ARTICLE

Efficient Computation of Comprehensive Statistical information of Large OWL Datasets: A Scalable Approach

Heba Mohamed[a,b], Said Fathalla[a,b], Jens Lehmann[a,c], and Hajira Jabeen[d]

[a]Smart Data Analytics (SDA), University of Bonn, Bonn, Germany; [b]Faculty of Science, University of Alexandria, Alexandria, Egypt; [c]NetMedia Department, Fraunhofer IAIS, Dresden Lab, Germany; [d]GESIS-Leibniz Institute for the Social Sciences, Cologne, Germany

ABSTRACT

In recent years, there has been a growing interest in computing dataset statistics for exploring their internal structure. Since the size of the datasets has increased, this task became more challenging. Obtaining detailed statistical analyses of datasets facilitates a variety of instances of imperative use and offers key benefits, such as link target identification, vocabulary reuse, quality analysis, big data analytics, and coverage analysis. Several machine learning algorithms have been developed on top of the ontology axioms to predict, classify, and make sense of the data. Therefore, there is a pressing need to obtain a clear view of OWL datasets that have become more prevalent. In this paper, we present the first attempt of developing a distributed approach (OWLStats) for collecting comprehensive statistics over large-scale OWL datasets. OWLStats is a distributed in-memory approach for computing 50 statistical criteria for OWL datasets utilizing Apache Spark, with the ability to scale horizontally to a cluster of machines. We have successfully integrated OWLStats into the SANSA framework. Furthermore, two use cases are also presented. The results prove that OWLStats is linearly scalable in terms of both node and data scalability and can process various OWL datasets formats.

KEYWORDS

Distributed Processing; In-Memory Approach; SANSA Framework; Scalable Architecture; Semantic Web; Statistics Computations.

1. Introduction

Enterprise data management (EDM) is a process that encompasses effectively collecting, managing, and analysing enterprise data with the objective of producing useful information that supports decision-making as well as the subsequent data usage to improve business efficiency. The structure of enterprise data nowadays is far from being static, which made it more difficult for an enterprise to efficiently manage, understand, and use their data (Wood 2010). The management and the understanding of enterprise data has been improved by using Semantic Web technologies such as the Resource Description Framework (RDF), RDF Schema, and the Web Ontology Language (OWL) (Ma et al. 2009). Over the past decade, we have observed an increasing volume of semantic data, belonging to various domains available on the Web, resulting in large-scale semantic datasets (either in RDF or OWL format). These datasets are produced

CONTACT Mohamed: heba.ibrahim@alexu.edu.eg, Fathalla: $sm_fathalla@alex - sci.edu.eg, Lehmann : jens.lehmann@cs.uni - bonn.de, and Jabeen : hajira.jabeen@gesis.org$

based on the ontology to which they conform (Li and Sima 2015). Ontologies are particularly widespread in the life sciences, where several large biomedical ontologies have been developed, including the Biological Pathways Exchange (BioPAX) ontology[1], and the National Cancer Institute thesaurus[2], and enterprise data representation, including Zachman's Enterprise Ontology (Kappelman and Zachman 2013). Ontologies are being used in application areas like Software Engineering (Wongthongtham et al. 2017; Pileggi et al. 2018), Bioinformatics (Facchiano 2017), Data Integration (De Giacomo et al. 2018) and Enterprise Data Management (Rajabi et al. 2013).

It is of vital importance to collect comprehensive statistics on datasets illustrating their internal structure and external consistency to assess the efficiency of the individual datasets as well as to monitor the progress of Web data publishing and integration. Obtaining detailed statistical analyses of datasets facilitates a variety of instances of imperative use and offers key benefits. For example:

- *Link target identification:* To build a web of data, the linking between different datasets is of crucial importance for many Linked Data applications, such as ontology merging and fusion. Getting insights about the inner structure of a dataset (i.e., statistics about classes, properties, vocabularies, etc.), boosts the process of integrating and reusing datasets in semantics-based systems.
- *Vocabulary reuse:* Evaluating the vocabulary reuse is of significance since existing vocabularies constitute a significant prerequisite for an interoperable Web of Data. Hence, calculating the commonly used vocabularies simplifies dataset creation and integration.
- *Quality analysis:* Assessing and evaluating the quality expected, and determining whether it is sufficient for a particular application is highly important. It is crucial to analyze datasets concerning incoming and outgoing links, the used vocabularies, properties values, and their ranges, in order to create similar measures on the Web of Data.
- *Coverage analysis:* To ensure that the frequent dataset properties are used with similar entities. Furthermore, namespace frequency is an indicator of a dataset domain, i.e., the more namespaces belonging to a domain, the more relevant the dataset to that domain.
- *Big data analytics :* Process large-scale enterprise data, particularly when the data is distributed across different locations.

A variety of approaches have attempted to compute statistics about RDF datasets (Langegger and Woss 2009; Auer et al. 2012; Sejdiu et al. 2018). Although these approaches are interesting, they do not allow for computing statistics over large-scale OWL datasets. To the best of our knowledge, previous work has failed to address statistical computations for OWL datasets. Most studies have only tended to focus on triple structure analysis rather than the axiom structure of the datasets. OWLStats is the first attempt to develop a distributed approach for providing comprehensive statistical information about large-scale OWL datasets. To achieve scalability, we have implemented our approach using Apache Spark[3], a distributed in-memory computing framework. Spark is horizontally scalable and can run on multiple machine clusters, i.e.,

[1] http://www.biopax.org/
[2] https://ncit.nci.nih.gov
[3] https://spark.apache.org/

the workload is spread across multiple machine memories. Due to its efficiency in handling large-scale datasets and scalability, Apache Spark has recently gained considerable attention. The primary abstraction that Spark provides is the *Resilient Distributed Dataset* (RDD). Additional advantages of using RDDs are in-memory computation, fault tolerance, distributed partitioning, and persistence.

The ultimate goal of enterprise data management research is to increase the productivity of organizations by building scalable data architectures that allow to adapt quickly to changes and process very large amount of data, guarantee data consistency, conduct valuable data analysis, and produce effective insights. Therefore, in this article, we propose a novel approach (i.e., OWLStats) for comprehensive statistical computation of large-scale OWL datasets that extends the work presented in (Mohamed et al. 2020). We are providing 20 new axiom-based statistical criteria (Criteria 30–50 in Table 1) that have not been proposed before. Also, we conducted a new experiment in which we vary the number of cluster workers in order to measure the node scalability with more datasets with various sizes to measure the data scalability. In the proposed approach, we adopted 30 criteria for evaluating large-scale RDF datasets proposed in (Auer et al. 2012; Sejdiu et al. 2018) together with proposing 20 additional criteria. We compute the respective statistics in three main steps: 1) storing the OWL dataset into a scalable distributed storage, 2) converting the input dataset into the main data structure (i.e., `RDD[OWLAxiom]`), and 3) computing the statistical criteria. OWLStats is a generalised approach that allows computing statistics for any OWL dataset (i.e., Functional, Manchester, and OWL/XML syntaxes. To emphasize the usefulness of OWLStats, we successfully integrated it into SANSA framework (Lehmann et al. 2017)(see section 7). SANSA is an open source[4] large-scale processing engine for efficient processing of large-scale semantic dataset that empowers business-critical applications in many companies and enterprises around the globe, including BigDataEurope[5], Boost 4.0[6], and more.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of the related work. Section 3 describes the formal definitions of the statistical criterion and the RDD operation used. The proposed approach, its statistical criteria selection, architecture, implementation, and complexity analysis are described in Section 4. The experimental setup and the two different experiments are presented in Section 5. The discussion of the obtained results and three use cases are stated in Section 6. Three use cases are presented in Section 7. The availability and sustainability that the OWLStats fulfil are presented in section 8. Ultimately, we conclude and the potential extension of OWLStats is discussed in Section 9.

