Efficient Query Processing for SPARQL Federations with Replicated Fragments

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

[1] LINA– Nantes University, France {first.last}@univ-nantes.fr
[2] Universidad Simón Bolívar, Venezuela mvidal@ldc.usb.ve

Abstract. Low reliability and availability of public SPARQL endpoints prevent real-world applications from exploiting all the potential of these querying infrastructures. Fragmenting data on servers can improve data availability but degrades performance. Replicating fragments can offer new tradeoff between performance and availability. We propose FEDRA, a framework for querying Linked Data that takes advantage of client-side data replication, and performs a source selection algorithm that aims to reduce the number of selected public SPARQL endpoints, execution time, and intermediate results. FEDRA has been implemented on the state-of-the-art query engines ANAPSID and FedX, and empirically evaluated on a variety of real-world datasets.

Keywords: SPARQL Federation, Replicated Fragments, Source Selection

1 Introduction

Linked Data [4] provides millions of triples for data consumers, however, recent studies suggest that data availability is currently the main bottleneck to the success of the Semantic Web as a viable technology [2, 19]. Particularly, it has been reported by Buil-Aranda et el. [2] that only one third of the 427 public SPARQL endpoints have an availability rate equal or greater than 99%; representing this limitation the major obstacle to developing Web real-world applications that access Linked Data by using these infrastructures. Recently, the Linked Data Fragments (LDF) approach [20, 19] has addressed availability issues by delegating query processing to clients, and by transforming public endpoints into simple HTTP-based triple-pattern fragments providers that can be easily cached by clients. This tradeoff effectively improves data availability, but it can significantly degrade performance [20]. However, we speculate that fragments caching approaches like LDF can have two important consequences for consuming Linked Data: (i) Each client is able to process SPARQL queries on replicated fragments cached from servers. Consequently, if clients are ready to cooperate, the cost of executing SPARQL queries can be significantly reduced, and a new compromise between availability and performance can be achieved. (ii) Potentially, clients could cache triple-pattern fragments from different data providers creating new localities for federated queries. Therefore, some joins can be now executed on one machine without contacting the public endpoints. This vision is also clearly proposed by Ibañez [9], where triple-pattern fragments can be replicated from SPARQL public endpoints, modified

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

Table 1: Execution time and results for the same query over one and two replicas of DBpedia for FedX and ANAPSID

locally, and made available through consumer data endpoints. Approaches in [9, 20] demonstrate how SPARQL processing resources and simple triple-based fragments can be obtained from data consumers. We believe this represents a new opportunity for federated query processing engines to improve SPARQL query processing performance by taking advantage of opportunistic replication and SPARQL processing offered by data consumers.

However, current SPARQL federated query engines [1, 3, 7, 14, 17] may exhibit poor performance in presence of replication. As presented in Figure 1, we duplicated DBpedia and executed a three triple pattern query against one instance and next two instances of DBpedia. We can observe that the performance in terms of execution time and number of results is seriously degraded. This problem has been partially addressed by recent duplicate-aware source selection strategies [8, 15]. The proposed solutions rely on summary of datasets to detect overlapping and do not consider fragments [9, 19, 20]. With fragments, replication is defined declaratively and does not need to be detected.

In this paper, we propose FEDRA, a source selection strategy that exploits fragment definition to select non-redundant data sources. In contrast to [8, 15], FEDRA does not require information about the content of the data sources to detect overlapping. FEDRA just relies on knowledge about the endpoint replicated fragments to reduce the number of endpoints to be contacted, and delegate join execution to endpoints. FEDRA implements a set covering heuristic [6] to minimize the number of sources to be contacted. The implemented source selection approach ensures that triple patterns in the same basic graph pattern are assigned to the same endpoints, and consequently, it reduces the size of intermediate results. We extend the state-of-the art federated query engines FedX [17] and ANAPSID [1] with FEDRA, and compare these extensions with the original engines. We empirically study these engines and the results suggest that FEDRA efficiently reduces the number of public, replicated endpoints, and intermediate results. The paper is organized as follows: Section 2 presents related works. Section 3 describes FEDRA and the source selection algorithm. Section 4 reports our experimental results. Finally, conclusions and future works are outlined in Section 5.

2 Related Work

In distributed databases, data fragmentation and replication improve data availability and query performance [12]. Linked Data [4] is intrinsically a federation of autonomous participants where federated queries are unknown from a single participant, and a tight coordination of data providers is difficult to achieve. This makes data fragmentation [12] and distributed query processing [10] of distributed databases not a viable solution [18] for Linked Data.

Recently, the Linked Data fragments approach (LDF) [20, 19] 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. *A Linked Data Fragment of a Linked Data dataset is a resource consisting of the dataset triples that match a specific selector*. A triple pattern fragment is a special kind of fragments where the selector is a triple pattern; a triple pattern fragment minimizes the effort of the server to produce the fragments. LDF chose a clear tradeoff by shifting query processing to clients, at the cost of slower query execution [19]. On the other hand, LDF could create many data consumers resources that hold replicated fragments in their cache and is able to process SPARQL queries. This opens the opportunity to use these new resources to process SPARQL federated queries. FEDRA aims to improve source selection algorithm of federated query engine to consider these new endpoints, and decreases the load on public endpoints.

