SILURIAN: a Sparql vIsuaLizer for UndeRstanding querIes And federatioNs

Simón Castillo, Guillermo Palma, Maria-Esther Vidal

Universidad Simón Bolívar, Caracas, Venezuela
{scastillo, gpalma, mvidal}@ldc.usb.ve

Abstract. SPARQL federated queries can be affected by both characteristics of the query and datasets in the federation. We present SILURIAN a Sparql visualizer for understanding queries and federations. SILURIAN visualizes SPARQL queries and, thus, it allows the analysis and understanding of a query complexity with respect to relevant endpoints and shapes of the possible plans.

1 Introduction

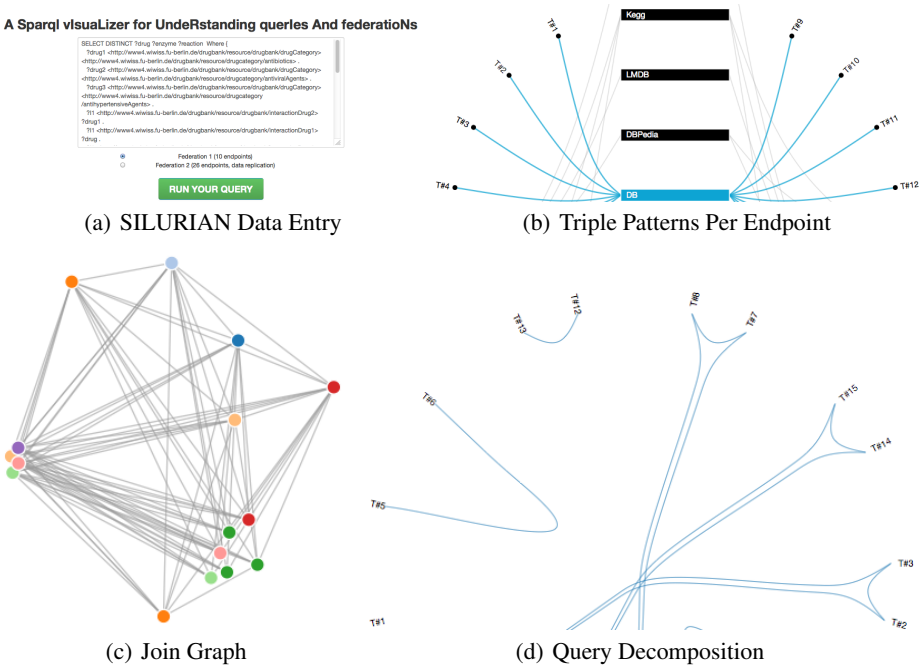
Over the past decade, the number of datasets in the Linking Open Data cloud has exploded as well as the number of SPARQL endpoints. As more linked data becomes available, applications from different domains are frequently developed, and queries that require gathering data from several endpoints are more likely everyday. So far several approaches have addressed the problem of executing federated SPARQL queries on the Web of Data [1, 2, 4]. For example, FedX [4] is a rule-based system able to generate left-linear plans comprised of subqueries that can be exclusively answered by existing endpoints (*Exclusive Groups (EG)*); ANAPSID [1] resorts to source descriptions to determine all the triple patterns that can be executed on the same endpoints and that can be grouped as star-shaped queries; finally, SPLENDID [2] exploits statistics during source selection and query planning to identify the subqueries that will be executed to gather the query answers. Performance of SPARQL queries against these federated engines can be affected by diverse parameters, e.g., number of triple patterns in the query, number of endpoints that can answer a triple pattern, and shape of the query. Analyzing a query and the federation where this query is going to be executed provides the basis not only to understand the performance of a given federated query engine, but also can be useful during query benchmarking. We present SILURIAN a Sparql visualizer for understanding queries and federations. We will demonstrate SILURIAN; attendees will be able to visualize SPARQL queries and understand complexity of both federations and possible plans. The demo is published at `http://choroni.ldc.usb.ve/silurian`.

