The Sevod Vocabulary for Dataset Descriptions for Federated Querying

Stasinos Konstantopoulos, Angelos Charalambidis, Antonis Troumpoukis, Giannis Mouchakis, and Vangelis Karkaletsis

Institute of Informatics and Telecommunications,
NCSR ‘Demokritos’, Athens, Greece
{konstant,acharal,antru,gmouchakis,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 paper we describe the Sevod vocabulary, an extension of VoID that provides the expressivity needed in order to support the query planning methods typically used in federated querying. We also present a tool for automatically scraping such metadata from RDF dumps and give statistics about the size of the descriptions for the FedBench datasets.

Keywords: RDF store histograms, RDF vocabulary, optimizing federated SPARQL query processing

1 Introduction

Machine-readable descriptions of Web data, such as general metadata, quality features, statistical information about the entities and how they link, and licensing/provenance metadata, are becoming an increasingly important element of the architecture of the Web. Without such descriptions, client applications cannot be informed of the nature, scope and characteristics of data from particular sources, limiting applications to consuming well-known reference datasets.

Existing vocabularies operate both at the level of cataloguing datasets, as well as at the more detailed level of statistics about the resources contained and described in each dataset [6, Section 5.6]. The *Data Catalogue Vocabulary (DCAT)* [17] is the W3C Recommendation for describing datasets in data catalogs. DCAT is used to publish attribution, licensing, and thematic information about complete datasets. The *Asset Description Metadata Schema (ADMS)* [3] is the DCAT extension that targets semantic interoperability by linking datasets to *semantic assets* (schemas, taxonomies, codelists, etc.) DCAT and ADMS afford discoverability and the expression of provenance and licensing, while linking to the semantic assets used by each dataset works towards enabling client applications to assess if data from multiple datasets is interoperable.

The *Vocabulary of Interlinked Datasets (VoID)* [2] is the W3C recommendation for publishing details about the internal structure of datasets. Besides the schemas used in a dataset, VoID also allows declaring the namespace of the resources described in it. This further enhances discoverability, as a client application can reason about which datasets might contain data that describes a given resource. Finally, VoID also supports query optimization use cases, by foreseeing terms for providing multifaceted instance-level statistics, such as the number of triples with a specific predicate, the number of triples with subjects that are members of a specific class, the number of distinct subjects, predicates, and objects, and similar.

Although dataset statistics in VoID go some of the way towards supporting client applications to efficiently plan and execute queries, they do not cover all information needs of modern *federated querying* systems. Federated querying is a key technology for sustaining the decentralized nature of the Web and has been formally added to the architecture of the Semantic Web in the new edition of the SPARQL specification. In this paper, we present *Sevod*, an extension of VoID that specifically addresses the aspects of dataset description that are relevant to efficient and transparent federated SPARQL query processing. More specifically, we first review federated SPARQL query processing, focusing on the data descriptions needed by the different federation engines (Section 2). Based on this review, we extract requirements for a dataset description schema that can support the state of the art of federated SPARQL query processing and discuss how well these requirements are met by VoID (Section 3). We then proceed to present Sevod and to explain how the proposed extensions cover the requirements that are not met by VoID (Section 4), present a method for generating Sevod description (Section 5), and conclude (Section 6).

2 Federated SPARQL Query Processing

Given a non-trivial query, there are multiple alternative query execution plans that will all produce the same response, or equally valid responses. Although any of these can be used, they might vary radically in their *cost*, that is, in the resources and the amount of time that they need to execute. *Query optimization*, i.e., selecting the most cost-efficient plan, has been studied in a variety of contexts and from many different angles [10]. These different approaches converge on relying on a *cost function* that assigns a numerical *cost* to each candidate plan. This cost is meant to reflect the resources needed to execute the plan.

To calculate cost, these functions refer to statistics that estimate the volume of data that needs to be lifted from the persistent store and to be transferred. Such statistics are 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 joins against another query pattern. In conventional databases, these statistics are stored in *histograms* [14], internal data structures that are part of the database index. Prominent relevant work includes *statistics on intermediate tables (SITs)*, one-dimensional cardinality counts for an attribute [7]. SITs are organized in *buckets*; each bucket stores a range of values for a given attribute and the number of tuples in this range. *Multidimensional histograms* [1] avoid the propagation of errors through a sequence of operators by directly storing statistics about more complex sub-expressions that match longer intermediate sub-expressions of the query, rather than individual attributes. Naturally, not all possible combinations can be stored. To limit space requirements, buckets are merged when their statistics are comparable and merging does not degrade the estimations, and *sub-buckets* are created whenever there is a narrow value range with diverging statistics.

