Federated SPARQL Query Processing Via CostFed

Alexander Potocki[1,2], Muhammad Saleem[2], Tommaso Soru[2],
Olaf Hartig[3], Martin Voigt[1], and Axel-Cyrille Ngonga Ngomo[2,4]

[1] Ontos GmbH, {alexander.potocki,martin.voigt}@ontos.com
[2] AKSW, Germany, {saleem,tsoru,ngonga}@informatik.uni-leipzig.de
[3] IDA, Linköping University, Sweden, olaf.hartig@liu.se
[4] University of Paderborn, Germany, axel.ngonga@upb.de

Abstract. Efficient source selection and optimized query plan generation belong to the most important optimization steps in federated query processing. This paper presents a demo of CostFed, an index-assisted federation engine for federated SPARQL query processing. CostFed's source selection and query planning is based on the index generated from the SPARQL endpoints. The key innovation behind CostFed is that it considers the skew distribution of the resources to perform efficient source selection and cost-based query planning. Our experiments on the FedBench benchmark that CostFed on average is 3 to 121 times faster than the state of the art.

1 Introduction

Answering complex queries on the Web of data often requires merging partial results contained across different data sources. The optimization of engines that support this type of queries, called *federated query engines*, is thus of central importance for the efficient and scalable deployment of Semantic Web technologies. Current cost-based SPARQL endpoint federation approaches [2,4,9] assume that the resources pertaining to a predicate are uniformly distributed. Hence, they make use of average selectivities to estimate the cardinality of triple patterns. However, in reality the resources are not uniformly distributed in RDF datasets [3]. Our analysis[5] of the well-known federation benchmark named FedBench [7] confirms that the FedBench resources are not uniformly distributed. The downside of using average selectivities for triple patterns cardinality estimation is that it can lead to poor cardinality estimation when a high-frequency resource (i.e., a resource that occurs in a large number of triples) is used in that triple pattern. Consequently, the query planning can be significantly affected as suggested by our evaluation (see Section 2).

CostFed is an index-assisted SPARQL endpoint federation engine. CostFed's *query planning* is based on estimating query costs by using selectivity information stored in an index. In contrast to the state of the art, CostFed takes the skew in distribution of subjects and objects across predicates into account. In addition, CostFed extends the join-aware source selection technique introduced in HiBISCuS [6]. Our join implementation is based on both bound [1] and symmetric hash joins. A comparison of CostFed with state-of-the-art federation engines (ANAPSID [1], SemaGrow [2], SPLENDID [4], HiBISCuS

[5] FedBench analysis: https://github.com/AKSW/CostFed/tree/master/stats

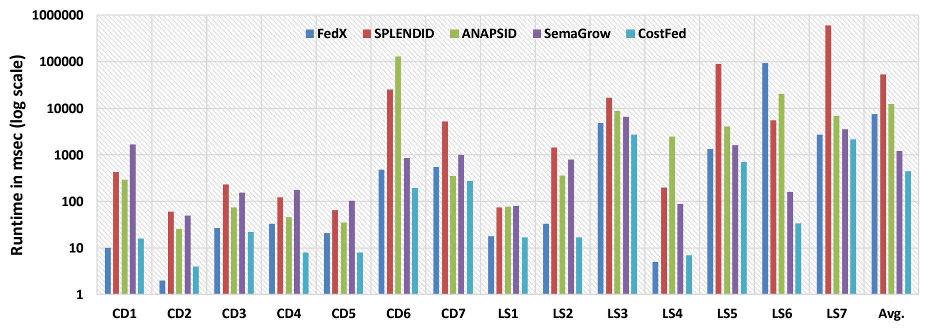


Fig. 1: Comparison of the query execution time on FedBench

[6], and FedX [8]) show that we outperform these engines on the overall query runtime on the majority of the FedBench [7] queries.

