Source Selection for SPARQL Endpoints: Fit for Heterogeneous Federations of RDF Data Sources?

Sijin Cheng[1], Olaf Hartig[1]

[1]Dept. of Computer and Information Science (IDA), Linköping University, Linköping, Sweden

Abstract

To answer queries over a federation of multiple data sources, a preliminary task is source selection; i.e., identify the federation members that can answer each part of the query and decompose the query into subqueries assigned to each federation member. Existing source selection approaches for federations of RDF data sources have been developed based on the assumption that the federation member are SPARQL endpoints. This paper presents an analytical study that investigates whether these approaches are still effective in the context of federations that are heterogeneous in terms of the types of data access interface. In particular, we identify what information about the data of the federation members is required by the approaches and analyze the possibilities and the effort of obtaining this information via the different types of data access interfaces. We find that almost all existing source selection approaches can be adopted for heterogeneous federations but obtaining the required information may not be practical.

1. Introduction

Processing queries that cannot be answered by a single data source requires combining data from multiple data sources in a federation. For such federated queries, it is significant to identify relevant federation members for executing particular parts of a given query, since requesting data from irrelevant federation members is an unnecessary overhead or may result in high costs on subsequent query processing. The task of source selection is to identify relevant federation members that may answer parts of the query and to determine the subqueries to be executed at each federation member. Existing source selection approaches have, by and large, focused only on the source selection problem in federations that are homogeneous in terms of the interfaces provided by the federation members; that is, the federation members have typically be assumed to be SPARQL endpoints. For federations that are heterogeneous in terms of interface types (i.e., each federation member may support a different type of interface to access its RDF dataset), the source selection problem has yet to be studied.

As a first step in this direction, this paper focuses on the existing source selection approaches for homogeneous federations and investigates whether these approaches are still applicable in the context of heterogeneous federations. The main question that we aim to answer is whether the information that the approaches assume to have about the federation members can still be obtained even if the federation members provide interfaces different from the SPARQL endpoint

sijin.cheng@liu.se (S. Cheng); olaf.hartig@liu.se (O. Hartig)
https://www.ida.liu.se/~sijch63/ (S. Cheng); http://olafhartig.de/ (O. Hartig)
0000-0003-4363-0654 (S. Cheng); 0000-0002-1741-2090 (O. Hartig)

CEUR Workshop Proceedings (CEUR-WS.org)

interface. Additionally, we aim to achieve a general understanding of the effort of obtaining this information when relying on the different types of interfaces.

As a preliminary for the analysis, we first provide an overview of the types of interfaces that we consider in this study; cf. Section 2. Thereafter, we present our analysis in Section 3 and summarize our findings in Section 4.

2. Types of Interfaces

This section provides an overview of the different types of interfaces for accessing RDF datasets that have been proposed in the literature. For each interface, we focus on the types of requests that clients can send to the server and the possible metadata that the server needs to include in its responses (if any).

SPARQL endpoints [1]. The clients can request the result of any SPARQL query over the RDF dataset hosted at the server. The responses to such requests contain only the corresponding query result and no additional metadata. However, SPARQL queries can be constructed to also retrieve some forms of metadata and statistics about the dataset from a SPARQL endpoint.

Triple Pattern Fragments (TPF) [2]. The TPF interface restricts the type of queries that can be answered by the server to single triple patterns. More specifically, clients can request all triples from the server-side dataset that match a given triple pattern. Since some triple patterns may have many matching triples, TPF servers use paging. That is, they return only a limited number of matching triples per the response, together with metadata that provides a URI via which the next subset of matching triples can be requested. Each such response is called a *page* and the typical limit that TPF servers apply is a maximum of 100 matching triples per page [2]. In addition to paging-related metadata, the specification of the TPF interface requires the server to include further metadata in the responses; in particular, an estimate of the total number of triples that match the given triple pattern must be included as metadata in each page.

Bindings-Restricted TPF (brTPF) [3]. The brTPF interface is an extension of the TPF interface and has been proposed to mitigate the shortcomings of TPF (such as a large number of requests and unnecessary data transfer). The extension of brTPF over TPF is that it enables clients to include a set of solution mappings in their requests (in addition to the triple pattern). The response to such a brTPF request contains RDF triples from the underlying dataset that not only match the given triple pattern but are also guaranteed to contribute to a join with the given set of solution mappings (which would typically be an intermediate result produced for other triple patterns of the query evaluated by a brTPF client). As for TPF, brTPF servers may use paging and the responses to brTPF requests must also include metadata with cardinality estimates. Additionally, brTPF servers are required to be able to respond to TPF requests.