As histograms are internally used by the database's own query processor, they never received an explicit representation or serialization. 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. The open nature of the Web, however, creates new use cases where federated query processors are less tightly integrated. In addition to the optimizations performed internally to each data source, federated systems also need to identify data sources that contain relevant data and to optimize the execution of the various sub-queries among them [20, 21]. Some federated systems do not rely on any prior knowledge about the data sources they federate, and base their planning on universal, hard-wired assumptions [25, FedX] or information they discover during query execution [5, Avalance]. But most systems base query planning on histograms, adapting methods originally proposed in the databases literature and assigning explicit semantics to these originally internal data structures. Federated querying systems such as DARQ [19], SPLENDID [12], LHD [27], and Semagrow [9] consume detailed, instance-level VoID descriptions for data source discovery and for query plan cost estimation.

Several systems, however, have out-grown VoID and use methods that require more sophisticated data source descriptions. Recent versions of Semagrow use an inclusion hierarchy of multi-dimensional *histogram buckets*, each providing statistics about triple patterns or sets of joined triple patterns [28]. Although VoID subsets can represent the inclusion hierarchy of buckets, VoID focuses on star-shaped descriptions of resources and more complex joins of triple patterns cannot be represented. The Odyssey system [18] uses *characteristic sets* to represent statistics about how resources are linked, which also fall outside VoID's ability to express information about the objects of triples. Finally, the QUETSAL system integrates a line of research on sophisticated data source selection [23, 22, 8]. QUETSAL advances beyond source selection based on isolated triple pattern matching to also consider the joins between triple patterns.

All of these recent developments necessitate the extension of VoID with the expressivity needed to represent not only statistics about how resources are described in star-shaped joins, but also how they link and combine in arbitrary joins. This will allow making explicit information structures that are currently internally computed and stored by these federation systems, so that they can be shared and, hopefully, exposed directly by future triple store implementations.

3 Vocabulary Requirements

As discussed in the previous section, database histograms comprise *buckets*. A bucket holds a value range for an attribute and the cardinality of this attribute range, that is, the number of tuples where the given attribute has a value within the given range. Following what is common practice in representing relational data in RDF, an attribute value in a relational table's tuple would be a triple where the subject is the key of the tuple, the predicate is the attribute name, and the object is the attribute value.

We will generalize buckets here, to allow not only the object (attribute value) but any of the elements of a triple to be specified by a range and not fixed. We will also call buckets *subsets*, to match more familiar terminology:

Requirement 1 *The elementary unit of information in a Web dataset description is a* subset *that holds:*

- *the subject URI or a specification of a range of subjects*
- *the predicate URI or a specification of a range of predicates*
- *the object URI or value, or a specification of a range of objects*
- *the* cardinality*: the number of triples that match the above.*

This requirement makes subsets useful for retrieving the cardinality of isolated SPARQL triple patterns, by searching in the histogram for a subset where the binded values in the triple pattern fall within the range given in the subset.

Multidimensional buckets hold the number of tuples that have values within the range given in the bucket for *all* the attributes covered by the bucket. Again, we transfer this to RDF in a way that reflects SPARQL queries, requiring that histograms can also capture the cardinality of triple pattern joins:

Requirement 2 *A subset can be specified by a set of triple specifications like those in Requirement 1, with added constraints on triple elements that should be identical.*

Buckets are organized in an inclusion hierarchy, where the attribute ranges in a child bucket are subsets of the attribute ranges in its parent. Since the children of a bucket might or might not be a partitioning of the parent, depending on the histogram algorithm used, it must also be possible to specify if the children of a given bucket are a partitioning or not.

Requirement 3 *Subsets are organized in an inclusion hierarchy, where the triples matching a child subset are included in the triples matching the parent subset. The children of a subset might or might not be a partitioning, but if they are it must be possible to express this fact.*

A further consideration is that, depending on the method used to construct the histogram, selectivity and cardinality values might be approximated or known be within a given range, rather than exactly known. Srivastava et al. [26] and Kaushik and Suciu [15], for instance, presented methods that model cardinalities and distinct value counts based on the *entropy maximization* principle.

