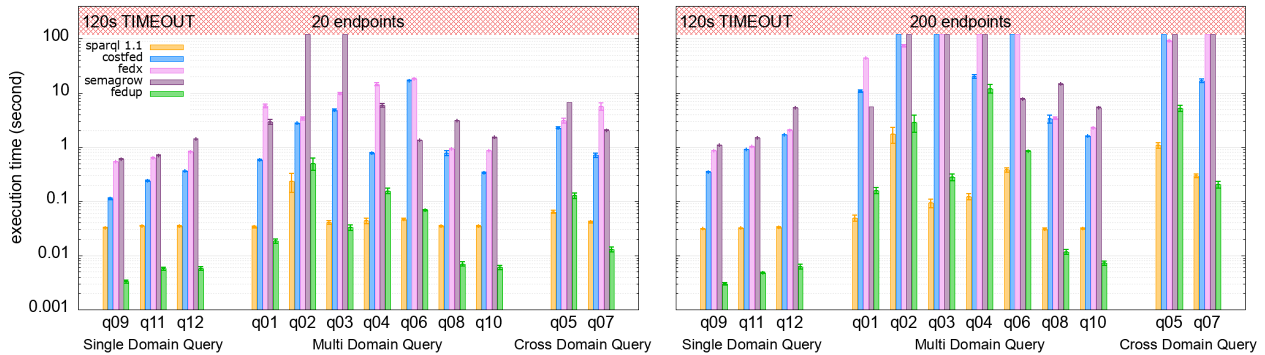


Figure 6: Query execution time of federation engines on FedShop queries for 20 and 200 endpoints.

4.2 LargeRDFBench: Parity Among Engines

In this experiment, we compared FedUP with FedX [23], HiBISCuS [20] (Ask dominant), and CostFed [21] (Ask dominant). We also included the SPARQL 1.1 Service queries available in the LargeRDFBench that we executed with Apache Jena. These queries are hand-crafted with predefined source selection.

Figure 5 presents the performance of the different engines. The x-axis represents the competitors for each query, and the y-axis displays the execution time on a logarithmic scale. This execution time is defined as the time spent by each federation engine from source selection to federated query execution. Each query underwent 5 runs, and the reported measurements in Figure 5 are the averages of these runs. We set a 20-minute timeout before stopping the federated query execution.

Figure 5 shows that most of the time, federation engines yield comparable execution time. Federation engines build logical plans that are equivalent. This is attributed to either the simplicity of queries (20 out of 24 queries require a single combination of sources), or their low selectivity regarding source selection for FedUP to make a difference.

Figure 5 shows that, most notably on Query $S13$, CostFed outperforms its competitors by orders of magnitude, which highlights the need for better join ordering: when disabled, CostFed executes Query $S13$ in $10s$ instead of $50ms$.

Overall, FedUP does not provide significant improvements over state-of-the-art on LargeRDFBench. In the LargeRDFBench context, joins-over-unions query plans are also Result-Aware query plans.

4.3 FedShop: FedUP Outperforms the Others

In this experiment, we compared FedUP with FedX [23], Semagrow [6], and CostFed [21] (Ask Dominant). Fedshop comes with RSA queries written as SPARQL 1.1 Service queries that we executed with Apache Jena. These queries are hand-crafted with predefined source selection that follows unions-over-joins logical plans.

Figure 6 reports the performance of federation engines in terms of execution time. We run the 10 configurations of FedShop but only reported results for the 20 and 200 endpoints configurations. The x-axis denotes the query templates.

For each templated query, each bar on the x-axis represents the evaluated engine, while its height represents the average execution time of the templated queries on a logarithmic scale. On the left, the federation comprises 20 endpoints, while on the right, the federation comprises 200 endpoints. The timeout is configured for 120 seconds to align with FedShop's focus on interactive eCommerce use cases, where end-users expect quick answers.

Figure 6 shows that, for all queries, FedUP outperforms its competitors from 1 to 3 orders of magnitude in terms of execution time. On SD queries, FedUP's summary allows it to efficiently find the best logical plan comprising a single exclusive group. FedUP built its quotient summary using the authority of URIs, and since SD queries stay on a single domain, evaluating the source selection query on this summary is fast and accurate. Competitors find the same plan but spend most of the time in source selection. For instance, CostFed spends $1s$ of source selection for $10ms$ of actual execution on template $q12$ when there are 200 endpoints.

