Decomposing federated queries in presence of replicated fragments

Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, Maria-Esther Vidal

Decomposing Federated Queries in presence of Replicated Fragments

Gabriela Montoya[a,*], Hala Skaf-Molli[a], Pascal Molli[a], Maria-Esther Vidal[b]

[a] *LINA – UFR de Sciences et Techniques, Nantes University*
2, rue de la Houssinière. 44322 NANTES CEDEX 3, France
[b] *Universidad Simón Bolívar*
Baruta, Edo. Miranda - Apartado 89000 Cable Unibolivar Caracas, Venezuela.

Abstract

Federated query engines allow for linked data consumption using SPARQL endpoints. Replicating data fragments from different sources enables data re-organization and provides the basis for more effective and efficient federated query processing. However, existing federated query engines are not designed to support replication. In this paper, we propose a replication-aware framework named LILAC, sparq**L** query decompos**I**tion against federations of rep**L**ic**A**ted data sour**C**es, that relies on replicated fragment descriptions to accurately identify sources that provide replicated data. We defined the query decomposition problem with fragment replication (QDP-FR). QDP-FR corresponds to the problem of finding the sub-queries to be sent to the endpoints that allows the federated query engine to compute the query answer, while the number of tuples to be transferred from endpoints to the federated query engine is minimized. An approximation of QDP-FR is implemented by the LILAC replication-aware query decomposition algorithm. Further, LILAC techniques have been included in the state-of-the-art federated query engines FedX and ANAPSID to evaluate the benefits of the proposed source selection and query decomposition techniques in different engines. Experimental results suggest that LILAC efficiently solves QDP-FR and is able to reduce the number of transferred tuples and the execution time of the studied engines.

Keywords: Linked Data, Federated Query Processing, Query Decomposition, Fragment Replication

1. Introduction

Billions of RDF triples have been made accessible through SPARQL endpoints by data providers. [1] Recent studies reveal unreliability and unavailability of existing public SPARQL endpoints [4]. According to the SPARQLES monitoring system [32] less than a third out of the 545 studied public endpoints exhibits an availability rate of 99-100% (values for November 2015).

Traditionally in distributed databases, fragmentation and replication techniques have been used to improve data availability [24]. Distributed database administrators are able to design the fragmentation and replication schema according to the applications and the expected workload. The Linking Open Data (LOD) cloud [7] datasets are published by autonomous data providers. Hence fragmentation and replication schema cannot be designed. Clearly, any data provider can partially or totally replicate datasets from other data providers. The LOD Cloud Cache SPARQL endpoint[2] is an example of an endpoint that provides access to total replicas of several datasets. DBpedia live[3] allows a third party to replicate DBpedia live changes in almost real-time. Data consumers may also replicate RDF datasets for efficient and reliable execution of their applications. However, given the size of the LOD cloud datasets, data consumers may just replicate subsets of RDF datasets or replicated fragments in a way that their applications can be efficiently executed. Partial replication allows for speeding up query execution time. Partial replication can be facilitated by data providers, e.g., DBpedia 2016-04[4] consists of over seventy dump files each of them providing different fragments of the same dataset, or can be facilitated by third party systems. Publish-Subscribe systems such as sparqlPuSH [25] or iRap RDF Update Propagation Framework [11] allow to partially replicate datasets. Additionally, data consumers are also autonomous and can declare federations composed of any set of SPARQL endpoints to execute their federated queries.

Consequently, partial or total replication can exist in federations of SPARQL endpoints; replicated data can be at different levels of consistency [17]; and a federated query engine has to be aware of the replication at runtime in order to efficiently produce correct answers. On one

*Corresponding author
Email addresses: gabriela.montoya@univ-nantes.fr (Gabriela Montoya), hala.skaf@univ-nantes.fr (Hala Skaf-Molli), pascal.molli@univ-nantes.fr (Pascal Molli), mvidal@ldc.usb.ve (Maria-Esther Vidal)

[1] http://stats.lod2.eu

[2] http://lod2.openlinksw.com/sparql

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

[4] http://downloads.dbpedia.org/2016-04/core-i18n/en/

hand, if a federated query engine is unaware of data replication, the engine performance may be negatively impacted whenever replicated data is collected from the available sources [22, 28]. On the other hand, if a federated query engine is aware of data replication, sources can be efficiently selected [16, 28] and data localities created by replication can be exploited [22] to significantly speed up federated query processing. These data localities, created by endpoints with replicated fragments from different datasets but relevant for federated queries, are not attainable by data providers.

Exploiting replicas can be beneficial. However, replicating data has the intrinsic problem of ensuring data consistency. Traditionally in distributed databases, replicas can be strongly consistent thanks to distributed transactions [24]. However, in the case of the Web, there is no mechanism to ensure that all the available data are strongly consistent. Regarding consistency of replicas, we have identified three main scenarios.

- First, if specific dataset versions are replicated, then the replicas are always perfectly synchronized, e.g., a replica of DBpedia 3.8 is always perfectly synchronized with DBpedia 3.8. This case is especially pertinent when a federated query is defined on a particular version of the available datasets in order to ensure reproducible results.

- Second, if the replicated data is locally modified, the local data is no longer a replica. For instance, if processes of data quality assessment are performed by a data consumer on a replica of DBpedia 3.8, changes to the replica need to be evaluated based on the trustiness of the data consumer. Clearly, data consumers have to change their federation according with what source they trust.

- Third, if the most up-to-date dataset versions are used, because strong consistency cannot be ensured in the context of LOD cloud datasets, replicas may be unsynchronized during query execution. Therefore, it is possible that some of the replicas used to evaluate the query have not integrated all the latest changes before queries are executed. This third scenario can be handled by measuring the replica divergence and the divergence incurred by the sources used to evaluate the query. Out of date replicas can be pruned during the source selection as proposed in [21].

In this paper, for the sake of simplicity, we work under the assumption that replicas are perfectly synchronized as in the first scenario and focus on query processing optimization under this assumption.

Query processing against sources with replicated data has been addressed in [22], while the related problem of query processing against sources with duplicated data has been tackled in [16, 28]. These three approaches prune redundant sources at source selection time. This selection may prevent the decomposer from assigning joins between a group of triple patterns to the same endpoint(s), even if this choice produces the most selective sub-query. To illustrate, consider a BGP with three triple patterns $tp1, tp2$, and $tp3$. Suppose a SPARQL endpoint $C1$ is relevant for $tp1$ and $tp3$, while $C2$ is relevant for $tp1$ and $tp2$. The source selection strategies implemented by these approaches, will prevent from assigning $tp1.tp3$ to $C1$ and $tp1.tp2$ to $C2$, even if these sub-queries generate less intermediate results. Consequently, as we show in Section 2, managing replication only at source selection time may impede a query decomposer to generate the most selective sub-queries.

In this paper, we exploit the replication knowledge during query decomposition, to generate query decompositions where the limitations of existing replication-aware source selection approaches are overcome. Furthermore, we formalize the query decomposition problem with fragment replication (QDP-FR). QDP-FR corresponds to the problem of finding the sub-queries to be sent to the endpoints that allow the federated query engine to compute the query answer, while the number of tuples to be transferred from the endpoints is minimized. We also propose an approximate solution to QDP-FR, called LILAC, sparq**L** query decompos**I**tion against federations of rep**L**ic**A**ted data sour**C**es, that decomposes SPARQL queries and ensures complete and sound query answers, while reducing the number of transferred tuples from the endpoints.

Specifically, the contributions of this paper are:

- We outline the limitations of solving the source selection problem independently of the query decomposition problem in the context of replicated and fragmented data. We propose an approach where these two federated query processing tasks should be interleaved to support engines in finding better execution plans.

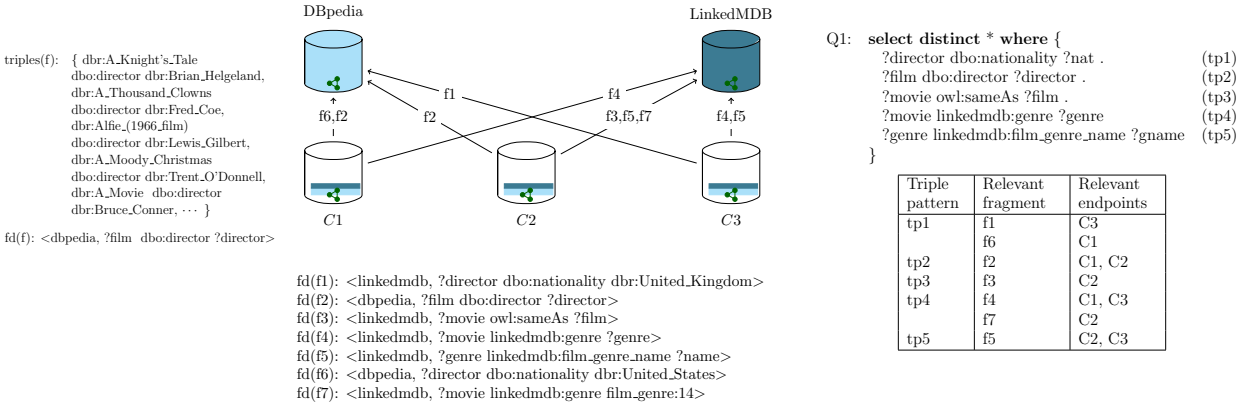
- Based on the replication-aware framework introduced in [22], we propose a query decomposition strategy that relies on this framework to exploit fragments replicated by various endpoints.

- We formalize the query decomposition problem with fragment replication (QDP-FR).

- We propose a sound and complete algorithm to solve the QDP-FR problem.

- We reduce the QDP-FR problem to the set covering problem and use existing set covering heuristics to produce *good* solutions to the QDP-FR problem.

- We extend federated query engines FedX and ANAPSID to perform LILAC query decomposition, i.e., we extend the engines and create the new engines LILAC+FedX and LILAC+ANAPSID. We



(a) Fragment description (b) Replicated Fragments (c) SPARQL query *Q1* and its triple pattern relevant fragments and endpoints

Figure 1: **Querying Federations with Replicated Fragments**; (a) A fragment f is associated with a fragment description $fd(f)=<u, tp>$, i.e., an authoritative endpoint u, and a triple pattern tp; *triples(f)* corresponds to the RDF triples of f that satisfy $fd(f).tp$ in the dataset accessible through $fd(f).u$. (b) Three endpoints have replicated seven fragments from DBpedia and LinkedMDB. (c) A SPARQL query, *Q1*, with five triple patterns; relevant endpoints of this query access RDF triples that match these five triple patterns

study the performance of these engines and compare them with existing engines FedX, DAW+FedX, Fedra+FedX, ANAPSID, DAW+ANAPSID, and Fedra+ANAPSID. Results suggest that query decompositions produced by LILAC contribute to reduce the number of transferred tuples and the query execution time.

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

2. Background and Motivation

In this section, we illustrate the impact that exploiting metadata about replicated fragments has on federated query processing. First, we assume that data consumers replicate fragments composed of RDF triples that satisfy a given triple pattern; URIs in the original RDF dataset are kept for all the replicated resources. In Figure 1a, a fragment of DBpedia is illustrated. This fragment comprises RDF triples that satisfy the triple pattern `?film dbo:director ?director`; triples included in Figure 1a correspond to a copy of the DBpedia RDF triples where URIs are preserved. Fragments are described using a 2-tuple *fd* that indicates the authoritative source of the fragment, e.g., DBpedia; the triple pattern that is met by the triples in the fragment is also included in *fd*, e.g., `?film dbo:director ?director`.

Figure 1b depicts a federation of three SPARQL endpoints: $C1$, $C2$, and $C3$; these endpoints expose seven replicated fragments from DBpedia and LinkedMDB. Replicated fragments correspond to copies of the RDF triples in DBpedia and LinkedMDB. Query processing against $C1$, $C2$, and $C3$ can be more efficient in terms of execution time while query completeness can be ensured under certain conditions. To explain, consider the SPARQL query $Q1$ presented in Figure 1c[5], this query comprises five triple patterns: $tp1$, $tp2$, $tp3$, $tp4$, and $tp5$. Based on the distribution and replication of the seven fragments in Figure 1b, relevant SPARQL endpoints for these triple patterns are selected, i.e., $C1$ is relevant for $tp1$, $tp2$, $tp4$; $C2$ is relevant for $tp2$, $tp3$, $tp4$, $tp5$; and $C3$ is relevant for $tp1$, $tp4$, $tp5$. However, because fragments are replicated, only one endpoint could be selected to execute triple patterns $tp2$-$tp5$, e.g., $C2$ for $tp2$, $tp3$, and $C3$ for $tp4$, $tp5$.

Existing federated SPARQL engines [1, 6, 12, 26, 30] are not tailored for Linked Data replication. In consequence, none of these engines prevents redundant data from being retrieved or merged even in the presence of federations as the one presented in Figure 1b. Relevant endpoints for each query triple pattern of the federation in Figure 1b are given in Figure 2a. All of these relevant endpoints are included in logical plan produced by FedX [30] and given in Figure 2c. While some of these endpoints are pruned by ANAPSID heuristics [23] and are not included in logical plan given in Figure 2e. Because of the lack of description about the replicated data accessible through the endpoints, FedX is not able to detect that

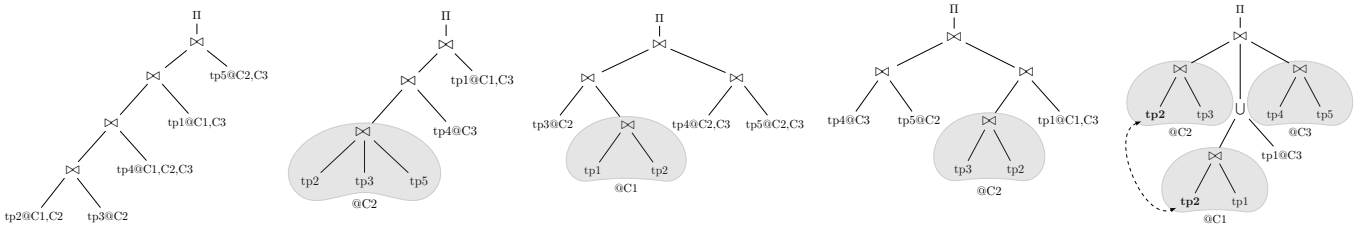
[5]The federation of SPARQL endpoints and query depicted in Figures 1b and 1c extend the example given in [22].