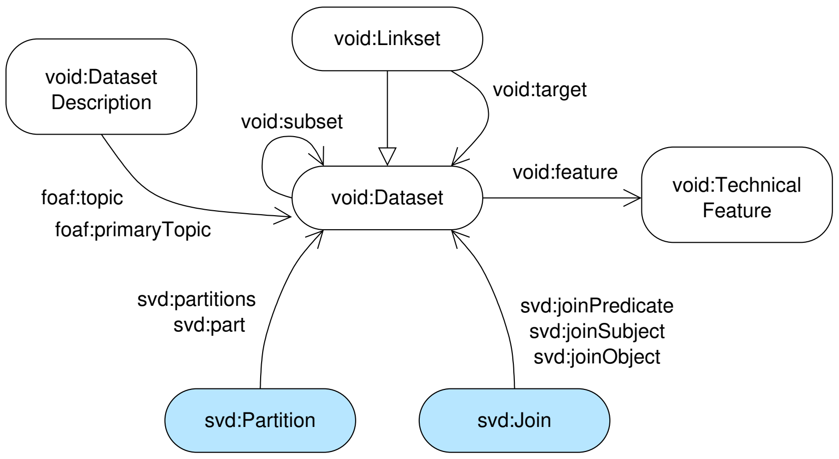

Fig. 1. Sevod top classes and relationship to VoID.

Requirement 4 *Subset statistics do not need to be scalar values, but may also be more complex representations of constraints over or distributions of scalar values that are not precisely known.*

4 The Sevod Vocabulary

Sevod extends VoID to provide the expressivity needed in order to support the query planning methods typically used in federated querying. The top Sevod classes and their relationship to their VoID super-classes is shown in Figure 1. The `svd:Join` class connects the `void:Dataset` instances that bind joins of triple patterns with the selectivity of these joins. The `svd:Partition` class connects the `void:Dataset` instances that form a partitioning of another `void:Dataset` instance and allows to express the local closure of the data that might be present in this superset.

The vocabulary can be downloaded from its namespace URL, `http://www.w3.org/2015/03/sevod`

4.1 Joins of Triple Patterns

For the purposes of resource discovery, VoID is adequate for finding sources for grounding star patterns in the query, where the properties of a known 'central' resource need to be retrieved. It is less well-suited for finding sources for path and sink patterns, since VoID makes no assertions regarding the individuals in the object position of triples. In order to fully cover Requirement 1, Sevod defines the following properties:

Definition 1 (Triple Pattern) *The properties `svd:subjectRegexPattern`, `svd:predicateRegexPattern`, and `svd:objectRegexPattern` denote that all triples in a dataset have subject URIs, predicate URIs, and object URIs/lexical forms that match the given regular expressions.*

```
svd:subjectRegexPattern rdf:type rdf:Property ;
        rdfs:domain void:Dataset ; rdfs:range xsd:string .
svd:predicateRegexPattern rdf:type rdf:Property ;
        rdfs:domain void:Dataset ; rdfs:range xsd:string .
svd:objectRegexPattern rdf:type rdf:Property ;
        rdfs:domain void:Dataset ; rdfs:range xsd:string .
```

To cover Requirement 2, Sevod introduces the `svd:Join` class. A `svd:Join` instance expresses that two `void:Dataset` instances can be joined

Definition 2 (Join) *The `svd:joins` property links a `svd:Join` instance with the `void:Dataset` instances that are joined.*

```
svd:Join  rdf:type rdfs:Class .
svd:joins rdf:type rdf:Property ;
          rdfs:domain svd:Join ;
          rdfs:range void:Dataset .
```

The `svd:joins` property is refined into three sub-properties, so that a `svd:Join` instance can also specify on with triple element (subject, predicate, object) the two datasets join.

Definition 3 (joinSubject) *The `svd:joinSubject` property links a `svd:Join` instance `j` with a `void:Dataset` instance `d1` iff*

- *there is also a triple `j svd:joins d2`; and*
- *for all joinable triples `(S P O)` of `d1`, the corresponding joined triples in `d2` have `S` as one of their elements.*

```
svd:joinsSubject rdfs:subPropertyOf svd:joins .
```

Definition 4 (joinPredicate) *The `svd:joinPredicate` property links a `svd:Join` instance `j` with a `void:Dataset` instance `d1` iff*

- *there is also a triple `j svd:joins d2`; and*
- *for all joinable triples `(S P O)` of `d1`, the corresponding joined triples in `d2` have `P` as one of their elements.*

```
svd:joinsPredicate rdfs:subPropertyOf svd:joins .
```

Definition 5 (joinObject) *The `svd:joinObject` property links a `svd:Join` instance `j` with a `void:Dataset` instance `d` iff*

- *there is also a triple `j svd:joins d2`; and*

– *for all joinable triples* `(S P O)` *of* `d1`*, the corresponding joined triples in* `d2` *have* `O` *as one of their elements.*

```
svd:joinsObject rdfs:subPropertyOf svd:joins .
```

Different join patterns can be expressed using these properties, such as stars (using `svd:joinSubject` only), paths (using both `svd:joinObject` and `svd:joinSubject`), and sinks (using `svd:joinObject` only).