On MD queries, FedUP remains close to the baseline except for $q04$ with 200 endpoints. Similarly to SD queries, FedUP's summary allows it to efficiently find the minimal set of combinations, each comprising a single exclusive group. The baseline and FedUP prove that a federation engine could execute these queries under $2s$, however, competing engines present drastically worse execution times, even reaching the 2-minute timeout on occasions when 200 endpoints are involved. Their source selection process is fast but builds joins-over-unions plans that cannot be transformed into efficient physical plans: not only do they fail to identify exclusive groups, but they create combinations without results that still need to be checked at execution time, hence wasting resources. For $q04$, FedUP and CostFed build equivalent plans that need to check numerous combinations without results, hence providing similar performance.

On CD queries, FedUP remains close to the baseline as well. It extensively uses `ASK` queries to kickstart its source selection query execution, restricting the research space of solutions mappings to build its logical plans. For $q07$, its plans are equivalent to the SPARQL 1.1 baseline. However, for $q05$, FedUP creates plans of 200 combinations while the baseline needs 17 combinations on average, hence spending more time to evaluate the federated query.

Figure 6 shows that half of the time, FedUP performs better than the RSA SPARQL 1.1 queries. Indeed, FedUP benefits from parallel execution, where up to 8 FedX instances are in charge of executing subparts of the logical plan. The baseline uses Apache Jena to evaluate its `SERVICE` queries and, therefore, does not benefit from such a feature.

Overall, FedUP outperforms state-of-the-art federations engines by orders of magnitude. Thanks to its summary and `ASK` queries, FedUP quickly produces better logical plans. Consequently, FedUP can execute the federated query before reaching the timeout even on large-scale federations comprising up to 200 endpoints.

5 RELATED WORK

Given a federation of SPARQL endpoints, federation engines process SPARQL queries in three steps [2]: (i) Source selection and query decomposition, (ii) query optimization, and (iii) query execution. In this paper, we focused on the source selection and query decomposition step.

The source selection aims to identify the set of sources to contact per triple pattern [21] in order to generate a logical plan representing the federated query [7]. To perform source selection, some federation engines such as FedX [23] or Lusail [1] are "zero-knowledge" as they only require a catalog of SPARQL endpoints to contact through `ASK` queries. However, most often, federation engines require the existence of summaries computed over the federation of SPARQL endpoints [3, 6, 10, 11, 15, 16, 18, 20, 21, 25].

Without summaries, federation engines [23] are limited to triple-pattern-wise source selection where every triple pattern is independently associated with its list of relevant sources to contact. Using summaries, federation engines such as HiBISCuS [20] or CostFed [21] further reduce the size of relevant sources by performing BGP-aware (or join-aware) source selection, i.e., they detect and prune sources that do not contribute to the final results of the query. FedUP operates in a similar fashion using its tiny HiBISCuS-like summary. However, both triple-pattern-wise and BGP-aware source selection are expressed as a mapping from a triple pattern to a set of sources. By default, they produce joins-over-unions logical plans [7]. By default, some optimal plans are unavailable to them [7].

To alleviate this issue, we refined the source selection problem to keep the relationship between operators and sources. A Result-Aware source selection produced by FedUP is a tree where every subtree under the root union produces results. Result-Aware source selection plans provide additional interesting features: (i) the subexpressions in unions are independent and therefore, easily parallelized for improved performance as shown in Section 4.3; and (ii) triple patterns that share a same source can be gathered in exclusive groups [23] which is more likely to happen in Result-Aware query plans, as shown in Figure 3b compared to Figure 3a. It is worth noting that Lusail [1] improves the grouping of sources by determining when join variables are local or global using online set differences, i.e., by executing simple `FILTER NOT EXISTS` queries. The Lusail grouping approach can be applied on top of any existing source selection technique, including Result-Aware source selection.

6 CONCLUSION

In this paper, we refined the problem of source selection by proposing a new Result-Aware source selection problem. Result-Aware query plans ensure that all combinations of relevant sources contribute to the final results of the query. To avoid the combinatory explosion induced by a normalization phase of joins-over-unions query plans, we proposed to iteratively build such Result-Aware query plans.

