QUETSAL: A Query Federation Suite for SPARQL

Muhammad Saleem and Axel-Cyrille Ngonga Ngomo

Universität Leipzig, IFI/AKSW, PO 100920, D-04009 Leipzig
{lastname}informatik.uni-leipzig.de

Abstract. Efficient source selection is one of the most important optimization steps in federated SPARQL query processing. 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 been shown to yield great improvement potential. This work presents QUETSAL, a novel query engine that combines a global join-aware source selection approach with SPARQL query re-writing for federated SPARQL query processing. We compared our approach with state-of-the-art SPARQL query federation engines in terms of efficient source selection and overall query runtime on FedBench. Our evaluation shows that QUETSAL is the first engine to remain under 5% overall relevant sources overestimation with average source selection time under 160 msec. Moreover, our approach outperforms existing engines with respect to its execution time on most of the FedBench queries.

1 Introduction

The distribution of information across several data sources is intrinsic to the architectural choices behind the Web of Data [3]. For example, information on Slovenia can be found in `DBpedia`, `LinkedGeoData`, `GeoNames` and other data sources. Hence, answering certain queries requires retrieving results contained across different data sources (short: sources). The optimization of engines that support this type of queries, called *federated query engines*, is thus of central importance to ensure the usability of the Web of Data in real-world applications. The efficient selection of relevant sources for a query has been shown to be essential for the performance of federated query engines in previous works [12,11]. To ensure that a recall of 100% is achieved, most SPARQL query federation approaches [4,7,9,15,16,12,11] 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 [12]. However, it is possible that a relevant source does not *contribute* (formally defined in section 3) 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. For example, consider the FedBench query named `LS2` shown in Listing 1.1. A TPWSS that retrieves all relevant sources for each individual triple pattern would lead to all sources in the benchmark being queried. This is because the third triple pattern (`?caff ?predicate ?object`) can be answered by all of the datasets. Yet, the complete result set of the query given in Listing 1.1 can be computed by only querying `DrugBank` and `DBpedia` due to the object-subject join on `?caff` used in the query.

Listing 1.1: Motivating Example. FedBench query LS2: Find all properties of Caffeine in Drugbank. Find all entities from all available databases describing Caffeine, return the union of all properties of any of these entities.

```
PREFIX drugbank-drug: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?predicate ?object WHERE {
  { drugbank-drug:DB00201 ?predicate ?object . }   //DrugBank
UNION
  { drugbank-drug:DB00201 owl:sameAs ?caff .       //DrugBank
    ?caff ?predicate ?object . }                   //DrugBank, DBpedia, ChEBI
}
```

HiBISCuS [11], a *join-aware approach to TPWSS* (i.e., take care of the query triple pattern joins during source selection), was proposed with the aim to only select those sources that actually contribute to the final result set of the query. This approach makes use of the different *URIs authorities*[1] to prune irrelevant sources during the source selection. While HiBISCuS can significantly remove irrelevant sources [11], it fails to prune those sources which share the same URI authority. For example, all the `Bio2RDF`[2] sources contains the same URI authority `bio2rd.org`. Similarly, all the `Linked TCGA` sources[3] [13] share the same URI authority `tcga.deri.ie`. Coming back to the motivating example, imagine we only had the three sources presented in Figure 1. HiBISCuS would select `DrugBank` as single relevant source for the second triple pattern of the query given in Listing 1.1. The set of distinct *object* authorities (`DBpedia.org`, `bio2rdf.org` in our case) for predicate `owl:sameAs` of `DrugBank` would be retrieved from the HiBISCuS index. The set of distinct *subject* authorities (`DBpedia.org`, `bio2rdf.org` in our case) would be retrieved for all predicates (as the predicate is variable) and for all data sources (as all are relevant). Finally, the intersection of the set of authorities would again result in the same authorities set. Thus, HiBISCuS would not prune any source. This is because all three datasets in the example share the same URI authorities for the given query triple patterns.

We thus propose a *novel federated SPARQL query engine* dubbed QUETSAL which combines *join-aware approach to TPWSS based on common prefixes* with *query rewriting* to outperform the state-of-the-art on federated query processing. By moving away from authorities, our approach is flexible enough to distinguish between URIs from different datasets that come from the same namespace (e.g., as in `Bio2RDF`). Moreover, our query rewriting allows reducing the number of queries shipped across the network and thus improve the overall runtime of our federated engine.

Overall, our contributions are thus as follows:

1. We present a novel source selection algorithm based on labelled hypergraphs. Our algorithm relies on a novel type of data summaries for SPARQL endpoints which relies on most common prefixes for URIs.

