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, Venezuela
{gmontoya, mvidal, macosta}@ldc.usb.ve
[2] Institute AIFB, Karlsruhe Institute of Technology, 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 Cloud of Linked Data 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 schedulers to

[1] http://labs.mondeca.com/sparqlEndpointsStatus/
[2] http://www.w3.org/TR/rdf-sparql11-query/

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.

2 The DEFENDER Architecture

DEFENDER comprises a *Query Planner*, an *Adaptive Query Engine* and a *Catalog of Endpoint Descriptions*. DEFENDER current version is built on top of ANAPSID, a SPARQL 1.1 adaptive query engine that opportunistically produces results even when endpoints become blocked while sending data. The DEFENDER *Query Planner* is composed of two main components: the *Query Decomposer* and the *Heuristic-Based Query Optimizer*. The former divides set of triple patterns in SPARQL 1.0 queries into sub-sets of triple patterns that will be executed against the available endpoints; it implements a greedy algorithm to group in the same sub-query the triple patterns that share one variable and can be executed by the same endpoint. The query decomposer begins creating single sub-queries with triple patterns, 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.

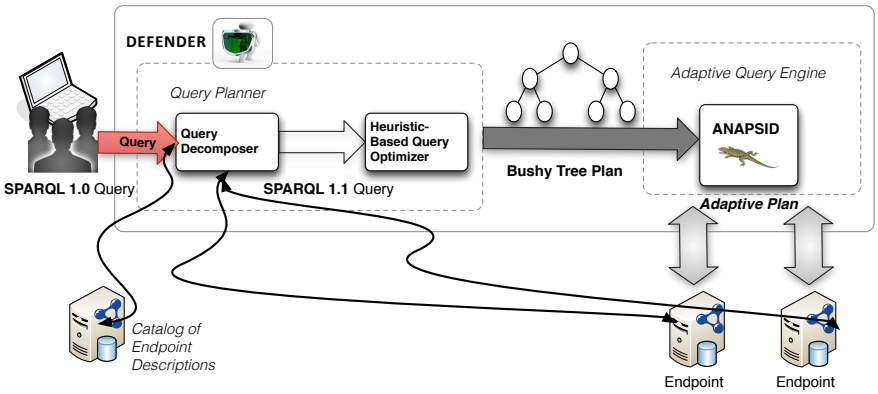

Fig. 1. The DEFENDER Architecture

Once the query is rewritten in SPARQL 1.1, heuristic-based optimization techniques are followed to generate a bushy tree plan, where the leaves correspond to the sub-queries previously identified. Optimization techniques do not rely on statistics recollected from the endpoints, just information about predicates in the dataset accessible through the endpoint. 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 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.

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.*

```
(0) SELECT DISTINCT ?II ?D ?GN2
(1) WHERE {
(2)    ?CT1<http://data.linkedct.org/resource/linkedct/condition> ?C1 .
(3)    ?CT1 <http://data.linkedct.org/resource/linkedct/intervention> ?I .
(4)    ?I <http://data.linkedct.org/resource/linkedct/intervention_type> "Drug" .
(5)    ?C1 <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?D.
(6)    ?I <http://www.w3.org/2000/01/rdf-schema#seeAlso> ?II.
(7)    ?C <http://data.linkedct.org/resource/linkedct/condition_name> "Prostate Cancer" .
(8)    ?CT <http://data.linkedct.org/resource/linkedct/intervention> ?I .
(9)    ?CT <http://data.linkedct.org/resource/linkedct/condition> ?C .
(10)   ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/associatedGene> ?GN2 .
(11)   ?D <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/possibleDrug> ?II}
```

The result set is composed of 192 tuples when data from the datasets 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. DEFENDER decomposes the former query into the following SPARQL 1.1 query which is comprised of four sub-queries:

```
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 of HTTP requests. During the demonstration, attendees will be able to observe the

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

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

behavior of any of the 29 queries against the FedBench collections: cross-domain, linked data and life science [2]. These queries include 24 FedBench queries and 5 conjunctive queries, i.e., graph pattern queries[5]; conjunctive queries are comprised of between 6 and 16 triple patterns and can be decomposed into up to 7 sub-queries. FedBench collections will be accessed through 8 Virtuoso[6] endpoints which time out at 240 secs. or 71,000 tuples. Endpoint simulators will be used to configure data transfer delays, endpoint availability and network packet size; this simulator is 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. We will consider three types of delays and run 24 instances of this script that will be listened on different ports. The delay will follow a gamma distribution with different average latency to simulate fast, medium fast, and slow networks, and set up different network packet sizes. Additionally, we produced SPARQL 1.1 queries for these decompositions and executed these queries in ARQ 2.8.8. BSD-style[7] that supports the Federation extension of SPARQL 1.1. We will demonstrate the behavior of ARQ, DEFENDER, and FedX [3] in these network configurations. Also, we will illustrate the impact of different decompositions and plans on the performance of these engines and the completeness of their answers. The attendees will observe:

Effects of network delays on query execution performance. In an ideal network without delays, we will observe 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 300 secs. On the other hand, delays considerably may 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. On the other hand, when delays

[5] `http://www.ldc.usb.ve/~mvidal/FedBench/queries/ComplexQueries`

[6] `http://virtuoso.openlinksw.com/`

[7] `http://sourceforge.net/projects/jena/`

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

Effects of the plan shape on execution time and answer completeness. Optimal bushy trees, left-linear plans and naive bushy trees[8] will be generated for each query and executed in ARQ, DEFENDER and FedX. DEFENDER 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 large or complex conjunctive queries, and are competitive with other execution strategies when simple queries are processed. Finally, the execution time of plans comprised of the sub-queries identified by DEFENDER is low; these plans may reduce by up to two orders of magnitude the time consumed by plans comprised of unitary sub-queries in ARQ and DEFENDER.

4 Conclusions

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.

References

1. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, pages 18–34, Berlin, Heidelberg, 2011. Springer-Verlag.
2. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference (1)*, pages 585–600, 2011.
3. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, pages 601–616, 2011.

[8] A naive bushy tree is a binary tree where Cartesian products can be presented.