Federated SPARQL Queries Processing with Replicated Fragments

Gabriela Montoya[1,2], Hala Skaf-Molli[1], Pascal Molli[1], and Maria-Esther Vidal[3]

[1] LINA – Nantes University, Nantes, France
{gabriela.montoya,hala.skaf,pascal.molli}@univ-nantes.fr
[2] Unit UMR6241 CNRS, Nantes, France
[3] Universidad Simón Bolívar, Caracas, Venezuela
mvidal@ldc.usb.ve

Abstract. Federated query engines provide a unified query interface to federations of SPARQL endpoints. Replicating data fragments from different Linked Data sources facilitates data re-organization to better fit federated query processing needs of data consumers. However, existing federated query engines are not designed to support replication and replicated data can negatively impact their performance. In this paper, we formulate the source selection problem with fragment replication (SSP-FR). For a given set of endpoints with replicated fragments and a SPARQL query, the problem is to select the endpoints that minimize the number of tuples to be transferred. We devise the Fedra source selection algorithm that approximates SSP-FR. We implement Fedra in the state-of-the-art federated query engines FedX and ANAPSID, and empirically evaluate their performance. Experimental results suggest that Fedra efficiently solves SSP-FR, reducing the number of selected SPARQL endpoints as well as the size of query intermediate results.
Keywords: Linked Data · Federated Query Processing · Source Selection · Fragment Replication

1 Introduction

SPARQL endpoints enable to consume RDF data exploiting the *expressiveness* of the SPARQL query language. Nevertheless, recent studies reveal that existing public SPARQL endpoints main limitation is availability [4].

In distributed databases [17], a common practice to overcome availability problems is to replicate data near data consumers. Replication can be achieved by complete dataset replication, active caching, pre-fetching or fragmentation [13].

RDF data consumers can replicate *subsets* of RDF datasets or *replicated fragments*, and make them accessible through SPARQL endpoints. This will provide the support for an efficient RDF data re-organization according to the *needs* and *computational resource capacity* of data consumers, while these data can be still *accessed* using SPARQL endpoints. Unfortunately, although SPARQL endpoints can *transparently* access replicated fragments, as well as maintain their *consistency* [13], federated query engines are not tailored to exploit the benefits of replicated fragments.

Federated SPARQL engines [1], [6], [9], [18], [21] allow data consumers to execute SPARQL queries against a federation of SPARQL endpoints. However, these engines are just designed to select the SPARQL endpoints that ensure both a complete answer and an efficient execution of the query. In presence of replication, existing federated query engines may retrieve data from every relevant endpoint, and produce a large number of intermediate results that trigger many requests to the endpoints. Thus, federated query engines may exhibit poor *performance* while *availability* of the selected SPARQL endpoints is negatively impacted.

Although the problem of managing RDF data *overlapping* during federated query processing has been addressed in [12], [20], the problem of managing *replication* in a federation of RDF datasets still remains open. DAW [20] is able to detect overlapping between datasets and optimize source selection based on that. However, because DAW is not designed to manage data replication, there is no support for explicitly define and use replicated fragments. In consequence, DAW may select redundant data sources and generate a high number of intermediate results as we will report in our experiments.

In this paper, we build a replication-aware SPARQL federated query engine by integrating into state-of-the art federated query engines FedX [21] and ANAPSID [1], a source selection strategy called Fedra that solves the source selection problem with fragment replication (SSP-FR). For a given set of SPARQL endpoints with replicated fragments and a SPARQL query, the problem is to minimize the number of transferred data from endpoints to the federated query engines, while preserving answer completeness and reducing data redundancy.

We empirically study federated query engines FedX and ANAPSID extended with Fedra and DAW on synthetic and real datasets. The results suggest that Fedra efficiently reduces intermediate results and data redundancy.

The paper is organized as follows. Section 2 describes background and motivations. Section 3 defines replicated fragments and presents the source selection problem for fragment replication. Section 4 presents the Fedra source selection algorithm. Section 5 reports our experimental results. Section 6 summarizes related works. Finally, conclusions and future works are outlined in Section 7.

2 Background and Motivations

Existing SPARQL federated query engines do not support replicated data. To illustrate, we replicated the DBpedia dataset and defined two federations. The first is composed of one mirror of DBpedia, and the second of two identical mirrors of DBpedia. We used FedX [21] and ANAPSID [1] to execute the query in Figure 1a against both federations. In the first federation, these engines produced the same query answers. On the other hand, for the second federation, these query engines have no knowledge about the relationships among the mirrors of DBpedia, and they contact both data sources. In this way, performance in terms

(a) DBpedia Query

```
select distinct ?p ?m ?n ?d where {
  ?p  dbprop:name ?m .
  ?p  dbprop:nationality ?n .
  ?p  dbprop:doctoralAdvisor ?d
}
```

(b) Query Execution

#DBpedia Replicas	Execution Time (ms)		# Results	
	FedX	ANAPSID	FedX	ANAPSID
1	1,392	22,972	8,921	8,921
2	215,907	⩾ 1,800,000	418	8,921

Fig. 1: DBpedia query and its execution time and number of results against one and two replicas of DBpedia for FedX and ANAPSID

of execution time and number of results, is seriously degraded as depicted in Figure 1b.[1]

Furthermore, if the DAW approach were used, resources of data providers and consumers would be used to compute and download data summaries. DAW could select different DBpedia data sources per triple pattern, and execute thus the join between retrieved data at the federated engine level.

Of course, if federated query engines would know that one endpoint is the mirror of the other, the source selection pruning could be done more efficiently, i.e., only one source would be selected to execute the query. This problem is even more challenging if we consider that one endpoint can partially replicate data from several RDF datasets, i.e., only fragments of several datasets are replicated.