Naturally, `svd:Join` instances hold statistics about the selectivity of the join. To cover Requirement 4, 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 instances of this `svd:SelectivityValue` class 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.

Definition 6 (selectivity) *The* `svd:selectivity` *property links an instance of* `svd:Join` *with an instance of* `svd:SelectivityValue` *that is (or estimates or approximates) the selectivity of the join denoted by the* `svd:Join` *instance. Instances of the* `svd:SelectivityValue` *class denote an exact, estimated, or approximated measurement.* `svd:SelectivityValue` *instances must have the* `rdf:value` *property with a filler that is an* `xds:integer` *that is either the value itself (if the measurement is exact and certain) or a value that is appropriate to use by applications that do not take into account uncertainty or approximation parameters. Where appropriate, uncertainty or approximation parameters are given by application-specific properties.*

```
svd:selectivity rdf:type rdf:Property ;
                rdfs:domain svd:Join ;
                rdfs:range svd:SelectivityValue .
svd:SelectivityValue rdf:type rdfs:Class .
```

Join Example To give an example, consider the following SPARQL query which retrieves from the New York Times dataset topic pages about places in Greece, using links to Geonames to identify which of the places in the New York Times dataset are in Greece:

```
SELECT ?Page WHERE
{
  ?X nyt:topicPage ?Page .
  ?X owl:sameAs ?G .
  ?G geonames:countryCode "GR" .
}
```

Let us assume that two SPARQL endpoints serving relevant data can be identified, based on `void:property` annotations: the `nyt:topicPage` predicate

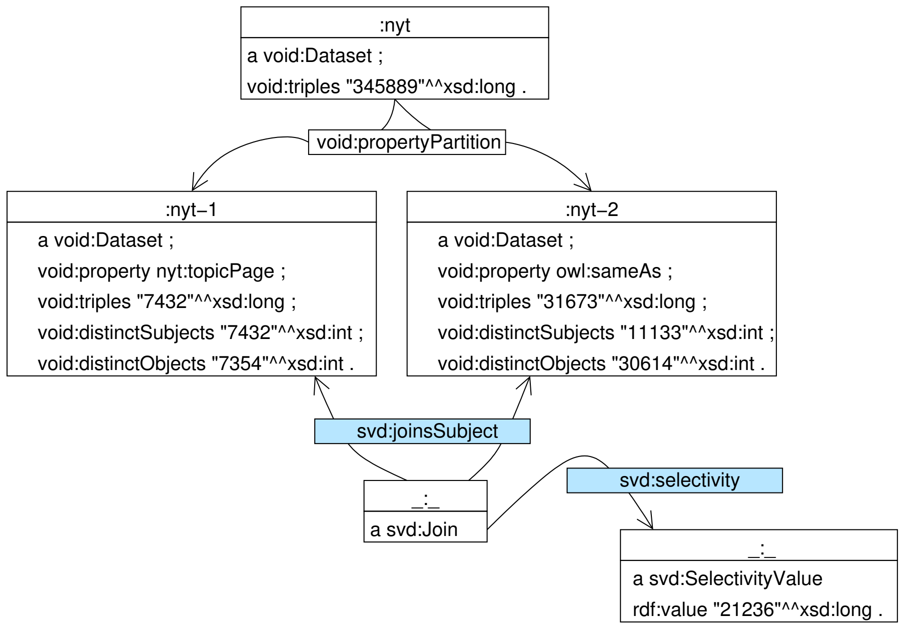
:nyt
a void:Dataset ;
void:triples "345889"^^xsd:long .
void:propertyPartition
:nyt−1
a void:Dataset ;
void:property nyt:topicPage ;
void:triples "7432"^^xsd:long ;
void:distinctSubjects "7432"^^xsd:int ;
void:distinctObjects "7354"^^xsd:int .
:nyt−2
a void:Dataset ;
void:property owl:sameAs ;
void:triples "31673"^^xsd:long ;
void:distinctSubjects "11133"^^xsd:int ;
void:distinctObjects "30614"^^xsd:int .
svd:joinsSubject
_:_
a svd:Join
svd:selectivity
_:_
a svd:SelectivityValue
rdf:value "21236"^^xsd:long .

Fig. 2. Example usage of `svd:Join`.

only appears in the New York Times endpoint and the `geonames:countryCode` predicate only appears in the Geonames endpoint. For the, more ubiquitous, `owl:sameAs` predicate finer-grained description that gives information about about the subjects and objects is used to point at the New York Times endpoint.

