HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation

Muhammad Saleem and Axel-Cyrille Ngonga Ngomo

Universität Leipzig, IFI/AKSW, P.O. 100920, 04009 Leipzig, Germany
{lastname}informatik.uni-leipzig.de

Abstract. Efficient federated query processing is of significant importance to tame the large amount of data available on the Web of Data. Previous works have focused on generating optimized query execution plans for fast result retrieval. However, devising source selection approaches beyond triple pattern-wise source selection has not received much attention. This work presents HiBISCuS, a novel hypergraph-based source selection approach to federated SPARQL querying. Our approach can be directly combined with existing SPARQL query federation engines to achieve the same recall while querying fewer data sources. We extend three well-known SPARQL query federation engines with HiBISCus and compare our extensions with the original approaches on FedBench. Our evaluation shows that HiBISCuS can efficiently reduce the total number of sources selected without losing recall. Moreover, our approach significantly reduces the execution time of the selected engines on most of the benchmark queries.

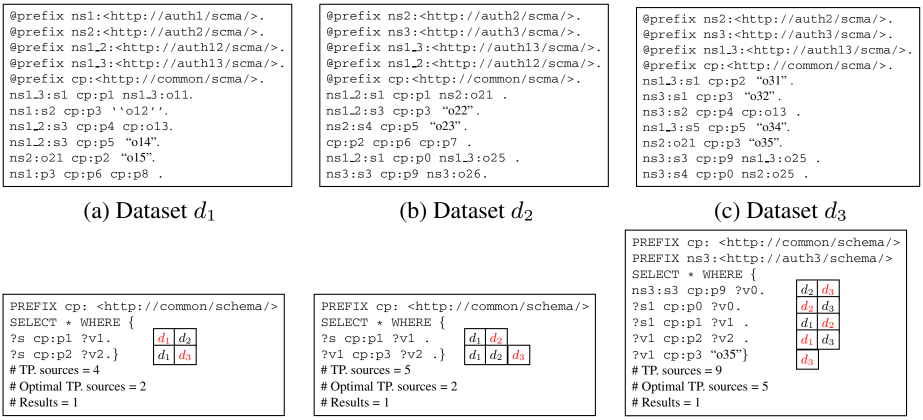
Keywords: #eswc2014Saleem.

1 Introduction

The Web of Data is now a large compendium of interlinked data sets from multiple domains with large datasets [12] being added frequently [3]. Given the complexity of information needs on the Web, certain queries can only be answered by retrieving results contained across different data sources (short: sources). Thus, the optimization of engines that support this type of queries, called *federated query engines*, is of central importance to ensure the usability of the Web of Data in real-world applications. One of the important optimization steps in federated SPARQL query processing is the efficient selection of relevant sources for a query. To ensure that a recall of 100% is achieved, most SPARQL query federation approaches [4,8,10,14,15,11] perform *triple pattern-wise source selection* (TPWSS). The goal of the TPWSS is to identify the set of relevant (also called capable, formally defined in section 3) sources against individual triple patterns of a query [11]. However, it is possible that a relevant source does not *contribute* to the final result set of the complete query. This is because the results from a particular data source can be excluded after performing *joins* with the results of other triple patterns contained in the same query. An overestimation of such sources increases the network traffic and can significantly affect the overall query processing time.

An example for such a query is SSQ1 in Figure 1. A TPWSS that retrieves all relevant sources for each individual triple pattern would lead to all sources in the example being queried. Yet, the complete result set of SSQ1 can be computed without querying

V. Presutti et al. (Eds.): ESWC 2014, LNCS 8465, pp. 176–191, 2014.



```
@prefix ns1:<http://auth1/scma/>.
@prefix ns2:<http://auth2/scma/>.
@prefix ns1_2:<http://auth12/scma/>.
@prefix ns1_3:<http://auth13/scma/>.
@prefix cp:<http://common/scma/>.
ns1_3:s1 cp:p1 ns1_3:o11.
ns1:s2 cp:p3 ``o12''.
ns1_2:s3 cp:p4 cp:o13.
ns1_2:s3 cp:p5 "o14".
ns2:o21 cp:p2 "o15".
ns1:p3 cp:p6 cp:p8 .
```

(a) Dataset d_1

```
@prefix ns2:<http://auth2/scma/>.
@prefix ns3:<http://auth3/scma/>.
@prefix ns1_3:<http://auth13/scma/>.
@prefix ns1_2:<http://auth12/scma/>.
@prefix cp:<http://common/scma/>.
ns1_2:s1 cp:p1 ns2:o21 .
ns1_2:s3 cp:p3 "o22".
ns2:s4 cp:p5 "o23".
cp:p2 cp:p6 cp:p7 .
ns1_2:s1 cp:p0 ns1_3:o25 .
ns3:s3 cp:p9 ns3:o26.
```

(b) Dataset d_2

```
@prefix ns2:<http://auth2/scma/>.
@prefix ns3:<http://auth3/scma/>.
@prefix ns1_3:<http://auth13/scma/>.
@prefix cp:<http://common/scma/>.
ns1_3:s1 cp:p2 "o31".
ns3:s1 cp:p3 "o32".
ns3:s2 cp:p4 cp:o13 .
ns1_3:s5 cp:p5 "o34".
ns2:o21 cp:p3 "o35".
ns3:s3 cp:p9 ns1_3:o25 .
ns3:s4 cp:p0 ns2:o25 .
```

(c) Dataset d_3

```
PREFIX cp: <http://common/schema/>
SELECT * WHERE {
?s cp:p1 ?v1.          d1 d2
?s cp:p2 ?v2.}         d1 d3
# TP. sources = 4
# Optimal TP. sources = 2
# Results = 1
```

(d) Star-shaped query SSQ1

```
PREFIX cp: <http://common/schema/>
SELECT * WHERE {
?s cp:p1 ?v1 .         d1 d2
?v1 cp:p3 ?v2 .}       d1 d2 d3
# TP. sources = 5
# Optimal TP. sources = 2
# Results = 1
```

(e) Path-shaped query PSQ2

```
PREFIX cp: <http://common/schema/>
PREFIX ns3:<http://auth3/schema/>
SELECT * WHERE {
ns3:s3 cp:p9 ?v0.      d2 d3
?s1 cp:p0 ?v0.         d2 d3
?s1 cp:p1 ?v1 .        d1 d2
?v1 cp:p2 ?v2 .        d1 d3
?v1 cp:p3 "o35"}       d3
# TP. sources = 9
# Optimal TP. sources = 5
# Results = 1
```

(f) Hybrid query HSQ3

Fig. 1. Motivating Examples. #TP is the total number of triple pattern-wise sources selected. The relevant sources for each are shown next to each triple pattern. The sources marked in red contribute to the final query result set.

d_2 due to the type of join used in the query. We thus propose a *novel join-aware approach to TPWSS* dubbed HiBISCuS. Our approach goes beyond the state of the art by aiming to compute the sources that actually contribute to the final result set of an input query and that for each triple pattern. To the best of our knowledge, this join-aware approach to TPWSS has only been tackled by an extension of the ANAPSID framework presented in [9]. Yet, this extension is based on evaluating namespaces and sending ASK queries to data sources at runtime. In contrast, HiBISCuS relies on an index that stores the authorities of the resource URIs[1] contained in the data sources at hand. Our approach proves to be more time-efficient than the ANAPSID extension as shown by our evaluation in Section 5.