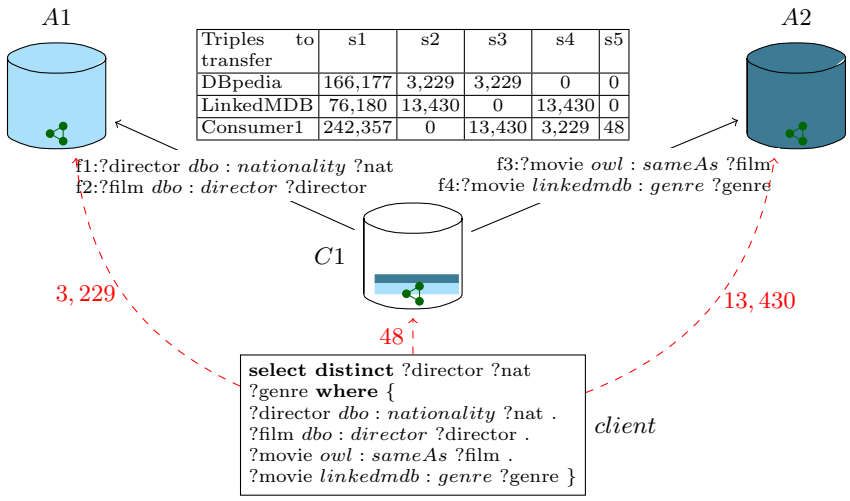

Fig. 2: Client defines a federation composed of DBpedia (A1), LinkedMDB (A2), and one Consumer (C1) endpoints with four replicated fragments

[1] FedX retrieves less results with two mirrors of DBpedia because it reaches the endpoints maximum number of result rows.

Suppose a Web application poses federated queries against endpoints $A1$ (DBpedia) and $A2$ (LinkedMDB). In order to speed up the queries, a data consumer endpoint $C1$ with replicated fragments has been installed as in Figure 2. Fragments are defined as simple CONSTRUCT SPARQL queries with one triple pattern. Fragments allow for the re-organization of RDF data *on* $C1$ to better address needs of data consumers..

Even in this simple setup, processing our running query against a federation including $A1$, $A2$, and $C1$ raises the problem of source selection with fragment replication (SSP-FR). There are at least five options to select sources for executing this query; these choices produce different number of transferred tuples as shown in Figure 2: (i) If no information about replicated fragments is available, all sources may be selected to retrieve data for all the triple patterns. The number of intermediate results is given in the solution $s1$. This will be the behavior of a federated query engine like FedX that ensures answer completeness.[2] (ii) Endpoints $A1$ and $A2$ could be chosen, in this case the number of intermediate results is given in $s2$. The number of intermediate results in $s2$ is less than $s1$ since some joins could be executed at $A1$ and $A2$. (iii) Another choice may be to use the $C1$ endpoint in combination with either $A1$ or $A2$ ($s3, s4$). This produces the same number of intermediate results as in $s2$, but they have the advantage of accessing less public endpoints. (iv) A last choice could be to use the $C1$ endpoint to retrieve data for all the triple patterns ($s5$). This solution profits from replicated fragments to execute opportunistic joins at $C1$; thus, it is able to achieve the *best* performance in terms of the number of intermediate results.

As the number of transferred tuples increases, the availability of the contacted SPARQL endpoints can be affected. A replication aware federated query engine could select the best sources to reduce the size of intermediate results while preserving answer completeness. In this paper, we formally address the following problem: Given a SPARQL query and a set of relevant SPARQL endpoints with replicated fragments, choose the SPARQL endpoints to contact in order to produce a complete query answer and transfer the minimum amount of data. We aim to develop an algorithm that produces solution $s5$ whenever possible, providing as output the sources to be used by a federated query engine.

3 Definitions and Problem Description

This section introduces definitions and the source selection problem with fragment replication (SSP-FR).

3.1 Definitions

Fragments are used to replicate RDF data. The data of a fragment is defined by means of the dataset public endpoint, or *authoritative endpoint*, and a CONSTRUCT query with *one* triple pattern.

[2] In order to preserve joins between different endpoints, each triple pattern should be posed to each endpoint individually.

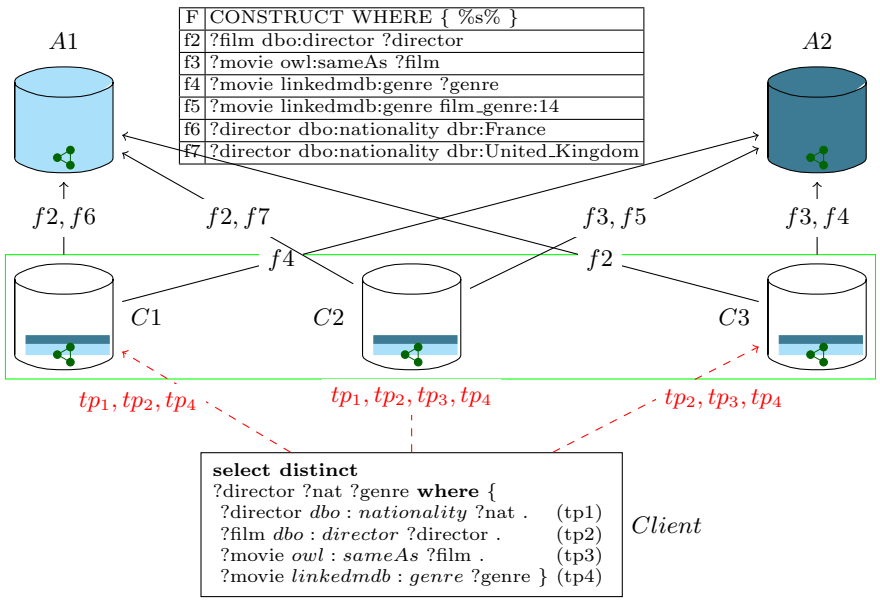

Fig. 3: Client defines a federation composed of $C1$,$C2$, and $C3$ that replicates fragments $f2 - f7$