One piece of information needed for efficient query execution planning is the cardinality of the query fragment that will be executed at each endpoint. Figure 2 gives an example of how `svd:Join` instances represent statistics about joins of triple patterns.

4.2 Partitions

VoID defines the `void:subset` property for structuring the overall dataset into homogeneous subsets. This partially covers Requirement 3 by defining the vocabulary needed for expressing inclusion hierarchies of subsets. To fully cover Requirement 3, Sevod introduces classes and properties that express that the subsets of a given dataset are exhaustive and no other subsets exist.

Definition 7 (Partition) *`svd:Partition` denotes that a set of `void:Dataset` instances 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.*

```
svd:Partition rdf:type rdfs:Class .
```

Definition 8 (partitions) *`svd:partitions` is the* functional *property that links a `svd:Partition` instance with the `void:Dataset` for which it is a partition.*

```
svd:partitions rdf:type rdf:Property ;
               rdfs:domain svd:Partition ;
               rdfs:range void:Dataset .
```

Definition 9 (part) *The `svd:part` property links a `svd:Partition` instance with each of the `void:Dataset` instances that make up the partition. 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.*

```
svd:part rdf:type rdf:Property ;
         rdfs:domain svd:Partition ;
         rdfs:range void:Dataset .
```

Partition Example Building up on our previous example, Figure 3 gives a more detailed description of the contents of `nyt2`, the `owl:sameAs` subset of the New York Time dataset. As already hinted in the discussion of joins, details about the subjects and objects of ubiquitous predicates such as `rdf:type` or `owl:sameAs` are needed to avoid involving irrelevant datasets in a query. Figure 3 shows how `svd:Partition` instances can be used to express two partitionings of the same data along two facets (subject namespace and object namespace).

5 Generating Sevod Descriptions

Detailed instance-level descriptions are cumbersome to acquire and maintain, ruling out manual curation. The process can, however, be automated by observing the responses to queries. SWOOGLE [11], RDFStats [16], LODStats [4] and STRHist [28] generate rich enough statistics to populate the VoID and Sevod models. Useful as they might be, these estimators of what lies behind a SPARQL endpoint are developed to work around the lack of an authoritative data source description by the data source itself. Although providing a DCAT or VoID description is becoming increasingly popular, these are restricted to general metadata about the dataset and schema-level descriptions, including when VoID is used.

We present here Sevod Scraper,[1] a tool which extracts Sevod metadata directly from the actual data, operating over an RDF dump file of the dataset. Sevod Scraper is meant to be used by the data providers to prepare and maintain dataset descriptions to be published together with the actual data.