Building a Result-Aware query plan is driven by results; however, query results are unavailable during source selection. We solved this issue by computing a Result-Aware query plan on quotient summaries. Of course, summaries introduce inaccuracies; however, the results of benchmarks demonstrate huge performance improvements, especially when the size of the federation grows. On the FedShop benchmark, Result-Aware query plan outperforms traditional approaches by an order of magnitude, offering new perspectives for federated query processing.

In future work, we plan to support other challenging SPARQL clauses such as `MINUS`, and `NOT EXISTS`. There is also important room for improvement for optimizing unions-over-joins query plan. Better factorization, and better join ordering could improve the performance and fill the remaining gap with RSA queries in the FedShop benchmark.

Acknowledgments This work is supported by the French ANR project DeKaloG (Decentralized Knowledge Graphs) `ANR-19-CE23-0014`, and the French Labex CominLabs project MiKroloG (The Microdata Knowledge Graph).

REFERENCES

[1] I. Abdelaziz, E. Mansour, M. Ouzzani, A. Aboulnaga, and P. Kalnis. Lusail: A system for querying linked data at scale. *Proc. VLDB Endow.*, 11(4):485–498, 2017.

[2] M. Acosta, O. Hartig, and J. Sequeda. *Federated RDF query processing*, pages 1–8. Springer, Cham, 2018.

[3] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: An adaptive query processing engine for SPARQL endpoints. In *10th International Semantic Web Conference (ISWC)*, pages 18–34, Berlin, Heidelberg, 2011. Springer.

[4] Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou, and M. Zneika. Summarizing semantic graphs: A survey. *VLDB J.*, 28(3):295–327, 2019.

[5] Š. Čebirić, F. Goasdoué, and I. Manolescu. A framework for efficient representative summarization of RDF graphs. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC)*, volume 1963

of *CEUR Workshop Proceedings*, page 4, Vienna, Austria, 2017. CEUR-WS.org.

[6] A. Charalambidis, A. Troumpoukis, and S. Konstantopoulos. SemaGrow: Optimizing federated SPARQL queries. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 121–128, New York, NY, USA, 2015. ACM.

[7] S. Cheng and O. Hartig. FedQPL: A language for logical query plans over heterogeneous federations of RDF data sources. In *the 22nd International Conference on Information Integration and Web-Based Applications & Services*, page 436–445, New York, NY, USA, 2021. ACM.

[8] S. Cheng and O. Hartig. Towards query processing over heterogeneous federations of RDF data sources. In *The Semantic Web: ESWC 2022 Satellite Events*, pages 57–62, Crete, Greece, 2022. Springer.

[9] M.-H. Dang, J. Aimonier-Davat, P. Molli, O. Hartig, H. Skaf-Molli, and Y. L. Crom. FedShop: A benchmark for testing the scalability of SPARQL federation engines. In *International Semantic Web Conference (ISWC)*, pages 285–301, Athens, Greece, 2023. Springer.

[10] K. M. Endris, M. Galkin, I. Lytra, M. N. Mami, M. Vidal, and S. Auer. MULDER: Querying the linked data web by bridging RDF molecule templates. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 3–18, Lyon, France, 2017. Springer.

[11] O. Görlitz and S. Staab. SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In *Proceedings of the Second International Conference on Consuming Linked Data*, volume 782, pages 13–24, Aachen, DEU, 2011. CEUR-WS.org.

[12] S. Harris and A. Seaborne. SPARQL 1.1 query language, 2013.

[13] L. Heling and M. Acosta. Federated SPARQL query processing over heterogeneous linked data fragments. In *Proceedings of the ACM Web Conference 2022*, pages 1047–1057, New York, NY, USA, 2022. ACM.

[14] P. Maillot, O. Corby, C. Faron, F. Gandon, and F. Michel. Indegx: A model and a framework for indexing RDF knowledge graphs with sparql-based test suits. *J. Web Semant.*, 76:100775, 2023.

[15] G. Montoya, H. Skaf-Molli, and K. Hose. The Odyssey approach for optimizing federated SPARQL queries. In *International Semantic Web Conference (ISWC)*, pages 471–489, Vienna, Austria, 2017. Springer.

[16] G. Montoya, H. Skaf-Molli, P. Molli, and M.-E. Vidal. Decomposing federated queries in presence of replicated fragments. *Journal of Web Semantics*, 42:1–18, 2017.

