HAL
archives-ouvertes.fr

Query Processing for SPARQL Federations with Data Replication

Gabriela Montoya, Luis-Daniel Ibanez, Hala Skaf-Molli, Pascal Molli, Maria-Esther Vidal

▶ **To cite this version:**

Gabriela Montoya, Luis-Daniel Ibanez, Hala Skaf-Molli, Pascal Molli, Maria-Esther Vidal. Query Processing for SPARQL Federations with Data Replication. AP. 2014. <hal-00952830>

HAL Id: hal-00952830
http://hal.univ-nantes.fr/hal-00952830

Submitted on 27 Feb 2014

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.

Query Processing for SPARQL Federations with Data Replication

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

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

Abstract. Data replication and deployment of local SPARQL endpoints improve scalability and availability of public SPARQL endpoints, making the consumption of Linked Data a reality. This solution requires synchronization and specific query processing strategies to take advantage of replication. However, existing replication aware techniques in federations of SPARQL endpoints do not consider data dynamicity. We propose Fedra, an approach for querying federations of endpoints that benefits from replication. Participants in Fedra federations can copy fragments of data from several datasets, and describe them using provenance and views. These descriptions enable Fedra to reduce the number of selected endpoints while satisfying user divergence requirements. Experiments on real-world datasets suggest savings of up to three orders of magnitude.
Keywords: Source Selection, SPARQL Endpoints, Data Replication.

1 Introduction

Query limitations imposed by existing public SPARQL endpoints may be too restrictive for real-world applications. Data replication and deployment of local SPARQL endpoints may be a solution to improve data availability and to reduce the workload of public endpoints, and thus, make public endpoints ready for action [2]. However, exposing replicas through local SPARQL endpoints pose two problems. Firstly, querying replicas may produce obsolete answers, whenever Linked Data change[9] and replicas are unsynchronized and diverge with the original sources [7]. Secondly, using existing federated engines on replicas may deteriorate their performance because the number of contacted endpoints increases. To illustrate, consider a DBpedia dataset d_1, and a federation that only accesses a public SPARQL endpoint of DBpedia. Using FedX [12], the execution of a three-triple pattern query with predicates from DBpedia is quite simple because the query can be exclusively executed in one endpoint. If the same query were executed in a federation with several endpoints, we would expect better performance. Surprisingly, this is not the case. Suppose d_2 is a replica of d_1, and the federation now includes the public DBpedia SPARQL endpoint and another local SPARQL endpoint that accesses d_2. Since each triple pattern is executed against both endpoints, the query performance deteriorates.

A recent index-based approach named DAW [11] has been proposed to detect data duplication and to reduce the number of selected sources. The index is built from data in the federation at a given time. In the previous example, FedX with DAW would contact only one DBpedia endpoint. However, DAW does not consider source dynamicity and consequently, the datasets will diverge whenever data change. In this paper, we propose FEDRA, an approach for querying federations of SPARQL endpoints that takes advantage of data replication. FEDRA relies on knowledge about replicas to reduce the number of selected sources even in presence of dynamic datasources and divergent replicas. Replicas are defined as SPARQL views[10] and the containment relationship between views. These descriptions are annotated with data provenance and timestamps that identify the state of the original datasource when the replica was created. Divergent replicas will have different timestamps. In the above example, d_1 is the original source, with timestamp t_0, while d_2 is a replica with the same timestamp, *i.e.*, d_1 and d_2 have the same RDF triples. Suppose that d_1 inserts or deletes some triples, then the timestamp of the new state of d_1 is t_1, where $t_1 > t_0$. Since $t_1 > t_0$, d_1 and d_2 diverge, *i.e.*, d_1 has a more recent state than d_2. The divergence can be computed as the number of operations that need to be integrated into the replica to synchronize it with the original dataset. Depending on the value of the divergence tolerated by a user, the query can be executed against d_2 even if d_1 is not available, at the risk of obtaining an inexact number of answers. Executing queries against replicas allows to overcome public endpoints limitations [2], but to avoid generating obsolete answers, divergence has to be controlled.

FEDRA solves the source selection problem in presence of divergence, and it is able to compute the divergence between replicas to decide which sources should be chosen to answer a query. FEDRA does not require access to dataset instances to check overlapping. Thus, it can be used in federations with frequently updatable datasets where different replicas coexists.

