SemCat: Source Selection Services for Linked Data

Pascal Molli, Hala Skaf-Molli, Arnaud Grall

SemCat: Source Selection Services for Linked Data

Pascal Molli[1], Hala Skaf-Molli[1], and Arnaud Grall[1,2]

[1] LS2N – University of Nantes, France
{pascal.molli,hala.skaf,arnaud.grall}@univ-nantes.fr
[2] GFI Informatique - IS/CIE, Nantes, France arnaud.grall@gfi.fr

Abstract. As a web search engine is able to find relevant sources for a keyword query, the web of data clearly needs a source selection engine able to find relevant endpoints for a SPARQL query. However, source selection requires to get informations about the content of endpoints and currently, it remains difficult to automatically explore the content of endpoints as web robots explore the content of web servers. Thanks to the web preemption principle, we propose to automatically build RDF summaries of endpoints through SPARQL queries. We propose SemCat, an approach to compute the source selection of a query Q by evaluating a rewriting of Q on summaries. As all queries terminate under the web preemption paradigm, SemCat is able to provide a web automated source selection service relying on preemptable SPARQL servers. We empirically demonstrate that various summaries can be extracted with a data transfer proportional to the size of the summary, and highlight the trade-off between the size of the summaries, the accuracy of source selection and the execution time of source selection.

Keywords: Linked Data; Semantic Web; SPARQL query service

1 Introduction

Context and motivation: Following the Linked Data principles [3] hundreds of interconnected knowledge graphs are available through public SPARQL Endpoints [22]. However, executing a SPARQL query at the web scale is still challenging. The main issue concerns source selection, i.e. given a SPARQL query, find the minimal number of relevant sources on the web of data to be contacted to execute the query.

Existing Federated Query Engines (FQE) [18] perform source selection just over a local catalog of sources and not over the web. In these conditions, a federated query engine barley build a federated database, not a global data space. As a web search engine is able to find relevant sources for a keyword query, the web of data clearly needs a source selection engine able to find relevant endpoints for a SPARQL query. As web search engines help the web to be decentralized, a source selection engine should help the web of data to be decentralized.

Related works and problem: Source selection has been extensively studied [18, 16]. Efficient source selection requires to know what are inside endpoints. This can be done by asking endpoints [18] but this cannot scale to large number of endpoints. Another option relies on summaries but this supposes that summaries can be obtained or computed in a fully automated and reliable way at web scale [8]. Clearly, there is no equivalent to web robots of web search engines able to crawl the whole set of endpoints. This prevents the automation of indexing of endpoints at the web scale. Recently, web preemption[10] allowed SPARQL queries to be executed sliced time, and [7] presented how it can be used to compute aggregates. This opens the door to the traversing of endpoints while collecting summaries.

Approach and Contributions: In this paper, we foster on web preemption to build SemCat; a source selection service for SPARQL endpoints. SemCat is a SPARQL endpoint hosting an RDF dataset containing summaries of endpoints. Thanks to preemptable servers, summaries of endpoints are obtained by executing summary functions as SPARQL queries over endpoints. The same summary functions can also be applied to any query Q to be executable on SemCat. The mappings produced by the execution of a query over SemCat allow to compute the source selection of the query Q over endpoints. The contributions of this paper are the following:

- We define a model for building a source selection service at web scale using only the semantic web technologies RDF and SPARQL.
- We demonstrate that summaries can be obtained with a data transfer proportional to the size of the summary and not to the size of data sources.
- We demonstrate how source selection can be obtained by executing SPARQL queries on summaries.
- We validate empirically SemCat on a subset of LargeRDFBench.

This paper is organized as follows. Section 2 summarizes related works. Section 3 details the approach of SemCat for building summaries and source selection. Section 4 presents our experimental results. Finally, conclusions and future work are outlined in Section 5.

2 Related Works

The source selection for a SPARQL query, finds the minimal number of relevant sources on the web of data to be contacted to execute the query. In this paper, we focus on RDF data hosted in SPARQL endpoints.

SPARQL 1.1 Federated queries [19] allow executing queries over different SPARQL endpoints. The *SERVICE* clause advises a federated query processor to execute a portion of SPARQL query against a remote SPARQL endpoint. In this case, the source selection is performed by users when authoring SPARQL queries.

Federated query engines [18, 6] suppose the existence of a catalog of SPARQL endpoints and achieves automatic source selection based on the catalog. In its

simplest form, this catalog is just a set of endpoints. Automatic source selection can be seen as the rewriting of a SPARQL query into SPARQL 1.1 federated query given a catalog of endpoints.

Different techniques for source selection are proposed [15]. We can distinguish two categories of source selection: catalog/index-free and catalog/index-assisted. In catalog/index-free source selection, the federated query engine performs source selection without using any stored data summaries, in contrast, in catalog/index-assisted source selection, the federated query engine uses data summaries that have been collected in a pre-processing step.

