How Diverse Are Federated Query Execution Plans Really?

Jakobsen, Anders Langballe; Montoya, Gabriela; Hose, Katja

Published in:
The Semantic Web

DOI (link to publication from Publisher):
10.1007/978-3-030-32327-1_21

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

How diverse are federated query execution plans really?

Anders Langballe Jakobsen, Gabriela Montoya(✉), and Katja Hose

Aalborg University, Denmark
{alja,gmontoya,khose}@cs.aau.dk

Abstract. Federated query engines optimize and execute SPARQL queries over the data accessible via SPARQL endpoints; even in the absence of information about which sources provide relevant data. Different query optimization strategies may produce different execution plans, which in many cases explains the huge differences in performance that we encounter with state-of-the-art federated query engines. Related work has so far mainly focused on execution time and number of selected sources, overlooking the importance of the execution plans themselves. In this demonstration paper, we therefore present PIPE, a tool that allows for comparing federated query engines in terms of performance, execution plans, and query answers. Currently, PIPE supports five state-of-the-art federated query engines; we provide a Java library for straightforward integration of additional federated query engines.

1 Introduction

The efforts of the Semantic Web community have led to the publication of billions of RDF triples organized into thousands of datasets. A key element of Linked Data is that the datasets are not isolated but they include links to other datasets, which allows users to pose queries that cannot be answered using a single dataset alone. Instead, such *federated* queries require data from multiple datasets. In this paper, we consider federated queries that do not include hints about where the triple patterns should be executed but rely on a federated query engine to optimize the query, i.e., select relevant sources, decompose the query into subqueries, delegate those subqueries to the appropriate sources, combine the partial answers, and compute the final result to the original query. We assume that a source provides access to a dataset via a SPARQL endpoint, i.e., an RDF interface that supports full SPARQL expressiveness.

Several federated query engines, such as [1–3, 6], have been proposed in the last decade. Even if these engines have many differences, including different implementations of the SPARQL operators, the most important difference is their query optimization strategies. Different strategies produce quite different execution plans, and in many cases the huge differences in performance are mainly due to key differences in the execution plans produced by the engines. So far, comparisons among these engines have been mainly based on execution time and to some extent on the number of selected sources, but comparisons that take the diversity of execution plans into consideration have been very limited and mainly used for motivating examples.

In this paper, we therefore present PIPE, Performance Inspector and Plan Explorer, a tool that allows for comparing different federated query engines in terms of their performance, i.e., planning and execution time, computed execution plans, and query answers. A key novelty of PIPE is providing a uniform framework for studying the impact of optimization strategies used by the engines on the shapes of the produced execution plans. Supporting the analysis of these plans allows for easing the understanding of query optimization strategies, including the identification of their limitations and the proposal of novel query optimization strategies.

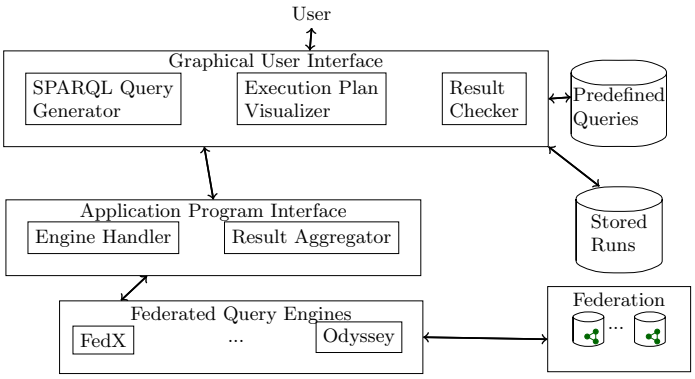

Fig. 1. PIPE's architecture

2 System Architecture

Figure 1 illustrates PIPE's architecture. The user interacts with our system through a Graphical User Interface (GUI) that relies on an Application Program Interface (API) to interact with the federated query engines, and provide an abstract representation of the engines' execution plans, results, and performance measurements. In the remainder of this section, we sketch purpose and core functionality of each system component.

GUI. It allows to load a predefined SPARQL query or provide a new query using the *simple* and *advanced* query interfaces. These query interfaces allow for creating queries with different trade-offs between expressiveness, e.g., possible operators, and usability for novice users, e.g., well-defined query structure. It also allows for choosing the endpoints for the federation, the federated query engines to study, and a timeout value. It visualizes the execution plans as trees and allows to explore the complete answers. If two or more engines produce the same plan, they are combined in order to ease the comparison of the plans that are actually different. Further functionality includes saving executions for later study and checking if the answers obtained by different engines are consistent. Currently, consistency of the answers is limited to checking the number of answers.

