Interactive Comparison of Triple Pattern Fragments Query Approaches

Joachim Van Herwegen, Ruben Verborgh, Erik Mannens, and Rik Van de Walle

Multimedia Lab – Ghent University – iMinds
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
joachim.van.herwegen@ugent.be

Abstract. In order to reduce the server-side cost of publishing queryable Linked Data, Triple Pattern Fragments (TPF) were introduced as a simple interface to RDF triples. They allow for SPARQL query execution at low server cost, by partially shifting the load from servers to clients. The previously proposed query execution algorithm provides a solution that is highly inefficient, often requiring an amount of HTTP calls that is magnitudes larger than the optimal solution. We have proposed a new query execution algorithm with the aim to solve this problem. Our solution significantly improves on the current work by maintaining a complete overview of the query instead of just looking at local optima. In this paper, we describe a demo that allows a user to easily compare the results of both implementations. We show both query results and number of executed HTTP calls, proving a clear picture of the difference between the two algorithms.

Keywords: Linked Data, SPARQL, query execution, query optimization, demo

1 Introduction

In the past few years, there has been a steady increase of available RDF data [3]. If a publisher decides to provide live queryable access to datasets, the de-facto default choice is to offer a public SPARQL endpoint [2]. The downside of the flexibility of SPARQL is that some queries require significant processing power. Asking a lot of these complex queries can put a heavy load on the server, causing a significant delay or even downtime. Recently, Triple Pattern Fragments (TPF, [5]) were introduced as a way to reduce this load on the server by restricting the TPF server interface to more simple queries. Clients can then obtain answers to complex SPARQL queries by requesting multiple simple queries and combining the results locally. Concretely, a TPF server only replies to requests for a single triple pattern. The response of the server is then a list of matching triples, which can be paged in case the response would be too large. Furthermore, each TPF contains metadata and hypermedia controls to aid clients with query execution.

The biggest challenge for the client is deciding which triple pattern queries result in the most efficient solution strategy. Since every subquery causes a new

HTTP request to the server, minimizing the number of queries reduces the network load and improves the total response time. The algorithm proposed by Verborgh et al. [5] is greedy: at each decision point, clients choose the local optimum by executing the request that has the fewest results. This works fine for certain classes of queries, but others can perform quite badly.

In our paper at ESWC2015 [4] we propose a new algorithm that tries to minimize the number of HTTP requests. For our demo we have created an interface to compare the performance of both algorithms. This allows users to easily see the difference in HTTP calls and execution time for both implementations.

2 Related work

The two most common ways to access linked data on the web currently are SPARQL endpoints and data dumps. Both of these have certain disadvantages.

SPARQL endpoints execute the complete query the client aims to solve. These queries can be quite complex and require a lot of computation power. This likely contributes to the low availability of public SPARQL endpoints [1].

An other solution is downloading a data dump from the server and then executing the query locally on the data dump. This entails a high cost for the client, and a major disadvantage is that data is not live: all changes that happen after the data was downloaded are not visible to the client. If the client wants an update, the entire dataset has to be downloaded again from the server.

Triple Pattern Fragments is a solution that proposes a middle ground between fully-fledged SPARQL endpoints and simple data dumps [5]. Server only answer single Basic Graph Pattern (BGP) queries which are combined client-side to answer complete SPARQL queries. This methodology has the live data advantage of SPARQL endpoints and the low server cost of data dumps.

3 Problem statement

Because of the greedy implementation of the original algorithm [5], the performance varies highly between different SPARQL queries: for some queries the greedy solution might be optimal, while for others it is exponentially worse. We will exemplify this with such a worst-case query and show how our, more optimal, solution handles it.

```
SELECT ?person ?city WHERE {
  ?person a dbpedia-owl:Architect. # p1 :          ±1,200 triples
  ?person dbpprop:birthplace ?city. # p2 :        ±430,000 triples
  ?city dc:subject dbpedia:Capitals_in_Europe. # p3 :    57 triples
}
```

Listing 1.1. SPARQL query to find European architects

We assume a page size of 100, i.e. a single server call returns 100 results for the corresponding triple pattern. Since the original implementation is greedy, it executes the query as follows:

1. Since p_3 has the least triples, download all corresponding `?city` bindings.
2. Bind all the `?city` values to p_2 and request all 57 patterns from the server, this requires a total of 430 calls since some cities have multiple pages of people and results in a total of 43,000 `?person` bindings.
3. Bind all the `?person` values to p_1 and request all 43,000 patterns from the server, requiring an additional 43,000 calls.