[17] J. Pérez, M. Arenas, and C. Gutiérrez. Semantics and complexity of SPARQL. *ACM Transations on Database Systems*, 34(3):16:1–16:45, 2009.

[18] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *Extended Semantic Web Conference (ESWC)*, pages 524–538, Tenerife, Canary Islands, Spain, 2008. Springer.

[19] M. Saleem, A. Hasnain, and A. N. Ngomo. LargeRDFBench: A billion triples benchmark for SPARQL endpoint federation. *J. Web Semant.*, 48:85–125, 2018.

[20] M. Saleem and A.-C. N. Ngomo. HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In *European Semantic Web Conference (ESWC)*, pages 176–191, Anissaras, Crete, Greece, 2014. Springer.

[21] M. Saleem, A. Potocki, T. Soru, O. Hartig, and A.-C. N. Ngomo. CostFed: Cost-based query optimization for SPARQL endpoint federation. In *14th International Conference on Semantic Systems (SEMANTICS)*, pages 163–174, Vienna, Austria, 2018. Elsevier.

[22] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. In *10th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, pages 585–600, Bonn, Germany, 2011. Springer.

[23] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference (ISWC)*, pages 601–616, Bonn, Germany, 2011. Springer.

[24] P. Vandenbussche, J. Umbrich, L. Matteis, A. Hogan, and C. B. Aranda. SPARQLES: monitoring public SPARQL endpoints. *Semantic Web*, 8(6):1049–1065, 2017.

[25] M. Vidal, S. Castillo, M. Acosta, G. Montoya, and G. Palma. On the selection of SPARQL endpoints to efficiently execute federated SPARQL queries. *Trans. Large Scale Data Knowl. Centered Syst.*, 25:109–149, 2016.

A APPENDIX

A.1 The Eighth Equivalence Rule

Let φ_1, φ_2, and φ_3 be FedQPL expressions that are valid for F. In a federation context, it does not hold that:

(R8) $\mathit{leftjoin}(\varphi_1, \mathit{union}(\varphi_2, \varphi_3)) \stackrel{F}{\equiv} \mathit{union}(\mathit{leftjoin}(\varphi_1, \varphi_2), \mathit{leftjoin}(\varphi_1, \varphi_3))$

To illustrate, let us consider the query Q_o with an `OPTIONAL` and two triple patterns:

```
SELECT * WHERE {
  ?artist foaf:based_near ?location .                    #tp1
  OPTIONAL { ?location geo:parentFeature ?germany }} #tp2
```

The federation F comprises 2 members f_1 and f_2 with 3 triples as follows:

http://f1	http://f1/Scorpions	foaf:based_near	http://f1/Hanover
http://f1	http://f1/Kraftwerk	foaf:based_near	http://f2/Berlin
http://f2	http://f2/Berlin	geo:parentFeature	http://f2/Germany

With such a federation F and query Q_o, the FedQPL expression is:

$$\varphi_o = leftjoin(req_{f_1}^{tp1}, mu\ \{req_{f_1}^{tp2}, req_{f_2}^{tp2}\})$$

The evaluation $[\![\varphi_o]\!]_F$ of φ_o over F returns:

	?artist	?location	?germany
μ_1	http://f1/Scorpions	http://f1/Hanover	
μ_2	http://f1/Kraftwerk	http://f2/Berlin	http://f2/Germany

However, after applying the equivalence rule $R8$ on φ_o, we get the following expression $\varphi_{o'}$:

$$\varphi_{o'} = mu\ \{leftjoin(req_{f_1}^{tp1}, req_{f_1}^{tp2}), leftjoin(req_{f_1}^{tp1}, req_{f_2}^{tp2})\}$$

The evaluation of $\varphi_{o'}$ over F returns unexpected results:

	?artist	?location	?germany
μ_1	http://f1/Scorpions	http://f1/Hanover	
μ_2	http://f1/Kraftwerk	http://f2/Berlin	http://f2/Germany
μ_3	http://f1/Kraftwerk	http://f2/Berlin	

A.2 $\mathcal{A}$ Returns Complete Results

Proof. Let Q be a SPARQL query and F be a federation, $\mathcal{A}(Q,F)$ returns complete results if and only if $\forall\mu \in [\![Q]\!]_F, \mu \in sols(\mathcal{A}(Q,F))$. To demonstrate that $\mathcal{A}$ returns complete results, we use the FedQPL equivalences with the SPARQL algebra as defined in Definition 6 [7]. We proceed by contradiction assuming that $\exists\mu \notin sols(\mathcal{A}(Q,F)), \mu \in [\![Q]\!]_F$.

