A Scalable Framework for Quality Assessment of RDF Datasets

Gezim Sejdiu[1], Anisa Rula[1,2], Jens Lehmann[1,3] and Hajira Jabeen[1]

[1] Smart Data Analytics, University of Bonn, Germany
sejdiu@cs.uni-bonn.de, rula@cs.uni-bonn.de, jabeen@cs.uni-bonn.de,
jens.lehmann@cs.uni-bonn.de
[2] University of Milano-Bicocca, Department of Computer Science, Systems and
Communication (DISCo), Italy
anisa.rula@disco.unimib.it
[3] Fraunhofer IAIS, Germany
jens.lehmann@iais.fraunhofer.de

Abstract. Over the last years, Linked Data has grown continuously. Today, we count more than 10,000 datasets being available online following Linked Data standards. These standards allow data to be machine readable and inter-operable. Nevertheless, many applications, such as data integration, search, and interlinking, cannot take full advantage of Linked Data if it is of low quality. There exist a few approaches for the quality assessment of Linked Data, but their performance degrades with the increase in data size and quickly grows beyond the capabilities of a single machine. In this paper, we present DistQualityAssessment – an open source implementation of quality assessment of large RDF datasets that can scale out to a cluster of machines. This is the first distributed, in-memory approach for computing different quality metrics for large RDF datasets using Apache Spark. We also provide a quality assessment pattern that can be used to generate new scalable metrics that can be applied to big data. The work presented here is integrated with the SANSA framework and has been applied to at least three use cases beyond the SANSA community. The results show that our approach is more generic, efficient, and scalable as compared to previously proposed approaches.

1 Introduction

Large amounts of data are being published openly to Linked Data by different data providers. A multitude of applications such as semantic search, query answering, and machine reading [18] depend on these large-scale[4] RDF datasets. The quality of underlying RDF data plays a fundamental role in large-scale data consuming applications. Measuring the quality of linked data spans a number of dimensions including but not limited to: accessibility, interlinking, performance, syntactic validity or completeness [22]. Each of these dimensions can be expressed through one or more quality

---

[4] `http://lodstats.aksw.org/`

metrics. Considering that each quality metric tries to capture a particular aspect of the underlying data, numerous metrics are usually provided against the given data that may or may not be processed simultaneously.

On the other hand, the limited number of existing techniques of quality assessment for RDF datasets are not adequate to assess data quality at large-scale and these approaches mostly fail to capture the increasing volume of big data. To date, a limited number of solutions have been conceived to offer quality assessment of RDF datasets [11,13,4,10]. But, these methods can either be used on a small portion of large datasets [13] or narrow down to specific problems e.g., syntactic accuracy of literal values [4], or accessibility of resources [17]. In general, these existing efforts show severe deficiencies in terms of performance when data grows beyond the capabilities of a single machine. This limits the applicability of existing solutions to medium-sized datasets only, in turn, paralyzing the role of applications in embracing the increasing volumes of the available datasets.

To deal with big data, tools like Apache Spark[5] have recently gained a lot of interest. Apache Spark provides scalability, resilience, and efficiency for dealing with large-scale data. Spark uses the concepts of Resilient Distributed Datasets (RDDs) [21] and performs operations like transformations and actions on this data in order to effectively deal with large-scale data.

To handle large-scale RDF data, it is important to develop flexible and extensible methods that can assess the quality of data at scale. At the same time, due to the broadness and variety of quality assessment domain and resulting metrics, there is a strong need to provide a generic pattern to characterize the quality assessment of RDF data in terms of scalability and applicability to big data.

In this paper, we borrow the concepts of data *transformation* and *action* from Spark and present a pattern for designing quality assessment metrics over large RDF datasets, which is inspired by design patterns. In software engineering, design patterns are general and reusable solutions to common problems. Akin to design pattern, where each pattern acts like a blueprint that can be customized to solve a particular design problem, the introduced concept of Quality Assessment Pattern ($\mathcal{QAP}$) represents a generalized blueprint of scalable quality assessment metrics. In this way, the quality metrics designed following $\mathcal{QAP}$ can exhibit the ability to achieve scalability to large-scale data and work in a distributed manner. In addition, we also provide an open source implementation and assessment of these quality metrics in Apache Spark following the proposed $\mathcal{QAP}$.