Definition 1 (Fragment). *A fragment is a tuple* $f = \langle u, s \rangle$
- *u is the non-null URI of the authoritative endpoint where f is available;*
- *s is a CONSTRUCT query with one triple pattern.*

Without loss of generality, s is limited to one triple pattern as in [13], [22]; this reduces the complexity of *fragment containment* problem as described in Definition 2. Additionally, we assume replicated fragments comprise RDF data accessible from public endpoints, i.e., the authoritative endpoints of the replicated fragments are disjoint with data consumer endpoints. This will allow data consumers to re-organize RDF data replicated from different public endpoints to fit in this way, their needs and requirements.

In this work, we make the following assumptions: (i) Fragments are replicated from public endpoints, and there is just one level of replication. (ii) Fragments are read-only and perfectly synchronized; the fragment synchronization problem is studied in [13], while querying fragments with divergence is addressed in [16]. (iii) For the sake of simplicity, we suppose that RDF data accessible through the endpoints are described as fragments.

To illustrate, consider the federation given in Figure 3. This federation extends the setup in Figure 2. Suppose three Web applications pose queries against DBpedia and LinkedMDB. To speed up query processing, data consumer endpoints: $C1$, $C2$, and $C3$ with replicated fragments have been configured.

At startup, the federated query engine loads the fragments description for each of the federation endpoints, and computes both the *fragment* and *containment* mappings. The **fragment mappings** is a function that maps fragments to a set of endpoints; the **containment mapping** is based on *containment* relation ($f_l \sqsubseteq f_k$) described in the Definition 2.

Two fragments loaded from two different endpoints Ci, Cj that have the same authoritative endpoint and equivalent construct queries are concatenated in the fragment mapping. For example, the federated engine loads fragments $\langle$*http://dbpedia.org/sparql, ?film db:director ?director*$\rangle$ from $C1, C2, C3$, computes equivalence, and adds in its fragment mapping $\langle$*http://dbpedia.org/sparql, ?film db:director ?director*$\rangle \rightarrow$ {*C1,C2,C3*}.

Query containment and equivalence have been studied extensively. We adapt the definition given in [11] for the case of a triple pattern query.

Definition 2 (Triple Pattern Containment). *Let $TP(D)$ denote the result of execution of the triple pattern TP against an RDF dataset D. Let TP_1 and TP_2 be two triple patterns. We say that TP_1 is contained in TP_2, denoted by $TP_1 \sqsubseteq TP_2$, if for any RDF dataset D, $TP_1(D) \subseteq TP_2(D)$. We say that TP_1 is equivalent to TP_2, denoted by $TP_1 \equiv TP_2$, if $TP_1 \sqsubseteq TP_2$ and $TP_2 \sqsubseteq TP_1$.*

In the case of triple patterns, testing containment [10] amounts to finding a substitution of the variables in the triple patterns.[3] $TP_1 \sqsubseteq TP_2$, iff there is a substitution θ such that applying θ to TP_2 returns the triple pattern TP_1. Testing triple pattern containment has a complexity of $O(1)$. Solving the decision problem of triple pattern containment between TP_1 and TP_2, $TP_1 \sqsubseteq TP_2$, requires to check if $TP1$ imposes at least the same restrictions as TP_2 on the subject, predicate, and object positions, i.e., TP_1 should have at most the same number of unbounded variables as TP_2.

For the federation in Figure 3, $f5 \sqsubseteq f4$ because $f4$ and $f5$ share the same authoritative endpoint and there is a substitution θ defined as $\theta(?genre) = film\_genre : 14$, $\theta(?movie) = ?movie$, and applying θ to $f4$ returns $f5$. After identifying a substitution θ for all pair-wise fragments, it is straightforward to compute a **containment mapping** for a federation of SPARQL endpoints.

We can rely on fragment descriptions and the containment property to determine relevant fragments to a query. Relevant fragments contain relevant RDF data to each of the triple patterns of the query. A fragment is *relevant* to a query Q, if it is relevant to *at least one* triple pattern of the query.

Definition 3 (Fragment relevance). *Let f be a fragment defined by a triple pattern TP_1. Let TP_2 be a triple pattern of a query Q. f is relevant to Q if $TP_2 \sqsubseteq TP_1$ or $TP_1 \sqsubseteq TP_2$.*

Table 1a shows the relevant fragments to the triple patterns in query Q, and the endpoints that provide these fragments. For example, the triple pattern $tp1$ has two relevant fragments: $f6$ and $f7$, and triple pattern $tp4$ has two relevant fragments: $f4$ and $f5$. Fragment $f4$ can produce the complete answer of $tp4$ because $f5 \sqsubseteq f4$, while both $f6$ and $f7$ are required to answer $tp1$.

[3] The substitution operator preserves URIs and literals, only variables are substituted.

Table 1: SSP-FR for query Q over a federation of $C1$, $C2$, and $C3$ of Figure 3

(a) Relevant Fragments to Q

Q triple pattern		RF	Endpoints
tp_1	?director dbo:nationality ?nat	f6	C1
		f7	C2
tp_2	?film dbo:director ?director	f2	C1,C2,C3
tp_3	?movie owl:sameAs ?film	f3	C2,C3
tp_4	?movie linkedmdb:genre ?genre	f4	C1,C3
		f5	C2

(b) Answer completeness preservation

TP	D_0(tp)	D_1(tp)	D_2(tp)
tp_1	{C1,C2}	{C1,C2}	{C1,C2}
tp_2	{C1,C2,C3}	{C1}	{C3}
tp_3	{C2,C3}	{C2}	{C3}
tp_4	{C1,C2,C3}	{C3}	{C3}
Triples to transfer	421,675	170,078	8,953

3.2 Source Selection Problem with Fragment Replication (SSP-FR)

Given a SPARQL query Q, a set of SPARQL endpoints E, the set of fragments F that have been replicated by at least one endpoint in E, a fragment mapping *endpoints()*, a containment mapping $\sqsubseteq$.