HiBISCuS addresses the problem of source selection by several innovations. Our first innovation consists of modelling SPARQL queries as a sets of *directed labelled hypergraphs* (DLH). Moreover, we rely on a *novel type of summaries* which exploits the fact that the resources in SPARQL endpoints are Uniform Resource Identifiers (URIs). Our *source selection algorithm is also novel* and consists of two steps. In a first step, our approach *labels the hyperedges* of the DLH representation of an input SPARQL query q with relevant data sources. In the second step, the summaries and the type of joins in q are used to *prune the edge labels*. By these means, HiBISCuS can discard sources (without losing recall) that are not pertinent to the computation of the final result set of the query. Overall, our contributions are thus as follows:

1. We present a formal framework for modelling SPARQL queries as directed labelled hypergraphs.
2. We present a novel type of data summaries for SPARQL endpoints which relies on the authority fragment of URIs.

[1] http://tools.ietf.org/html/rfc3986

3. We devise a pruning algorithm for edge labels that enables us to discard irrelevant sources based on the types of joins used in a query.
4. We evaluate our approach by extending three state-of-the-art federate query engines (FedX, SPLENDID and DARQ) with HiBISCuS and comparing these extensions to the original systems. In addition, we compare our most time-efficient extension with the extension of ANAPSID presented in [9]. Our results show that we can reduce the number of source selected, the source selection time as well as the overall query runtime of each of these systems.

The structure of the rest of this paper is as follows: we first give a brief overview of federated query engines. Then, we present our formalization of SPARQL queries as directed labelled hypergraphs. The algorithms underlying HiBISCuS are then explained in detail. Finally, we evaluate HiBISCuS against the state-of-the-art and show that we achieve both better source selection and runtimes on the FedBench [13] SPARQL query federation benchmark.

2 Related Work

The approaches related to query federation over the Web of Data can be divided into three main categories (see Table 1 obtained from our public survey results[2]).

(1) *Query federation approaches over multiple SPARQL endpoints* make use of the SPARQL endpoints due to which they provide a time-efficient solution to SPARQL query federation. However, the RDF data needs to be exposed as SPARQL endpoints. Due to which they are unable to deal with whole LOD. (2) *Query federation over Linked Data* do not require the data to be exposed via SPARQL endpoints. The only requirement is that it should follow the Linked Data principles.[3] Due to URI lookups at runtime, these type of approaches are commonly slower than the previous type of approaches. (3) *Query federation approaches on top of Distributed Hash Tables* store the RDF data on top of Distributed Hash Tables (DHTs). This is a space-efficient solution and can reduce the network cost as well. However, an important fraction of the LOD datasets is not stored using DHTs.

The source selection approaches used in each of the categories can further divided into three sub-categories(see Table 1). (1)*Catalog/index-assisted source selection* only makes use of an index/data catalog (also called data summaries) to perform TPWSS. The result completeness (100% recall) must be ensured by keeping the index up-to-date. (2)*Catalog/index-free source selection* approaches do not make use of any pre-stored index and can thus always compute complete and up-to-date records. However, they commonly have a longer query execution time due to the extra processing required for collecting on-the-fly statistics (e.g. SPARQL ASK operations). (3) *Hybrid source selection* approaches are a combination of the previous approaches.

In this paper, we propose a novel hybrid source selection approach for SPARQL endpoint federation systems dubbed HiBISCuS. In contrast to the state of the art, HiBISCuS uses hypergraphs to detect sources that will not generate any relevant results

[2] Survey: `http://goo.gl/iXvKVT`, Results: `http://goo.gl/CNW5UC`

[3] `http://www.w3.org/DesignIssues/LinkedData.html`

Table 1. Classification of SPARQL federation engines. (**SEF** = SPARQL Endpoints Federation, **DHT** = DHT Federation, **LDF** = Linked Data Federation, **Ctg.** = Federation Type, **FdX** = FedX, **SPL** =SPLENDID, **ADE** = ADERIS, **I.F** = Index-free, **I.O** = Index-only, **HB** = Hybrid, **C.A.** = Code Availability, **S.S.T.** = Source Selection Type, **I.U.** = Index Update, **NA** = Not Applicable, **(A+I)** = SPARQL ASK and Index, **(C+L)** = Catalog and online discovery via Link-traversal, *only source selection approaches.).

	FedX	LHD	SPL	DAW*	ANAPSID	ADE	DARQ	LDQP	WoDQA	Atlas	QTree*	HiBISCus*
	[14]	[15]	[4]	[11]	[1]	[8]	[10]	[7]	[2]	[6]	[5]	
Ctg.	SEF	SEF	SEF	-	SEF	SEF	SEF	LDF	LDF	DHT	-	-
C.A	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✗	✓
S.S.T	I.F	HB(A+I)	HB(A+I)	HB(A+I)	HB(A+I)	I.O	I.O	HB(C+L)	HB(A+I)	I.O	I.O	HB(A+I)
Cache	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✓
I.U	NA	✗	✗	✗	✓	✗	✗	✗	✓	✗	✓	✓

both at triple-pattern level and at query level. By these means, HiBISCuS can generate better approximations of the sources that should be queried to return complete results for a given query.

3 Preliminaries

In the following, we present some of the concepts and notation that are used throughout this paper. RDF resources are identified by using a Unified Resource Identifier (URI). Each URI has a generic syntax consists of a hierarchical sequence of components namely the *scheme*, *authority*, *path*, *query*, and*fragment*[4]. For example, the prefix $ns1 = <http : //auth1/scma/>$ used in Figure 1 consist of scheme $http$, authority $auth1$, and path $scma$. The details of the remaining two components are out of the scope of this paper. In the rest of the paper, we jointly refer to the first two components (path, authority) as *authority* of a URI.