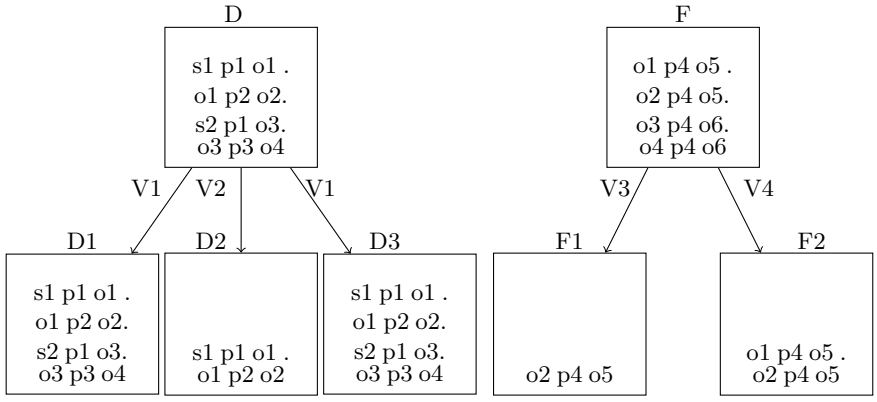
The main contributions of this paper are: *i*) Endpoint descriptions in terms of SPARQL views, containment relationships between views, data provenance, and timestamps; *ii*) FEDRA source selection algorithm that reduces the number of selected sources while divergence between replicas is controlled; and *iii*) an experimental study that reveals the benefits of both exploiting knowledge encoded in endpoint descriptions and controlling divergence to avoid obsolete results.

The paper is organized as follows: Section 2 presents FEDRA and the source selection algorithm. Section 3 reports our experimental study. Section 4 summarizes related work. Finally, conclusions and future work are outlined in Section 5.

2 Fedra Query Processing for SPARQL Federations with Replication

FEDRA enables federated engines to take advantage of data replication during query processing. A dataset is a set of RDF triples accessible through a public SPARQL endpoint. A replica is defined as a view expressed as a *CONSTRUCT* SPARQL query that retrieves a dataset fragment, *i.e.*, a subset of the dataset.

Replicas can be accessible through local SPARQL endpoints and can be part of a federation. Figure (1a) illustrates three fragments of a given dataset D: $D1$, $D2$, and $D3$. $D1$ and $D3$ are defined by view $V1$, they contain all the triples of D. $D2$ is defined by view $V2$ and corresponds to a subset of D. There is a containment relationship among these datasets, *i.e.*, $D \sqsubseteq_{V1} D1$, $D \sqsubseteq_{V2} D2$, and $D \sqsubseteq_{V1} D3$. The public SPARQL endpoint $E1$ provides access to D, while local SPARQL endpoints $E3$, $E4$, and $E5$ access $D1$, $D2$, and $D3$, respectively, as seen in Figure (1d). In addition, an endpoint can provide access to data from different datasets like $E5$ that combines $D3$ and $F1$, as seen in Figure (1d). Public and local SPARQL endpoints can contribute during federated query processing. Therefore, workload on public endpoints can be reduced, and local SPARQL endpoints can replace the public ones whenever they are not available.



(a) Fragment of Dataset D (b) Fragments of Dataset F

View	Definition
V1	CONSTRUCT { ?x p1 ?y . ?y ?p ?z } WHERE { ?x p1 ?y . ?y ?p ?z }
V2	CONSTRUCT { ?x p1 ?y . ?y p2 ?z } WHERE { ?x p1 ?y . ?y p2 ?z }
V3	CONSTRUCT { o2 p4 ?x } WHERE { o2 p4 ?x }
V4	CONSTRUCT { ?x p4 o5 } WHERE { ?x p4 o5 }

(c) Views definitions

Dataset	D	F	D1	D2	D3	F1	F2
Endpoint	E1	E2	E3	E4	**E5**	**E5**	E6

(d) Endpoints contents

Fig. 1: Example of Different Fragments and Containment Relationships

Listing 1: Query Q1

```
SELECT DISTINCT ?s ?o ?r
WHERE {
  ?s p1 ?o .
  ?o p4 ?r
}
```

Given a SPARQL query Q, we are interested in the following source selection problem: selecting the SPARQL endpoints that produce the most complete answer for Q while avoiding the selection of: *i*) public endpoints, and *ii*) multiple local endpoints that retrieve the same answer for a given sub-query of Q.

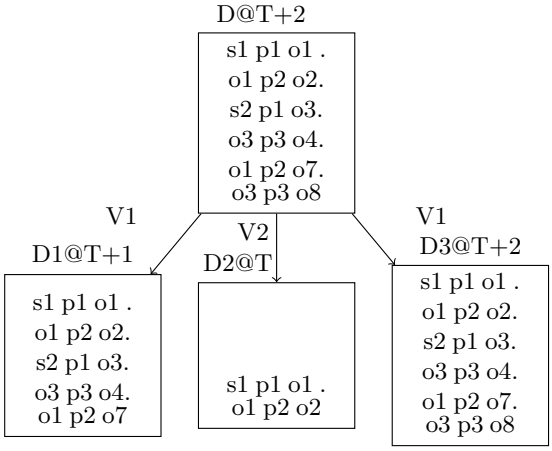
For the first triple pattern of query in Listing 1, FEDRA can select either endpoint $E3$ or endpoint $E5$ because they provide access to data defined by the same view. Endpoints $E5$ and $E6$ provide access to fragments $F1$ and $F2$ which are defined by different views, and they may have different instantiations for the second triple pattern; thus, both endpoints $E5$ and $E6$ need to be selected. To reduce the number of selected sources, $E5$ will be chosen instead of $E3$ because it will also be used for the second triple pattern. This choice gives the opportunity to the query engine of reducing the number of contacted endpoints while it maximizes the number of operations performed by an endpoint.