Star Pattern Fragments (SPF) [4]. The SPF interface presents another extension over TPF and brTPF. Instead of a single triple pattern, SPF requests may contain a set of triple patterns that form a so-called star pattern; that is, the triple patterns all have the same subject. As with brTPF, SPF requests may additionally include a set of solution mappings. SPF servers may use paging and the responses must include metadata with cardinality estimates.

smart-KG [5]. The smart-KG interface combines TPF with shipping of whole partitions of the server-side RDF dataset to the clients in order to reduce redundant data transfer between servers and clients. Specifically, upon set up, the server partitions the triples of the dataset based on characteristic sets [6] and compresses the partitions using HDT [7]. Then, clients can request the partitions in their compressed form, cache them, and query them locally for triples that match given triple patterns. To facilitate the interaction between smart-KG clients and servers, each client is equipped with a server-provided catalog that contains metadata about the structure of the partitions. The metadata in these catalogs that is most relevant for our analysis in this paper is a list of all URIs that appear as predicates in the triples of the server-side dataset.

WiseKG [8]. WiseKG dynamically balances the query processing load between client and server by combining SPF and smart-KG. That is, for each star-shaped subquery, the WiseKG server chooses either the SPF or the smart-KG approach by comparing estimated costs for both approaches. The cost model used to this end considers the current server load, client capabilities, an estimation of necessary data transfer between client and server (for intermediate query results), and network bandwidth. Since WiseKG is a combination of SPF and smart-KG, we do not consider it explicitly in the rest of this paper, but the observations that we make for both SPF and smart-KG also apply to WiseKG.

SaGe [9, 10, 11, 12]. SaGe applies a preemption method in order to provide stable and responsive SPARQL services. That is, clients can request the execution of a SPARQL query over the server-side dataset and the server performs the execution for a fixed time quantum; when that time is exceeded, the server suspends the execution and encodes the current state of the execution in a response to the client, together with any solutions of the query result that may have been produced already. By returning the encoded state to the server, the client can then request the server to resume the execution for another time quantum. This process of the client requesting the server to make step-wise progress on the query execution continues until either the query execution completes or the client decides to discontinue. It is important to note that only a fragment of SPARQL can be implemented using preemptable query operators and, thus, only queries from this fragment can be requested to be executed by a SaGe server. The original work presented preemptable operators for joins, unions, projections (i.e., SELECT), and filters with pure logical expressions [9]. Follow-up work introduced further preemptable operators for aggregations (COUNT, AVG, SUM, MIN, MAX) and aggregations with DISTINCT [10], as well as property path patterns [11]. While some of these preemptable operators can be executed completely by the server, others have a client-side component that produces the result in collaboration with the server-side component of the operator. Moreover, for queries that use features of SPARQL for which no preemptable operators exist (e.g., ORDER BY, GROUP BY, MINUS, FILTER EXISTS), these features need to be implemented completely in the client.

3. Analysis of Existing Source Selection Approaches

In a federation of RDF data sources, the maintainer of such a data source may choose any of the aforementioned types of interfaces to provide access to the dataset of their federation member. In this section, we analyze existing source selection approaches in terms of their applicability to federations that are heterogeneous in terms of interface types. Specifically,

```
ASK {
  ?book rdf:type dbo:Book .
}
```

(a)

```
SELECT * WHERE {
  ?book rdf:type dbo:Book .
} LIMIT 1
```

(b)

Figure 1: Example check queries for triple pattern-wise source selection for (a) SPARQL endpoints and (b) SaGe servers.

we examine what information about the datasets of federation members is required for each approach, and then analyze whether and how it is possible to collect that information based on the different types of data access interfaces. We first focus on source selection approaches that do not require any prior information about the datasets of the federation members (Sections 3.1–3.3). Thereafter, we examine source selection approaches that use a data summary or index for federation members together with runtime requests (Sections 3.4–3.6). In addition, we also investigate source selection approaches that rely solely on a pre-computed data catalog or index for federation members (Sections 3.7 and 3.8). Finally, we consider approaches that require federation members to compute and share statistics (Sections 3.9 and 3.10) and discuss alternative options if the required information is not provided by federation members.