The standard for querying RDF is SPARQL.[5] The result of a SPARQL query is called its *result set*. Each element of the result set of a query is a set of *variable bindings*. *Federated SPARQL queries* are defined as queries that are carried out over a set of sources D = $\{d_1, \ldots, d_n\}$. Given a SPARQL query q, a source $d \in D$ is said to *contribute* to q if at least one of the variable bindings belonging to an element of q's result set can be found in d.

Definition 1 (Relevant source Set). *A source $d \in D$ is* relevant *(also called* capable*) for a triple pattern $tp_i \in TP$ if at least one triple contained in d matches tp_i.*[6] *The relevant source set $R_i \subseteq D$ for tp_i is the set that contains all sources that are relevant for that particular triple pattern.*

For example, the set of relevant sources for the triple pattern $<?s, \texttt{cp:p1}, ?v1>$ of SSQ1 is $\{d_1, d_2\}$. It is possible that a relevant source for a triple pattern does not *con-*

[4] URI syntax: `http://tools.ietf.org/html/rfc3986`

[5] `http://www.w3.org/TR/rdf-sparql-query/`

[6] The concept of matching a triple pattern is defined formally in the SPARQL specification found at `http://www.w3.org/TR/rdf-sparql-query/`

tribute to the final result set of the complete query q. This is because the results computed from a particular source d for a triple pattern tp_i might excluded while performing *joins* with the results of other triple patterns contained in the query q. For example, consider SSQ1. The results from d_2 for $<?s, \texttt{cp:p1}, ?v1>$ and d_3 for $<?s, \texttt{cp:p2}, ?v1>$ are excluded after performing the *join* between the results of the two triple patterns.

Definition 2 (Optimal source Set). *The optimal source set $O_i \subseteq R_i$ for a triple pattern $tp_i \in TP$ contains the relevant sources $d \in R_i$ that actually contribute to computing the complete result set of the query.*

For example, the set of optimal sources for the triple pattern $<?s, \texttt{cp:p2}, ?v2>$ of SSQ1 is $\{d_3\}$, while the set of relevant sources for the same triple pattern is $\{d_1, d_3\}$. Formally, the problem of TPWSS can then be defined as follows:

Definition 3 (Problem Statement). *Given a set D of sources and a query q, find the optimal set of sources $O_i \subseteq D$ for each triple pattern tp_i of q.*

Most of the source selection approaches [4,8,10,14,15] used in SPARQL endpoint federation systems only perform TPWSS, i.e., they find the set of relevant sources R_i for individual triple patterns of a query and do not consider computing the optimal source sets O_i. In this paper, we present an index-assisted approach for (1) the time-efficient computation of relevant source set R_i for individual triple patterns of the query and (2) the approximation of O_i out of R_i. HiBISCuS approximates O_i by determining and removing irrelevant sources from each of the R_i. We denote our approximation of O_i by RS_i. HiBISCuS relies on directed labelled hypergraphs (DLH) to achieve this goal. In the following, we present our formalization of SPARQL queries as DLH. Subsequently, we show how we make use of this formalization to approximate O_i for each tp_i.

4 HiBISCuS

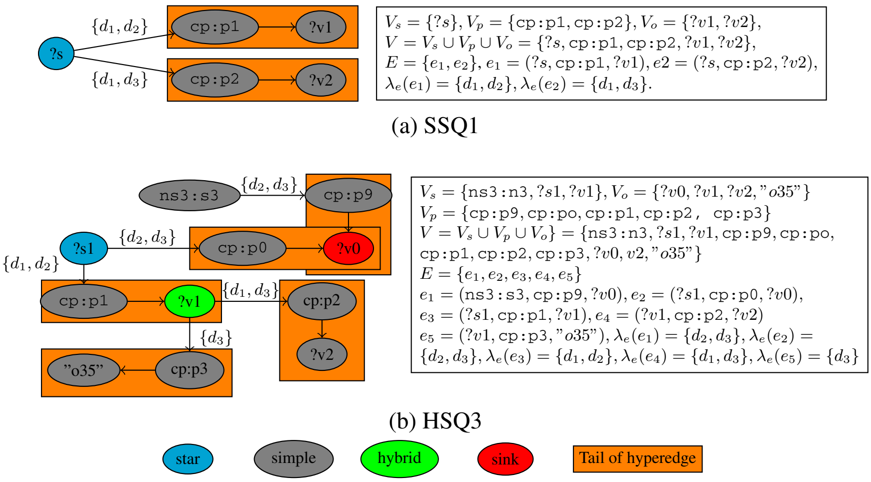
In this section we present our approach to the source selection problem in details. We begin by presenting our approach to representing BGPs[7] of a SPARQL query as DLHs. Then, we present our approach to computing *lightweight data summaries*. Finally, we explain our approach to source selection.

4.1 Queries as Directed Labelled Hypergraphs

An important intuition behind our approach is that each of the BGP in a query can be executed separately. Thus, in the following, we will mainly focus on how the execution of a single BGP can be optimized. The representation of a query as DLH is the union of the representations of its BGPs. Note that the representations of BGPs are kept disjoint even if they contain the same nodes to ensure that the BGPs are processed independently. The DLH representation of a BGP is formally defined as follows:

Definition 4. *Each basic graph patterns BGP_i of a SPARQL query can be represented as a DLH $HG_i = (V, E, \lambda_e, \lambda_{vt})$, where*

[7] http://www.w3.org/TR/sparql11-query/#BasicGraphPatterns



$V_s = \{?s\}, V_p = \{\texttt{cp:p1}, \texttt{cp:p2}\}, V_o = \{?v1, ?v2\}$,
$V = V_s \cup V_p \cup V_o = \{?s, \texttt{cp:p1}, \texttt{cp:p2}, ?v1, ?v2\}$,
$E = \{e_1, e_2\}, e_1 = (?s, \texttt{cp:p1}, ?v1), e2 = (?s, \texttt{cp:p2}, ?v2)$,
$\lambda_e(e_1) = \{d_1, d_2\}, \lambda_e(e_2) = \{d_1, d_3\}$.

