SHEPHERD: A Shipping-Based Query Processor to Enhance SPARQL Endpoint Performance

Maribel Acosta[1], Maria-Esther Vidal[2], Fabian Flöck[1],
Simón Castillo[2], Carlos Buil-Aranda[3], 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
[3] Department of Computer Science, Pontificia Universidad Católica, Chile
cbuil@ing.puc.cl

Abstract. Recent studies reveal that publicly available SPARQL endpoints exhibit significant limitations in supporting real-world applications. In order for this querying infrastructure to reach its full potential, more flexible client-server architectures capable of deciding appropriate shipping plans are needed. Shipping plans indicate how the execution of query operators is distributed between the client and the server. We propose SHEPHERD, a SPARQL client-server query processor tailored to reduce SPARQL endpoint workload and generate shipping plans where costly operators are placed at the client site. We evaluated SHEPHERD on a variety of public SPARQL endpoints and SPARQL queries. Experimental results suggest that SHEPHERD can enhance endpoint performance while shifting workload from the endpoint to the client.

1 Introduction

Nowadays, public SPARQL endpoints are widely deployed as one of the main mechanisms to consume Linked Data sets. Although endpoints are acknowledged as a promising technology for RDF data access, a recent analysis by Buil-Aranda et al. [1] indicates that performance and availability vary notably between different public endpoints. One of the main reasons for the at times undesirable performance of public SPARQL endpoints is the unpredictable workload, since a large number of clients may be concurrently accessing the endpoint and some of the queries handled by endpoints may incur prohibitively high computational costs. To relieve endpoints of some of the workload they face, many operators of the query can potentially be executed at the client side. Shipping policies [2] allow for deciding which parts of the query will be executed at the client or the server according to the abilities of SPARQL endpoints.

The goal of this work is to provide a system to access SPARQL endpoints that shifts workload from the server to the client taking into account the capabilities of the addressed endpoint for executing a certain query – while still offering a competitive performance in terms of execution time and the number of answers produced. We propose SHEPHERD, a SPARQL query processor that mitigates the workload posed to public SPARQL endpoints by tailoring hybrid shipping plans to every specific endpoint. In particular, SHEPHERD performs the following tasks: (*i*) decomposing SPARQL

queries into lightweight sub-queries that will be posed against the endpoint, (*ii*) traversing the plan space in terms of formal properties of SPARQL queries, and (*iii*) generating shipping-based query plans based on the public SPARQL endpoint performance statistics collected by SPARQLES [1]. We designed a set of 20 different SPARQL queries over four public SPARQL endpoints. We empirically analyzed the performance of the hybrid shipping policies devised by SHEPHERD and the query shipping policy when submitting a query directly to a SPARQL endpoint.

2 The SHEPHERD Architecture

SHEPHERD is a SPARQL query processor based on the wrapper architecture [4]. SHEPHERD implements different shipping policies to reduce the workload posed over public SPARQL endpoints. Figure 1 depicts the SHEPHERD architecture which consists of three core components: the SHEPHERD optimizer, the engine broker, and the SPARQL query engine that is considered a *black box* component.

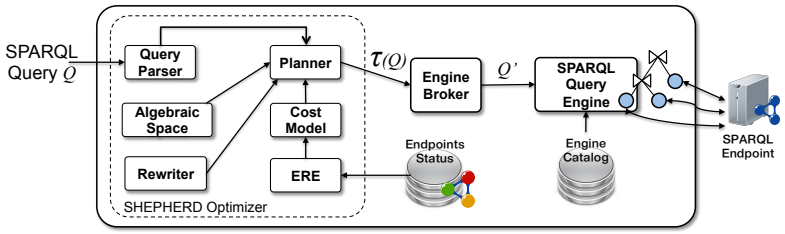

Fig. 1. The SHEPHERD architecture

The SHEPHERD optimizer is designed for enhancing SPARQL query plans since it relies on formal properties of SPARQL and statistics of SPARQL endpoints to estimate the plan cost. In the following, we elaborate on each of the sub-components that comprise the proposed optimizer.

- **Query parser**: Translates the input query Q into internal structures that will be processed by the planner.
- **Planner**: Implements a Dynammic Programming algorithm to traverse the space of plans. During the optimization process, SHEPHERD decides whether to place the operators at the server (i.e., endpoint), or client (i.e., SHEPHERD) according to statistics of the endpoint. In this way, SHEPHERD explores shipping polices tailored for each public endpoint. The planner generates bushy-tree plans, where the leaves correspond to light-weight sub-queries and the nodes correspond to operators (annotated with the shipping policy to follow).
- **Algebraic space and rewriter**: The algebraic space defines a set of algebraic rules to restrict plan transformations. The algebraic rules correspond to the formal properties for well-designed SPARQL patterns [3]. The rewriter transforms a query in terms of the algebraic space to produce a more efficient equivalent plan.
- **Cost model**: The cost of executing sub-queries and SPARQL operators at the endpoint is obtained from the ERE component. Based on these values, SHEPHERD

estimates the cost of combining sub-plans with a given operator. The cost model is designed to gradually increase the cost of operators when more complex expressions are evaluated. This behavior is modeled with the Boltzmann distribution.[1]

– **Endpoint Reliability Estimator (ERE)**: Endpoint statistics collected by the SPARQLES tool [1] are used to provide reliable estimators for the SHEPHERD cost model. The endpoint statistics are aggregated and stored in the SHEPHERD catalog, and used to characterize endpoint in terms of opertator performance.