API. Given an analysis request, consisting of a query, federated engines, and sources to use, the API creates an execution job in which it sequentially runs each engine to avoid any interference between different executions. Once a query has been executed on all engines, the API aggregates the results and makes them

available to the user. When execution jobs are created, users are provided with a unique token that can be used to cancel the execution job at any time. As the API does not rely on any specific implementation detail from the query engines, it can work with any federated query engine as long as the engine outputs a JSON object that adheres to the expected structure.

Federated Query Engine. It identifies relevant sources, decomposes the query into subqueries, and combines the subquery answers to produce the query answer. Minor changes were done to the engines to homogenize the output of the query optimization (execution plan), query execution (query results), and the measurement of relevant metrics. To minimize the code changes and ease the inclusion of other engines, our changes are mainly provided as a Java library that can be used to integrate federated query engines that rely on the RDF4J framework[1] into our comparison tool. In addition to performing serialization, the library automatically converts the execution plans into a canonical abstract version that is supported by our GUI. PIPE currently supports FedX [6], HiBISCuS [4], Odyssey [3], SemaGrow [1], and SPLENDID [2].

```
SELECT DISTINCT * WHERE {
  ?film dbo:director ?director .                (tp1)
  ?film rdf:type dbo:Film .                     (tp2)
  ?movie owl:sameAs ?film .                     (tp3)
  ?movie movie:film_subject film_subject:444 . (tp4)
  ?movie dcterms:title ?title                   (tp5)
}
```

Listing 1.1. `Q1`: find the time travel film's directors (subject 444)

3 Processing Federated SPARQL Queries

Consider query `Q1` (Listing 1.1), which asks for directors of time travel films, and a federation of SPARQL endpoints composed of ChEBI, KEGG, Drugbank, Geonames, DBpedia, Jamendo, NYTimes, SWDF, and LMDB. The execution plans computed by PIPE's currently supported federated query engines are depicted in Fig. 2. FedX, HiBISCuS, SemaGrow, and SPLENDID identify eight endpoints as relevant for `tp3` because there is at least one triple in each of these endpoints that has the same predicate as `tp3`, i.e., `owl:same`. Even if these engines select the same sources for all the triple patterns, SemaGrow computes a considerably better execution plan at the tradeoff of spending more time during planning (Fig. 3 (b)). Semagrow relies on voiD statistics to correctly estimate that `tp4` is the most selective triple pattern, and as such, it is better to execute `tp4` first. FedX and HiBISCuS rely solely on heuristics to assess selectivity, e.g., `tp1` and `tp2` are identified as the most selective triple patterns because they can be executed exclusively by one endpoint. Similarly to SemaGrow, Odyssey relies on statistics to identify that `tp4` is the most selective triple pattern. Differently from SemaGrow, Odyssey uses its federated statistics to identify that from all the endpoints that could be relevant for `tp3` and `tp5`, only LMDB has triples that describe entities that also have the property used as predicate in the triple pattern `tp4`, and therefore is the only one that is actually relevant for the

[1] http://rdf4j.org

execution of this query. A video of PIPE and its functionality is available at http://qweb.cs.aau.dk/pipe.