3.1. Source Selection in FedX

FedX [13] uses a source selection approach that does not require any prior information about the data of the federation members before performing any query. Instead, FedX obtains the information it needs for its source selection approach from each federation member at query runtime. The particular information that FedX requires for each federation member is:

Info1: For each triple pattern of the given query, determine whether the federation member contains at least one triple that matches the triple pattern.

To obtain this information for **SPARQL endpoints** (for which FedX was designed), FedX iterates over the triple patterns of the given query and, for each triple pattern, issues a SPARQL ASK query with that triple pattern. Every federation member that returns TRUE in response to such a query is identified as a federation member containing triples that match the triple pattern.

For federation members using the **SaGe interface**, checking whether such federation members contain matching triples for a triple pattern can be done by requesting partial results using SPARQL SELECT queries with a LIMIT clause [8]. Figure 1 lists such check queries for an example triple pattern against a SPARQL endpoint and a SaGe interface, respectively.

For federation members with a **TPF interface**, a **brTPF interface**, or an **SPF interface**, using ASK queries or LIMIT clauses is not possible. The required information (Info1) can instead be obtained by requesting the first page of matching triples for each triple pattern of the query. The count metadata in the retrieved pages indicates whether the federation member has matching triples for the corresponding triple pattern.

Finally, considering federation members that use a **smart-KG interface**, recall that smart-KG clients are provided with a catalog that contains metadata about the structure of the compressed

dataset partitions available at the smart-KG server. This catalog can be leveraged to obtain the source selection information (Info1) required by the FedX approach. That is, for every triple pattern in which the predicate is bound, and subject and object are not, the corresponding predicate set of this catalog can be checked to determine whether the smart-KG server contains triples that match the triple pattern. This check does not even require contacting the smart-KG server. For any other triple pattern, a TPF first-page request can be used as suggested above for the TPF interface (because the smart-KG interface is an extension of the TPF interface).

3.2. Source Selection in Lusail

Lusail [14] applies another source selection approach that does not rely on prior information of the federation members and their datasets. The particular information that Lusail requires for each federation member is the same as required by FedX:

Info1 (same as in Section 3.1)**:** For each triple pattern of the given query, determine whether the federation member contains at least one triple that matches the triple pattern.

Additionally, Lusail aims to determine whether two or more triple patterns can be grouped as a subquery. To this end, Lusail requires the following information for each federation member.

Info2: For each pair of triple patterns in the query that share a variable, information whether the set difference is empty between

(i) the set of all RDF terms at the position of the shared variable within triples that match the first triple pattern and

(ii) the set of all RDF terms at the position of the shared variable within triples that match the second triple pattern.

Since the options to obtain Info1 via the different types of interfaces have been discussed already in Section 3.1, the following discussion focuses on Info2.

For federation members that provide a **SPARQL endpoint**, the set difference check can be performed by issuing a SPARQL SELECT query with FILTER NOT EXISTS and LIMIT 1 [14] (e.g., Figure 2), since it is only essential to know if the result is an empty set. Another option would be to use ASK instead of SELECT, in which case LIMIT 1 can be omitted.

For federation members with a **SaGe interface**, the set differences cannot be checked directly based on preemptable SPARQL queries because SaGe servers can only process filter conditions with pure logical expressions but no NOT EXISTS filters. As an alternative, the set difference may be computed within the federation engine, for which the relevant sets of RDF terms (i.e., Info2(i) and Info2(ii)) have to be retrieved via preemptable SPARQL queries that contain the corresponding triple patterns. The drawback of this alternative option is that large amounts of data may end up being shipped from the server to the federation engine.

If federation members offer a **TPF interface** or a **smart-KG** interface, the same alternative as for SaGe servers can be applied, just with using TPF requests instead of SPARQL queries, which also has the same drawback as in the case of SaGe (potentially large amounts of data might need to be transferred from the federation members to the federation engine). For smart-KG

```
SELECT ?book WHERE {
  ?book rdf:type dbo:Book .
  FILTER NOT EXISTS {
    SELECT ?book WHERE {
      ?book dbo:author ?author .
    }
  } .
} LIMIT 1
```

(a)

```
SELECT ?book WHERE {
  ?book dbo:author ?author .
  FILTER NOT EXISTS {
    SELECT ?book WHERE {
      ?book rdf:type dbo:Book .
    }
  } .
} LIMIT 1
```

(b)

Figure 2: Example queries for checking set difference at SPARQL endpoints [14].

servers it is additionally possible to fetch the corresponding partitions from the server and, then, extract the required sets of RDF terms locally, instead of via TPF requests.