This results in a total of ± 43,431 HTTP calls. A more optimal solution would be the following:

1. Download all `?city` bindings from p_3. (1 call)
2. Download all `?person` bindings from p_2 after binding the values from the previous step. (430 calls)
3. Download all `?person` bindings from p_1. (12 calls)
4. Locally compare the results from the previous two steps to find the final `?person` bindings. (0 calls)

This solution only requires ± 443 calls, which is 100 times less than the greedy solution. Our algorithm [4] tries to solve queries in by checking which HTTP calls might provide the most efficient results. This is done by looking at the complete query all the time, not just the local bindings we got from the previous HTTP call.

4 Algorithm comparison demo

The demo which will be shown at the conference is an adaption of the Linked Data Fragments (LDF) web client found at `http://client.linkeddatafragments.org/`. It features an enhanced client which executes queries twice, for research purposes: once with the original algorithm and once with the optimized implementation. An example of this can be seen in Figure 1. Besides just showing the results, we also show some statistics about how the algorithms operate, such as the number of HTTP calls executed so far and how many results they have found with these HTTP calls. This allows the user to see the immediate effect of our implementation. The greedy algorithm might execute its HTTP calls faster, since it requires less local processing, while still having found less results. As mentioned in our paper [4], the optimized algorithm is a lot better in some cases while only being equal or even worse in other cases. With our demo it will be easy to see the different results for all types of queries.

The optimized algorithm we propose has a more complex execution than the original algorithm. This demo allows a user to more easily follow the execution without having read or fully understood the original paper. This might help people who want to extend the implementation to allow for more advanced queries or to improve the existing parts.

References

1. Buil-Aranda, C., Hogan, A., Umbrich, J., Vandenbussche, P.Y.: SPARQLWeb-querying infrastructure: Ready for action? In: Proceedings of the 12th International Semantic

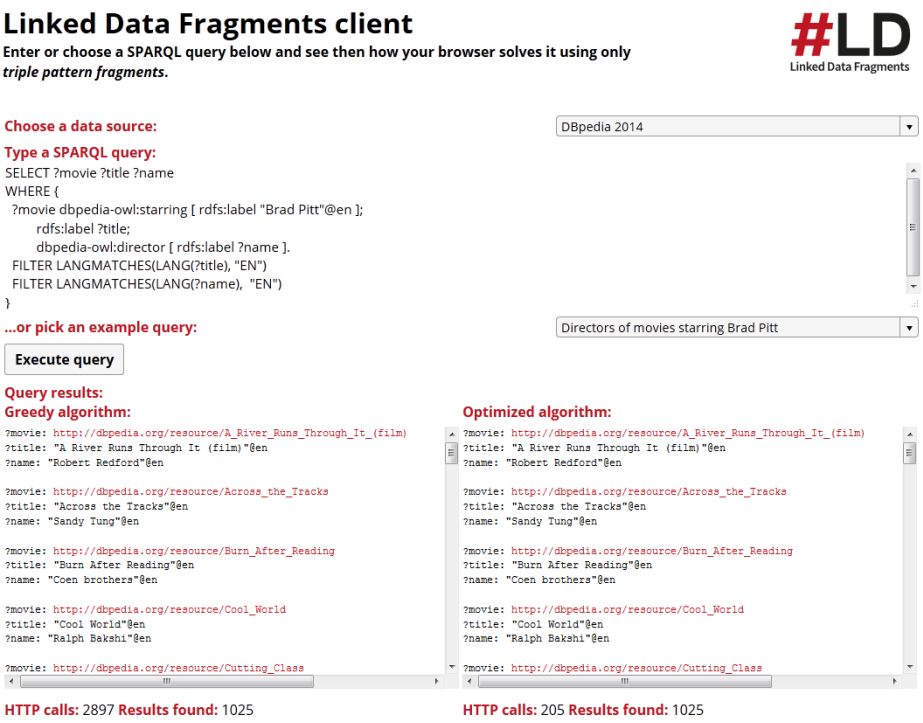

Fig. 1. The demonstrator shows live SPARQL query execution over the DBpedia fragments interface (`http://fragments.dbpedia.org/`). On the left-hand side are the results of the greedy algorithm; the right-hand side shows the optimized algorithm.

Web Conference (Nov 2013), `http://link.springer.com/chapter/10.1007/978-3-642-41338-4_18`

2. Harris, S., Seaborne, A.: SPARQL 1.1 query language. Recommendation, World Wide Web Consortium (Mar 2013), `http://www.w3.org/TR/sparql11-query/`
3. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the Linked Data best practices in different topical domains. In: ISWC 2014, vol. 8796, pp. 245–260 (2014), `http://dx.doi.org/10.1007/978-3-319-11964-9_16`
4. Van Herwegen, J., Verborgh, R., Mannens, E., Van de Walle, R.: Query execution optimization for clients of triple pattern fragments. In: Proceedings of the 12th Extended Semantic Web Conference (Jun 2015)
5. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through Linked Data Fragments. In: Proceedings of the 7th Workshop on Linked Data on the Web (Apr 2014), `http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_04.pdf`