LODStats – Large Scale Dataset Analytics for Linked Open Data

Ivan Ermilov, Jan Demter, Michael Martin, Jens Lehmann, Sören Auer

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

Abstract. In order to reuse, link, revise or query data made available on the Web, it is important to know the structure, size and coverage of it. To achieve this, we developed and evaluated LODStats – a statement-stream-based approach for gathering comprehensive statistics about data adhering to the RDF data model. LODStats is based on the declarative description of statistical dataset characteristics. Its main advantages over related approaches are a smaller memory footprint and significantly better performance and scalability. We integrated LODStats with CKAN and obtained a comprehensive picture of the current state of a significant part of the Data Web. This analysis is regularly published and enhanced over the past two years at the public platform stats.lod2.eu.

1 Introduction

For assessing the state of the Web of Data, 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 Web Data and the deployment of semantic technologies. In order to reuse, link, revise or query a dataset published on the Web, for example, 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. Data catalogs such as CKAN enable organizations to upload or link and describe datasources using comprehensive meta-data schemes. Similar to digital libraries, networks of such data catalogs can support the description, archiving and discovery of data on the Data Web. Recently, we have seen a rapid growth of data catalogs being made available on the Web. The data catalog registry *datacatalogs.org*, for example, already lists 325 data catalogs worldwide. Examples for the increasing popularity

[1] http://thedatahub.org

of data catalogs are Open Government Data portals (which often include data sources about energy networks, public transport, environment etc.), data portals of international organizations and NGOs, scientific data portals as well as master data catalogs in large enterprises.

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 (i.e. data items) 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 that the tradeoff between performance and accuracy can be controlled, in particular it can be adjusted to the available main memory size. Furthermore, LODStats is also easily extensible with novel analytical criteria. 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 analysis about datasets made available on the Web of Data facilitates a number of important use cases and provides crucial benefits. These include (UC = use case):

Vocabulary Reuse (UC1). One of the advantages of semantic technologies is to simplify data integration via common vocabularies. However, it is often difficult to identify relevant vocabulary elements. The LODStats web interface stores the usage frequency of vocabulary elements and provides search functionality. This allows knowledge engineers to find the most frequent schema elements, which can be used to model the task at hand. Having this functionality encourages reuse of schema elements and, therefore, simplifies data integration, which is one of the central advantages of semantic technologies. LODStats also provides a webservice for this functionality, such that third party tools can easily integrate search for similar classes and properties. For instance, the *Linked Open Vocabularies* (LOV)[2] project can utilize class usage frequency as another metric to filter the search results inside the Linked Open Vocabularies Catalogue.

Quality analysis (UC2). A major problem when using Web 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

[2] `http://lov.okfn.org`

restrictions, their values etc. Hence, a statistical analysis of datasets can provide important insights with regard to the expectable quality.

Coverage analysis (UC3). Similarly important as quality is the coverage a certain dataset provides. LODStats can be used to compute several coverage dimensions. For instance, the most frequent properties for a particular dataset can be computed and allow to get an overview over instance data, e.g. whether it contains address information. Furthermore, the frequency of namespaces may also be an indicator for the domain of a dataset. The ranges of properties can give insights on whether spatial or temporal information is present in the dataset. 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.

Privacy analysis (UC4). For quickly deciding whether a dataset potentially containing personal information can be published on the Data Web, we need to get a swift 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 (UC5). Establishing links between datasets is a fundamental requirement for many Linked Data applications (e.g. data integration and fusion). However, as we learned (cf. Section 4) the Web of Linked Data currently still lacks coherence (with less than 10% of the entities actually being linked). Meanwhile, there are a number of tools available which support the automatic generation of links (e.g. [9,10]). 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.

In this article, we present a *declarative representation of statistical dataset criteria*, which allows a straightforward extension and implementation of analytical dataset processors and additional criteria (Section 2). We demonstrate our LODStats *reference implementation* (Section 3), in particular we describe the architecture of the tool itself and the web platform stats.lod2.eu which builds on it. We also provide a *comprehensive analysis of the current state of the LOD datasets* available on the Web (Section 4) highlighting some crucial problems and obstacles. We also show that LODStats is 30-300% faster than two other tools for computing RDF statistics and allows to generate a statistic view on large parts of the Data Web in less than a day of processing time. The analysis

of hundreds of datasets, a more comprehensive discussion, an improved reference implementation, a public platform as well as the regular online publication and visualisation of statistics to enable the use cases outlined above are the core contributions compared to a previous short paper introducing LODStats [6].