FedX [18] is a representative of index-free SPARQL federated query engine. The source selection relies completely on a simple catalog of URLs of SPARQL endpoints and SPARQL ASK query. For each triple pattern of a query, FedX sends ASK queries to all endpoints in the catalog and those that pass the SPARQL ASK test are selected. The source selection of FedX does not require to store any data summaries. However, it requires a lot of communication with the endpoints before to start the query execution. For instance, for a catalog of 100 endpoints and a query with 10 triple patterns, FedX sends 100*10=1000 ASK queries to compute source selection before starting the execution of the federated query.

Index-assisted federated query engines improve the source selection but require preprocessing. Different levels of detail of statistics of SPARQL endpoints are pre-computed and used by the federated query engine during query processing. Different techniques and formats are proposed for precomputed indexes. Some approaches require sources to compute and maintain statistics[6, 11], other do not [13, 16].

Both Odyssey[11] Splendid[6] require data sources to compute and share statistics. Odyssey uses sophisticated statistics for source selection. The statistics detail information about the data provided by remote endpoints and links between them. Splendid[6] relies on VOID statistics and ASK queries to perform source selection. The statistics of VOID description are aggregated in a local index. This index is used to map triple patterns with bound predicate to relevant data sources. A SPARQL ASK query is used when the subject or object of the triple pattern is bound or unbound predicate. WoDQA [1] is another federated query engine that uses VOID, more precisely, it uses linksets description to retrieve relevant datasources.

Relaying on statistics precomputed by data sources appears as promising solution for source selection. However, recent studies [8] observed that only third of public SPARQL endpoints give static description of their content using VOID vocabularies, and even when they are provided, it is unclear the level of details. To mitigate burden on publishers to provide description of their endpoints', Sportal [8] proposes to compute VOID descriptions directly from the endpoints. Sportal defines sources self-description queries that allow to compute VOID. Unfortunately, as highlighted in [8], Sportal naturally inherits the limitations of SPARQL endpoints, many endpoints do not return results or produce incomplete results.

DARQ [13] and HiBISCuS[16] pre-compute the indexes directly from data in the endpoints. DARQ uses an index known as service description for source selection. The description includes hand crafted source description similar to Vocabulary of Linked Datasets VOID [2]. DARQ source selection matchs the predicates in the query against the predicates in the service description. HbiBISCuS [16] proposes a join-aware source selection algorithm. HbiBISCuS relies on the authority fragments of resources URIs to estimate whether combing data of multiple sources can lead to any join results. For unbound predicates or common predicates, an ASK query is used for sources selection.

Relaying on statistics precomputed by federated query engines seem affordable, however, existing federated query engines propose different level of information and different formats for storing the indexes. Moreover, the index is locked up by the federated query engine and cannot be shared with other federated query engines.

Different summary approaches for source selection with different accuracy and different granularity have been proposed [21, 5]. Some summaries are approximative such as QTree, others are exact such as schema-level indexing.

Overall, many techniques have been proposed to build catalogs for source selection, but currently, there is no reliable method to build and maintain such catalogs over the web. In SemCat, we foster on web preemtion[10] that allows SPARQL queries to terminate. Thanks to preemptable servers, it is possible to compute aggregations as proposed in [7]. This allows to rely on endpoints to compute summaries and transfer only summaries. Using the same summary functions, a query Q over endpoints can be rewritten as a query Q_s, to be executed on SemCat. The provenance of the mappings of Q_s contains the source selection of Q.

3 The SemCat approach

A source selection service for the linked data has the same objective than a keyword search service in the web of documents, i.e. ensuring the findability of sources over a decentralized web of data. The general use-case is the following:

1. A user aims to execute the SPARQL query $Q1$ of the figure 1a over the whole set of SPARQL endpoints with no prior knowledge.
2. She loads a SPARQL 1.1 query engine in her web browser, as proposed by Communica [20] or SaGe [10] , and launchs the execution.
3. The query engine contacts a source selection service that returns the set of endpoints to contact per triple pattern. The query engine is free to refine the source selection, optimize the query and finally rewrites the original query into the SPARQL 1.1 federated query $Q2$ as in Figure 1a.
4. Finally, the web browser starts the execution of $Q2$ and returns complete results.

```
SELECT ?p ?o WHERE {
?s <sameas>
  <http://dba.org/b_obama>. #tp1
?s ?p ?o                    #tp2
}
```

(a) $Q1$

```
SELECT ?p ?o WHERE {
SERVICE <dba.org>
?s <sameas> <http://dba.org/b_obama> }.
SERVICE <wda.com> {?s ?p ?o}.
SERVICE <nyt.com> {?s ?p ?o}.
}
```

(b) $Q2$

Fig. 1: SPARQL queries Q_1 and Q_2

Such scenario raises many issues: how to build this source selection service? How to contact it? Is the source selection optimal? What is the execution time of the source selection? The objective of SemCat is to answer the above questions and makes the above use-case possible.

3.1 Preliminaries

We consider three disjoint sets I (IRIs), L (literals) and B (blank nodes) and denote the set T of RDF terms $I \cup L \cup B$.

An RDF triple $(s\,p\,o) \in (I \cup B) \times I \times T$ connects subject s through predicate p to object o. *An RDF graph* G is a finite set of RDF triples. We denote $Val(G)$ the set of all values (IRI, blank nodes and literals) in G. A mapping μ from V to T is a partial function $\mu : V \to T$, the domain of μ, denoted $dom(\mu)$ is the subset of V where μ is defined. Mappings μ_1 and μ_2 are compatible on the variable $?x$, written $\mu_1(?x) \sim \mu_2(?x)$ if $\mu_1(?x) = \mu_2(?x)$ and $?x \in dom(\mu_1) \cap dom(\mu_2)$.

