Towards Query Processing over Heterogeneous Federations of RDF Data Sources

Sijin Cheng[1][0000-0003-4363-0654] and Olaf Hartig[1][0000-0002-1741-2090]

Linköping University, Linköping, Sweden
{sijin.cheng, olaf.hartig}@liu.se

Abstract A federation of RDF data sources offers enormous potential when answers or insights of queries are unavailable via a single data source. As various interfaces for accessing RDF data are proposed, one challenge for querying such a federation is that the federation members are heterogeneous in terms of the type of data access interfaces. There does not exist any research on systematic approaches to tackle this challenge. To provide a formal foundation for future approaches that aim to address this challenge, we have introduced a language, called *FedQPL*, that can be used for representing query execution plans in this setting. With a poster in the conference we generally want to outline the vision for the next generation of query engines for such federations and, in this context, we want to raise awareness in the Semantic Web community for our language. In this extended abstract, we first discuss challenges in query processing over such heterogeneous federations; thereafter, we briefly introduce our proposed language, which we have extended with a few new features that we did not have in the version published originally.

Keywords: Heterogeneous Federations · Query Plan Language

1 Motivation and Challenges

Efficiently processing queries over federation members that provide the same type of interface (i.e., SPARQL endpoint) has been explored extensively [2]. However, as different types of interfaces are proposed to publish RDF data sources, such as Triple Pattern Fragment (TPF) interface [16], Bindings-Restricted TPF (brTPF) interface [8], Star Pattern Fragments (SPF) [3], SaGe interface [11], smart-KG interface [5], and WiseKG [4], providers of RDF data sources may choose to publish their RDF data via a different type of interface depending on the properties of these interfaces [9,12]. As a result, federations of RDF data sources may become heterogeneous in terms of access interfaces. Due to this heterogeneity, query processing over such federations faces extra challenges, involving the key sub-tasks of federated query processing, such as source selection, query decomposition, planning and optimization, and query execution.

Example 1 *As a motivating example, consider a federation F_{ex} with two members: Federation member fm_1 provides a SPARQL endpoint interface for the RDF graph $G_1 = \{(\mathsf{a}, \mathsf{foaf{:}knows}, \mathsf{c}), (\mathsf{c}, \mathsf{foaf{:}knows}, \mathsf{d})\}$, whereas fm_2 provides a TPF interface for the RDF graph $G_2 = \{(\mathsf{c}, \mathsf{foaf{:}name}, \texttt{"Alice"}), (\mathsf{c}, \mathsf{foaf{:}age}, \mathsf{21})\}$.*

Example 2 *Consider a basic graph pattern (BGP) $B_{\mathsf{ex}} = \{tp_1, tp_2, tp_3\}$ with the three triple patterns $tp_1 = (?x, \texttt{foaf:knows}, ?y)$, $tp_2 = (?y, \texttt{foaf:name}, ?z)$ and $tp_3 = (?y, \texttt{foaf:age}, ?g)$. When evaluating B_{ex} over the example federation F_{ex}, we expect to obtain one solution mapping: $\mu_1 = \{?x \rightarrow \texttt{a}, ?y \rightarrow \texttt{c}, ?z \rightarrow \texttt{"Alice"}, ?g \rightarrow \texttt{21}\}$.*

In order to devise an efficient solution for answering the BGP B_{ex}, the properties and constraints of each member's interface must be considered. In terms of source selection, existing engines [14,1,6,13] generally rely on a set of SPARQL ASK queries or on metadata about federation members to determine which federation member(s) can evaluate each triple pattern. Due to the fact that the TPF interface, however, cannot answer such SPARQL ASK queries, it is necessary to consider the type of interface during source selection in heterogeneous federations. In addition, we are unable to apply an existing query decomposition approach easily since not all forms of subqueries can be answered directly by every interface. For instance, FedX [14] would group the triple patterns tp_2 and tp_3 into a subquery as they can be evaluated exclusively at federation member fm_2. While such an exclusive group is beneficial for, e.g., a SPARQL endpoint, the TPF interface provided by fm_2 cannot answer such a group pattern directly. Furthermore, some works leverage information and metadata about the federation member for query planning and optimization, such as estimating join cardinality or pruning data sources that do not contribute to the final results. This process should consider the properties of each interface as different interfaces provide different types of metadata and support exploring of different information.

When it comes to physical plans, different interfaces may require the engine to leverage specific physical operators. For instance, possible algorithms for the implementation of a *join* operator are a standard (local) nested-loops join, or RDF-specific variations of the semijoin and the bind join [2]. The latter algorithms rely on a data access interface in which the given input solution mappings can be captured as part of the requests. If the interface is more expressive (e.g., a SPARQL endpoint), concrete examples of such algorithms can be found in the SPARQL endpoint federation engines [14,6]. However, for less expressive interfaces (such as the TPF interface), the algorithm can be implemented using a variation of an index nested-loops join in which a separate request is created for each input solution mapping [16].

