LODStats – An Extensible Framework for High-performance Dataset Analytics

Sören Auer, Jan Demter, Michael Martin, Jens Lehmann

AKSW/BIS, Universität Leipzig, Germany, http://aksw.org
{lastname}@informatik.uni-leipzig.de

Abstract. One of the major obstacles for a wider usage of web data is the difficulty to obtain a clear picture of the available datasets. In order to reuse, link, revise or query a dataset published on the Web it is important to know the structure, coverage and coherence of the data. In order to obtain such information we developed LODStats – a statement-stream-based approach for gathering comprehensive statistics about datasets adhering to the Resource Description Framework (RDF). LODStats is based on the declarative description of statistical dataset characteristics. Its main advantages over other approaches are a smaller memory footprint and significantly better performance and scalability. We integrated LODStats with the CKAN dataset metadata registry and obtained a comprehensive picture of the current state of a significant part of the Data Web.

1 Introduction

For assessing the state of the Web of Data in general, for evaluating the quality of individual datasets as well as for tracking the progress of Web data publishing and integration it is of paramount importance to gather comprehensive statistics on datasets describing their internal structure and external cohesion. We even deem the difficulty to obtain a clear picture of the available datasets to be a major obstacle for a wider usage of the Web of Data. In order to reuse, link, revise or query a dataset published on the Web it is important to know the structure, coverage and coherence of the data.

In this article we present LODStats – a statement-stream-based approach for gathering comprehensive statistics from resources adhering to the Resource Description Framework (RDF). One rationale for the development of LODStats is the computation of statistics for resources from the Comprehensive Knowledge Archive (CKAN, "The Data Hub"[1]) on a regular basis. Datasets from CKAN are available either serialised as a file (in RDF/XML, N-Triples and other formats) or via SPARQL endpoints. Serialised datasets containing more than a few million triples tend to be too large for most existing analysis approaches as the size of the dataset or its representation as a graph exceeds the available main memory, where the complete dataset is commonly stored for statistical processing. LODStats' main advantage when compared to existing approaches is its

[1] http://thedatahub.org

superior performance, especially for large datasets with many millions of triples, while keeping its extensibility with novel analytical criteria straightforward. It comes with a set of 32 different statistics, amongst others are those covering the statistical criteria defined by the Vocabulary of Interlinked Datasets [1] (VoID). Examples of available statistics are property usage, vocabulary usage, datatypes used and average length of string literals. Our implementation is written in *Python* and available as a module for integration with other projects.

Obtaining *comprehensive* statistical analyses about datasets facilitates a number of important use cases and provides crucial benefits. These include:

Quality analysis. A major problem when using Linked Data is quality. However, the quality of the datasets itself is not so much a problem as assessing and evaluating the expected quality and deciding whether it is sufficient for a certain application. Also, on the traditional Web we have very varying quality, but means were established (e.g. page rank) to assess the quality of information on the document web. In order to establish similar measures on the Web of Data it is crucial to assess datasets with regard to incoming and outgoing links, but also regarding the used vocabularies, properties, adherence to property range restrictions, their values etc. Hence, a statistical analysis of datasets can provide important insights with regard to the expectable quality.

Coverage analysis. Similarly important as quality is the coverage a certain dataset provides. We can distinguish *vertical* and *horizontal* coverage. The former providing information about the properties we can expect with the data instances, while the later determines the range (e.g. spatial, temporal) of identifying properties. In the case of spatial data, for example, we would like to know the region the dataset covers, which can be easily derived from minimum, maximum and average of longitude and latitude properties (horizontal coverage). In the case of organizational data we would like to determine whether a dataset contains detailed address information (vertical coverage).

Privacy analysis. For quickly deciding whether a dataset potentially containing personal information can be published on the Data Web, we need to get a quick overview on the information contained in the dataset without looking at every individual data record. An analysis and summary of all the properties and classes used in a dataset can quickly reveal the type of information and thus prevent the violation of privacy rules.