RDF graphs are published in the web following the Linked data principles [3]. These RDF graphs could be accessible through public SPARQL endpoints. A SPARQL endpoint is defined as a couple (E_i, G_i), where E_i is the URL of the endpoint and G_i is the RDF graph accessible by E_i. A SPARQL query has the form $Head \leftarrow Body$ where $Head$ is an expression indicates how to construct the answer of the body and the body is a complex RDF graph pattern expression. Assume the existence of an infinite set V of variables, disjoint with previous sets. A SPARQL graph pattern expression P is defined recursively as follows [9, 12, 17].

1. A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a triple pattern.
2. If $P1$ and $P2$ are graph patterns, then expressions (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are graph patterns (a conjunction graph pattern, an optional graph pattern, and a union graph pattern, respectively).
3. If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern (a filter graph pattern).

The evaluation of a graph pattern P over a SPARQL endpoint E_i denoted by $[\![P]\!]_{G_i}$ returns a set of mappings, called *result set*. Each element of the result of a query is a set of *variable bindings*.

We define a *federation of SPARQL endpoint* $F(E, G)$ as a set of couple of $F(E, G) = \{(E_1, G_1), \ldots, (E_n, G_n)\}$ where $E = \{E_1, \ldots, E_n\}$ and $G = \bigcup_{i=1}^{n} G_i$, respectively. For the sake of simplicity, we consider the RDF graphs of endpoints

do not have blank nodes. We consider *federated SPARQL queries* as queries that are defined over a federation of SPARQL endpoints. Given a SPARQL query Q, a data source $E_i \in E$ is said *contribute* to the query Q if at least one of the variable bindings in the result set of Q can be found in E_i.

3.2 SemCat definitions and problem description

In this paper, we focus on federation of SPARQL endpoints $F(E, G)$. Discovering and maintaining E is out of the scope of the paper. For instance, E could be discovered and maintained by crawling the web as proposed by Google Datasearch[4]. We define source selection as :

Definition 1 (Source Selection). *Given a query Q, a source selection of Q over F is the set of sources $E_{tp_i} \subseteq E$ per triple pattern of Q that potentially contributes to the result set of Q.*

Ideally, a source selection provides the minimal set of endpoints to contact in order to produce the complet results of the query. We formalise the problem of source selection service as follows:

Definition 2 (Source Selection Service Problem (SSS-P)). *A source selection service S is a SPARQL service hosting an RDF graph SC extracted from G. Given a query Q, S computes the source selection of Q over G by evaluating a rewriting of Q over SC.*

Building a source selection service raises critical challenges:

SC **is constructed thanks to SPARQL queries,** following the idea proposed in Sportal [8]. Computing summaries based on SPARQL queries overcomes the poor adoption of service description conventions and the impracticability of dumps practices for web automation.

SC **is a summary of** *G* . We expect the data transfer from E to SC to be proportional to the size of the summary of graphs and not to the original size of graphs, e.g. extracting a summary of 1000 triples from DBpedia should transfer ideally 1000 triples.

Source selection is minimal, sound and complete. Source selection returns, as possible, the minimal sources per triple pattern. According to the "accuracy" of summary, source selection could be overestimated, i.e. contain false positives. However, it should always produce sound and complete answer.

Source selection time complexity. The complexity of the source selection should be proportional to number of source selected per triple pattern.

3.3 Building SemCat Summaries

We rely on structural graph summarization to define SC. Structural graph summarization is essentially a reduced version of the original RDF graphs where nodes have been merged according to some notion of structural similarity [5]. Consequently, we consider summaries defined as an RDF graph homomorphism.

dba
<dba/b_obama> <isa> <dba/president>
<dba/b_obama> <sameas> <wda/barack_o>

nyt
<nyt/ba> <sameas> <dba/b_obama>
<nyt/ba> <said> "hello"

wda
<wda/barack_o> <isa> <wda/person>
<wda/barack_o> <birth> "1967"

$\boldsymbol{SC(\psi_p, \{dba, nyt, wda\})}$
<s1> <isa> <o1> <dba>
<s1> <isa> <o1> <wda>
<s1> <sameas> <o1> <dba>
<s1> <sameas> <o1> <nyt>
<s1> <said> <o1> <nyt>
<s1> <birth> <o1> <wda>

$\boldsymbol{SC(\psi_h, \{dba, nyt, wda\})}$
<dba> <isa> <dba> <dba>
<wda> <isa> <wda> <wda>
<dba> <sameas> <wda> <dba>
<nyt> <sameas> <dba> <nyt>
<nyt> <said> "lit" <nyt>
<wda> <birth> "lit" <wda>

$\boldsymbol{SC(\psi_s, \{dba, nyt, wda\})}$
<dba/ma> <isa> <dba/nt> <dba>
<wda/_o> <isa> <wda/on> <wda>
<dba/ma> <sameas> <wda/_o> <dba>
<nyt/ba> <sameas> <dba/ma> <nyt>
<nyt/ba> <said> "lit" <nyt>
<wda/_o> <birth> "lit" <wda>