[1] Open source, available at `https://github.com/semagrow/sevod-scraper`

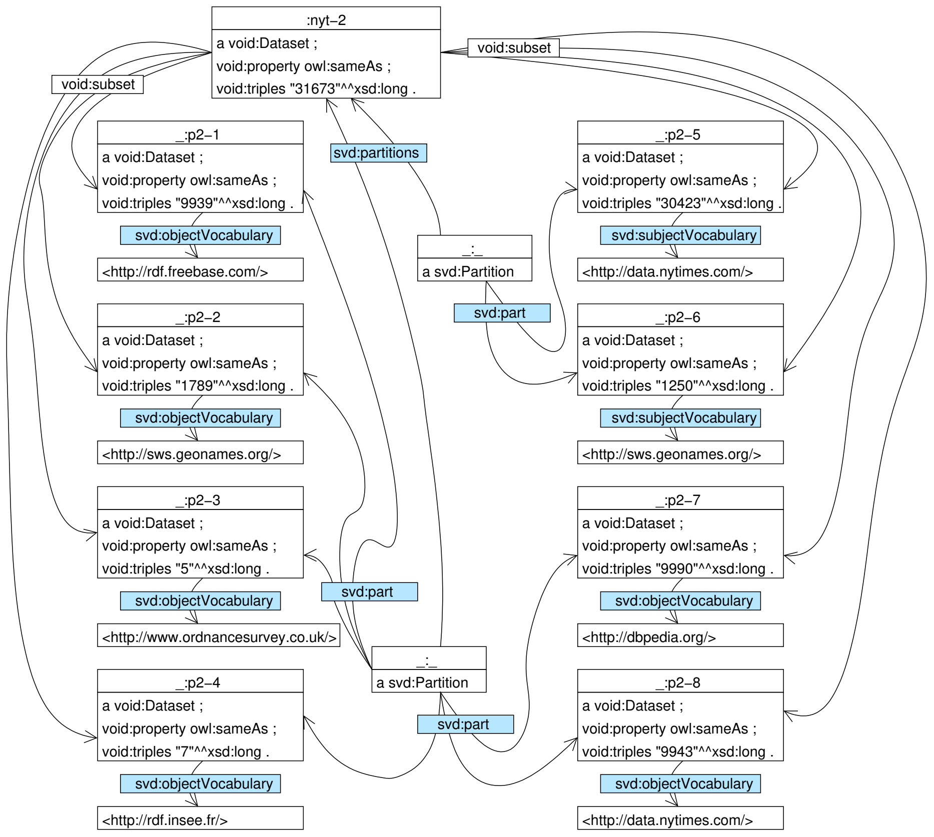
:nyt–2
a void:Dataset ;
void:property owl:sameAs ;
void:triples "31673"^^xsd:long .
void:subset
void:subset
svd:partitions
_:p2–1
a void:Dataset ;
void:property owl:sameAs ;
void:triples "9939"^^xsd:long .
svd:objectVocabulary
<http://rdf.freebase.com/>
_:p2–2
a void:Dataset ;
void:property owl:sameAs ;
void:triples "1789"^^xsd:long .
svd:objectVocabulary
<http://sws.geonames.org/>
_:p2–3
a void:Dataset ;
void:property owl:sameAs ;
void:triples "5"^^xsd:long .
svd:objectVocabulary
<http://www.ordnancesurvey.co.uk/>
_:p2–4
a void:Dataset ;
void:property owl:sameAs ;
void:triples "7"^^xsd:long .
svd:objectVocabulary
<http://rdf.insee.fr/>
_:_
a svd:Partition
svd:part
_:_
a svd:Partition
svd:part
svd:part
_:p2–5
a void:Dataset ;
void:property owl:sameAs ;
void:triples "30423"^^xsd:long .
svd:subjectVocabulary
<http://data.nytimes.com/>
_:p2–6
a void:Dataset ;
void:property owl:sameAs ;
void:triples "1250"^^xsd:long .
svd:subjectVocabulary
<http://sws.geonames.org/>
_:p2–7
a void:Dataset ;
void:property owl:sameAs ;
void:triples "9990"^^xsd:long .
svd:objectVocabulary
<http://dbpedia.org/>
_:p2–8
a void:Dataset ;
void:property owl:sameAs ;
void:triples "9943"^^xsd:long .
svd:objectVocabulary
<http://data.nytimes.com/>

Fig. 3. Example usage of `svd:Partition`.

Sevod Scraper partitions the dataset into several (possibly overlapping) subsets. Provided that we need to generate metadata not only for properties, but for subjects and objects as well, we generate one subset for each property, one subset for *some* subject URI prefixes and one subset for *some* object URI prefixes. The decision to use prefixes to specify URI ranges is based on the observation [16] that string (including URI) range estimations can be given:

– using one bucket for each distinct string or storing the range as a set of strings, resulting in large histograms;
– using a hash function to reduce the number of distinct strings. However, hashing the string representation of URIs fails to take into account the semantic similarity between resources, so it is unlikely that a universally good function can be identified [13]; or
– by reducing strings to prefixes.

Table 1. FedBench datasets statistics and number of triples of the metadata obtained by the Sevod Scraper using the default parameters.

Dataset	numb. triples	numb. properties	numb. distinct subjects	numb. distinct objects	Metadata triples
ChEBI	4,775,935	49	51,297	772,930	437
Dbpedia	42,852,838	1,080	9,496,685	13,518,436	6,282
DrugBank	520,252	141	20,511	408,964	1,142
GeoNames	107,953,314	48	7,480,534	35,454,090	367
Jamendo	1,052,868	49	336,745	593,727	2,777
KEGG	1,094,059	42	35,080	940,041	426
LMDB	6,151,225	244	695,220	2,053,739	2,143
NYT	338,426	57	22,486	191,288	420

Determining which URI prefixes we want to keep balances between the size and detail of the resulting metadata. To make this decision easier for the user, the user sees parameters from the perspective of size limits and the tool decides how to adhere to these by selecting which URI prefixes to include in the description.

Table 1 gives a sense of the size of the descriptions generated by Sevod Scraper using default parameters. The table lists the number of triples used to describe well-known datasets from the FedBench suite [24].

Given a user-provided bound B, if a number of k URI prefixes s.t. $k > B$ have the same prefix, we would like to replace these URI prefixes with their longest common prefix. Sevod Scraper uses for this reason two *path tries*, one for the subject and the other for the object URIs. In these path tries each edge corresponds to a path component, allowing also the $\star$ special component to denote, unsurprisingly, any number of characters or an empty string. If during any insert we have a situation that one node contains more than B children, some of its children will be combined to a single node using their common URI prefix. Instead of subject and object URIs, these nodes are specified using the `svd:subjectRegexPattern` and `svd:objectRegexPattern` properties.