The Source Selection Problem with Fragment Replication (SSP-FR) is to assign to each triple pattern in Q, the set of endpoints from E that need to be contacted to answer Q. A solution of SSP-FR corresponds to a mapping D that satisfies the following properties:

1. **Answer completeness preservation:** sources selected in D do not reduce the query engine answer completeness.
2. **Data redundancy minimization:** *cardinality(D(tp))* is minimized for all triple pattern *tp* in Q, i.e., redundant data is minimized.
3. **Data transfer minimization:** executing the query using the sources selected in D minimizes the number of transferred data.

We illustrate SSP-FR on running query Q of Figure 3. Table 1a presents relevant fragments for each triple pattern. Table 1b shows three $D(tp)$ that ensure the completeness preservation property. It may seem counterintuitive that these three *D(tp)* do ensure the completeness preservation property, as they do not include existing DBpedia triples for *dbo:nationality* predicate with object different from *dbr:France* and *dbr:United_Kingdom*, but as they are not included in endpoints in E, these triples are inaccessible to the federation. Even if D_1 and D_2 minimize the number of selected endpoints per triple pattern, only D_2 minimizes the transferred data. Indeed, executing $tp1, tp2, tp3$ against replicated fragments that are located in the same data consumer endpoint will greatly reduce the size of intermediate results.

The approach proposed by Saleem et al. [20] is not designed for solving SSP-FR. Indeed, it does not take into account replicated data, and may produce a solution as D_1. The Fedra algorithm exploits properties of the replicated fragments and is able to find solution D_2.

4 FEDRA: an Algorithm for SSP-FR

The goal of Fedra is to reduce data transfer by taking advantage of the replication of relevant fragments for several triple patterns on the same endpoint.

Algorithm 1 FEDRA Source Selection algorithm

Require: Q: SPARQL Query; F: set of Fragments; endpoints : Fragment → set of Endpoint; ⊑ : TriplePattern × TriplePattern
Ensure: selectedEndpoints: map from TriplePattern to set of Endpoint.
1: **function** SOURCESELECTION(Q,F,endpoints,⊑)
2: triplePatterns ← get triple patterns in Q
3: R, E ← ∅, ∅
4: **for each** tp ∈ triplePatterns **do**
5: R(tp) ← RELEVANTFRAGMENTS(tp, F) ▷ Relevant fragments as in Definition 3
6: R(tp) ← $\{\{f : f \in R(tp) : tp \sqsubseteq f\}\} \bigcup \{\{f\} : f \in R(tp) : f \sqsubseteq tp \wedge \neg(\exists g : g \in R(tp) : f \sqsubset g \sqsubseteq tp)\}$
7: E(tp) ← { (⋃ endpoints(f) : f ∈ fs) : fs ∈ R(tp) }
8: basicGP ← get basic graph patterns in Q
9: **for each** bgp ∈ basicGP **do**
10: UNIONREDUCTION(bgp, E) ▷ endpoints reduction for multiple fragments triples
11: BGPREDUCTION(bgp, E) ▷ endpoints reduction for the bgp triples
12: **for each** (tp, E(tp)) ∈ E **do**
13: selectedEndpoints(tp) ← for each set in E(tp) include one element
14: **return** selectedEndpoints

Algorithm 2 Union reduction algorithm

Require: tps : set of TriplePattern; E : mapping from TriplePattern to set of set of Endpoint
15: **procedure** UNIONREDUCTION(tps, E)
16: triplesWithMultipleFragments ← { tp : tp ∈ tps ∧ cardinality(E(tp)) > 1 }
17: **for each** tp ∈ triplesWithMultipleFragments **do**
18: commonSources ← (⋂ f : f ∈ E(tp)) ▷ get sources in all subsets in E(tp)
19: **if** commonSources ≠ ∅ **then**
20: E(tp) ← { commonSources }

Algorithm 1 proceeds in four main steps: I. Identify relevant fragments for triple patterns, a Basic Graph Pattern (BGP) triple pattern can be contained in one fragment or a union of fragments (lines 5-6). II. Localize relevant replicated fragments on the endpoints, e.g., Figure 4 (line 7). III. Prune endpoints for the unions (line 10). IV. Prune endpoints for the BGPs using a set covering heuristic (line 11).

Next, we illustrate how Algorithm 1 works on our running query Q and data consumer endpoints $C1, C2, C3$ from Figure 3.[4]

First, for each triple pattern, FEDRA computes relevant fragments in *R(tp)*, and groups them if they provide the same relevant data. For tp1, $R(tp1) \rightarrow \{\{f6\}, \{f7\}\}$. For tp4, as $f5 \sqsubseteq f4$, $f5$ is safely removed at line 6, and $R(tp4) \rightarrow \{\{f4\}\}$. Second, FEDRA localizes fragments on endpoints in $E(tp)$. For tp1, $E(tp1) \rightarrow \{\{C1\}, \{C2\}\}$. For tp4, $E(tp4) \rightarrow \{\{C1, C3\}\}$. Figure 4 shows the execution plans encoded in *R(tp)* and *E(tp)*. Triple patterns like tp1, with more than one relevant fragment, represent unions in the execution plan.