Triple pattern	Relevant fragment	Relevant endpoints
tp1	f1	C3
	f6	C1
tp2	f2	C1, C2
tp3	f3	C2
tp4	f4	C1, C3
	f7	C2
tp5	f5	C2, C3
Known containments		
triples(f7) ⊆ triples(f4)		

(a) Relevant endpoints and known containments

Transfered tuples	Q1		Q1D	Q1		Q1D
	FedX	FEDRA+FedX	**FedX**	ANAPSID	FEDRA+ANAPSID	**ANAPSID**
from C1	113,702	1	162	1,036	20,998	1,036
from C2	108,739	1,000,001	9,890	64,066	10,005	9,889
from C3	26,364	158	225	41,176	45,824	45,824
Total	248,805	1,000,160	**10,277**	106,278	76,827	**56,749**
Exec Time secs	549.90	⩾1,800	**233.57**	30.25	20.67	**16.90**

(b) Number of transferred tuples and execution time



(c) FedX $Q1$ (d) FEDRA + FedX $Q1$ (e) ANAPSID $Q1$ (f) FEDRA + ANAPSID $Q1$ (g) $Q1D$ decomposition for $Q1$

Figure 2: **Execution of $Q1$ (Figure 1c) on federation defined in Figure 1b.** (a) Endpoints in federation of Figure 1b that are relevant for $Q1$. (b) Number of Transferred Tuples and Query Execution Time (secs) for $Q1$. ; **Bold** values correspond to the lower number of transferred tuples and execution time for each engine; (c) FedX Left-linear plan; (d) the replication-aware FEDRA + FedX plan; (e) ANAPSID Bushy tree plan; (f) the replication-aware FEDRA + ANAPSID plan. Both FEDRA + FedX and FEDRA + ANAPSID plans delegate join execution to the endpoints and assign as many triple patterns as possible to the same endpoint. (g) $Q1D$ is a possible decomposition of $Q1$.

more than one triple pattern can be exclusively evaluated by one single endpoint, while ANAPSID heuristics allows for delegating the evaluation of just one join to the endpoints. In consequence, the query decompositions include many sub-queries with only one triple pattern. The resulting plans lead to retrieve redundant data from endpoints. Figure 2b reports on the results of execution of FedX and ANAPSID logical plans; as expected, they generate many intermediate results that impact negatively their performance.

Figures 2d and 2f show the logical plans that FedX and ANAPSID produce when the replication-aware source selection FEDRA [22] is used. The replication-aware source selection approaches aim to prune redundant sources for triple patterns. In this example, $tp2$ can be executed on $C1$ and $C2$ and FEDRA assigns $tp2$ to $C2$, forcing the query decomposer to evaluate $tp2$ on $C2$ only. Results reported in Figure 2b, demonstrate that FEDRA reduces the number of transferred tuples and improves the execution time compared to ANAPSID. Unfortunately, this is not the case for FedX, FEDRA source selection leads FedX to include sub-queries with Cartesian products that negatively impact on the number of transferred tuples and execution time.

However, for this example, there exists another decomposition, $Q1D$, for the query $Q1$ (Figure 1c) presented in Figure 2g. FedX and ANAPSID use different heuristics to optimize and generate physical plans for this decomposition. For decomposition $Q1D$, FedX transfers less tuples and exhibits a lower execution time than FedX or FEDRA+FedX for $Q1$ (Figure 2b). We observe the same behavior with ANAPSID.

$Q1D$ is a *better* decomposition because endpoints receive more selective sub-queries. Unlike all others decompositions, $tp2$ is assigned to two endpoints: to $C1$ in conjunction with $tp1$ and to $C2$ in conjunction with $tp3$ (dashed line in Figure 2g). We notice clearly the effect of sending more selective sub-queries on intermediate results and execution times.

Unfortunately, this decomposition cannot be computed by existing decomposers after solving the source selection problem with replicated fragments. Replication aware source selection strategies prune redundant sources for a triple pattern, i.e., a triple pattern assigned to one endpoint cannot be reassigned to another. For example, in Figure 2d, FEDRA assigned $tp2$ to $C2$, so it cannot assign $tp2$ to $C1$. In order to find decompositions as $Q1D$, source selection and query decomposition have to be interleaved.

In this paper, we propose a source selection and a query decomposition strategy called LILAC able to produce query decompositions as $Q1D$ (Figure 2g).

3. Definitions and Problem Description

This section introduces definitions and the query decomposition problem with fragment replication (QDP-FR).

3.1. Definitions

Fragments of datasets are replicated preserving the URIs of the resources. A fragment is composed of a set of RDF triples, these triples may be described using the

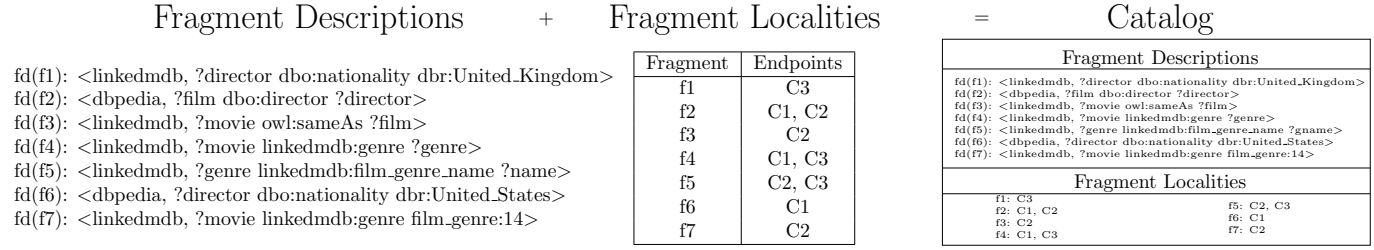


Figure 3: Data consumers expose the fragment description of their replicated fragments, LILAC computes the fragments localities using the descriptions provided by several endpoints and stores this information in its catalog

authoritative endpoint from where the triples have been replicated and the triple pattern that they match.

Without loss of generality, fragments are limited to fragments that can be described using one triple pattern as in [17, 33].

Definition 1 (Fragment Description) *Given a fragment identifier f and a set of RDF triples triples(f), the description of f is a tuple $fd(f) = \langle u, tp \rangle$*

- *u is the non-null URI of the authoritative endpoint where the triples triples(f) are accessible;*
- *tp is a triple pattern matched by all the RDF triples in triples(f) and no other triple accessible through u matches it.*

We assume that there is just one level of replication, i.e., one endpoint can provide access to fragments replicated from one or several authoritative endpoints, but these endpoints cannot be used as authoritative endpoints of any fragment. Additionally, our approach relies on the further assumptions: (i) Fragments are read-only and perfectly synchronized; [6] (ii) SPARQL endpoints are described in terms of the fragments whose RDF triples can be accessible through the endpoint.

Assumption (i) allows us to focus in the first scenario of replicating a precise dataset version (scenario discussed in Section 1), and whose solutions can be combined with divergence metrics for scenarios where latest dataset version is required. An initial approach to integrate divergence metrics during source selection has been proposed in [21]. This approach performs the source selection using the sources under a given threshold of allowed divergence.

Albeit restrictive, this replication strategy allows for accessing RDF data replicated from different authoritative endpoints using one single endpoint. Re-locating data in this way opens different data management opportunities, and provides the basis to query optimization techniques that can considerably reduce data transfer delays and query execution time, as will be observed in the experimental study results.

To illustrate the proposed replication strategy, consider the federation given in Figure 1b. Consider the two triple patterns that share one variable and have the predicates `dbo:director` and `owl:same` in query *Q1* (Figure 1c). Further, assume that the endpoint $C2$ is selected to process this query instead of the authoritative endpoints, DBpedia and LinkedMDB. Because the execution of the join can be delegated to $C2$, the number of transferred tuples can be reduced whenever not all the triples with predicate `dbo:director` in DBpedia have a joinable triple with predicate `owl:sameAs` in LinkedMDB.

The federated query engine catalog includes the descriptions of replicated fragments and the endpoints that provide access to them, i.e., the fragment localities. The catalog for the federation in Figure 1b is shown in Figure 3.

The catalog is computed during the configuration/definition of the federation, and this should be done before the execution of any query. Each endpoint exposes the description of the fragments it has replicated; The catalog is locally created by contacting the endpoints in the federation to retrieve the descriptions of the replicated fragments. These descriptions and the fragment containment is used to identify fragments that have the same data and obtain the fragment localities. For example, if the descriptions of fragments fa and fb are obtained from endpoints $C1$ and $C2$ and fa, fb differ only in variable names, they are identified as the same fragment. Therefore, only one fragment and two endpoints are included in the catalog.

Several fragments present in the catalog may be relevant for a given triple pattern. However, not all of them need to be accessed to evaluate the triple pattern. Among the relevant fragments, only the fragments that are not *contained* in another relevant fragment need to be accessed. For example, consider query *Q1* (Figure 1c), the federation given in Figure 1b, and its catalog given in Figure 3. There are two fragments that have triples that match the triple pattern `?movie linkedmdb:genre ?genre`, $f4$ and $f7$. But as all the triples in $f7$ are also in $f4$, then $f7 \sqsubseteq f4$ and consequently only $f4$ needs to be accessed.

We adapt the query containment and equivalence definition given in [14] for the case of a triple pattern query.

Definition 2 (Triple Pattern Query Containment) *Let $TP(D)$ denote the result of execution of query TP over an RDF dataset D. Let TP_1 and TP_2 be two triple pattern queries. We say that TP_1 is contained in*

[6] The fragment synchronization problem has been studied in [11, 17].

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* $TP_1 \equiv TP_2$ *iff* $TP_1 \sqsubseteq TP_2$ *and* $TP_2 \sqsubseteq TP_1$.

Testing the containment of two triple pattern queries[7] amounts to finding a substitution of the variables in the triple patterns[8]. $TP_1 \sqsubseteq TP_2$, iff there is a substitution θ such that applying θ to TP_2 returns the triple pattern query TP_1. Solving the decision problem of triple pattern query containment between TP_1 and TP_2, $TP_1 \sqsubseteq TP_2$, requires to check if TP_1 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. Therefore, testing triple pattern containment has a complexity of $O(1)$. Moreover, we overload operator $\sqsubseteq$ and use it to relate triple patterns if and only if the triple pattern queries composed by such triple patterns are related.

The decision problem of *fragment containment* of two fragments f_i and f_j, is solved based on the satisfaction of the triple patterns that described these fragments, i.e., $fd(f_i).tp \sqsubseteq fd(f_j).tp$. Restricting the description of fragments to a single triple pattern allows for the reduction of the complexity of this problem. For the federation in Figure 1b, $f7 \sqsubseteq f4$ because $f4$ and $f7$ share the same authoritative endpoint and there is a substitution θ defined as { *(?genre, film_genre:14), (?movie, ?movie)* } and applying θ to $f4$ returns $f7$.

We can rely on both fragment descriptions and on the containment relation to identify a query relevant fragments. A relevant fragment contains RDF triples that match at least one triple pattern of the query.

Definition 3 (Fragment relevance) *Let f be a fragment identifier and TP be a triple pattern of a query Q. f is relevant to Q if $TP \sqsubseteq fd(f).tp$ or $fd(f).tp \sqsubseteq TP$.*

Table 1 shows the relevant fragments to the triple patterns in query *Q1* (Figure 1c) and the federations in Figure 1b. For example, the triple pattern $tp1$ has two relevant fragments, $f1$ and $f6$; the triple pattern $tp2$ has one relevant fragment, $f2$; the triple pattern $tp3$ has one relevant fragment, $f3$; the triple pattern $tp4$ has two relevant fragments, $f4$ and $f7$; and triple pattern $tp5$ has one relevant fragment, $f5$. Because $f7 \sqsubseteq f4$, a complete answer for $tp4$, can be produced from RDF triples *triples(f4)*, i.e., by accessing only a single endpoint the complete set $[\![tp4]\!]_{\text{LinkedMDB}}$ is collected. In contrast, both fragments $f1$ and $f6$ are required to answer $tp1$ and two endpoints need to be contacted to collect the complete answer for $tp1$. This example illustrates the impact of using triple pattern containment during source selection to reduce the number of selected endpoints.

[7]Containment testing is adapted from [13].

[8]The substitution operator preserves URIs and literals, i.e., only variables are substituted.

Table 1: Relevant Fragments for federation in Figure 1b and query *Q1* (Figure 1c)

Q triple pattern	Relevant Fragment
tp1	f1 f6
tp2	f2
tp3	f3
tp4	f4 f7
tp5	f5

Definition 4 (Federation with Replicated Fragments) *A SPARQL federation with replicated fragments, Fed, is defined as a 6-tuple, <E, F, endpoints(), relevantFragments(), DS, ⊑>. E is a set of endpoints that compose the federation Fed. F is a set of identifiers of the fragments accessible through endpoints in E. DS is a set of triples accessible through the federation Fed. Function endpoints(f) maps a fragment identifier f to the set of endpoints in E that access the fragment. Function relevantFragments(tp) maps a triple pattern tp to a subset F′ of F that corresponds to the relevant fragments of tp (Definition 3). Operator ⊑, relates triple pattern queries according to Definition 2.*

3.2. Query Decomposition Problem with Fragment Replication (QDP-FR)

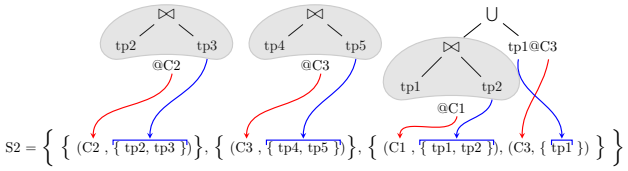
Given a SPARQL query Q, and a federation *Fed=<* *E*, *F*, *endpoints()*, *relevantFragments()*, ⊑, *DS>*, the Query Decomposition Problem with Fragment Replication (QDP-FR) decomposes each basic graph pattern (BGP) of Q (*bgp*) into sub-queries, i.e., sets of triple patterns that can be posed to the same endpoint. These sub-queries can be combined using joins and unions to obtain an execution plan for *bgp* that provides the same answers than the evaluation of *bgp* against *Fed*, while the number of tuples transferred during the execution of these sub-queries is minimal.