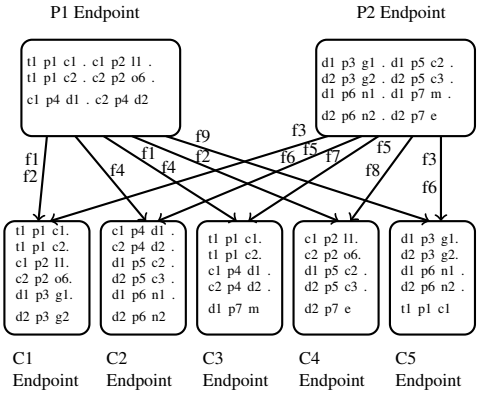
Col-graph [9] enables data consumers to materialize triple pattern fragments and to expose them through SPARQL endpoints to improve data quality. A data consumer can update her local fragments and share updates with data providers and consumers. Col-graph proposes a coordination free protocol to maintain the consistency of replicated fragments. Compared to LDF, Col-graph clearly creates SPARQL endpoints available for other data consumers, and allows federated query engines to use local fragments. As for LDF, FEDRA can take advantage of these data consumer resources.

Recently, HiBISCuS [16] a source selection approach has been proposed to reduce the number of selected sources. The reduction is achieved by annotating sources with the URIs authorities they contain, 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 fragment.

Existing federated query engines [1, 3, 7, 14, 17] are not able to take advantage of replicated fragments, and data overlapping can seriously degrade their performance as reported in Figure 1 and shown in [11, 15]. We integrated FEDRA within FedX and ANAPSID to make existing engines aware of replicated fragments. With FEDRA, replications as in Figure 1 will be detected, and performance will remain stable.

Recently, QBB[8] and DAW[15] 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) [5], 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 below predefined benefits are discarded and not queried.

Compared to DAW, FEDRA does not require to compute data summaries because FEDRA relies on fragment definitions to deduce containments. Computing containments based on fragment descriptions is less expensive than computing data summaries, moreover data updates are more frequent than fragment description updates. FEDRA



(a) Public and Consumer Endpoints Federation

F	Selector	AS	Endpoint
f1	CONSTRUCT WHERE { ?x p1 ?y }	P1	C1,C3
f2	CONSTRUCT WHERE { ?x p2 ?y }	P1	C1, C4
f3	CONSTRUCT WHERE { ?x p3 ?y }	P2	C1, C5
f4	CONSTRUCT WHERE { ?x p4 ?y }	P1	C2, C3
f5	CONSTRUCT WHERE { ?x p5 ?y }	P2	C2, C4
f6	CONSTRUCT WHERE { ?x p6 ?y }	P2	C2, C5
f7	CONSTRUCT WHERE { ?x p7 m }	P2	C3
f8	CONSTRUCT WHERE { ?x p7 e }	P2	C4
f9	CONSTRUCT WHERE { ?x p1 c1 }	P1	C5

(b) Fragments Definitions

Fig. 1: figure (a) describes how fragments of public endpoints are replicated at consumer endpoints. Table (b) describes the selector of each fragment, "authoritative" source (AS) and Endpoint where the fragment is available

also try to remove public endpoints and minimize the number of endpoints to execute a query. Consequently, if DAW and FEDRA could find the same number of sources to execute a query, FEDRA minimizes the number of public endpoints to be contacted. Additionally, 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.

3 Fedra Approach

Fragments and Endpoints Descriptions To define a fragment, we will use the Linked Data Fragment definition given by Verborgh et al. [19]. Let $\mathcal{U}, \mathcal{L}$, and $\mathcal{V}$ denote the set of all URIS, literals and variables, respectively. $\mathcal{T}^* = \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$ is a finite set of blank-node-free RDF triples. Every dataset G published via some kind of fragments on the Web is a finite set of blank-node free triples; *i.e.*, $G \subseteq \mathcal{T}^*$. Any tuple $tp \in (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V} \cup \mathcal{L})$ is a triple pattern.

Definition 1 (Fragment [19]). *A Linked Data Fragment (LDF) of G is a tuple $f = \langle u, s, \Gamma, M, C \rangle$ with the following five elements: i) u is a URI (which is the "authoritative" source from which f can be retrieved); ii) s is a selector; iii) Γ is a set consisting of all subsets of G that match selector s, that is, for every $G' \subseteq G$ it holds that $G' \in \Gamma$ if and only if $G' \in dom(s)$ and $s(G') = true$; iv) M is a finite set of (additional) RDF triples, including triples that represent metadata for f; and v) C is a finite set of controls.*

We restrict fragments to triple pattern fragments as in [9, 20]. Hereafter, we consider that fragments are read-only and data cannot be updated; the fragment synchronization

Listing 1.1: C1 Endpoint Description

```
@prefix sd:<http://www.w3.org/ns/sparql-service-description#>.
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
[] <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> sd:Service ;
 sd:endpoint <http://consumer1/sparql>;
 dcterms:hasPart [
  dc:description "Construct where{ ?x p1 ?y }";
  dcterms:source <http://publicEndpoint1/sparql>; ] ;
 dcterms:hasPart [
  dc:description "Construct where{ ?x p2 ?y }";
  dcterms:source <http://publicEndpoint1/sparql>; ] ;
 dcterms:hasPart [
  dc:description "Construct where{ ?x p3 ?y }";
  dcterms:source <http://publicEndpoint2/sparql>; ] ;
```

Listing 1.2: Queries Q1 and Q2

```
Q1: CONSTRUCT
    where { ?x1 p1 ?x2 }

Q2: CONSTRUCT
    where { ?x1 p4 ?x2 .
            ?x1 p7 ?x3 }
```

problem is studied in [9].
Consider the federation in Figure 1a, where five data consumer endpoints (*C1-C5*) include fragments (*f1-f9*) from public endpoints *P1* and *P2*. Table 1b shows the SPARQL CONSTRUCT query used as a selector for a fragment. Fragments *f1*, *f2*, *f4*, and *f9* have as "authoritative" source *P1*, and fragments *f3* and *f5-f8* have as "authoritative" source *P2*. The last column presents the consumer endpoints where fragments are available. To participate in a FEDRA federation, data consumers annotate each fragment exposed through their endpoints with the fragment selector s and the public endpoint that provides the data u. The vocabulary term *sd:endpoint* refers to the SPARQL endpoint that publishes the fragment. The vocabulary term *dcterms:hasPart* introduces a fragment description, the vocabulary term *dc:description* refers to the SPARQL CONSTRUCT query s, and the vocabulary term *dcterms:source* specifies u, the fragment "authoritative" source URI. Listing 1.1 shows the description of the endpoint *C1* of Figure 1a.