[1] URI syntax: `http://tools.ietf.org/html/rfc3986`
[2] Bio2RDF: `http://download.bio2rdf.org/release/2/release.html`
[3] Linked TCGA: `http://tcga.deri.ie/`

```
@PREFIX drugbank-drug:
<http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/>.
@PREFIX drugbank-ref:
<http://www4.wiwiss.fu-berlin.de/drugbank/resource/references/>.
@PREFIX dbpedia-resource: <http://dbpedia.org/resource/>.
@PREFIX bio2rdf-pubmed: <http://bio2rdf.org/pubmed>.
drugbank-drug:DB00201 owl:sameAs dbpedia-resource:Caffeine.
drugbank-ref:1002129 owl:sameAs bio2rdf-pubmed:1002129.
```

(a) DrugBank

```
@PREFIX dbpedia-resource: <http://dbpedia.org/resource/>.
dbpedia-resource:Caffeine foaf:name "Caffeine" .
dbpedia-resource:AC_Omonia foaf:name "AC_Omonia" .
```

(b) DBpedia

```
@PREFIX bio2rdf-chebi: <http://bio2rdf.org/chebi:>.
bio2rdf-chebi:21073 chebi:Status bio2rdf-chebi:status-C .
bio2rdf-chebi:21073 rdf:type bio2rdf-chebi:Compound .
```

(c) ChEBI

Fig. 1: Motivating Example. FedBench selected datasets slices. Only relevant prefixes are shown.

2. We devise a pruning algorithm for edge labels that enables us to discard irrelevant sources based on common prefixes used in joins.
3. We compared our approach with state-of-the-art federate query engines (FedX [15], SPLENDID [4], ANAPSID [1], and SPLENDID+HiBISCus [11]). Our results show that we have reduced the number of sources selected (without losing recall), the source selection time as well as the overall query runtime by executing many remote joins (i.e., joins shipped to SPARQL endpoints).

The rest of the paper is structured as follows: we first give a brief overview of federated query engines. Then, we present formal concepts and notation required to understand our work. The algorithms underlying QUETSAL are then explained in detail. Finally, we evaluate our approach against the state-of-the-art and show that we achieve both better source selection and runtimes on the FedBench [14] SPARQL query federation benchmark.

2 Related Work

Federated query processing approaches over the Web of Data can be divided into two main categories (see Table 1 obtained from [10]): (1) *SPARQL endpoints federation approaches* federate SPARQL queries over a known set of SPARQL endpoints, provided as input to the federation engine. These approaches can retrieve fast results because of the optimization both at client side(i.e, federation engine) and server side (i.e., SPARQL endpoints). However, the RDF data needs to be exposed as SPARQL endpoints. Due to which they are unable to deal with whole LOD. (2) *Linked Data federation approaches or*

Table 1: Classification of SPARQL federation engines. (**SEF** = SPARQL Endpoints Federation, **LDF** = Linked Data Federation, **Ctg.** = Federation Type, **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 [15]	LHD [16]	SPL [4]	DAW* [12]	ANAPSID [1]	ADE [7]	DARQ [9]	LDQP [6]	WoDQA [2]	QTree* [5]	HiBISCus* [11]	QUETSAL
Ctg.	SEF	SEF	SEF	-	SEF	SEF	SEF	LDF	LDF	-	-	SEF
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	HB(A+I)	HB(A+I)
I.U	NA	✗	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓

link traversal approaches do not require the data to be exposed via SPARQL endpoints. The only requirement is that it should follow the Linked Data principles[4]. Due to URI lookups at runtime, these type of approaches may be on a slower side than the previous type of approaches.

The source selection approaches used in both categories can further divided into three sub-categories (see Table 1). (1)*Index-assisted source selection* only makes use of the index (also called data summaries) to perform TPWSS. The result completeness (100% recall) must be ensured by keeping the index up-to-date. (2)*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. These type of approaches usually make use of the SPARQL ASK queries or online link traversal to perform TPWSS [10]. However, they commonly have a longer query execution time due to the extra processing required for collecting on-the-fly statistics. (3) *Hybrid source selection* approaches are a combination of the previous approaches. A detailed discussion of the SPARQL query federation approaches is provided in [10].

QUETSAL is a SPARQL endpoint federation approach which makes use of a novel a novel hybrid source selection approach based on *trie* data structure and *directed labelled hypergraphs (DLH)*.

3 Preliminaries

In the following, we present the core concepts and notation that are used throughout this paper. Note that some of the definitions are reused from [11]. 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*.