2. Related Work

Semantic web technologies have been widely used in the enterprise data management (da Silva Serapião Leal et al. 2020; Suga and Iijima 2018; Ali et al. 2019). The last few years have witnessed considerable growth in the Linked Open Data (LOD), which can be consumed by software agents. These agents can explore, stream, recommend, and organize information in intelligent ways to assist web users. Comprehensive statistics calculation on such datasets has become a vital factor in describing their internal

[4] https://github.com/SANSA-Stack
[5] https://www.big-data-europe.eu/
[6] https://boost40.eu/

structure and coverage. These statistics are increasingly important in many areas, e.g., involving data analysis (e.g., quality analysis and coverage analysis), query optimization, and data interlinking and reuse. In this section, we outline the work related to RDF datasets and OWL ontologies statistics computations. Previous work has primarily focused on calculating RDF statistics, ignoring to address OWL statistics, especially for large-scale OWL ontologies.

RDF datasets statistics computations. Few researchers have addressed the problem of calculating RDF statistics. Make-void[7] is a tool, written in Java, that computes statistics about RDF files, such as the number of triples, classes, and properties, by running several SPARQL queries using the ARQ query engine provided in the Apache Jena framework. It generates statistics in RDF format using VoID vocabulary in order to make it machine-readable. RDFStat (Langegger and Woss 2009) is another framework, based on the Jena framework, for calculating statistics from RDF sources, such as documents and SPARQL endpoints. It can generate statistical data, such as the number of instances, as well as histograms for a variety of various data types. Contrary to make-void, RDFStats can utilize SPARQL endpoints for querying RDF data and provides visualizations for its statistics. Furthermore, it does not use VoID (Vocabulary of Interlinked Datasets) for statistics description but rather describes them using SCOVO (Statistical Core Vocabulary (Hausenblas et al. 2012)).

LODStats (Auer et al. 2012) is an approach, written as a Python module and uses the Redland library (Beckett 2001), for computing 32 different statistical criteria, such as typed string length, max per property, and class hierarchy depth, the results are described using VoID. The main advantage of LODStats, when compared to current approaches, is its significantly better performance and scalability as well as low memory consumption. One of the limitations of LODStat is that it can operate only on a single triple pattern, i.e., it does not support, for example, star patterns (Gottron et al. 2013). Nevertheless, it provides several schema-level statistics, such as RDFS sub-hierarchy depth, and data-level statistics, such as counting triples with literals. LODStats has been integrated with the Comprehensive Knowledge Archive (CKAN)[8] dataset metadata registry to get a general overview of the current state of the Data on the Web.

Concerning distributed processing-based approaches, BÃűhm et al. (Böhm et al. 2011) have developed a scalable approach that automatically generates VoID descriptions for large corpora of Linked Data in a distributed manner. A distributed, in-memory approach for the computation of large RDF datasets statistics is DistLODStats (Sejdiu et al. 2018). DistLODStats extends LODStats by calculating the same statistical criteria but in a distributed and scalable manner. DistLODStats is implemented using the Apache Spark framework and is integrated into SANSA. One of the pitfalls of DistLODStats and LODStats is that it can only calculate statistics for RDF datasets. Despite these efforts, no one as far as we know has proposed an approach for computing statistics about OWL datasets. Therefore, we took the step towards implementing a scalable approach that can provide statistical information about large-scale OWL datasets (up to 55 millions axioms in the biomedical domain (Matentzoglu et al. 2013)). Actually, OWL datsets are widely used by semantics-based systems, such as Swoogle (Ding et al. 2004) and Watson (d'Aquin and Motta 2011). The proposed approach is the first attempt to compute comprehensive statistics over large-scale OWL datasets. Previous

[7] https://github.com/cygri/make-void
[8] http://thedatahub.org/

studies (Langegger and Woss 2009; Auer et al. 2012; Sejdiu et al. 2018) have focused on triple structure analysis rather than the axiom structure of the datasets. Consequently, the proposed approach cannot be compared with any of these approaches. Differently, we proposed 20 novel axiom-based criteria that are not yet proposed in any of the related work.

3. Definitions

The following definition formalizes the concept of statistical criteria (Auer et al. 2012):

Definition 1 (Statistical Criterion): *A statistical criterion C is a triple $C = (F, D, P)$, where: F is a SPARQL filter condition, D is a derived dataset from the input dataset (RDD of OWLAxioms[9]) after applying F, and P is a post-processing filter operating on the data structure D.*

In Definition 1, F serves as a filter operation to decide whether an axiom matches the condition of a specific criterion. The dataset is processed axiom by axiom, where each axiom examined is matched against each triple design of each criterion. D is the result RDD after applying F on the input OWL dataset. In most cases, the post-processing step is not required. However, P returns values from the derived dataset D. Post-processing operation performs further computational steps, such as retrieving the top-n elements of D. A formal representation for each statistical criterion is shown in Table 1.

For instance, consider *Criteria #29* in Table 1, this criterion, i.e., the average per property computes the entities with their average values in the dataset. Here, the *Criterion name* is "Average per property (int, float, time)", the *Description* is "Lists all entities and their average values in the dataset", the *Filter* is "Filter all the data property assertions and check if the object is literal and of type: *int*, or *float*, or *time*", the *Action* is "Calculate the number of objects connected with each property $m1$ and the number of properties $m2$", and the *Post-Processing* is $m1/m2$.

Definition 2 (RDD Operations): *All the statistical criteria implemented using the following operations: map, filter, reduceByKey, groupBy, and combineByKey. All RDD operations are documented in RDD Programming Guide[10]. A brief formalization for each operation is given as follows:*

- *map: The map function iterates over every line in RDD and converts it into a new RDD based on a specific function.*
- *flatMap: flatMap is similar to map operation, except that map return one element, while flatMap return a list of elements based on a specific function.*
- *filter: New RDD returned containing only the elements that match a certain condition.*
- *reduceByKey: The input RDD is a key-value (K, V) pairs, the pairs on the same machine with the same key are combined before the data is shuffled.*
- *groupBy: The input RDD is a key-value (K, V) pairs, returns a new RDD of (K, Iterable<V>) pairs after applying a grouping function (e.g., average, count) over the*

[9] http://owlcs.github.io/owlapi/apidocs_5/org/semanticweb/owlapi/model/OWLAxiom.html
[10] https://spark.apache.org/docs/latest/rdd-programming-guide.html

input RDD.

4. Approach

OWLStats adopted 30 criteria proposed in (Auer et al. 2012; Sejdiu et al. 2018) and added 20 more criteria. In contrast to (Sejdiu et al. 2018), we perform the statistical computation on the axiom structure of the OWL datasets. We carried out the computation in a Spark distributed environment using RDDs (See Definition 2). The proposed approach involves the conversion of the input OWL dataset to RDDs of OWLAxioms.

4.1. *Statistical Criteria Selection*

To obtain a comprehensive set of statistical criteria, we generated our criteria from:

(1) collecting and merging statistical criteria from related work, including LODStats (Auer et al. 2012) and DistLODStats (Sejdiu et al. 2018),

(2) analyzing OWL data model elements, i.e., potential elements as `owl:Class`, `owl:ObjectProperty`, and `owl:DatatypeProperties`, literals (including datatypes and language tags), class descriptions (including intersection, union and complement), and value constrains (including *allValuesFrom* and *someValuesFrom*) ... etc, and

(3) interviewing expert from the SDA research group and other related organizations.

