# Dataset Descriptions for Optimizing Federated Querying

Angelos Charalambidis
Institute of Informatics and Telecommunications, NCSR 'Demokritos', Athens, Greece
acharal@iit.demokritos.gr

Stasinos Konstantopoulos
Institute of Informatics and Telecommunications, NCSR 'Demokritos', Athens, Greece
konstant@iit.demokritos.gr

Vangelis Karkaletsis
Institute of Informatics and Telecommunications, NCSR 'Demokritos', Athens, Greece
vangelis@iit.demokritos.gr

## ABSTRACT

Dataset description vocabularies focus on provenance, versioning, licensing, and similar metadata. VoID is a notable exception, providing some expressivity for describing subsets and their contents and can, to some extent, be used for discovering relevant resources and for optimizing querying. In this poster we describe an extension of VoID that provides the expressivity needed in order to support the query planning methods typically used in federated querying.

## Categories and Subject Descriptors

H.2.4 [**Systems**]: Distributed databases

## General Terms

Algorithms

## Keywords

RDF store histograms; RDF vocabulary

## 1. INTRODUCTION

Repository federation and distributed querying are key technologies for the efficient and scalable deployment of semantic technologies. Although supported by most major triple stores and formalized in the new edition of the SPARQL specification, we have not yet reached the point where federation is transparent to the user: it is the client application's responsibility to declare which remote data source each query fragment should be dispatched to.

One major issue that needs to be addressed in order to achieve efficient and transparent federated querying, is the construction of the query plan that guides query execution: Given a query, there are many plans that a database system can consider to process it and retrieve the answer. All those plans are *equivalent* in the sense that will return the same answer, but they vary in their *cost*, that is the resources and the amount of time that they need to run. *Query optimization*, i.e., selecting the most cost-efficient plan, is absolutely essential for database systems as the cost difference between two equivalent plans can be enormous. Query optimization has been studied in a variety of contexts and from many different angles [1], but the literature converges to solutions involving: (a) a search space of query plans; (b) a cost function that assigns a cost to each plan in the search space, intuitively corresponding to an estimation of the resources need to execute the plan; and (c) an enumeration algorithm that searches among the possible query plans and selects the one with the least cost.

Since retrieving data from the disk is a crucial factor in cost, cost functions typically refer to statistics regarding the *cardinality* of query patterns, i.e., the number of tuples that match a given pattern, and the *selectivity* of joins, i.e., the ratio of the cardinality of a pattern that can be joined on certain column against another pattern. Such statistics are stored in *histograms*, data structures that are typically an internal part of the database index. In conventional databases, this internal structure never needs to receive an explicit representation or be serialized and communicated. This includes distributed databases, where the central node controls the way data is distributed and also maintains such histograms; or where distributed histograms are communicated in implementation-specific serializations.

Recent developments in and *federated querying*, however, create new use cases where distributed systems are less tightly integrated. In an increasingly popular scenario, also supported by the new SPARQL specification, a federated querying engine concentrates and serves to applications multiple remote data sources. Such scenarios are becoming increasingly common in the context of services that federate *Semantic Web* and *Linked Open Data*.

In such a system, and regardless of any optimization performed internally to each data source, the federation server maintains the statistics needed to identify data sources that contain relevant data and to optimize the distribution of the various sub-queries among them. The approaches proposed to acquire and maintain such statistics include executing queries, or otherwise imposing a database workload, in order to update histograms or updating histograms by observing the client-requested query workload.

An important development that came with Semantic Web systems' adopting histogram-based query optimizers, was that what originally was an internal structure of the database system, started becoming explicit: federated querying systems such as SPLENDID [2] and ANAPSID [3] use explicit data source descriptions, represented in RDF using the VoID vocabulary as well as probing the data source with ASK

*WWW 2015 Companion,* May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742779.

queries. Whereas for SPLENDID probing is a fall-back mechanism in the absence of VoID descriptions, for ANAPSID it is a way of complementing VoID descriptions with further information needed for its operation.