For source selection with replicated fragments, we need to define when a fragment is relevant for answering a query. A fragment is relevant for answering a query, if it is relevant for at least one triple pattern of the query.

Definition 2 (Fragment relevance). *A fragment $f = \langle u, s, \Gamma, M, C\rangle$ is relevant for a triple pattern tp, if the triple pattern evaluated over Γ, $[\![tp]\!]_\Gamma$ [13], is not empty.*

Consider queries Q1 and Q2 (cf. Listing 1.2), and the federation of Figure 1. Fragments *f1* and *f9* are relevant for query *Q1*, while fragments *f4*, *f7*, and *f8* are relevant for query *Q2*. We can define two types of containments: containment between SPARQL endpoints and containment between fragments.

Definition 3 (Endpoint Containment). *Let e1 and e2 be the URI of two SPARQL endpoints that respectively expose fragments $f1 = \langle u1, s, \Gamma1, M1, C1\rangle$ and $f2 = \langle u2$, $s, \Gamma2, M2, C2\rangle$ such that f1 and f2 have the same selector s and the same "authoritative" source ($u2 = u1$). Then all triples in f2 are contained in f1 and vice versa, i.e., $\Gamma2 \subseteq \Gamma1 \wedge \Gamma1 \subseteq \Gamma2$. And we use the notation e1 $\subseteq_s$ e2, and e2 $\subseteq_s$ e1 to represent the endpoint containment relationship.*

f1 triples in consumers *C1* and *C3* from Figure 1 are the same (from Definition 3): $C1 \subseteq_s C3$ and $C3 \subseteq_s C1$ where s is $f1$ selector. Endpoint containment can be used to reduce the number of endpoints to contact to answer a query. *f1* and *f9* are relevant for query *Q1*, from the endpoint containments *f1* data is also available through *P1*, *C1*, and *C3* and *f9* data is also available through *P1* and *C5*. Therefore, only one endpoint per fragment needs to be selected to answer the query. A good choice could be *C1*,

```
SELECT DISTINCT *
WHERE {
{?x1p1?x2.?x2p4?x3}
UNION
{?x1p2?x2.?x2p5?x3}
UNION
{?x1p3?x2.?x2p6?x3}
}
```

(a) Query Q3

TP	RF	Endpoints
?x1 p1 ?x2	f1	C1, C3, P1
	f9	C5, P1
?x2 p4 ?x3	f4	C2, C3, P1
?x1 p2 ?x2	f2	C1, C4, P1
?x2 p5 ?x3	f5	C2, C4, P2
?x1 p3 ?x2	f3	C1, C5, P2
?x2 p6 ?x3	f6	C2, C5, P2

(b) RF for *Q3*

tp	D_1(tp)	D_2(tp)
?x1 p1 ?x2	{ C1 }	{ C3 }
?x2 p4 ?x3	{ C2 }	{ C3 }
?x1 p2 ?x2	{ C1 }	{ C4 }
?x2 p5 ?x3	{ C2 }	{ C4 }
?x1 p3 ?x2	{ C1 }	{ C5 }
?x2 p6 ?x3	{ C2 }	{ C5 }

(c) Conditions 1-3

BGP	tp	D_2(tp)
BGP1	?x1 p1 ?x2	{ C3 }
	?x2 p4 ?x3	{ C3 }
BGP2	?x1 p2 ?x2	{ C4 }
	?x2 p5 ?x3	{ C4 }
BGP3	?x1 p3 ?x2	{ C5 }
	?x2 p6 ?x3	{ C5 }

(d) Conditions 1-4

Fig. 2: SSP solutions for query Q3: (a): Query Q3, (b):Relevant fragments for Q3, (c): Maps D_1 and D_2 that satisfy Source Selection Problem (SSP) conditions 1-3., (d): Map D_2 that satisfies Source Selection Problem (SSP) conditions 1-4

C5 or *C3*, *C5*; this will reduce the load of the public endpoint *P1* and will improve *P1* availability. By contacting only *C1*, *C5* or *C3*, *C5*, complete query answers are obtained because the triple pattern fragment defines a copy of the data source using the fragment selector. Another type of containment that allows to reduce the number of sources to be contacted is defined based on the fragment selector.

Definition 4 (Fragment Containment). *Let* $f1 = \langle u, s1, \Gamma 1, M1, C1 \rangle$ *and* $f2 = \langle u, s2, \Gamma 2, M2, C2 \rangle$ *be two fragments that share the same "authoritative" source* u*, and a triple pattern* tp*. If for all possible values of* $\Gamma 1$ *and* $\Gamma 2$*, always the triples in* $\Gamma 1$ *that match* tp *are also in* $\Gamma 2$*, i.e.,* $[\![tp]\!]_{\Gamma 1} \subseteq [\![tp]\!]_{\Gamma 2}$*. Then, regarding* tp*,* $f1$ *is contained in* $f2$*. And we use the notation f1* $\sqsubseteq$ *f2.*

Triples of fragment *f9* replicated at consumer *C5* in Figure 1 are contained in fragment *f1* at *C1* and *C3* (*f9* $\sqsubseteq$ *f1*) because *f1* and *f9* share the same "authoritative" source, and all the triples that match predicate *p1* and object *c1* always matches predicate *p1* in *f1*. Using fragment containment, contacting *C1* or *C3* is enough to answer *Q1*.