Link target identification. Establishing links between datasets is a fundamental requirement for many Linked Data applications (e.g. data integration and fusion). Meanwhile, there are a number of tools available which support the automatic generation of links (e.g. [11,10]). An obstacle for the broad use of these tools is, however, the difficulty to identify suitable link targets on the Data Web. By attaching proper statistics about the internal structure of a dataset (in particular about the used vocabularies, properties etc.) it will be dramatically simplified to quickly identify suitable target datasets for linking. For example,

the usage of longitude and latitude properties in a dataset indicates that this dataset might be a good candidate for linking spatial objects. If we additionally know the minimum, maximum and average values for these properties, we can even identify datasets which are suitable link targets for a certain region.

The contributions of our work are in particular: **(1)** A *declarative representation of statistical dataset criteria*, which allows a straightforward extension and implementation of analytical dataset processors and additional criteria (Section 2.1). **(2)** A comprehensive *survey of statistical dataset criteria* derived from RDF data model elements, survey and combination of criteria from related work and expert interviews (Section 2.2). **(3)** A LODStats *reference implementation* (Section 3), which outperforms the state-of-the-art on average by 30-300% and which allows to generate a statistic view on the complete Data Web in just a few hours of processing time. We provide an overview on related work in the areas of RDF statistics, stream processing and Data Web analytics in Section 4 and conclude with an outlook on future work in Section 5.

2 Statistical Criteria

In this section we devise a definition for statistical dataset criteria, survey analytical dataset statistics and explain how statistics can be represented in RDF.

2.1 Definition

The rationale for devising a declarative definition of statistical criteria is that it facilitates *understandability and semantic clarity* (since criteria are well defined without requiring code to be analyzed to understand what is actually computed), *extensibility* (as a statistical dataset processor can generate statistics which are defined after design and compile time) and to some extent *scalability* (because the definition can be designed such that statistical analyses can be performed efficiently). This definition formalizes our concept of a statistical criteria:

Definition 1 (Statistical criteria). *A statistical criterion is a triple* (F, D, P)*, where:*

- F *is a SPARQL filter condition.*
- D *is a data structure for storing intermediate results and a description how to fill this data structure with values from the triple stream after applying* F*.*
- P *is a post-processing filter operating on the data structure* D*.*

Explanations: F serves as selector to determine whether a certain triple triggers the alteration of a criteria. We use single triple patterns (instead of graph patterns) and additional filter conditions to allow an efficient stream processing of the datasets. The dataset is processed triple by triple and each triple read is matched against each triple pattern of each criterion.[2] With the filter condition

[2] In fact many criteria are triggered by the same triple patterns and have thus to be tested only once, which is used as an optimization in our implementation.

we can further constrain the application of a criteria, e.g. only to triples having a literal as object (using `isLiteral(?o)`).

For the data structure D, we usually make use of some counter code referring to variables of the triple pattern, for example, `H[ns(?subject)]++`, where H is a hash map and the function `ns` returns the namespace of the IRI supplied as a parameter. The counter code is an assertion of values to certain elements of the hash table D. Our survey of statistical criteria revealed that simple arithmetics, string concatenation (and in few cases the ternary operator) are sufficient to cover many purposes. Of course there are tasks for which LODStats is not suitable, e.g. measuring data duplication via string similarities.

In the simplest case P just returns exactly the values from the data structure D, but it can also retrieve the top-k elements of D or perform certain additional computations. In most cases, however, post-processing is not required.

Example 1 (Subject vocabularies criterion). This example illustrates the statistical criterion *subject vocabularies*, which collects a list of the 100 vocabularies mostly used in the subjects of the triples of an RDF dataset.

- *Criterion name:* Subject vocabularies
- *Description:* Lists all vocabularies used in subjects together with their occurrence
- *Filter clause:* - (empty)
- *Counter data structure:* hash map (initially empty)
- *Counter code:* `H[ns(?subject)]++` (`ns` is a function returning the namespace of the IRI given as parameter, i.e. the part of the IRI after the last occurrence of '/' or '#')
- *Post-processing filter:* `top(H,100)`

Statistical criteria can also be understood as a rule-based formalism. In this case, F is the condition (body of the rule) and D an action (head of the rule). P is only applied after executing such a rule on all triples. In Table 1, we present the more compact rule syntax of our statistical criteria to save space. Statistical criteria are evaluated in our framework according to Algorithm 1. Note that the triple input stream can be derived from various sources, specifically RDF files in different syntactic formats and SPARQL endpoints.