Procedure UNIONREDUCTION (cf. Algorithm 2) prunes non common endpoints, if possible, to access triple patterns from as few endpoints as possible. In our running example, it is not possible because there is no common endpoint

[4] As DBpedia is not included in the federation for processing Q, only fragments $f6$ and $f7$ are available to retrieve data for tp_1 and the engine will not produce all the answers that would be produced using DBpedia.

Algorithm 3 Basic graph pattern reduction algorithm

Require: tps : set of TriplePattern; E : mapping from TriplePattern to set of set of Endpoint
21: **procedure** BGPREDUCTION(tps, E)
22: triplesWithOneFragment ← { tp : tp ∈ tps ∧ cardinality(E(tp)) = 1 }
23: (S, C) ← minimal set covering instance using triplesWithOneFragment◁E
24: C' ← MINIMALSETCOVERING(S, C)
25: selected ← get endpoints encoded by C'
26: **for each** tp ∈ triplesWithOneFragment **do**
27: E(tp) ← E(tp) ⋂ selected

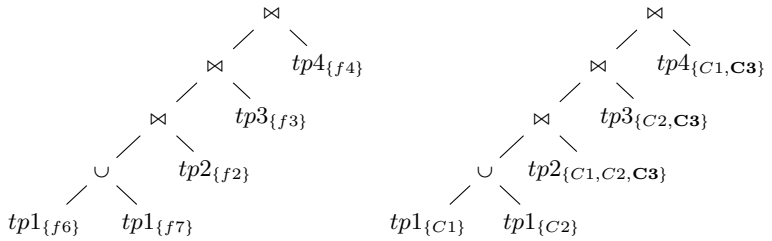

Fig. 4: Execution plan encoded in data structures R (left) and E (right); multiple subsets represent union of different fragment (ex. {f6}, {f7}); elements of the subset represent alternative location of fragments (ex. {C1,C3}); bold sources are the selected sources after set covering is used to reduce number of selected sources

that replicates both $f6$ and $f7$. However, if, for example, $f7$ were also replicated at $C1$, then only $C1$ would be selected to execute tp1.

Procedure BGPREDUCTION (cf. Algorithm 3) transforms the join part of $E(tp)$ (cf. Figure 4) into a set covering problem (cf. line 23). Each triple pattern is an element of the set to cover, e.g., tp2, tp3, tp4 correspond to s2, s3, s4 (cf. Figure 5a). And for each endpoint in $E(tp)$, we include the subset of triple patterns associated with that endpoint, e.g., for endpoint C1 we include the subset {s2,s4} as relevant fragments tp2 and tp4 are replicated by C1 (cf. Figure 5b). Line 24 relies on an existing heuristic [14] to find the minimum set covering. In our example, it computes *C'*={{*s2,s3,s4*}}. Line 25 computes the selected endpoints, in our example, selected={ C3 }.

Finally, (Algorithm 1, line 13) chooses among endpoints that provide the same fragment and reduces data redundancy. For query Q, the whole algorithm returns D_2 of Table 1b.

Proposition 1. *Algorithm 1 has a time complexity of* $O(n.m^2)$*, with* n *the number of triple patterns in the query,* m *the number of fragments,* k *the number of endpoints,* l *the number of basic graph patterns in the query, and* $m \gg k \wedge k \gg l$ *holds.*

The upper bound given in Proposition 1 is unlikely to be reached, as it requires for all fragments to be relevant for each of the triple patterns. In practice

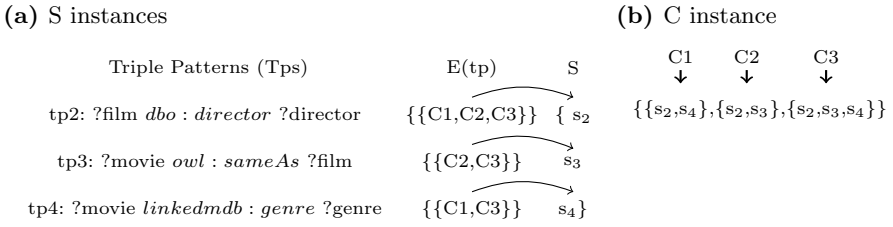

Fig. 5: Set covering instances of S and C of BGP reduction Algorithm 3 for the query Q (Figure 3)

Table 2: Dataset characteristics: version, number of different triples (# DT), and predicates (# P)

Dataset	Version date	# DT	# P
Diseasome	19/10/2012	72,445	19
Semantic Web Dog Food	08/11/2012	198,797	147
DBpedia Geo-coordinates	06/2012	1,900,004	4
LinkedMDB	18/05/2010	3,579,610	148
WatDiv1	-	104,532	86
WatDiv100	-	10,934,518	86

(e.g., experiments from Section 5), even for high number of fragments (> 450), the source selection time remains low (< 2 secs).

Theorem 1 *If all the RDF data accessible through the endpoints of a federation are described as replicated fragments,* Fedra *source selection does not reduce query engine answer completeness.*

5 Experimental Study

The goal of the experimental study is to evaluate the effectiveness of Fedra. We compare the performance of federated SPARQL queries using FedX, DAW+FedX, Fedra+FedX, ANAPSID, DAW+ANAPSID, and Fedra+ANAPSID.

We expect to see that Fedra selects less sources than DAW, and transfers less data from endpoints to the query engines.

Datasets and Queries: We use the real datasets: Diseasome, Semantic Web Dog Food, LinkedMDB, and DBpedia Geo-coordinates. Further, we consider two instances of the Waterloo SPARQL Diversity Test Suite (WatDiv) synthetic dataset [2, 3] with 10^5 and 10^7 triples. Table 2 shows the characteristics of these datasets. The datasets are hosted on local Linked Data Fragment (LDF) servers.