2.1 Describing Local SPARQL Endpoints

In a FEDRA federation, datasets accessible through local SPARQL endpoints are described explicitly in terms of views and implicitly, in terms of containment relationships between these views. In addition, they are annotated with data provenance and timestamps. A timestamp states the date when the fragment data is originally published in the public dataset.

Listing 2: Endpoint description

```
[] a sd:Service;
   sd:endpointUrl  <http://myExampleEndpoint/sparql>;
   sd:fragment  [
      sd:definition "CONSTRUCT { ?d bio2rdf : url ?u }
         WHERE { ?d bio2rdf : url ?u }";
      sd:origin <http://chebi.bio2rdf.org/sparql> ;
      sd:date "2000-01-12" ; ] ;
   sd:fragment  [
      sd:definition "CONSTRUCT { ?d bio2rdf : url ?u }
         WHERE { ?d bio2rdf : url ?u }";
      sd:origin <http://kegg.bio2rdf.org/sparql> ;
      sd:date "2000-05-13" ; ] ;
   ]
```



(a) Fragments of Dataset D

Date	Insertions
T	s1 p1 o1
	o1 p2 o2
	s2 p1 o3
	o3 p3 o4
T+1	o1 p2 o7
T+2	o3 p3 o8

(b) Inserts for D

Fig. 2: Coexisting versions of fragments of dataset D

2.2 Updating Public SPARQL Endpoints

Data accessible through public SPARQL endpoints can be updated, and divergence could exist with their replicas in the local endpoints. For example, DBpedia data frequently change, and existing replicas that do not integrate these last changes will be divergent with data in DBpedia. FEDRA can select local endpoints to obtain answers but with some divergence, or it can choose a public SPARQL endpoint, if this is available. Although public SPARQL endpoints offer access to the freshest data, they may time out executing complex or non-selective queries. Therefore, selecting possible divergent local endpoints could be a reliable choice whenever divergence is controlled.

Consider datasets from Figure (1a), where triples *k1=(o1 p2 o7)* and *k2=(o3 p3 o8)* are inserted in D after the fragments are created. Figure (2a) presents a possible scenario where dataset $D1$ has been created at timestamp *T+1*. $D1$ contains *k1* but not *k2*. Dataset $D2$ is created at the timestamp T, and it does not reflect the updates of D. Finally, dataset $D3$ is created at the timestamp *T+2*, therefore, it reflects the last state of D, *i.e.*, $D3$ has the freshest data of the public endpoint. As the triple *k2* is not available through endpoint $E3$, but it is available through endpoint $E5$, then there is a divergence between the dataset D and its fragments $D1$ and $D2$ at *T+2*.

Divergence is an editing distance between a public dataset and a replica. It can be defined as the number of the *insert* and *delete* operations that need to be applied to the replica in order to synchronize it with the original dataset. An operation in the public data introduces divergence with a replica, whenever it inserts or deletes a triple k that is relevant to a view V of the replica, *i.e.*, there is an instantiation of V that includes k. In the above example, triple *k1=(o1 p2*

o7) inserted into D at *T+1* is relevant to $V1$ and $V2$, while triple *k2=(o3 p3 o8)* inserted to D at *T+2* is only relevant to view $V1$. However, $D1$ and $D2$ do not include *k2* and *k1*, respectively. Thus, only one insertion needs to be performed on $D1$ and $D2$ to reach convergence, *i.e.*, their divergence is 1. Divergence of $D3$ is 0 because $D3$ is up to date with D. In general divergence of a replica $D1$ with respect to dataset D can be defined as in Equation 1, where k is a triple in D.

$$\begin{aligned} Divergence(D1, D) = \; & \Sigma_{k \in inserts(D) \wedge date(D1) < date(k) \leq date(D)} 1 \\ & + \Sigma_{k \in deletes(D) \wedge date(D1) < date(k) \leq date(D)} 1 \end{aligned} \tag{1}$$

Divergence defined in Equation 1 is for replicas of D that contain all its triples. If a replica only includes a subset of D, *i.e.*, it is defined by a view that does not retrieve all its triples, then divergence can be defined as in Equation 2. Predicate *matches(p, V)* is satisfied if triple k is relevant for view V.

$$\begin{aligned} Divergence(D1, D) = \; & \Sigma_{k \in inserts(D) \wedge date(D1) < date(k) \leq date(D) \wedge matches(k,V)} 1 \\ & + \Sigma_{k \in delete(D) \wedge date(D1) < date(k) \leq date(D) \wedge matches(k,V)} 1 \end{aligned} \tag{2}$$

Live updates of DBpedia[3] can be used to compute divergence between a replica of DBpedia and the public DBpedia relying on the timestamp of the last update performed at a given time. If we consider a version of DBpedia of May 13, 2013 and a replica defined by view V whose last update was done on May 1st, 2013, then divergence can be computed as the sum of the number of inserts and deletes performed during days 2-13 of May that are relevant to view V.