Our contributions can be summarized in the following points:

– We present a Quality Assessment Pattern $\mathcal{QAP}$ to characterize scalable quality metrics.
– We provide DistQualityAssessment[6] – a distributed (open source) implementation of quality metrics using Apache Spark.
– We perform an analysis of the complexity of the metric evaluation in the cluster.

---

[5] https://spark.apache.org/

[6] https://github.com/SANSA-Stack/SANSA-RDF/tree/develop/sansa-rdf-spark/src/main/scala/net/sansa_stack/rdf/spark/qualityassessment

- We evaluate our approach and demonstrate empirically its superiority over a previous centralized approach.
- We integrated the approach into the SANSA[7] framework. SANSA is actively maintained and uses the community ecosystem (mailing list, issues trackers, continues integration, web-site etc.).
- We briefly present three use cases where DistQualityAssessment has been used.

The paper is structured as follows: Our approach for the computation of RDF dataset quality metrics is detailed in section 2 and evaluated in section 3. Related work on the computation of quality metrics for RDF datasets is discussed in section 5. Finally, we conclude and suggest planned extensions of our approach in section 6.

2   Approach

In this section, we first introduce basic notions used in our approach, the formal definition of the proposed quality assessment pattern and then describe the workflow.

2.1   Quality Assessment Pattern

Data quality is commonly conceived as a multi-dimensional construct [2] with a popular notion of 'fitness for use' and can be measured along many dimensions $\mathcal{D}$ such as accuracy ($d_{accu} \in \mathcal{D}$), completeness ($d_{comp} \in \mathcal{D}$) and timeliness ($d_{tmls} \in \mathcal{D}$). The assessment of a quality dimensions d is based on quality metrics $\mathcal{QM} = m_1, m_2...m_k$ where m_i is a heuristic that is designed to fit a specific assessment dimension. The following definitions form the basis of $\mathcal{QAP}$.

Definition 1 (Filter). *Let $\mathcal{F} = f_1, f_2...f_l$ be a set of filters where each filter f_i sets a criteria for extracting predicates, objects, subjects, or their combination. A filter f_i takes a set of RDF triples as input and returns a subgraph that satisfies the filtering criteria.*

Definition 2 (Rule). *Let $\mathcal{R} = r_1, r_2...r_j$ be a set of rules where each rule r_i sets a conditional criteria. A rule takes a subgraph as input and returns a new subgraph that satisfies the conditions posed by the rule r_i.*

Definition 3 (Transformation). *A transformation $\tau : \mathcal{G} \rightarrow \mathcal{G}'$ is an operation that applies rules defined by $\mathcal{R}$ on the RDF graph $\mathcal{G}$ and returns an RDF subgraph $\mathcal{G}'$. A transformation τ can be a union $\cup$ or intersection $\cap$ of other transformations.*

Definition 4 (Action). *An action $\alpha : \mathcal{G} \rightarrow \mathbb{R}$ is an operation that triggers the transformation of rules on the filtered RDF graph $\mathcal{G}'$ and generates a numerical value. Action α is the count of elements obtained after performing a τ operation.*

Definition 5 (Quality Assessment Pattern $\mathcal{QAP}$). *The Quality Assessment Pattern $\mathcal{QAP}$ is a reusable template to implement and design scalable quality metrics. The $\mathcal{QAP}$ is composed of transformations and actions. The output of a $\mathcal{QAP}$ is the outcome of an action returning a numeric value against the particular metric.*

---

[7] http://sansa-stack.net/

Table 1. Quality Assessment Pattern

```
Quality Metric := Action |(Action OP Action)
OP             := * |− |/ |+
Action         := Count(Transformation)
Transformation := Rule(Filter) |(Transformation BOP Transformation)
Filter         := getPredicates ~?p |getSubjects ~?s |getObjects ~?o |getDistinct(Filter)
                  |Filter or Filter |Filter && Filter)
Rule           := isURI(Filter) |isIRI(Filter) |isInternal(Filter) |isLiteral(Filter)
                  |!isBroken(Filter) |hasPredicateP |hasLicenceAssociated(Filter)
                  |hasLicenceIndications(Filter) |isExternal(Filter) |hasType((Filter)
                  |isLabeled(Filter)
BOP            := ∩ — ∪
```

$\mathcal{QAP}$ is inspired by Apache Spark operations and designed to fit different data quality metrics (for more details see Table 1). Each data *quality metric* can be defined following the $\mathcal{QAP}$. Any given data quality metric m_i that is represented through the $\mathcal{QAP}$ using transformation τ and action α operations can be easily transformed into Spark code to achieve scalability.