Definition 1 (Contributing source). *Let $D = \{d_1, \ldots, d_n\}$ be the set of sources. 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 2 (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.*[5] *The*

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

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

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, `DrugBank` is the single relevant source for the second triple pattern of the example query given in Listing 1.1. As explained earlier, it is possible that a relevant source for a triple pattern does not *contribute* to the final result set of the complete query q. `ChEBI` is an example of such source which is relevant to third triple pattern of the motivating example query. However, it does not contributes to the final result set of the complete query.

Definition 3 (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 optimal source set for the third triple pattern of the motivating example query is $\{DBpedia\}$, while the set of relevant sources for the same triple pattern is $\{DrugBank, DBpedia, ChEBI\}$. Formally, the problem of TPWSS can then be defined as follows:

Definition 4 (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 [4,7,9,15,16] 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 a hybrid approach for (1) the time-efficient computation of the relevant source set R_i for individual triple patterns of the query and (2) the approximation of O_i out of R_i. QUETSAL 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. Like HiBISCuS, QUETSAL relies on directed labelled hypergraphs (DLH) to achieve this goal. In the following, we present a formalization of SPARQL queries as DLH. Subsequently, we show how we make use of this formalization to approximate O_i for each tp_i.

3.1 Queries as Directed Labelled Hypergraphs

The DLH representation of a BGP[6] is formally defined as follows:

Definition 5. *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*

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;*

[6] BGP: `http://www.w3.org/TR/sparql11-query/#BasicGraphPatterns`

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'.*

A DLH representation of a SPARQL query is given in QUETSAL's supplementary material[7] provided with the submission (ref. Listing 1, Figure 1 of supplementary material). We can now reformulate our problem statement as follows:

Definition 6 (Problem Reformulation). *Given a query* q *represented as a set of hypergraphs* $\{HG_1, \ldots, HG_x\}$ *representing the BGPs of* q*, find the labelling of the hyperedges of each hypergraph* HG_i *that leads to an optimal source selection.*

4 QUETSAL

In this section we present our approach in details. First, we explain our *lightweight data summaries* construction based on common prefixes, followed by the source selection algorithms and then we explain the SPARQL query rewriting and execution.

4.1 Data Summaries

QUETSAL relies on *capabilities* to compute data summaries. Given a source d, we define a capability as a triple $(p, SP(d,p), OP(d,p))$ which contains (1) a predicate p in d, (2) the set $SP(d,p)$ of all distinct *subject prefixes* of p in d and (3) the set $OA(d,p)$ of all distinct *object prefixes* of p in d. In QUETSAL, 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 innovation behind the QUETSAL data summaries is to use common prefixes to characterize resources in data sets. By going beyond mere authorities, we can ensure that URIs that come from different datasets published by the same authority can still be differentiated. To achieve this goal, we rely on prefix trees (short: tries). The idea here is to record the most common prefixes of URIs used as subject or object of any given property and store those in the `Quetsal` index.

Let ρ be a set of resources for which we want to find the most common prefixes (e.g., the set of all subject resources for a given predicate). We begin by adding all the resources containing the same *URI authority* in ρ to a trie. While we use a character-by-character insertion in our implementation, we present word-by-word insertion for the sake of clarity and space in the paper. Inserting the resources `drugbank-drug:DB00201`

[7] QUETSAL's supplementary material can also be found at `http://goo.gl/O9NH7M`

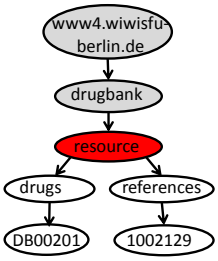

Fig. 2: Trie of URIs.

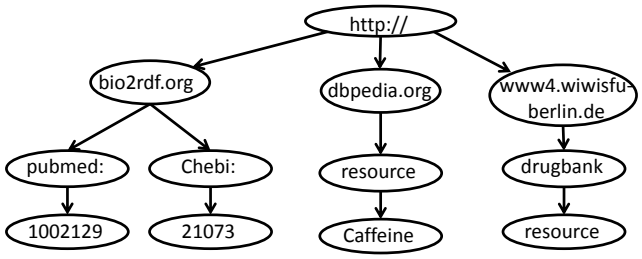

Fig. 3: Trie of all Prefixes.

and `drugbank-ref:1002129` from DrugBank leads to the trie shown Figure 2. We consider a node to be the end of a common prefix if (1) it is not the root of the tree and (2) the branching factor of the said node is higher than a preset threshold. For example, if our threshold were 1, the node `resource` would be the end of a common prefix. By inserting all nodes from ρ into a trie and marking all ends of common prefix, we can now compute all common prefixes for ρ by simply traversing computing all paths from the root to the marked nodes. In our example, we get exactly one common prefix for the branching limit equal to 1.

