PLANET: Query Plan Visualizer for Shipping Policies against Single SPARQL Endpoints

Maribel Acosta[1], Maria-Esther Vidal[2], Fabian Flöck[1],
Simon Castillo[2], and Andreas Harth[1]

[1] Institute AIFB, Karlsruhe Institute of Technology, Germany
{maribel.acosta,fabian.floeck,harth}@kit.edu
[2] Universidad Simón Bolívar, Venezuela
{mvidal, scastillo}@ldc.usb.ve

Abstract. Shipping policies allow for deciding whether a query should be executed at the server, the client or distributed among these two. Given the limitations of public SPARQL endpoints, selecting appropriate shipping plans is crucial for successful query executions without harming the endpoint performance. We present PLANET, a query plan visualizer for shipping strategies against a single SPARQL endpoint. We demonstrate the performance of the shipping policies followed by existing SPARQL query engines. Attendees will observe the effects of executing different shipping plans against a given endpoint.

1 Introduction and Overview

In the context of the Web of Data, endpoints are acknowledged as promising SPARQL server infrastructures to access a wide variety of Linked Data sets. Nevertheless, recent studies reveal high variance in the behavior of public SPARQL endpoints, depending on the queries posed against them [3]. One of the factors that impact on the endpoint performance is the type of shipping policy followed to execute a query.

Shipping policies [4] define the way that the workload of executing a query is distributed among servers and clients. Query-shipping policies conduct the execution of query operators at the server, while plans following data-shipping exploit the capacity of the client and execute the query operators locally. In contrast, hybrid approaches pose sub-queries and operators according to the complexity of the queries, and the server workload and availability. Current SPARQL query engines implement different policies. For example, FedX [5] implements a query-shipping strategy, executing the whole query at the endpoint, when the federation is comprised by one endpoint. ANAPSID [2] usually follows a hybrid-shipping strategy, it locally gathers the results of star-shaped sub-queries executed by the endpoint. To showcase an adaptive hybrid approach alongside FedX and ANAPSID, we also demonstrate SHEPHERD [1], an endpoint-tailored SPARQL client-server query processor that aims for reducing the endpoint workload and benefits the generation of hybrid shipping plans.

Analyzing the shipping policies followed to execute a query provides the basis not only to understand the behavior of an endpoint, but also allows for the development of endpoint-aware query processing techniques that preserve endpoint resources. The goal of the work presented here is to assist data consumers or data providers in understanding the effects of posing different shipping plans against an existing public SPARQL

endpoint. We introduce PLANET, a query plan visualizer for shipping strategies that provides an intuitive overview of the plan structure, the used shipping strategies as well as key metrics to understand the behavior of different engines when executing a query. PLANET is designed to shed light on the distribution of operator execution between client and server, which is crucial for investigating the type of plans that may lead to severe under-performance of endpoints. Attendees will observe the impact of different shipping strategies when queries are posed against a single endpoint. The demo is published at `http://km.aifb.kit.edu/sites/planet/`.

2 The PLANET Architecture

PLANET invokes SPARQL query engines that implement different shipping strategies. In this work, we studied the query-shipping plans produced by FedX, and the hybrid shipping plans of ANAPSID and SHEPHERD. The plan retrieved from each engine is processed by PLANET's query plan parser, which translates the plans into JSON structures to encode the visualization data that will be consumed by the rendering module. Currently PLANET is able to parse plans generated by engines that use the Sesame[1] framework, or the ANAPSID or SHEPHERD internal structures.

The rendering module uses the "Collapsible Tree" layout of the D3.js JavaScript library[2] to generate the visualizations of plans produced by the SPARQL query processing engines. Figures 1(a) and 1(b) show snapshots of plans rendered by PLANET. Plan operator nodes are filled with different colors to distinguish whether the operator is executed by the engine (locally) or at the server (remotely), which allows to easily identify the type of shipping strategy followed in each plan.

The plan descriptor reports a set of metrics characterizing the shipping plans. Execution performance is measured by the execution time of the query and the number of results produced. In addition, the metric *hybrid ratio* measures the quantitative relation between the SPARQL operators executed at the local engine and the ones executed at the endpoint. The hybrid ratio of a plan p is calculated as follows:

$$hybridRatio(p) = \frac{clientOp(p) \cdot serverOp(p)}{totalOp(p) \cdot max\{clientOp(p), serverOp(p)\}}$$

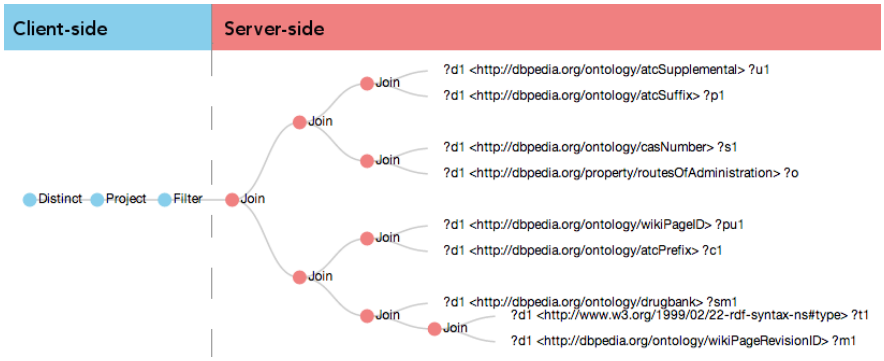
where $clientOp(p)$, $serverOp(p)$ stand for the number of operations executed for plan p at the client and server, respectively, and $totalOp(p) = clientOp(p) + serverOp(p)$. Note that the hybrid ratio for plans following data- or query-shipping strategies is zero.