Once the query has been parsed, for each BGP sources are selected and the BGP is decomposed following LILAC strategies. For the sake of clarity, we formalize the properties of a query decomposition and give the algorithms for the decomposition at BGP level. Naturally, any SPARQL query that includes UNIONs, OPTIONALs, blocks, or FILTERs, can be decomposed starting at the BGP level and combining the BGP decompositions according to the query structure.

A query decomposition *Q'* is represented as a set S composed of sets of sets of pairs (endpoint, set of triple patterns), where endpoint represents a single endpoint from the federation and set of triple patterns is a subset of the BGP in *Q'*. S is built from *Q'*, where each element jo of S corresponds to a join operand in the corresponding query plan, i.e., elements jo in S correspond to sub-queries connected by joins in the final plan. Further, each pair uo=(e, ts) in a join operand jo indicates an endpoint e where the set of triple patterns ts will be posed; elements uo in a join operand jo are connected by unions in the final

Triple pattern	Relevant fragment	Relevant endpoints
tp1	f1	**C3**
	f6	**C1**
tp2	f2	**C1**, **C2**
tp3	f3	**C2**
tp4	f4	C1, **C3**
	f7	C2
tp5	f5	C2, **C3**

(a) Selected endpoints



(b) Decomposition and model

Figure 4: Selected endpoints are included in the LILAC query decomposition of *Q1* (Figure 1c) for federation in Figure 1b, the decomposition is modeled with a set of sets of pairs (endpoint, set of triple patterns)

plan. Selected endpoints (Figure 4a) are used to produce the LILAC query decomposition presented in Figure 4b for the SPARQL query *Q1* in Figure 1c and federation in Figure 1b. The set based representation S of this LILAC decomposition is also presented.

Given a query decomposition Q' that corresponds to a non-empty solution of QDP-FR, the set based representation S of Q' meets the following properties:

1. **Answer soundness:** evaluating Q' produces only answers that would be produced by the evaluation of Q over the set of all triples available in the federation. Formally this property can be written as:
   $\text{eval}(Q') \subseteq Q(DS)$

2. **Answer completeness:** evaluating Q' produces all the answers that would be produced by the evaluation of Q over the set of all triples available in the federation.

   Formally this property can be written as:
   $\text{eval}(Q') \supseteq Q(DS)$

3. **Data transfer minimization:** executing Q' minimizes the number of tuples transferred from the selected endpoints, i.e., each sub-query is as large as possible without Cartesian products and there are no redundant sub-queries. Formally, this property can be written as:
   No Cartesian products:
   $(\forall \text{ sq, e, jo} : \text{jo} \in S \wedge (\text{e, sq}) \in \text{jo} : (\forall \text{ sq'} : |\text{sq'}| \geqslant 1 \wedge \text{sq'} \subset \text{sq} : (\exists \text{ t, t'} : \text{t} \in \text{sq'} \wedge \text{t'} \in (\text{sq - sq'}) : \text{Vars(t)} \bigcap \text{Vars(t')} \neq \varnothing)))$
   Sub-queries are as large as possible:
   $(\forall \text{ sq, e, jo} : \text{jo} \in S \wedge (\text{e, sq}) \in \text{jo} : \neg (\exists \text{ t} : \text{t} \in \text{bgp-sq} : (\exists \text{ f} : \text{f} \in \text{relevantFragments(t)} \wedge \text{e} \in \text{endpoints(f)}) \wedge (\exists \text{ t'} : \text{t'} \in \text{sq} : \text{Vars(t)} \bigcap \text{Vars(t')} \neq \varnothing)))$
   No redundant sub-queries:
   $(\forall \text{ jo} : \text{jo} \in S : (\exists \text{ t} : \text{t} \in \text{bgp} \wedge (\forall \text{ e, sq} : (\text{e, sq}) \in \text{jo} : \text{t} \in \text{sq}) : \neg(\exists \text{ jo'} : \text{jo'} \in S \wedge \text{jo'} \neq \text{jo} : (\forall \text{ e', sq'} : (\text{e', sq'}) \in \text{jo'} : \text{t} \in \text{sq'})))$

We illustrate QDP-FR on running query *Q1* of Figure 1c and federation in Figure 1b. Figure 4a presents the relevant fragments and relevant endpoints for each triple pattern. Decompositions in generated by FedX, ANAPSID, FEDRA+FedX, FEDRA+ANAPSID and LILAC (Figures 2c, 2e, 2d, 2f, and 2g) retrieve all the relevant data in the federation and only relevant data, i.e., they ensure properties 1-2. Even if decomposition generated by FEDRA+ANAPSID (Figure 2f) significantly reduces the number of transferred tuples with respect to the decomposition generated by ANAPSID (Figure 2e), only the decomposition generated by LILAC (Figure 2g) minimizes the transferred data. LILAC includes more selective sub-queries and these sub-queries contribute to reduce the number of transferred tuples and the query execution time, e.g., instead of a sub-query with just *tp1* as FEDRA+ANAPSID and ANAPSID include in their decompositions, LILAC includes the more selective sub-query composed of *tp1* and *tp2*.

A source selection approach in combination with existing query decomposition strategies may produce decompositions as the ones in Figures 2d and 2f. Furthermore, the number of transferred tuples may be significantly reduced whenever source selection is interleaved with query decomposition and information about the replicated fragments are available.

4. LILAC: An Algorithm for QDP-FR

The goal of LILAC is to reduce data transfer by taking advantage of the replication of relevant fragments for several triple patterns on the same endpoint. Function `Decompose` solves a two-fold problem; first relevant fragments are selected and then sub-queries and relevant endpoints are simultaneously chosen, e.g., two triple patterns are assigned to the same sub-query if they share a variable and can be evaluated at the same endpoint.

Algorithm 1 proceeds in four steps:

I. Selection of the non redundant fragments for each triple pattern (line 2).

II. Identification of the candidate endpoints reducing the unions if possible (line 3).

III. Generation of the largest Cartesian product free sub-queries to evaluate the BGP triple patterns with just one relevant fragment (line 4).

IV. Reduction of the non selective sub-queries by merging *joinable* sub-queries that can be executed against the same endpoint, i.e., sub-queries that share at least one variable and are assigned to the same endpoint (line 5).

Notice that the first two steps are also performed by the FEDRA source selection strategy. Even if both, FEDRA and LILAC, use a reduction to the set covering problem to refine the set of selected sources for the triple patterns in a BGP in the third step, the reduction used by LILAC meets the requirement of delegating join execution to the endpoints, and ensures that the largest sub-queries obtained from this third step are effectively Cartesian product free sub-queries. Finally, LILAC implements the fourth step, being able to place the same triple pattern in more than one sub-query. This is a *unique* characteristic of the decompositions generated by LILAC.

Algorithm 1 Decompose Algorithm

Require: A Basic Graph Pattern *bgp*
A Federation *Fed=<E,F,endpoints(),relevantFragments(),* $\sqsubseteq$*,DS>*
Ensure: A decomposition representation *decomposition*
1: **function** `Decompose`(bgp,Fed)
2: fragments ← `SelectNonRedundantFragments`(bgp, Fed)
3: endpoints ← `ReduceUnions`(fragments)
4: (subqueries, ljtps) ← `ReduceBGPs`(endpoints)
5: decomposition ← `IncreaseSelectivity`(endpoints,subqueries, ljtps)
6: **return** decomposition
7: **end function**

Next, we illustrate how Algorithm 1 works on our running example of query $Q1$ (Figure 1c) and federation in Figure 1b. [9]

First, for each triple pattern, LILAC computes the non redundant set of relevant fragments; two or more fragments are grouped together, if they *share relevant data* for the triple pattern. Formally, the output of the function `SelectNonRedundantFragments` satisfies the following properties: *fragments(tp)* is a set of pairs (*tp*,*rfs*) where *tp* is a triple pattern in BGP and *rfs* is a set of sets of relevant fragments for *tp*. The set *rfs* is composed of sets $fs_1, ..., fs_m$ of fragments such that each fs_i comprises fragments that *share revelant data* for tp. A singleton fs_i={ f_i } includes a fragment such that $fd(f_i).tp$ is contained by the triple pattern *tp*, i.e., $fd(f_i).tp \sqsubseteq tp$; additionally, there is no other fragment f_k such that $fd(f_k).tp \sqsubseteq tp$ and $fd(f_k).tp$ contains $fd(f_i).tp$. A non singleton fs_i includes all the fragments f_i such that $fd(f_i).tp$ subsumes the triple pattern *tp*, i.e., $tp \sqsubseteq fd(f_i).tp$.

For the federation in our running example, relevant fragments (Figure 5b) are used to produce the *fragments* (Figure 5c) by function `SelectNonRedundantFragments`. The pair (*tp*4, {{$f4$}}) is part of the set *fragments*; note that even if $f5$ is a relevant fragment for *tp*4, i.e., $f5$ ∈ *relevantFragments(tp4)*, because *f4* contains *f5*, i.e., $f5 \sqsubseteq f4$, $f5$ is not included in *fragments* for *tp4*.

Second, the function `ReduceUnions` (Algorithm 2) localizes fragments on endpoints, i.e., performs an initial endpoint selection based on the endpoints that provide access to the fragments in *selectedFragments*; the function

[9]Notice that only triples in fragments $f1$ and $f6$ are accessible in this federation to retrieve data for *tp*1, therefore it will not produce all the answers that would be produced using DBpedia.

Algorithm 2 ReduceUnions Algorithm

Require: A set *selectedFragments*, composed of pairs (tp, fss), with tp a triple pattern and fss a set of sets of fragments
Ensure: A set *candidateEndpoints*, composed of pairs (tp, ess), with tp a triple pattern and ess a set of sets of endpoints
8: **function** `ReduceUnions`(selectedFragments)
9: candidateEndpoints ← ∅
10: **for** (tp, fs) ∈ selectedFragments **do**
11: **if** cardinality(fs) > 1 **then**
12: ce ← (⋂ endpoints(f) | f ∈ fs)
13: **end if**
14: **if** ce = ∅ ∨ card(fs) = 1 **then**
15: ce ← { endpoints(f) : f ∈ fs }
16: **end if**
17: candidateEndpoints ← candidateEndpoints ⋃ { (tp, ce) }
18: **end for**
19: **return**
20: **end function**

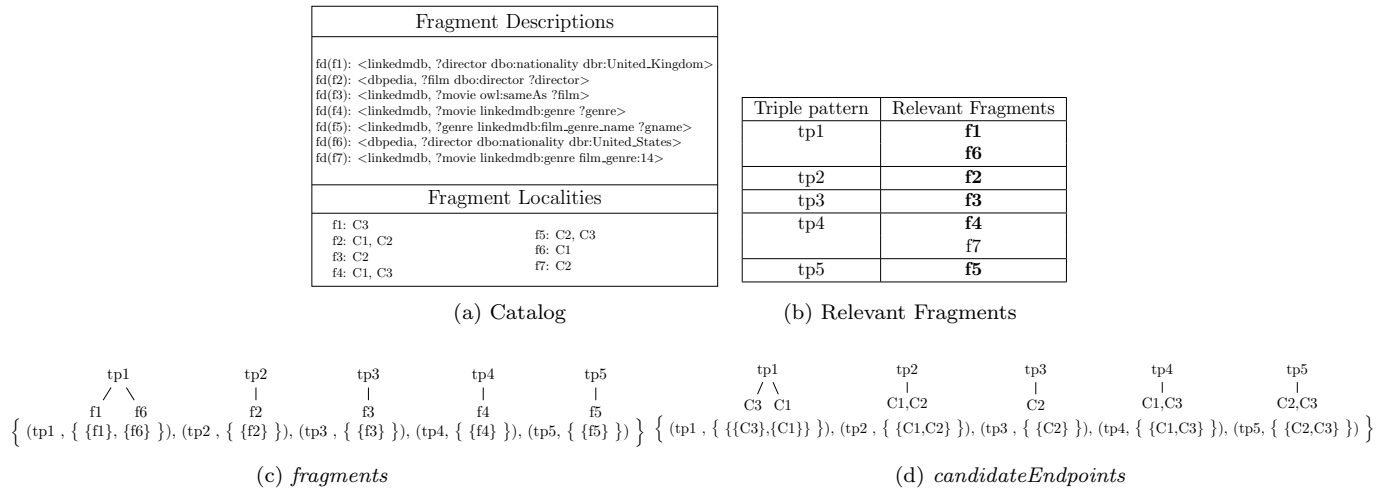
`endpoints()` provides this information. Additionally, for triple patterns that have several relevant fragments, only the endpoints that simultaneously access all relevant fragments are included in this initial endpoint selection if they exist.

For our running example, function `ReduceUnions` (Algorithm 2) uses the *fragments* (Figure 5c) to produce the *candidateEndpoints* (Figure 5d). Endpoints *C1* and *C3* are candidate endpoints for *tp1*, i.e., (*tp1*, { { *C3* }, { *C1* } }) ∈ *candidateEndpoints*; this is because (*tp1*, { {*f1*}, {*f6*} }) ∈ *selectedFragments*, *f1* is accessed by endpoint *C3*, and *f6* is accessed by endpoint *C1*. Further, even if *tp1* has two relevant sets of fragments, as they are not accessed simultaneously by any endpoint, their relevant endpoints are included as two sets of endpoints, i.e., { *C3* }, { *C1* }.