We give a detailed description about rule syntax of our statistical criteria in Table 1. We split the 50 statistical criteria we collected into two groups; 1) data-level and 2) schema-level criteria.

Data-Level criteria: OWLStats collects statistical information on data items, such as the list of datatypes used for literals. If the dataset contains a string or untyped literals, then the dataset's overall average string length is calculated. Furthermore, the frequencies of links between entities of different namespaces, number of all axioms, labeled subjects, typed subjects, and the axioms which define the same or different individuals are computed as well. OWLStats can generate a list of various datatypes used for literals, such as string, integer, date, ... etc.

Schema-Level criteria: we collect detailed statistics on the schema elements, such as classes, subclasses, and properties (both declared and used in the dataset). Moreover, OWLStats can analyze more complicated schema-level properties, such as property restrictions (i.e., value and cardinality constraints), and class descriptions (i.e., intersection, union, and complement).

4.2. *OWLStats Architecture*

Figure 1 illustrates the main workflow of the statistical computation proposed by OWLStats. OWLStats approach consists of three main steps: 1) Input: Storing the OWL dataset into a scalable distributed storage, 2) Processing: Converting the input dataset into the main data structure (i.e. `RDD[OWLAxiom]`), and computing the statistical criteria, and 3) Output: generating the results.

Table 1.: Statistical criterion defined by Spark rules.

No.	Criterion	Rule Filter	→ Rule Action	Postproc.
1	used classes	A=Class_Assertion	→ map(_.getClassExpression)	-
2	class usage count	A=Class_Assertion	→.map(a =>(a.getClassExpression,1)) .reduceByKey(_+_)	take(100)
3	classes defined	A=Declaration.isOWLClass	→ map (_.getEntity.getIRI)	-
4	class hierarchy depth	A=SubClass_Of	G +=(a.getSubClass, a.getSuperClass)	depth(G)
5	data property usage	A=Data_Property_Assertion	→map (a => (a.getProperty, 1)) .reduceByKey(_+_)	take(100)
6	object property usage	A=Object_Property_Assertion	→map (a => (a.getProperty, 1)) .reduceByKey(_+_)	take(100)
7	property usage distinct per subj.		→ groupBy(_.getSubject) .reduceByKey(_+_)	count()
8	property usage distinct per obj.		→ groupBy(_.getObject) .reduceByKey(_+_)	count()
9	properties distinct per subj.		→ groupBy(_.getSubject) .combineByKey(_+_)	sum/count
10	properties distinct per obj.		→ groupBy(_.getObject) .combineByKey(_+_)	sum/count
11	outdegree		→map(_.getSubject) .map(a => (a, 1)) .combineByKey(_+_)	sum/count
12	indegree		→map(_.getObject) .map(a => (a, 1)) .combineByKey(_+_)	sum/count
13	data Property hierarchy depth	A=Sub_Data_Property_Of	G += (a.getSubProperty, a.getSuperProperty)	depth(G)
14	object Property hierarchy depth	A=Sub_Object_Property_Of	G += (a.getSubProperty, a.getSuperProperty)	depth(G)
15	subclass usage	A=SubClass_Of	→count()	-
16	axioms		→count()	-
17	entities mentioned		→map(a=>(a.getSubject,a.getObject) .count()	-
18	distinct entities		→map(a=>(a.getSubject,a.getObject) .distinct()	-
19	literals	A=Data_Proeprty_Assertion && obj.isLiteral	→ count()	-
20	datatypes	A=Data_Proeprty_Assertion && obj.isLiteral	→ map (a => (o.getDatatype, 1)) .reduceByKey(_+_)	-
21	languages	A=Data_Proeprty_Assertion && obj.isLiteral	→map (a => (o.getLang, 1)) .reduceByKey(_+_)	-
22	average typed string length	A=Data_Proeprty_Assertion && obj.isLiteral && obj.getDatatype = XSD_STRING	→count() len+=o.length	len/count
23	average untyped string length	A=Data_Proeprty_Assertion && obj.isLiteral && !obj.getDatatype.isEmpty()	→count() len+=o.length	len/count
24	typed subject	A=Axiom_TYPES && p=RDF_TYPE	→count()	-
25	labeled subject	A=Annotation_Assertion && a.getProperty.isLabel	→count()	-
26	sameAs	A=SAME_INDIVIDUAL	→count()	-
27	namespace links	A=Axiom_TYPES && s.getNS_ != o.getNS	→map (a => ((s.getNS,o.getNS),1)) .reduceByKey(_+_)	-
28	max per property {int,float,time}	A=Data_Proeprty_Assertion && o.isLiteral&& o.getDatatype = isInt | isFloat | isDateTime	→map (_.getProperty, _.getObject) .reduceByKey(_ max _)	-
29	avg per property {int,float,time}	A=Data_Proeprty_Assertion && o.isLiteral&& o.getDatatype = isInt | isFloat | isDateTime	→m1 => map(_.getObject).count() m2 => map(_.getProperty).count()	m1/m2
30	subj. vocabularies		→map (a => (a.getsubject.getNS,1)) .reduceByKey(_+_)	-
31	pred. vocabularies		→map (a =>(a.getProperty.getNS,1)) .reduceByKey(_+_)	-
32	obj. vocabularies		→map (a => (a.getObject.getNS,1)) .reduceByKey(_+_)	-
33	subdata property usage	A=Sub_Data_Property	→count()	-
34	subobject property usage	A=Sub_Object_Property	→count()	-
35	subannotation property usage	A=Sub_Annotation_Property	→count()	-
36	classes count		→flatMap(a => a.classesInSignature) .count()	-

No.	Criterion	Rule Filter	→ Rule Action	Postproc.
37	data properties count		→flatMap(a => a.dataProperties InSignature).count()	-
38	object properties count		→flatMap(a => a.objectProperties InSignature).count()	-
39	class assertion count	A=Class_Assertion	→count()	-
40	data property assertion count	A=Data_Property_Assertion	→count()	-
41	object property assertion count	A=Object_Property_Assertion	→count()	-
42	annotation property assertion count	A=Annotation_Property_Assertion	→count()	-
43	different individuals	A=Different_Individuals	→count()	-
44	object intersection	A=Equivalent_Classes && Type=Object_Intersection_Of	→map(_.getOperandsAsList.get(1))	count()
45	object union	A=Equivalent_Classes && Type=Object_Union_Of	→map(_.getOperandsAsList.get(1))	count()
46	object complement	A=Equivalent_Classes && Type=Object_Complement_Of	→map(_.getOperandsAsList.get(1))	count()
47	data SomeValuesFrom	A=Equivalent_Classes && Type=Data_Some_Values_From	→map(_.getOperandsAsList.get(1))	count()
48	data AllValuesFrom	A=Equivalent_Classes && Type=Data_All_Values_From	→map(_.getOperandsAsList.get(1))	count()
49	data Cardinality	A=Equivalent_Classes && Type=Data_Min_Cardinality || Type=Data_Max_Cardinality	→map(_.getOperandsAsList.get(1))	count()
50	data HasValue	A=Equivalent_Classes && Type=Data_Has_Value	→map(_.getOperandsAsList.get(1))	count()