2 Statistical Criteria

In this section we devise a definition for statistical dataset criteria, summarize the results of a comprehensive survey of 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 in a way that statistical analyses can be performed efficiently). The following definition formalizes our concept of a statistical criteria:

Definition (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 criterion. 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.[3] With the filter condition we can constrain the application of a criterion, 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, `M[ns(?subject)]++`, where M 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

[3] 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.

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 (Subject vocabularies criterion): This example illustrates the statistical criterion *subject vocabularies*, which collects a list of the 100 vocabularies most frequently 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:* Hashmap (initially empty)
- *Counter code:* `M[ns(?subject)]++` (`ns` is a function returning the namespace of the IRI given as parameter, i.e. the part of the IRI before the last occurrence of '/' or '#')
- *Post-processing filter:* `top(M,100)`

Statistical criteria can also be understood as a rule-based formalism. In this case, F is a condition (body of the rule) and D is 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.

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.F then
            execute C.D;
            if datastructures exceed threshold then
                purge datastructures;
forall the Criteria C ∈ CS do
    execute C.P and return results
```

2.2 Statistical Criteria Survey

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

- *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),
- *surveying and combining statistical criteria from related work* particularly from the Vocabulary of Interlinked Datasets (VoID [4]) and RDFStats [8],
- *expert interviews*, which we performed with representatives from the AKSW research group and the LOD2 project.

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[4].

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 cannot 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).

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

[4] `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[type(?o)]++`	–
21	languages	`isLiteral(?o)`	→ `M[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 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.

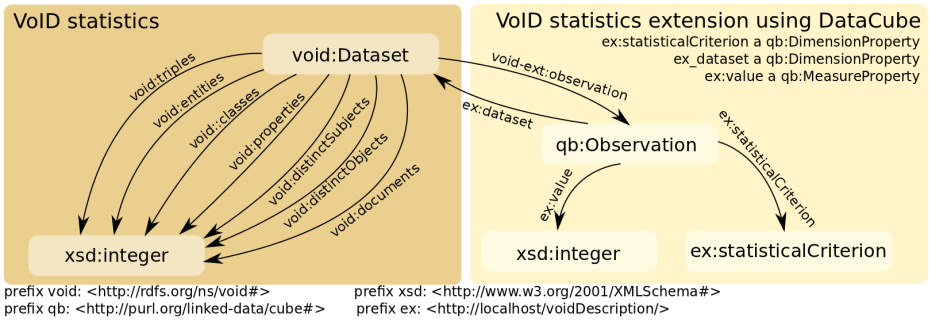

Fig. 1. Representing complex statistics using VoID and Data Cube observations.

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 [5]. 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. As depicted in Figure 1, we link a `void:Dataset` to a `qb:Observation` using the newly defined object property `void-ext:observation`, which is a simple extension to VoID.

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 usage*, *class usage distinct per subject*, *property usage*, *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

Architecture. Figure 2 shows an overview of the general architecture of LODStats. LODStats is written in Python and uses the Redland library [3] and its bindings to Python [2] 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* [7] 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 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.