Algorithm 3 ReduceBGPs Algorithm

Require: A set *candidateEndpoints*, composed of pairs (tp, ess), with tp a triple pattern and ess a set of sets of endpoints
Ensure: A pair *(subqueries, ljtps)*, with *subqueries* and *ljtps* sets of pairs (e, tps), e an endpoint and tps a set of triple patterns
21: **function** `ReduceBGPs`(candidateEndpoints)
22: es ← ∅
23: **for** (tp, ess) ∈ candidateEndpoints **do**
24: **for** es' ∈ ess **do**
25: **for** e ∈ es' **do**
26: es ← es ⋃ { e }
27: **end for**
28: **end for**
29: **end for**
30: ljtps ← ∅
31: triples ← ∅
32: **for** e ∈ es **do**
33: tps ← ∅
34: **for** (tp, ess) ∈ candidateEndpoints **do**
35: **if** cardinality(ess)=1 **then**
36: let es' be the element of ess, i.e., ess = { es' }
37: tps ← tps ⋃ { tp }
38: **end if**
39: **end for**
40: triples ← triples ⋃ { (e, tps) }
41: **end for**
42: **for** (e, tps) ∈ triples **do**
43: **for** tp_i ∈ tps **do**
44: largest ← the largest joinable sub-set of tps that includes tp_i
45: ljtps ← ljtps ⋃ { (e, largest) }
46: **end for**
47: **end for**
48: subqueries ← `SetCoveringSubqueries`(ljtps)
49: **return** (subqueries, ljtps)
50: **end function**

After selecting candidate endpoints for the BGP triple



(a) Catalog (b) Relevant Fragments

(c) *fragments* (d) *candidateEndpoints*

Figure 5: Catalog's information and relevant fragments are used to compute *fragments* in function `SelectNonRedundantFragments` and *candidateEndpoints* in function `reduceUnions` (Algorithm 2) for the federation in Figure 1b, and query *Q1* (Figure 1c)

patterns, the function `ReduceBGPs` (Algorithm 3) conducts the following steps for the triple patterns with only one relevant fragment, i.e., (*tp*, *ess*) ∈ *candidateEndpoints* and *cardinality*(*ess*)=1:

1. For each endpoint e, the computation of the set of maximal size of joinable triple patterns that e can answer, largest, i.e., (*e*, *largest*) ∈ *ljtps*.
2. Reduction and resolution of the set covering instance to select the *minimal* number of joinable triple patterns that cover the set of triple patterns in *ljtps* with only one relevant fragment.

Computing the set of maximal size of joinable triple patterns may be done in a loop starting with singleton sets and in each iteration making the union of two sets of triple patterns if they are joinable until a fixed-point is reached.

Because the BGP reduction has been reduced to an existing optimization problem (set covering problem). This reduction allows for relying on existing algorithms for solving the set covering problem [18], and efficiently and effectively implement the function `SetCoveringSubqueries` in Algorithm 3 (line 48). However, even if using heuristics, as the one presented in [18], allows for the identification of solutions quickly it does not guarantees the generation of an optimal solution and it limits LILAC to produce only an approximate solution.

Our approximate solution to the set covering is performed on the triple patterns *tps* associated with each endpoint *e* in pairs (*e*, *tps*) ∈ *ljtps*.

For our running example, function `ReduceBGPs` (Algorithm 3) uses the *candidateEndpoints* (Figure 6a) and the joins in the BGP (Figure 6b) to produce the values of *ljtps* and *subqueries* given in Figures 6c and 6d. Since (*tp2*, {{ *C1*, *C2* }}), (*tp3*, {{*C2*}}), (*tp4*, {{*C1*,*C3*}}), (*tp5*, {{ *C2*, *C3* }}) are part of *candidateEndpoints*, the variable *tmp* is composed of the following pairs: (*C1*, { *tp2*, *tp4* }), (*C2*, { *tp2*, *tp3*, *tp5*}), (*C3*, { *tp4*, *tp5*}). Additionally, the pairs (*C1*, { *tp2* }), (*C1*, { *tp4* }), (*C2*, { *tp2*, *tp3*}), (*C2*, { *tp5* }), (*C3*, { *tp4*, *tp5* }) are part of *ljtps*. Note that because *tp2* and *tp4* do not share any variable, *tp2* and *tp4* have been included in different elements of *ljtps* with *C1*. However, *tp2* and *tp3* are included in the same set of triple patterns because they share the variable *?film*. The output of evaluating the function `SetCoveringSubqueries` is the set { (*C2*, { *tp2*, *tp3*}), (*C3*, { *tp4*, *tp5* })} because all triple patterns *tp2*, *tp3*, *tp4*, and *tp5* are covered by this set of two elements.

Algorithm 4 IncreaseSelectivity Algorithm

Require: A set *subqueries*, composed of pairs (e, tps), with e an endpoint and tps a set of triple patterns
A set *candidateEndpoints*, composed of pairs (tp, ess), with tp a triple pattern and ess a set of sets of endpoints
A set *ljtps*, composed of pairs (e, tps), with e an endpoint and tps a set of triple patterns
Ensure: a decomposition representation *decomposition*

```
51: function IncreaseSelectivity(subqueries,endpoints, ljtps)
52:   decomposition ← ∅
53:   for sq ∈ subqueries do
54:     decomposition ← decomposition ⋃ { { sq } }
55:   end for
56:   for (tp, ess) ∈ endpoints do
57:     if cardinality(ess) > 1 then
58:       sq ← ∅
59:       for es ∈ ess do
60:         select (e, ts) ∈ ljtps such that e ∈ es
                                  ∧ ts is the largest sub-query joinable with tp
61:         sq ← sq ⋃ { ( e, ts ⋃ { tp } )}
62:       end for
63:       decomposition ← decomposition ⋃ { sq }
64:     end if
65:   end for
66:   return decomposition
67: end function
```

Finally, sub-queries involving triple patterns with more than one relevant fragment are computed, and their selectivity is increased by combining these triple patterns with joinable triple patterns assigned to the same endpoint. Endpoints able to answer the largest joinable sub-query

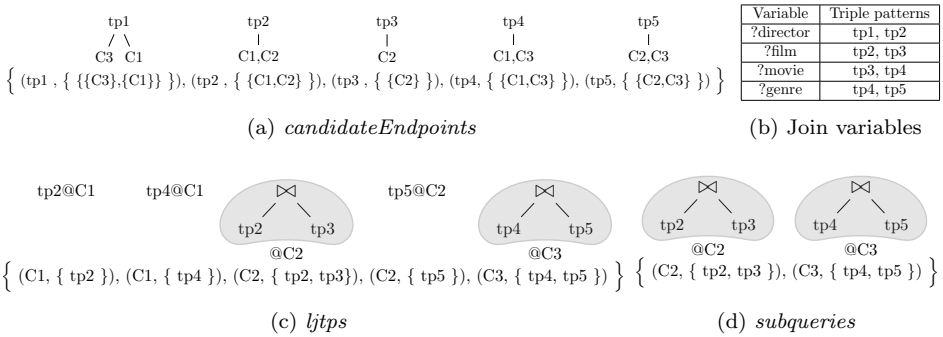


Figure 6: *candidateEndpoints* and joins in the query are used by function `reduceBGPs` (Algorithm 3) to produce the set of largest joinable triple patterns (*ljtps*) and the set of non redundant subqueries (*subqueries*) for federation in Figure 1b) and query *Q1* (Figure 1c)

.

```
select distinct *
where {
  SERVICE <C2> { tp2 . tp3 } .
  SERVICE <C3> { tp4 . tp5 } .
  {
     SERVICE <C1> { tp1 . tp2 }
  } UNION {
     SERVICE <C3> { tp1 }
  }
}
```

Figure 7: Decomposition obtained with Algorithm 1 for federation in Figure 1b, and query *Q1* (Figure 1c)

are selected, and joinable sub-queries are included in *ljtps*. The endpoints and sub-queries selected for each relevant fragment are combined and included as one element of the decomposition representation, i.e., the set *jo* is a join operand and comprises pairs (*e*, *ts*) in the query decomposition. Notice that each element of *jo* corresponds to an union operand in the query decomposition. Sub-queries already included in *subqueries*, i.e., sub-queries with just one relevant fragment in the query decomposition, are represented as singleton sets in the query decomposition.

For our running example, Figure 7 presents the decomposition computed by function `IncreaseSelectivity` (Algorithm 4). Sub-queries { *tp2 . tp3*} and { *tp4 . tp5* } are evaluated at endpoints *C2* and *C3*, respectively. Further, since the pair (*C1*, { *tp2* }) belongs to *ljtps*, the pair (*tp1*, { { *C1* }, { *C3* }}) is part of *candidateEndpoints*, and the triple patterns *tp1* and *tp2* share the variable *?director*, the sub-query { *tp1*, *tp2* } is evaluated at *C1* and the sub-query { *tp1* } is evaluated at *C3* and the union of their resulting mappings is produced by the federated query engine. As proposed in Section 2, a decomposition with selective sub-queries has been produced by Algorithm 1.

Proposition 1. Algorithm 1 has a time complexity of $\mathcal{O}(n^3m^2k)$, with n the number of triple patterns in the query, m the number of fragments, k the number of endpoints.

Details about time complexity are available in Appendix A.

Theorem 1 *If all the RDF data accessible through the federation endpoints are described as replicated fragments,* LILAC *query decomposition produces a solution to the QDP-FR problem, i.e.,* LILAC *generates a query decomposition that satisfies the answer soundness, answer completeness, and data transfer minimization properties stated in Section 3.2.*

The proof of Theorem 1 is available in Appendix B.

5. Experimental Study

The goal of the experimental study is to evaluate the effectiveness of LILAC. We compare the performance of the federated query engines FedX and ANAPSID with respect to their extensions using DAW, Fedra, and LILAC, i.e., the evaluation compares FedX, DAW+FedX, Fedra+FedX, and LILAC+FedX, as well as ANAPSID, DAW+ANAPSID, Fedra+ANAPSID, and LILAC+ANAPSID. The aim of extending both FedX and ANAPSID with LILAC is to show that LILAC is not tailored for one federated query engine, i.e., LILAC strategy works well with different engines. Moreover, it should be noted that our experimental study does not aim to compare federated query engines FedX and ANAPSID, because their source selection strategies have very different warranties, i.e., FedX source selection *theoretically* ensures answer completeness, while ANAPSID may prune relevant sources that are required to produce a complete answer in order to reduce the execution time. We expect to see that LILAC improves these federated query engines performance in terms of number transferred tuples, number of selected sources, and execution time. Also, we hypothesize that LILAC does not significantly degrade the performance of federated query engines in terms of decomposition time and completeness. Moreover, LILAC+ANAPSID *theoretically* produces complete answers, i.e., LILAC may increase query answer completeness of ANAPSID.

Datasets: We use four real datasets: Diseasome, Semantic Web Dog Food (SWDF), Linked Movie Data Base

Table 2: Dataset characteristics: version, number of different triples (# DT), predicates (# P), and coherence [10]

Dataset	Version date	# DT	# P	Coherence
Diseasome	19/10/2012	72,445	19	0.32
SWDF	08/11/2012	198,797	147	0.23
DBpedia Geo	06/2012	1,900,004	4	1.00
LinkedMDB	18/05/2009	3,579,610	148	0.81
WatDiv1	-	104,532	86	0.44
WatDiv100	-	10,934,518	86	0.38

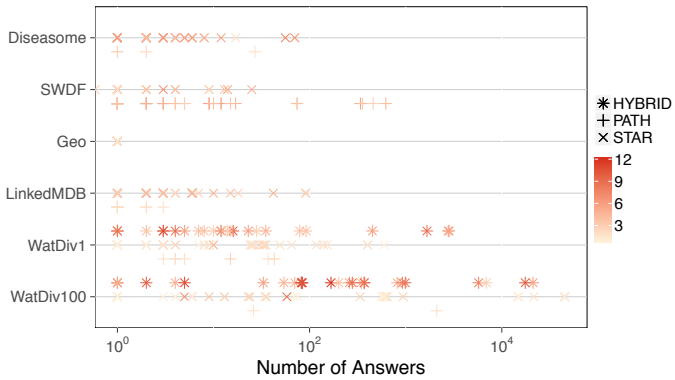


Figure 8: Characteristics of the randomly selected queries evaluated in the experimental study. The number of triples in the queries goes from 1 up to 14, their shapes are PATH, STAR, and HYBRID, and their number of answers goes from 1 up to 45,129. X-axis is presented using logarithmic scale

(LinkedMDB), and DBpedia Geo-coordinates (Geo). Further, we consider two instances of the Waterloo SPARQL Diversity Test Suite synthetic dataset [2, 3] with 10^5 and 10^7 triples (WatDiv1 and WatDiv100). Table 2 shows the characteristics of these datasets. These datasets have sizes varying between 72 thousand triples up to 10 million triples, their number of predicates goes from just four predicates for highly structured dataset Geo, until 148 predicates for quite structured dataset LinkedMDB. We used the coherence metric [10] to measure the structuredness of the datasets. The structuredness indicates how well instances conform to their types, coherence values range between 0.0 and 1.0, e.g., a dataset where all the type instances have the same attributes has a coherence value of 1.0, and a dataset where all the type instances have different attributes has a coherence value of 0.0.

Queries: We generate 50,000 queries from 500 templates for the WatDiv datasets. These queries have between 1 and 14 triple patterns, PATH, STAR, and HYBRID (called Snowflake in [2, 3]) shapes. For the real datasets, we generate PATH and STAR shaped templates with two to eight triple patterns, and they are instantiated with random values from the datasets to generate over 10,000 queries per dataset. Figure 8 presents the characteristics of the 100 randomly selected queries to be executed for each of the six datasets.

All the queries have the DISTINCT modifier, i.e., duplicated variable mappings are not included in the query answers. In this way, the results of evaluating these queries will be not affected by not collecting replicated data from the relevant sources.

Fragment Replication: For each dataset, we set up a federation with ten SPARQL endpoints. Each endpoint can access data from replicated fragments that are relevant for 100 random queries; the same fragment is replicated in at most three endpoints. Fragments are created from triple patterns in these 100 randomly selected queries. SPARQL construct queries are executed to populate these fragments; data is collected using the LDF client[10] against a local LDF server that hosts the corresponding datasets.