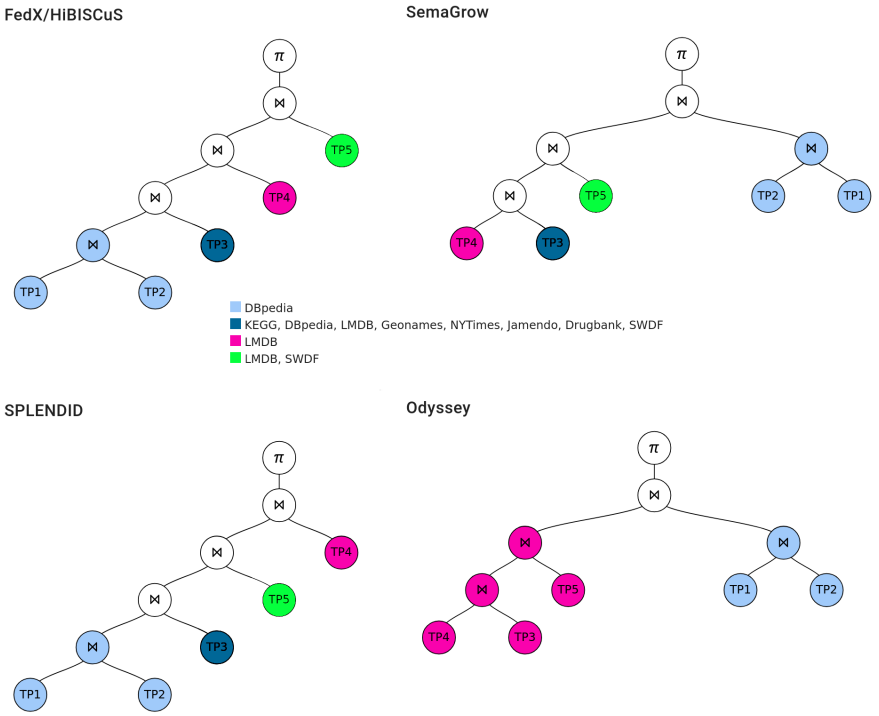

Fig. 2. Execution plans for `Q1`

4 Demonstration

At the conference, PIPE will be used to compare state-of-the-art federated query engines: FedX [6], HiBISCuS [4], Odyssey [3], SemaGrow [1], and SPLENDID [2].

Conference attendees will be able to formulate arbitrary SPARQL queries if they are experts or use the simple query interface to compose a basic graph pattern component by component. Moreover, users can choose a predefined query among the 25 queries defined in the FedBench benchmark [5] and focus on using PIPE to study how diverse the produced execution plans and their impact on the engines' performances are.

For the purposes of the demonstration, a federation of nine SPARQL endpoints as defined in the General Linked Open Data Collection and the Life Science Data Collection of the FedBench benchmark [5] will be available locally to avoid any network connectivity issues during the demonstration.

Figure 3 shows PIPE's GUI following the execution of `Q1` (Fig. 3(a)). It shows the performance of the engines that did not time out after 180s (Fig. 3(b)) using a logarithmic scale on the vertical axis (Fig. 3(c)). Using PIPE's GUI, conference attendees can also choose to look at the execution plans (as shown in Fig. 2) by

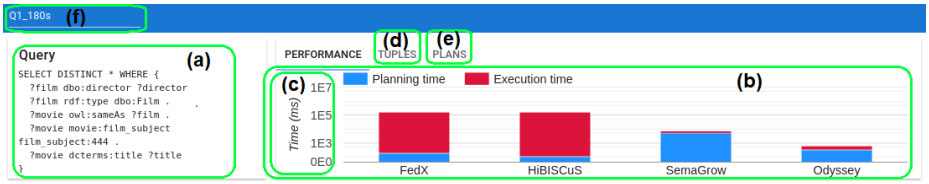

Fig. 3. Performance of the federated query engines for `Q1`

clicking on "PLANS" (Fig. 3(d)) or examine the query answers by clicking on "TUPLES" (Fig. 3(e)). The analysis of Q1 is saved to the browser's local storage after giving it a name (Fig. 3(f)).

5 Conclusion

In this paper, we have presented PIPE, a tool for comparing federated query engines in terms of performance, computed execution plans, and query answers. PIPE allows for inspecting the query optimization result, i.e., the execution plans, and in many cases, it allows for explaining the huge differences observable in terms of execution time. Notably, PIPE allows to assess how diverse the query execution plans produced by the federated query engines are. Currently, the query execution plans represent the operations as logical operators; as part of our future work, we plan to also depict physical operators. In addition, we plan to introduce additional performance measures, such as the number of requests made to the endpoints and the amount of transferred data.

Acknowledgments. This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301B and Aalborg University's Talent Programme.

References

1. Charalambidis, A., Troumpoukis, A., Konstantopoulos, S.: SemaGrow: Optimizing Federated SPARQL queries. In: SEMANTICS'15. pp. 121–128 (2015)
2. Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: COLD'11 (2011)
3. Montoya, G., Skaf-Molli, H., Hose, K.: The Odyssey Approach for Optimizing Federated SPARQL Queries. In: ISWC'17. pp. 471–489 (2017)
4. Saleem, M., Ngomo, A.N.: HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In: ESWC'14. pp. 176–191 (2014)
5. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In: ISWC'11. pp. 585–600 (2011)
6. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: ISWC'11. pp. 601–616 (2011)