Source Selection Problem (SSP) Given a set of SPARQL endpoints *E*, a set of public endpoints *P*, $P \subseteq E$, the set of fragments contained in each endpoint as a function *frags : Endpoint* $\rightarrow$ *set of Fragment*, a containment relation among endpoints (given by Definition 3) for $f \in frags(e_i) \wedge f \in frags(e_j)$, $e_i \subseteq_f e_j$, a containment relation among fragment selectors (given by Definition 4) $f_l \sqsubseteq f_k$, and a SPARQL query *Q*. Find a map *D*, such that for each triple pattern *tp* in *Q*, *D(tp)* $\subseteq$ *E* and: *1*) For each endpoint *e* that may contribute with relevant data to answer query *Q*, *e* is included in *D*, or *D* includes another endpoint that contributes with at least the same relevant data as *e*. *2*) *D(tp)* contains as few public endpoints as possible. *3*) *size(D(tp))* is minimized for all triple pattern *tp* in *Q*. *4*) The number of different endpoints used within each basic graph pattern is minimized.

Condition 1 states that the selected sources will produce an answer as complete as possible given the set of fragments accessible through the endpoints E, but answer may be incomplete if some fragments definitions are missing. Condition 2 ensures that public endpoints availability problem will be avoided whenever is possible. Condition 3 establishes that the number of selected sources is reduced. Condition 4 aims to reduce the size of intermediate results, and to delegate the join execution to endpoints whenever is possible. Even if the public endpoint can provide all the fragments, the use of several

consumer endpoints is preferable. To illustrate these four conditions, consider query *Q3* in Figure 2a and fragments and endpoints of Figure 1. Table 2b shows the relevant fragments for *Q3* triple patterns, and the endpoints that provide these fragments. For example, for the triple pattern *?x1 p1 ?x2*, there are two relevant fragments $f1$ and $f9$. As previously discussed using endpoint containments, contacting *C1* or *C3* is enough to answer this triple pattern without contacting the public endpoint. The maps D_1 and D_2 in Figure 2c satisfy the SSP conditions 1-3: all relevant fragments have been included directly or through containment relation, the number of selected endpoints per triple pattern has been minimized, and no public endpoints has been included in the map. However, only the map D_2 satisfies condition 4, as the number of different endpoints selected per basic graph pattern has been also minimized (see Figure 2d). Then, joins are delegated to the selected endpoints, and the size of intermediate results is reduced. *Current state-of-the-art [15] is triple pattern wise and does not guarantee condition 4*.

Source Selection Algorithm Algorithm 1 sketches the FEDRA source selection algorithm. First, the algorithm pre-selects for each triple pattern in *Q* the sources that can be used to evaluate it (lines 2-29). All the endpoints e and their exposed fragments f are considered (lines 5-27). In line 6, the function *canAnswer()* is used to determine if endpoint e can provide triples from fragment f that matches triple pattern tp. An initial check based on the selector of f and tp is done, and when it is satisfied, a dynamic check using an ASK query is done to ensure that f is relevant for triple pattern tp (as in Definition 2). An ASK query is used to avoid considering fragments that are not relevant for the triple pattern, in the case the triple pattern has constants where the fragment definition has variables. Query $Q3$ relevant endpoints and fragments are given in Table 2b.

The function *subFrag* determines if the data provided by one fragment is also provided by another fragment. This function has as arguments the fragments and endpoints that provide them, and also the containment relationships. For each relevant fragment f, it determines if the considered fragment f in endpoint e provides the same data as already found fragments or if it provides at least the same data as already found fragments or if it provides at most the same data as already found fragments (lines 8-22). Function *subFrag* tests if there is a containment in both senses or only in one of them. Accordingly, the fragment is grouped with the fragments that provide the same data (lines 10-12), some of the already found fragments are not anymore of interest (line 14-15), or it is chosen not to include the fragment (lines 17-18). Between the fragments $f1$ and $f9$ selected as relevant fragments for the first triple pattern of $Q3$, there is a containment $f9 \sqsubseteq f1$, and in consequence, only the fragment $f1$ needs to be selected. Moreover, $f1$ is provided by endpoints $C1$, $C3$, and $P1$, and as they provide the same data, each of them can be selected alone to provide data for this triple pattern, i.e., they offer alternative sources for the same fragment. Table 2 (left) shows the changes in $fragments$ for the first triple pattern of $Q3$. Contrarily, between the fragments $f7$ and $f8$ selected as relevant for the second triple pattern of $Q2$, there is no containment relation, and in consequence, both fragments are selected. Table 2 (right) shows the changes in $fragments$ for the second triple pattern of $Q2$.

When the fragment provides non-redundant data, it is included in the selected fragments (lines 23-24). In the previous examples, it happens once for the first triple pattern