Implementations: Source selection and query decomposition components of FedX 3.1[11] and ANAPSID[12] have been modified. First, FEDRA and DAW [28] replaced FedX and ANAPSID source selection strategies; we name these new engines, FEDRA + FedX, DAW + FedX, FEDRA + ANAPSID, and DAW + ANAPSID, respectively. Likewise, these engines were also modified to execute the LILAC source selection and query decomposition strategies; LILAC + FedX and LILAC + ANAPSID correspond to these new versions of the engines. Moreover, FedX was modified to include an heuristic that produces plans free of Cartesian products whenever it is possible. Cartesian product free plans are positively impacted by good query decomposition strategies as we have argued in Section 2. Because FedX is implemented in Java and ANAPSID is implemented in Python, LILAC, FEDRA, and DAW[13] are implemented in both Java 1.7 and Python 2.7.3. Thus, LILAC, FEDRA and DAW are integrated in FedX and ANAPSID, reducing the performance impact of including these strategies. The code and experimental setup are available at `https://github.com/gmontoya/lilac`.

Hardware and configuration details: The Grid'5000 testbed[14] is used to run the experiments. In total 11 machines Intel Xeon E5520 2.27 GHz, with 24GB of RAM are used. Ten machines are used to host the SPARQL endpoints, and one to run the federated query engines. Federated query engines use up to 7GB of RAM. SPARQL endpoints are deployed using Virtuoso 7.2.1[15]. Virtuoso parameters *number of buffers* and *maximum number of dirty buffers* are set up to 1,360,000 and 1,000,000, respectively. Virtuoso maximum number of result rows is setup to 100,000 and the maximum query execution time and maximum query cost estimation are set up to 600 seconds.

Evaluation Metrics: *i) Number of Selected Sources (NSS):* is the sum of the number of sources that has been selected per query triple pattern. *ii) Query Decomposition and Source Selection Time (DST):* is the elapsed time since

[10] https://github.com/LinkedDataFragments, March 2015.
[11] `http://www.fluidops.com/fedx/`, September 2014.
[12] `https://github.com/anapsid/anapsid`, September 2014.
[13] We had to implement DAW as its code is not available.
[14] `https://www.grid5000.fr`
[15] `https://github.com/openlink/virtuoso-opensource/releases/tag/v7.2.1`, June 2015.

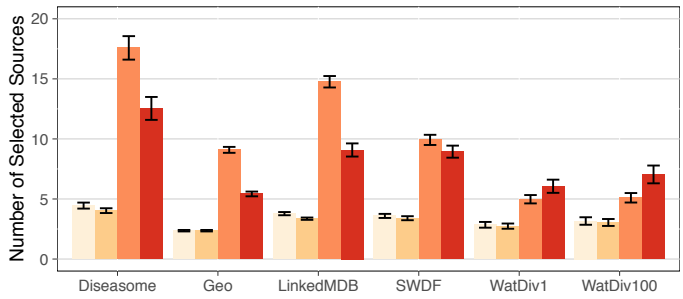


Figure 9: Number of Selected Sources for LILAC+ANAPSID ( ), Fedra+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

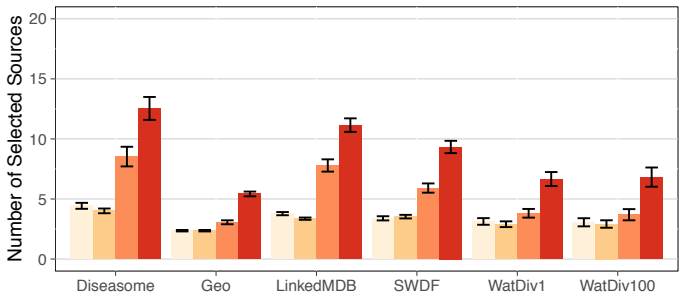


Figure 10: Number of Selected Sources for LILAC+FedX ( ), Fedra+FedX ( ), DAW+FedX ( ) and FedX ( )

the query is posed until the query decomposition is produced. *iii*) *Execution Time (ET):* is the elapsed time since the query is posed until the complete answer is produced. We used a timeout of 1,800 seconds. *iv*) *Completeness (C):* is the ratio between the number of query answers produced by the engine and the number of answers produced by the evaluation of the query over the set of all the triples available in the federation; values range between 0.0 and 1.0. *v*) *Number of Transferred Tuples (NTT):* is the number of tuples transferred from all the endpoints to the engine during a query evaluation. Endpoints are accessed through proxies that count the number of transferred tuples. [16] Presented values correspond to the average of the execution of 100 random queries per each of the six federations.

Statistical Analysis: The Wilcoxon signed rank test [36] for paired non uniform data is used to study the significance of the improvements on performance obtained when LILAC decomposition is used. [17]

5.1. Source Selection

Experiments were conducted against the federations with replicated fragments and the 600 queries described in the previous section, i.e., 100 random queries for each of the six federations. The aim of this section is to report on the results of the query decomposition and source selection time (DST), as well as on the number of selected sources (NSS). Although the proposed approach LILAC requires more time for source selection and query decomposition, i.e., LILAC exhibits a higher DST, it allows for reducing the number of selected sources considerably, i.e., the NSS is lower for LILAC. Moreover, the achieved reduction has a great impact on the query execution time (ET).

5.1.1. Number of Selected Sources

Figures 9 and 10 show the performance of the engines in terms of the number of selected sources. LILAC+FedX and LILAC+ANAPSID select significantly less sources than FedX, DAW+FedX, ANAPSID, and DAW+ANAPSID, respectively. These results are natural because LILAC uses the knowledge present in the catalog to precisely identify multiple endpoints that access the same fragments and fragments that only provide redundant data. Contrarily to LILAC, the engines are not aware of the replicated fragments, and consequently, they may incur in redundant data accesses. Even a duplicate-aware technique like DAW, is not capable of achieving a reduction as great as the one achieved by LILAC. The behavior exhibited by DAW is the consequence of its duplicate detection technique, which is based on the overlap between summaries and cannot provide a precise description about the replicated fragments. However, LILAC+FedX and LILAC+ANAPSID select slightly more sources than Fedra+FedX and Fedra+ANAPSID, respectively. LILAC may assign one triple pattern to more than one endpoint to increase the selectivity of the sub-queries in the decomposition. On the other hand, Fedra selects endpoints to facilitate the query decomposition of the federated query engines. Thus, Fedra chooses the smallest number of endpoints to evaluate each triple pattern and enables FedX and ANAPSID to build decompositions that include exclusive and star-shaped groups, respectively.

To confirm that LILAC+FedX and LILAC+ANAPSID select less sources than FedX, DAW+FedX, ANAPSID, and DAW+ANAPSID, respectively, Wilcoxon signed rank tests were run with the hypotheses:

H0: LILAC+FedX (LILAC+ANAPSID) selects the same number of sources as FedX and DAW+FedX (resp., ANAPSID and DAW+ANAPSID) do.

H1: LILAC+FedX (LILAC+ANAPSID) selects less sources than FedX and DAW+FedX (resp., ANAPSID and DAW+ANAPSID) do.

Test p-values are presented in Tables C.5 and C.6, Appendix C. For all the federations and engines, p-values are inferior to 0.05. These low p-values allow for rejecting the null hypothesis that LILAC+FedX, LILAC+ANAPSID selected the same number of sources than the other studied engines. Additionally, they support the acceptance of the alternative hypothesis that LILAC+FedX and

[16] Proxies are implemented in Java 1.7. using the Apache HttpComponents Client library 4.3.5 (`https://hc.apache.org/`).

[17] The Wilcoxon signed rank test was computed using the R project (`http://www.r-project.org/`).

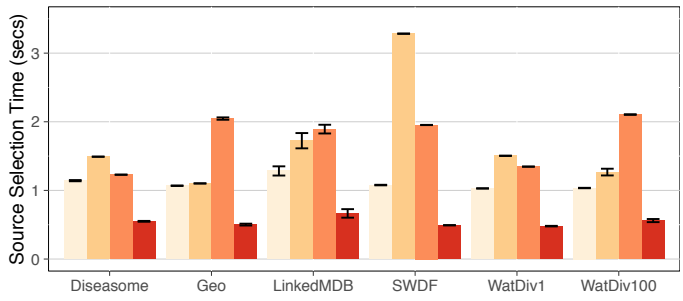


Figure 11: Query Decomposition and Source Selection Time (in secs) for LILAC+ANAPSID ( ), FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

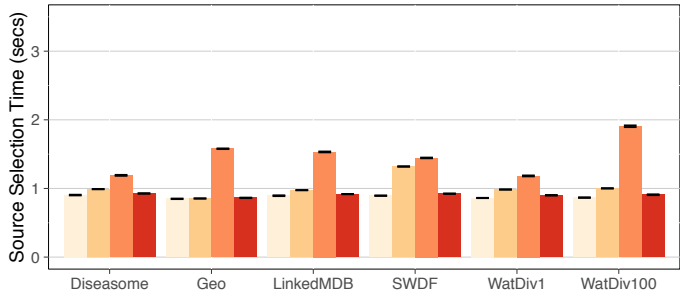


Figure 12: Query Decomposition and Source Selection Time (in secs) for LILAC+FedX ( ), FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

LILAC+ANAPSID reduce the number of sources selected by FedX and ANAPSID, respectively; the reductions achieved by LILAC+FedX and LILAC+ANAPSID are greater than the ones achieved by DAW+FedX and DAW+ANAPSID, respectively.

5.1.2. Source Selection and Query Decomposition Time

Figures 11 and 12 show the performance of the engines in terms of query decomposition and source selection time (DST). DST for LILAC+FedX is significantly better than DST for FedX and DAW+FedX, while DST for ANAPSID is significantly better than DST for LILAC+ANAPSID, FEDRA+ANAPSID, and DAW +ANAPSID. For FedX and DAW the source selection takes a considerable amount of time, because during the source selection phase, FedX contacts all the endpoints for each triple pattern to check if the endpoints access relevant data for the triple pattern. Similarly, DAW conducts an exhaustive search and computes the overlapping data of the endpoint to prune endpoints that only provide redundant data. Contrary, LILAC and FEDRA exploit metadata stored in their catalogs to speed up source selection, while ANAPSID uses heuristics to rapidly prune sources that are not likely to provide relevant data for the query at the cost of answer completeness.

To confirm that DST for LILAC+FedX and LILAC+ANAPSID are lower than for FedX, DAW+FedX, and DAW+ANAPSID, respectively, Wilcoxon signed rank tests were run with the hypotheses:

Table 3: Number of queries that timed out and that aborted execution

	WatDiv100	
	# timeouts	# aborted
LILAC+FedX	0	0
FEDRA+FedX	0	2
DAW+FedX	0	2
FedX	5	4
LILAC+ANAPSID	0	0
FEDRA+ANAPSID	0	0
DAW+ANAPSID	0	0
ANAPSID	5	0

H0: LILAC+FedX's (LILAC+ANAPSID's) DST is the same as for FedX and DAW+FedX (DAW+ANAPSID).
H1: LILAC+FedX's (LILAC+ANAPSID's) DST is lower than for FedX and DAW+FedX (DAW+ANAPSID).

Test p-values are presented in Table C.7, Appendix C. For all the federations and engines, p-values are inferior to 0.05. These low p-values allow for rejecting the null hypothesis that DAW, FedX, and LILAC query decomposition and source selection times are similar. Additionally, they support the acceptance of the alternative hypothesis that LILAC source selection and query decomposition strategies are faster than DAW and FedX source selection techniques.

5.2. Query Execution

We consider the same federations and queries from the previous section and study their execution performance in terms of execution time (ET), answer completeness (C), and number of transferred tuples (NTT). As the query executions that aborted or timed out do not have significance in terms of the studied metrics, these executions have been removed from the results; Table 3 summarizes the removed queries. Using LILAC in combination with the federated query engines reduces the number of queries that time out or finish with an error. This is mostly a natural consequence of the reduction of the number of transferred tuples (NTT) achieved by LILAC.

5.2.1. Execution Time

Figures 13 and 14 show the performance of the engines in terms of execution time (ET). Execution time (ET) for LILAC+FedX is better for all the datasets except GeoCoordinates. Moreover, execution time (ET) for LILAC+ANAPSID is better for all the datasets except for GeoCoordinates and WatDiv1. The GeoCoordinates dataset is characterized by high structuredness, i.e., coherence for GeoCoordinates is 1.00 (Table 2); additionally, GeoCoordinates queries are very selective STAR shape queries. Moreover, most of the executed queries over GeoCoordinates and WatDiv1 have an execution time of around 1 second. In this type of scenarios, FedX and ANAPSID are already capable of generating execution

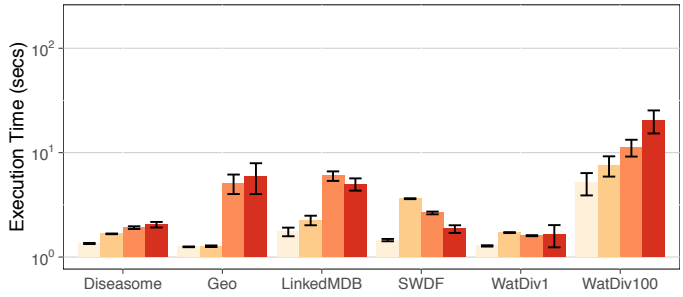


Figure 13: Execution Time (in secs) for LILAC+ANAPSID ( ), Fedra+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

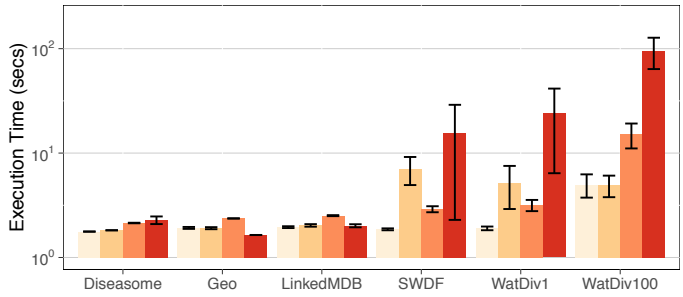


Figure 14: Execution Time (in secs) for LILAC+FedX ( ), Fedra+FedX ( ), DAW+FedX ( ) and FedX ( )

plans that reduce the number of transferred tuples and intermediate results. Consequently, when executing "easy" queries it may not be worthy to include (more) expensive source selection and decomposition strategies as the one provided by LILAC. To test this conjecture, we consider only the queries that FedX and ANAPSID executions transfer at least 100 tuples. The number of considered queries is 49 for ANAPSID and WatDiv1 federation, seven for ANAPSID and GeoCoordinates federation, and zero for FedX and GeoCoordinates federation. Seven queries are not enough to draw a conclusion about the significance of the execution time reduction by LILAC on GeoCoordinates queries, but 49 queries are enough to compare the execution time (ET) achieved by ANAPSID and its extensions with DAW, Fedra, and LILAC. Figure 15 presents the execution time (ET) for these 49 queries. For queries with at least 100 transferred tuples, LILAC+ANAPSID reduces considerably the execution time with respect to ANAPSID; the reductions achieved by LILAC+ANAPSID are greater than the ones achieved by DAW+ANAPSID and Fedra+ANAPSID.