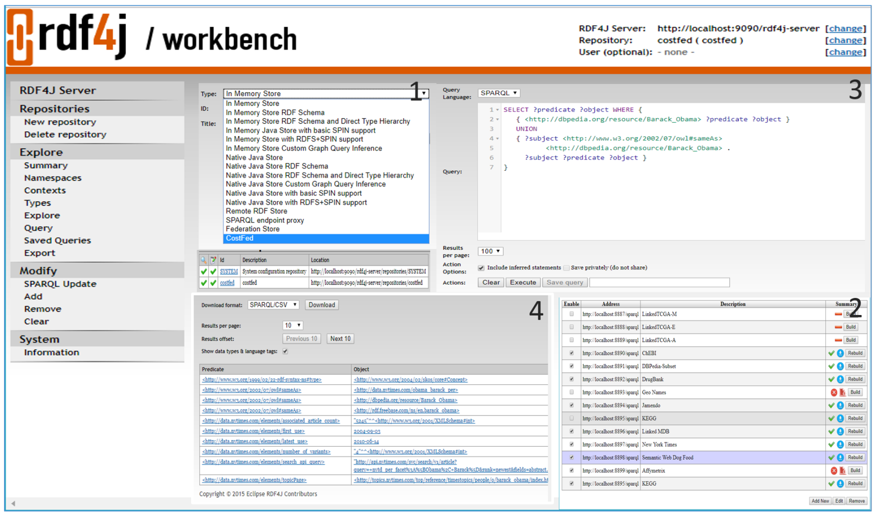
2 Evaluation Results

We used FedBench [7] for evaluation which comprises 25 queries, 14 of which (CD1-CD7, LS1-LS7) are for SPARQL endpoint federation approaches (the other 11 queries (LD1-LD11) are for Linked Data federation approaches [5]). As CostFed is a SPARQL endpoint federation, we used all 14 SPARQL endpoint federation queries in our evaluation.

The query execution time is often used as key metric to compare federation engines. Herein, we consider the query execution time to be the time necessary to gather all the results from the result set iterator of each engine. Figure 1 shows the runtime performance of the state-of-the-art SPARQL endpoint federation engines. Overall, CostFed clearly outperforms the other selected systems. On FedBench, CostFed is better than FedX on 11/14 queries and outperforms SPLENDID, ANAPSID and SemwGrow on all 14 queries. CostFed's average runtime across all 14 FedBench queries is only 440ms while FedX needs 7,468ms (i.e., 16 times the runtime of CostFed), SPLENDID's is 5,3404ms (i.e., 121× slower than CostFed), ANAPSID's is 12,467ms (i.e., 28× that of CostFed), and SemaGrow's is 1,203ms (i.e., 3× slower than CostFed). Since the execution times for the FedBench queries are very small, i.e., less than 3 seconds on CostFed, the average runtime performance for a system is greatly affected if a particular query takes too long. For example, FedX takes 94,519ms to execute LS6, due to which overall runtime performance is greatly decreased comparing to CostFed. If we remove the LS6 runtime, then FedX's average (across the remaining 13 queries) runtime is 771 ms (2× CostFed's).

3 CostFed Online

The CostFed online demo along with the source code is available from the CostFed homepage `https://github.com/AKSW/costfed`.


rdf4j / workbench
RDF4J Server
Repositories
New repository
Delete repository
Explore
Summary
Namespaces
Contexts
Types
Explore
Query
Saved Queries
Export
Modify
SPARQL Update
Add
Remove
Clear
System
Information
1
2
3
4
In Memory Store
In Memory Store RDF Schema
In Memory Store RDF Schema and Direct Type Hierarchy
In Memory Java Store with basic SPIN support
In Memory Store with RDFS+SPIN support
In Memory Store Custom Graph Query Inference
Native Java Store
Native Java Store RDF Schema
Native Java Store RDF Schema and Direct Type Hierarchy
Native Java Store Custom Graph Query Inference
Native Java Store with basic SPIN support
Native Java Store with RDFS+SPIN support
Remote RDF Store
SPARQL endpoint proxy
Federation Store
CostFed
Clear
Execute
Save query
Download
Previous 10
Next 10
Add New
Edit
Remove

RDF4J Server: http://localhost:9090/rdf4j-server [change]
Repository: costfed (costfed) [change]
User (optional): - none - [change]

Type: In Memory Store
ID:
Title:

		Id	Description	Location
✓	✓	SYSTEM	System configuration repository	http://localhost:9090/rdf4j-server/repositories/SYSTEM
✓	✓	costfed	costfed	http://localhost:9090/rdf4j-server/repositories/costfed

Query Language: SPARQL

Query:

```
SELECT ?predicate ?object WHERE {
  { <http://dbpedia.org/resource/Barack_Obama> ?predicate ?object }
  UNION
  { ?subject <http://www.w3.org/2002/07/owl#sameAs>
        <http://dbpedia.org/resource/Barack_Obama> .
    ?subject ?predicate ?object }
}
```

Results per page: 100
Action Options: [✓ Include inferred statements] Save privately (do not share)
Actions:

Download format: SPARQL/CSV
Results per page: 10
Results offset:
Show data types & language tags: ✓

Predicate	Object
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
<http://www.w3.org/2002/07/owl#sameAs>	<http://data.nytimes.com/obama_barack_per>
<http://www.w3.org/2002/07/owl#sameAs>	<http://dbpedia.org/resource/Barack_Obama>
<http://www.w3.org/2002/07/owl#sameAs>	<http://rdf.freebase.com/ns/en.barack_obama>
<http://data.nytimes.com/elements/associated_article_count>	"5245"^^<http://www.w3.org/2001/XMLSchema#int>
<http://data.nytimes.com/elements/first_use>	2004-09-03
<http://data.nytimes.com/elements/latest_use>	2010-06-14
<http://data.nytimes.com/elements/number_of_variants>	"4"^^<http://www.w3.org/2001/XMLSchema#int>
<http://data.nytimes.com/elements/search_api_query>	"http://api.nytimes.com/svc/search/v1/article?query=+nytd_per_facet%3A%5BObama%2C+Barack%5D&rank=newest&fields=abstract,
<http://data.nytimes.com/elements/topicPage>	<http://topics.nytimes.com/top/reference/timestopics/people/o/barack_obama/index.ht