Algorithm 1 Source Selection algorithm

Require: Q: SPARQL Query; E: set of Endpoints; frags : Endpoint → set of Fragment; P : set of Endpoint; $\subseteq_f$: Endpoint × Endpoint; $\sqsubseteq$: BGP × BGP
Ensure: D: map from Triple Pattern to set of Endpoints.

```
1: function SOURCESELECTION(Q,E,frags,P,⊆f,⊆)
2:   triplePatterns ← get triple patterns in Q
3:   for each tp ∈ triplePatterns do
4:     fragments ← ∅
5:     for each e ∈ E ∧ f ∈ frags(e) do
6:       if canAnswer(e, f, tp) then
7:         include ← true
8:         for each fs ∈ fragments do
9:           (f',e') ← take one element of fs
10:          if subFrag(f,e,f',e',⊆f,⊆) ∧ subFrag(f',e',f,e,⊆f,⊆) then
11:            fs.add((f,e))
12:            include ← false
13:          else
14:            if subFrag(f',e',f,e,⊆f,⊆) then
15:              fragments.remove(fs)
16:            else
17:              if subFrag(f,e,f',e',⊆f,⊆) then
18:                include ← false
19:              end if
20:            end if
21:          end if
22:        end for
23:        if include then
24:          fragments.add({(f,e)})
25:        end if
26:      end if
27:    end for
28:    G(tp) ← getEndpoints(fragments)
29:  end for each
30:  basicGP ← get basic graph patterns in Q
31:  for each bgp ∈ basicGP do
32:    (S, C) ← minimal set covering instance using bgp◁G
33:    C' ← minimalSetCovering(S, C)
34:    for each tp ∈ bgp do
35:      G(tp) ← filter G(tp) according to C'
36:    end for each
37:  end for each
38:  for each tp ∈ domain(G) do
39:    D(tp) ← for each set in G(tp) include one element
40:  end for each
41:  return D
42: end function
```

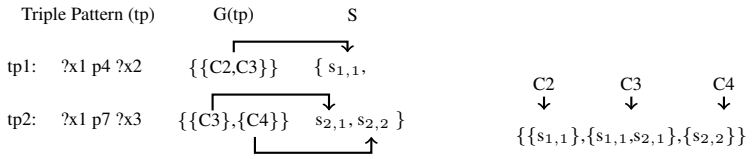
Table 2: $fragments$ changes due to execution of lines 5-27 of Algorithm 1 for the first triple pattern of *Q3* (left) and the second triple pattern of *Q2* (right)

line	fragments
4	{ }
24	{ { (f1, C1) } }
11	{ { (f1, C1), (f1, C3) } }
11	{ { (f1, C1), (f1, C3), (f1, P1) } }

line	fragments
4	{ }
24	{ { (f7, C3) } }
24	{ { (f7, C3) }, { (f8, C4) } }
11	{ { (f7, C3), (f7, P2) }, { (f8, C4) } }
11	{ { (f7, C3), (f7, P2) }, { (f8, C4), (f8, P2) } }

of *Q3* and twice for the second triple pattern of *Q2*. Function *getEndpoints* in line 28 takes the endpoints of *fragments*, and when for one subset *fs* there are endpoints in *E - P*, then all endpoints in *P* are removed. In the previous examples, the public endpoints are removed and the value of *G(?x1 p1 ?x2)* is $\{\{C1,C3\}\}$, and the value of *G(?x1 p7 ?x3)* is $\{\{\ C3\ \},\{C4\}\}$. At the end of the first for loop (lines 3-29), a set whose elements are a set of endpoints and fragments that can be used to evaluate the triple pattern is produced for each triple pattern in the query. All the endpoints in the same set offer the same data for that fragment, and during execution only one of them needs to be contacted. And different elements of this resulting set correspond to different fragments that should be considered in order to obtain an answer as complete as possible, modulo the considered fragments. For queries *Q2* and *Q3*, *C1* and *C3* provide the same data for the first triple pattern of *Q3*, and *C3*, *C4* provide different data for the second triple pattern of *Q2* .

Next, for each basic graph pattern, a general selection takes place, considering the pre-selected sources for each triple pattern in the basic graph pattern. This last part can be reduced to the well-known set covering problem, and an existing heuristic like the



(a) S instance (b) C instance

Fig. 3: Set covering instances of S and C for the query $Q2$ and federation given in Figure 1. For each element in G(tp), one element is included in set S. For each endpoint in G, one set is included in collection C and its elements are the elements of S related to the endpoint

one given in [6] may be used to perform the procedure indicated in line 33. To use the set covering heuristic, an instance of the set covering problem is produced in line 32 using $bgp \lhd G$ [3]. Figure 3a shows *G* values obtained after lines 3-29 loop has ended for query *Q2*, and S instance for the set covering problem. For each set in *G(tp)*, one element is included in *S*, e.g., for set $\{C2,C3\}$, the element $s_{1,1}$ is included in *S*. We have used subscripts *i,j* to denote that the element comes from the triple pattern *i*, and it is the *j*-th element coming from this triple pattern. The collection *C* is composed of one set for each endpoint that is present in *G*, and its elements are the elements of *S* related to each endpoint. Figure 3b shows the instance of *C* for this example. The collection *C'* obtained in line 33 is $\{C3,C4\}$, as the union of *C3* and *C4* is equals to *S*, and there is no smaller subset of *C* that can achieve this. The instruction in line 35 removes from each *G(tp)* the elements that do not belong to *C'*. In the example, *C2* is removed from *G(?x1 p4 ?x2)*. A last step may be performed to choose among endpoints that provide the same fragment and ensure a better query processing by existing federated query engines (lines 38-40). Nevertheless, these alternative sources could be used to speed up query processing, e.g., by getting a part of the answer from each endpoint.

4 Experiments

We conducted many experimentations in different setups to demonstrate the impact of FEDRA on existing approaches, complete results are available at the FEDRA web site[4]. The performance of the FedX and the ANAPSID query engines is our baseline. General results are the comparison of performance of FedX alone, FedX+DAW, FedX+FEDRA, same thing for ANAPSID. Compared to FedX and ANAPSID, FEDRA should reduce selected sources significantly and speed up queries execution time. Compared to DAW, we expect FEDRA to achieve same source reduction but without pre-computed MIPS index, and generate less intermediate results thanks to endpoints reduction and to finding join opportunities.