$\boldsymbol{SC(\psi_1, \{dba, nyt, wda\})}$
<s1> <p1> <o1> <dba>
<s1> <p1> <o1> <nyt>
<s1> <p1> <o1> <wda>

Fig. 2: Three RDF Graphs hosted by dba,nyt and wda and their 4 summaries

Definition 3 (RDF graph homomorphism). *Let G, G' be two RDF graphs. A function $\psi : Val(G) \rightarrow Val(G')$ is a homomorphism from G to G' iff for every RDF triple $(s, p, o) \in G$ there is an RDF triple $(\psi(s), \psi(p), \psi(o)) \in G'$.*

A homomorphism from G to G' ensures that the graph structure present in G has an "image" in G'.

Definition 4 (SemCat summary). *Let $F(E, G)$ be a federation of SPARQL endpoints, the summary of E using ψ, $SC(\psi)$ is a set of quads such that:*

$$SC(\psi, E) = \{(\psi(s), \psi(p), \psi(o), g) | (s, p, o) \in G_i \, and \, E_i \in E\}$$

For a federation $F(E, G)$, we can define summaries with different "accuracy" using different ψ functions. Figure 2 describes summaries of three dummy graphs *dba*,*nyt*, and *wda*.

Identity summary: For this summary, ψ_{id} is defined as the identity function. $SC(\psi_{id}, E)$ is composed of all graphs in the federation. This is the most accurate summary, but unrealistic.

1-triple summary For this summary, ψ_1 is defined as:

$$(\psi_1(s), \psi_1(p), \psi_1(o)) = (: s1, : p1, : o1)$$

where $: s1$, $: p1$ and $: o1$ are Literals. A 1-triple summary $SC(\psi_1, E)$ contains a single triple per endpoint. $SC(\psi_1, E)$ represents the set of all endpoints in the federation, i.e. the catalog of the endpoints. The size of this summary is proportional to the number of endpoints.

predicate-aware summary: ψ_p is defined as:

$$(\psi_p(s), \psi_p(p), \psi_p(o)) = \begin{cases} (:s1, p, :s1) & \text{if } s, o \in I) \\ (:s1, p, "lit") & \text{if } s \in I, o \in L \end{cases}$$

This function projects all subjects and objects to two constants : $s1$ and "lit". $SC(\psi_p, E)$ is the set of all predicates of E. In the worst case, if all sources have all predicates, the size of this summary is proportional to the number of predicates ($\#predicates$) multiplied by the number of sources ($\#sources$).

authorities-aware summary: psi_h function is defined as:

$$(\psi_h(s), \psi_h(p), \psi_h(o)) = \begin{cases} (auth(s), p, auth(o)) & \text{if } s, o \in I \\ (auth(s), p, "lit") & \text{if } s \in I, o \in L \end{cases}$$

$Auth(s)$ is a function that returns the domain of an URI. ψ_h builds a summary inspired by the hibiscus summaries [16]. In the worst case, a predicate could have all authorities ($\#auth$) as subjects and as objects If all sources have such predicate then the size of the summary is $(\#predicates * \#auth^2) * \#sources$.

suffix-authority-aware summary: ψ_s function is defined as:

$$(\psi_s(s), \psi_s(p), \psi_s(o)) = \begin{cases} (cc(auth(s), lt(s,2)), p, cc(auth(o), lt(o,2))) & \text{if } s, o \in I \\ (cc(auth(s), lt(s,2)), p, "lit")) & \text{if } s \in I, o \in L \end{cases}$$

The function $lt(string, 2)$ returns the last 2 characters of the string and $cc()$ is the string concatenation function. This summary is an extension of ψ_h with suffixes of URIs. The summary is more accurate than psi_h summary, however, its size grows quickly. If we consider only 26 different letters, the numbers of nodes in the summary is now equal to $\#auth * 26^2$. Therefore, the size of the summary is bounded by $(\#predicates * (\#auth * 26^2)^2) * \#sources$. The summary is more accurate but it is much more bigger.
The different summaries behave as *Russian dolls*, for instance:

$$SC(\psi_h, SC(\psi_s, E)) = SC(\psi_h, E)$$

The extraction of the most accurate summary allows to build less accurate ones.

$$SC(\psi_1, E) \leftarrow SC(\psi_p, E) \leftarrow SC(\psi_h, E) \leftarrow SC(\psi_s, E) \leftarrow SC(\psi_{id}, E)$$

3.4 Source selection on SemCat summaries

Source selection for a query Q as defined in section 3.2 computes a set of endpoints to contact per triple pattern of Q.

Triple-pattern based source selection (TPSS) As a triple pattern of Q is defined on G, it cannot be executed directly on the summary to find the endpoints to contact. To make a triple pattern executable of a summary, we need

Table 1: Source selection for query Q_1 (Figure 1a) and ψ_h summary (Figure 2)

Q triple pattern	TP Source selection	BGP source selection
tp1	nyt	nyt
tp2	dba,wda, nyt	nyt

to extend ψ function to handle triple patterns, i.e. summarization functions are defined for RDF triples, they cannot be applied on variables of triple patterns.