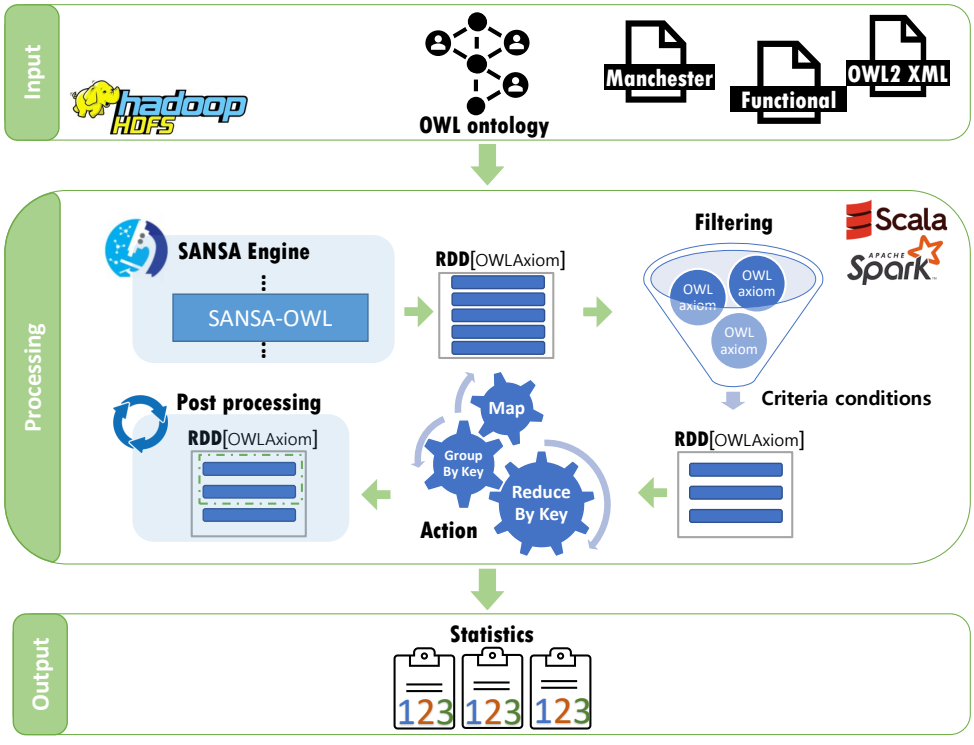

Figure 1.: OWLStats Architecture.

Step 1 (Input): To read the OWL dataset efficiently, Spark needs the dataset to be stored in a large-scale storage system. The Hadoop Distributed File-System (HDFS) (Shvachko et al. 2010) is used for data storage. HDFS is designed to store and stream large datasets for user applications efficiently. HDFS splits the data into separate blocks when the data is loaded into HDFS, then it replicates and distributes the blocks to various nodes in a cluster, allowing highly efficient parallel processing and fault-tolerance. Consequently, the node information that crash can be found in a cluster elsewhere.

Step 2 (Processing): To convert the input OWL dataset, we used SANSA-OWL[11] layer for the conversion. SANSA-OWL layer supports the conversion for three input formats: *Functional, Manchester, and OWL/XML*. SANSA-Stack influences current big data frameworks such as Apache Spark and Apache Flink (Ermilov et al. 2017). Apache Spark is a powerful analytics engine for large-scale data analysis. SANSA utilizes Scala programming language to provide the distributed implementations of the proposed algorithms. For each criterion, we start an execution plan to filter the input dataset and calculate the output. It is intended as a bridge between RDF data publishers and consumers, with applications ranging from data exploration to data cataloging and dataset archiving.

Step 3 (Output): A sample of the output results for LUBM-5, shown in Listing 1. Lines 1-2 in Listing 1 show the outcomes of computing the *Classes Defined* and *Object Property Usage* criteria in LUBM-5, which are 43 and 12, respectively. While, line 4 illustrates the number of axioms containing the object property `takesCourse`, which are 134051 axioms. Spark transformations, i.e., *map, filter, groupBy, reduceByKey*, perform the calculations. Computing phase output would be the statistical results expressed in a human-readable format, e.g., VoID. VoID[12] is an RDF Schema vocabulary to represent metadata about RDF datasets.

Listing 1: A snippet of sample output of the statistics generated for LUBM-5 dataset

```
1  void:classes 43;
2  void:objectProperties 12;
3  void:propertyPartition
4  [ void:objectProperty <#takesCourse>; void:axioms 134051; ],
5  [ void:objectProperty <#publicationAuthor>; void:axioms 67702; ],
6  [ void:objectProperty <#memberOf>; void:axioms 48582; ],
7  [ void:objectProperty <#advisor>; void:axioms 19371; ],
8  [ void:objectProperty <#undergraduateDegreeFrom>; void:axioms 15273; ],
9  [ void:objectProperty <#teacherOf>; void:axioms 10095; ],
10 [ void:objectProperty <#mastersDegreeFrom>; void:axioms 3373; ],
11 [ void:objectProperty <#doctoralDegreeFrom>; void:axioms 3373; ],
```

4.3. *Implementation*

This section explains the implementation of OWLStats framework. All phases of the OWLStats have been implemented using Apache Spark. Scala[13] programming language API has been used to provide a distributed implementation of the proposed approach. Algorithm 1 establishes the primary dataset from an OWL file (as constructed in line

[11] https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-owl
[12] https://www.w3.org/TR/void/
[13] https://www.scala-lang.org/

2). The algorithm takes as input: the OWL dataset, the syntax of the OWL dataset (we support *Functional*, *Manchester* and *OWL/XML* syntax), and the list of the statistical criteria.

Algorithm 1: OWLStats computation over a set of statistical criteria

```
   Input: spark: Spark Session,
          input: The OWL dataset,
          syntax: Syntax type (func, manch, owlxml),
          CL: List of criterion
 1 begin
 2 |  RDD axioms = input.convert(spark, syntax)
 3 |  axioms.cache()
 4 |  foreach c ∈ CL do
 5 |  |  rdd ← c.filter(axioms)
 6 |  |  rdd.cache()
 7 |  |  rdd ← c.action(rdd)
 8 |  |  if c.hasPostProc then
 9 |  |  |  rdd ← c.postProc(rdd)
10 |  end
11 end
```

Line 2 converts the input OWL file into `RDD[OWLAxiom]`. Afterwards, for each criterion defined inside OWLStats, the algorithm computes them using the *filter*, *action*, and *post-processing* operations (lines 5, 7, and 9). Every transformed RDD can be recalculated by default each time running an action on it. Nevertheless, an RDD could be persisted in memory for quicker access when needed instead of reconstructing the RDD. Spark caching techniques, persist or cache actions, can be used for faster access to RDD elements. In the OWLStats algorithm, caching is used twice to persist RDD elements in memory. In line 3, the OWLAxioms RDD to be used for each criterion is cached. Afterwards, caching derived RDD after applying the criterion filter condition on the input dataset (line 6).

Algorithm 2 describes the filter, action, and post-processing operations for *Criterion 2* listed in Table 1. Function *filter()* extracts all the `Class_Assertion` axioms from the input OWLAxioms RDD and then convert it to `RDD[OWLClassAssertionAxioms]` instead of `RDD[OWLAxiom]` (lines 3-4), then get all the class expressions within the `OWLClassAssertionAxioms` (line 5). Afterwards, *action()* function is applied to calculate the frequency of each `OWLClassExpression` (lines 8-9). Finally, *postProc()* function is called to sort the calculated *action()* and return the first 100 `OWLClassExpression` with their corresponding frequencies (line 12).