2 The SILURIAN architecture

SILURIAN is built on top of existing federated engines to visualize plans generated by the engines for a given query and federation of endpoints. In this first version, SILURIAN was built on top of ANAPSID[1], and exploits visualization services implemented by the D3.js JavaScript library[1]. Figures 1(a), (b), (c), and (d) show SILURIAN

[1] `http://d3js.org/`

snapshots; users will be able to introduce their own SPARQL queries and select the federation (Figure 1(a)). Different type of plots will be used to illustrate the properties of queries and federations. Figure 1(b) uses a `Concept Network Browser` plot[2] to illustrate the endpoints that can answer the triple patterns in a query. Figure 1(c) uses `Force-Directed Graph`[3] to visualize a join graph of the input SPARQL query. Each node in the graph represents a triple pattern in the query; an edge between two nodes exists if the corresponding triple patterns do not share a join variable or there is no endpoint in the federation that can answer both triple patterns. Finally, Figure 1(d) relies on a `Hierarchical Edge Bundling`[4] to visualize the decomposition of a query into subqueries of triple patterns; nodes correspond to triple patterns while edges connect triple patterns in the same subquery of the decomposition.



(a) SILURIAN Data Entry (b) Triple Patterns Per Endpoint

(c) Join Graph (d) Query Decomposition

Fig. 1. The SILURIAN snapshots for two Federations of Endpoints on FedBench data collections.

3 Demonstration of Use Cases

We motivate our work by observing how the performance of existing federation engines can be affected during the execution of SPARQL queries with triple patterns bound to

[2] `http://www.findtheconversation.com/concept-map`
[3] `http://bl.ocks.org/mbostock/4062045`
[4] `http://mbostock.github.io/d3/talk/20111116/bundle.html`

predicates of general vocabularies such as RDFS or OWL. These vocabulary terms may occur in almost all data sources, e.g., `rdf:type`, `owl:sameAs`, or `rdfs:seeAlso`; we denominate these terms *general predicates*. We designed a set of three queries q_j $(j = 0...2)$, where q_{i+1} is comprised of more triple patterns bound to general predicates than q_i. First, q_0 retrieves the Kegg compound identifier and among their drugs, those that have a substrate that is an enzyme. Next, q_1 selects drugs that meet q_0 and their `owl:sameAs` link to Drugbank; and finally, q_2 checks that these drugs are also drugs in the DBpedia ontology. Triple patterns bound to *general predicates* are highlighted.

```
q0 Select * WHERE {?d drugbank:keggCompoundId ?c. ?e bio2rdf-kegg:xSubstrate ?c.
                   ?e rdf:type bio2rdf-kegg:Enzyme }
q1 Select * WHERE {?d drugbank:keggCompoundId ?c. ?e bio2rdf-kegg:xSubstrate ?c.
                   ?e rdf:type bio2rdf-kegg:Enzyme.?d owl:sameAs ?d1 .}
q2 Select * WHERE {?d drugbank:keggCompoundId ?c. ?e bio2rdf-kegg:xSubstrate ?c.
                   ?e rdf:type bio2rdf-kegg:Enzyme.?d owl:sameAs ?d1 .
                   ?d1 rdf:type dbpedia-owl:Drug .}
```