(a) SSQ1

$V_s = \{\texttt{ns3:n3}, ?s1, ?v1\}, V_o = \{?v0, ?v1, ?v2, "o35"\}$
$V_p = \{\texttt{cp:p9}, \texttt{cp:po}, \texttt{cp:p1}, \texttt{cp:p2}, \ \texttt{cp:p3}\}$
$V = V_s \cup V_p \cup V_o\} = \{\texttt{ns3:n3}, ?s1, ?v1, \texttt{cp:p9}, \texttt{cp:po}$,
$\texttt{cp:p1}, \texttt{cp:p2}, \texttt{cp:p3}, ?v0, v2, "o35"\}$
$E = \{e_1, e_2, e_3, e_4, e_5\}$
$e_1 = (\texttt{ns3:s3}, \texttt{cp:p9}, ?v0), e_2 = (?s1, \texttt{cp:p0}, ?v0)$,
$e_3 = (?s1, \texttt{cp:p1}, ?v1), e_4 = (?v1, \texttt{cp:p2}, ?v2)$
$e_5 = (?v1, \texttt{cp:p3}, "o35"), \lambda_e(e_1) = \{d_2, d_3\}, \lambda_e(e_2) =$
$\{d_2, d_3\}, \lambda_e(e_3) = \{d_1, d_2\}, \lambda_e(e_4) = \{d_1, d_3\}, \lambda_e(e_5) = \{d_3\}$

(b) HSQ3

Fig. 2. Labelled hypergraph of query SSQ1 and query HSQ3 of Figure 1

1. *$V = V_s \cup V_p \cup V_o$ is the set of vertices of HG_i, V_s is the set of all subjects in HG_i, V_p the set of all predicates in HG_i and V_o the set of all objects in HG_i;*
2. *E =$\{e_1, \ldots, e_t\} \subseteq V^3$ is a set of directed hyperedges (short: edge). Each edge e= (v_s,v_p,v_o) emanates from the triple pattern <v_s, v_p, v_o> in BGP_i. We represent these edges by connecting the head vertex v_s with the tail hypervertex (v_p, v_o). In addition, we use $E_{in}(v) \subseteq E$ and $E_{out}(v) \subseteq E$ to denote the set of incoming and outgoing edges of a vertex v;*
3. *$\lambda_e : E \mapsto 2^D$ is a hyperedge-labelling function. Given a hyperedge $e \in E$, its edge label is a set of sources $R_i \subseteq D$. We use this label to the sources that should be queried to retrieve the answer set for the triple pattern represented by the hyperedge e;*
4. *λ_{vt} is a vertex-type-assignment function. Given an vertex $v \in V$, its vertex type can be 'star', 'path', 'hybrid', or 'sink' if this vertex participates in at least one join. A 'star' vertex has more than one outgoing edge and no incoming edge. 'path' vertex has exactly one incoming and one outgoing edge. A 'hybrid' vertex has either more than one incoming and at least one outgoing edge or more than one outgoing and at least one incoming edge. A 'sink' vertex has more than one incoming edge and no outgoing edge. A vertex that does not participate in any join is of type 'simple'.*

Figure 2a shows the hypergraph of SSQ1 and Figure 2b represents the hypergraph of HSQ3 of motivating example given in Figure 1. We can now reformulate our problem statement as follows:

Definition 5 (Problem Reformulation). *Given a query q represented as a set of hypergraphs $\{HG_1, \ldots, HG_x\}$, find the labelling of the hyperedges of each hypergraph HG_i that leads to an optimal source selection.*

Listing 1.1. HiBISCuS example. Prefixes are ignored for simplicity

```
[] a ds:Service ;
   ds:endpointUrl <http://dbpedia.org/sparql> ;
  ds:capability
    [ ds:predicate  dbpedia:kingdom ;
      ds:sbjAuthority  <http://dbpedia.org/> ;
      ds:objAuthority  <http://dbpedia.org/> ;
      ] ;
  ds:capability
    [ ds:predicate  rdf:type ;
      ds:sbjAuthority  <http://dbpedia.org/> ;
      ds:objAuthority  owl:Thing, dbpedia:Station; #we store all distinct classes
      ] ;
  ds:capability
    [ds:predicate  dbpedia:postalCode ;
     ds:sbjAuthority  <http://dbpedia.org/> ;
     #No objAuthority as the object value for dbpedia:postalCode is string
     ] ;
```

4.2 Data Summaries

HiBISCuS relies on *capabilities* to compute data summaries. Given a source d, we define a capability as a triple $(p, SA(d,p), OA(d,p))$ which contains (1) a predicate p in d, (2) the set $SA(d,p)$ of all distinct *subject authorities* (ref. section 3) of p in d and (3) the set $OA(d,p)$ of all distinct *object authorities* of p in d. In HiBISCuS, a *data summary* for a source $d \in D$ is the set $CA(d)$ of all *capabilities* of that source. Consequently, the total number of capabilities of a source is equal to the number of distinct predicates in it.

The predicate `rdf:type` is given a special treatment: Instead of storing the set of all distinct object authorities for a capability having this predicate, we store the *set of all distinct class URIs* in d, i.e., the set of all resources that match $?x$ in the query $?y$ `rdf:type` $?x$. The reason behind this choice is that the set of distinct *classes* used in a source d is usually a small fraction of the set of all resources in d. Moreover, triple patterns with predicate `rdf:type` are commonly used in SPARQL queries. Thus, by storing the complete class URI instead of the object authorities, we might perform more accurate source selection. Listing 1.1 shows an example of a data summary. In the next section, we will make use of these data summaries to optimize the *TPWSS*.

4.3 Source Selection Algorithm

Our source selection comprise two steps: given a query q, we first *label all hyperedges* in each of the hypergraphs which results from the BGPs of q, i.e., we compute $\lambda_e(e_i)$ for each $e_i \in E_i$ in all $HG_i \in DHG$. We present two variations of this step and compare them in the evaluation section. In a second step, we *prune the labels of the hyperedges* assigned in the first step and compute $RS_i \subseteq R_i$ for each e_i. The pseudo-code of our approaches is shown in Algorithms 1, 2 (labelling) as well as 3 (pruning).

Labelling Approaches. We devised two versions of our approach to the hyperedge labelling problem, i.e., an ASK-dominant and an index-dominant version. Both take the set of all sources D, the set of all disjunctive hypergraphs DHG of the input query q and the data summaries $HiBISCuS_D$ of all sources in D as input (see Algorithms 1,2).