The predicate `rdf:type` is given a special treatment: Instead of storing the set of all distinct object prefixes 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. The QUETSAL's data summaries (branching limit = 1) for the three datasets of our motivating example is provided in the supplementary material of the paper (ref. Listing 3 of supplementary material). Note we only compute prefixes for URIs (common prefixes can be found). In the next section, we will make use of these data summaries to optimize the *TPWSS*.

4.2 Source Selection Algorithm

QUETSAL's source selection comprises 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 $HG_i \in DHG$. 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 Algorithm 1 (labelling) as well as Algorithm 2 (pruning).

Hyperedge Labelling The hyperedge labelling Algorithm 1 takes the set of all sources D, the set of all disjunctive hypergraphs DHG (one per BGP of the query) of the input query q and the data summaries QUETSAL_D of all sources in D as input and returns a set of *labelled* disjunctive hypergraphs as output. For each hypergraph and each hyperedge,

Algorithm 1 QUETSAL's hyperedge labelling algorithm

Require: D= $\{d_1, \ldots, d_n\}$; DHG = $\{HG_1, \ldots, HG_x\}$; $QUETSAL_D$ //sources, disjunctive hypergraphs of a query, QUETSAL sumaries 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: $\lambda_e(e_i)$ = $QUETSAL_D lookup$(p, o)
11: **else if** !commonpredicate(p) $\vee$ (!bound(s) $\wedge$!bound(o)) **then**
12: $\lambda_e(e_i)$ = $QUETSAL_D lookup$(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

the subject, predicate, object, subject authority, and object authority are collected (Lines 2-5 of Algorithms 1). 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). 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).

Algorithm 1 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).

4.3 Pruning approach

The intuition behind our pruning approach is that knowing which stored prefixes 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 2) 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 2).

For each node v of a DLH that is not of type 'simple', we first retrieve the sets (1) $SPrefix$ of the subject prefixes contained in the elements of the label of each outgoing edge of v (Lines 5-7 of Algorithm 2) and (2) $OPrefix$ of the object prefixes contained in the elements of the label of each ingoing edge of v (Lines 8-10 of Algorithm 2). Note that these are sets of sets of prefixes. For the node $?caff$ of the query in our running example given in Listing 1.1, we get
$SPrefix = \{\{$`http://www4.wiwiss.fu-berlin.de/drugbank/resource`$\}$, $\{$`http://dbpedia.org/resource/`$\}, \{$`http://bio2rdf.org/chebi:`$\}$, $\{$`http://bio2rdf.org/chebi:21073`$\}\}$ for the outgoing edge and $OAuth = \{\{$`http://dbpedia.org/resource/Caffeine`$\}$, $\{$`http://bio2rdf.org/pubmed:1002129`$\}\}$ for the incoming edges. Now we merge these two sets to the set P of all prefixes (Line 11 of Algorithm 2). Next, we plot all of the prefixes of P in to a trie (no branching limit) shown in Figure 3. Now we check each prefix in P whether at ends at a child node of trie T or not. If a prefix does not end at child node then we get all of the paths from the prefix last node (say n) to each leaf of n. In our example, `http://dbpedia.org/resource/`, `http://bio2rdf.org/chebi:` does not end at leaf nodes (see Figure 3), so they will be replaced with `http://dbpedia.org/resource/Caffeine`, `http://bio2rdf.org/chebi:21073`, respectively in P (Line 13-22 of Algorithm 2). The intersection $I = \left(\bigcap_{p_i \in P} p_i \right)$ of these elements sets is then computed. In our example, this results in
$I = \{$`http://dbpedia.org/resource/Caffeine`$\}$. 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 of e which is such that the set of prefixes of each of its elements is not disjoint with I (see Lines 27-35 of Algorithm 2). These are the only sources that will really contribute to the final result set of the query. In our example, `ChEBI` will be removed since the `ChEBI` subject prefixes does not intersect (i.e., disjoint) with the elements in I and `DBpedia` will be selected since its subject prefix `http://dbpedia.org/resource/` was replaced with `http://dbpedia.org/resource/Caffeine` which intersect with I.

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.

4.4 SPARQL 1.1 Query Re-writing

QUETSAL converts each SPARQL 1.0 query into corresponding SPARQL 1.1 query. Before going into the details of our SPARQL 1.1 query rewriting, we first introduce the notion of exclusive groups (used in SPARQL 1.1 query rewrite) in the SPARQL query.