Table 2 demonstrates a few selected quality metrics defined against proposed $\mathcal{QAP}$. As shown in Table 2, each quality metric can contain multiple rules, filters or actions. It is worth mentioning that action count(triples) returns the total number of triples in the given data. This can also be seen that the action can be an arithmetic combination of multiple actions i.e. ratio, sum etc. We illustrate our proposed approach on some metrics selected from [10,22]. Given that the aim of this paper is to show the applicability of the proposed approach and comparison with existing methods, we have only selected those which are already provided out-of-box in Luzzu.

2.2 System Overview

In this section, we give an overall description of the data model and the architecture of DistQualityAssessment. We model and store RDF graphs $\mathcal{G}$ based on the basic building block of the Spark framework, RDDs. RDDs are in-memory collections of records that can be operated in parallel on a large distributed cluster. RDDs provide an interface based on *coarse-grained* transformations (e.g *map*, *filter* and *reduce*): operations applied on an entire RDD. A *map* function transforms each value from an input RDD into another value while applying τ rules. A *filter* transforms an input RDD to an output RDD, which contains only the elements that satisfy a given condition. *Reduce* aggregates the RDD elements using a specific function from τ.

The computation of the set of quality metrics $\mathcal{QM}$ is performed using Spark as depicted in Figure 1. Our approach consists of four steps:

Defining quality metrics parameters (step 1) The metric definitions are kept in a dedicated file which contains most of the configurations needed for the system to evaluate quality metrics and gather result sets.

	Metric	Transformation τ	Action α
L1	Detection of a Machine Readable License	`r = hasLicenceAssociated(?p)`	α = `count(r)` α > `0 ? 1 : 0`
L2	Detection of a Human Readable License	`r = isURI(?s)` ∩ `hasLicenceIndications(?p)` ∩ `isLiteral(?o)` ∩ `isLicenseStatement(?o)`	α = `count(r)` α > `0 ? 1 : 0`

I2	Linkage Degree of Linked External Data Providers	`r_1 = isIRI(?s)` ∩ `internal(?s)` ∩
		   `isIRI(?o)` ∩ `external(?o)`
		`r_2 = isIRI(?s)` ∩ `external(?s)` ∩
		   `isIRI(?o)` ∩ `internal(?o)`
		`r_3 = r_1` ∪ `r_2`
			α_1 = `count(r_3)`
			α_2 = `count(triples)`
			α = `a_1/a_2`

U1	Detection of a Human Readable Labels	`r_1 = isURI(?s)` ∩ `isInternal(?s)` ∩
		   `isLabeled(?p)`
		`r_2 = isInternal(?p)` ∩ `isLabeled(?p)`
		`r_3 = isURI(?o)` ∩ `isInternal(?o)` ∩
		   `isLabeled(?p)`
			α_1 = `count(r_1)` +
			      `count(r_2)` +
			      `count(r_3)`
			α_2 = `count(triples)`
			α_1/ α_2

RC1	Short URIs	`r_1 = isURI(?s)` ∪ `isURI(?p)` ∪ `isURI(?o)` `r_2 = resTooLong(?s, ?p, ?o)`	α_1 =`count(r_2)` α_1/`count(triples)`
SV3	Identification of Literals with Malformed Datatypes	`r = isLiteral(?o)` ∩ `getDatatype(?o)` ∩ `isLexicalFormCompatibleWithDatatype(?o)`	α = `count(r)`
CN2	Extensional Conciseness	`r = isURI(?s)` ∩ `isURI(?o)`	α_1 = `count(r)` α_2 = `count(triples)` (α_2- α_1)/ α_2

Table 2. Definition of selected metrics following $\mathcal{QAP}$.
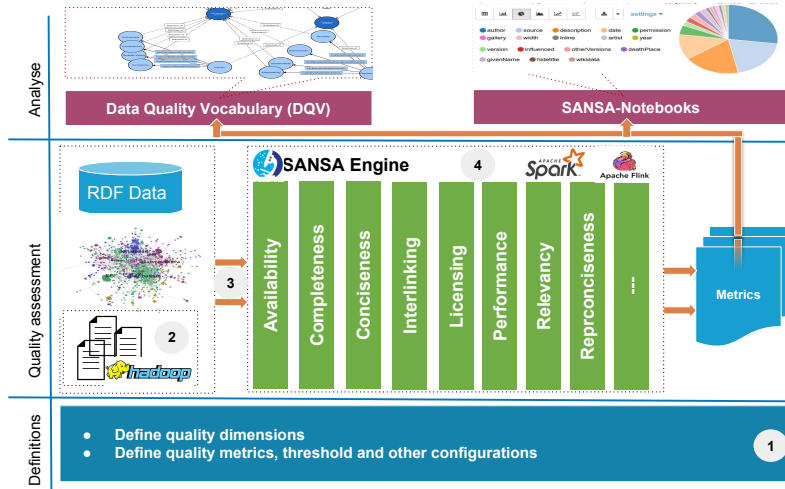