If federation members use a **brTPF interface**, the option outlined for TPF and smart-KG can be simplified as follows. For each pair of triple patterns with a shared variable, first retrieve the set of triples matching one of the triple patterns (i.e., either Info2(i) or Info2(ii)) by using a TPF requests (as are supported by brTPF servers). Thereafter, attach a set of solution mappings (variable bindings for the shared variable) to a brTPF request for the other triple pattern, and obtain a second set of triples. If the number of distinct RDF terms bound to the shared variable in the second set of triples is less than the number of distinct RDF terms bound to the shared variable in the first set of triples, it indicates the set difference is not empty.

For federation members providing the **SPF interface**, the option outlined for brTPF can be used (and so can the option outlined for TPF and smart-KG). Additionally, for pairs of triple patterns that both have the shared variable in the subject position, another option exists: a star pattern request can be constructed and issued for retrieving matching triples for the pair of triple patterns. If the number of distinct RDF terms bound to the shared variable in matching triples for the star pattern is less than the number of distinct RDF terms bound to the shared variable in matching triples of any one of the single triple pattern requests, it indicates that the set difference is not empty.

3.3. Fed-DSATUR

Fed-DSATUR [15] is another source selection approach that does not rely on statistics, indices, or any kind of estimates of the federation members and their datasets. Fed-DSATUR uses a greedy iterative strategy to get optimal query decomposition by considering properties of the SPARQL queries (e.g., the number of subqueries, and the triple patterns assigned to the subqueries as well as the federation members selected for each subquery). The particular information that Fed-DSATUR requires for each federation member is the same as FedX:

Info1 (same as in Section 3.1)**:** For each triple pattern of the given query, determine whether the federation member contains at least one triple that matches the triple pattern.

```
SELECT DISTINCT ?p
WHERE {
  ?s ?p ?o .
}
```

(a)

```
SELECT DISTINCT ?s
WHERE {
  ?s <predicate> ?o .
  FILTER isIRI(?s)
}
```

(b)

```
SELECT DISTINCT ?o
WHERE {
  ?s <predicate> ?o .
  FILTER isIRI(?o)
}
```

(c)

```
SELECT DISTINCT ?p ?s WHERE {
  VALUES ?p { <pred1> <pred2> <pred3> }
  ?s ?p ?o .
  FILTER isIRI(?s)
}
```

(d)

```
SELECT DISTINCT ?p ?o WHERE {
  VALUES ?p { <pred1> <pred2> <pred3> }
  ?s ?p ?o .
  FILTER isIRI(?o)
}
```

(e)

Figure 3: Queries to obtain the data needed for constructing data summaries for HiBISCuS.

3.4. HiBISCuS

The HiBISCuS approach [16] relies on a combination of runtime requests and a summary of the dataset in each federation member. In order to build such a data summary for a federation member the following information needs to be available about the dataset.

Info3: All the URIs that are used as predicates in triples of the dataset.

Info4: For each of these predicate URIs, all the URIs that are used as subjects in triples with the predicate URI.

Info5: For each of these predicate URIs, all the URIs that are used as objects in triples with the predicate URI.

For federation members that provide a **SPARQL endpoint**, the SPARQL query in Figure 3a may be used for retrieving the predicate URIs (Info3). Thereafter, for each of the predicate URIs, two queries with DISTINCT together with an `isIRI()` filter can be used to get the set of URIs that are used as subjects (Info4) and as objects (Info5) in triples with the predicate URI. Figures 3b and 3c illustrate these two queries. As a more efficient alternative, instead of issuing these pairs of queries separately for each predicate URI, the requests can be batched by including multiple predicate URIs via a VALUES clause (or UNION or FILTER [17]), as illustrated in Figures 3d–3e.

For federation members with a **SaGe interface**, the same approach as for SPARQL endpoints can be applied. Notice, however, that the preemptable operator for DISTINCT has a client-side component, which means that the federation engine would need to do some processing itself for the queries that retrieve the relevant information (Figures 3a–3c) and that potentially some irrelevant intermediate solutions would be retrieved from the SaGe server.