Exclusive Groups In normal SPARQL query (i.e., not a federated query) execution, the user sends a complete query to the SPARQL endpoint and gets the results back from

Algorithm 2 Hyperedge label pruning algorithm for removing irrelevant sources

Require: DHG //disjunctive hypergraphs
for each $HG_i \in DHG$ **do**
  for each v $\in$ vertices(HG_i) **do**
    if $\lambda_{vt}(v) \neq$ 'simple' **then**
      $SPrefix = \emptyset$; $OPrefix = \emptyset$;
      for each $e \in E_{out}(v)$ **do**
        $SPrefix = SPrefix \cup \{$subjectPrefixes($e$)$\}$
      end for
      for each $e \in E_{in}(v)$ **do**
        $OPrefix = OPrefix \cup \{$objectPrefixes($e$)$\}$
      end for
      $P = SPrefix \cup OPrefix$ // set of all prefixes
      $T =$ getTrie(P) //get Trie of all prefixes, no branching limit
      for each $p \in P$ **do**
        if !isLeafPrefix(p,T) // prefix does not end at leaf node of Trie **then**
          C = getAllChildPaths(p) // get all paths from prefix last node n to each leaf of n
          $A = \emptyset$ //to store all possible prefixes of a given prefix
          for each $c \in C$ **do**
            $A = A \cup p$.Concatenate(c)
          end for
          P.replace(p,A) // replace p with its all possible prefixes
        end if
      end for
      $I = P$.get(1) //get first element of prefixes
      for each $p \in P$ **do**
        $I = I \cap p$ //intersection of all elements of A
      end for
      for each $e \in E_{in}(v) \cup E_{out}(v)$ **do**
        $label = \emptyset$ //variable for final label of e
        for $d_i \in \lambda_e(e)$ **do**
          if prefixes(d_i) $\cap I \neq \emptyset$ **then**
            $label = label \cup d_i$
          end if
        end for
        $\lambda_e(e) = label$
      end for
    end if
  end for
end for

the endpoint, i.e., the complete query is executed at SPARQL endpoint. Unlike normal SPARQL query execution, in general, the federated engine sends sub-queries to the corresponding SPARQL endpoints and get the sub-queries results back which are locally integrated by using different *join* techniques. The local execution of joins results in high costs, in particular when intermediate results are large[15]. To minimize these costs, many of the existing SPARQL federation engines [15,4,1] make use of the notion of exclusive groups which is formally defined as:

Definition 7. *Let BGP =* $\{tp_1, \ldots, tp_m\}$ *be a basic graph patterns containing a set of triple patterns* $tp_1, \ldots, tp_m$*,* $D = \{d_1, \ldots, dn\}$ *be the set of distinct data sources, and* $R_{tp_i} = \{d_1, \ldots, d_o\} \subseteq D$ *be the set of relevant data sources for triple pattern* tp_i*. We define* $EG_d = \{tp_1, \ldots, tp_p\} \subseteq BGP$ *be the exclusive groups of triple patterns for a data soruce* $d \in D$ *s.t.* $\forall_{tp_i \in EG_d} R_{tp_i} = \{d\}$*, i.e., the triple patterns whose single relevant source is d.*

The advantage of exclusive groups (size greater than 1) is that they can be combined together (as a conjunctive query) and send to the corresponding data source (i.e., SPARQL endpoints) in a single sub-query, thus greatly minimizing: (1) the number of remote requests, (2) the number of local joins, (3) the number of irrelevant intermediate results and (4) the network traffic [15]. This is because in many cases the intermediate results of the individual triple patterns are often excluded after performing join with the intermediate results of another triple pattern in the same query. On the other hand, the triple pattern joins in the exclusive groups are directly performed by the data source itself (we call them remote joins), thus all intermediate irrelevant results are directly filtered without sending via network. Correctness is guaranteed as no other data source can contribute to the group of triple patterns with further information.

Consider the query given in Listing 1 of the supplementary material. The last three triple patterns form an exclusive group, since `DrugBank` is the single relevant source for these triple patterns. Thus these triple patterns can be directly executed by `DrugBank`.

Our SPARQL 1.0 to SPARQL 1.1 query rewrite makes use of the exclusive groups, SPARQL SERVICE, and SPARQL UNION clauses as follows: (1) Identify exclusive groups from the results of the source selection, (2) group each exclusive group of triple patterns into a separate SPARQL SERVICE, and (3) write a separate SPARQL SERVICE clause for each triple patterns (which are not part of any exclusive group) and for each relevant source of that triple pattern and union them using SPARQL UNION clause.

