DEFENDER: A DEcomposer for quEries agaiNst feDERations of Endpoints

Gabriela Montoya[1(✉)], Maria-Esther Vidal[1(✉)], and Maribel Acosta[1,2]

[1] Universidad Simón Bolívar, Caracas, Venezuela
{gmontoya,mvidal,macosta}@ldc.usb.ve
[2] Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany
maribel.acosta@kit.edu

Abstract. We present DEFENDER and illustrate the benefits of identifying promising query decompositions and efficient plans that combine results from federations of SPARQL endpoints. DEFENDER is a query decomposer that implements a two-fold approach. First, triple patterns in a SPARQL query are decomposed into simple sub-queries that can be completely executed on one endpoint. Second, sub-queries are combined into a feasible bushy tree plan where the number of joins is maximized and the height of tree is minimized. We demonstrate DEFENDER and compare its performance with respect to state-of-the-art RDF engines for queries of diverse complexity, networks with different delays, and dataset differently distributed among a variety of endpoints.

1 Introduction

During the last years, the number of datasets in the Linked Open Data cloud has exploded as well as the number of SPARQL endpoints that provide access to these datasets[1]. Although existing endpoints should be able to execute any SPARQL query, some endpoints reject the execution of queries whose estimated execution time or cardinality is greater than a certain number, while others simply time out without producing any answer. With the appropriate endpoint technology not ready, there is a need to develop techniques to decompose complex queries into queries that can be executed as well as strategies to integrate retrieved data. We present DEFENDER, a decomposer for queries against federations of endpoints that stores information about the available endpoints and the ontologies used to describe the data accessible through the endpoints, and decomposes queries into sub-queries that can be executed by the selected endpoints. Additionally, DEFENDER combines sub-queries into an execution plan where the number of joins is maximized and the height is minimized. The former condition implies that the number of Cartesian products is minimized, while the latter benefits the generation of plans where leaves can be independently executed. DEFENDER was implemented on top of ANAPSID [1], an adaptive query engine for the SPARQL 1.1 federation extension[2] that adapts query execution

[1] http://labs.mondeca.com/sparqlEndpointsStatus/.
[2] http://www.w3.org/TR/rdf-sparql1-query/.

DOI: 10.1007/978-3-662-46641-4_49

schedulers to data availability and runtime conditions. We demonstrate the performance of the plans identified by DEFENDER, and show that these plans are competitive with the plans generated by existing RDF engines. A portal that publishes results presented at the demo section can be found at http://code.google.com/p/defender-portal/.

2 The DEFENDER Architecture

DEFENDER comprises a *Query Planner*, an *Adaptive Query Engine* and a *Catalog of Endpoint Descriptions*. The DEFENDER *Query Planner* is composed of two main components: the *Query Decomposer* and the *Heuristic-Based Query Optimizer*. The former divides sets of triple patterns in SPARQL 1.0 queries into sub-sets of triple patterns (TPs) that can be executed by the same endpoint and: *(i)* share exactly one variable, or *(ii)* share one variable with at least one of the TPs in the sub-query. The query decomposer begins creating single sub-queries with TPs, then it merges the sub-queries that share exactly one variable, and repeats this process until a fixed-point is reached in the process of creating the sub-queries. Then, TPs that share one variable with any TPs are added (Fig. 1).

Once the SPARQL 1.1 query is created, heuristic-based optimization techniques are followed to generate a bushy tree plan, where the leaves correspond to the sub-queries of TPs previously identified. Optimization techniques do not rely on statistics recollected from the endpoints, just information about predicates in the datasets accessible through the endpoints. A greedy heuristic-based algorithm is implemented; it traverses the space of bushy plans in iterations and outputs a bushy tree plan of the SPARQL 1.1 query with the service clause where the number of joins is maximized and the height of tree is minimized. Thus, the size of intermediate results and the number of HTTP requests are reduced.