In the following, we extend the definition of function ψ_h to handle triple patterns. As a triple pattern can have one the the following forms [21], where ? denotes variables: ($?s_1$ $?p_1$ $?o_1$), (s_1 $?p_1$ $?o_1$), ($?s_1$ p_1 $?o_1$), ($?s_1$ $?p_1$ o_1), (s_1 p_1 $?o_1$), (s_1 $?p_1$ o_1),($?s_1$ p_1 o_1), (s_1 p_1 o_1). We distinguish the following cases:

$$\psi_h(s), \psi_h(p), \psi_h(o)) = \begin{cases} s, p, o & \text{if } s, p, o \in V \\ auth(s), p, o & \text{if } s \in I, o \in V \\ auth(s), p, auth(o) & \text{if } s, o \in I \\ auth(s), p, "lit" & \text{if } s \in I, o \in L \\ s, p, auth(o) & \text{if } s \in V, o \in I \\ s, p, "lit" & \text{if } s \in V, o \in L \end{cases}$$

We define the source selection query for a triple pattern as:

Definition 5 (TPSS query). *The source selection query for a triple pattern (s, p, o) on a summary $SC(\psi, E)$ is defined by the query:*

$$SS(\psi, (s, p, o)) = \pi_g(\psi(s), \psi(p), \psi(o), g) \, and \, g \in V$$

By abusing of notations, we consider $\pi_x(q)$ as the projection operator on variable x of a quad q. $[\![SS(\psi, (s, p, o))]\!]_{SC(\psi,E)}$ returns the source selection of (s, p, o). Table 1 presents the the results of TPSS of query $Q1$ defined in (Figure 1a).

The source selection has different time complexity according to the form of the triple pattern, and the summary.

For (?s ?p ?o), $SS(\psi_1(?s, ?p, ?o))$ returns results in a time complexity proportional to the number of selected sources, i.e. $O(|SS(?s, ?p, ?o)|)$.

For (?s p ?o), $SS(\psi_p(?s, p, ?o))$ returns results in $O(|SS((?s, p, ?o)|)$.

For (s p ?o), $SS(\psi_h(s, p, ?o))$ returns the results in the worst case in $O(\#auth * \#sources)$.

For (s p ?o), $SS(\psi_s(s, p, ?o))$ maybe return a more accurate selection but, in the worst case, in $O((\#auth * 26^2) * \#sources)$.

It possible to choose among different summaries according to the form of triple patterns. Summaries that handle constants such ψ_h or ψ_s cannot answer in $O(|SS((s, p, o)|)$, but will depend of the selectivity of the constants.

The TPSS overestimates the number of sources to contact because it is not aware of the join variables, i.e.variables shared among triple patterns of the query. For instance, for the triple $\#tp2$ ($?s?p?o$) of query Q_1 (cf. figure 1a), TPSS selects all the sources present in the summary (Table 1) even if a subset of sources really contribute to the results of the query.

BGP-Based source selection query (BGPSS) For the sake of simplicity, we focus on queries with conjunctive graph patterns (BGP queries). However, any SPARQL query with union graph patterns, optional graph patterns, etc can be rewritten following the same approach. A BGP query is a set of triple patterns.

Definition 6 (BGPSS Query).
Let Q a BGP query a set of triple patterns (s_i, p_i, o_i), the source selection query is defined as:

$$SS(\psi, Q) = \pi_{g_i} \bowtie_{(s_i,p_i,o_i)\in Q} (\psi(s_i), \psi(p_i), \psi(o_i), g_i), g_i \in V$$

The evaluation of $SS(\psi, Q)$ over $SC(\psi, E)$ returns for all triple patterns tp_i of Q their respective selected sources in g_i. As the homomorphic query of the original query is executed on the summary, the source selection is optimal for that summary.

Following this definition, the query $Q1$ with ψ_h can be rewritten as:

SELECT DISTINCT ?g1, ?g2 **WHERE** {
graph ?g1 {?s <sameas> <http://dba.org>}
graph ?g2 {?s ?p ?o}
}

Then the execution of this query on the ψ_h summary of the figure 2, return the result of described in Table 1. As we can see, the BGP-aware source selection is more accurate than the previous TP-based source selection.

Therefore, executing $SS(\psi_{id}, Q)$ on $SC(\psi_{id}, E)$ returns the exact source selection for Q. executing $SS(\psi_1, Q)$ on $SC(\psi_1, E)$ returns all sources for all triple patterns of Q.

The BGPSS requires to execute all triple patterns of the query on the same summary because mappings of join variables are shared among the triple patterns. This is not the case of TPSS where different summaries can be used. In the worst case, each triple pattern scan the whole summary, so complexity is the number of triple pattern multiplied by the size of the summary. For example, executing a source selection on ψ_h is now in the worst case in $(\#predicates * \#auth^2) * \#sources$ multiplied by the number of triple patterns in the query. However, in an average case, for a bounded authority and a bounded predicate in a triple pattern, the worst time complexity is $\#auth * \#source$ which is much more tractable. Concretely, there is a trade-off between the accuracy of the source selection and the time for source selection. For example, for the BGP query $Q1$ of figure 1a, executing the source selection query for $Q1$ on the ψ_s summary returns a better source selection than executing the source selection for $Q1$ on the ψ_h. However, the source selection query for Q_1 on the ψ_h, generally, returns the source selection much faster than with ψ_s.