2.3 Source Selection Algorithm

Algorithm 1 selects the endpoints that can answer each triple pattern in a query, and the views that define the datasets accessible through these endpoints (e and v in line 7). The function *canAnswer* can be implemented using an `ASK` query for dynamic data, or relying on view definition for more stable data. *origin* function is used to determine if one of the already considered endpoints provides the same data that e for the fragment defined by v; in that case, e is included in $fragment$ as a valid alternative to obtain data provided by the other members of $fragment$ (lines 11-13). If the data is already provided by another endpoint f, then it is not necessary to consider e (lines 16-17, 20-21). If the data provided by a set of endpoints $fragment$ is already provided by e, then it is enough to query e and $fragment$ can be safely removed from the endpoints to query (lines 14-15, 18-19). When the user provides a threshold of divergence TD, then this can be included in line 7 as $divergence(e, v) \leq TD$. This algorithm produces a list whose elements are list of equivalent endpoints under a given view. This means that they offer the same fragment defined by the view, then during execution only one of them needs to be contacted. And different elements of this resulting list correspond to different fragments that should be considered in order to obtain

[3] `http://live.dbpedia.org/liveupdates/`

an answer as complete as possible, modulo the considered endpoints and the allowed divergence threshold.

After the source selection per triple pattern is done, a source selection for the whole query is performed, this is presented in Algorithm 2. We can perform the source selection using an heuristic for the finding the minimal set cover [6]. In the set cover problem, given a set of elements S and a collection of subsets of set C, the goal is to obtain a collection C', $C' \subseteq C$, such that all the elements in S belong to at least one element in C', and C' has the minimal size possible.

Algorithm 1 Triple Pattern Source Selection algorithm

Input: Q: SPARQL Query
Input: Es: set of Endpoints
Input: $views : Endpoint \rightarrow set\ of\ views$ ▷ views offered by each endpoint
Input: $origin : Endpoint \times view \rightarrow set\ of\ Endpoint$ ▷ endpoints from which the content of each view is taken
Input: $overloadedEndpoints : set\ of\ Endpoint$ ▷ endpoints that should not be selected
Input: $containedIn : view \rightarrow set\ of\ view$ ▷ containment relation among views
Output: D: Dictionary From Triple Pattern to List of List of Endpoints ▷ endpoints where each fragment of each triple pattern can be found

```
1: function tripleSourceSelection(Q,Es, views, origin, overloadedEndpoints, containedIn)
2:     for each k ∈ tp(Q) do                                              ▷ k is a triple pattern in Q
3:         endpoints ← ∅
4:         for each e ∈ Es do
5:             for each v ∈ views(e) do
6:                 if canAnswer(e, v, k) then
7:                     include ← true
8:                     for each fragment ∈ endpoints do
9:                         (f, w) ← getOne(fragment)
10:                        if origin(e, v) = origin(f, w) ∧ v ∈ containedIn(w) ∧ w ∈ containedIn(v)
    then
11:                            fragment.add((e, v))
12:                        include ← false
13:                    else if origin(e, v) = origin(f, w) ∧ v ∈ containedIn(w) then
14:                        endpoints.remove(fragment)
15:                    else if origin(e, v) = origin(f, w) ∧ w ∈ containedIn(v) then
16:                        include ← false
17:                    else if f ∈ origin(e, v) ∧ origin(e, v) ≠ origin(f, w) then
18:                        endpoints.remove(fragment)
19:                    else if e ∈ origin(f, w) ∧ origin(e, v) ≠ origin(f, w) then
20:                        include ← false
21:                    end if
22:                end for
23:                if include then
24:                    n ← {(e, v)}
25:                    endpoints.add(n)
26:                end if
27:            end if
28:        end for
29:    end for
30:    D(k) ← removeOverloadedIfPossible(endpoints, overloadedEndpoints)
31:    end for
32:    return D
33: end function
```

Consider a triple $k1$, such that $D(k1) = \{(v1, \{e1, e2\}), (v2, \{e3\}), (v3, \{e4, e5\})\}$, this means that to obtain all the possible bindings for $k1$, then it should be evaluated in at least one endpoint of each of the sets, for example, evaluating it in $e1$, $e3$ and $e5$ would be enough. We can model each endpoint by one subset of

the collection C, and the elements of each subset are the triples that can be evaluated by the corresponding endpoint. In order to respect the different fragments (three in the example), we include as many different elements in S as different fragments should be considered (three in the example). In the example: $k1_1 \in e1$, $k1_1 \in e2$, $k1_2 \in e3$, $k1_3 \in e4$ and $k1_3 \in e5$ (lines 2-13) The selected collection defines the endpoints where each triple pattern should be evaluated (lines 15-24). The *overloadedEndpoints* variable allows to restrict the set of endpoints where the triples are evaluated.