Implementation. 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.

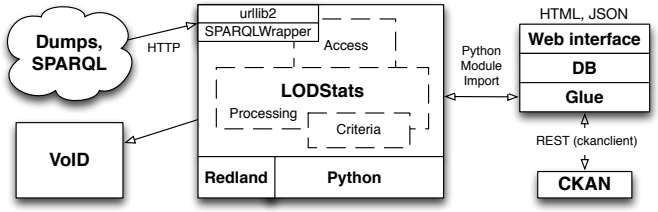

Fig. 2. Architecture of the LODStats reference implementation.

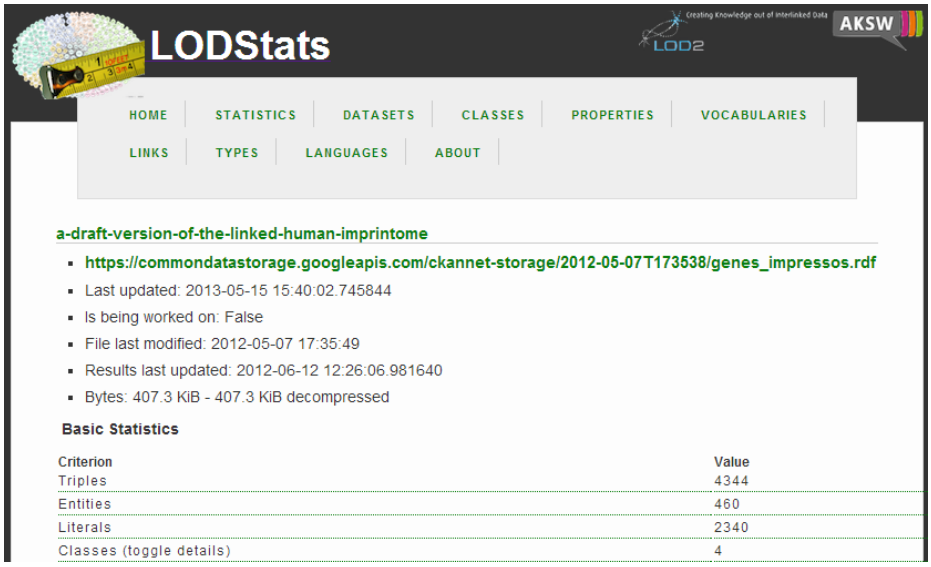

Fig. 3. LODStats interface at `http://stats.lod2.eu`.

The stats.lod2.eu Interface. We implemented a web interface[5] (cf. Figure 3), which provides continuously updated information about RDF datasets from the Data Hub (CKAN). Beyond the various statistics made, the interface allows to search for common classes or properties, helping to encourage vocabulary reuse. Class and property search functionality as well as the statistics are also available via a RESTful web service for integration with other projects. The interface supports the functionality listed below. In brackets, the use cases from the introdution, which are supported by the respective functionality are listed:

- general LOD cloud statistics,
- report of warnings and errors for each dataset (UC2),
- report on statistical criteria for each individual dataset (UC3, UC4),
- export as VoID and DataCube statistical metadata (UC3, UC4),
- dataset linkage explorer (UC2, UC5),
- search function for datasets, vocabularies, classes, properties, languages, datatypes (UC1),
- REST interface for the above search functions (UC1),
- Linked Data publication of statistics (technical support for UC1-UC5),
- SPARQL endpoint to query all extracted statistics (technical support UC1-UC5),
- CubeViz installation for facet-based browsing and visualisation of the statistical metadata (UC2-UC5).

The service permanently regularly crawls data portals such as TheDataHub.org for new and updated datasets as well as (re-)computes the respective statistics.

[5] `http://stats.lod2.eu`

	2011-12-09	2012-04-25	2013-03-01	2013-05-15
Datasets	452	506	699	2289
SPARQL-only	198	215	200	511
Triples	950M	1,174M	7,4B	11B
Dumps	235M	534M	1,3B	1.5B
SPARQL	714M	640M	6,1B	9.5B
Errors	248	276	366	592
Unreachable	45	38	121	374
SPARQL issues	139	153	150	121
Parse errors	57	66	94	91
Archive issues	3	2	1	6
Warnings	2334	5029	3801	5870

Table 2. Aggregated LODStats results at different points in time.

Criterion	Mean	Min	Max	Median
Triples per dataset	2,180,651	2	247,620,294	2,486.0
Entities	390,325.95	0	63,494,920	106.5
Literals	790,000.57	0	88,315,872	127.0
Blanks	484,540.68	0	202,745,495	0.0
as subject	399,680.75	0	166,901,812	0.0
as object	143,005.6	0	50,803,539	0.0
Subclasses	14.07	0	2,000	0.0
Typed subjects	109,790.35	0	25,848,850	22.0
Labeled subjects	28,652.13	0	11,588,129	0.0
Properties per entity	2.86	0	27.27	2.54
String length				
typed	9.5	0	1,854.0	0.0
untyped	38.24	0	2,688.0	20.0
Class hierarchy depth	1.63	0	6	0.0
Property hierarchy depth	1.04	0	5	0.0
Links	610,219.4	0	54,913,515	977.0
Vocabularies	6.35	1	23	6.0
Classes	20.09	1	1,328	5.0
Properties	30.36	1	885	20.0

Table 3. LOD Cloud statistics on 2013-05-17.

4 Evaluation with CKAN LOD Datasets

Results. We evaluated every dataset available via CKAN in one of the formats LODStats can handle (i.e. N-Triples, RDF/XML, Turtle, N-Quads, N3). We excluded datasets for which we know that they duplicate fractions of already analysed data such as the LOD cloud cache[6]. Results (see Table 2) show that a significant amount of datasets is solely available via SPARQL endpoints (22.3%), with 23.7% of those having errors. Problems most likely originate from either limitations set up on queries to mitigate abuse of endpoints that made gathering statistics infeasible or from not supporting necessary extensions to SPARQL for counting triples on the endpoint side. We aim to tackle this problem by directly integrating support for the generation of statistics inside the SPARQL endpoint (cf. Section 5).