We generate 50,000 queries from 500 templates for the WatDiv federation. We remove the queries that caused engines to abort execution, and queries that returned zero results. For the real datasets, we generate more than 10,000 queries using PATH and STAR shaped templates with two to eight triple patterns, that are instantiated with random values from the datasets. We include the

DISTINCT modifier in all the queries, in order to make them susceptible to a reduction in the set of selected sources without changing the query answer.

For each dataset, we setup a ten consumer SPARQL endpoint federation (ten as in [20]). A consumer SPARQL endpoint is implemented using Jena Fuseki 1.1.1[5]. Each consumer endpoint selects 100 random queries. Each triple pattern of the query is executed as a SPARQL construct query with the LDF client[6]. The results are stored locally if not present in at least three consumer endpoints and a fragment definition is created. This replication factor of three was set to avoid federations where all the fragments were replicated by all the endpoints.

In order to measure the number of transferred data, the federated query engine accesses data consumer endpoints through a proxy.

Implementations: FedX 3.0[7] and ANAPSID[8] have been modified to call FEDRA and DAW [20] source selection strategies during query processing. Thus, each engine can use the selected sources to perform its own optimization strategies. FEDRA and DAW[9] are implemented in both Java 1.7 and Python 2.7.3. Thus, FEDRA and DAW are integrated in FedX (Java) and ANAPSID (Python), reducing the performance impact of including these new source selection strategies. Proxies are implemented in Java 1.7. using the Apache HttpComponents Client library 4.3.5[10]. We used R[11] to compute the Wilcoxon signed rank test [24].

Evaluation Metrics: *i*) *Number of Selected Sources (NSS):* is the sum of the number of sources that have been selected per triple pattern. *ii*) *Number of Transferred Tuples (NTT):* is the number of tuples transferred from all the endpoints to the query engine during a query execution.

Further informations (implementation, results, setups details, tests p-values) are available at `https://sites.google.com/site/fedrasourceselection`.

5.1 Data Redundancy Minimization

To measure the reduction of the number of selected sources, 100 queries were randomly chosen, and the source selection was performed for these queries for each federation using ANAPSID and FedX with and without FEDRA or DAW. For each query, the sum of the number of selected sources per triple pattern was computed. Boxplots are used to present the results (Figure 6). Both FEDRA and DAW significantly reduce the number of selected sources, however, the reduction achieved by FEDRA is greater than the achieved by DAW.

To confirm it, we formulated the null hypothesis: "FEDRA selects the same number of sources as DAW does", and performed a Wilcoxon signed rank test, p-values were inferior or equal to 1.4e-05 for all federations and engines. These

[5] `http://jena.apache.org/`, January 2015.
[6] https://github.com/LinkedDataFragments, March 2015.
[7] `http://www.fluidops.com/fedx/`, September 2014.
[8] `https://github.com/anapsid/anapsid`, September 2014.
[9] We had to implement DAW as its code is not available.
[10] `https://hc.apache.org/`, October 2014.
[11] http://www.r-project.org/

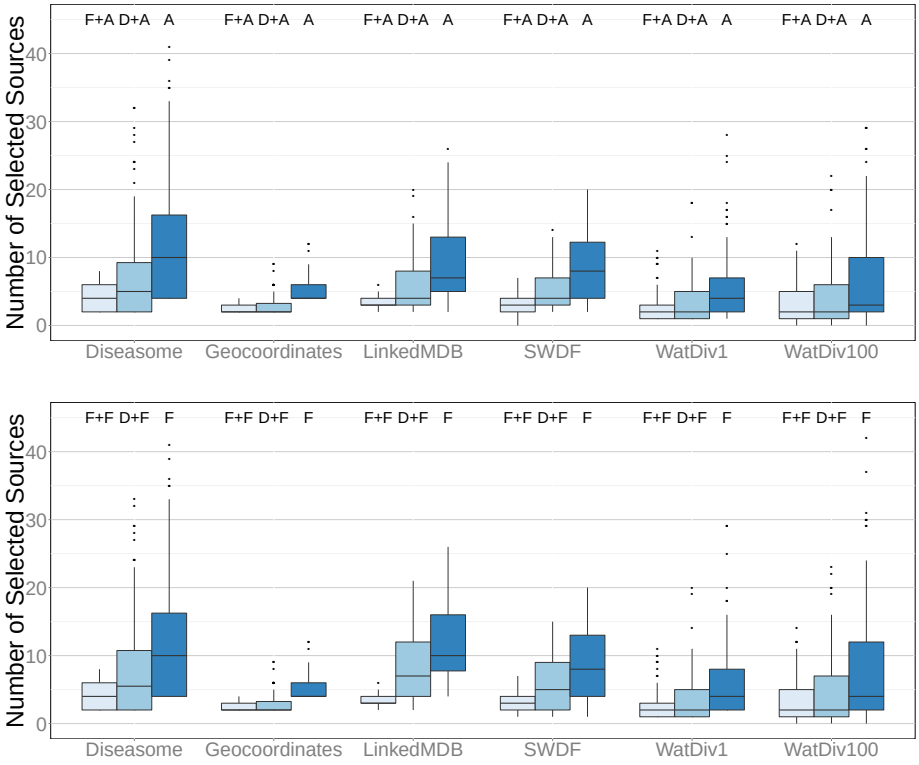

Fig. 6: Number of Selected Sources for execution of ANAPSID (A) and FedX (F) using Fedra (F+), DAW (D+), and the engine source selection

low p-values allow for rejecting the null hypothesis that DAW and Fedra reduction are similar, and accepting the alternative hypothesis that Fedra reduction is greater than the one achieved by DAW. Fedra source selection strategy identifies the relevant fragments and endpoints that provide the same data. Only one of them is actually selected; in consequence, a huge reduction on the number of selected sources of up to 400% per query is achieved.