This hierarchy of trie nodes is then represented as a hierarchy of VoID subsets. For each subset, Sevod Scraper extracts the standard VoID statistics, namely the number of triples properties, distinct subjects and distinct objects of the subset. Especially for the property subsets, the *authority* component[2] of all subjects and objects is extracted and added to the description using the `svd:subjectVocabulary` and `svd:objectVocabulary` terms. Finally, the scraper also computes join selectivity metadata. For every pair of property subsets, the tool computes the selectivity values of the star (i.e., subject-subject), the sink (i.e., object-object) and the path (i.e., object-subject) joins between these subsets.

[2] The URI component after the scheme, if foreseen by the scheme; for the http scheme used here, the authority is the host name between `http://` and the immediately following /

6 Conclusions

We presented the *Sevod* vocabulary, an extension of VoID that specifically addresses the aspects of dataset description that are relevant to efficient and transparent federated SPARQL query processing. The extension is designed to address federated SPARQL requirements extracted by analysing the information needs of current federated SPARQL query processors.

Sevod is the first vocabulary that makes explicit and share-able the data summaries used to optimize query processing. The adoption and maintenance of Sevod can facilitate the transfer of optimization methods between the databases and the Semantic Web communities. But of more interest to the Semantic Web is the ability to publish these detailed data summaries, allowing endpoints to provide the metadata needed to be discovered as relevant to federated queries and to be included in an efficient query execution plan. In this manner, federated querying can be made as efficient as the querying of distributed databases while maintaining the dynamic and decentralized nature of the Semantic Web.

In order to realize this ability, we have started by making it easy for data providers to publish Sevod descriptions. We developed the Sevod Scraper tool that automates the generation Sevod descriptions of varying detail by setting the intended description size. Future work on the Scraper will be on using past query load to make more informed decisions about where the descriptions should be more detailed and where they can be left more shallow, in order to adhere to the publishers' maximum description size requirement with minimal loss of query plan efficiency. This will be based on our previous work on workload aware histogram construction on the client side [28], but extended to take advantage of the direct access to the data enjoyed by the publisher.

The next step will be to develop tools for serializing and deserializing Sevod descriptions for the most prominent, current federated query processors. The expectation is that this will close the loop between data publishers and data consumers, and kick-start the adoption of Sevod.

Acknowledgements

The work described here has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644564. For more details, please visit https://www.big-data-europe.eu

Bibliography

[1] Aboulnaga, A., Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: Proceedings of the 1999 ACM International Conference on Management of Data (SIGMOD '99). pp. 181–192. ACM, New York, NY, USA (1999), http://doi.acm.org/10.1145/304182.304198

[2] Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets with the VoID vocabulary. W3C Interest Group Note, 3 March 2011 (Mar 2011), http://www.w3.org/TR/void

[3] Archer, P., Shukair, G.: Asset description metadata schema (ADMS). W3C Working Group Note, 1 August 2013 (2013), http://www.w3.org/TR/vocab-adms, this version of the WG Note is based on M. Dekkers (ed., 2012), ADMS Draft Specification, ISA Deliverable D1.1. URL https://joinup.ec.europa.eu/asset/adms

[4] Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats – an extensible framework for high-performance dataset analytics. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW '12). pp. 353–362. Springer-Verlag, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33876-2_31

[5] Basca, C., Bernstein, A.: Querying a messy web of data with Avalanche. Journal of Web Semantics 26(1), 1–28 (2014), http://dx.doi.org/10.1016/j.websem.2014.04.002

[6] Ben Ellefi, M., Bellahsene, Z., Breslin, J., Demidova, E., Dietze, S., Szymanski, J., Todorov, K.: RDF dataset profiling: A survey of features, methods, vocabularies and applications. Semantic Web Journal, accepted for publication (2017).

[7] Bruno, N., Chaudhuri, S.: Exploiting statistics on query expressions for optimization. In: Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD '02). pp. 263–274. ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/564691.564722

[8] Cem Ozkan, E., Saleem, M., Dogdu, E., Ngonga Ngomo, A.C.: UPSP: unique predicate-based source selection for SPARQL endpoint federation. In: Proceedings of 3rd International Workshop on Dataset Profiling and Federated Search for Linked Data (PROFILES 2016), held on 30 May 2016 at ESWC 2016, Anissaras, Crete, Greece (2016)

[9] Charalambidis, A., Troumpoukis, A., Konstantopoulos, S.: SemaGrow: Optimizing federated SPARQL queries. In: Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015), Vienna, Austria, 15-18 September 2015 (2015), http://dx.doi.org/10.1145/2814864.2814886