4.4. *Complexity Analysis*

Communications between nodes at large-scale data processing greatly affects the performance of enterprise systems (Xhafa 2021; Mishra et al. 2021). In this section, we analyze the performance of our approach in criteria computing which is primarily affected by the performance of two main tasks:

- **Data Scanning:** All operations that involve assessing the condition on an RDD are considered scan based operations. The data is only scanned once for all criteria

Algorithm 2: Classes usage count criterion

```
Input  : axioms: rdd of the converted OWL dataset
Output: result: first 100 Classes and there usage frequency
begin
    Function filter():RDD[OWLClassExpression]
        val f = extractAxioms(axioms, AxiomType.CLASS_ASSERTION)
                .asInstanceOf[RDD[OWLClassAssertionAxiom]]
                .map(_.getClassExpression)
        return f
    Function action():RDD[(OWLClassExpression, Int)]
        val a = filter().map(e => (e, 1))
                        .reduceByKey(_ + _)
        return a
    Function postProc():RDD[(OWLClassExpression, Int)]
        val p = action().sortBy(_._2, false).take(100)
        return p
end
```

when using the criterion filter along the same data. However, whenever data changes its status, for example, when removing the duplicate elements from the RDD, a new scan of the new status is required. Eventually, if the data is moved between cluster nodes (i.e., data is shuffled), a new scan is required.

- **Data Shuffling and Filtering:** Data shuffling means moving data around the network, i.e., between cluster nodes. If there is data movement required during the distributed processing, then the calculation might place more processing overhead which will significantly affect the performance. For example, to do a distributed `groupBy` or `groupByKey` operations, Spark typically has to move the data between nodes and gather them with `groupBy` key. Filters are another aspect that affects the performance of criteria calculations. It is preferable to filter the data in the early phases so that the subsequent phases take less processing.

Table 2 summarizes the complexity analysis of the computation of each criterion, which depends on the performance of the aforementioned tasks. We consider the complexity to be linear since just mostly one scan is necessary. However, in other cases, (such as data sorting in criterion 2, 5, and 6), the complexity will rise when there are iterative executions.

4.5. *Benchmark*

Lehigh University (LUBM) (Guo et al. 2005) synthetic benchmark has been used for the experiment. For the evaluation of Semantic Web repositories, LUBM is a commonly used benchmark for evaluating the efficiency of such repositories regarding extensional queries over a large dataset. LUBM benchmark[14] comprises four components: 1) the domain ontology (i.e., Univ-Bench), 2) the data generator (UBA) which generates the OWL or DAML+OIL data over the domain ontology. 3) Currently, the benchmark supports 14 test queries, and 4) The test module (UBT). LUBM generator generates many A-Box axioms but no T-Box axioms. We use the LUBM data generator in our experiment to generate five datasets of different sizes: LUBM-50, LUBM-200, LUBM-500, LUBM-1000, and LUBM-2000. The numbers attached to the benchmark name

[14] http://swat.cse.lehigh.edu/projects/lubm/

Table 2.: Complexity analysis breakdown by statistical criterion (n is the number of OWLAxioms).

Criterion	**Complexity**	**Shuffling**	**Performance Analysis**
1, 3	$O(n)$	No	Data is locally filtered and returned.
2, 5, 6	depending on the sorting algorithm applied.	Yes	Sorting in post-processing entails the movement of data. As classes are originally distributed over the cluster, computing their counts involves shuffling and reducing the data.
7, 8, 9, 10	$O(n)$	Yes	Before invoking the `groupBy` operation, the data needs to be shuffled then it is reduced by subject or object.
4, 13, 14	$O(n^3)$	No	Data is locally filtered to extract the parent and child relations and returned, so no data movement is needed.
11, 12, 20, 21	$O(n)$	Yes	Data is mapped to `(subject, 1)` pairs and then reduced by subject counting the 1s following the map-reduce technique, therefore data needs to be shuffled before invoking the `groupBy` operation.
15, 16, 17, 33, 34, 35	$O(n)$	No	No data transmission is required since the count is simultaneously computed locally in each node and then aggregated together for the cluster.
18	$O(n)$	No	It requires no data transmission since it returns the OWL Axioms after verifying `isNamed` condition.
19, 24, 25, 26, 39, 40,41, 42, 43	$O(n)$	No	No data transmission is required since data is locally filtered and counted in parallel and the separate counts are aggregated together.
22, 23	$O(n)$	No	Map the objects to their own length, then calculate the average of the object strings. The overall average is calculated by collecting single values from each node, which requires no data shuffling.
27	$O(n)$	Yes	Data is mapped to `((subject, object), 1)` pairs and then reduced by `(subject, object)` keys and count the 1s following the map-reduce technique. Therefore, data movement is needed.
28	$O(n)$	Yes	Data is moved in the cluster to get the maximum value per property, i.e., data is reduced to get the maximum.
29	$O(n)$	Yes	Data is reduced by property, then moved across the cluster to calculate the average.
30, 31, 32	$O(n)$	Yes	Data is mapped to `(target, 1)` pairs; (target is *subject*, or *predicate*, or *object*) and then reduced by `target`, and count the 1s following the map-reduce technique. Therefore, data movement is needed.
36, 37, 38	$O(n)$	No	No data transmission is needed, since the data is flat mapped first to get the desired axiom type then it locally filtered. Separate counts are calculated locally and summed up to calculate the overall count.
44, 45, 46, 47, 48, 49, 50	$O(n)$	No	Data is locally filtered to extract the specific `ClassExpressionType`. The counting in the post-processing step requires no data transmission because it is also computed locally in each cluster node before calculating the overall count.

Table 3.: LUBM benchmark datasets (functional syntax)

Dataset	Size (GB)	Load Time (m)	#Axioms
LUBM-10	0.3	1	1,316,517
LUBM-50	1.3	8	6,654,756
LUBM-150	3.8	15	26,782,455
LUBM-200	4.8	23	31,178,382
LUBM-500	10.5	82	77,577,078
LUBM-1000	20.3	126	151,066,104
LUBM-2000	41.2	230	306,002,524

are the number of generated universities. Properties of the generated datasets, loading time to the HDFS, and the number of axioms of each dataset are listed in Table 3. To create larger datasets from the ontology files created from the LUBM benchmark, we implemented a merge tool. In addition, we use LUBM datasets with various formats (such as OWL/XML and Manchester). The results reported are in functional syntax format.

5. Experiments

In this section, we describe the evaluation of OWLStats. We are aiming at answering the following questions concerning scalability and flexibility:

Q1) How efficient can OWLStats compute the introduced statistics?

Q2) How is the speedup ratio affected with respect to the number of worker nodes?

Q3) How does OWLStats process different datasets with various sizes?

Q4) How does OWLStats scale to larger datasets?

We start with the experimental setup, afterwards we give a brief description of the introduced experiments.

5.1. *Experimental Setup*

System configuration. All distributed experiments ran on a cluster with five nodes. Among these nodes, one is reserved to act as the master, and four nodes are used as computing workers. Each node has AMD Opteron 2.3 GHz processors (64 Cores), 250.9 GB memory, and the configured capacity is 1.7 TB. The nodes are connected with 1 Gb/s Ethernet. Moreover, Spark v2.4.4 and Hadoop v2.8.1 with Java 1.8.0 is installed on this cluster. Local-mode experiments are all carried out on a single cluster instance. All distributed experiments run three times, and the results indicate the average execution time.