Furthermore, the greatest reduction in execution time is observed in SWDF, WatDiv1 and WatDiv100 for FedX, and GeoCoordinates for ANAPSID for non-selective queries. These combinations of federations and queries are precisely those where the engines transfer the largest number of tuples. These observations in combination with the previous statement about query "easiness", provide evidence of a correlation between the number of transferred tuples and the execution time. Moreover, the

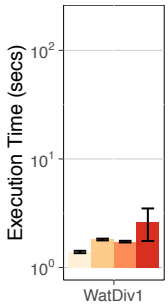


Figure 15: Execution Time (in secs) for LILAC+ANAPSID ( ), Fedra+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( ) for queries with at least 100 intermediate results

benefits that can be obtained from using LILAC may lastly depend on the number of transferred tuples incurred by the engines.

To confirm that LILAC query decomposition techniques significantly reduce the execution time (ET) of the studied engines, a Wilcoxon signed rank test was run with the hypotheses:
H0: using LILAC query decomposition does not change the engine query execution time.
H1: LILAC query decompositions lead to query executions faster than the engines.

Test p-values are presented in Table C.8, Appendix C. With the exception of GeoCoordinates with LILAC+FedX and LILAC+ANAPSID, and WatDiv1 with LILAC+ANAPSID, all the p-values are inferior to 0.05. These low p-values allow for rejecting the null hypothesis that the execution time of LILAC+FedX (resp., LILAC+ANAPSID) and FedX (resp., ANAPSID) are the same. Additionally, they support the acceptance of the alternative hypothesis that LILAC+FedX and LILAC+ANAPSID have lower execution time (ET). Furthermore, we repeated the test for the WatDiv1 federation and the ANAPSID based engines using only the 49 queries that transfer at least 100 tuples (execution times given in Figure 15); a p-value of **0.01454** was obtained. This low p-value allows for rejecting the null hypothesis and accepting the alternative hypothesis that LILAC enhances ANAPSID and allows for a significant reduction on the execution time for the WatDiv1 federation for queries with at least 100 intermediate results.

To confirm that the engines enhanced with LILAC achieve a greater reduction in execution time than the engines enhanced with DAW, a Wilcoxon signed rank test was run with the hypotheses:
H0: using LILAC or DAW in combination with the engines leads to similar execution times.
H1: LILAC decompositions lead to query execution times faster than source selection DAW.

Test p-values are presented in Table C.9, Appendix C. For all the federations and engines, p-values are inferior to 0.05. These low p-values allow for rejecting the null hypothesis that the execution times when using LILAC

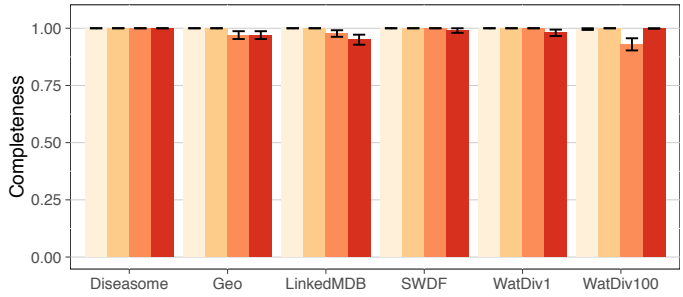


Figure 16: Completeness for LILAC+ANAPSID ( ), FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

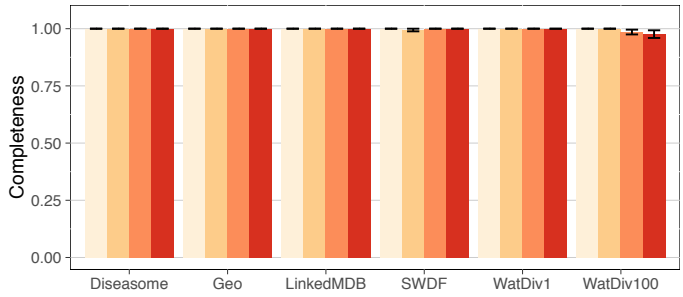


Figure 17: Completeness for LILAC+FedX ( ), FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

query decomposition strategy and when using the engines enhanced with DAW are similar, and accepting the alternative hypothesis that LILAC reduction in the execution time is greater. As expected, the fragment descriptions used by LILAC *pave the way for* better characterization of the replicated fragments accessible by the endpoints, and consequently, LILAC selects significantly less sources than DAW. Furthermore, LILAC identifies endpoints that are able to execute larger sub-queries and reduces both number of transferred tuples (NTT) and execution time (ET).

5.2.2. Completeness

Figures 16 and 17 show the performance of the engines in terms of completeness. LILAC+FedX and LILAC+ANAPSID are able to produce complete answers in all cases except for one query against WatDiv100 and LILAC+ANAPSID. In this particular query, ANAPSID selects a physical operator that transfers all the mappings matching each sub-query. Because there is one triple pattern with the general predicate *rdf:type* that has 136,012 mappings, the execution of this sub-query exceeds the maximal number of returned tuples by the endpoints (100,000), and LILAC+ANAPSID then fails to produce a complete answer for this query. It should be noticed that in a federation where endpoints have a maximum for the number of results that can be transferred, even approaches such as FedX that theoretically always provide complete answers are no longer able to ensure completeness in practice. For example, FedX has completeness 0.0 for two

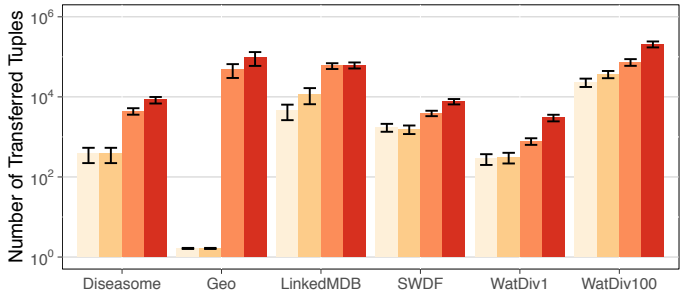


Figure 18: Number of Transferred Tuples during execution of LILAC+ANAPSID ( ), FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

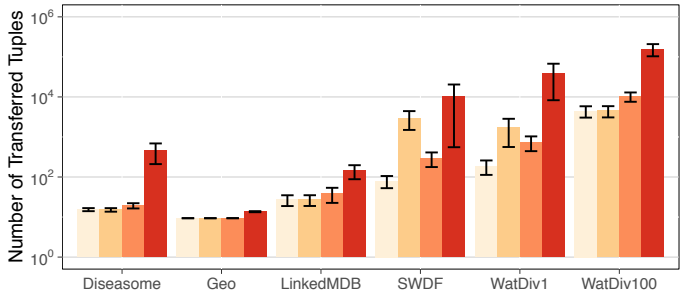


Figure 19: Number of Transferred Tuples during execution of LILAC+FedX ( ), FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

queries as shown in Figure 17.

5.2.3. Number of Transferred Tuples

Figures 18 and 19 show the performance of the engines in terms of number of transferred tuples (NTT). LILAC+FedX and LILAC+ANAPSID transfer less tuples than the other approaches; similar behavior is observed for FEDRA+FedX and FEDRA+ANAPSID, in the federations with data from datasets Diseasome, GeoCoordinates, and LinkedMDB. These real datasets have little in common, i.e., their sizes, numbers of predicates, and coherence values are very different. However, the random queries executed in these federations have some common characteristics, they are mainly selective STAR shape queries (cf. Figure 8). To confirm that LILAC reduces the number of transferred tuples by the engines and that LILAC reduction is greater than the reduction achieved by DAW, Wilcoxon signed rank tests were run with the hypotheses:

H0: FedX, ANAPSID, DAW+FedX, DAW+ANAPSID, LILAC+FedX, and LILAC+ANAPSID transfer the same number of tuples.

H1: LILAC+FedX and LILAC+ANAPSID transfer less tuples than FedX, ANAPSID, DAW+FedX, and DAW+ANAPSID.

Test p-values are presented in Table C.11, Appendix C. With the exception of GeoCoordinates with LILAC+FedX and DAW+FedX, p-values are inferior to 0.05. These low p-values allow for rejecting the null

Table 4: Limitations of existing approaches and impact on obtained results for Number of Selected Sources (NSS), Query Decomposition and Source Selection Time (DST), Execution Time (ET), Completeness (C), and Number of Transferred Tuples (NTT). ✓ indicates that the approach exhibits the limitation and ✗ indicates that the approach is free of the limitation

Limitations	Approach					Impacted Metrics
	FedX	ANAPSID	DAW	FEDRA	LILAC	
Has no knowledge about replication	✓	✓	✓	✗	✗	NSS, NTT, ET
Performs very expensive computations	✗	✗	✓	✗	✗	DST
Is unaware of BGPs or joins	✓	✓	✓	✗	✗	NTT
Uses completeness risking heuristics	✗	✓	✗	✗	✗	C
Performs uninformed pruning	✗	✗	✗	✓	✗	NTT
Is too expensive for "easy" queries	✗	✗	✓	✓	✓	DST,ET
Is an approximate solution	✗	✗	✓	✓	✓	ET, NTT
Relies on endpoints for completeness	✓	✓	✓	✓	✓	C

hypothesis that the engines enhanced with LILAC and enhanced with DAW transfer similar number of tuples. Additionally, they support the acceptance of the alternative hypothesis that LILAC reduction in the number of transferred tuples is greater for all the combinations of federations and engines, except by GeoCoordinates federation and FedX. GeoCoordinates queries are STAR-shaped queries with low number of triple patterns and a very limited number of answers and transferred results when executed with FedX (cf., Figures 8 and 19). It is only in this "easy" setup that LILAC does not outperform DAW in terms of number of transferred tuples. We also performed a Wilcoxon signed rank test to check if the reduction achieved by DAW is greater than the reduction achieved by LILAC in combination with FedX and a p-value of 0.9772 was obtained for this test. Therefore, for GeoCoordinates federation and FedX engine, both DAW and LILAC achieved reductions in the number of transferred tuples, but it cannot be stated that one of them is significantly better than the other.

The observed results and p-values have confirmed that LILAC does enhance both FedX and ANAPSID, and the new federated query engines LILAC+FedX and LILAC+ANAPSID are capable of finding better execution plans that transfer significantly less tuples than FedX and ANAPSID. Knowledge about the replicated fragments in LILAC catalog *paves the way for* a reduction on the number of selected sources and allows for delegating join execution to the endpoints whenever is possible.

5.3. Discussion

Obtained results confirm that LILAC provides a better solution to the QDP-FR than existing approaches. Table 4 shows the limitations exhibited by LILAC and existing approaches and the metrics impacted by these limitations. Because FedX, ANAPSID, and DAW are unaware of the replicated fragments, the Number of Selected Sources (NSS), the Number of Transferred Tuples (NTT), and the Execution Time (ET) are negatively impacted and these approaches select sources that provide redundant data. Because DAW computes the similarity among data exposed by different endpoints, the Query Decomposition and Source Selection Time (DST) is negatively impacted and the DST of DAW+FedX are higher than the ones of the other FedX-based approaches. Because FedX and DAW do not take into account how the triple patterns are connected in the query, the Number of Transferred Tuples (NTT) is negatively impacted and FedX plans may include sub-queries with Cartesian products and DAW may select different sources to evaluate triple patterns in a join that can be locally evaluated in one source. Because ANAPSID implements an heuristic to select the most likely source to have relevant data, the Completeness (C) is negatively impacted and ANAPSID may fail to produce some of the query answers. Because FEDRA performs a replication-aware source selection and is tailored to work with existing query decomposers, the Number of Transferred Tuples (NTT) is negatively impacted and FEDRA has to select just one source per triple pattern if possible to allow for delegating join evaluation to the endpoints. Moreover, doing this may prevent the generation of plans that have lower number of transferred tuples. Because DAW, FEDRA, and LILAC have more knowledge about the data distribution on the federation, using this knowledge to produce better source selections that reduce the transfer of redundant data negatively impacts on the Query Decomposition and Source Selection Time (DST). Moreover, for "easy" queries where the DST domains the Execution Time (ET), i.e., queries that transfer very few tuples from endpoints to the engine for any source selection, the ET is also negatively impacted because only negligible reductions are attainable. Because DAW relies on estimations based on data summaries and FEDRA and LILAC rely on heuristic solutions to the set covering problem, the Number of Transferred Tuples (NTT) and the Execution Time (ET) are negatively impacted. These approaches only produce approximate solutions. Therefore, they risk retrieving redundant data or delegating less joins to the endpoints that actually possible, and consequently exhibiting a higher NTT or ET than possible. Finally, because all the approaches rely on endpoints providing all the results for the evaluated sub-queries and endpoints only send a limited number of results, the Completeness (C) is negatively impacted and the engines may fail to produce some of the answers.