Algorithm 1. ASK-dominant hybrid algorithm for labelling all hyperedges of each disjunctive hypergraph of a SPARQL query

```
Require: $D = \{d_1, \ldots, d_n\}$; $DHG = \{HG_1, \ldots, HG_x\}$; $HiBISCuS_D$ //sources, disjunctive
    hypergraphs of a query, HiBISCuSmaries of sources
 1: for each $HG_i \in DHG$ do
 2:   E = hyperedges ($HG_i$)
 3:   for each $e_i \in E$ do
 4:     $s$ = subjvertex($e_i$); $p$ = predvertex($e_i$); $o$ = objvertex($e_i$);
 5:     $sa$ = subjauth($s$); $oa$ = objauth($o$); //can be null i.e. for unbound s, o
 6:     if !bound($s$) $\wedge$ !bound($p$) $\wedge$ !bound($o$) then
 7:       $\lambda_e(e_i) = D$
 8:     else if bound($p$) then
 9:       if $p = rdf : type$ $\wedge$ bound($o$) then
10:         $\lambda_e(e_i) = HiBISCuS_D lookup(p, o)$
11:       else if !commonpredicate($p$) $\vee$ (!bound($s$) $\wedge$ !bound($o$)) then
12:         $\lambda_e(e_i) = HiBISCuS_D lookup(sa, p, oa)$
13:       else
14:         if cachehit($s$, $p$, $o$) then
15:           $\lambda_e(e_i) = cachelookup(s, p, o)$
16:         else
17:           $\lambda_e(e_i)$ = ASK($s$, $p$, $o$, $D$)
18:         end if
19:       end if
20:     else
21:       Repeat Lines 14-18
22:     end if
23:   end for
24: end for
25: return $DHG$ //Set of labelled disjunctive hypergraphs
```

They return a set of *labelled* disjunctive hypergraphs as output. For each hypergraph and each hyperedge, the subject, predicate, object, subject authority, and object authority are collected (Lines 2-5 of Algorithms 1,2). Edges with unbound subject, predicate, and object vertices (e.g e = (?s, ?p, ?o)) are labelled with the set of all possible sources D (Lines 6-7 of Algorithms 1,2). A data summary lookup is performed for edges with the predicate vertex `rdf:type` that have a bound object vertex. All sources with matching capabilities are selected as label of the hyperedge (Lines 9-10 of Algorithms 1,2).

The *ASK-dominant version* of our approach (see Algorithm 1, Line 11) makes use of the notion of *common predicates*. A common predicate is a predicate that is used in a number of sources above a specific threshold value θ specified by the user. A predicate is then considered a common predicate if it occurs in at least $\theta|D|$ sources. We make use of the ASK queries for triple patterns with common predicates. Here, an ASK query is sent to all of the available sources to check whether they contain the common predicate *cp*. Those sources which return `true` are selected as elements of the set of sources used to label that triple pattern. The results of the ASK operations are stored in a cache. Therefore, every time we perform a cache lookup before SPARQL ASK operations (Lines 14-18). In contrast, in the *index-dominant* version of our algorithm, an index lookup is performed if any of the subject or predicate is bound in a triple pattern.We will see later that the index-dominant approach requires less ASK queries than the ASK-dominant algorithm. However, this can lead to an overestimation of the set of relevant sources (see section 5.2).

Algorithm 2. Index-dominant hybrid algorithm for labelling all hyperedges of each disjunctive hypergraph of a SPARQL query

```
Require: $D$= $\{d_1, \ldots, d_n\}$; $DHG = \{HG_1, \ldots, HG_x\}$; $HiBISCuS_D$ //sources, disjunctive
    hypergraphs of a query, HiBISCuSmaries of sources
for each $HG_i \in DHG$ do
    E = hyperedges ($HG_i$)
    for each $e_i \in E$ do
        $s$ = subjvertex($e_i$); $p$ = predvertex($e_i$); $o$ = objvertex($e_i$);
        $sa$ = subjauth($s$); $oa$ = objauth($o$); //can be null i.e. for unbound s, o
        if !bound($s$) $\wedge$ !bound($p$) $\wedge$ !bound($o$) then
            $\lambda_e(e_i)$ = $D$
        else if bound(s) $\vee$ bound(p) then
            if bound(p) $\wedge$ $p = rdf : type$ $\wedge$ bound($o$) then
                $\lambda_e(e_i)$ = $HiBISCuS_D lookup(p, o)$
            else
                $\lambda_e(e_i)$ = $HiBISCuS_D lookup(sa, p, oa)$
            end if
        else
            if cachehit($s, p, o$) then
                $\lambda_e(e_i)$ = $cachelookup(s, p, o)$
            else
                $\lambda_e(e_i)$ = ASK($s, p, o, D$)
            end if
        end if
    end for
end for
return $DHG$ //Set of labelled disjunctive hypergraphs
```

4.4 Pruning Approach

The intuition behind our pruning approach is that knowing which authorities are relevant to answer a query can be used to discard triple pattern-wise (TPW) selected sources that will not contribute to the final result set of the query. Our source pruning algorithm (ref. Algorithm 3) takes the set of all labelled disjunctive hypergraphs as input and prune labels of all hyperedges which either incoming or outgoing edges of a 'star','hybrid', 'path', or 'sink' node. Note that our approach deals with each BGP of the query separately (Line 1 of Algorithm 3).

For each node v of a DLH that is not of type 'simple', we first retrieve the sets (1) $SAuth$ of the subject authorities contained in the elements of the label of each outgoing edge of v (Lines 5-7 of Algorithm 3) and (2) $OAuth$ of the object authorities contained in the elements of the label of each ingoing edge of v (Lines 8-10 of Algorithm 3). Note that these are sets of sets of authorities. For the node $?v1$ of HSQ3 in our running example (see Figure 3), we get $SAuth = \{\texttt{auth13, auth2}\}$ for the ingoing edge and $OAuth = \{\{\texttt{auth13, auth2}\}, \{\texttt{auth2}\}\}$ for the outgoing edges. Now we merge these two sets to the set A of all authorities. For node $?v1$ in HSQ3, $A = \{\{\texttt{auth13, auth2}\}, \{\texttt{auth13, auth2}\}, \{\texttt{auth2}\}\}$. The intersection $I = \left(\bigcap\limits_{a_i \in A} a_i\right)$ of these elements sets is then computed. In our example, this results in $I = \{\texttt{auth2}\}$. Finally, we recompute the label of each hyperedge e that is connected to v. To this end, we compute the subset of the previous label

Algorithm 3. Hyperedge label pruning algorithm for removing irrelevant sources

```
Require: $DHG$ //disjunctive hypergraphs
 1: for each $HG_i \in DHG$ do
 2:    for each v $\in$ vertices($HG_i$) do
 3:       if $\lambda_{vt}(v) \neq$ 'simple' then
 4:          $SAuth = \emptyset$; $OAuth = \emptyset$;
 5:          for each $e \in E_{out}(v)$ do
 6:             $SAuth = SAuth \cup \{$subjectauthories($e$)$\}$
 7:          end for
 8:          for each $e \in E_{in}(v)$ do
 9:             $OAuth = OAuth \cup \{$objectauthories($e$)$\}$
10:          end for
11:          $A = SAuth \cup OAuth$ // set of all authorities
12:          $I$ = $A$.get(1) //get first element of authorities
13:          for each $a \in A$ do
14:             $I$ = $I \cap a$ //intersection of all elements of $A$
15:          end for
16:          for each $e \in E_{in}(v) \cup E_{out}(v)$ do
17:             $label = \emptyset$ //variable for final label of $e$
18:             for $d_i \in \lambda_e(e)$ do
19:                if authorities($d_i$) $\cap I \neq \emptyset$ then
20:                   $label = label \cup d_i$
21:                end if
22:             end for
23:             $\lambda_e(e) = label$
24:          end for
25:       end if
26:    end for
27: end for
```

of e which is such that the set of authorities of each of its elements is not disjoint with I (see Lines 16-23 of Algorithm 3). These are the only sources that will really contribute to the final result set of the query.