If federation members expose data through a **TPF interface**, a **brTPF interface** or an **SPF interface**, the set of predicate URIs (Info3) can be obtained by a request with a triple pattern of the form (?s, ?p, ?o), which retrieves the complete dataset of the federation member. The removal of duplicate (predicate) URIs has to be taken care of by the federation engine. After retrieving the entire dataset in order to obtain all the predicate URIs, the required information about subject URIs (Info4) and object URIs (Info5) can then also be extracted from the retrieved data.

For federation members using a **smart-KG** interface, the set of distinct predicate URIs (Info3) can be obtained from the smart-KG catalog provided to the federation engine by the smart-KG server. Thereafter, for each of the predicate URIs, the set of subject URIs (Info4) and object URIs (Info5) can be obtained by fetching the relevant partitions from the smart-KG server or by issuing TPF requests with triple patterns of the form (?s, <predicate>, ?o). Since this would be done for every predicate URI, the federation engine would also end up retrieving the complete dataset from the smart-KG server. Similar to TPF-like interfaces, removing duplicate URIs and filtering subjects/objects that are not URIs can only be done within the federation engine.

3.5. Source Selection in CostFed

CostFed [18] also relies on a combination of runtime requests and a summary of the dataset in each federation member. The information that CostFed needs to build its type of data summary for a federation member is exactly the same information as is needed by HiBISCuS.

Info3 (same as in Section 3.4)**:** All the URIs that are used as predicates in triples of the dataset.

Info4 (same as in Section 3.4)**:** For each of these predicate URIs, all the URIs that are used as subjects in triples with the predicate URI.

Info5 (same as in Section 3.4)**:** For each of these predicate URIs, all the URIs that are used as objects in triples with the predicate URI.

As already described in Section 3.4, Info3, Info4 and Info5 can be obtained in different ways depending on the interface of the federation member, where for several types of interfaces (TPF, brTPF, SPF, and smart-KG), this would mean retrieving the complete dataset from the corresponding federation member.

3.6. Source Selection in ANAPSID

ANAPSID [21] makes use of service descriptions and runtime request to determine whether SPARQL endpoints can answer a triple pattern or not. The particular information that the source selection of ANAPSID requires from these service descriptions is a list of predicate URIs, which is the same as **Info3** (cf. Section 3.4).

3.7. Source Selection in ADERIS

ADERIS [19] requires specific information about the data contained in each federation member *before* performing any query. The particular information that the source selection of ADERIS requires for each federation member is:

Info3 (same as in Section 3.4)**:** All the URIs that are used as predicates in triples of the dataset.

Info6: For each of these predicate URIs, the number of triples with that predicate URI.

As already described in Section 3.4, the set of distinct predicates URIs (Info3) can be obtained in different ways depending on the interface of the federation member.

```
SELECT DISTINCT ?s
WHERE {
  ?s ?p ?o .
}
```

(a)

```
SELECT DISTINCT ?p
WHERE {
  <subject> ?p ?o .
}
```

(b)

```
SELECT DISTINCT ?o1 WHERE {
  ?s dbo:birthDate ?o1 .
  ?s dbo:activeYearsStartYear ?o2 .
  ?s foaf:name ?o3 .
  FILTER isIRI(?o1)
}
```

(c)

Figure 4: Example queries for retrieving (a–b) Info7, and (c) Info9 for Odyssey.

Once the predicate URIs are known to the federation engine, for federation members that provide a **SPARQL endpoint** or a **SaGe interface**, the number of triples with the same predicate URI (Info6) can be obtained via queries using the COUNT function.

For federation members with a **smart-KG interface**, a request with a triple pattern of the form (?s, <predicate>, ?o) can be issued for each predicate URI to obtain the first page of matching triples with that predicate, and estimates of the count metadata (Info6) can be obtained from the retrieved pages.

For federation members supporting **TPF**, **brTPF**, or **SPF**, since obtaining the set of predicates URIs (Info3) required retrieving the complete dataset from the federation member (cf. Section 3.4), the counting of triples per predicate URI (Info6) can then be done in the federation engine.

3.8. Source Selection in DARQ

DARQ [20] relies on a so-called service description (represented in RDF), which describes the data available from a federation member in form of capabilities. The particular information that DARQ requires each federation member to provide includes **Info3** (cf. Section 3.4) and hand-crafted constraints on subjects and objects. These constraints are represented using SPARQL filter expressions.

3.9. Source Selection in Odyssey

Odyssey [22] relies on statistics computed by federation members and shared with the federation engine. The particular information that Odyssey requires from each federation member is the following:

Info7: For each URI that appears in the subject of some triple, the characteristic set [6] of that subject URI; i.e., all the URIs that are used as predicates in triples with that subject URI.