Datasets, Queries and Federations Benchmark: we used Diseasome, Semantic Web Dog Food, LinkedMDB, and DBpedia geo-coordinates datasets, Table 3a shows char-

[3] $bgp \lhd G$ represents function domain restriction, *i.e.*, it takes the elements of map *G* that relates elements in *bgp* with some set of set of endpoints

[4] `https://sites.google.com/site/fedrasourceselection`

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

(a) Datasets

Dataset	ST	2P	3P	4P	2S	3S
Diseasome	5	4	5	2	5	5
Semantic Web Dog Food	5	7	7	4	5	5
DBpedia Geo-coordinates	5	0	0	0	5	5
LinkedMDB	5	0	0	0	0	0

(b) Queries sizes and number

Dataset	J	OP	U	F(R)	L	OB
Diseasome	19	2	1	11(7)	4	9
Semantic Web Dog Food	24	5	2	8(7)	4	9
DBpedia Geo-coordinates	10	0	0	6(1)	6	6
LinkedMDB	0	0	0	1(1)	1	2

(c) Queries with operators, modifiers

Dataset	# FD	# E1F	# E2F
Diseasome	16	16	120
Semantic Web Dog Food	40	40	780
DBpedia Geo-coordinates	4	4	6
LinkedMDB	4	4	6

(d) Federations

Table 3: Datasets, queries and federations characteristics. For the datasets: the version, number of different triples (# DT), and predicates (# P). For the queries: the number of queries with 1 triple pattern (ST), 2, 3 or 4 triple patterns in star shape (S) or path shape (P). Also the number of queries with: joins (J), optionals (OP), unions (U), filter (F), regex expressions (R), limit (L), and order by (OB). For the federations: the number of fragments definitions (FD), endpoints exposing one and two fragments (E1F, E2F)

acteristics of the evaluated datasets. We studied the datasets and queries used in [15][5]. However, we modified the queries to include the DISTINCT modifier in all the queries. Additionally, the ORDER BY clause was included in the queries with the LIMIT clause, in order to make them susceptible to a reduction in the set of selected sources without changing the query answer, and to ensure a semantically unambiguous query answer. Tables 3b and 3c present queries characteristics. As federations used in [15] do not take into account fragments, they were not reprised. A federation was set up for each dataset; each federation is composed of the triple pattern fragments that are relevant for the studied queries. The federation endpoints offer one or two fragments; endpoints with two fragments offer opportunities to execute joins for the engine. Table 3d shows the federation characteristics. In average, DAW indexes were computed in 2,513 secs, and FEDRA containments in 32 secs.

Notice that as FEDRA containments depends only on fragment descriptions, their updates are less frequent than DAW indexes. Virtuoso 6.1.7[6] endpoints were used. A Virtuoso server was set up to provide all the endpoints as virtual endpoints. It was configured with timeouts of 600 secs. and 100,000 tuples. In order to measure the size of intermediate results, proxies were used to access the endpoints.

Implementations: FedX 3.0[7] and ANAPSID[8] have been modified to call FEDRA and DAW [15] source selection strategies during query processing. Thus, each engine can use the selected sources to perform its own optimization strategies. Because FedX is implemented in Java, while ANAPSID is implemented in Python, FEDRA and DAW[9] were implemented in both Java 1.7 and Python 2.7.3.. Thus, FEDRA and DAW were inte-

[5] They are available at `https://sites.google.com/site/dawfederation`.

[6] `http://virtuoso.openlinksw.com/`, November 2013.

[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.

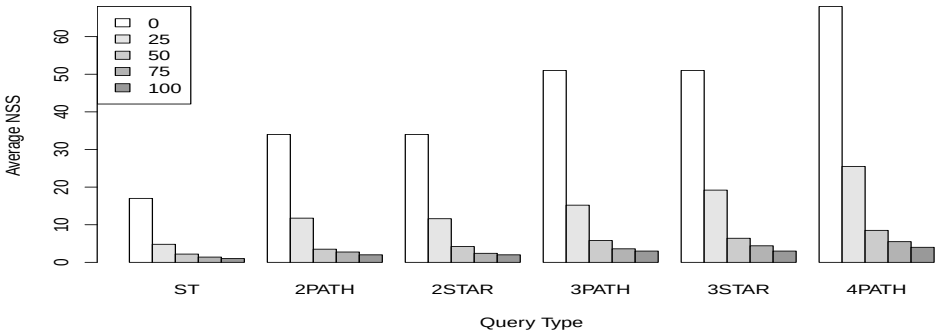


Fig. 4: FedX and FEDRA Number of Selected Sources (NSS) when the percentage of known containments is 0%, 25%, 50%, 75% and 100%, for Diseasome Federation

grated in FedX and ANAPSID, reducing the performance impact of including these new source selection strategies. Proxies were implemented in Java 1.7. using the Apache HttpComponents Client library 4.3.5.[10]

Evaluation Metrics: *i*) *Number of Selected Public Sources (NSPS):* is the sum of the number of public sources that has been selected per triple pattern. *ii*) *Number of Selected Sources (NSS):* is the sum of the number of sources that has been selected per triple pattern. *iii*) *Execution Time (ET):* is the elapsed time since the query is posed by the user and the answers are completely produced. It is detailed in source selection time (SST), and total execution time (TET). Time is expressed in seconds (secs.). A timeout of 300 secs. has been enforced. Time was measured using the bash time command. *iv*) *Intermediate Results (IR):* is the number of tuples transferred from all the endpoints to the query engine during a query evaluation. *v*) *Recall (R):* is the proportion of results obtained by the underlying engine, that are also obtained including proposed strategy.

Impact of the number of containments over FEDRA behavior We study the impact of the number of containments known during the source selection on the number of selected sources and intermediate results size. For each query, the set of known containments have been set to contain 0%, 25%, 50%, 75%, and 100% of the containments concerning the relevant fragments, and queries have been executed in these five configurations. Results show that the number of selected public sources is equal to the number of triple patterns in the queries when no containment is known. However, as soon as some containments are known (25%-100%), this number is reduced to zero. Also, the number of selected sources is considerably reduced when FEDRA source selection strategy is used instead of just using ANAPSID and FedX source selection strategies; see FedX results in Figure 4. ANAPSID results exhibit a similar behavior.