2.2 Statistical criteria survey

In order to obtain a set of statistical criteria which is as complete as possible, we derived criteria from:

(1) *analysing RDF data model elements*, i.e. possible elements as subjects, predicates and objects in an RDF statement, composition of IRIs (comprising namespaces) and literals (comprising datatypes and language tags, **(2)** *surveying and combining statistical criteria from related work* particularly from VoID and RDFStats [9], **(3)** *expert interviews*, which we performed with representatives from the AKSW research group and the LOD2 project.

Algorithm 1: Evaluation of a set of statistical criteria on a triple stream.

```
Data: CriteriaSet CS; TripleInputStream S
forall the Criteria C ∈ CS do
    initialise datastructures
while S.hasNext() do
    Triple T = S.next();
    forall the Criteria C ∈ CS do
        if T satisfies C.T and C.F then
            execute C.D;
            if datastructures exceed threshold then
                purge datastructures;
forall the Criteria C ∈ CS do
    execute C.P and return results
```

While collecting and describing criteria, we put an emphasis on *generality* and *minimality*. Hence, we tried to identify criteria which are as general as possible, so that more specialized criteria can be automatically derived. For example, collecting minimum and maximum values for all numeric and date time property values also allows us to determine the spatial or temporal coverage of the dataset, by just extracting values from the result list for spatial and temporal properties. The 32 statistical criteria that we obtained can be roughly divided in schema and data level ones. A formal representation using the definition above is given in Table 1. A complete list of all criteria and detailed textual explanations is available from the LODStats project page[3].

Schema Level. LODStats can collect comprehensive statistics on the schema elements (i.e. classes, properties) defined and used in a dataset. LODStats is also able to analyse more complex schema level characteristics such as the class hierarchy depth. In this case we store the depth position of each encountered class, for example, in a hash table data structure. In fact, since the position can not be determined when reading a particular `rdfs:subClassOf` triple, we store that the hierarchy depth of the subject is one level more than the one of the object. The exact values then have to be computed in the post-processing step. This example already illustrates that despite its focus on efficiency in certain cases the intermediate data structure or the time required for its post-processing might get very large. For such cases (when certain pre-configured memory thresholds are exceeded), we implemented an approximate statistic where the anticipated least popular information stored in our intermediate data structure is purged. Such a proceeding guarantees that statistics can be computed efficiently even if datasets are adversely structured (i.e. a very large dataset containing deep class hierarchies such as the the NCBI Cancer ontology). Results are in such cases appropriately marked to be approximate.