We are sure not to lose any recall by this operation because joins act in a conjunctive manner. Consequently, if the results of a data source d_i used to label a hyperedge cannot be joined to the results of at least one source of each of the other hyperedges, it is guaranteed that d_i will not contribute to the final result set of the query. In our example, this leads to d_1 being discarded from the label of the ingoing edge, while d_3 is discarded from the label of one outgoing hyperedge of node $?v1$ as shown in Figure 3. This step concludes our source selection.

5 Evaluation

In this section we describe the experimental evaluation of our approach. We first describe our experimental setup in detail. Then, we present our evaluation results. All data used in this evaluation is either publicly available or can be found at the project web page.[8]

[8] `https://code.google.com/p/hibiscusfederation/`

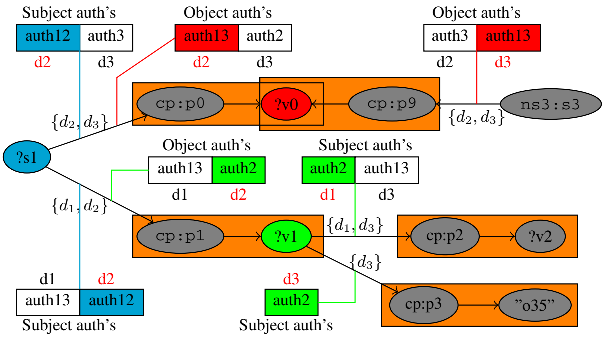


Fig. 3. Source pruning of Labeled hypergraph HSQ3 of Figure 1. All the sources highlighted in red are finally selected.

5.1 Experimental Setup

Benchmarking Environment: We used FedBench [13] for our evaluation. It is the only (to the best of our knowledge) benchmark that encompasses real-world datasets and commonly used queries within a distributed data environment. Furthermore, it is commonly used in the evaluation of SPARQL query federation systems [14,4,9,11]. Each of FedBench's nine datasets was loaded into a separate physical virtuoso server. The exact specifications of the servers can be found on the project website. All experiments were ran on a machine with a 2.70GHz i5 processor, 8 GB RAM and 300 GB hard disk. The experiments were carried out in a local network, so the network costs were negligible. Each query was executed 10 times and results were averaged. The query timeout was set to 30min (1800s). The threshold for the ASK-dominant approach was best selected to 0.33 after analysing results of different threshold values.

Federated Query Engines: We extended three SPARQL endpoint federation engines with HiBISCuS: DARQ [10] (index-only), FedX [14] (index-free), and SPLENDID [4] (hybrid). In each of the extensions, we only replaced the source selection with HiBISCuS. The query execution mechanisms remained unchanged. We compared our best extension (i.e., SPLENDID+HiBISCuS) with ANAPSID as this engine showed competitive results w.r.t. its index compression and number of TPW sources selected.

Metrics: We compared the three engines against their HiBISCuS extension. For each query we measured (1) the total number of TPW sources selected, (2) the total number of SPARQL ASK requests submitted during the source selection, (3) the average source selection time and (4) the average query execution time. We also compare the source index/data summaries generation time and index compression ratio of various state-of-the art source selection approaches.

5.2 Experimental Results

Index Construction Time and Compression Ratio. Table 2 shows a comparison of the index/data summaries construction time and the compression ratio[9] of various state-

[9] The compression ratio is given by (1 - index size/total data dump size).

Table 2. Comparison of index construction time and compression ratio. QTree's compression ratio is taken from [5]. (NA = Not Applicable).

	FedX	SPLENDID	LHD	DARQ	ANAPSID	Qtree	HiBISCuS
Index Generation Time (min)	NA	75	75	102	6	-	36
Compression Ratio (%)	NA	99.998	99.998	99.997	99.999	96	99.997

of-the art approaches. A high compression ratio is essential for fast index lookup during source selection. HiBISCuS has an index size of 458KB for the complete FedBench data dump (19.7 GB), leading to a high compression ratio of 99.99%. The other approaches achieve similar compression ratios. HiBISCuS's index construction time is second only to ANAPSID's. This is due to ANAPSID storing only the distinct predicates in its index. Our results yet suggest that our index containing more information is beneficial to the query execution time on FedBench.

Efficient Source Selection. We define efficient source selection in terms of: (1) the total number of TPW sources selected, (2) total number of SPARQL ASK requests used to obtain (1), and (3) the TPW source selection time. Table 3 shows a comparison of the source selection approaches of FedX, SPLENDID, ANAPSID and HiBISCuS based on these three metrics. Note that FedX (100% cached) means that we gave FedX enough memory to use only its cache to perform the complete source selection. This is the best-case scenario for FedX. Overall, HiBISCuS (ASK-dominant) is the most efficient approach in terms of total TPW sources selected, HiBISCuS (Index-dominant) is the most efficient *hybrid* approach in terms of total number of ASK request used, and FedX (100% cached) is most efficient in terms of source selection time. However, FedX (100% cached) clearly overestimates the set of sources that actually contributes to the final result set of query. In the next section, we will see that this overestimation of sources greatly leads to a slightly higher overall query runtime. For ANAPSID, the results are based on Star-Shaped Group Multiple endpoint selection (SSGM) heuristics presented in its extension [9]. Further, the source selection time represents the query decomposition time as both of these steps are intermingled.