6. Related Work

6.1. Distributed Databases and Data Fragmentation and Replication

A distributed database corresponds to a *logically unified* database which is *physically* partitioned into multiple portions or *fragments* distributed across a computer network [24]. Fragments can be replicated over a distributed database to improve *data availability* and to enhance *scalability* and *performance* of the data management system that provides the access to the distributed database. Queries and frequency of general data management operations are known before hand. Thus, a distributed database can be physically designed to *speed up* its expected work load, i.e., fragments and replication are tailored for a particular application.

The Linking Open Data (LOD) cloud [7] is a *logically unified* dataset composed of Linked Data sets publicly available on the Web and accessible following specific Web access interfaces, e.g., SPARQL endpoints or Triple Pattern Fragments (TPFs). Albeit logical similarities, the LOD cloud cannot be considered a distributed database. LOD datasets are published by autonomous participants and *Linked Data availability* cannot be always ensured [4]. Different approaches have been defined to empower data consumers to access federations of LOD datasets [27], e.g., federated query engines. However, there is no known set of queries to be executed over the LOD cloud; in consequence, a general physical Linked Data distribution designed to enhance the *performance* of a particular Linked Data application cannot be achieved.

Albeit the differences between distributed databases and the LOD cloud, data fragmentation and replication techniques could be reused to enhance the LOD cloud applications performance. However, because of the lack of information about the cloud work load, traditional data fragmentation and replication design [24] and distributed query processing approaches [19] require to be redefined to ensure *Linked Data availability* [31].

LILAC relies on a data distribution strategy tailored for Linked Data. This distribution relies on data consumers that, having interest on certain federation of LOD datasets, deploy new endpoints to provide access for replicated fragments of these LOD datasets. Thus, even if the LOD datasets become unavailable, *Linked Data availability* can be still ensured by the data consumers. A replicated fragment corresponds to a set of the RDF triples that are part of one dataset in the LOD cloud and satisfy a given triple pattern. Descriptions of the replicated fragments are kept in the LILAC catalog and the fragment containment relation is computed to determine fragment overlap; reducing thus the retrieval of redundant data and enhancing the *performance* of Linked Data applications.

6.2. Linked Data Fragmentation

The problem of accessing fragments of Linked Data has been recently addressed in the Semantic Web community, and approaches to support fragment-aware data management have been defined [15, 17, 33].

Verborgh et al. [15, 33] proposed client-side query processing techniques, named Linked Data Fragment (LDF), to *opportunistically exploit* fragments that materialized set of triples of a LOD dataset. Fragments are accessible through specific Web access interfaces named Triple Pattern Fragments (TPFs) [33], and TPF servers make available the fragments on the Web. LDF improves *Linked Data availability* by moving query execution load from servers to clients. During query processing, TPF clients download fragments required to execute queries from TPF servers through simple HTTP requests, and then locally execute the queries. This strategy allows clients to cache fragments locally and decreases the load on the TPF server. LDF chooses a clear tradeoff by shifting query processing to clients, at the cost of slower query execution [33].

LDF restricts the operations that TPF servers perform, and relies on TPF clients to perform SPARQL operations. Additionally, fragments are accessed by using the TPF Web access interfaces and have not been designed for compatibility with the SPARQL protocol. Contrarily, LILAC relies on data consumer endpoints that implement the SPARQL protocol; thus, federated query engines can be used for federated query processing. Both approaches propose strategies to improve *Linked Data availability*, but while in LDF the cost of query execution is individually assumed by one client, in LILAC, the client shares this cost across the data consumer endpoints.

Ibáñez et al. [17] also assume that RDF data from LOD datasets is fragmented and replicated by data consumers; however, fragments can be updated, i.e., data consumers and providers can edit the fragments. Thus, Ibáñez et al. proposed data synchronization techniques, named Col-Graph, that allow for the synchronization of changes in the fragments in the consumer-side, i.e., fragments are kept up to date in the presence of concurrent editions.

LILAC assumes that all replicated fragments are replicas of portions of LOD datasets, i.e., all the replicas of a given fragment comprise the same RDF data. This assumption can be unrealistic for LOD datasets that change regularly, e.g., DBpedia Live. In consequence, techniques as the ones proposed by Ibáñez et al. in Col-Graph can be used to enhance LILAC for managing scenarios where replicated fragment triples are not the same as in the LOD dataset, i.e., replicated fragments with *divergence*. For instance, potential source selection criteria in this scenario, would be choosing endpoints that access fragments with the lowest divergence or pruning fragments with divergence greater than a given threshold. A preliminary solution to this problem is outlined in [21]. However, managing divergent replicated fragments is out of the scope of LILAC.

6.3. Source Selection in Federations of SPARQL Endpoints

Different techniques for source selection have been implemented into existing federated query engines. Some of them are based on runtime contacts to the endpoints, i.e., execution of SPARQL `ASK` queries [1, 26, 30]. Other approaches rely on indexes, precomputed or built at runtime, that provide some statistics about the triples accessible by the endpoints [1, 12, 26]. DARQ [26] relies on both an index based structure on source descriptions and basic statistics, e.g., number of triples per predicate, to identify the sources that can be used to evaluate triple patterns with a given predicate. For each query triple pattern, FedX resorts to the evaluation of SPARQL `ASK` queries for determining the endpoints relevant for a triple pattern, i.e., endpoints that access at least one triple that match the triple pattern. SPLENDID [12] combines both lookups to precomputed VOID indexes with runtime endpoint contacts. Endpoints are *initially selected* according to the bindings in the triple patterns. For instance, triple patterns with a bound predicate, or triple patterns with bound predicate rdf:type and bound object are looked up in the index; however, all the possible endpoints are selected for triple patterns with an unbound predicate. For triple patterns with a bound subject or object, this initial source selection is refined by contacting the selected endpoints, thus, the set of selected sources may be reduced. ANAPSID [1] maintains for each endpoint in a federation, the predicates that can be evaluated by these endpoints, i.e., endpoints are described in terms of predicates. These endpoint descriptions are used to perform an initial source selection. Additionally, heuristics [23] [18] are followed to reduce the initial set of selected sources and relevant endpoints are reduced according to the bindings of the triple patterns. For example, if the predicate bindings share the same namespace with predicates associated with an endpoint description, then this endpoint is selected.

Saleem et al. [29] proposed a join-aware source selection strategy, named HiBISCuS. HiBISCuS is based on pruning sources that cannot contribute to produce query answers. It uses an index with the predicates and the authority URIs of the described resources in the triples accessible by the federation endpoints. The authorities are used to prune sources that either do not match any of the query triple patterns, or that when considering the query joins would produce an empty join evaluation. HiBISCuS has been combined with existing federated query engines such as FedX and SPLENDID, and it has been shown to successfully reduce the number of selected sources.

These existing source selection strategies do not have knowledge about replication of fragments in the federation of endpoints. Hence, they should select all the relevant sources in order to produce complete answers. We recognize that existing source selection techniques may be used to enhance LILAC source selection. However, the main goal of this paper is to show how by exploiting knowledge about replicated fragments in concise source descriptions such as, fragment containment relations, good query decompositions and source selections can be identified.

6.4. Source Selection in Federations with Duplicated and Replicated Data

Recently, source selection strategies aware of triple duplication have been proposed [16, 28]. These approaches use data summaries or *sketches* to estimate the overlapping among sources. Benefit-Based Query routing (BBQ) [16] extends ASK queries with Bloom filters [8] that provide a summary of the results, in order to prune sources that provide low number of triple pattern mappings, i.e., endpoints with low benefit. DAW [28] uses a combination of Min-Wise Independent Permutations (MIPs) [9] 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 a predefined benefit threshold, are discarded and not selected.

BBQ [16] and DAW [28] address closely related issue of data duplication; however, data replication is not considered by these approaches. In data duplication, triples are copied without following a replication schema. Thus, more detailed information about the triples accessible through the endpoints is needed to be able to assess overlapping among sets of triples and to prune sources that do not provide access to any further triple. The accuracy of these approaches relies on the precise description of the metadata about their accessible triples and overlapping estimations. Because keeping all this information is not always feasible, these approaches may be too expensive and not accurate enough in the context of replicated fragments of LOD datasets. Additionally, these approaches perform source selection at triple pattern level, i.e., sources are selected for each triple pattern without taking into account that the triple pattern belongs to a BGP. Contrarily, LILAC exploits knowledge encoded in the fragment description, e.g., the fragment containment relation, and precisely identifies sources that have replicated the same fragments. This knowledge is used to choose among these replicas, the ones that can evaluate more query joins in order to reduce the number of transferred tuples, i.e., the source selection is performed at the BGP level. Lastly, these approaches prune sources with duplicated data during source selection and may prevent the query decomposer from producing the most selective sub-queries. Contrarily, LILAC interleaves replication-aware source selection and query decomposition. This allows the query decomposer to take advantage of opportunities to delegate more selective sub-queries to the endpoints.

[18] In practice, these heuristics have been shown to be accurate for most queries, but they could prune relevant sources needed to provide a complete answer.

In previous work, we define FEDRA [22], a source selection approach that also addresses the problem of querying federations of endpoints with replicated fragments. In FEDRA, fragment descriptions are used to prune sources that provide only redundant data and BGPs are taken into account to delegate join execution to the endpoints, if possible; consequently, FEDRA reduces the number of transferred tuples. FedX and ANAPSID are extended with FEDRA source selection and experimental studies show that in general FEDRA+FedX and FEDRA+ANAPSID transfer less tuples than FedX and ANAPSID, respectively. Because FEDRA performs only source selection, it has to choose among multiple endpoints that access the same fragment before knowing how the query is decomposed into sub-queries. This prevents the query decomposer from producing the most selective sub-queries. Contrarily, LILAC interleaves replication-aware source selection and query decomposition and can be used in combination with existing federated query engines to produce execution plans that are unreachable by the existing engines.

6.5. Query Decomposition in Federations of SPARQL Endpoints

FedX [30] decomposes queries into *exclusive groups*. All the triple patterns with the same unique relevant endpoint are evaluated together in the same sub-queries in order to reduce the *number of calls* that the engines makes during query execution. The triple patterns with more than one relevant endpoint are evaluated as a sub-query with just one triple pattern in *all* the relevant endpoints. Then, obtained results from different endpoints are combined. It is important to notice that FedX may evaluate in the same sub-query triple patterns that do not share any variable, i.e., FedX may generate sub-queries with Cartesian products, if they can be exclusively evaluated by one endpoint. This decision may negatively impact on query execution time, because, pushing down Cartesian products to the endpoints can greatly increase the number of tuples to transfer.

ANAPSID [1] decomposes queries into *star-shaped groups*. After ANAPSID heuristics [23] are used to select the relevant endpoints for each triple pattern, sub-queries are built according to the number of relevant endpoints selected for the triple patterns. Triple patterns with more than one relevant endpoint are evaluated using sub-queries with just one triple pattern in all the selected sources and their results are combined in the federated query engine. For the other triple patterns, they are grouped into *star-shaped groups*, i.e., sets of triple patterns with just one variable in common are evaluated in the same sub-queries. The use of *star-shaped groups* may reduce the size of intermediate results, and the number of tuples transferred from endpoints to the query engine, as suggested by Vidal et al. [35].

Recently, Vidal et al. [34] have proposed a source selection and query decomposition strategy, named Fed-DSATUR. Fed-DSATUR is based on a reduction to the vertex coloring problem and extends an existing approximate solution for the vertex coloring problem named DSATUR to produce query decompositions that maximize the answer completeness and minimize execution time. Fed-DSATUR can find optimal solutions for certain queries, e.g., if the query is reduced to a bipartite graph. The minimization of the number of colors corresponds to minimize the number of sub-queries in the query decomposition, while this minimization may have a positive impact on the execution time, it may also have a *negative impact* on the query completeness.

Similarly to ANAPSID and Fed-DSATUR, LILAC decomposes queries into Cartesian product free sub-queries. Nevertheless, differently from existing approaches, LILAC catalog includes the description of the replicated fragments accessible through the federation endpoints. Knowledge encoded in the fragment descriptions is exploited by LILAC to safely prune relevant sources that share the same relevant data. Moreover, LILAC may include one triple pattern in more than one sub-query in order to increase the sub-queries selectivity if these sub-queries are evaluated by endpoints accessing all the relevant triples in the federation.

7. Conclusions and Future Work

In this paper, we illustrated how replicating fragments allow for data re-organization to better fit federated query needs. We have overcome intrinsic limitations of replication-aware source selection strategies and have enabled federated queries engines to take advantage of replication during query decomposition to produce sub-queries that are more selective and transfer less tuples. Concise description of the replicated data using fragment descriptions paves the way for safely pruning sources with replicated data, and generating sub-queries that push joins down to the endpoints. We have formalized the problem of source selection and query decomposition for federations with replicated fragments (QDP-FR). Moreover, we proposed a replication-aware federated query decomposition algorithm LILAC that approximates QDP-FR and ensures soundness and completeness of query answers. Federated query engines ANAPSID and FedX were extended with LILAC source selection and query decomposition strategies. Experimental results demonstrate that LILAC achieves significant reduction of the number of transferred tuples. Additionally, for queries not "easily" executable by the studied engines, also the execution time was significantly reduced.

This work opens several perspectives. First, we made the hypothesis that replicated fragments are perfectly synchronized and cannot be updated. We can leverage this hypothesis and manage the problem of federated query processing with divergence. For instance in [21], divergence incurred by the endpoints in the federation

is measured and endpoints that exceed a divergence threshold are pruned from the relevant sources.

Several variants of QDP-FR can also be developed. QDP-FR does not distinguish between endpoints, i.e., the cost of accessing endpoints is considered the same. For example, QDP-FR and LILAC can be modified to compute a source selection and query decomposition in order to consider user preferences or other type of data quality values and exploit these meta-data to minimize the number of relevant endpoints. Another perspective is to take further advantage of replicated data. The presence of the same replicated fragment in several endpoints may be exploited by a federated query engine, these endpoints can be used to share the evaluation of sub-queries.