[3] `http://aksw.org/Projects/LODStats`

	Criterion	Rules (Filter → Action)		Postproc.
1	used classes	`?p=rdf:type && isIRI(?o)`	→ `S += ?o`	–
2	class usage count	`?p=rdf:type && isIRI(?o)`	→ `M[?o]++`	`top(M,100)`
3	classes defined	`?p=rdf:type && isIRI(?s) &&(?o=rdfs:Class||?o=owl:Class)`	→ `S += ?s`	–
4	class hierarchy depth	`?p = rdfs:subClassOf && isIRI(?s) && isIRI(?o)`	→ `G += (?s,?o)`	`hasCycles(G) ? ∞ : depth(G)`
5	property usage		→ `M[?p]++`	`top(M,100)`
6	property usage distinct per subj.		→ `M[?s] += ?p`	`sum(M)`
7	property usage distinct per obj.		→ `M[?o] += ?p`	`sum(M)`
8	properties distinct per subj.		→ `M[?s] += ?p`	`sum(M)/size(M)`
9	properties distinct per obj.		→ `M[?o] += ?p`	`sum(M)/size(M)`
10	outdegree		→ `M[?s]++`	`sum(M)/size(M)`
11	indegree		→ `M[?o]++`	`sum(M)/size(M)`
12	property hierarchy depth	`?p=rdfs:subPropertyOf && isIRI(?s) && isIRI(?o)`	→ `G += (?s,?o)`	`hasCycles(G) ? ∞ : depth(G)`
13	subclass usage	`?p = rdfs:subClassOf`	→ `i++`	–
14	triples		→ `i++`	–
15	entities mentioned		→ `i+=size(iris( {?s,?p,?o}))`	–
16	distinct entities		→ `S+=iris({?s,?p,?o})`	–
17	literals	`isLiteral(?o)`	→ `i++`	–
18	blanks as subj.	`isBlank(?s)`	→ `i++`	–
19	blanks as obj.	`isBlank(?o)`	→ `i++`	–
20	datatypes	`isLiteral(?o)`	→ `M[dtype(?o)]++`	–
21	languages	`isLiteral(?o)`	→ `H[language(?o)]++`	–
22	∅ typed string length	`isLiteral(?o) && datatype(?o)=xsd:string`	→ `i++; len+=len(?o)`	`len/i`
23	∅ untyped string length	`isLiteral(?o) && datatype(?o) = NULL`	→ `i++; len+=len(?o)`	`len/i`
24	typed subj.	`?p = rdf:type`	→ `i++`	–
25	labeled subj.	`?p = rdfs:label`	→ `i++`	–
26	sameAs	`?p = owl:sameAs`	→ `i++`	–
27	links	`ns(?s) != ns(?o)`	→ `M[ns(?s)+ns(?o)]++`	–
28	max per property {int,float,time}	`datatype(?o)={xsd:int| xsd:float|xsd:datetime}`	→ `M[?p]=max( M[?p],?o)`	–
29	∅ per property {int,float,time}	`datatype(?o)={xsd:int| xsd:float|xsd:datetime}`	→ `M[?p]+=?o; M2(?p)++`	`M[?p]/M2[?p]`
30	subj. vocabularies		→ `M[ns(?s)]++`	–
31	pred. vocabularies		→ `M[ns(?p)]++`	–
32	obj. vocabularies		→ `M[ns(?o)]++`	–

Table 1. Definition of schema level statistical criteria. Notation conventions: G = directed graph; M = map; S = set; i, len = integer. `+=` and `++` denote standard additions on those structures, i.e. adding edges to a graph, increasing the value of the key of a map, adding elements to a set and incrementing an integer value. `iris` takes a set as input and all elements of it, which are IRIs. `ns` returns the namespace of a resource . `len` returns the length of a string. `language` and `dtype` return datatype resp. language tag of a literal. `top(M,n)` return first `n` elements of the map `M`.

Data Level. In addition to schema level statistics we collect all kinds of data level ones. As the simplest statistical criterion, the number of all triples seen is counted. Furthermore, entities (triples with a resource as subject), triples with blanks as subject or object, triples with literals, typed subjects, labeled subjects and triples defining an `owl:sameAs` link are counted. A list of the different datatypes used for literals, i.e. string, integer etc., can be compiled by LODStats. Data about which languages and how often they are used with literals by the dataset is made available to the user. If string or untyped literals are used by the dataset, their overall average string length can be computed. Statistics about internal and external links (i.e. triples where the namespace of the object differs from the subject namespace) are also collected. Spatial and temporal statistics can be easily computed using the min/max/avg. per integer/float/time property.

2.3 Representing Dataset Statistics with VoiD and Data Cube

Currently, 32 different statistical criteria can be gathered with LODStats. To represent these statistics we used the *Vocabulary of Interlinked Datasets* (VoID, [1]) and the *Data Cube Vocabulary* [8]. VoID is a vocabulary for expressing metadata about RDF datasets comprising properties to represent a set of relatively simple statistics. DataCube is a vocabulary based on the SDMX standard and especially designed for representing complex statistics about observations in a multidimensional attribute space. Due to the fact that many of the listed 32 criteria are not easily representable with VoID we encode them using the Data Cube Vocabulary, which allows to encode statistical criteria using arbitrary attribute dimensions. To support linking between a respective `void:Dataset` and a `qb:Observation`, we simply extended VoID with the object property `void-ext:observation`.