Therefore, for the next generation of federation engines, it is not sufficient to merely combine and integrate existing solutions as the features and constraints of each interface must be thoroughly considered in the context of heterogeneous federations. Some work has been done on dealing with query processing for heterogeneous interfaces. Comunica [15] is a modular engine that can be used to run queries on heterogeneous data sources, however it simply handles query execution at the triple pattern level, without considering features and optimization options for heterogeneous federations. We believe that any principled approach to querying heterogeneous federations of such RDF data sources must be based on a solid formal foundation. In recent work, Heling and Acosta [10] introduce interface-aware approaches for query decomposition and query planning. Similar to our work, they also formalize the concept of federations. We argue that the formal foundation should provide not only a formal data model capturing the federation concepts, including the corresponding query semantics, but also formal concepts that precisely define the artifacts produced by the various steps of query processing, including source selection, query decomposition, planning and optimization, and query execution.

2 FedQPL: A Language for Query Plans over Federations

We have defined *FedQPL* [7], a language for formally specifying logical query plans. This language can be applied to more precisely define query planning and optimization approaches, as well as to represent the logical plans in a query engine. The key innovations of this language over the standard SPARQL algebra are that it contains operators to make explicit which federation member is accessed in each part of a query plan and to distinguish different ways of accessing a federation member. FedQPL features operators that explicitly capture the intention to execute a certain subquery at a specific federation member, as well as explicitly distinguish whether such access is meant to be based solely on the given subquery or also on intermediate results obtained for other subqueries. While approaches that focus on homogeneous federations can also benefit from these features, we argue that such features are essential for any principled approach to query planning in heterogeneous federations where the properties and the limitations of different data access interfaces must be considered.

To give an intuition of what FedQPL provides, we briefly go through the syntax of FedQPL expressions in this poster paper. Note that we have extended FedQPL with four additional operators that we did not have in the original paper. With this extension, FedQPL can now capture plans for the complete version 1.0 fragment of SPARQL. In our original paper, we also provide an extensive set of equivalences for FedQPL expressions that can be used as query rewriting rules for query optimization.

Definition 1. *A **FedQPL expression** φ can be constructed from the following grammar, in which* req*,* tpAdd*,* bgpAdd*,* join*,* union*,* mj*,* mu*, (,),* filter*,* leftJoin*,* tpOptAdd*, and* bgpOptAdd *are terminal symbols.*[1] *ρ is an expression in the request language L_{req} of some interface [7], fm is a federation member, tp is a triple pattern, B is a BGP, F is a SPARQL filter expression, and Φ is a nonempty set of FedQPL expressions.*

$$\begin{aligned}\varphi ::= {} & \mathsf{req}^{\rho}_{\mathit{fm}} \mid \mathsf{tpAdd}^{tp}_{\mathit{fm}}(\varphi) \mid \mathsf{tpOptAdd}^{tp}_{\mathit{fm}}(\varphi) \mid \mathsf{bgpAdd}^{B}_{\mathit{fm}}(\varphi) \mid \mathsf{bgpOptAdd}^{B}_{\mathit{fm}}(\varphi) \mid \\ & \mathsf{join}(\varphi,\varphi) \mid \mathsf{leftjoin}(\varphi,\varphi) \mid \mathsf{union}(\varphi,\varphi) \mid \mathsf{mj}\ \Phi \mid \mathsf{mu}\ \Phi \mid \mathsf{filter}^{F}(\varphi)\end{aligned}$$

While these operators are generally independent of the type of interface provided by the corresponding federation member *fm*, some operators can be used only for federation members with an interface that has specific properties.

[1] The operators captured by the last four symbols are the ones that we have added.

The first operator, req, captures the intention to retrieve the result of a certain (sub)query ρ from a given federation member. For instance, the aim to retrieve solution mappings for tp_1 (of B_{ex} in Example 2) from the member fm_1 (of F_{ex} in Example 1) can be represented as $\mathit{req}^{tp_1}_{fm_1}$. The forms of the (sub)queries ρ that can be used for this operator depend on the interface provided by the federation member, which our formalization abstracts by the notion of request languages [7]. For instance, with req we can represent a request with a whole BGP, but only for interfaces that support BGP requests.

The unary operator tpAdd captures the intention to access a federation member to obtain solution mappings for a single triple pattern that must be compatible with solution mappings obtained from the plan represented by the given subexpression. For instance, in our running example we observe that only one of the solution mappings for tp_1 from fm_1 can be joined with solution mappings for tp_2 from fm_2. To produce the join between the two sets of solution mappings we may use the output of $\mathit{req}^{tp_1}_{fm_1}$ as input to retrieve only the compatible solution mappings for tp_2 from fm_2, which can be represented as the following FedQPL expression: $\mathit{tpAdd}^{tp_2}_{fm_2}\big(\mathit{req}^{tp_1}_{fm_1}\big)$. The operator bgpAdd is a BGP-based variation of tpAdd. In contrast to these operators, join is a binary operator that joins two inputs, capturing the intention to get the input sets of solution mappings independently, and then join them in the query federation engine.