[6] `http://lod.openlinksw.com/sparql`

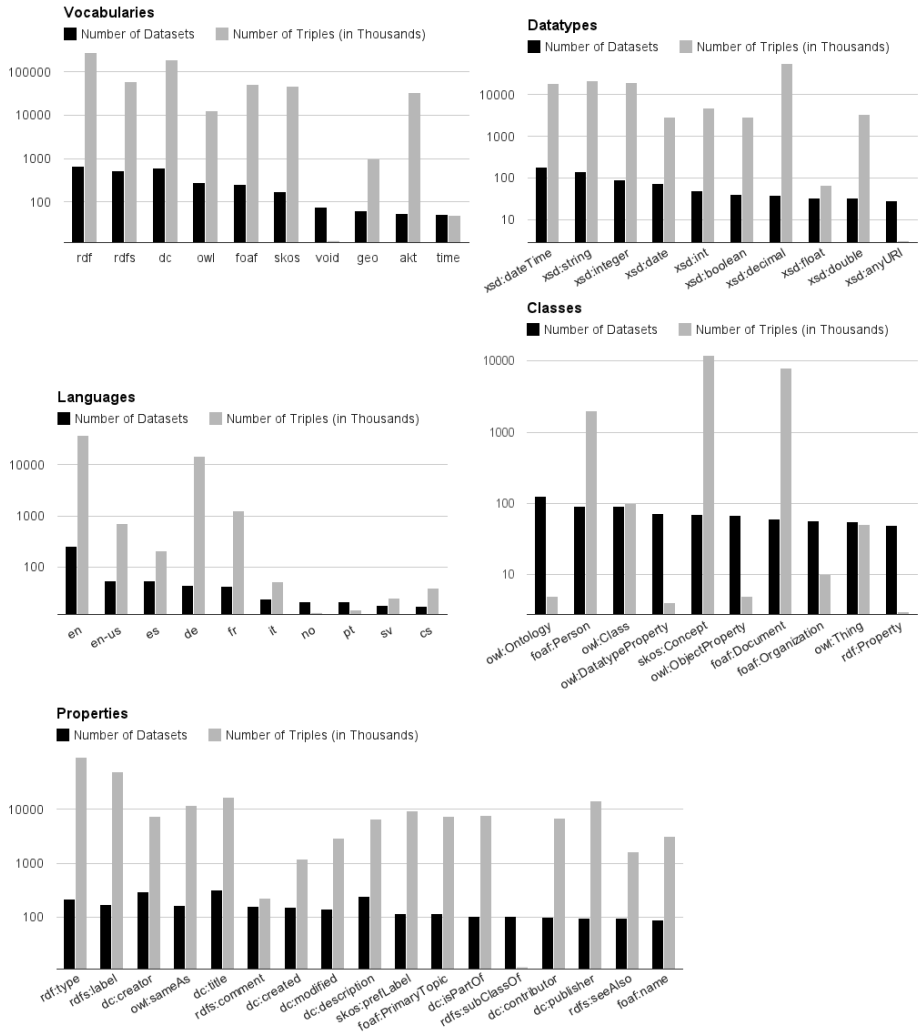

Fig. 4. Statistics about frequent vocabularies, datatypes, languages, classes and properties.

Common Errors. Of those datasets available as dumps in one of the aforementioned formats, 25.9% exhibited some sort of problem that prevented the generation of statistics. 16.3% were unreachable (i.e. unavailable, access denied, connection time out) at the advertised URL. Overall, 3.9% had fatal syntax errors that prevented Redland from successfully parsing the entire file. 1.2% of the datasets had errors during decompression/unarchiving. Syntax errors stem from errors in data provided by CKAN about the serialisation in some cases (e.g. the format given in CKAN is RDF/XML while it is in fact N-Triples). Guessing the

serialisation of resources is currently possible with LODStats by file extension. However, files often do not have an extension at all or one that is misleading. Testing a sample of datasets from CKAN showed that missing or false extensions are found more often than incorrect information about serialisation formats and it was thus chosen to generally trust CKAN on serialisation formats rather than guessing them. Implementing better strategies for automatically choosing the correct serialisation format are subject to future work. Errors with archives are almost exclusively due to datasets served with misleading extensions by the original publisher that result in the wrong extractor being used. Data dumps published in archives (i.e. zip/tar) often contained redundant files.

Problems with superfluous results when querying SPARQL endpoints can be avoided if such datasets registered in CKAN additionally provide the default graph URI, since sometimes multiple registry entries refer to the same endpoint URI. Also, Linked Data URIs should be tested automatically to enforce higher quality on the given information (e.g. is an example resource dereferenceable, SPARQL endpoint URL reachable etc.)

Serialisation Formats. 58.1% of the datasets are available as RDF/XML, 10.4% as Turtle, 5.5% as N-Triples, 1.7% as N3 and 1.0% as N-Quads. Datasets are often available in multiple serialisation formats via CKAN. If this is the case, our evaluation with LODStats prefers N-Triples, then in descending order, N-Quads, RDF/XML, Turtle and N3.

Observations. Table 3 summarizes some statistics generated by our approach over all LOD datasets which could be successfully processed. Overall, it should be noted that the absolute numbers we report here are significantly lower than those at `http://lod-cloud.net/state`. The statistics reported there partially rely on, possibly incorrect, user input, e.g. actual triple counts are often lower than reported in CKAN. In addition, due to the various errors discussed above, we cannot process all datasets, e.g. do not include offline data sources.

Figure 4 summarizes results for some of the statistical criteria computed by LODStats. *CubeViz*[7] allows to browse and visualize various diagrams for the extracted statistics represented in DataCube.