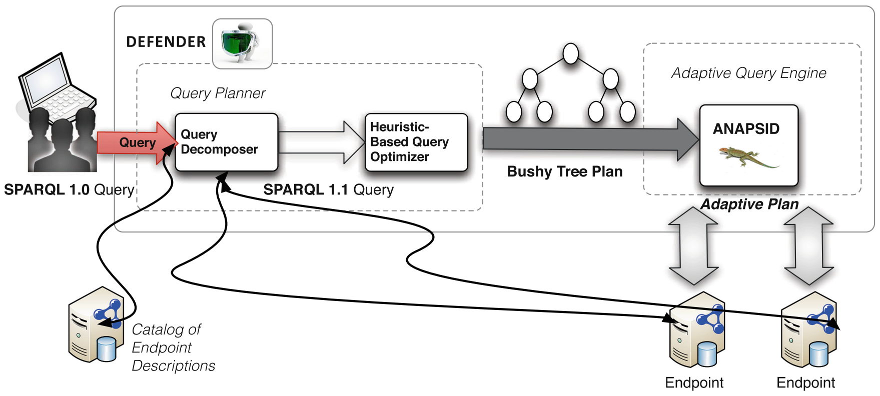

Fig. 1. The DEFENDER architecture

3 Demonstration of Use Cases

Consider the following SPARQL 1.0 query: *Retrieve diseases and genes associated with drugs tested in clinical trials where Prostate Cancer was studied.*

```
SELECT DISTINCT ?II ?D ?GN2
WHERE {
  ?CT1<http://data.linkedct.org/resource/linkedct/condition> ?C1 .
  ?CT1 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?I <http://data.linkedct.org/resource/linkedct/intervention_type> "Drug" .
  ?C1 <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?D.
  ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II.
  ?C <http://data.linkedct.org/resource/linkedct/condition_name> "Prostate Cancer" .
  ?CT <http://data.linkedct.org/resource/linkedct/intervention> ?I .
  ?CT <http://data.linkedct.org/resource/linkedct/condition> ?C .
  ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/associatedGene> ?GN2 .
  ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/possibleDrug> ?II}
```

The answer is composed of 192 tuples when data from Diseasome and LinkedCT are retrieved. However, if the query is run against existing endpoints, Diseasome[3] or LinkedCT[4], the answer is empty. This problem is caused by the need to traverse links between these datasets to answer the query. Still, the majority of existing endpoints have been created for lightweight use and they are not able to dereference data from other datasets. Existing approaches [2,4] are able to decompose this query into sub-queries; although these approaches can be very efficient and effective, if queries are comprised of a large number of triple patterns that can be executed by different endpoints, they may time out without producing any answer. To overcome these limitations, DEFENDER decomposes the former query into the following SPARQL 1.1 query with the service clause, which is comprised of four *star-shaped* sub-queries. These sub-queries are composed of TPs that can be executed by the same endpoint, and that share exactly one variable or share one variable with at least one of the TPs in the sub-query.

```
SELECT ?II ?D ?GN2
WHERE {
   { SERVICE <http://www4.wiwiss.fu-berlin.de/diseasome/sparql> {
      ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/associatedGene> ?GN2 .
      ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/possibleDrug> ?II. } }.
   { SERVICE <http://linkedct.org/sparql> {
      ?I <http://data.linkedct.org/resource/linkedct/intervention_type> "Drug" .
      ?CT1 <http://data.linkedct.org/resource/linkedct/condition> ?C1 .
      ?CT1 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
      ?C1 <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?D. }} .
   { SERVICE <http://linkedct.org/sparql> {
      ?C <http://data.linkedct.org/resource/linkedct/condition_name> "Prostate Cancer" .
      ?CT <http://data.linkedct.org/resource/linkedct/condition> ?C. }}.
   { SERVICE <http://linkedct.org/sparql> {
      ?I <http://data.linkedct.org/resource/linkedct/intervention_type> "Drug" .
      ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II .
      ?CT <http://data.linkedct.org/resource/linkedct/intervention> ?I }}. }
```

Once the query is decomposed, DEFENDER builds a plan that combines the sub-queries; the generated plan minimizes intermediate results and the number

[3] http://www4.wiwiss.fu-berlin.de/diseasome/sparql.