An experiment was set up in order to evaluate the performance of different federated SPARQL query engines: FedX, SPLENDID, and ANAPSID. Queries q_0, q_1, and q_2 were executed on 26 Virtuoso endpoints that locally access the FedBench collections[5], November 2011. Each collection was assigned to one Virtuoso endpoint, except *Geonames* and DBpedia that were fragmented to impact on the performance of the query decomposition techniques. *Geonames* was horizontally partitioned into eleven fragments and each fragment was assigned to a different endpoint. Additionally, each of the DBpedia files was made available through a different SPARQL endpoint, i.e., DBpedia was vertically partitioned. This study was executed on a Linux Mint machine with an Intel Pentium Core 2 Duo E7500 2.93GHz 8GB RAM 1333MHz DDR3. We could observe that the performance of all these engines is deteriorated as the number of triple patterns on general predicates increases. Based on these results, we formulated the following research questions: *1*) is the observed behavior due to limitations of these federation engines?, or *2*) is this behavior caused by the properties of these queries?. We will visualize the characteristics of queries and federations that provide evidences to answer our research questions. FedBench 10 collections: DBpedia, NY Times, Geonames, KEGG, ChEBI, Drugbank, Jamendo, LinkedMDB, SW Dog Food, and SP2B-10M, were integrated into two federations of endpoints. $\texttt{Fed}_1$ comprises the previously explained 26 Virtuoso[6] endpoints, and $\texttt{Fed}_2$ is composed of 10 endpoints, one per FedBench collection. In both federations, Virtuoso timeout was set up to 300 secs. or 100,000 tuples. Different criteria to decompose SPARQL queries into subqueries answerable by existing endpoints will be demonstrated; e.g., Exclusive Groups (EG) [2, 4] , Star-Shaped Group Single endpoint selection (SSGS), and Star-Shaped Group Multiple endpoint selection (SSGM) [3]. We will demonstrate the following use cases:

Effects of number of triple patterns bound to general predicates. We will demonstrate that in queries as the ones presented in the previous example, almost all the endpoints in the federation can instantiate variables in the triple patterns of the query. Particularly, triple patterns bound to `owl:sameAs` could be answerable for 24 out of 26

[5] `http://fedbench.fluidops.net`

[6] `http://virtuoso.openlinksw.com/`, November 2011.

endpoints of $\texttt{Fed}_1$ and all the endpoints of $\texttt{Fed}_2$. Federated engines may have to consider all these endpoints to produce a complete answer of the query.

Effects of the number of triple patterns and shape of the query. Attendees will observe that in queries will a large number of triple patterns that comprised star-shaped or chain-shaped subqueries, the space of possible plans of the query may exponentially explode. For example, we will show queries comprise of 46 triple patterns which can be decomposed into 9 star-shaped subqueries, which could not be executed in any of existing federated engines in less than 30 minutes. These queries may constitute challenges for federation engines and should be included in future benchmarks.

Effects of the data fragmentation and replication. The aim of this use case is to show the effects of data fragmentation and replication in the complexity of SPARQL federated queries. In federation $\texttt{Fed}_1$, data in Geonames is horizontally partitioned while DBpedia is vertically fragmented. Attendees will observe that the number of relevant endpoints increases according to fragments of data are made available from different endpoints of a federation. For example, in queries with triple patterns bound to predicates in Geonames the number of relevant endpoints is larger in $\texttt{Fed}_1$ than in $\texttt{Fed}_2$. Because Geonames data is horizontally partitioned, many of the relevant data may not actually provide the instantiations of the variables required to execute the query. Thus, federated engines have to either contact all the endpoints to decide which one can execute the corresponding subqueries or simply pay the price of executing the subquery in all of them, and the execution of these queries can be costly. These queries may be challenging for existing federation engines and should be included in future benchmarks.

4 Conclusions

SILURIAN visualizes SPARQL federated queries as well as the properties of the federations that may impact on the complexity of these queries. Particularly, SILURIAN helps to understand why data fragmentation and replication among different endpoints, shape of the queries and the type of predicates in the triple patterns, may affect the performance of a federated query engine. Because main sources of query complexity can be analyzed, SILURIAN provides the basis for understanding the behavior of existing engines and may help during the design of benchmarks to evaluate these engines.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. In *ISWC*, pages 18–34, 2011.
2. O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *COLD*, Bonn, Germany, 2011.
3. G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012.
4. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, pages 601–616, 2011.