As expected, the W3C standard vocabularies like RDF, RDFS and OWL occur very frequently. In particular, they are present in a broad range of ontologies. Some other vocabularies like W3C Geo occur in fewer knowledge bases, but are overall very frequently to the large size of geospatial datasets. The void metadata descriptions on the opposite are used in many knowledge bases, but naturally only represent a small percentage of the size of the dataset.

Regarding datatypes, all the most frequent datatypes use the `xsd` namespace. Dates are widely used, following by string and numerical datatypes. It is worth noting that different datatypes for floating point numbers (`xsd:double`, `xsd:float`, `xsd:decimal`) as well as integers (`xsd:int`, `xsd:integer`) are widely used, although they serve a similar purpose. Similar datatypes, such as `xsd:short`

[7] `http://stats.lod2.eu/cubeviz`

and `xsd:long` are rarely used. The results are surprising, since arguably the `xsd:decimal` and `xsd:integer`, which do not require a specific bit length would appear to be a more natural semantic representation of information whereas bit lengths are rather technical details.

In terms of languages, English is dominant with German also being very frequent. In general, American and European languages currently seem to be disproportionally frequent compared to, e.g., Asian languages.

Finally, we also analysed class and property frequency. Naturally, standard classes defined by the W3C dominate here. In particular, `owl:Ontology` is used to add meta information to many knowledge bases. However, FOAF and SKOS also contain classes, which are very widely and frequently used.For properties, many elements of Dublin Core are also frequently used in addition to SKOS and FOAF. We noted that `owl:sameAs` is used in only 167 (9.4%) of the analysed dataset dumps with about 12.2 million triples overall. Such links are only a small fraction of all triples (0.8%) and suggest that the LOD cloud is not that heavily interlinked.

Performance. To evaluate the performance of our approach we selected datasets containing different domain data of various sizes and serializations: *Diseasome*, *Gemeenschappelijke Thesaurus Audiovisuele Archieven*, *Texai Lexicon*, *LinkedCT*, *Wikipedia3* and the *Library of Congress Name Authority File (NAF)*[8]. Performance comparisons (see Table 4 and Figure 5) show that LODStats performs significantly better than both make-void and RDFStats, especially for larger datasets. LODStats memory consumption is constant depending on the size of the thresholds set for purging the intermediate data structures. Make-void and RDFStats are not able to compute statistics for larger datasets. The speedup of LODStats compared to make-void lies between 26% (for the 1M GTTA triple in nt format) and 306% (for the 91k Diseasome triple). To compare the tools, we computed all the statistical criteria defined by VoID. Note that the statistics computed by make-void and RDFStats are a subset of those supported by LODStats.

5 Conclusions

With LODStats we developed an extensible and scalable approach for large-scale dataset analytics on the Web of Data. 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 50-80% 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

[8] `http://thedatahub.org/dataset/... fu-berlin-diseasome, gemeenschappelijke-thesaurus-audio-visuele-archieven, texai_lexicon, linkedct, wikipedia3, library-of-congress-name-authority-file`