[4] http://data.linkedct.org/sparql/.

of HTTP requests. The DEFENDER portal[5] presents the behavior of 36 queries against the FedBench collections: Cross-Domain, Linked Data and Life Science [3]. These queries include 25 FedBench queries and 11 complex queries[6]; complex queries are comprised of between 6 and 48 triple patterns and can be decomposed into up to 8 sub-queries. FedBench collections were accessed through 9 Virtuoso[7] endpoints which time out at 240 s. or 71,000 tuples. Endpoint simulators were used to configured network latency, endpoint availability and network packet size; simulators are comprised of servers and proxies. Servers correspond to real endpoints that are contacted by the proxies, which send data between servers and RDF engines following a particular transfer delay and respecting a given network packet size. Different types of delays are illustrated; all of them follow a Gamma distribution with different average latency to simulate perfect, fast, and medium-fast, and set up the network packet size. Additionally, we produced SPARQL 1.1 queries for different decompositions and executed these queries in ARQ 2.8.8. BSD-style[8] that supports the Federation extension of SPARQL 1.1. The behavior of ARQ, DEFENDER, and FedX [4] is demonstrated in these network configurations. Also, it can be observed the impact of different decompositions on performance and answer completeness.

Effects of network delays on query execution performance. In an ideal network without delays, it can be seen that ARQ may time out without producing any answer, while DEFENDER may be able to finalize the query processing task before reaching a timeout of 1,800 s. On the other hand, delays may considerably affect the performance of DEFENDER and ARQ depending on the type of decomposition. For example, the majority of queries may either time out or produce empty answers when unitary sub-queries are executed, i.e., when sub-queries are comprised of only one triple pattern. In contrast, plans comprised of non-unitary sub-queries, i.e., the ones identified by DEFENDER, are not equality affected by network delays. Although DEFENDER performance can be deteriorated, execution time of around half of the queries remain in the same order of magnitude with respect to these queries executed in a perfect network. ARQ is also able to execute some of these plans in the delayed networks without decreasing performance significantly. The observed behavior of the plans comprised of DEFENDER sub-queries is caused by a reduced number of HTTP requests as well as the size of intermediate results which usually can be delivered from the endpoints in a small number of network packets. Thus, even in presence of delayed networks, the performance of these plans is acceptable.

Answer completeness when different decompositions are executed. In a perfect network, DEFENDER and ARQ produce all the answers for the majority of the queries before timing out. But, when delays are considered the quality is decreased, mainly when plans are comprised of unitary

[5] http://code.google.com/p/defender-portal/.
[6] http://www.ldc.usb.ve/~mvidal/FedBench/queries/ComplexQueries.
[7] http://virtuoso.openlinksw.com/.
[8] http://sourceforge.net/projects/jena/.

sub-queries are executed. These results are consequence of the poor performance exhibited by both engines when unitary sub-queries are run. However, if intermediate results remain small, quality is not equally affected in DEFENDER plans even in presence of network latency. We also show the scenario where the same predicate is accessible through different endpoints and demonstrate how dataset distributions impact on answer completeness.

Effects of the plan shape on execution time and answer completeness. Optimal bushy trees and left-linear plans are reported for each query and executed in DEFENDER. These plans may reduce execution time by up to one order of magnitude when optimal bushy trees are executed. FedX also exhibits good performance when FedBench queries are executed, being able to produce most of the answers. In contrast, DEFENDER plans may outperform the ones generated by FedX when the queries are comprised of a large number of triple patterns. Bushy trees are able to scale up to complex queries, and are competitive with other execution strategies when simple queries are processed. Finally, the execution time of plans comprised of DEFENDER sub-queries is low; these plans may reduce by up to two orders of magnitude the time consumed by plans comprised of unitary sub-queries.

4 Conclusions and Future Work

We present DEFENDER and illustrate results that suggest that our proposed techniques may reduce execution times by up to two orders of magnitude, and are able to produce answers when other engines fail. Also, depending on data distributions among different endpoints and transfer delays, DEFENDER query plans overcome plans generated by existing RDF engines if size of intermediate results and the number of HTTP requests are reduced. In the future we plan to provide DEFENDER endpoint for real-world applications.

References

1. Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 18–34. Springer, Heidelberg (2011)
2. Görlitz, O., Staab, S.: SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Proceedings of the 2nd International Workshop on Consuming Linked Data, Bonn, Germany (2011)
3. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: FedBench: a benchmark suite for federated semantic data query processing. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 585–600. Springer, Heidelberg (2011)
4. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: optimization techniques for federated query processing on linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)