Acknowledgements

Experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). This work is partially supported by the French National Research agency (ANR) through the SocioPlug project (code: ANR-13-INFR-0003). The first author was part of the Unit UMR6241 of the Centre National de la Recherche Scientifique (CNRS), while implementing and evaluating the presented approach.

References

[1] M. Acosta, M. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In Aroyo et al. [5], pages 18–34.

[2] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified Stress Testing of RDF Data Management Systems. In Mika et al. [20], pages 197–212.

[3] G. Aluç, M. T. Ozsu, K. Daudjee, and O. Hartig. chameleon-db: a Workload-Aware Robust RDF Data Management System. *University of Waterloo, Tech. Rep. CS-2013-10*, 2013.

[4] C. B. Aranda, A. Hogan, J. Umbrich, and P. Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In H. Alani et al., editors, *ISWC 2013, Part II*, volume 8219 of *LNCS*, pages 277–293. Springer, 2013.

[5] 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.

[6] 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.

[7] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM*, 13(7):422–426, July 1970.

[9] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-Wise Independent Permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[10] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 145–156. ACM, 2011.

[11] K. M. Endris, S. Faisal, F. Orlandi, S. Auer, and S. Scerri. Interest-based RDF update propagation. In M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, volume 9366 of *Lecture Notes in Computer Science*, pages 513–529. Springer, 2015.

[12] O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In O. Hartig, A. Harth, and J. Sequeda, editors, *COLD*, 2011.

[13] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, and J. Pérez. Foundations of Semantic Web databases. *J. Comput. Syst. Sci.*, 77(3):520–541, 2011.

[14] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[15] J. V. Herwegen, R. Verborgh, E. Mannens, and R. V. de Walle. Query execution optimization for clients of triple pattern fragments. In F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, editors, *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, volume 9088 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2015.

[16] K. Hose and R. Schenkel. Towards benefit-based RDF source selection for SPARQL queries. In R. D. Virgilio, F. Giunchiglia, and L. Tanca, editors, *SWIM*, page 2. ACM, 2012.

[17] L. D. Ibáñez, H. Skaf-Molli, P. Molli, and O. Corby. Col-Graph: Towards Writable and Scalable Linked Open Data. In Mika et al. [20], pages 325–340.

[18] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. In A. V. Aho et al., editors, *ACM Symposium on Theory of Computing*, pages 38–49. ACM, 1973.

[19] D. Kossmann. The state of the art in distributed query processing. *ACM Computer Survey*, 32(4):422–469, 2000.

[20] P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, editors. *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*. Springer, 2014.

[21] 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.

[22] G. Montoya, H. Skaf-Molli, P. Molli, and M.-E. Vidal. Federated SPARQL Queries Processing with Replicated Fragments. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference*, pages 36–51, Bethlehem, United States, Oct. 2015.

[23] G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012.

[24] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*. Springer, 2011.

[25] A. Passant and P. N. Mendes. sparqlpush: Proactive notification of data updates in rdf stores using pubsubhubbub. In *SFSW*, 2010.

[26] B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In S. Bechhofer et al., editors, *ESWC 2008*, volume 5021 of *LNCS*, pages 524–538. Springer, 2008.

[27] N. A. Rakhmawati, J. Umbrich, M. Karnstedt, A. Hasnain, and M. Hausenblas. A comparison of federation over SPARQL endpoints frameworks. In P. Klinov and D. Mouromtsev, editors, *Knowledge Engineering and the Semantic Web - 4th International Conference, KESW 2013, St. Petersburg, Russia, October 7-9, 2013. Proceedings*, volume 394 of *Communications*

in Computer and Information Science, pages 132–146. Springer, 2013.
[28] 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 et al., editors, *ISWC 2013, Part I*, volume 8218 of *LNCS*, pages 574–590. Springer, 2013.
[29] M. Saleem and A. N. Ngomo. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In V. Presutti et al., editors, *ESWC 2014*, volume 8465 of *LNCS*, pages 176–191. Springer, 2014.
[30] 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. [5], pages 601–616.
[31] J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over linked data. *World Wide Web*, 14(5-6):495–544, 2011.
[32] P.-Y. Vandenbussche, J. Umbrich, A. Hogan, and C. Buil-Aranda. SPARQLES: Monitoring Public SPARQL Endpoints. *Semantic Web*, 2016. To appear.
[33] R. Verborgh, O. Hartig, B. D. Meester, G. Haesendonck, L. D. Vocht, M. V. Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. V. de Walle. Querying Datasets on the Web with High Availability. In Mika et al. [20], pages 180–196.
[34] M.-E. Vidal, S. Castillo, M. Acosta, G. Montoya, and G. Palma. On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 2015.
[35] M.-E. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres. Efficiently joining group patterns in sparql queries. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2010.
[36] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992.

Appendix A. Computation of complexity given in Proposition 1

Let n be the number of triple patterns in the query, m be the number of fragments, and k be the number of endpoints. The time complexity of Algorithm 1, $T(n, m, k)$, can be computed as:

$$T(n, m, k) = n.U(m) + V(n, m, k) + W(n, k) + X(n, m, k)$$

with $U(m)$ the complexity of finding the non redundant relevant fragments of a triple pattern, $V(n, m, k)$ the complexity of `ReduceUnions`, $W(n, k)$ the complexity of `ReduceBGPs`, and $X(n, m, k)$ the complexity of `IncreaseSelectivity`. $U(m) = \mathcal{O}(m^2)$, as each fragment should be considered once and it should be checked if there is a containment with the other fragments. $V(n, m, k) = \mathcal{O}(n.m.k)$, as potentially for each triple pattern, all the fragments should be considered for each endpoint. $W(n, k) = \mathcal{O}(n^3.k)$, as potentially for each endpoint, all the pairs of sub-queries should be considered to find the two that share a variable, in the worst case it should be considered that all triple patterns can be evaluated in the endpoints, and in each iteration the number of sub-queries is reduced by one. $X(n, m, k) = \mathcal{O}(n^2.m.k)$, as each triple pattern may be associated to as many endpoint subsets as the number of fragments and each fragment may be accessed by all the endpoints. Also, each endpoint may be associated with as many subsets as the number of triple patterns in *ljtps*. Then $T(n, m, k)$ can be written as:

$$T(n, m, k) = max(\mathcal{O}(n.m^2), \mathcal{O}(n^3.k), \mathcal{O}(n^2.m.k))$$

And it can be more concisely written as:

$$T(n, m, k) = \mathcal{O}(n^3.m^2.k)$$

Notice that execution of runtime queries to confirm fragment relevance have been omitted from the complexity computation. If they are considered, it could increase the complexity to be linear in the number of triples stored by the endpoints if no efficient indexing structures are provided by the data store.

Appendix B. Proof of Theorem 1

Theorem 1 states that Algorithm 1 produces a query decomposition that satisfies the properties 1-3 (Section 3.2).

Property 1. Proof by contradiction, suppose Algorithm 1 produces an answer that is not sound, then at least one triple pattern should have been assigned to an endpoint that does not access any of the triple pattern relevant fragments or a join between two triple patterns in Q, has not been included in the output of Algorithm 1. All triple patterns have been included in sub-queries either in line 54 or in line 63. Moreover, these sub-queries assign triple patterns to endpoints that access relevant fragments for the triple patterns, which have been obtained by functions `SelectNonRedundantFragments` and *endpoints()*. Furthermore, Algorithm 1 preserves the existing joins in Q, i.e., all the triple patterns in Q have been included in sub-queries either in line 54 or in line 63, and shared variables among triple pattern remain unchanged in the output of Algorithm 1. Therefore, it cannot be the case that a triple pattern has been assigned to an endpoint with no relevant fragment, or that a join has been removed from the query, and Algorithm 1 should produce only sound answers.

Property 2. Proof by contradiction, suppose Algorithm 1 fails to produce a valid answer to query Q. In this case, data for at least one triple pattern is missing, or there are additional joins imposed by the query decomposition. For one triple pattern to be missing some data, one fragment that provided no redundant data should have been pruned in line 2, but in that line only redundant fragments were pruned, then this cannot be the case. Only in line 61 additional joins have been included, but only triple patterns that can be completely provided by the endpoint belong to *ljtps*, hence tuples filtered by these joins would be anyway pruned before returning the query answer. In consequence, no tuple that should be present in the query answer is actually removed by Algorithm 1, ensuring its completeness.

Property 3. To satisfy this property any sub-query should be Cartesian product free, as large as possible, and no redundant. Proof by contradiction, suppose Algorithm 1 output includes a sub-query that has a Cartesian product or is not as large as possible or that is redundant, *sq*. Proof by cases, case a) *sq* includes a Cartesian product. However, sub-queries are only built in lines 44 and 60, and these sub-queries are by construction Cartesian product free. Case b) *sq* is not as large as possible. However, sub-queries are only built in lines 44 and 60, and these sub-queries are by construction as large as possible. Case c) *sq* is redundant. Sub-queries are only built in lines 44 and 60. However, sub-queries built in line 44 are pruned in line 48 using a set covering solution, i.e., triple patterns are covered using as few sub-queries and no redundant sub-queries can remain in the output of function `SetCoveringSubqueries`. Moreover, sub-queries built in line 60 access different relevant fragments for triple pattern tp, with non known containment relationships among them, therefore if all the replicated fragments have been described none of these sub-queries is redundant either. Hence, Algorithm 1 produces an output with Cartesian product free, as large as possible, and no redundant sub-queries, and *minimizes* the data transfer.

Appendix C. P-values of the Wilcoxon signed rank tests performed

Table C.5: Wilcoxon signed rank test p-values for testing if LILAC and the engines select the same number of sources or if LILAC selects less sources. **Bold** p-values allow to accept that LILAC selects less sources than the engines

Federation	p-value	
	ANAPSID	FedX
Diseasome	< **2.2e-16**	< **2.2e-16**
SWDF	< **2.2e-16**	< **2.2e-16**
LinkedMDB	**2.626e-16**	< **2.2e-16**
Geocoordinates	< **2.2e-16**	< **2.2e-16**
WatDiv1	**5.258e-16**	< **2.2e-16**
WatDiv100	< **2.2e-16**	**1.267e-15**

Table C.6: Wilcoxon signed rank test p-values for testing if LILAC and DAW select the same number of sources or if LILAC selects less sources. **Bold** p-values allow to accept that LILAC selects less sources than DAW

Federation	p-value	
	ANAPSID	FedX
Diseasome	< **2.2e-16**	**2.292e-07**
SWDF	< **2.2e-16**	**3.357e-12**
LinkedMDB	< **2.2e-16**	**2.792e-11**
Geocoordinates	< **2.2e-16**	**1.301e-05**
WatDiv1	**3.782e-13**	**2.361e-05**
WatDiv100	**4.402e-14**	**0.0009134**

Table C.7: Wilcoxon signed rank test p-values for testing if LILAC and DAW/FedX DST is the same or if LILAC source selection is faster. **Bold** p-values allow to accept that LILAC DST is faster than DAW/FedX

Federation	p-value	
	DAW+FedX	FedX
Diseasome	< **2.2e-16**	**7.728e-13**
SWDF	< **2.2e-16**	< **2.2e-16**
LinkedMDB	< **2.2e-16**	**1.118e-14**
Geocoordinates	< **2.2e-16**	**1.326e-11**
WatDiv1	< **2.2e-16**	**3.76e-16**
WatDiv100	< **2.2e-16**	**4.235e-14**

Table C.8: Wilcoxon signed rank test p-values for testing if LILAC in combination with the engines and the engines alone have the same execution time or if LILAC reduces the engines execution time. **Bold** p-values allow to accept that LILAC reduces the engines execution time

Federation	p-value	
	ANAPSID	FedX
Diseasome	**6.421e-13**	< **2.2e-16**
SWDF	**0.0069**	< **2.2e-16**
LinkedMDB	**1.37e-06**	**6.442e-10**
Geocoordinates	1	1
WatDiv1	0.9995	< **2.2e-16**
WatDiv100	**0.002381**	**9.497e-06**

Table C.9: Wilcoxon signed rank test p-values for testing if LILAC in combination with the engines and the engines alone have the same execution time or if LILAC reduces the engines execution time. **Bold** p-values allow to accept that the engines enhanced with LILAC achieve a greater reduction in execution time than the engines enhanced with DAW

Federation	p-value	
	ANAPSID	FedX
Diseasome	< **2.2e-16**	< **2.2e-16**
SWDF	< **2.2e-16**	< **2.2e-16**
LinkedMDB	< **2.2e-16**	**3.714e-14**
Geocoordinates	< **2.2e-16**	**6.653e-13**
WatDiv1	< **2.2e-16**	**2.368e-16**
WatDiv100	**1.267e-11**	**9.179e-16**

Table C.10: Wilcoxon signed rank test p-values for testing if LILAC with the engines and the engines alone transfer the same number of tuples or if LILAC reduces the number of transferred tuples. **Bold** p-values allow to accept that LILAC reduces the number of transferred tuples by the engines

Federation	p-value	
	ANAPSID	FedX
Diseasome	**< 2.2e-16**	**< 2.2e-16**
SWDF	**4.688e-14**	**< 2.2e-16**
LinkedMDB	**< 2.2e-16**	**< 2.2e-16**
Geocoordinates	**< 2.2e-16**	**< 2.2e-16**
WatDiv1	**3.029e-16**	**< 2.2e-16**
WatDiv100	**7.689e-16**	**2.432e-15**

Table C.11: Wilcoxon signed rank test p-values for testing if LILAC with the engines and the engines alone transfer the same number of tuples or if LILAC reduction in the number of transferred tuples is greater. **Bold** p-values allow to accept that LILAC reduction in the number of transferred tuples is greater

Federation	p-value	
	ANAPSID	FedX
Diseasome	**2.905e-16**	**1.017e-06**
SWDF	**6.147e-09**	**2.365e-06**
LinkedMDB	**4.161e-15**	**3.421e-05**
Geocoordinates	**1.29e-12**	0.5
WatDiv1	**3.301e-07**	**3.074e-09**
WatDiv100	**1.205e-07**	**8.762e-08**