5.2. *Scalability Experiments*

We evaluate our approach using the aforementioned datasets to study its performance as well as scalability. In this evaluation, we measure the scalability of OWLStats based

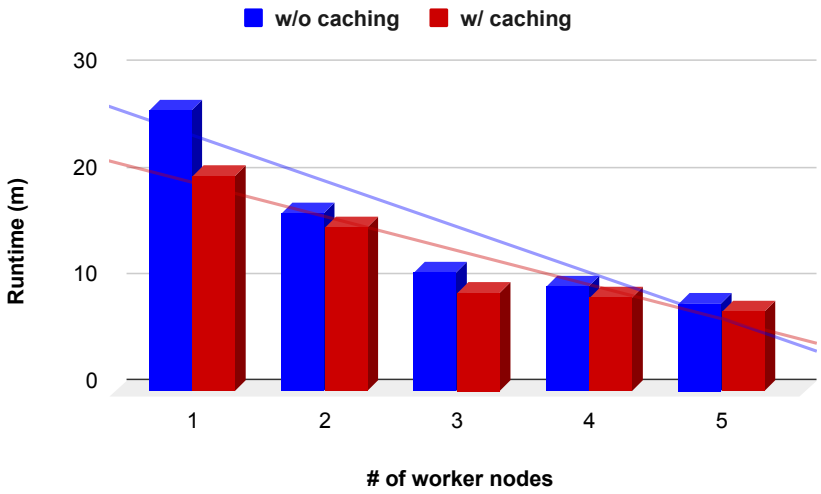

Figure 2.: OWLStats node scalability performance evaluation

on node and data scalability.

Experiment 1 (Node Scalability). This experiment was designed to measure the performance evaluation in terms of the number of nodes, which proves that the runtime decreases when the number of nodes increases. This experiment is aiming at answering the research question Q1. In this experiment, we increase the number of cluster workers to measure the node scalability. We have increased the number of workers from one to four (with one step each time) on the same dataset (i.e., LUBM-200).

Experiment 2 (Data Scalability). This experiment was designed to assess whether the OWLStats can handle large-scale datasets. This experiment is aiming at answering the research questions Q2, Q3, and Q4. In this experiment, we measure the efficiency of OWLStats by increasing the size of the input dataset. We retain a constant number of nodes (workers) at five in the cluster and increase the size of datasets. To test the data scalability of OWLStats, we run the experiments on the LUBM benchmark with five different sizes. We begin by generating a dataset of 50 universities (LUBM-50), then we iteratively increase the number of universities (i.e., scaling up the size).

6. Results and Discussion

Figure 2 and Figure 3 report the results of efficiency analysis for node and data scalability, respectively.

Experiment 1 Results: Figure 2 shows LUBM-200's speedup efficiency by raising the number of worker nodes from one to five. The execution time decreased around three times (from 26.3 min to 8.2 min). It is evident that as the number of workers increases, the execution time decreases linearly. The speedup ratio (S) is an essential metric for calculating the performance of parallel algorithms (addressing Q1). The speedup ratio is the ratio $S = T_L/T_N$, where T_L is the execution time of the algorithm in local mode, and T_N is the time on N workers. Efficiency E measures the speedup per worker. It is the time taken to run the algorithm in local mode against the time

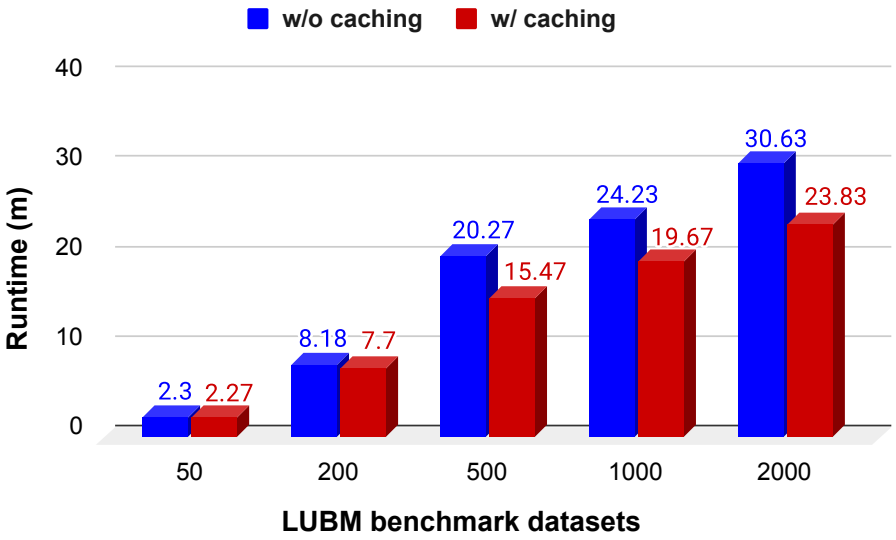

Figure 3.: OWLStats sizeup performance evaluation

on N workers, $E = S/N$. Figure 5 illustrates the speedup and efficiency ratios for the LUBM-200 dataset. The speedup for the selected data is increased sequentially with respect to the number of workers. In conclusion, the findings illustrate that OWLStats would achieve near-linear scalability of output in the sense of speedup.

Experiment 2 Results: Figure 3 displays the run time of the proposed distributed algorithm with and without caching for each dataset.

Caching is a mechanism to speed up applications that have multiple access to the same RDD, which keeps the data in memory and accelerates the computations. It is apparent from Figure 3, the reduction in execution time between OWLStats with caching (blue columns) and without caching (red columns). The x-axis represents the LUBM datasets produced with an increase in the number of universities, while the y-axis represents the execution time within minutes. For example, calculating the 50 statistics criteria with LUBM-2000 costs around 31 minutes without using the caching mechanism, while the time after caching was triggered, decreased to 24 minutes. Spark has the performance advantage of using in-memory data storage. The use of data storage in memory contributes to a decrease in the average time spent on network communication and data read/write using disk-based approaches. It is evident that the execution time increases linearly as the size of the dataset increases. The results show that our algorithm can be scaled according to the dataset size, which answers *Q3* and *Q4*.

Figure 4 shows the output obtained from running OWLStats in a multi-machine (five machines) cluster environment. For more illustrations, consider the LUBM-1000 dataset; the execution time decreased from 48.83 minutes in a single machine environment (local) down to 19.67 minutes in multiple machines environment (cluster). The observed time decrease can be interpreted as a result of multiple machine distribution of the computation. For LUBM-1000, the use of cluster mode speeds up the performance by two times, which addresses *Q2*.

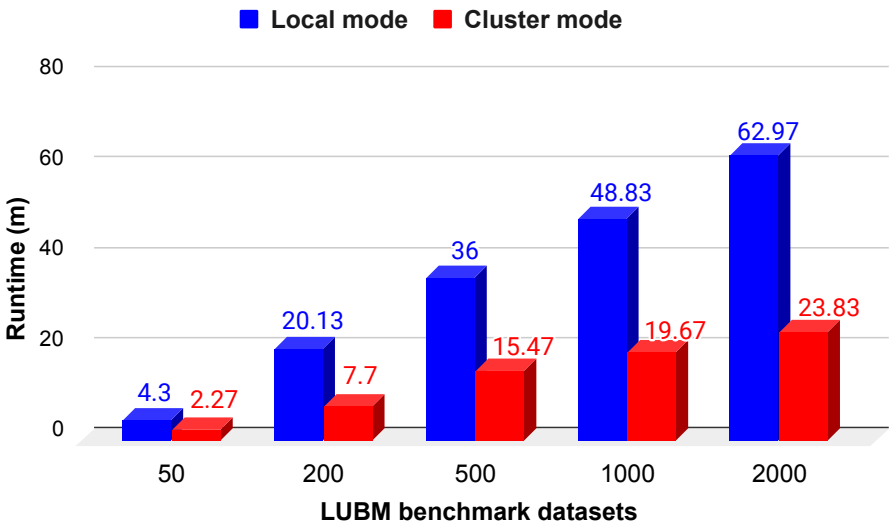

Figure 4.: OWLStats speedup performance evaluation in cluster and local environments.

Criteria Execution.