5.2 Data Transfer Minimization

To measure the reduction in the number of transferred tuples, queries were executed using proxies that measure the number of transmitted tuples from endpoints to the engines. Because queries that timed out have no significance on number of transferred tuples, we removed all these queries from the study.[12] Results (Figure 7) show that Fedra source selection strategy leads to executions

[12] Up to six queries out of 100 queries did not successfully finish in 1,800 seconds, details available at the web page.

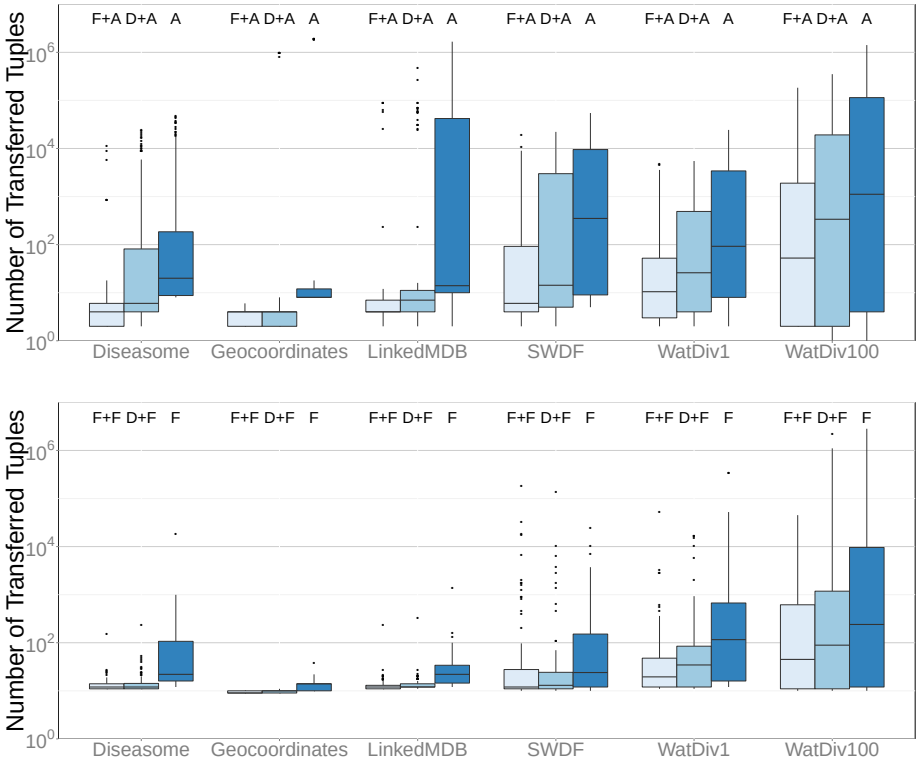

Fig. 7: Number of Transferred Tuples during execution with ANAPSID (A) and FedX (F) using Fedra (F+), DAW (D+), and the engine source selection

with considerably less intermediate results in all the federations except in the SWDF federation. In some queries of the SWDF federation, Fedra+FedX sends exclusive groups that include BGPs with triple patterns that do not share a variable, i.e., BGPs with Cartesian products; in presence of Cartesian product, large intermediate results may be generated. Queries with Cartesian products counters Fedra positive impact over other queries.

Despite that, globally Fedra shows an effective reduction of the number of transferred tuples. To confirm it, we formulated the null hypothesis: "using sources selected by Fedra leads to transfer the same number of tuples as using sources selected by DAW"; and performed a Wilcoxon signed rank test, p-values were inferior or equal to 0.002 for all federations and engines except SWDF federation + FedX engine. In consequence, for all combinations of federation and engines except SWDF+FedX, we can reject the null hypothesis DAW and Fedra number of transferred tuples are similar and accept the alternative hypothesis that Fedra achieves a greater reduction of the number of transferred tuples than DAW. The reduction of the number of transferred tuples is mainly due to Fedra source selection strategy aims to find opportunities to execute joins in

the endpoints, and mostly, it leads to a significant reduction of the intermediate results size of up to four orders of magnitude.

6 Related Work

In distributed databases, data fragmentation and replication improve data availability and query performance [17]. Data fragmentation is tailored for representative queries; fragments are smartly allocated and replicated across servers for balancing workload and reducing size of intermediate results. Linked Data [7] is intrinsically a federation of autonomous participants where federated queries are unknown to a single participant, and a tight coordination of data providers is difficult to achieve. Consequently, federated query engines cannot rely on properties ensured by an allocation algorithm. SSP-FR challenge is to best use fragment localities to reduce intermediate results in a given federation.

Recently, the Linked Data fragments approach (LDF) [22, 23] proposes to improve Linked Data availability by moving query execution load from servers to clients. A client is able to execute locally a restricted SPARQL query by downloading fragments required to execute the query from an LDF server through a simple HTTP request. This strategy allows clients to cache fragments locally and decreases the load on the LDF server. LDF chooses a clear tradeoff by shifting query processing to clients, at the cost of slower query execution. In experiments, we present how to federate several SPARQL consumer endpoints that replicate fragments from LDF servers. Re-organizing fragments on data consumers opens the opportunity to process federated queries even with LDF servers.

Col-graph [13] enables data consumers to materialize triple pattern fragments and to expose them through SPARQL endpoints to improve data quality. A data consumer can update her local fragments and share updates with data providers and consumers. Col-graph proposes a coordination free protocol to maintain the consistency of replicated fragments. Currently, Fedra can process federated queries over Col-graph collaboration networks if the topology of Col-graph is restricted to two layers without cycles. Fedra does not yet consider divergence between fragments produced by concurrent editing, but it is addressed in [16].