A SPARQL 1.1 query rewrite of the SPARQL 1.0 query given in Listing 1 of the supplementary material is shown in Listing 2 of the supplementary material. The exclusive group of triple patterns (i.e., last three triple patterns) are grouped into `DrugBank` SERVICE. Since both KEGG and `ChEBI` are the relevant data sources for the first triple pattern, a separate SPARQL SERVICE is used for each of the data source and the results are combined using SPARQL UNION clause. The final SPARQL 1.1 query is then directly executed using Sesame API. The number of remote joins for an exclusive group is one less than the number of triple patterns in that group. For example, the number of remote joins in our SPARQL 1.1 query is 2. The number of remote joins for a complete query is the sum of the number of remote joins of all exclusive groups in that query.

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]

5.1 Experimental Setup

Benchmarking Environment: We used FedBench [14] for our evaluation as it is commonly used in the evaluation of SPARQL query federation systems [15,4,8,12]. FedBench comprises a total of 25 queries out of which 14 queries are for SPARQL endpoint federation approaches and 11 queries for Linked Data federation approaches. We considered all of the 14 SPARQL endpoint federation queries. Each of FedBench's nine datasets was loaded into a separate physical virtuoso server. The exact specification of the

[8] QUETSAL home page: `https://code.google.com/p/quetsal/`

Table 2: Comparison of Index Generation Time **IGT** in minutes, Compression Ratio **CR**, Average (over all 14 queries) Source Selection Time **ASST**, and over all overestimation of relevant data sources **OE** . QTree's compression ratio is taken from [5] and DARQ, LHD results are calculated from [10]. (NA = Not Applicable, B1 = QUETSAL branching limit 1, B2 = QUESTSAL branching limit 2 and so on).

	FedX	SPLENDID	LHD	DARQ	ANAPSID	Qtree	HiBISCuS	QUETSAL				
								B1	B2	B3	B4	B5
IGT	NA	110	100	115	5	-	36	40	45	51	54	63
CR	NA	99.998	99.998	99.997	99.999	96.000	99.997	99.997	99.997	99.996	99.995	99.993
ASST	5.5	332	14	8	136	-	26	101	108	116	129	160
OE	50%	50%	69%	62%	16.2%	-	11.80%	9.40%	9.40%	9.40%	4.20%	4.20%

servers can be found on the project website. All experiments were ran on a machine with a 2.9GHz i7 processor, 8 GB RAM and 500 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 10min (1800s). The threshold for the labelling Algorithm 1 was best set to 0.33 based on a prior experiments.
Federated Query Engines: We compared QUETSAL with the latest versions of FedX [15], ANAPSID [1], SPLENDID [4], and the best HiBISCuS extension (i.e., SPLENDID+HiBISCuS). To the best of our knowledge, these are the most up-to-date SPARQL endpoint federation engines.
Metrics: We compared the selected engines based on: (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, (4) the number of remote joins produced by the source selection (ref. Section 4.4), and (5) the average query execution time. We also compared 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-of-the art approaches. A high compression ratio is essential for fast index lookup during source selection. QUETSAL-B1 (i.e., QUETSAL with trie branching limit = 1) has an index size of only 520KB and QUETSAL-B5 (branching limit =5) has an index size of 1MB for the complete FedBench data dump (19.7 GB), leading to a high compression ratio of 99.99% in all B1-B5. The other approaches achieve similar compression ratios except Qtree. Our index construction time ranges from 40min to 63min for QUETSAL-B1 to QUETSAL-B5. This is the first index construction time, later updates are possible per individual capability (as they are independent of each others) of the data source by using simple SPARQL update operations (our index is in RDF). QUETSAL-B4 is the best choice in terms of source selection accuracy and average execution time (shown in next section).

Efficient Source Selection We define efficient source selection in terms of three metrics: (1) the total number of TPW sources selected, (2) total number of SPARQL ASK requests

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

Table 3: Comparison of the source selection in terms of total TPW sources selected **#T**, total number of SPARQL ASK requests **#A**, source selection time **ST** in msec, and total number of remote joins **#J** generated by the source selection. **ST*** represents the source selection time for FedX (100% cached i.e. #A =0 for all queries), #T1, #T4 QUETSAL total TPW sources selected for branching limit 1 and 4, respectively, #J1, #J4 QUETSAL number of remote joins for branching limit 1 and 4, respectively. (**T/A** = Total/Avg., where Total is for #T, #A, and Avg. is ST, ST*)