3.5 Implementing summaries with web preemption

We can implement ψ functions as a SPARQL 1.1 query. For example, computing authorities-aware summary ψ_h can be done by executing the following SPARQL query over the SPARQL endpoints of the federation.

```
CONSTRUCT { ?ps ?p ?po } where { ?s ?p ?o
 FILTER  isiri(?s)
 BIND(URI(REPLACE(STR(?s),
    "^(https?://?.*?)/.*", "$1")) AS ?ps)
 BIND( if (isiri (?o),URI(REPLACE(STR(?o),
    "^(https?://?.*?)/.*", "$1")),"lit") as ?po)}
```

However, executing this query over an endpoint is challenging:

- A public SPARQL endpoint will interrupt the query after a time quota as reported in[8].
- A TPF server [23] or SaGe [10] server return complete results for queries, but FILTERS, BIND and CONSTRUCT operators are executed on client side. Consequently, the query execution first transfers all mappings for the $(?s, ?p, ?o)$ triple pattern from the server to the client, then the summary is computed on client-side. This clearly require to transfer all the RDF graphs of the federation to compute a summary.

An affordable solution is to follow the approach of SaGe-agg [7] to implement this query without interruption and low data transfer. In SaGe-agg, the SaGEserver is able to compute partial aggregates per quantum, thanks to the decomposability property of aggregate functions.

We extended the SaGe server to handle BIND operation to express summary functions in SPARQL. We extended also the SaGe server to handle CONSTRUCT per quantum, i.e. a graph is constructed during one quantum and transferred to client at the end of the quantum. As BIND statements summarize subjects and objects, most of the element of the graph are likely to be duplicates. Consequently, the compression of the graph is mostly done on server-side and data transfer is dramatically reduced, as demonstrated in the experimental study.

If all the summary query can be processed in one quantum, then the transfer is optimal. If not, some triples can be transferred several times from the server to the client. This is an overhead intrinsic to the web preemption approach.

This overhead mainly depends on the summary function and the order of scanned triples. To illustrate, suppose we are computing the ψ_h summary. It is important to scan triples following a PSO index or POS index. As triples are ordered by predicate, followed by subject or object, it is very likely that all duplicates are eliminated during the same quantum. The following table illustrates this processus for POS index:

birthyear 1967 http://dbp/Bob	$\xrightarrow{\psi_h}$	birthyear lit http://dbp
birthyear 1967 http://dbp/Alice		birthyear lit http://dbp

The result of the CONSTRUCT is only one triple:*birthyear lit http://dbp*.

The ψ_s summary is less likely to remove all duplicates in a quantum. SaGe should provide low overhead for authorities-aware summaries.

4 Experimental Study

We want to empirically answer the following questions: (i) What is the data transfer and execution time of computing different kind of summaries on online SPARQL servers? (ii) How good is the source selection for TP-based and BGP-based source selection? (iii) What is the execution time of the source selection for a source selection service?

We extended the SaGe server to support the execution of summaries functions. The SaGe server now supports CONSTRUCT, REDUCED keyword, BIND operations and custom functions for efficient computation. All extensions and experimental results are available at https://github.com/momo54/semcat.

Dataset and Queries: We consider a workload (SP) of 14 SPARQL queries extracted from the LargeRDF Benchmark [14]. These queries run on the 9 datasets presented in figure 3a (orig).

Summaries and source selection We compare the performances of the TPSS and BGPSS services on ψ_p, ψ_h and ψ_s summaries, named respectively $void$, hib and suf. In the experimentations, the TPSS engine uses the same summary for all triples of the query.

Server configuration: We run experimentations on personal computer 4 GHz Intel Core i5 four CPU, 8 Go 2133 MHz LPDDR3.

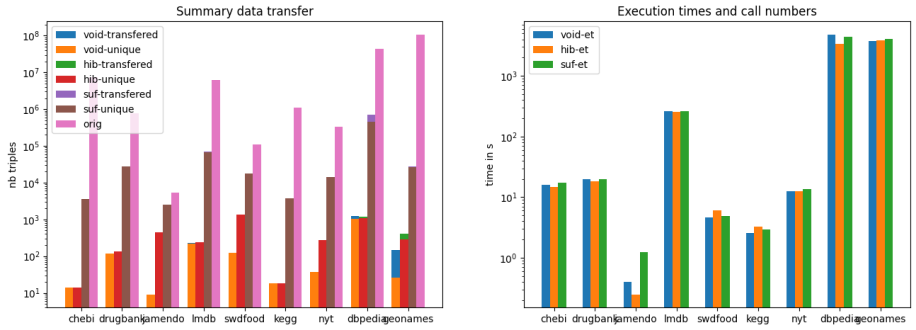
Evaluation Metrics: (i) *Summary Data transfer*: is the number of triple transferred from a SPARQL server to SemCat to compute a summary. (ii) *Summary Execution time*: is the time required by SemCat to compute the summary per a SPARQL server. (iii) *Summary size*: is the number of triples in the summary per graph. (iv) SSQ: is the sum of sources selected per query. For example, if a query has two triple patterns tp1 and tp2 and the source selection for tp1 is $s1$ and the source selection for tp2 is $s1, s2$, then SSQ is 3. (v) SSt: is the source selection time, i.e. time to perform the source selection of a query.