The output of PLANET is a set of plan visualizations and a summary report with the metrics computed for each plan.

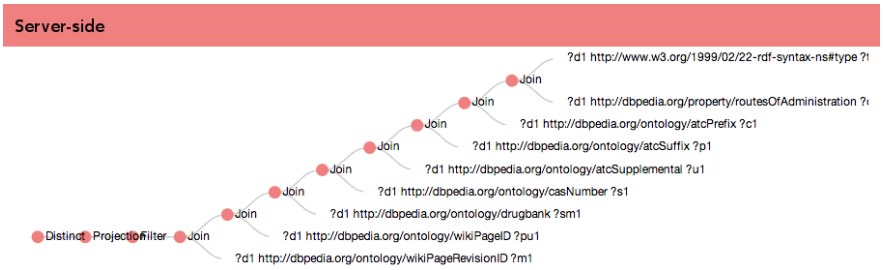
3 Demonstration of Use Cases

As an illustrating example, consider the following query included in our online demo: GP Query 2 against the DBpedia endpoint, comprised of 9 triple patterns and one FILTER operator. Figure 1 reports on the plans depicted by PLANET for the previous query. The plan reported in Figure 1(a) follows a hybrid shipping strategy where the quantitive relation between the SPARQL operators executed locally and the ones

[1] `http://www.openrdf.org/`
[2] `http://d3js.org/`



(a) Hybrid Shipping Plan. Sub-query under red bar is posed against DBpedia endpoint; Operators under blue bar are executed at the client side



(b) Query Shipping Plan. The whole query is posed to the DBpedia endpoint

Fig. 1. Plans against the DBpedia endpoint. Blue circles represent operators executed locally; red circles correspond to operators that will be posed against the endpoint

posed against the DBpedia endpoint or *hybrid ratio* is 0.27; the execution time is 0.63 secs. and one tuple is retrieved. On the other hand, when the query shipping-based plan presented in Figure 1(b) is executed, the execution time is 1.44 secs. and no answer is received. Finally, if the query is executed directly against the endpoint, the answer is effectively retrieved but the execution time is 6.91 secs. For the three different engines, we can observe that the combined performance of engine and endpoint deteriorates as the number of operators posed against the endpoint increases.

In congruence with the previous example result, the following research questions arose: (*i*) is the observed behavior due to limitations of the endpoints? Or (*ii*) is this behavior caused by the shipping plan followed during query execution? As part of this demo, we will visualize characteristics of different plans and public endpoints that provide evidence enabling to answer our research questions. By this time, we have performed a comprehensive study of the execution of 70 queries against seven different public SPARQL endpoints. We selected the query targets from the list of endpoints monitored by the SPARQLES tool [3] and classified them in quartiles. These included two high-performant endpoints (Top25% Quartile), two medium-performant ((25%; 50%] Quartile), and three second-least performant endpoints ((50%;75%] Quartile). We crafted ten SPARQL queries for each endpoint; five are composed of basic graph pat-

terns (BGP queries), and the others comprise any SPARQL graph pattern (GP queries). Attendees of the demo have the possibility of analyze the results of executing these queries currently loaded in the system, or visualize the plans of their own SPARQL queries. Query plans are computed on-the-fly while the reported results were computed off-line to facilitate demonstration. We will demonstrate the following use cases:
Effects of Shipping Policies in BGP Queries. We show that in setups as the one reported in Figure 1 and where endpoints receive large number of concurrent request per day, i.e., the endpoint is in the (50%;75%] quartile, hybrid-shipping policies can reduce execution time and the results surpasses the 79% of the answers retrieved by query-shipping plans. For high- and medium-performant endpoints, there is a trade-off between execution time and size of retrieved answers. Nevertheless, in all the queries the effectiveness of the endpoints is increased by up to one order of magnitude, i.e., the number of answers produced per second following a hybrid-shipping plan can be up 20 times the number of answers produced by a query-shipping plan.
Effects of Shipping Policies in GP Queries. We observed that for highly work-loaded endpoints, 90% of hybrid-shipping plans reduce execution time by up two orders of magnitude. Hybrid plans generated by SHEPHERD achieved the highest performance on DBpedia. For endpoints in other quartiles, a competitive performance between hybrid- and query-shipping plans is observed, but hybrid plan performance never significantly deteriorates. This suggests that hybrid-shipping policies are appropriate to achieve reasonable performance while shifting load from the server to the client.

4 Conclusions

PLANET visualizes the impact of shipping policies and provide the basis for the understanding of the conditions that benefit the implementation of hybrid shipping plans, e.g., attendees will be able to observe that for non-selective queries hybrid plans significantly outperforms the others. Thus, PLANET facilitates the analysis of the behavior of public endpoints, as well as, the development of scalable real-world client-server applications against single SPARQL endpoints.

Acknowledgements

The authors acknowledge the support of the European Community's Seventh Framework Programme FP7-ICT-2011-7 (XLike, Grant 288342).

References

1. M. Acosta, M.-E. Vidal, F. Flock, S. Castillo, C. Buil-Aranda, and A. Harth. Shepherd: A shipping-based query processor to enhance sparql endpoint performance. In *ISWC Poster Track*, 2014.
2. M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: an adaptive query processing engine for SPARQL endpoints. In *ISWC*, pages 18–34, 2011.
3. C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *ISWC*, pages 277–293, 2013.
4. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *SIGMOD Conference*, pages 149–160, 1996.
5. A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *ISWC*, pages 601–616, 2011.