	FedX					SPLENDID				ANAPSID				HiBISCuS				QUETSAL							Optimal	
Qry	#T	#A	ST	ST*	#J	#T	#A	ST	#J	#T	#A	ST	#J	#T	#A	ST	#J	#T1	#T4	#A	ST1	ST4	#J1	#J4	#T	#J
CD1	11	27	221	5	0	11	26	293	0	3	19	227	1	4	18	36	0	4	4	18	50	139	0	0	3	1
CD2	3	27	211	6	1	3	9	293	1	3	1	46	1	3	9	8	1	3	3	9	19	26	1	1	3	1
CD3	12	45	255	5	2	12	2	400	2	5	2	82	3	5	0	45	3	5	5	0	76	122	3	3	5	3
CD4	19	45	265	7	1	19	2	340	1	5	3	74	3	5	0	52	3	5	5	0	207	210	3	3	5	3
CD5	11	36	250	8	1	11	1	330	1	4	1	54	2	4	0	23	2	4	4	0	124	160	2	2	4	2
CD6	9	36	245	5	0	9	2	303	0	9	10	35	0	8	0	23	0	8	8	0	85	90	0	0	8	2
CD7	13	36	252	6	0	13	2	354	0	6	5	32	1	6	0	34	1	6	6	0	114	162	1	1	6	3
LS1	1	18	200	5	0	1	0	189	0	1	0	55	0	1	0	9	0	1	1	0	12	11	0	0	1	0
LS2	11	27	230	6	0	11	26	592	0	15	19	356	0	7	18	28	0	5	4	18	226	270	0	0	3	0
LS3	12	45	267	5	2	12	1	334	2	5	11	262	2	5	0	27	3	5	5	0	228	250	3	3	5	3
LS4	7	63	288	7	5	7	2	299	5	7	0	333	5	7	0	9	5	7	7	0	22	26	5	5	7	5
LS5	10	54	243	5	2	10	1	355	2	7	4	105	3	8	0	26	2	8	7	0	88	140	2	3	6	3
LS6	9	45	264	3	1	9	2	262	1	5	12	180	2	7	0	22	1	7	5	0	98	133	1	3	5	3
LS7	6	45	254	5	1	6	1	252	1	5	2	81	2	6	0	23	1	6	6	0	71	80	1	1	6	2
T/A	134	549	246	**5**	16	134	77	328	16	80	89	137	**25**	76	**45**	26	22	74	**70**	**45**	101	129	22	**25**	67	31

used to obtain (1), (3) the TPW source selection time, (4) and the number of remote joins (ref. Section 4.4) produced by the source selection. Table 3 shows a comparison of the source selection of the selected engines. Note that FedX (100% cached) means that the complete source selection is done by only using cache, i.e., no SPARQL ASK request is used. This is the best-case scenario for FedX and rare in practical applications. Overall, QUETSAL-B4 is the most efficient approach in terms of total TPW sources selected, the total number of ASK request used, and the total number of remote joins (equal with ANAPSID) produced. 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.

Query execution time As mentioned before we considered all of the 14 FedBench's SPARQL endpoint federation queries. However, QUETSAL gives runtime error for three queries (reason explained at the end of this section). Thus, the selected federation engines were compared for the remaining 11 queries shown in Figure 4. Overall, QUETSAL-B1 performs best in terms of the number of queries count for which its runtime is better than others. QUETSAL-B1 is better than FedX in 6/11 queries. FedX is better than ANAPSID in 7/11 which in turn better than QUETSAL-B4 in 7/11 queries. QUETSAL-B4 is better than SPLENDID+HiBISCuS in 6/11 queries which in turn better than SPLENDID in 10/11 queries. However, QUETSAL-B4 is better of all in terms of the average (over all 11 queries) query runtimes. As an overall net performance evaluation (based on overall average query runtime), QUETSAL-B4 is 15% better than FedX which in turn 10% better than ANAPSID. ANAPSID is 15% better than QUETSAL-B1 which is 11% better than SPLENDID+HiBISCUS.

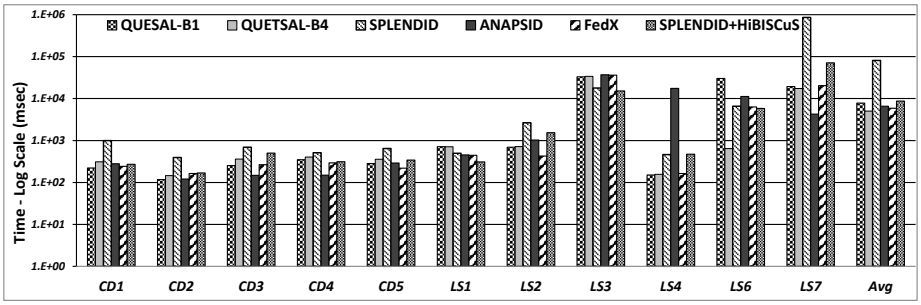

Fig. 4: Query runtime evaluation.