4.1 Building ψ summaries

For this experiment, we setup a SaGe server configured with a quantum of 60s. We ingested the nine datasets and we executed the different summary functions as SPARQL queries on the server. We measured the data transfer and the execution time as shown in Figure 3.

Figure 3a presents the number of triples in the summary (unique), the number of triples retrieved to compute this summary (transferred) and the original size of the graph ($orig$). For most endpoints the data transfer is optimal, in one quantum, the SaGe server is able to scan all the graph and return the summary. For large graphs, several quantums are necessary and duplicates appear for DBpedia and geonames. However, the overheads remains marginal.

Figure 3b presents the time required to compute the summary. We observe that the time remains slightly the same whatever the summary. This is normal because computing the summary requires to scan the complete graph and the scan speed remains the same whatever the summary function.



(a) Summary data transfer and summary size

(b) Summary execution time

Fig. 3: Results for a federation of nine SPARQL endpoints using *void*, *hib* and *suf* summaries. *orig* is the size of the original RDF graph

4.2 The number of sources selected per query (SSQ)

Figures 4a, 4b present, respectively, the selected sources by BGPSS engine and TPSS engines. As expected, for both source selection engines, a more accurate summary improves the accuracy of the source selection, i.e. produces less SSQ. For instance, the *suf* summary (ψ_s), returns the best results. The *void* summary (ψ_p) makes no difference between engines. For *hib* and *suf*, the BGPSS engine improves the source selection compared to TPSS engine. For instance, for $S2$ using the *hib* summary SSQ is 7 with TPSS and pruned to 3 with BGPSS, for $S4$ is pruned from 20 to 5. In total, the SSQ for all queries is improved with BGPSS engine except for S14 and S11. The SSQ of S8 remains unchanged.

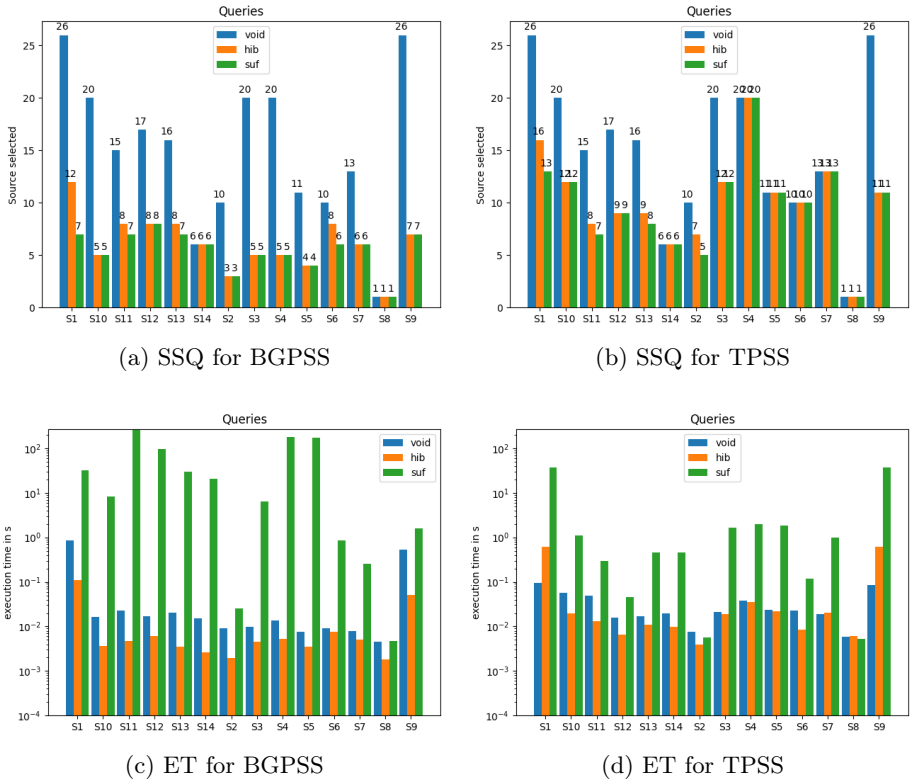
The *suf* summary with BGPSS, as *hib* with BGPSS, improves most of the queries SSQ. Compared to *hib* with BGPSS only the SSQ of 4 queries $S1, S11, S13$ and $S6$ is improved. Overall, the *suf* summary with the BGPSS clearly dominates the source selection accuracy.

4.3 Execution time Source selection

Figues 4a and 4b present, respectively, the execution time of BGPSS engine and TPSS engine. We run the experiment with RDFLib, Virtuoso (without quota) and SaGe. For space limitations, only the execution time obtained with Virtuoso is presented. All the results are available at *anonymized*.