Using the default configuration of LODStats the following criteria are gathered and represented using the listed VoID properties: *triples* (`void:triples`), *entities* (`void:entities`), *distinct resources* (`void:distinctSubjects`), *distinct objects* (`void:distinctObjects`), *classes defined* (`void:classes`), *properties defined* (`void:properties`), *vocabulary usage* (`void:vocabulary`). Additional criteria such as *class/property usage*, *class/property usage distinct per subject*, *property usage distinct per object* are represented using `void:classPartition` and `void:propertyPartition` as subsets of `void:document`.

3 LODStats Architecture and Implementation

LODStats is written in Python and uses the Redland library [4] and its bindings to Python to parse files and process them statement by statement. Resources reachable via HTTP, contained in archives (e.g. zip, tar) or compressed with gzip or bzip2 are transparently handled by LODStats. *SPARQLWrapper*[4] is used for augmenting LODStats with support for SPARQL endpoints. Statistical criteria may also have an associated equivalent SPARQL query that will be used in

[4] `http://sparql-wrapper.sourceforge.net/`

case a certain dataset is not available in serialised form. Classes for gathering statistics support this by implementing a method with one or more SPARQL queries that produce results equivalent to those gathered using the statement-based approach. However, during our experiments it turned out that this variant is relatively error prone due to timeouts and resource limits. A second option we experimentally evaluated is to emulate the statement-based approach by passing through all triples stored in the SPARQL endpoint using queries retrieving a certain dataset window with `LIMIT` and `OFFSET`. Note that this option seems to work generally in practice, but requires the use of an (arbitrary) `ORDER BY` clause to return deterministic results. Consequently, this option also did not always render satisfactory results, especially for larger datasets due to latency and resource restrictions. However, our declarative criteria definition allows SPARQL endpoint operators to easily generate statistics right on their infrastructure. We plan to discuss support for the generation, publishing, advertising and discovery of statistical metadata by directly integrating respective modules into triple store systems.

LODStats has been implemented as a Python module with simple calling conventions, aiming at general reusability. It is available for integration with the Comprehensive Knowledge Archiving Network (CKAN), a widely used dataset metadata repository, either as a patch or as an external web application using CKAN's API. Integration with other (non-Python) projects is also possible via a command line interface and a RESTful web service.

4 Related Work

In this section we provide an overview on related work in the areas of RDF statistics, stream processing and Data Web analytics.

RDF Statistics. Little work has been done in the area of RDF statistics, mainly including *make-void*[5] and *rdfstats* [9]. *Make-void* is written in Java and utilizes the Jena toolkit to import RDF data and generate statistics conforming to VoID using SPARQL queries via Jena's ARQ SPARQL processor. *RDFStats* uses the same programming environment and library for processing RDF as make-void. It does not aim at generating VoID, but rather uses the collected statistics for optimising query execution.

RDF Stream Processing and SPARQL Stream Querying. Processing data datum by datum is commonplace in general (e.g. SAX[6]) and especially was so before the advent of computing machinery with larger main memory and software facilitating the retrieval of specific data. Processing RDF resources statement by statement is not a new concept as well; most RDF serialisation parsers internally work this way. Redland [4] additionally offers a public interface (`Parser.parse_as_stream`) for parsing and working with file streams in this manner.

[5] `https://github.com/cygri/make-void`
[6] `http://www.saxproject.org`

Furthermore, there are approaches for querying RDF streams with SPARQL such as *Continous SPARQL* (C-SPARQL) [3] and *Streaming SPARQL* [6]. Both represent adaptions of SPARQL which enable the observation of RDF stream windows (most recent triples of RDF streams). A further advancement, *EP-SPARQL* [2], is a unified language for event processing and stream reasoning. The main difference between these approaches and LODStats is that LODStats works on single triple patterns, thus does not require processing windows and offers even higher performance while limiting the processing capabilities. Such a processing limitation is sensible for the generation of dataset statistics, but not acceptable for general purpose stream processing.