Fig. 1. Overview of distributed quality assessment's abstract architecture.

Retrieving the RDF data (step 2) RDF data first needs to be loaded into a large-scale storage that Spark can efficiently read from. We use Hadoop Distributed File-System[8] (HDFS). HDFS is able to fit and stores any type of data in its Hadoop-native format and parallelizes them across a cluster while replicating them for fault tolerance. In such a distributed environment, Spark automatically adopts different data locality strategies to perform computations as close to the needed data as possible in HDFS and thus avoids data transfer overhead.

Parsing and mapping RDF into the main dataset (step 3) We first create a distributed dataset called *main dataset* that represent the HDFS file as a collection of triples. In Spark, this dataset is parsed and loaded into an RDD of triples having the format *Triple<(s,p,o)>*.

Quality metric evaluation (step 4) Considering the particular quality metric, Spark generates an execution plan, which is composed of one or more τ transformations and α actions. The numerical output of the final action is the quality of the input RDF corresponding to the given metric.

2.3   Implementation

We have used the Scala[9] programming language API in Apache Spark to provide the distributed implementation of the proposed approach.

The DistQualityAssessment (see algorithm 1) constructs the *main dataset* (line 1) while reading RDF data (e.g. NTriples file or any other RDF serialization format) and converts it into an RDD of triples. This latter undergoes the transformation operation of applying the filtering through rules in *R* and producing a new *filtered* RDD ($\mathcal{G}'$) (line 5). At the end, $\mathcal{G}'$ will serve as an input to the next step which applies a set of α actions (line 8). The output of this step is the metric output represented as a numerical value (line 8). The result set of different quality metrics (line 12) can be further visualized and monitored using SANSA-Notebooks [12].
The user can also choose to extract the output in a machine-readable format (line 10). We have used the data quality vocabulary[10] (DQV) to represent the quality metrics.

Furthermore, we also provide a Docker image of the system integrated within the BDE platform[11] - an open source Big Data processing platform allowing users to install numerous big data processing tools and frameworks and create working data flow applications.

The work done here (available under *Apache License 2.0*) has been integrated into SANSA [16], an open source[12] *data flow processing engine* for scalable processing of large-scale RDF datasets. SANSA uses Spark offering fault-tolerant, highly available and scalable approaches to process massive sized datasets efficiently. SANSA provides the facilities for semantic data representation, querying, inference, and analytics at

---

[8] `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

[9] `https://www.scala-lang.org/`

[10] `https://www.w3.org/TR/vocab-dqv/`

[11] `https://github.com/big-data-europe`

[12] `https://github.com/SANSA-Stack`

Algorithm 1: Spark-based parallel quality assessment algorithm.

---

   input : *RDF*: an RDF dataset, *param*: quality metrics parameters.
   output: *dqv* description or *metric* numerical value

1   *triples = spark.**rdf**(lang)(input)*

2   *triples.persist()*

3   *dqv ← ∅*

4   **foreach** *m ∈ param.getListOfMetrics* **do**

5   *triples ← triples.Tranform { t =>*

6   *rule ← m.Rule*

7   *t.apply(rule) }*

8   *metric ← triples.apply(m.Action)*

9   **if** *m.hasDQVdescription* **then**

10   *dqvify ← metric.dqvify()*

11   *dqv.add(dqvify)*

12   **return** *(dqv, metric)*

---

scale. Being part of this integration, DistQualityAssessment can take advantage of having the same user community as well as infrastructure build via SANSA project. Doing so, it can also ensure the sustainability of the tool given that SANSA is supported by several grants until at least 2021.