Query Execution Time. The most important criterion when optimizing federated query execution engines is the query execution time. Figures 4, 5, and 6 show the results of our query execution time experiments. Our main results can be summarized as follows:

(1) Overall, *the ASK-dominant (AD) version of our approach performs best.* AD is on average (over all 25 queries and 3 extensions) 27.82% faster than the index-dominant (ID) version. The reason for this improvement is due to ID overestimating sources in some queries. For example, in CD1, AD selects the optimal number of sources (i.e., 4) while ID selects 12 sources. In some cases, the overestimation of sources by ID also slows down the source pruning (e.g. CD2),

(2) A comparison of our extensions with AD shows that *all extensions are more time-efficient than the original systems.* In particular, FedX's (100% cached) runtime is

Table 3. Comparison of the source selection in terms of total TPW sources selected **#T**, total number of SPARQL ASK requests **#A**, and source selection time **ST** in msec. **ST*** represents the source selection time for FedX(100% cached i.e. #A =0 for all queries) which is very rare in practical. **ST**** represents the source selection time for HiBISCuS (AD,warm) with #A =0 for all queries. (**AD** = ASK-dominant, **ID** = index-dominant, **ZR** = Zero results, **NS** = Not supported, **T/A** = Total/Avg., where Total is for #T, #A, and Avg. is ST, ST*, and ST**).

	FedX				SPLENDID			DARQ			ANAPSID			HiBISCuS(AD)				HiBISCuS(ID)		
Qry	#T	#A	ST	ST*	#T	#A	ST	#T	#A	ST	#T	#A	ST	#T	#A	ST	ST**	#T	#A	ST
CD1	11	27	285	**6**	11	27	392	NS	NS	NS	**3**	20	667	4	18	215	36	12	**0**	363
CD2	**3**	27	200	6	**3**	18	294	10	**0**	6	**3**	1	42	**3**	9	4	**3**	**3**	**0**	57
CD3	12	45	367	**8**	12	18	304	20	**0**	12	**5**	2	73	**5**	**0**	77	41	**5**	**0**	91
CD4	19	45	359	**8**	19	9	310	20	**0**	12	**5**	3	128	**5**	**0**	54	52	**5**	**0**	179
CD5	11	36	374	7	11	9	313	11	**0**	**4**	**4**	1	66	**4**	**0**	25	23	**4**	**0**	58
CD6	9	36	316	**8**	9	9	298	10	**0**	11	10	11	140	**8**	**0**	36	23	**8**	**0**	54
CD7	13	36	324	9	13	9	335	13	**0**	**6**	**6**	5	ZR	**6**	**0**	30	35	**6**	**0**	55
LS1	**1**	18	248	9	**1**	**0**	217	**1**	**0**	**4**	**1**	**0**	35	**1**	**0**	5	6	**1**	**0**	9
LS2	11	27	264	**8**	11	27	390	NS	NS	NS	12	30	548	**7**	18	118	60	**7**	**0**	118
LS3	12	45	413	**8**	12	9	310	20	**0**	9	**5**	13	808	**5**	**0**	31	27	**5**	**0**	200
LS4	**7**	63	445	**7**	**7**	18	287	15	**0**	15	**7**	1	314	**7**	**0**	8	9	**7**	**0**	15
LS5	10	54	440	**8**	10	9	308	18	**0**	13	**7**	4	885	8	**0**	20	21	8	**0**	44
LS6	9	45	430	8	9	18	347	17	**0**	**7**	**5**	13	559	7	**0**	23	22	7	**0**	42
LS7	**6**	45	389	8	**6**	9	292	6	**0**	**5**	7	2	193	**6**	**0**	18	17	**6**	**0**	24
LD1	8	27	297	8	8	9	295	11	**0**	**7**	**3**	1	428	**3**	**0**	24	19	**3**	**0**	21
LD2	**3**	27	320	**7**	**3**	9	268	3	**0**	9	**3**	**0**	34	**3**	**0**	3	5	**3**	**0**	6
LD3	16	36	330	**9**	16	9	324	16	**0**	11	**4**	2	130	**4**	**0**	31	29	**4**	**0**	48
LD4	**5**	45	326	7	**5**	18	290	5	**0**	17	**5**	**0**	33	**5**	**0**	**6**	7	**5**	**0**	10
LD5	5	27	280	8	5	18	236	13	**0**	**4**	**3**	2	210	**3**	**0**	9	9	**3**	**0**	19
LD6	14	45	385	**8**	14	9	331	14	**0**	**8**	14	12	589	**7**	**0**	32	30	**7**	**0**	136
LD7	3	18	258	7	3	9	235	4	**0**	**4**	**2**	4	223	4	**0**	7	7	4	**0**	11
LD8	15	45	337	8	15	9	333	15	**0**	**7**	9	7	1226	**5**	**0**	23	25	**5**	**0**	41
LD9	**3**	27	228	12	3	18	188	6	**0**	**3**	**3**	3	1052	**3**	9	50	3	**3**	**0**	17
LD10	10	27	274	8	10	9	309	11	**0**	**6**	**3**	4	2010	**3**	**0**	19	18	**3**	**0**	27
LD11	15	45	351	**7**	15	9	260	15	**0**	9	**5**	2	2904	7	**0**	23	24	7	**0**	42
T/A	**231**	**918**	**330**	**8**	**231**	**315**	**299**	**274**	**0**	**8**	**134**	**143**	**554**	**123**	**54**	**36**	**22**	**131**	**0**	**67**

improved in 20/25 queries (net query runtime improvement of 24.61%), FedX's (cold) is improved in 25/25 queries (net improvement: 53.05%), SPLENDID 's is improved in 25/25 queries (net improvement: 82.72%) and DARQ's is improved in 23/23 (2 queries are not supported) queries (net improvement: 92.22%). Note that these values were computed only on those queries that did not time-out. Thus, the net improvement brought about by AD is actually even better than the reported values. The reason for our slight (less than 5 msec) greater runtime for 5/25 queries in FedX (100% cached) is due to FedX (100% cached) already selecting the optimal sources for these queries. Thus, the overhead due to our pruning of the already optimal list of sources affects the overall query runtime.

(3) *Our extensions allow some queries that timed out to be carried out before the time-out.* This is especially the case for our DARQ extension, where LD6 and LD10 are

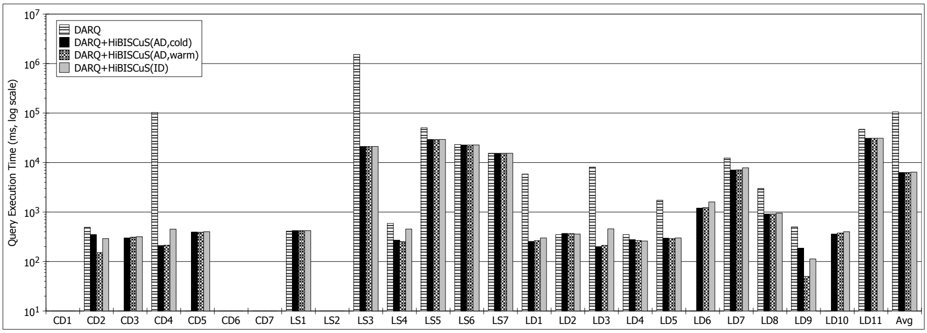


Fig. 4. Query runtime of DARQ and its HiBISCuS extensions. CD1, LS2 not supported, CD6 runtime error, CD7 time out for both. LD6, LD10 timeout and CD3 runtime error for DARQ.