Info8: For each such characteristic set, all subject URIs that have this characteristic set.

Info9: For each such characteristic set, all URIs that are used as objects in triples with each predicate within the characteristic set.

If the federation members do not provide such information, the federation engine may obtain the information via requests to the data access interfaces of the federation members.

For **SPARQL endpoints** and **SaGe servers**, to get all predicates of each subject URI (Info7), a set of all subject URIs needs to be retrieved first, which can be achieved via a query with DISTINCT (shown in Figure 4a). Then, for each subject URI, the predicate URIs that belong to the characteristic set of that subject URI (Info7) can be retrieved by a SPARQL query of the form in Figure 4b. These requests may also be batched for multiple subject URIs by using a VALUES clause (similar to the batching outlined in Figure 3d in Section 3.4, but with having subject URIs in the VALUES clause, instead of predicate URIs). Once the characteristic sets for all subject URIs have been obtained, all subject URIs that share the same characteristic set (Info8) can be determined within the federation engine. Thereafter, the set of object URIs for each predicate within a characteristic set (Info9) can be obtained by queries such as the one in Figure 4c. Notice that all these queries use the DISTINCT keyword which, in the case of SaGe, as already discussed in Section 3.4, requires some processing within the federation engine and potential retrieval of some irrelevant intermediate solutions.

For federation members with a **TPF interface**, a **brTPF interface**, a **SPF interface**, or a **smart-KG interface**, obtaining the set of predicates per subject URI requires retrieving the whole RDF dataset to the federation engine. After retrieving the entire dataset, all required information can be determined within the federation engine.

3.10. VoID-Based Approaches

To identify and exclude federation members that have no matching triples for some of the triple patterns of a given query, some engines such as SPLENDID [23], Semagrow [24] and LHD [25] rely on external metadata that is shared by federation members and described using the Vocabulary of Interlinked Datasets (VoID) [26]. For source selection, these engines mainly make use of predicate information in the VoID descriptions, which allows them to identify whether federation members are relevant or irrelevant for any triple pattern with a bound predicate. For triple patterns with an unbound predicate, the engines use ASK queries as done by FedX. Hence, in general, the information that the source selection approaches of these engines use is the same as **Info1** (Section 3.1) and **Info3** (Section 3.4).

4. Summary of Findings

When taking into account the capabilities of different interfaces for providing access to RDF data, we observe that all existing source selection approaches can be adopted for heterogeneous federations by making some interface-specific modifications to the processes they use to collect the information they rely on. However, for many cases, these modifications would result in processes that are of limited suitability from a practical perspective.

In particular, for all approaches that rely on some form of data summary, index, or statistics about the data of each federation member—which includes the DARQ approach [20], ADERIS [19], HiBISCuS [16], CostFed [18], Odyssey [22], as well as VoID-based approaches as used in SPLENDID [23], LHD [25], Semagrow [24]—if federation members provide certain types of interfaces, obtaining the information required for building the corresponding data summary or index would result in retrieving the complete dataset from these federation members. As

a consequence, the retrieved datasets may then also be stored locally, making the use of a federation engine obsolete.

Even for the approach of Lusail [14], potentially large amounts of data would need to be transferred from the federation members to the query engine for the purpose of deciding about combining triple patterns into subqueries.

The only approaches for which obtaining the relevant information is practical are the approach of FedX [13] and the Fed-DSATUR approach [15]. For these approaches, all required information can be obtained by issuing only a few requests to the federation members, where these requests are guaranteed *not* to result in retrieving large portions of the datasets.

Acknowledgments

This work was funded by CUGS (the National Graduate School in Computer Science, Sweden), by Vetenskapsrådet (the Swedish Research Council, project reg. no. 2019-05655), and by the CENIIT program at Linköping University (project no. 17.05).

References

[1] L. Feigenbaum, G. T. Williams, K. G. Clark, E. Torres, SPARQL 1.1 Protocol, W3C Recommendation, Online at http://www.w3.org/TR/sparql11-protocol/, 2013.

[2] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web, Journal of Web Semantics 37–38 (2016) 184–206.

[3] O. Hartig, C. Buil-Aranda, Bindings-Restricted Triple Pattern Fragments, in: Proceedings of the 15th Int. Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE), 2016.