Algorithm 2 Source Selection algorithm

Input: Q: SPARQL Query; Es: set of Endpoints; $views : Endpoint \rightarrow set\ of\ views$; $origin : Endpoint \times view \rightarrow Endpoint$; $overloadedEndpoints : set\ of\ Endpoint$; $containedIn : view \rightarrow set\ of\ view$.
Output: E: Dictionary From Triple Pattern to List of Endpoints.

1: $usableEndpoints \leftarrow Es - overloadedEndpoints$
2: $D \leftarrow$ tripleSourceSelection(Q, usableEndpoints, views, origin, overloadedEndpoints, containedIn)
3: $S \leftarrow \emptyset$
4: $C \leftarrow \emptyset$ ▷ C(e) has as default value the empty set
5: **for** each $k \in tp(Q)$ **do** ▷ k is a triple pattern in Q
6: $i \leftarrow 0$
7: **for** each $es \in D(k)$ **do**
8: **for** each $e \in es$ **do**
9: $C(e) \leftarrow C(e) \bigcup \{k_i\}$
10: **end for**
11: $S \leftarrow S \bigcup \{k_i\}$
12: $i \leftarrow i + 1$
13: **end for**
14: **end for**
15: $C' \leftarrow$ MinimalSetCovering(S, C)
16: $endpoints \leftarrow$ getEndpoints(C')
17: $E \leftarrow \emptyset$
18: **for** each $k \in tp(Q)$ **do**
19: $e \leftarrow \emptyset$
20: **for** each $es \in D(k)$ **do**
21: $selected \leftarrow es \bigcap endpoints$
22: $e \leftarrow e \bigcup \{$takeOne$(selected)\}$
23: **end for**
24: $E(k) \leftarrow e$
25: **end for**

3 Experiments

We formulate the following research questions for a given SPARQL query: *RQ1*) can Fedra reduce the number of selected sources?, *RQ2*) can Fedra reduce the number of selected public endpoints?, *RQ3*) can Fedra select sources that produce more complete answers?, and *RQ4*) Fedra can select sources that satisfy a user threshold of divergence. How does increasing the threshold affect the completeness of the answer?.

Datasets and Query Benchmarks: we used the datasets Diseasome and Publication [4], and executed 89 queries where 59 queries are those executed in [11]

[4] `https://sites.google.com/site/dawfederation/data-sets`, January 2014.

and 30 conjunctive queries that are composed of a larger number of triple patterns. The queries are composed of single triple pattern queries (ST), star shaped queries of 2-6 triple patterns (2S-6S), and path queries of 2-4 triple patters (2P-4P). Contents of these datasets were characterized using conjunctive star-shaped CONSTRUCT queries such that the union of their evaluation provides the whole dataset. The views were randomly produced such that they have between two and ten triple patterns, and they contain as much overlap as possible, i.e., triples are considered for all the views they can match. For the Diseasome federation 58 views were generated, and 508 views for the Publication federation. In order to reproduce a realistic scenario, we consider that few views are highly used and many views are rarely used. Hence the views were sorted according to their popularity, i.e., for how many queries they are useful, and distributed using a power law distribution over a set of endpoints [5]. Samples of the power law distribution determines in how many endpoints each view is assigned, and a uniform distribution is used to assign them to endpoints. Parameters of the power law distribution are X_{min}=2 and α=2.5. The number of local endpoints for the Diseasome federation is 18, and for Publication is 345. Virtuoso[6] endpoints are used, and timeouts were set up to 1,800 secs. and 100,000 tuples. Table (1a) shows data distribution over ten of the endpoints of the federation for Diseasome, and Listings (3) and (4) present the views that define two fragments of Diseasome [7].

Divergence among replicas: We have created two divergent datasets of Diseasome: *Diseasome1* and *Diseasome2*. *Diseasome1* is composed of 90% of the Diseasome tuples, while *Diseasome2* has just 80% of these triples. The removed triples were selected following an uniform distribution over all the views. In the divergent setup the included data in each endpoint was uniformly taken from *Diseasome*, *Diseasome1*, and *Diseasome2*. The number of views that belong to each dataset per endpoint in this setup is shown in Table (1b).

Listing 3: view49

```
CONSTRUCT WHERE {
 ?x1 owl:sameAs ?x2.
 ?x1 diseasome:geneId ?x3.
 ?x1 rdfs:label ?x4.
 ?x1 diseasome:bio2rdfSymbol ?x5.
 ?x1 rdf:type ?x6.
 ?x1 diseasome:hgncId ?x7
}
```

Listing 4: view27

```
CONSTRUCT WHERE {
 ?x1 rdf:type ?x2 .
 ?x1 diseasome:omimPage ?x3
}
```

[5] Library `https://github.com/Data2Semantics/powerlaws` for power law distribution sampling.