(1) If Q is a triple pattern tp then $\mu \notin sols(\mathcal{A}(Q,F))$
$\Leftrightarrow \mu \notin \bigcup_{f\in F} sols(req_f^{tp})$
$\Leftrightarrow \mu \notin \bigcup_{f\in F} [\![tp]\!]_f$
$\Leftrightarrow \mu \notin [\![Q]\!]_F$

(2) If Q is P_1 `AND` P_2 then $\mu \notin sols(\mathcal{A}(Q,F))$
$\Leftrightarrow \mu \notin \bigcup_{\varphi_1,\varphi_2\in\Phi_1,\Phi_2} sols(mj\ \{\varphi_1,\varphi_2\})$
$\Leftrightarrow \mu \notin (\bigcup_{\varphi_1\in\Phi_1} sols(\varphi_1)) \bowtie (\bigcup_{\varphi_2\in\Phi_2} sols(\varphi_2))$
$\Leftrightarrow \mu \notin [\![P_1]\!]_F \bowtie [\![P_2]\!]_F$
$\Leftrightarrow \mu \notin [\![Q]\!]_F$

(3) If Q is P_1 `UNION` P_2 then $\mu \notin sols(\mathcal{A}(Q,F))$
$\Leftrightarrow \mu \notin (\bigcup_{\varphi_1\in\Phi_1} sols(\varphi_1)) \cup (\bigcup_{\varphi_2\in\Phi_2} sols(\varphi_2))$
$\Leftrightarrow \mu \notin [\![P_1]\!]_F \cup [\![P_2]\!]_F$
$\Leftrightarrow \mu \notin [\![Q]\!]_F$

(4) If Q is P_1 `FILTER` R then $\mu \notin sols(\mathcal{A}(Q,F))$
$\Leftrightarrow \mu \notin \bigcup_{\varphi\in\Phi} sols(filter^R(\varphi))$
$\Leftrightarrow \mu \notin \{\mu' \mid \mu' \in \bigcup_{\varphi\in\Phi} sols(\varphi) \wedge \mu' \models R\}$
$\Leftrightarrow \mu \notin \{\mu' \mid \mu' \in [\![P_1]\!]_F \wedge \mu' \models R\}$
$\Leftrightarrow \mu \notin [\![Q]\!]_F$

(5) If Q is P_1 `OPTIONAL` P_2 then $\mu \notin sols(\mathcal{A}(Q,F))$
$\Leftrightarrow \mu \notin \bigcup_{\varphi_1\in\Phi_1} leftjoin(\varphi_1, \Phi_{join}^{\varphi_1})$
$\Leftrightarrow \mu \notin (X \cup ((\bigcup_{\varphi_1\in\Phi_1} sols(\varphi_1)) \setminus X))$
$\Leftrightarrow \mu \notin (Y \cup ((\bigcup_{\varphi_1\in\Phi_1} sols(\varphi_1)) \setminus Y))$
$\Leftrightarrow \mu \notin ([\![P_1 \bowtie P_2]\!]_F \cup ([\![P_1]\!]_F \setminus [\![P_1 \bowtie P_2]\!]_F))$
$\Leftrightarrow \mu \notin [\![Q]\!]_F$
$\Phi_{join}^{\varphi_1} = \{\varphi_2 \mid \varphi_2 \in \Phi_2 \wedge sols(mj\ \{\varphi_1,\varphi_2\}) \neq \varnothing\}$
$X = \bigcup_{\varphi_1\in\Phi_1} sols(mj\ \{\varphi_1, mu\ \{\Phi_{join}^{\varphi_1}\}\})$
$Y = \bigcup_{\varphi_1\in\Phi_1} sols(mj\ \{\varphi_1, mu\ \{\varphi_2 \mid \varphi_2 \in \Phi_2\}\})$