The engine broker translates the optimized plan $\tau(Q)$ into the corresponding input for the SPARQL query engine that will execute the plan. The engine broker can specify the plan in two different ways: i) as a query Q' with the SPARQL Federation Extension, to execute the query with a SPARQL 1.1 engine; ii) translating the plans directly into the internal structures of a given query engine.

3 Experimental Study

We empirically compared the performance of the hybrid shipping policies implemented by SHEPHERD with the query shipping policy when executing queries directly against the endpoint. We selected the following four public SPARQL endpoints monitored by SPARQLES [1]: DBpedia, IproBio2RDF, data.oceandrilling.org and TIP.[2] We designed a query benchmark comprising five different queries for each endpoint.[3] Each query contains modifiers as well as graph patterns that include UNIONs, OPTIONALs, and filters. SHEPHERD was implemented using Python 2.7.6. Queries were executed directly against the endpoints using the command `curl`. All experiments were performed from the Amazon EC2 Elastic Compute Cloud infrastructure.[4]

Figure 2 depicts the result of the queries in terms of the execution time (sec.) as reported by the Python `time.time()` function. We can observe that in the majority of the cases SHEPHERD retrieves the results notably faster, except in three queries. Concerning the cardinality of the result set retrieved, a similar picture emerges. For 18 of the overall 20 queries tested, both methods produced the same amount of results, while in one instance each SHEPHERD and the query shipping approach did not retrieve any answers. Both methods therefore seem to be on par in this regard.

Even though SHEPHERD is able to reduce the execution time to a certain extent, the most important finding is that it shifted in average 26% of the operators in the queries to the client, thereby relieving the servers of notable workload. The ratio of data shipping varies significantly from case to case depending on the individual shipping strategy chosen, but does not show a direct correlation with the achieved runtime decrease.[5]

Hence, we can affirm that SHEPHERD is able to reduce computational load on the endpoints in an efficient way; this could not be achieved neither by simply moving all

[1] The Boltzmann distribution is also used in Simulated Annealing to model the gradual decreasing of a certain function (temperature).

[2] Available at `http://dbpedia.org/sparql`, `http://iproclass.bio2rdf.org/sparql`, `http://data.oceandrilling.org/sparql` and `http://lod.apc.gov.tw/sparql`, respectively.

[3] `http://people.aifb.kit.edu/mac/shepherd/`

[4] `https://aws.amazon.com/ec2/instance-types/`

[5] Std.Dev. is 0.09. The poster will discuss further details about this ratio and its impact.

operator execution to the client – since this increments the bandwith consumption and the evaluation of non-selective queries may starve the resources of the client – nor by submitting the whole query to the endpoint as shown in our experiments.

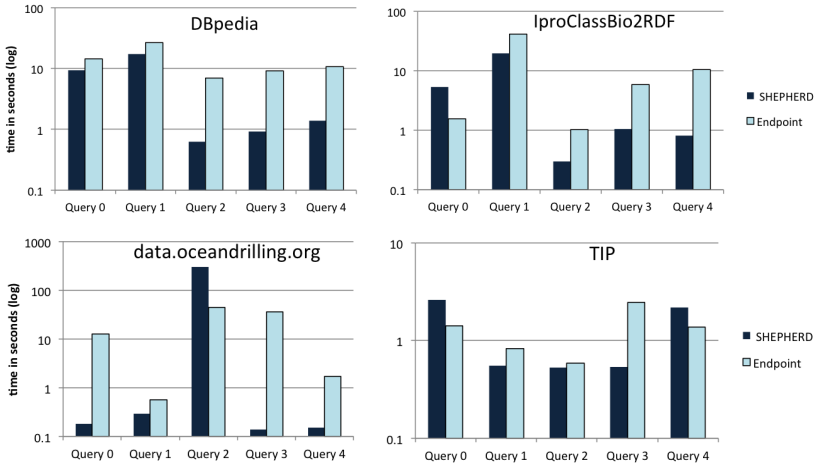

Fig. 2. Runtime results for the four different public endpoints and studied queries

4 Conclusions

We presented SHEPHERD, a SPARQL query processor that implements hybrid shipping policies to reduce public SPARQL endpoint workload. We crafted 20 different queries against four SPARQL endpoints and empirically demonstrated that SHEPHERD is *(i)* able to adapt to endpoints with different characteristics by varying the rate of operators executed at the client, and *(ii)* in doing so not only retrieves the same number of results as query shipping but even decreases runtime in the majority of the cases. While these results provide a first insight into SHEPHERD's capabilities, they showcase the potential of adaptive hybrid shipping approaches, which we will explore in future work.

Acknowledgements

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

References

1. C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *International Semantic Web Conf. (2)*, pp. 277–293, 2013.
2. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *SIGMOD Conference*, pp. 149–160, 1996.
3. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.
4. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.