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

[7] The data distribution of all the endpoints that conform the Federations and the views that describe all their replicated fragments are available at the project website: `https://sites.google.com/site/fedrasourceselection/`

Table 1: Data Distribution over ten endpoints, definitions of three fragments, and number of views that belong to each of the coexisting datasets in the divergent setup composed of Diseasome (D), *Diseasome1* (D1), and *Diseasome2* (D2).

(a) Data Distribution per Endpoint

Endpoint	Included Fragments
E1	view49,view11,view27,view22,view18,view13,view28,view33
E2	view16,view10,view45,view28,view26,view7,view11,view37,view33
E3	view12,view31,view5,view2,view49,view6,view22,view44,view58, view20,view43,view11,view18,view33,view26
E4	view57,view53,view24,view38,view40,view1,view33
E5	view51,view43,view9,view32,view11,view2,view34,view33
E6	view54,view34,view52,view42,view23,view6,view56,view41,view12, view11,view33,view21,view3
E7	view23,view10,view53,view36,view38,view11,view40,view58,view55, view33,view46,view21
E8	view25,view5,view1,view24,view35,view4,view31,view22,view33
E9	view23,view9,view28,view27,view32,view18,view42,view11,view6,view33
E10	view16,view10,view1,view21,view29,view19,view6,view33,view41

(b) Configuration of Divergent setup

Endpoint	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17	E18
D	4	3	6	1	3	5	4	2	1	3	4	6	2	3	3	3	7	6
D1	1	1	6	4	2	3	6	1	3	1	7	3	4	6	3	4	3	5
D2	3	5	3	2	3	5	2	6	6	5	7	4	3	2	4	3	3	5

Implementations: FEDRA is implemented in Java 1.7 and the Jena 2.11.0 library [8]. FEDRA produces SPARQL 1.1 queries where each triple pattern is annotated with a service clause that indicates where it will be executed. These queries are posed to FedX3.0[9], one state-of-the-art SPARQL 1.1 federated engine.

Evaluation Metrics: *i*) *Number of Triple-Wise Selected Public Endpoints (SPE):* corresponds to the sum of the number of times the public endpoint has been selected per triple pattern. *ii*) *Number of Triple-Wise Selected Sources (NTWSS):* corresponds to the sum of the number of different sources that has been selected per triple pattern. *iii*) *Execution Time (ET):* corresponds to 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 query execution by the underlying engine (ETUE). Time is expressed in seconds (secs.). A timeout of 300 secs. has been enforced for ETUE. SST was measured using System.currentTimeMillis() provided by Java and divided by 1000, and ETUE corresponds to the absolute wall-clock system time as reported by the Python `time.time()` function. *iv*) *Re-*

[8] `http://jena.apache.org/`, November 2013.
[9] `http://www.fluidops.com/fedx/`, November 2013.

call (R): corresponds to the size of the intersection between the obtained answers and the expected answers divided by the number of expected answers; where the expected answers are the ones obtained from an endpoint containing the whole dataset. *v*) *Duplicates (Du):* corresponds to the percentage of duplicates present in the answer. This is computed as (1 - *precision*)×100, where *precision* is defined as the size of the intersection between the obtained answers and the expected answers divided by the number of obtained answers; duplicates are considered. The range of Du is [0-100], where 0 means that the answer does not contain duplicates, while values close to 100 indicate that duplicate values outnumber the non-duplicate values. *vi*) *Divergence (Div):* corresponds to the total number of operations, insertions and deletions, that need to be performed in the relevant replicas to integrate the operations performed in the public dataset that have not been yet included in the relevant replicas. The divergence can be measured in a course-grained way considering the operations over the whole dataset, or fine-grained way considering the operations over a relevant fragment of the dataset. We have use a fine-grained measurement. For these metrics, we report the average value obtained per query type.

3.1 Reduction of the Number of Selected Sources

We ran FEDRA to compute the selected sources to answer our benchmark queries on the Diseasome and Publication federations, and thus, answer our research questions *RQ1* and *RQ2*. We used FedX to generate the execution plans of the queries, and from them compute the number of selected sources. We hypothesize that FEDRA is able to reduce the number of selected sources and the number of selected public endpoints, as far as the containment relationship holds among the replicas and the original dataset. Tables (2a) and (2b) show the NTWSS for FEDRA and FedX. These results suggest that our hypothesis are true, and FEDRA is able to reduce the number of selected sources up to two orders of magnitude, and the number of selected public endpoints is zero for all the queries but one. We notice that these savings are larger for federations with a large number of endpoints like Publication, and specially for queries with many triple patterns.

3.2 Increase of the Recall

We use FedX to run our benchmark queries on the Diseasome and Publication federations, and SPARQL 1.1 queries annotated with service clause according to sources selected by FEDRA. To answer our research question *RQ3*, we hypothesize that FEDRA is able to increase the recall (R) of the obtained answers, while keeping duplicates (Du) relatively low. Tables (2a) and (2b) show the R and Du for FEDRA and FedX. These results suggest that our hypothesis holds, and FEDRA is able to increase recall while producing relatively few duplicates. We notice that these savings are larger for federations with a large number of endpoints like Publication, and specially for queries with many triple patterns.