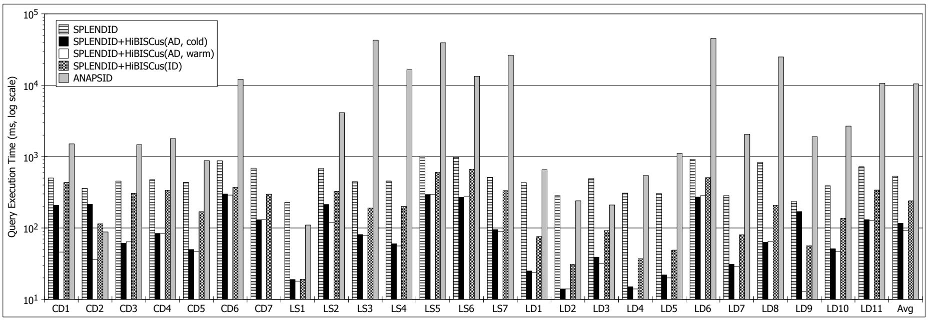


Fig. 5. Query runtime of ANAPSID, SPLENDID and its HiBISCuS extensions. We have zero results for ANAPSID CD7.

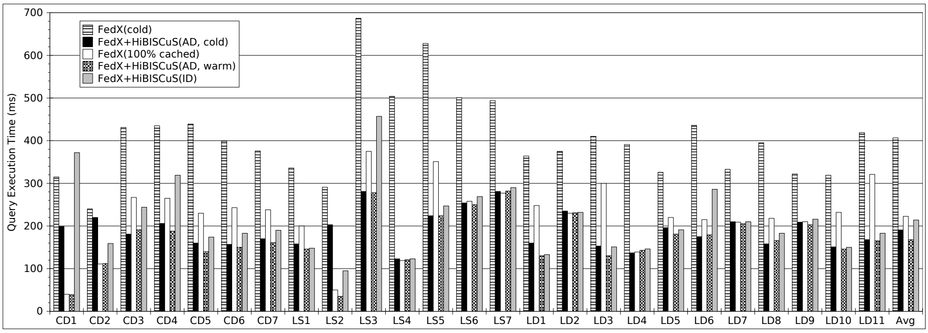


Fig. 6. Query runtime of FedX and its HiBISCuS extensions

carried out in 1123 msec and 377 msec respectively by DARD+AD, while they did not terminate within the time-out limit of 30 minutes on the original system.

(4) Our SPLENDID (AD) extension is 98.91% faster than ANAPSID on 24 of the 25 queries. For CD7, ANAPSID returned zero results.

An interesting observation is that FedX(100%) is better than SPLENDID in 25/25 queries and 58.17% faster on average query runtime. However, our AD extension of SPLENDID is better than AD extension of FedX(100%) in 20/25 queries and 45.20% faster on average query runtime. This means that SPLENDID is better than FedX in term of pure query execution time (excluding source selection time). A deeper investigation of the runtimes of both systems shows that SPLENDID spends on average 56.10% of total query execution on source selection. Thus, our extension showcase clearly that an efficient source selection is one of key factors in the overall optimization of federated SPARQL query processing.

6 Conclusion and Future Work

In this paper we presented HiBISCus, a labelled hypergraph based approach for efficient source selection for SPARQL endpoint federation. We evaluated our approach against DARQ, SPLENDID, FedX and ANAPSID. The evaluation shows that the query runtime of the first three systems is improved significantly on average.

In future, we will investigate the impact of the threshold θ on our approach. We will also study the effect of our source pruning algorithm on SPARQL 1.1 queries with SPARQL *service clause*, where the TPW sources are already specified by the user. Furthermore, we will evaluate our approach on big data as the query execution time for majority of the FedBench queries is less than 1s, which make it difficult to select the best SPARQL federation engine and have have a deeper look into the behaviour of these engines in different data environments.

Acknowledgments. This work was partially financed by the FP7 project GeoKnow (GA no. 318159).

References

1. Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: An adaptive query processing engine for SPARQL endpoints. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 18–34. Springer, Heidelberg (2011)
2. Akar, Z., Halaç, T.G., Ekinci, E.E., Dikenelli, O.: Querying the web of interlinked datasets using void descriptions. In: LDOW at WWW (2012)
3. Auer, S., Lehmann, J., Ngonga Ngomo, A.-C.: Introduction to linked data and its lifecycle on the web. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 1–75. Springer, Heidelberg (2011)
4. Görlitz, O., Staab, S.: Splendid: Sparql endpoint federation exploiting void descriptions. In: COLD at ISWC (2011)
5. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: WWW (2010)
6. Kaoudi, Z., Koubarakis, M., Kyzirakos, K.: Atlas: Storing, updating and querying rdf(s) data on top of dhts. JWS 8(4) (2010)

7. Ladwig, G., Tran, T.: Linked data query processing strategies. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 453–469. Springer, Heidelberg (2010)
8. Lynden, S., Kojima, I., Matono, A., Tanimura, Y.: ADERIS: An adaptive query processor for joining federated SPARQL endpoints. In: Meersman, R., et al. (eds.) OTM 2011, Part II. LNCS, vol. 7045, pp. 808–817. Springer, Heidelberg (2011)
9. Montoya, G., Vidal, M.-E., Acosta, M.: A heuristic-based approach for planning federated sparql queries. In: COLD (2012)
10. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
11. Saleem, M., Ngonga Ngomo, A.-C., Xavier Parreira, J., Deus, H.F., Hauswirth, M.: DAW: Duplicate-aWare federated query processing over the web of data. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 574–590. Springer, Heidelberg (2013)
12. Saleem, M., Shanmukha, S., Ngonga, A.-C., Almeida, J.S., Decker, S., Deus, H.F.: Linked cancer genome atlas database. In: I-Semantics (2013)
13. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A benchmark suite for federated semantic data query processing. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011)
14. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization techniques for federated query processing on linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)
15. Wang, X., Tiropanis, T., Davis, H.C.: Lhd: Optimising linked data query processing using parallelisation. In: LDOW at WWW (2013)