As in previous experimentations, TPSS uses the same summary for all triple patterns of the query, i.e. do not choose the summary according the characteristic of the triple pattern.

For both source selection engines, the *suf* summary is clearly more expensive than the others. This is normal as the size of the *suf* summary impacts significantly the evaluation of any triple pattern, especially the $(?s, ?p?o)$ triple



(a) SSQ for BGPSS

(b) SSQ for TPSS

(c) ET for BGPSS

(d) ET for TPSS

Fig. 4: Source selection and execution times

pattern. Concerning the *void* summary, the execution time is better with TPSS engine than with BGP engine. Joins with the *void* summary are just useless and slow down the execution. Concerning the *hib* summary, the execution time with the BGPSS engine is better than suf. Using the authority makes the joins selective. But when used in the suf summary, the performances are degrading quickly. The suf summary creates a dense graph that negatively impacts performances of joins.

Overall, considering the accuracy of the source selection and the execution time, the *hib* summary (ψ_s) delivers a good trade-off.

5 Conclusions

In this paper, we highlighted the need for a source selection service to make endpoints findable. Such service requires web automation for its creation and maintenance. Thanks to web preemption, we demonstrated how to query endpoints and collect efficiently summaries just relying on SPARQL queries. We

define different summary functions. We presented how a query Q on endpoints can be rewritten as a query Q' on summaries that returns the source selection of Q on endpoints. If all endpoints support web preemption, then any federated query terminates and delivers complete results.

We empirically demonstrated the different trade-off between the accuracy of the source selection, the execution time of the source selection and the size of the summary. An interesting conjecture could be that *one dimension has to be sacrificed to preserve the others.*

This approach raises several perspectives. First, it is clearly an open garden, i.e. it exists other summary functions and certainly many different way to combine summaries. We can imagine a BGP-based source selection combining different summaries. Bindings obtained on one summary can be transformed to be injected into other summaries. Second, in this paper we focused on source selection, the next step is to extend the summary functions to also collect statistics for join ordering.

References

1. Ziya Akar, Tayfun Gökmen Halaç, Erdem Eser Ekinci, and Oguz Dikenelli. Querying the web of interlinked datasets using void descriptions. *LDOW*, 937, 2012.
2. Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets with the void vocabulary. 2011.
3. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
4. Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, pages 1365–1375, 2019.
5. Sejla Cebiric, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. Summarizing Semantic Graphs: A Survey. *The VLDB Journal*, 28(3), June 2019.
6. Olaf Görlitz and Steffen Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *Proceedings of the Second International Conference on Consuming Linked Data*, volume 782, pages 13–24. CEUR-WS. org, 2011.
7. Arnaud Grall, Thomas Minier, Hala Skaf-Molli, and Pascal Molli. Processing SPARQL Aggregate Queries with Web Preemption. In *17th Extended Semantic Web Conference (ESWC 2020)*, The Semantic Web: ESWC 2020, Herkalion, Greece, June 2020. Springer, Cham.
8. Ali Hasnain, Qaiser Mehmood, and Syeda Sana e Zainab ang Aidan Hogan. SPORTAL: profiling the content of public SPARQL endpoints. *Int. J. Semantic Web Inf. Syst.*, 12(3):134–163, 2016.
9. Mark Kaminski, Egor V. Kostylev, and Bernardo Cuenca Grau. Query nesting, assignment, and aggregation in SPARQL 1.1. *ACM Trans. Database Syst.*, 42(3):1–46, August 2017.
10. Thomas Minier, Hala Skaf-Molli, and Pascal Molli. Sage: Web preemption for public SPARQL query services. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 1268–1278, 2019.
11. Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. The odyssey approach for optimizing federated sparql queries. In *International Semantic Web Conference*, pages 471–489. Springer, 2017.

12. Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. Semantics and complexity of SPARQL. *ACM Transations on Database Systems*, 34(3):16:1–16:45, 2009.
13. Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In Sean Bechhofer et al., editors, *ESWC 2008*, volume 5021 of *LNCS*, pages 524–538. Springer, 2008.
14. Muhammad Saleem, Ali Hasnain, and Axel-Cyrille Ngonga Ngomo. Largerdfbench: a billion triples benchmark for sparql endpoint federation. *Journal of Web Semantics*, 48:85–125, 2018.
15. Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga Ngomo. A fine-grained evaluation of sparql endpoint federation systems. *Semantic Web*, 7(5):493–518, 2016.
16. Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In *European semantic web conference*, pages 176–191. Springer, 2014.
17. Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *Database Theory - ICDT 2010*, pages 4–33, 2010.
18. Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International semantic web conference*, pages 601–616. Springer, 2011.
19. Harris Steve and Seaborne Andy. SPARQL 1.1 query language. In *Recommendation W3C*, 2013.
20. Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: A modular SPARQL query engine for the web. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*, volume 11137 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2018.
21. Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. Comparing data summaries for processing live queries over linked data. *World Wide Web*, 14(5-6):495–544, 2011.
22. Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil Aranda. SPARQLES: monitoring public SPARQL endpoints. *Semantic Web*, 8(6):1049–1065, 2017.
23. Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *J. Web Sem.*, 37-38:184–206, 2016.