Table 2: Number of Triple-Wise Selected Public Endpoints (SPE), Triple-Wise Selected Sources (NTWSS), Execution Time (ET), Recall and Duplicates (Du) using FEDRA and FedX for Diseasome and Publication Federations. For FEDRA, ET is detailed in Source Selection Time (SST) and Query Execution Time by the Underlying Engine (ETUE). Relevant results are highlighted in **bold**.

(a) Diseasome

Query	FEDRA						FedX				
Type	SPE	NTWSS	SST	ETUE	Recall	Du	SPE	NTWSS	ET	Recall	Du
ST	0.00	1.60	1.60	1.28	1.00	21.80	1.00	17.80	1.94	0.88	83.60
2P	0.00	3.00	3.00	7.15	1.00	48.50	2.00	35.75	96.79	0.50	25.00
3P	0.00	3.80	3.80	10.38	0.80	59.60	3.00	53.60	38.50	0.21	39.00
4P	0.00	6.00	6.00	162.44	0.02	48.50	4.00	76.00	324.74	0.00	0.00
2S	0.00	4.00	4.00	11.64	1.00	48.20	2.00	37.00	169.47	0.40	18.00
3S	0.40	6.00	6.00	123.17	0.60	34.60	3.00	42.40	174.81	0.40	39.40
4S	0.00	7.00	7.00	138.97	0.60	0.00	4.00	75.60	240.97	0.00	0.00
5S	0.00	9.80	9.80	193.15	0.40	0.00	5.00	93.40	253.70	0.00	0.00
6S	0.00	11.40	11.40	284.58	0.20	0.00	6.00	112.40	257.44	0.00	0.00

(b) Publication

Query	FEDRA						FedX				
Type	SPE	NTWSS	SST	ETUE	Recall	Du	SPE	NTWSS	ET	Recall	Du
ST	0.00	2.20	26.43	3.26	1.00	14.80	1.00	126.60	7.99	0.92	85.40
2P	0.00	2.29	5.85	19.16	1.00	30.57	2.00	3.60	99.84	0.34	36.57
3P	0.00	7.86	3.81	51.78	0.86	22.00	3.00	321.57	261.55	0.14	14.29
4P	0.00	4.25	4.84	4.89	1.00	12.50	4.00	285.75	62.55	0.50	50.00
2S	0.00	3.20	33.44	26.65	0.80	36.00	2.00	470.80	129.66	0.00	0.00
3S	0.00	4.20	40.12	66.21	0.60	0.20	3.00	634.40	241.37	0.00	0.00
4S	0.00	**5.20**	108.48	190.20	0.40	0.00	4.00	**1352.60**	316.32	0.00	0.00
5S	0.00	**6.40**	134.86	137.06	0.60	0.00	5.00	**1528.00**	300.17	0.00	0.00
6S	0.00	**8.60**	138.42	140.79	0.60	0.00	6.00	**1811.60**	251.82	0.00	0.00

3.3 Answering Queries in Presence of Divergent Data

We use FedX to run our benchmark queries and SPARQL 1.1 queries annotated with service clause according to sources selected by FEDRA for the Diseasome federation with divergence and thresholds of 0% and 25%. To answer our research question *RQ4*, we hypothesize that as the threshold increases, the obtained recall may decrease. Table (3) reports NTWSS, ET, Recall and Duplicate values for FEDRA and FedX. These results suggest that our hypothesis is true in most of the cases but not in all. When threshold increases, the recall should decreases. However, when this does not hold, increasing the threshold allows to either: *i*) reduce the number of selected sources, or *ii*) select most suitable sources. This happens in query types 3P, 4P, and 6S where the number of queries that time out is reduced from 3, 2, and 5 to 1, 1, and 4, respectively.

Table 3: Number of Triple-Wise Selected Public Endpoints (SPE), Triple-Wise Selected Sources (NTWSS), Execution Time (ET), Recall (R) and Duplicates (Du) using FEDRA and FedX for Diseasome federation with divergence and thresholds of 0 and 25%. For FEDRA, ET is detailed in Source Selection Time (SST) and query Execution Time by the Underlying Engine (ETUE).