Dataset	Format	Triples	make-void	RDFStats	LODStats	Speedup
Diseasome	nt	91k	46.9	35.4	11.6	≈ 4.06
GTAA	ttl	993k	212.4	201.4	142.1	≈ 1.49
GTAA	nt	993k	180.7	181.5	142.1	≈ 1.26
texai	rdf	9.2M	3,512.5	11,498.77	1,543.8	≈ 2.27
linkedct	nt	24.6M	out of mem.	out of mem.	3,915.5	
wikipedia3	rdf	47.1M	out of mem.	out of mem.	6,919.0	
NAF	nt	226.4M	out of mem.	out of mem.	36,422.3	

Table 4. Performance comparison (2.26 GHz Core 2 Duo P7550, 4GB RAM, 3GB Java mem heap size, timings in seconds).

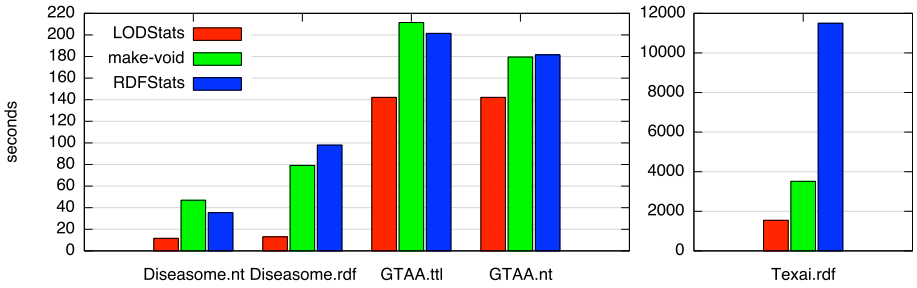

Fig. 5. Performance comparison: LODStats, make-void, RDFStats (left to right).

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. LODStats aimed for speeding up and reducing main and disk memory usage and cutting out intermediate steps like imports of datasets to triple stores.

Future Work. Currently, LODStats only collects statistics from one of the 325 dataset catalogs. Although, `http://datahub.io` contains a large amount of RDF resources, to acquire complete statistics on the Web of Data, a worldwide aggregation of these data catalogs should be used. For European open data *PublicData.eu* currently serves this purpose. However, at the time of writing no worldwide aggregation point exists. Consequently, we plan to extend LODStats in the future to be able to crawl and aggregate data from a large number of data catalogs. Another data catalog related improvement is the check for overlapping

CKAN entries, in particular several entries for the same dataset providing data in different formats.

A discoverability problem of LODStats is that the calculated statistics are not integrated back into the catalog itself. Thus, the statistics can not be easily accessed by users. We aim to make more statistics available between users interfaces by providing SPARQL interface to the complete statistical data generated by LODStats. This also enables dataset catalog integration.

While LODStats already delivers decent performance, a direct implementation of the approach in C/C++ utilizing Redland 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 for that purpose). 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 (e.g. countries).

Acknowledgment

This work was supported by grants from the EU's 7th Framework Programme provided for the projects LOD2 (GA no. 257943) and GeoKnow (GA no. 318159).

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. Beckett. Redland librdf language bindings. http://librdf.org/bindings/.
3. 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.
4. R. Cyganiak. Make-void. https://github.com/cygri/make-void, October 2010.
5. R. Cyganiak, D. Reynolds, and J. Tennison. The rdf data cube vocabulary, 2012. http://www.w3.org/TR/vocab-data-cube/.
6. Jan Demter, Sören Auer, Michael Martin, and Jens Lehmann. Lodstats – an extensible framework for high-performance dataset analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603. Springer, 2012. 29
7. I. Herman, S. Fernández, and C. Tejo. SPARQL endpoint interface to python. http://sparql-wrapper.sourceforge.net/.
8. A. Langegger and W. Wöß. Rdfstats - an extensible rdf statistics generator and library. In *DEXA Workshops*, pages 79–83. IEEE Computer Society, 2009.
9. Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of IJCAI*, 2011.
10. 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.