HiBISCuS [19] source selection approach has been proposed to reduce the number of selected sources. The reduction is achieved by annotating sources with their authority URIs, and pruning sources that cannot have triples that match any of the query triple patterns. HiBISCuS differs from our aim of both selecting sources that are required to the answer, and avoiding the selection of sources that only provide redundant replicated fragments. While not directly related to replication, HiBISCuS index could be used in conjunction with Fedra to perform join-aware source selection in presence of replicated fragments.

Recently, QBB [12] and DAW [20] propose duplicate-aware strategies for selecting sources for federated query engines. Both approaches use sketches to estimate the overlapping among sources. DAW uses a combination of Min-Wise Independent Permutations (MIPs) [8], and triple selectivity information to estimate the overlap between the results of different sources. Based on how many new

query results are expected to be found, sources that are below predefined benefits, are discarded and not selected. Compared to DAW, FEDRA does not require to compute data summaries because FEDRA relies on fragment definitions and fragment containment to manage replication. Computing containments based on fragment descriptions is less expensive than computing data summaries; moreover, data updates are more frequent than fragment description updates. FEDRA minimizes the number of endpoints and data transfer and produces complete query answers. Consequently, if DAW and FEDRA could find the same number of sources to execute a query, FEDRA source selection considers the query basic graph patterns to delegate join execution to the endpoints and reduce intermediate results size. This key feature cannot be achieved by DAW as it performs source selection only at the triple pattern level.

7 Conclusions

In this paper, we illustrated how replicating fragments allow for data re-organization from different data sources to better fit query needs of data consumers. Then, we proposed a replication-aware federated query engine by extending state-of-art federated query engine ANAPSID and FedX with FEDRA, a source selection strategy that approximates SSP-FR.

FEDRA exploits fragment localities to reduce intermediate results. Experimental results demonstrate that FEDRA achieves significant reduction of intermediate results while preserving query answer completeness.

This work opens several perspectives. First, we made the assumption that replicated fragments are perfectly synchronized and cannot be updated. We can leverage this assumption and manage the problem of federated query processing with divergence [16].

Several variants of SSP-FR can also be developed. SSP-FR does not differentiate between endpoints and the cost of accessing endpoints is considered the same. Finally, SSP-FR and FEDRA can be extended to solve the source selection problem where the number of public endpoint accesses is minimized [16].

References

1. Acosta, M., Vidal, M., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In: Aroyo et al. [5], pp. 18–34
2. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In: Mika et al. [15], pp. 197–212
3. Aluç, G., Ozsu, M.T., Daudjee, K., Hartig, O.: chameleon-db: a Workload-Aware Robust RDF Data Management System. University of Waterloo, Tech. Rep. CS-2013-10 (2013)
4. Aranda, C.B., Hogan, A., Umbrich, J., Vandenbussche, P.: SPARQL Web-Querying Infrastructure: Ready for Action? In: Alani, H., et al. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 277–293. Springer (2013)
5. Aroyo, L., et al. (eds.): ISWC 2011, Part I, LNCS, vol. 7031. Springer (2011)

6. Basca, C., Bernstein, A.: Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In: Polleres, A., Chen, H. (eds.) ISWC Posters&Demos. CEUR Workshop Proceedings, vol. 658. CEUR-WS.org (2010)
7. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)
8. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-Wise Independent Permutations. J. Comput. Syst. Sci. 60(3), 630–659 (2000)
9. Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Hartig, O., Harth, A., Sequeda, J. (eds.) COLD (2011)
10. Gutierrez, C., Hurtado, C.A., Mendelzon, A.O., Pérez, J.: Foundations of Semantic Web databases. J. Comput. Syst. Sci. 77(3), 520–541 (2011)
11. Halevy, A.Y.: Answering queries using views: A survey. VLDB J. 10(4), 270–294 (2001)
12. Hose, K., Schenkel, R.: Towards benefit-based RDF source selection for SPARQL queries. In: Virgilio, R.D., Giunchiglia, F., Tanca, L. (eds.) SWIM. p. 2. ACM (2012)
13. Ibáñez, L.D., Skaf-Molli, H., Molli, P., Corby, O.: Col-Graph: Towards Writable and Scalable Linked Open Data. In: Mika et al. [15], pp. 325–340
14. Johnson, D.S.: Approximation Algorithms for Combinatorial Problems. In: Aho, A.V., et al. (eds.) ACM Symposium on Theory of Computing. pp. 38–49. ACM (1973)
15. Mika, P., et al. (eds.): ISWC 2014, Part I, LNCS, vol. 8796. Springer (2014)
16. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Fedra: Query Processing for SPARQL Federations with Divergence. Tech. rep., Université de Nantes (May 2014)
17. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer (2011)
18. Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: Bechhofer, S., et al. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer (2008)
19. Saleem, M., Ngomo, A.N.: HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In: Presutti, V., et al. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 176–191. Springer (2014)
20. Saleem, M., Ngomo, A.C.N., Parreira, J.X., Deus, H.F., Hauswirth, M.: DAW: Duplicate-AWare Federated Query Processing over the Web of Data. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 574–590. Springer (2013)
21. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: Aroyo et al. [5], pp. 601–616
22. Verborgh, R., Hartig, O., Meester, B.D., Haesendonck, G., Vocht, L.D., Sande, M.V., Cyganiak, R., Colpaert, P., Mannens, E., de Walle, R.V.: Querying Datasets on the Web with High Availability. In: Mika et al. [15], pp. 180–196
23. Verborgh, R., Sande, M.V., Colpaert, P., Coppens, S., Mannens, E., de Walle, R.V.: Web-Scale Querying through Linked Data Fragments. In: Bizer, C., et al. (eds.) WWW Workshop on LDOW 2014. CEUR Workshop Proceedings, vol. 1184. CEUR-WS.org (2014)
24. Wilcoxon, F.: Individual comparisons by ranking methods. In: Breakthroughs in Statistics, pp. 196–202. Springer (1992)