The overall execution time of OWLStats per each criterion is illustrated in Figure 6. The runtime for each criterion is reported for both LUBM-50 and LUBM-200 datasets. OWLStats consists of 50 statistical criteria; each criterion execution time depends on the number of the input OWLAxiom. The findings obtained from both datasets' execution show that the runtime is less when there is no data shuffle inside the cluster. Criteria 25, requires less execution time since it relies on `OWLAnnotationAssertions` of type `owl:literal` and the number of annotation assertions in LUBM benchmark not too much. The longest execution time is taken by Criteria 17. The reason refers to that even though it requires no data shuffling, but no filter is applied to the input data; therefore, the whole dataset is processed to evaluate the criteria. LUBM benchmark generates many A-box axioms (i.e., assertions) but no T-Box axioms; thus, criteria from 33 to 43 consume a long execution time because they are all evaluated on assertion axioms. Criteria 30, 31, and 32 concerning the vocabularies used in the dataset are considered efficient since there is no data movement among the cluster nodes. Criteria 44 to 50 were not evaluated because the LUBM benchmark contains only six `OWLEquivalentClasses` such that those consume almost no time. Overall, the experiments led us to conclude that OWLStats can complete statistical computation execution in a reasonable time, proving that OWLStats distributed statistical criteria computation is scalable.

To evaluate OWLStats over more complex ontologies (i.e., ontologies with more T-Box axioms), we use it over Gene Ontology (GO)[15]. GO is considered the leading source of gene information. It consists of 601,235 axioms. OWLStats computes all criteria within 326 seconds on a cluster of five worker nodes. Furthermore, we evaluate OWLStats over the University Ontology Benchmark (UOBM)[16], which generates more

[15] http://geneontology.org/
[16] https://www.cs.ox.ac.uk/isg/tools/UOBMGenerator/

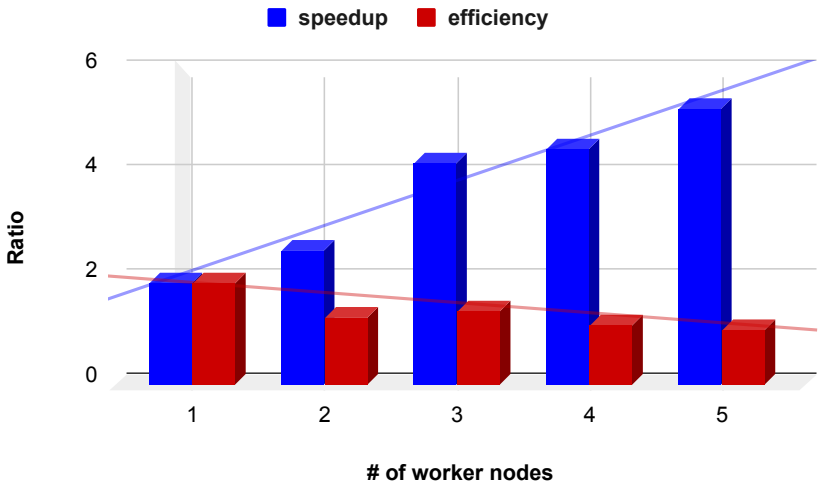

Figure 5.: OWLStats efficiency and speedup ratio.

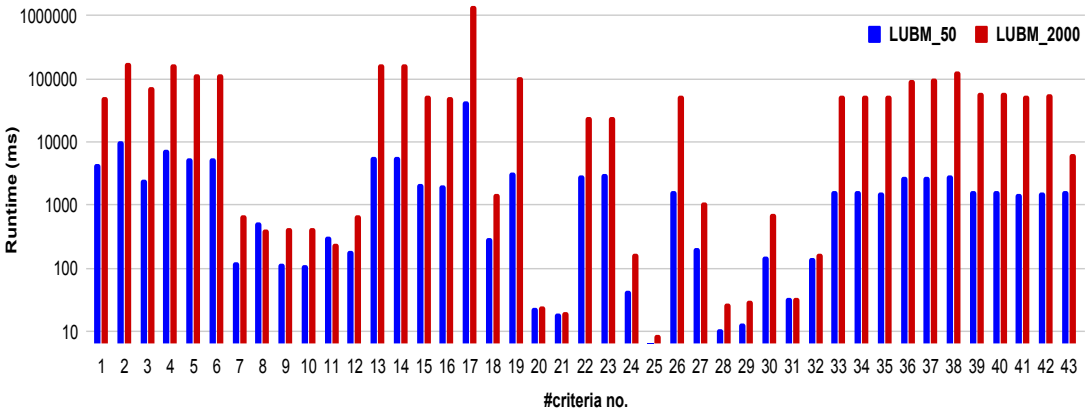

Figure 6.: Overall execution analysis per criteria (log scale).

complex and realistic datasets. We generate a dataset with ten universities (maximum number of universities offered by the benchmark), which contains 1,475,83 axioms. Within 195 seconds, OWLStats measures all statistical criteria on a cluster of five worker nodes.

7. Use Cases

OWLStats is a generic software component for computing statistical information about large-scale OWL datasets. In this section, we present three real-world projects that uses our proposed approach.

Use case 1 (SANSA-Stack[17]): OWLStats has been successfully integrated into Scalable Semantic Analytics Stack (SANSA-Stack) framework (Lehmann et al. 2017; Ermilov et al. 2017). SANSA is an open source[18] large-scale processing engine for

[17] https://sansa-stack.net/
[18] https://github.com/SANSA-Stack

efficient processing of large-scale RDF datasets. The need for utilizing fault-tolerant big data frameworks such as Apache Spark and Flink is raised to process this massive amount of data efficiently. SANSA is built on top of Spark, offering a set of facilities for the representation (RDF and OWL), querying, and inference of semantic data. Currently, SANSA-RDF[19] layer supports 32 statistical criteria for RDF datasets. Some of the machine learning algorithms in the inference and machine learning layers are built on the top axioms level. Therefore, we integrated OWLStats into the SANSA framework to support the SANSA-OWL layer to compute 50 statistical criteria based on OWL datasets (Functional, Manchester, and OWL/XML formats).

Use case 2 (PLATOON Project[20]): PLATOON is an EU-funded H2020 project that digitizes the energy sector with the adoption of AI techniques. The objective is to increase renewable energy consumption, smart grids management, and energy efficiency. The PLATOON reference architecture is used to construct and deploy scalable and replicable energy management solutions. PLATOON partners use many large-scale proprietary ontologies that reflect energy-related tools such as wind turbines, geo-location, weather, etc. OWLStats is used in PLATOON to collect statistical information about the inner structure of the underlying datasets to assist in the development of cluster and classification algorithms. For example, OWLStats is used to calculate how many classes are used on an accident ontology for classification purposes.

Use case 3 (ABSTAT): ABSTAT (Spahiu et al. 2016) is a summarization framework to support linked set understanding. ABSTATS[21] framework can provide compact as well as concise summaries for a given dataset. To derive summarization for more input datasets of OWL formats, OWLStats will be used.

8. Availability and Sustainability

Below, we describe the availability and sustainability that OWLStats fulfils, including the availability of the statistics, as well as how we support sustainability.