Preservation of the Query Answer The goal of this experiment is to determine the impact of FEDRA source selection strategy on query completeness. Queries were executed using both the ANAPSID and the FedX query engines, and then, we executed the same engines enhanced with the FEDRA source selection strategy. Recall was computed and

[10] `https://hc.apache.org/`, October 2014.

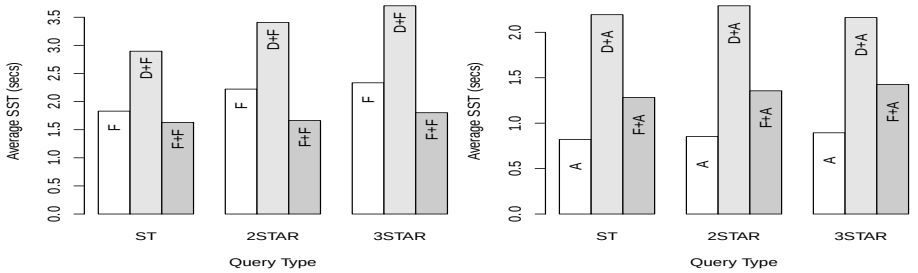


Fig. 5: Source selection time (SST) for Geo-coordinates federation. The FedX (F, left) and the ANAPSID (A, right) query engines are combined with FEDRA (F+F and F+A), and DAW (D+F and D+A)

was 1.0 in the majority of the cases. In few cases the recall was considerably reduced, but these cases correspond to queries with OPTIONAL operator using the FedX query engine, and it was due to an implementation error for this operator. FEDRA only discards relevant sources when relevant fragments data are provided by another source that was already selected. Then, it does not reduce the recall of the answer. Finally, the query engine implementation limitations were also the causes of the reduction of recall.

Source Selection Time To measure the FEDRA and DAW source selection cost, the source selection time using each engine with and without FEDRA or DAW was measured. Results are diverse, for federations with a large number of endpoints like the SWDF federation, the cost of performing the FEDRA source selection is considerably inferior to the cost of contacting all the endpoints using FedX, but similar to the cost of using the ANAPSID source selection strategy. On the other hand, the DAW cost is similar to the FedX cost, and it is considerably superior to the ANAPSID cost. For federations with a small number of endpoints like Geo-coordinates (see Figures 5a and 5b), the cost of performing source selection with FEDRA is less expensive than performing just FedX source selection, but more expensive than performing just ANAPSID source selection strategy. And the cost of performing source selection with DAW is more expensive than performing just FedX or ANAPSID source selection strategy. ANAPSID source selection mostly relies on the endpoints descriptions, and avoids to contact endpoints in most cases. On the other hand, FedX source selection strategy relies on endpoint contacts to determine which endpoints can be used to obtain data for each triple pattern. FEDRA source selection is somewhere in the middle, it does contact all the endpoints that are considered relevant according to their descriptions to confirm that they can provide relevant data, and has the added cost of using the containments to reduce the number of selected sources. DAW source selection does not contact sources, but relies on no negligible cost of operating Min-Wise Independent Permutations (MIPs) in order to determine the overlapping sources.

Execution Time To measure the FEDRA and DAW execution time, queries were executed using each engine with and without FEDRA or DAW. For small federations or queries with only one triple pattern, FEDRA and DAW achieve a similar reduction in

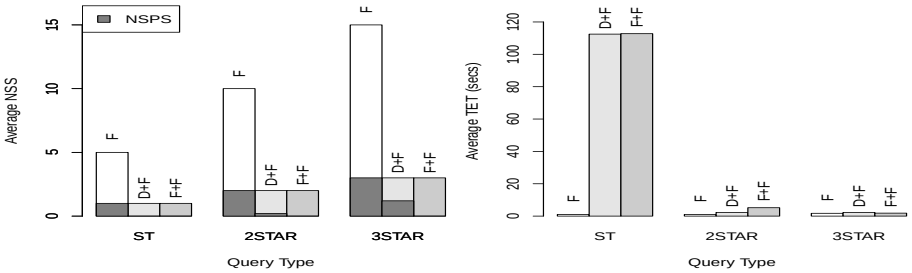
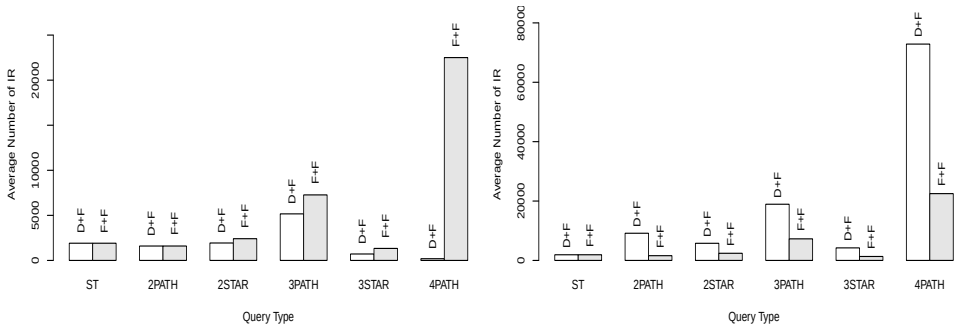


Fig. 6: Number of Selected Sources and Total Execution Time (TET) for Geo-coordinates federation and the FedX (F) query engine. FedX is also combined with FEDRA and DAW source selection strategies (F+F and D+F). For F, 4 out of 5 queries timed out for each query type. For F+F, 4 out of 5 queries timed out for 3STAR queries. For D+F, 4 out of 5 queries timed out for 2STAR and 3STAR queries

execution time. However, for larger federations and queries with more triple patterns, FEDRA reduction is larger than DAW. In all cases, the reduction is considerable when the combination of FEDRA and the query engine is compared to using the engine alone, e.g., Figure 6b shows the results for the Geo-coordinates federation and FedX. For star-shape queries, the use of FEDRA source selection strategy has made the difference between timing out or obtaining answers. For queries with two triple patterns, this difference is important, as FEDRA enhances FedX to obtain answers in just few seconds. The difference in execution time is a direct consequence of the selected sources reduction. Further, executing the joins in the endpoints whenever it is possible, may reduce the size of intermediate results and produce answers sooner.