Therefore, $\mathcal{A}$ returns complete results. ☐

A.3 $\mathcal{A}$ Returns Correct Results

Proof. Let Q be a SPARQL query and F be a federation, $\mathcal{A}(Q,F)$ returns correct results if and only if $\forall\mu \in \mathcal{A}(Q,F), \mu \in [\![Q]\!]_F$. The proof of correctness is analogous to the proof of completeness in Appendix A.2. ☐

A.4 $\mathcal{A}$ Returns Result-Aware Expressions

Proof. Let Q be a SPARQL query and F be a federation such that $[\![Q]\!]_F \neq \varnothing$. Let $\varphi = \mathcal{A}(Q,F)$ be a FedQPL expression that is not Result-Aware. Consequently, there exists $\varphi' \subseteq \varphi$ such that φ' does not contribute to $[\![Q]\!]_F$.

(1) If Q is a triple pattern tp, φ is not Result-Aware if there exists φ' in Φ_{TP} such that $sols(\varphi') = \varnothing$, which is impossible by definition of Φ_{TP}. Consequently, φ is Result-Aware.

(2) If Q is P_1 `AND` P_2, φ is not Result-Aware if:
  (A) there exists φ' in Φ_{JOIN} such that $sols(\varphi') = \varnothing$, which is impossible by definition of Φ_{JOIN}.
  (B) there exists φ_1 in Φ_1 such that φ_1 does not contribute to $[\![Q]\!]_F$. If $\varphi_1 \subset \varphi$, there exists $\varphi_{join} = mj\ \{\varphi_1,\varphi_2\}$ in Φ_{JOIN}. By definition, if $sols(\varphi_{join}) \neq \varnothing$, both φ_1 and φ_2 contribute to $sols(\varphi_{join})$. As φ_{join} contributes to $[\![Q]\!]_F$, φ_1 also contributes to $[\![Q]\!]_F$.
  (C) there exists φ_2 in Φ_2 such that φ_2 does not contribute to $[\![Q]\!]_F$. For the same reason as φ_1, if $\varphi_2 \subset \varphi$, φ_2 contributes to $[\![Q]\!]_F$.
  (D) there exists $\varphi' \subset \varphi_1$ where $\varphi_1 \in \Phi_1$ such that φ_1 contributes to $[\![Q]\!]_F$ but φ' does not. By induction, we assume that $\mathcal{A}(P_1,F)$ generates a Result-Aware FedQPL expression. As $\mathcal{A}(P_1,F) = mu\ \Phi_1$, all FedQPL expressions and subexpressions in Φ_1 contribute to $[\![P_1]\!]_F$. As φ_1 contributes to $[\![Q]\!]_F$, all subexpressions $\varphi' \subset \varphi_1$ also contributes to $[\![Q]\!]_F$.
  (E) there exists $\varphi' \subset \varphi_2$ where $\varphi_2 \in \Phi_2$ such that φ_2 contributes to $[\![Q]\!]_F$, but φ' does not. Using the same reasoning as for $\varphi' \subset \varphi_1$, we demonstrate that all subexpressions $\varphi' \subset \varphi_2$ contributes to $[\![Q]\!]_F$.

  As a result, if Q is P_1 `AND` P_2, there does not exist $\varphi' \subset \varphi$ such that φ' does not contribute to $[\![Q]\!]_F$, consequently, φ is Result-Aware.

(3) If Q is P_1 `OPTIONAL` P_2, φ is not Result-Aware if:
  (A) there exists φ' in Φ_{OPT} such that $sols(\varphi') = \varnothing$, which is impossible by definition of Φ_{OPT}.
  (B) there exists $\varphi_1 \in \Phi_1$ such that φ_1 does not contribute to $[\![Q]\!]_F$. If $\varphi_1 \subset \varphi$, there are two cases: i. $\varphi_1 \in \Phi_{OPT} \setminus \Phi_1$. In this case, there exists $\varphi_{opt} = leftjoin(\varphi_1, mu\ \Phi_{join}^{\varphi_1})$ in Φ_{OPT}. By definition, if $sols(\varphi_{opt}) \neq \varnothing$, φ_1 contributes to $[\![Q]\!]_F$. ii. $\varphi_1 \in \Phi_{OPT} \cap \Phi_1$. In this case, φ_1 contributes to $[\![Q]\!]_F$ by definition of Φ_{OPT}.