- *Availability.* OWLStats is available as an open-source software component in the OWL layer of the SANSA-Stack GitHub repository[22]. SANSA is well-maintained and makes use of community resources such as SANSA website[17], mailing list[23], issue tracker[24], documentation[25], ... etc . It is licensed under the Apache License 2.0.
- *Sustainability.* The sustainability of OWLStats is demonstrated through SANSA-Stack contributors[26]. A new release is published every six months since 2016. In addition, bugs and improvement suggestions can be submitted through the issue tracker on its GitHub repository

[19] https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-rdf
[20] https://platoon-project.eu/
[21] http://abstat.disco.unimib.it/
[22] https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-owl/sansa-owl-spark/src/main/scala/net/sansa_stack/owl/spark/stats
[23] https://sansa-stack.net/community/#MailingLists
[24] https://github.com/SANSA-Stack/SANSA-Stack/issues
[25] https://sansa-stack.net/user-guide/
[26] https://sansa-stack.net/community/#Contributors

9. Conclusion and Future Work

In this paper, we proposed the OWLStats approach that is a flexible, robust, and scalable approach. OWLStats is capable of computing 50 statistical criteria for large-scale OWL datasets. Astonishingly, after reviewing the literature, we found no tool capable of calculating these statistics for such datasets. Accordingly, OWLStats is the first attempt at developing a scalable, in terms of both data and node scalability, open-source distributed framework for computing comprehensive statistical information about OWL datasets. OWLStats has successfully generated 50 statistical criteria about large-scale datasets (+40 GB in size) on a cluster of five worker nodes. Three real-world use cases for OWLStats are presented; i.e., SANSA framework, PLATOON, and ABSTAT. To achieve the sustainability of OWLStats, we made it available as a software component in the OWL layer of the SANSA-Stack, which is constantly maintained by SANSA-Stack contributors. We carried out two experiments to test OWLStats's efficiency as well as the data and node scalability. The evaluation results demonstrated that OWLStats is usable and effective for unveiling the structure of large-scale OWL datasets in various OWL formats, i.e., Functional, Manchester, and OWL/XML. It achieved near-linear scalability of output in the sense of speedup, therefore the scalability of the proposed approach is assured.

To further our research, we are planning to add more criteria that cover further OWL axioms structures. Moreover, we aim at introducing further improvements in terms of code optimization, such as utilizing different persisting strategies as well as Alluxio[27] for bridging the gap between the application and storage system. Alluxio is the first open-source data platform for analytics and cloud AI in the world.

Acknowledgements

This work has been supported by the following EU Horizon2020 projects: LAMBDA project (GA no. 809965) and PLATOON (GA no. 872592).

References

Ali, M., S. Fathalla, S. Ibrahim, M. Kholief, and Y. F. Hassan (2019). Cloe: a cross-lingual ontology enrichment using multi-agent architecture. *Enterprise Information Systems 13*(7-8), 1002–1022.

Auer, S., J. Demter, M. Martin, and J. Lehmann (2012). Lodstats–an extensible framework for high-performance dataset analytics. In *International Conference on Knowledge Engineering and Knowledge Management*, pp. 353–362. Springer.

Beckett, D. (2001). The design and implementation of the redland rdf application framework. In *Proceedings of the 10th international conference on World Wide Web*, pp. 449–456.

Böhm, C., J. Lorey, and F. Naumann (2011). Creating void descriptions for web-scale data. *Journal of Web Semantics 9*(3), 339–345.

da Silva Serapião Leal, G., W. Guédria, and H. Panetto (2020). A semi-automated system for interoperability assessment: an ontology-based approach. *Enterprise Information Systems 14*(3), 308–333.

d'Aquin, M. and E. Motta (2011). Watson, more than a semantic web search engine. *Semantic Web 2*(1), 55–63.

[27] https://docs.alluxio.io/os/user/stable/en/Overview.html

De Giacomo, G., D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati (2018). Using ontologies for semantic data integration. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, pp. 187–202. Springer.

Ding, L., T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs (2004). 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.

Ermilov, I., J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, S. Bin, N. Chakraborty, H. Petzka, M. Saleem, et al. (2017). The tale of sansa spark. In *International Semantic Web Conference (Posters, Demos & Industry Tracks)*.

Facchiano, A. (2017). Bioinformatic resources for the investigation of proteins and proteomes. *Peptidomics 3*(1), 1–10.

Gottron, T., M. Knauf, S. Scheglmann, and A. Scherp (2013). A systematic investigation of explicit and implicit schema information on the linked open data cloud. In *Extended Semantic Web Conference*, pp. 228–242. Springer.

Guo, Y., Z. Pan, and J. Heflin (2005). Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web 3*(2-3), 158–182.

Hausenblas, M., D. Ayers, L. Feigenbaum, T. Heath, W. Halb, and Y. Raimond (2012). The statistical core vocabulary (scovo). *Digital Enterprise Research Institute (DERI), DERI Specification*.

Kappelman, L. A. and J. A. Zachman (2013). The enterprise and its architecture: ontology & challenges. *Journal of Computer Information Systems 53*(4), 87–95.

Langegger, A. and W. Woss (2009). Rdfstats-an extensible rdf statistics generator and library. In *2009 20th International Workshop on Database and Expert Systems Application*, pp. 79–83. IEEE.

Lehmann, J., G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C. N. Ngomo, et al. (2017). Distributed semantic analytics using the sansa stack. In *International Semantic Web Conference*, pp. 147–155. Springer.

Li, H. and Q. Sima (2015). Parallel mining of owl 2 el ontology from large linked datasets. *Knowledge-Based Systems 84*, 10–17.

Ma, L., X. Sun, F. Cao, C. Wang, X. Wang, N. Kanellos, D. Wolfson, and Y. Pan (2009). Semantic enhancement for enterprise data management. In *International Semantic Web Conference*, pp. 876–892. Springer.

Matentzoglu, N., S. Bail, and B. Parsia (2013). A corpus of owl dl ontologies. *Description Logics 1014*, 829–41.

Mishra, S., M. N. Sahoo, A. K. Sangaiah, and S. Bakshi (2021). Nature-inspired cost optimisation for enterprise cloud systems using joint allocation of resources. *Enterprise Information Systems 15*(2), 174–196.

Mohamed, H., S. Fathalla, J. Lehmann, and H. Jabeen (2020). OWLStats: Distributed computation of owl dataset statistics. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pp. In press. IEEE.

Pileggi, S., A. A. Lopez-Lorca, and G. Beydoun (2018). Ontologies in software engineering. In *29th Australasian Conference on Information Systems (ACIS2018)*, pp. 1–8.

Rajabi, Z., B. Minaei, and M. A. Seyyedi (2013). Enterprise architecture development based on enterprise ontology. *Journal of theoretical and applied electronic commerce research 8*(2), 85–95.

Sejdiu, G., I. Ermilov, J. Lehmann, and M. N. Mami (2018). Distlodstats: Distributed computation of rdf dataset statistics. In *International Semantic Web Conference*, pp. 206–222. Springer.

Shvachko, K., H. Kuang, S. Radia, R. Chansler, et al. (2010). The hadoop distributed file system. In *MSST*, Volume 10, pp. 1–10.

Spahiu, B., R. Porrini, M. Palmonari, A. Rula, and A. Maurino (2016). Abstat: ontology-driven linked data summaries with pattern minimalization. In *European Semantic Web Conference*, pp. 381–395. Springer.

Suga, T. and J. Iijima (2018). Algebra for enterprise ontology: towards analysis and synthesis of enterprise models. *Enterprise Information Systems 12*(3), 341–370.

Wongthongtham, P., U. Pakdeetrakulwong, and S. H. Marzooq (2017). Ontology annotation for software engineering project management in multisite distributed software development environments. In *Software Project Management for Distributed Computing*, pp. 315–343. Springer.

Wood, D. (2010). *Linking enterprise data*. Springer Science & Business Media.

Xhafa, F. (2021). Cloud services, storage and communications at large scale for reliable enterprise systems. *Enterprise Information Systems 15*(2), 131–132.