Reduction of the Number of Selected Sources To measure the reduction of the number of selected sources, the source selection was performed 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, for all the sources and just for the public sources. Figure 6 shows the results for the Geo-coordinates federation and FedX, similar results are observed for the other federations and for ANAPSID. DAW source selection strategy exhibits the same reduction in the total number of selected sources. Consequently, some of the selected public sources are pruned, but as it does not aim to reduce the public sources, it does not achieve a consistent reduction of them. On the other hand, FEDRA has as input the public condition of sources, and as one of its goals is to select as few public sources as possible, it is natural to observe such a reduction consistently for all the query types. FEDRA source selection strategy identifies the relevant fragments and endpoints that provide the same data. Only one of them is actually selected, and in consequence, a huge reduction on the number of selected sources is achieved. Moreover, public endpoints are safely removed from the selected sources as their data can be retrieved from other sources.

Reduction of the Intermediate Results Size To measure the intermediate results size reduction, queries were executed using proxies that measure the number of transmitted



(a) Federation with public endpoint (b) Federation without public endpoint

Fig. 7: Intermediate results size for Diseasome federation and FedX (F). FedX is also combined with FEDRA and DAW source selection strategies (F+F and D+F)

tuples from endpoints to the engines. Additionally, each query was executed against the federation with and without the public endpoint. Figure 7 shows the sizes of intermediate results for the Diseasome federation and FedX combined with FEDRA and DAW; similar results were obtained for the other federations and for ANAPSID. Figure 7a shows that when the public endpoint is part of the federation, DAW source selection strategy leads to executions with considerably less intermediate results.

Figure 7b shows that when the public endpoint is not part of the federation, FEDRA source selection strategy leads to executions with considerably less intermediate results. Since the FEDRA source selection strategy finds opportunities to execute joins in the endpoints, and mostly, it leads to significant reduction in the intermediate results size. These results are consequence of SSP Condition 4, and cannot be systematically achieved by DAW as it is a triple wise based approach. Nevertheless, as DAW source selection does not avoid public endpoints, it may select to execute all triple patterns in the public endpoint, and this comes with a huge reduction of the size of intermediate results. Figure 7b shows that when this "public endpoint" opportunity to execute all triple patterns in one endpoint is removed, DAW source selection strategy does not consistently reduce the intermediate results size.

5 Conclusions and Future Works

Recent works on replicated fragments spread data and SPARQL processing capabilities over data consumers. This opens new opportunities for federated query processing by offering new tradeoffs between availability and performance. We presented FEDRA, a source selection approach that takes advantage of replicated fragment definitions to reduce the use of public endpoints as they are mostly overloaded. FEDRA identifies and uses opportunities to perform join in the sources relieving the query engine of performing them, and reducing the size of intermediate results. Experimental results demonstrate that the number of selected sources remains low even when high number of endpoints and replicated fragments are part of the federations. Results rely on con-

tainments induced by replicated fragment definitions. Next, selecting the same sources for Basic Graph Patterns triples is a strategy that allows to reduce significantly the number of intermediate results. Perspectives include dynamic discovery of endpoints providing replicated fragments. This allows federated query engines to expand at runtime declared federations with consumer endpoints of interests. Such mechanism can improve both data availability and performances of federated queries in Linked Data.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In *ISWC*, pages 18–34, 2011.
2. C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *ISWC (2)*, pages 277–293, 2013.
3. C. Basca and A. Bernstein. Avalanche: Putting the spirit of the web back into semantic web querying. In A. Polleres and H. Chen, editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *IJSWIS*, 5(3):1–22, 2009.
5. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *JCSS*, 60(3):630–659, 2000.
6. S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill, 2008.
7. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In O. Hartig, A. Harth, and J. Sequeda, editors, *COLD*, 2011.
8. K. Hose and R. Schenkel. Towards benefit-based RDF source selection for SPARQL queries. In *SWIM*, page 2, 2012.
9. L.-D. Ibáñez, H. Skaf-Molli, P. Molli, and O. Corby. Col-graph: Towards writable and scalable linked open data. In *ISWC*, 2014.
10. D. Kossmann. The state of the art in distributed query processing. *ACM Computer Survey*, 32(4):422–469, 2000.
11. G. Montoya, H. Skaf-Molli, P. Molli, and M.-E. Vidal. Fedra: Query Processing for SPARQL Federations with Divergence. Technical report, Université de Nantes, May 2014.
12. M. T. Özsu and P. Valduriez. *Principles of distributed database systems*. Springer, 2011.
13. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3), 2009.
14. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *ESWC*, pages 524–538, 2008.
15. M. Saleem, A.-C. N. Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *ISWC*, pages 574–590, 2013.
16. M. Saleem and A. N. Ngomo. Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *ESWC*, pages 176–191, 2014.
17. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, pages 601–616, 2011.
18. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over linked data. *WWW*, 14(5-6):495–544, 2011.
19. R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the Web with high availability. In *ISWC*, 2014.
20. R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. Web-scale querying through Linked Data Fragments. In *LDOW*, 2014.