(C) there exists $\varphi_2 \in \Phi_2$ such that φ_2 does not contribute to $[\![Q]\!]_F$. If $\varphi_2 \subset \varphi$, there exists $\varphi_{opt} = \mathit{leftjoin}(\varphi_1, \mathit{mu}\ \Phi^{\varphi_1}_{join})$ in Φ_{OPT} such that $\varphi_2 \in \Phi^{\varphi_1}_{join}$. By definition, if $\varphi_2 \in \Phi^{\varphi_1}_{join}$ then $\mathit{sols}(\mathit{mj}\ \{\varphi_1, \varphi_2\}) \neq \varnothing$, and φ_2 contributes to $\mathit{sols}(\varphi_{opt})$. Consequently, φ_2 contributes to $[\![Q]\!]_F$.

(D) there exists $\varphi' \subset \varphi_1$ where $\varphi_1 \in \Phi_1$ such that φ_1 contributes to $[\![Q]\!]_F$ but φ' does not. Using the same reasoning as for $\varphi' \subset \varphi_1$ when Q is P_1 `AND` P_2, we demonstrate that all subexpressions $\varphi' \subset \varphi_2$ contributes to $[\![Q]\!]_F$ when Q is P_1 `OPTIONAL` P_2.

(E) there exists $\varphi' \subset \varphi_2$ where $\varphi_2 \in \Phi_2$ such that φ_2 contributes to $[\![Q]\!]_F$ but φ' does not. We use the same reasoning as for $\varphi' \subset \varphi_1$.

As a result, if Q is P_1 `OPTIONAL` P_2, there does not exist $\varphi' \subset \varphi$ such that φ' does not contribute to $[\![Q]\!]_F$, consequently, φ is Result-Aware.

(4) If Q is P_1 `FILTER` R, φ is not Result-Aware if:

(A) there exists φ' in Φ_{FILTER} such that $\mathit{sols}(\varphi') = \varnothing$, which is impossible by definition of Φ_{FILTER}.

(B) there exists φ' in Φ such that φ' does not contribute to $[\![Q]\!]_F$. If $\varphi' \subset \varphi$, there exists $\varphi_{filter} = \mathit{filter}^R(\varphi')$ in Φ_{FILTER}. By definition, if $\mathit{sols}(\varphi_{filter}) \neq \varnothing$, φ' contributes to $\mathit{sols}(\varphi_{filter})$. Consequently, φ' contributes to $[\![Q]\!]_F$.

(C) there exists $\varphi'' \subset \varphi'$ where $\varphi' \in \Phi$ such that φ' contributes to $[\![Q]\!]_F$ but φ'' does not. Using the same reasoning as for $\varphi' \subset \varphi_1$ when Q is P_1 AND P_2, we demonstrate that all subexpressions $\varphi'' \subset \varphi'$ contributes to $[\![Q]\!]_F$ when Q is P `FILTER` R.

As a result, if Q is P `FILTER` R, there does not exist $\varphi' \subset \varphi$ such that φ' does not contribute to $[\![Q]\!]_F$, consequently, φ is Result-Aware.

(5) If Q is P_1 `UNION` P_2, φ is not Result-Aware if:

(A) there exists φ' in Φ_{UNION} such that $\mathit{sols}(\varphi') = \varnothing$, which is impossible by definition of Φ_{UNION}.

(B) there exists $\varphi' \subset \varphi_1$ where $\varphi_1 \in \Phi_1$ such that φ_1 contributes to $[\![Q]\!]_F$ but φ' does not. Using the same reasoning as for $\varphi' \subset \varphi_1$ when Q is P_1 `AND` P_2, we demonstrate that all subexpressions $\varphi' \subset \varphi_2$ contributes to $[\![Q]\!]_F$ when Q is P_1 `UNION` P_2.

(C) there exists $\varphi' \subset \varphi_2$ where $\varphi_2 \in \Phi_2$ such that φ_2 contributes to $[\![Q]\!]_F$ but φ' does not. We use the same reasoning as for $\varphi' \subset \varphi_1$.

As a result, if Q is P_1 `UNION` P_2, there does not exist $\varphi' \subset \varphi$ such that φ' does not contribute to $[\![Q]\!]_F$, consequently, φ is Result-Aware.

Therefore, $\mathcal{A}$ returns Result-Aware expressions. □