Complexity Analysis We deem that the overall time complexity of the distributed quality assessment evaluation is $O(n)$. The performance of metrics computation depends on data shuffling (while filtering using rules in R) and data scanning. Our approach performs a direct mapping of any quality metric designed using $\mathcal{QAP}$ into a sequence of Spark-compliant Scala-commands, as a consequence, most of the operators used are a series of transformations like *map*, *filter* and *reduce*. The complexity of *map* and *filter* is considered to be linear with respect to the number of triples associated with it. The complexity of a metric then depends on the α operation that returns the count of the filtered output. This later step works on the distributed RDD between p nodes which imply that the complexity of each node then becomes $O(n/p)$, where n is number of input triples. Let be $O(\tau)$ a complexity of τ, then the complexity of the metric will be $O(n/p * O(\tau))$. This indicates that the runtime increases linearly when the size of an RDD increases and decreases linearly when more nodes p are added to the cluster.

3   Evaluation

The main aim of DistQualityAssessment is to serve massive large-scale real-life RDF datasets. We are interested in addressing the following additional questions.

– **Flexibility**: How fast our approach processes different types of metrics?
– **Scalability**: How large are the RDF datasets that DistQualityAssessment can scale to? What is the system speedup w.r.t the number of nodes in a cluster mode?
– **Efficiency**: How well our approach performs compared with other state-of-the-art systems on real-world datasets?

In the following, we present our experimental setup including the datasets used. There-after, we give an overview of our results.

3.1   Experimental Setup

We chose two real-world and one synthetic datasets for our experiments:

1. *DBpedia* [15] (v 3.9) – a cross domain dataset. DBpedia is a knowledge base with a large ontology. We build a set of 3 pipelines of increasing complexity: (i) $M^{en}_{DBpedia}$ ($\approx$ 813M triples); (ii) $M^{de}_{DBpedia}$ ($\approx$ 337M triples); (iii) $M^{fr}_{DBpedia}$ ($\approx$ 341M triples). DBpedia has been chosen because of its popularity in the Semantic Web community.
2. *LinkedGeoData* [20] – a spatial RDF knowledge base derived from OpenStreetMap.
3. *Berlin SPARQL Benchmark (BSBM)* [6] – a synthetic dataset based on an e-commerce use case containing a set of products that are offered by different vendors and reviews posted by consumers about products. The benchmark provides a data generator, which can be used to create sets of connected triples of any particular size.

Properties of the considered datasets are given in Table 3.

Table 3. Dataset summary information (nt format).

$\longrightarrow$	LinkedGeoData	DBpedia			BSBM		
		en	de	fr	2GB	20GB	200GB
#nr. of triples	1,292,933,812	812,545,486	336,714,883	340,849,556	8,289,484	81,980,472	817,774,057
size (GB)	191.17	114.4	48.6	49.77	2	20	200

We implemented DistQualityAssessment using Spark-2.4.0, Scala 2.11.11 and Java 8, and all the data were stored on the the HDFS cluster using Hadoop 2.8.0. The experiments in local mode are all performed on a single instance of the cluster. Specifically, we compare our approach with Luzzu [10] v4.0.0, a state-of-the-art quality assessment system[13]. All distributed experiments were carried out on a small cluster of 7 nodes (1 master, 6 workers): Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (32 Cores), 128 GB RAM, 12 TB SATA RAID-5. The machines were connected via a Gigabit network. All experiments have been executed three times and the average value is reported in the results.

3.2   Results

We evaluate the proposed approach using the above datasets to compare it against Luzzu [10]. We carry out two sets of experiments. First, we evaluate the runtime of our distributed approach in contrast to Luzzu. Second, we evaluate the horizontal scalability via increasing nodes in the cluster. Results of the experiments are presented in

---

[13] https://github.com/Luzzu/Framework

Table 4, Figure 2 and Figure 3. Based on the metric definition, some metrics make use of external access (e.g. Dereferenceability of Forward Links) which leads to a significant increase in Spark processing due to network latency. For the sake of the evaluation we have suspended such metrics. As of that, we choose seven metrics (see Table 2 for more details) where the level of difficulty vary from simple to complex according to combination of transformation/action operations involved.

Performance evaluation on large-scale RDF datasets   We started our experiments by evaluating the *speedup* gained by adopting a distributed implementation of quality assessment metrics using our approach, and compare it against Luzzu. We run the experiments on five datasets ($DBpedia_{en}$, $DBpedia_{de}$, $DBpedia_{fr}$, $LinkedGeoData$ and $BSBM_{200GB}$). Local mode represent a single instance of the cluster without any tuning of Spark configuration and the cluster mode includes further tuning. Luzzu was run in a local environment on a single machine with two strategies: (1) streaming the data for each metric separately, and (2) one stream/load – all metrics evaluated just once.