[10] 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). pp. 34–43 (1998)

[11] Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: A search and metadata engine for the semantic web. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management. pp. 652–659. CIKM '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1031171.1031289

[12] Görlitz, O., Staab, S.: SPLENDID: SPARQL endpoint federation exploiting VOID descriptions. In: Proceedings of the 2nd International Workshop on

Consuming Linked Data (COLD 2011), Bonn, Germany, October 23, 2011. CEUR Workshop Proceedings, vol. 782 (2011)

[13] Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA, 26-30 April 2010 (2010)

[14] Ioannidis, Y.: The history of histograms (abridged). In: Proceedings of the 29th International Conference on Very Large Databases (VLDB 2003), Berlin, Germany (2003), ten-Year Best Paper Award

[15] Kaushik, R., Suciu, D.: Consistent histograms in the presence of distinct value counts. Proc. VLDB Endow. 2(1), 850–861 (Aug 2009), `http://dx.doi.org/10.14778/1687627.1687723`

[16] Langegger, A., Wöss, W.: RDFStats – an extensible RDF statistics generator and library. In: 23rd International Workshop on Database and Expert Systems Applications. pp. 79–83. IEEE Computer Society, Los Alamitos, CA, USA (2009)

[17] Maali, F., Erickson, J., Archer, P.: Data catalog vocabulary (DCAT). W3C Recommendation 16 January 2014 (Jan 2014), `http://www.w3.org/TR/vocal-dcat`

[18] Montoya, G., Skaf-Molli, H., Hose, K.: The Odyssey approach for optimizing federated SPARQL queries. In: 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, 23–25 October 2017 (2017), preprint available at `https://arxiv.org/abs/1705.06135`

[19] Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) Proceedings of the 5th European Semantic Web Conference (ESWC 2008), Tenerife, Spain, 1–5 June 2008. Lecture Notes in Computer Science, vol. 5021 (2008)

[20] Rakhmawati, N.A., Hausenblas, M.: On the impact of data distribution in federated SPARQL queries. In: Proceedings of the Sixth IEEE International Conference on Semantic Computing (ICSC 2012). pp. 255–260 (2012)

[21] Rakhmawati, N.A., Umbrich, J., Karnstedt, M., Hasnain, A., Hausenblas, M.: Querying over federated sparql endpoints: A state of the art survey. Tech. rep., DERI (Jun 2013), `http://arxiv.org/abs/1306.1723`

[22] Saleem, M., Ngonga Ngomo, A.C.: HiBISCuS: Hypergraph-based source selection for SPARQL endpoint federation. In: Proceedings of the 11th ESWC Conference, Anissaras, Crete, Greece, 25–29 May 2014. pp. 176–191 (2014), `http://dx.doi.org/10.1007/978-3-319-07443-6_13`

[23] Saleem, M., Ngonga Ngomo, A.C., Xavier Parreira, J., Deus, H.F., Hauswirth, M.: DAW: Duplicate-aware federated query processing over the Web of Data. In: Proceedings of the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, 21–25 October 2013, Part I. Lecture Notes in Computer Science, vol. 8218. Springer (2013), `https://doi.org/10.1007/978-3-642-41335-3_36`

[24] Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing.

In: Proceedings of the 10th International Semantic Web Conference (ISWC 2011), Bonn, Germany, 23–27 October 2011. Lecture Notes in Computer Science, vol. 7031, pp. 585–600. Springer (2011), http://doi.org/10.1007/978-3-642-25073-6_37

[25] Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: A federation layer for distributed query processing on Linked Open Data. In: Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Heraklion, Crete, Greece, May 29 – June 2, 2011. Lecture Notes in Computer Science, vol. 6644, pp. 481–486. Springer (2011)

[26] Srivastava, U., Haas, P.J., Markl, V., Kutsch, M., Tran, T.M.: ISOMER: Consistent histogram construction using query feedback. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE '06). IEEE Computer Society, Washington, DC, USA (2006), http://dx.doi.org/10.1109/ICDE.2006.84

[27] Wang, X., Tiropanis, T., Davis, H.C.: LHD: optimising linked data query processing using parallelisation. In: Proceedings of Linked Data on the Web (LDOW 2013), Rio de Janeiro, 14 May 2013 (2013)

[28] Zamani, K., Charalambidis, A., Konstantopoulos, S., Zoulis, N., Mavroudi, E.: Workload-aware self-tuning histograms for the Semantic Web. Transactions on Large Scale Data and Knowledge-Centered Systems 28 (Sep 2016), http://dx.doi.org/10.1007/978-3-662-53455-7_6, published as LNCS 9940