Data Web analytics. Since 2007 the growth of the Linked Open Data Cloud is being examined. As a result, a visualization of the Linked Data space, its distribution and coherence are periodically published[7]. The main difference to `stats.lod2.eu` is that this information is partially entered manually in *The Data Hub* and updated infrequently, whereas using LODStats we can perform those calculations automatically. Both approaches lead to interesting information on the structure of the LOD cloud. Statistics about the Linked Data Cloud are summarized in [5], containing multiple aspects such as the usage of vocabularies as well as provenance and licensing information in published datasets. Furthermore, a comprehensive analysis of datasets indexed by the semantic web search engine Sindice[8] was published in [7]. It also contains a set of low-level statistical information such as the amount of entities, statements, literals and blank nodes. Unlike Sindice, which also indexes individual Web pages and small RDF files, LODStats focuses on larger datasets.

Another related area is data quality. One of the purposes of collecting statistics about data is to improve their quality. For instance, vocabulary reuse is improved, since `stats.lod2.eu` allows to easily check which existing classes and properties are widely used already. Since we do not directly pursue an improvement of data quality in this article, but focus on dataset analytics, we refrain from referring to the large body of literature in the data quality area.

5 Conclusions

With LODStats we developed an extensible and scalable approach for large-scale dataset analytics. By integrating LODStats with CKAN (the metadata home of the LOD Cloud) we aim to provide a timely and comprehensive picture of the current state of the Data Web. It turns out that many other statistics are actually overly optimistic – the amount of really usable RDF data on the Web might be an order of magnitude lower than what other statistics suggest. Frequent problems that we encountered are in particular outages and restricted access to or non-standard behavior of SPARQL endpoints, serialization and packaging

[7] `http://lod-cloud.net/state/`
[8] `http://sindice.com/`

problems, syntax errors and outdated download links. Some of these problems can also be attributed to CKAN entries not being sufficiently up-to-date. However, even though a dataset might be available at a different URL this poor user experience might be one of the reasons why Linked Data technology is still often perceived being too immature for industrial-strength applications. We hope that LODStats can contribute to overcoming this obstacle by giving dataset providers and stakeholders a more qualitative, quantitative and timely picture of the state of the Data Web as well as its evolution.

Future Work. While LODStats already delivers decent performance, a direct implementation of the approach in C/C++ and parallelization efforts might render additional performance boosts. Preliminary experimentation shows that such an approach would result in an additional performance increase by a factor of 2-3. Our declarative criteria definition allows SPARQL endpoint operators to easily generate statistics right on their infrastructure. We plan to integrate support for the generation, publishing, advertising and discovery of statistical metadata by directly integrating respective modules into the triple store systems (our Python implementation can serve as reference implementation). Last but not least we plan to increase the support for domain specific criteria, which can be defined, published and discovered using the Linked Data approach itself. For example, for geo-spatial datasets criteria could be defined, which determine the distribution or density of objects of a certain type in certain regions.

References

1. K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets. In *2nd WS on Linked Data on the Web*, Madrid, Spain, April 2009.
2. D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*. ACM, 2011.
3. D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Querying rdf streams with C-SPARQL. *SIGMOD Record*, 39(1):20–26, 2010.
4. D. Beckett. The design and implementation of the redland rdf application framework. In *Proc. of 10th Int. World Wide Web Conf.*, pages 449–456. ACM, 2001.
5. C. Bizer, A. Jentzsch, and R. Cyganiak. State of the LOD Cloud, Version 0.3, September 2011.
6. A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL - extending SPARQL to process data streams. In *European Semantic Web Conf.*, LNCS. Springer, 2008.
7. S. Campinas, D. Ceccarelli, T. E. Perry, R. Delbru, K. Balog, and G. Tummarello. The Sindice-2011 dataset for entity-oriented search in the web of data. In *1st Int. Workshop on Entity-Oriented Search (EOS)*, 26–32 2011.
8. R. Cyganiak, D. Reynolds, and J. Tennison. The rdf data cube vocabulary, 2012. http://www.w3.org/TR/vocab-data-cube/.
9. A. Langegger and W. Wöß. Rdfstats - an extensible rdf statistics generator and library. In *DEXA Workshops*, pages 79–83. IEEE Computer Society, 2009.
10. A.-C. Ngonga Ngomo and S. Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proc. of IJCAI*, 2011.
11. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC*, volume 5823 of *LNCS*. Springer, 2009.