Table 4. Performance evaluation on large-scale RDF datasets.

| | Runtime (m) (mean/std) | | | | |
| $\longrightarrow$ | **Luzzu** | | **DistQualityAssessment** | | |
	a) single	b) joint	c) local	d) cluster	e) speedup ratio w.r.t Luzzu |DistQualityAssessment$^{c)}$
LinkedGeoData	Fail	Fail	446.9/63.34	7.79/0.54	n/a|56.4x
$DBpedia_{en}$	Fail	Fail	274.31/38.17	1.99/0.04	n/a|136.8x
$DBpedia_{de}$	Fail	Fail	161.4/24.18	0.46/0.04	n/a|349.9x
$DBpedia_{fr}$	Fail	Fail	195.3/26.16	0.38/0.04	n/a|512.9x
$BSBM_{200GB}$	Fail	Fail	454.46/78.04	7.27/0.64	n/a|61.5x
$BSBM_{0.01GB}$	2.64/0.02	2.65/0.01	0.04/0.0	0.42/0.04	65x|(-0.9x)
$BSBM_{0.02GB}$	5.9/0.16	5.66/0.02	0.04/0.0	0.43/0.03	146.5x|(-0.9x)
$BSBM_{0.05GB}$	16.38/0.44	15.39/0.21	0.05/0.0	0.46/0.02	326.6x|(-0.9x)
$BSBM_{0.1GB}$	40.59/0.56	37.94/0.28	0.06/0.0	0.44/0.05	675.5x|(-0.9x)
$BSBM_{0.2GB}$	101.8/0.72	101.78/0.64	0.07/0.0	0.4/0.03	1453.3|(-0.8x)
$BSBM_{0.5GB}$	459.19/18.72	468.64/21.7	0.15/0.01	0.48/0.03	3060.3x|(-0.7x)
$BSBM_{1GB}$	1454.16/10.55	1532.95/51.6	0.4/0.02	0.56/0.02	3634.4x|(-0.3x)
$BSBM_{2GB}$	Timeout	Timeout	3.19/0.16	0.62/0.04	n/a|4.1x
$BSBM_{10GB}$	Timeout	Timeout	29.44/0.14	0.52/0.01	n/a|55.6x
$BSBM_{20GB}$	Fail	Fail	34.32/9.22	0.75/0.29	n/a|44.8x

(Left margin labels: "Large-scale" for the first group; "Small to medium" for the second group.)

Table 4 shows the performance of two approaches applied to five datasets. In Table 4 we indicate "Timeout" whenever the process did not complete within a certain amount of time[14] and "Fail" when the system crashed before this timeout delay. Column Luzzu$^{a)}$ represents the performance of Luzzu on bulk load – considering each metric as a sequence of the execution, on the other hand, the column Luzzu$^{b)}$ reports on the performance of Luzzu using a joint load by evaluating each metric using one

---

[14] We set the timeout delay to 24 hours of the quality assessment evaluation stage.

load. The last columns reports on the performance of DistQualityAssessment run on a local mode c), cluster mode d) and speedup ratio of our approach compared to Luzzu[b] $(d)/b) - 1$) and itself evaluated on local mode $(d)/c) - 1$) is reported on the column e). We observe that the execution of our approach finishes with all the datasets whereas this is not the case with Luzzu which either timeout or fail at some point.

Unfortunately, Luzzu was not capable of evaluating the metrics over large-scale RDF datasets from Table 4 (part one). For that reason we run yet another set of experiments on very small datasets which Luzzu was able to handle. Second part of the Table 4 shows a performance evaluation of our approach compared with Luzzu on very small RDF datasets. In some cases (e.g. RC1, SV3) for a very small dataset Luzzu performs better than our approach with a small margin of runtime in the local mode. It is due to the fact that in the streaming mode, when Luzzu[a] finds the first statement which fulfills the condition (e.g.finding the shortest URIs), it stops the evaluation and return the results. On the contrary, our approach evaluates the metrics over the whole dataset exploiting the fault-tolerance and resilient features build in Spark. In other cases Luzzu suffers from significant slowdowns, which are several orders of magnitude slower. Therefore, its average runtime over all metrics is worst as compared to our approach. It is important to note that our approach on these very small datasets degrades while running on the cluster mode. This is because of the network overhead while shuffling the data, but it outperforms Luzzu[a,b] when considering "average runtime" over all the metrics (even for very small datasets).