Copyright © 2015 Eclipse RDF4J Contributors

Enable	Address	Description	Summary
☐	http://localhost:8887/sparql	LinkedTCGA-M	Build
☐	http://localhost:8888/sparql	LinkedTCGA-E	Build
☐	http://localhost:8889/sparql	LinkedTCGA-A	Build
✓	http://localhost:8890/sparql	ChEBI	✓ Rebuild
✓	http://localhost:8891/sparql	DBPedia-Subset	✓ Rebuild
✓	http://localhost:8892/sparql	DrugBank	✓ Rebuild
☐	http://localhost:8893/sparql	Geo Names	✗ Build
✓	http://localhost:8894/sparql	Jamendo	✓ Rebuild
☐	http://localhost:8895/sparql	KEGG	✓ Rebuild
✓	http://localhost:8896/sparql	Linked MDB	✓ Rebuild
✓	http://localhost:8897/sparql	New York Times	✓ Rebuild
✓	http://localhost:8898/sparql	Semantic Web Dog Food	✓ Rebuild
✓	http://localhost:8899/sparql	Affymetrix	✗ Build
✓	http://localhost:8895/sparql	KEGG	✓ Rebuild

Fig. 2: The CostFed Online Interface

3.1 Interface

Figure 2 shows the online interface[6] of the CostFed which comprise of four main steps:

1. **Create CostFed repository**: The first step is to create CostFed repository. The user has to select 'CostFed' from the drop down menu. The rest of the process is exactly the same as creating a new RDF4J repository.
2. **Endpoint manager**: The second step is to register the set of SPARQL endpoints over which the given SPARQL query will be executed. Beside registering new SPARQL endpoints, the manager allows to enable/disable the given endpoint and generate CostFed index for the given endpoint. Please note that as mentioned before CostFed is index-assisted approach. Therefore, all indexes should be created first before running the SPARQL queries. The generated indexes can be downloaded as a Turtle file.
3. **Running queries**: The third step is to write SPARQL query and run it over given set of enabled SPARQL endpoints. Note that CostFed allows running both federated and non-federated SPARQL queries.
4. **Results**: The results of the query executed in step (3) will be shown in windows 4. Exactly same like RDF4J the results can be downloaded in different formats, e.g., CSV, TSV, etc. Also the number of results per page can be set.

3.2 Implementation

The CostFed demo comprises two main web applications: 1) the endpoint manager which manages the SPARQL endpoints and 2) the query executor which enables user to write

[6] Online at `http://costfed.aksw.org`.

and execute SPARQL queries. The front end of the endpoint manager was developed using the VueJs[7] framework while the backend is implemented as a Java servlet. The application uses a file system as a storage for SPARQL endpoints descriptors, errors, and indexes. Each descriptor is represented as a standard Java property file. The files contain all current endpoint parameters, including the state of the summary generation procedure. The SPARQL query executor application is developed on top of the well-known RDF4J[8] Workbench. Basically, it is an extension to the RDF4J workbench where we embedded CostFed in the form of an RDF4J repository.

4 Conclusion

We presented CostFed, a federated engine for SPARQL endpoint federation. CostFed implements innovative solutions for the selection of sources, the estimation of cardinalities and the planning of queries. We evaluated our approach against state-of-the-art federation systems.

Acknowledgements

This work was supported by the H2020 project HOBBIT (no. 688227), the BmBF project DIESEL (no. 01QE1512C), the BMWi project GEISER (no. 01MD16014), and by the BMWi project SAKE (no. 01MD15006E). Olaf Hartig's work was funded by the CENIIT program at Linköping University (project no. 17.05).

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*, 2011.
2. A. Charalambidis, A. Troumpoukis, and S. Konstantopoulos. Semagrow: Optimizing federated sparql queries. In *SEMANTICS*, 2015.
3. S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: a comparison of rdf benchmarks and real rdf datasets. In *ACM SIGMOD*, 2011.
4. O. Görlitz and S. Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *COLD at ISWC*, 2011.
5. M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A.-C. N. Ngomo. A fine-grained evaluation of sparql endpoint federation systems. *SWJ*, 2015.
6. M. Saleem and A.-C. Ngonga Ngomo. HiBISCuS: Hypergraph-based source selection for sparql endpoint federation. In *ESWC*, 2014.
7. M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: a benchmark suite for federated semantic data query processing. In *ISWC*, 2011.
8. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, 2011.
9. X. Wang, T. Tiropanis, and H. C. Davis. Lhd: Optimising linked data query processing using parallelisation. In *LDOW at WWW*, 2013.

[7] VueJs: `https://vuejs.org/v2/guide/`
[8] RDF4J formally known as Sesame: `http://RDF4J.org/`