The work presented here is the SEVOD vocabulary, the extension of the VoID vocabulary needed in order to fully represent the various histogram models found in the literature. Adopting SEVOD will provide Semantic Web data stores with an explicit interface between query optimization and histogram statistics maintenance, such that (a) current systems are fully supported and can represent the full histogram structure they are maintaining or consuming; and (b) the new use case is supported, where data stores publicly expose histogram statistics, which can be consumed and used by the federated querying systems that incorporate them into their federations.

## 2. THE SEVOD VOCABULARY

In the poster we will present Sevod, which extends VoID to provide the expressivity needed in order to support the query planning methods typically used in federated querying. The vocabulary can be downloaded from its namespace URL, `http://www.w3.org/2015/03/sevod`

More specifically, Sevod introduces the `svd:Partition` class, a set of `void:Dataset` instances that are a *partition* of another `void:Dataset` instance. This is done by using the property `svd:part` to link the `svd:Partition` instance with the instances that make up the partition and the property `svd:partitions` to link it with the instance that is partitioned by them. All fillers of this property must also be fillers of the `void:subset` property of the `void:Dataset` instance that fills the `svd:Partition` instance's `svd:partitions` property. This allows expressing information about triples that *cannot* be in a dataset.

Furthermore, Sevod introduces multi-dimensional buckets that cover *joins* of triple patterns. A `svd:Join` instance connects two `void:Dataset` instances with an integer value that is (or estimates or approximates) the *selectivity* of the join of these datasets. The triple element that is joined is denoted by the specific subpropery of the `svd:joins` property used to link the `svd:Join` instance with the `void:Dataset` instances.

`svd:Join` instances link to the `void:Dataset` instances they join using one of the following three properties, depending on the triple element they join on: `svd:joinSubject`, `svd:joinPredicate`, and `svd:joinObject`

The `svd:selectivity` property links a `svd:Join` instance with an the instance of `svd:SelectivityValue` that is (or estimates or approximates) the selectivity of the join denoted by the `svd:Join` instance. We define the range of the `svd:selectivity` property to be a class instead of simple numerical fillers. In this manner, we encapsulate statistics under a class that can be extended to cover application-specific requirements. In order to ensure compatibility, we further require that `svd:SelectivityValue` instances must have an `rdf:value` property and that this property has as value an `xds:integer`. Other, application specific, properties may be defined as needed for extensions of this class.

## 3. CONCLUSIONS

Sevod is used in the Semagrow Stack,[1] an infrastructure for deploying a SPARQL *federation endpoint* that automatically breaks up queries into *fragments*, each comprising one or more *query patterns*, and dispatches each fragment only to those data sources that hold triples satisfying patterns in the fragment. In this manner we allow client applications to execute queries without having to declare where each query fragment should be dispatched. Besides alleviating the effort for the endpoint client, this also allows to (transparently to the client) dynamically adapt remote querying policies in order to balance load or to circumvent temporarily unavailable services. Furthermore, we do not impose any central management requirements and can operate in the context of a loose federation of repositories; several of which might be public endpoints that are not even aware of partaking in the federation.

In the future, we are planning to develop a histogram publishing mechanism that will allow the federated endpoints to provide Sevod metadata to the federation server by serializing their internal query optimization structures. This will improve the accuracy of the statistics while retaining the dynamic federation nature of the SemaGrow Stack. In our envisaged scenario, it will be sufficient to provide an endpoint URL and the federation optimizer will be able to automatically retrieve accurate Sevod descriptions from the endpoints. Naturally, the federation server will still need to monitor query feedback in order to dynamically adjust the granularity of the descriptions that it keeps, in order to strike a balance between the accuracy of the statistics and the storage space they require. However, it will not be necessary either to impose querying overheads for discovering statistics or to lag behind updates before the histograms have had the opportunity to converge to the new data distributions.

## Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 318497. For more details please see `http://www.semagrow.eu`

## 4. REFERENCES

[1] Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '98). (1998) 34–43

[2] Görlitz, O., Staab, S.: SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Proc. 2nd Intl Workshop on Consuming Linked Data (COLD 2011), Bonn, Germany, Oct 2011. Volume 782 of CEUR Workshop Proceedings. (2011)

[3] Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: Proc Intl Semantic Web Conference. LNCS 7031, Springer (2011)

[1]Please see `http://www.semagrow.eu` for more details