(a) Threshold = 0%

Query	FEDRA					
Type	SPE	NTWSS	SST	ETUE	R	Du
ST	0.80	1.40	1.33	1.29	1.00	19.00
2P	2.00	3.25	2.53	153.83	0.50	24.50
3P	2.80	4.60	5.93	193.42	0.30	30.00
4P	3.50	6.00	7.56	302.29	0.00	0.00
2S	1.60	4.00	2.33	73.13	0.79	29.60
3S	2.60	4.60	3.96	13.03	0.96	48.60
4S	3.00	6.20	7.86	95.71	0.58	0.00
5S	4.20	8.00	9.02	190.70	0.30	0.00
6S	5.20	10.00	10.31	300.13	0.00	0.00

(b) Threshold = 25%

FEDRA					
SPE	NTWSS	SST	ETUE	R	Du
0.20	1.40	1.42	1.17	0.88	20.00
0.25	3.00	2.48	154.57	0.49	10.75
0.40	3.80	5.84	64.29	0.59	41.20
0.00	6.00	7.20	155.40	0.02	48.50
0.00	4.80	2.67	12.63	0.72	49.20
1.00	5.80	3.90	182.01	0.20	0.00
0.00	7.20	7.81	85.96	0.32	0.00
0.00	10.00	9.14	191.16	0.20	0.00
0.00	11.80	10.79	275.54	0.15	0.00

(c) FedX

Query	FedX				
Type	SPE	NTWSS	ET	R	Du
ST	1.00	14.00	1.72	0.91	75.80
2P	2.00	32.75	79.23	0.50	24.75
3P	3.00	51.00	34.55	0.25	35.20
4P	4.00	76.00	304.61	0.00	0.00
2S	2.00	35.60	142.88	0.60	37.20
3S	3.00	34.20	152.60	0.60	59.00
4S	4.00	75.60	300.14	0.00	0.00
5S	5.00	93.40	300.62	0.00	0.00
6S	6.00	112.40	302.10	0.00	0.00

4 Related Work

The Semantic Web community has proposed different approaches to consume Linked Data from federations of endpoints [1, 4, 8, 12]. Although source selection and query processing techniques have successfully implemented, none of these approaches is able to exploit information about data replication to enhance performance and answer completeness. Recently Saleem et al. propose DAW [11], a source selection technique that relies on data summarization to describe RDF replicas and thus, reduces the number of selected endpoints. For each triple pattern in a SPARQL query, DAW exploits information encoded in source summaries to rank relevant sources in terms of how much they can contribute to the answer. Source summaries are expressed as min-wise independent permutation

vectors (MIPs) that index all the predicates in a source. Although properties of MIPs are exploited to efficiently estimate the overlap of two sources, since Linked Data can frequently change, DAW source summaries may need to be regularly recomputed to avoid obsolete answers. To overcome this limitation, FEDRA provides a more abstract description of the sources which is less sensible to data changes; data provenance and timestamps are stored to control divergence.

Divergence and data replication, in general, have been widely studied in distributed systems and databases. There is a fundamental trade-off between data consistency, availability and tolerance to failures. In one extreme, distributed systems implement the ACID transactional model and ensure strong consistency [5]. Although ACID transactional models have been extensively implemented, it has been shown that in large-scale systems where data is partitioned, ensuring ACID transactions and mutual consistency is not feasible [5]. Based on these results, FEDRA does not rely on strong consistency that would constrain Linked Data participants. Contrary, FEDRA exploits source descriptions to reduce the number of contacted endpoints while satisfies divergence thresholds. Thus, depending on the divergence tolerated by a user, a query can be performed against replicas if accessing the original source is not possible.

5 Conclusions and Future Work

We presented FEDRA a source selection approach that takes advantage of replicated data, and reduces the number of selected endpoints given a threshold of divergence. A lower value of divergence allows query engines to produce fresher answers but at the risk of failing to get answers. Contrary, a higher value risks engines to retrieve answers that are obsolete, while chances of producing answers increases as the space of possible selected sources is larger.

In the future, we plan to consider a finer-grained granularity of divergence. Instead of considering divergence of a replica, we will compute divergence for specific predicates, *e.g.*, divergence of predicates that represent links between datasets. In addition, we plan to integrate DAW with FEDRA to handle replicas with missing descriptions.

Acknowledgments. We thank Andreas Schwarte, who provided us with a version of FedX that prints the execution plans.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In Aroyo et al. [3], pages 18–34.
2. C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. Sparql web-querying infrastructure: Ready for action? In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *International Semantic Web Conference (2)*, volume 8219 of *Lecture Notes in Computer Science*, pages 277–293. Springer, 2013.

3. L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, editors. *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*. Springer, 2011.
4. 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.
5. E. A. Brewer. Pushing the cap: Strategies for consistency and availability. *IEEE Computer*, 45(2):23–29, 2012.
6. S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill, 2008.
7. P. Dourish. The parting of the ways: Divergence, data management and collaborative work. In *ECSCW*, pages 213–, 1995.
8. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In O. Hartig, A. Harth, and J. Sequeda, editors, *COLD*, volume 782 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
9. T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing linked data dynamics. In P. Cimiano, Ó. Corcho, V. Presutti, L. Hollink, and S. Rudolph, editors, *ESWC*, volume 7882 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2013.
10. W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on sparql views. In *WWW*, pages 655–664, 2011.
11. 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 H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 574–590. Springer, 2013.
12. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In Aroyo et al. [3], pages 601–616.