Optimizing SPARQL Queries over Disparate RDF Data Sources through Distributed Semi-joins

Jan Zemánek
Department of Information and Knowledge Engineering
University of Economics, Prague
Czech Republic
xzemj22@vse.cz

Simon Schenk
Institute for Computer Science
University of Koblenz
Germany
sschenk@uni-koblenz.de

Vojtěch Svátek
Department of Information and Knowledge Engineering
University of Economics, Prague
Czech Republic
svatek@vse.cz

ABSTRACT
With the ever-increasing amount of data on the Web available at SPARQL endpoints [1] the need for an integrated and transparent way of accessing the data has arisen. It is highly desirable to have a way of asking SPARQL queries that make use of data residing in disparate data sources served by multiple SPARQL endpoints. We aim at providing such a capability and thus enabling an integrated way of querying the whole Semantic Web at a time.

Keywords
SPARQL, RDF, Distributed Querying

1. MOTIVATION
Imagine the following SPARQL query asking for all co-authors of Tim Berners-Lee's publications and their birthdates and relying on data from two data sources (DBLP and DBpedia) residing at two disparate SPARQL endpoints:

```
SELECT ?coauthor ?birthdate
FROM NAMED <http://www4.wiwiss.fu-berlin.de/dblp/>
FROM NAMED <http://dbpedia.org/>
WHERE {
    GRAPH <http://www4.wiwiss.fu-berlin.de/dblp/> {
        ?paper dc:creator <http://www4.wiwiss.fu-
berlin.de/dblp/resource/person/100007>.
        ?paper dc:creator ?coauthor.
        ?coauthor foaf:name ?name. }
    GRAPH <http://dbpedia.org/> {
        ?person foaf:name ?name.
        ?person dbpedia:birth ?birthdate. }
}
```

(1)

At the moment there are two ways how to deal with such a query: either to copy the content of both data sources into a local RDF repository and to issue the query against it, or to issue two separate queries each to the corresponding SPARQL endpoint, and to treat the received results programmatically. The former solution may be unfeasible at all due to the amount of data involved or may result in the local copy of data soon being out of date. The latter imposes further burden on the user.

Our solution allows to directly issue the query as mentioned above, thus relieving the user from the burden of maintaining the local copy of the data or consolidating the results.

2. STATE OF THE ART
There have been presented some solutions to the problem of querying the data sources at multiple SPARQL endpoints in an integrated manner, namely, DARQ [2] (http://darq.sourceforge.net/) and SemWIQ [3] (http://semwiq.faw.uni-linz.ac.at/). These approaches however require either supplying of statistics about SPARQL endpoints or registration of SPARQL endpoints in a catalog, respectively. Both of them then use statistics about SPARQL endpoints which are supplied or in case of SemWIQ generated dynamically in order to determine where to send sub-queries. DARQ and even SemWIQ impose restrictions on the expressivity of the SPARQL constructs used.

3. OUR SOLUTION
Our approach is slightly different, as we let the user determine at which SPARQL endpoint the triple patterns should be evaluated. We don't impose any restrictions on the SPARQL expressivity supported, and our solution is fully compatible with all current SPARQL-compliant endpoints.

4. IMPLEMENTATION AND OPTIMIZATION
We have built our solution called *Distributed SPARQL* [4] on the top of the Sesame RDF repository by extending Sesame's SAIL by our own component Distributed SAIL. The preliminary version of *Distributed SPARQL* was presented in [4]; the currently implemented version however addresses performance issues through substantial optimization, using the distributed semi-join operation.

4.1 Remote Queries
We have extended Sesame's SPARQL algebra model and enriched Sesame's SPARQL query tree by a special node called "Remote Node". Remote Node is a wrapper for a triple pattern which is to be evaluated remotely at a given SPARQL endpoint. The received results are then employed in the next steps of query evaluation. As mentioned before, the user determines to which SPARQL endpoint the sub-queries should be sent. This is now done in a configuration file, where the user is supposed to associate graph names with respective SPARQL endpoints at which they reside. (E.g.: graph name "http://dbpedia.org/" in our example above is a reference to DBpedia's SPARQL endpoint at "http://dbpedia.org/sparql")

4.2 Semi-joins

In order to lower the communication costs involved in sending and receiving SPARQL queries and results over the Internet, we implemented a distributed semi-join as an alternative to local join evaluation strategy. Sesame uses nested-loop implementation of joins as default, which results in sending a SPARQL query to a remote SPARQL endpoint for each obtained variable binding set and thus in low query evaluation performance. Distributed semi-joins are executed instead of local joins whenever a Remote Node is reached during Sesame's evaluation of SPARQL query tree. Our distributed semi-join algorithm buffers the obtained variable binding sets and sends them in a batch as conditions in a SPARQL FILTER expression, appended to the wrapped triple pattern, to remote SPARQL endpoint. Doing so we save as many sent and received SPARQL query and result messages as is the size of our buffer and thus improve the distributed query evaluation performance while retaining full compatibility with current SPARQL-compliant endpoints.

5. FIRST EXPERIENCE

The implemented distributed semi-joins according to preliminary testing results seem to enhance the query evaluation performance as expected. Detailed evaluation of this impact is ongoing.

6. FUTURE WORK

6.1 Distributed Join-aware Join Re-ordering

We would like to further improve the query evaluation performance by introducing a distributed join-aware join re-ordering. We will make use of the current Sesame optimization techniques for local queries and add our own component which will be re-ordering joins according to their relative costs. The costs will be based on statistics taking into account a sub-query selectivity combined with the distinction whether a triple pattern is supposed to be evaluated locally or at a remote SPARQL endpoint.

6.2 SPARQL Endpoints Statistics

In addition to join re-ordering we would like to make use of statistics about SPARQL endpoints in order to optimize queries even further. Hopefully the recent initiative called Vocabulary of Interlinked Datasets (http://community.linkeddata.org/MediaWiki/index.php?VoiD) will get to a point where it could be used for this purpose.

7. CONCLUSION

We briefly presented our Sesame extension *Distributed SPARQL* which aims at providing an integrated way of querying data sources scattered across multiple SPARQL endpoints. We shortly described its implementation and optimization used so far and outlined the direction for its future development. *Distributed SPARQL* is a part of Networked Graphs [4] project and is publicly available at https://launchpad.net/networkedgraphs.

8. ACKNOWLEDGMENTS

The authors are grateful to Steffen Staab for his support. The research was supported by the European Commission under contract FP6-027026, Knowledge Space of Semantic Inference for Automatic Annotation and Retrieval of Multimedia Content - K-Space.

9. REFERENCES

[1] Prud'hommeaux, E., Seaborne, A.: *SPARQL Query Language for RDF. W3C Recommendation (January 2008)* http://www.w3.org/TR/rdf-sparql-query/

[2] Quilitz, B., Leser, U.: *Querying Distributed RDF Data Sources with SPARQL*. In: The Semantic Web: Research and Applications. Springer Berlin / Heidelberg (2008) 524-538

[3] Langegger, A., Wöß, W., Blöchl, M.: *A Semantic Web Middleware for Virtual Data Integration on the Web.* Springer Berlin / Heidelberg (2008) 493-507

[4] Schenk, S., Staab, S.: *Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web*. WWW 2008: 585-594