Findings shown in Table 4 depict that our approach starts outperforming when the size of the dataset grows (e.g. $BSBM_{2GB}$). The runtime in the cluster mode stays constant when the size of the data fits into the main memory of the cluster. On other hand, Luzzu is not able to evaluate the metrics when the size of data starts increasing, the time taken lasts beyond the delay we set for small datasets. Because of the large differences, we have used a logarithmic scale to better visualize these results.

Scalability performance analysis   In this experiment we evaluate the efficiency of our approach. Figure 2 and Figure 3 illustrates the results of the comparative efficiency analysis.

Data scalability To measure the performance of *size-up* scalability of our approach, we run experiments on five different sizes. We fix the number of nodes to 6 and grow the size of datasets to measure whether DistQualityAssessment can deal with larger datasets. For this set of experiments we consider BSBM benchmark tool to generate syntethic datasets of different sizes, since the real-world dataset are considered to be unique in their size and attributes.

We start by generating a dataset of 2GB. Then, we iteratively increase the size of datasets. On each dataset, we run our approach and the runtime is reported on Figure 2. The *x*-axis shows the size of BSBM dataset with an increasing order of 10x magnitude.

By comparing the runtime (see Figure 2), we note that the execution time increases linearly and is near-constant when the size of the dataset increases. As expected, it stays near-constant as long as the data fits in memory. This demonstrates one of the advantages of utilizing the in-memory approach for performing the quality assessment computation. The overall time spent in data read/write and network communication found in disk-based approaches is saved. However, when the data overflows the memory, and
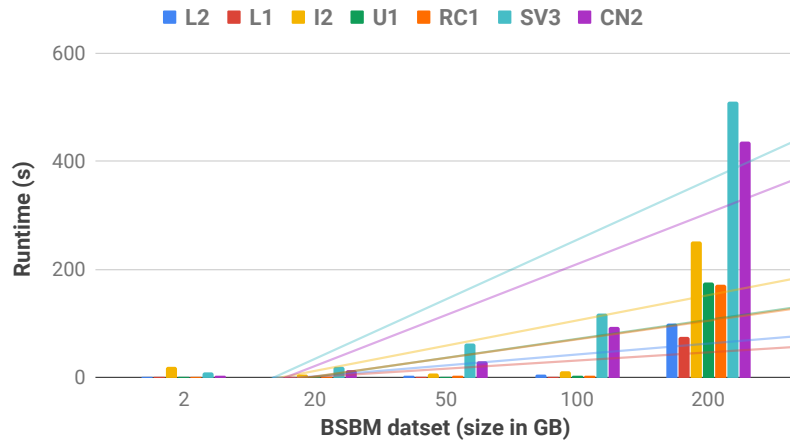
Fig. 2. Sizeup performance evaluation.

it is spilled to disk, the performance degrades. These results show the scalability of our algorithm in the context of size-up.

Node scalability In order to measure node scalability, we vary the number of the workers on our cluster. The number of workers have varied from 1, 2, 3, 4 and 5 to 6.
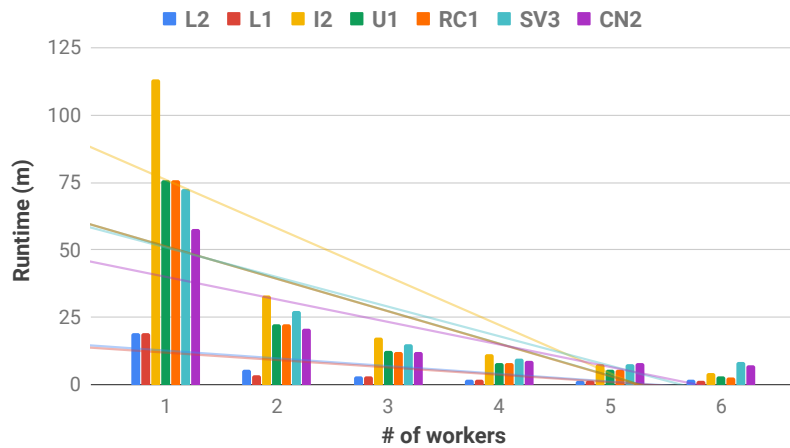

Fig. 3. Node scalability performance evaluation.