An interesting observation is that QUETSAL-B4 ranked fourth in terms of the total query count for which one system perform better than others. However, it ranks first in terms of the average performance evaluation. The reason is that QUETSAL-B4 spend an extra 30ms (on average) for the source selection as compared to QUETSAL-B1 (ref. Table 3). The majority of the FedBench queries are simple in complexity, thus the query runtimes are very small (9/11 queries have runtime less than 750msec for QUETSAL). Consequently, 30msec makes a big difference in FedBench, showing the need of another federation benchmark which contains more complex queries.

The most interesting runtime results was observed for query LS6 of FedBench. QUETSAL-B4 only spends 643msec while QUETSAL-B1 spends 31sec for the same query (other systems also spend more than 5 seconds). A deeper look in to the query results shown that QUETSAL-B4 produces the maximum number of remote joins (i.e., 3) for this query. There was only a single join needs to be performed locally by the QUETSAL. This shows that one of the main optimization step in SPARQL endpoint federation is to generate maximum remote joins (loads transferring). The number of remote joins are directly related to the number of exclusive groups generated which in turn directly related to efficient triple pattern-wise source selection. Thus, QUETSAL was proposed with the main aim to produce maximum remote joins via efficient triple pattern-wise source selection. Finally, we looked in to the reason behind the three queries (LD6, LD6, LS5) results in runtime errors in QUETSAL. It was noted that QUETSAL SPARQL 1.1 query rewrite produces many single triple pattern SPARQL SERVICE clauses and their results are union together using SPARQL UNION clauses. Thus Sesame first retrieve all of the results from each of the relevant source of that triple pattern and then union them, resulting in a to large number of intermediate results due to which a runtime error is thrown by the Sesame API. The runtime errors are provided at our project home. Finally, we believe that QUETSAL will perform more better in complex queries (containing many triple patterns) by producing many remote joins as well as on live SPARQL endpoints with Big Data where source overestimation is more expensive.

6 Conclusion and Future Work

In this paper we presented QUETSAL, a federated engine for SPARQL endpoint federation. We evaluated our approach against state-of-the-art federation systems. The evaluation shows that QUETSAL outperform the state-of-the-art by reducing the number

of sources selected and generating a considerable number of remote joins, a key to federated query processing optimization. However, the performance can be on a lower side if our approach fails to produce many remote joins. In future, we will make use of bind joins (as used in state-of-the-art engines) to solve the problem of retrieving many intermediate results for queries where many SPARQL `UNION` clauses are introduced by the SPARQL 1.1 query rewrite. Further, a more comprehensive SPARQL query federation benchmark (containing both simple and complex queries) based on real data and real queries is on the roadmap.

7 Acknowledgments

This work was partially financed by the FP7 project GeoKnow (GA no. 318159) as well as the BMWi project SAKE with the project number 01MD15006E.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. ANAPSID: an adaptive query processing engine for SPARQL endpoints. In *ISWC*, 2011.
2. Z. Akar, T. G. Halaç, E. E. Ekinci, and O. Dikenelli. Querying the web of interlinked datasets using void descriptions. In *LDOW at WWW*, 2012.
3. S. Auer, J. Lehmann, and A.-C. Ngonga Ngomo. Introduction to linked data and its lifecycle on the web. In *Reasoning Web*, 2011.
4. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD at ISWC*, 2011.
5. A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, 2010.
6. G. Ladwig and T. Tran. Linked data query processing strategies. In *ISWC*. 2010.
7. S. Lynden, I. Kojima, A. Matono, and Y. Tanimura. Aderis: An adaptive query processor for joining federated sparql endpoints. In *OTM*. 2011.
8. G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012.
9. B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, 2008.
10. M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. N. Ngomo. A fine-grained evaluation of sparql endpoint federation systems. *SWJ*, 2015.
11. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *ESWC*, 2014.
12. M. Saleem, A.-C. Ngonga Ngomo, J. X. Parreira, H. F. Deus, and M. Hauswirth. Daw: Duplicate-aware federated query processing over the web of data. In *ISWC*, 2013.
13. M. Saleem, S. S. Padmanabhuni, A.-C. N. Ngomo, A. Iqbal, J. S. Almeida, S. Decker, and H. F. Deus. TopFed: TCGA tailored federated query processing and linking to LOD. *JBMS*, 2014.
14. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: a benchmark suite for federated semantic data query processing. In *ISWC*, 2011.
15. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, 2011.
16. X. Wang, T. Tiropanis, and H. C. Davis. Lhd: Optimising linked data query processing using parallelisation. In *LDOW at WWW*, 2013.