The operator $\mathit{leftjoin}$ is binary operator that captures the intention to extend information using an optional part. If the optional part (right input) has no matching solution mappings, no bindings are created but it does not eliminate the solutions. Consider query pattern P_{opt} as a variation of the example BGP B_{ex} where $\{tp_2, tp_3\}$ are optional. Federation member fm_1 contributes two solution mappings for tp_1. Although $?y \to d$ is not compatible with solution mappings in the optional part, querying the federation F_{ex} (cf. Example 1) results in two solution mappings for P_{opt} as the non-optional information is returned anyway. Similar to the difference between join and tpAdd (respectively bgpAdd), $\mathit{tpOptAdd}$ and $\mathit{bgpOptAdd}$ are unary variations of $\mathit{leftjoin}$ that access a federation member to obtain bindings for a given triple pattern, respectively a BGP, to optionally extend the solution mappings of a given intermediate query result.

The operator filter captures the intention to impose a constraint on the solution mappings obtained from the plan represented by the given subexpression. Continuing with the example BGP B_{ex}, if adding a filter condition ($?g < 20$) on the solutions, the query engine will return no solution mapping.

As for the remaining operators, union lifts the standard SPARQL algebra operator union into the FedQPL language, whereas mj and mu are multiway variations of join and union to capture the intention to apply a multiway algorithm that can combine an arbitrary number of inputs.

3 Future Work

So far we have focused on providing the formal foundations of query processing approaches over heterogeneous federations. We believe that establishing these foundations is a necessary prerequisite for a systematic study of the next generation of federation engines. Consequently, we first plan to investigate what adaptations are needed in source selection approaches and query decomposition in the scenario of heterogeneous interfaces. Secondly, we plan to design effective and efficient query planning and optimization approaches for queries over heterogeneous federations. We will implement these approaches in a new federation engine for federated query processing over heterogeneous interfaces.

References

1. Abdelaziz, I., Mansour, E., Ouzzani, M., Aboulnaga, A., Kalnis, P.: Lusail: A System for Querying Linked Data at Scale. Proc. of the VLDB Endowment (2017)
2. Acosta, M., Hartig, O., Sequeda, J.F.: Federated RDF Query Processing. In: Encyclopedia of Big Data Technologies. (2019)
3. Aebeloe, C., Keles, I., Montoya, G., Hose, K.: Star Pattern Fragments: Accessing Knowledge Graphs through Star Patterns. arXiv preprint arXiv:2002.09172 (2020)
4. Azzam, A., Aebeloe, C., Montoya, G., Keles, I., Polleres, A., Hose, K.: WiseKG: Balanced Access to Web Knowledge Graphs. In: Proc. of the Web Conf. (2021)
5. Azzam, A., Fernández, J.D., Acosta, M., Beno, M., Polleres, A.: SMART-KG: Hybrid Shipping for SPARQL Querying on the Web. In: Proceedings of The Web Conference (WWW) (2020)
6. Charalambidis, A., Troumpoukis, A., Konstantopoulos, S.: SemaGrow: Optimizing Federated SPARQL Queries. In: Proceedings of the 11th International Conference on Semantic Systems (SEMANTICS) (2015)
7. Cheng, S., Hartig, O.: FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources. In: Proc. of the 22nd Int. Conference on Information Integration and Web-based Applications & Services (2020)
8. Hartig, O., Buil-Aranda, C.: Bindings-Restricted Triple Pattern Fragments. In: Proceedings of the 15th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE) (2016)
9. Hartig, O., Letter, I., Pérez, J.: A Formal Framework for Comparing Linked Data Fragments. In: Proc. of the 16th Int. Semantic Web Conference (ISWC) (2017)
10. Heling, L., Acosta, M.: Federated SPARQL Query Processing over Heterogeneous Linked Data Fragments. In: Proceedings of The Web Conference (WWW) (2022)
11. Minier, T., Skaf-Molli, H., Molli, P.: SaGe: Web Preemption for Public SPARQL Query Services. In: Proceedings of the Web Conference (WWW) (2019)
12. Montoya, G., Aebeloe, C., Hose, K.: Towards Efficient Query Processing over Heterogeneous RDF Interfaces. In: 2nd Workshop on Decentralizing the Semantic Web (DeSemWeb) (2018)
13. Saleem, M., Ngonga Ngomo, A.C.: Hibiscus: Hypergraph-based Source Selection for SPARQL Endpoint Federation. In: European semantic web Conf.Ṡpringer (2014)
14. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: Proceedings of the 10th International Semantic Web Conference (ISWC) (2011)
15. Taelman, R., Herwegen, J.V., Sande, M.V., Verborgh, R.: Comunica: a Modular SPARQL Query Engine for the Web. In: Int. Semantic Web Conf. (ISWC) (2018)
16. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web. Journal of Web Semantics (2016)