[4] C. Aebeloe, I. Keles, G. Montoya, K. Hose, Star Pattern Fragments: Accessing Knowledge Graphs through Star Patterns, arXiv preprint arXiv:2002.09172 (2020).

[5] A. Azzam, J. D. Fernández, M. Acosta, M. Beno, A. Polleres, SMART-KG: Hybrid Shipping for SPARQL Querying on the Web, in: Proceedings of The Web Conference (WWW), 2020.

[6] T. Neumann, G. Moerkotte, Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins, in: Proceedings of the 27th International Conference on Data Engineering (ICDE), 2011.

[7] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, M. Arias, Binary RDF Representation for Publication and Exchange (HDT), Journal of Web Semantics 19 (2013) 22–41.

[8] A. Azzam, C. Aebeloe, G. Montoya, I. Keles, A. Polleres, K. Hose, WiseKG: Balanced Access to Web Knowledge Graphs, in: Proceedings of the Web Conference (WWW), 2021.

[9] T. Minier, H. Skaf-Molli, P. Molli, SaGe: Web Preemption for Public SPARQL Query Services, in: Proceedings of the Web Conference (WWW), 2019.

[10] A. Grall, T. Minier, H. Skaf-Molli, P. Molli, Processing SPARQL Aggregate Queries with Web Preemption, in: Proc. of the 17th Extended Semantic Web Conference (ESWC), 2020.

[11] J. Aimonier-Davat, H. Skaf-Molli, P. Molli, Processing SPARQL Property Path Queries Online with Web Preemption, in: Proceedings of the 18th Extended Semantic Web Conference (ESWC), 2021.
[12] J. Aimonier-Davat, H. Skaf-Molli, P. Molli, A. Grall, T. Minier, Online Approximative SPARQL Query Processing for COUNT-DISTINCT Queries with Web Preemption, Semantic Web (2022).
[13] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: Proceedings of the 10th International Semantic Web Conference (ISWC), 2011.
[14] I. Abdelaziz, E. Mansour, M. Ouzzani, A. Aboulnaga, P. Kalnis, Lusail: A System for Querying Linked Data at Scale, Proc. of the VLDB Endowment (2017).
[15] M.-E. Vidal, S. Castillo, M. Acosta, G. Montoya, G. Palma, On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries, in: Transactions on Large-Scale Data- and Knowledge-Centered Systems XXV, Springer, 2016, pp. 109–149.
[16] M. Saleem, A.-C. Ngonga Ngomo, HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, in: Proceeding of the European Semantic Web Conference (ESWC), 2014.
[17] C. B. Aranda, A. Polleres, J. Umbrich, Strategies for Executing Federated Queries in SPARQL1.1, in: Proceedings of the 13th International Semantic Web Conference (ISWC), 2014.
[18] M. Saleem, A. Potocki, T. Soru, O. Hartig, A.-C. N. Ngomo, CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation, in: Proceedings of the SEMANTiCS Conference, 2018.
[19] S. Lynden, I. Kojima, A. Matono, Y. Tanimura, ADERIS: An Adaptive Query Processor for Joining Federated SPARQL Endpoints, in: Proceedings of the Int. Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE), 2011.
[20] B. Quilitz, U. Leser, Querying Distributed RDF Data Sources with SPARQL, in: Proceedings of the European Semantic Web Conference (ESWC), 2008.
[21] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints, in: Proceedings of the International Semantic Web Conference (ISWC), 2011.
[22] G. Montoya, H. Skaf-Molli, K. Hose, The Odyssey Approach for Optimizing Federated SPARQL Queries, in: Proceedings of the International Semantic Web Conference (ISWC), 2017.
[23] O. Görlitz, S. Staab, SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions, in: Proc. of the 2nd Int. Workshop on Consuming Linked Data (COLD), 2011.
[24] A. Charalambidis, A. Troumpoukis, S. Konstantopoulos, SemaGrow: Optimizing Federated SPARQL Queries, in: Proc. of the 11th Int. Conf. on Semantic Systems (SEMANTiCS), 2015.
[25] X. Wang, T. Tiropanis, H. C. Davis, LHD: Optimising Linked Data Query Processing Using Parallelisation, in: Proceedings of the Linked Data on the Web Workshop (LDOW), 2013.
[26] K. Alexander, R. Cyganiak, M. Hausenblas, J. Zhao, Describing Linked Datasets with the VoID Vocabulary, W3C Interest Group Note, Online at http://www.w3.org/TR/void/, 2011.