Figure 3 shows the speedup for $BSBM_{200GB}$ with the various number of worker nodes. We can see that as the number of workers increases, the execution time cost-decrease is almost linear. The execution time decreases about 14 times (from 433.31

minutes down to 28.8 minutes) as cluster nodes increase from one to six worker nodes. The results shown here imply that our approach can achieve near linear scalability in performance in the context of speedup.

Furthermore, we conduct the effectiveness evaluation of our approach. Speedup S is an important metric to evaluate a parallel algorithm. It is defined as a ratio $S = T_s/T_n$, where T_s represents the execution time of the algorithm run on a single node and T_n represents the execution time required for the same algorithm on n nodes with the same configuration and resources. Efficiency is defined as a ratio $E = S/n = T_s/nT_n$ which measures the processing power being used, in our case the speedup per node. The speedup and efficiency curves of DistQualityAssessment are shown in Figure 5. The trend shows that it achieves almost linearly speedup and even super linear in some cases. The upper curve in the Figure 5 indicates super linear speedup. The speedup grows faster than the number of worker nodes. This is due to the computation task for the metric being computationally intensive, and the data does not fit in the cache when executed on a single node. But it fits into the caches of several machines when the workload is divided amongst the cluster for parallel evaluation. While using Spark, the super linear speedup is an outcome of the improved complexity and runtime, in addition to efficient memory management behavior of the parallel execution environment.

Correctness of metrics   In order to test the correctness of implemented metrics, we assess the numerical values for metrics like L1, L2, and RC1 on very small datasets and the results are found correct w.r.t Luzzu. For metrics like I2 and CN2, Luzzu uses approximate values for faster performance, and that is not the same as getting the exact number as in the case of our implementation.

Overall analysis by metrics   We analyze the overall run-time of the metric evaluation. Figure 4 reports on the run-time of each metric considered in this paper (see Table 2) on both $BSBM_{20GB}$ and $BSBM_{200GB}$ datasets.
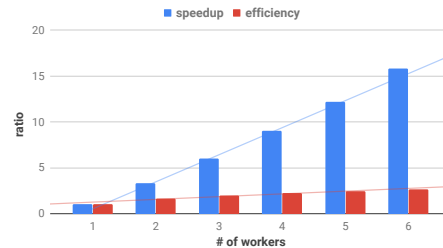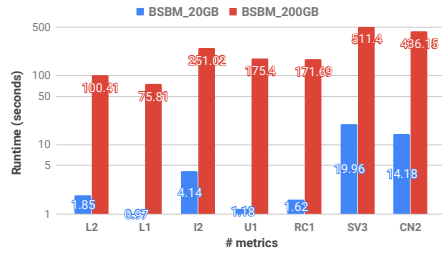

Fig. 4. Overall analysis by metric in the cluster mode (log scale).

Fig. 5. Effectiveness of DistQualityAssessment.

DistQualityAssessment implements predefined quality assessment metrics from [22]. We have implemented these metrics in a distributed manner such that most of them have a run-time complexity of $O(n)$ where n is the number of input triples. The overall performance of analysis for BSBM dataset with two instances is shown in Figure 4. The results obtained show that the execution is sometimes a little longer when there is a

shuffling involved in the cluster compared to when data is processed without movement e.g. Metric L2 and L1. Metric SV3 and CN2 are the most expensive ones in terms of runtime. This is due to the extra overhead caused by extracting the literals for objects, and checking the lexical form of its datatype.

Overall, the evaluation study carried out in this paper demonstrates that distributed computation of different quality measures is scalable and the execution ends in reasonable time given the large volume of data.

4   Use Cases

The proposed quality assessment tool is being used in many use cases. These includes the projects QROWD, SLIPO, and an industrial application by Alethio[15].

QROWD – Crowdsourcing Streaming Big Data Quality Assessment Use Case QROWD[16] is a cross-sectoral streaming Big Data integration project including geographic, transport, meteorological, cross domain and news data, aiming to capitalize on hybrid Big Data integration and analytics methods. One of the major challenges faced in QROWD, is to investigate options for effective and scalable data quality assessment on integrated (RDF) datasets using their crowdsourcing platform. In order to perform this task efficiently and effectively, QROWD uses DistQualityAssessment as an underlying quality assessment framework.

Blockchain – Alethio Use Case   Alethio[17] has build an Ethereum analytics platform that strives to provide transparency over the transaction pool of the whole Ethereum ecosystem. Their 18 billion triple data set[18] contains large scale blockchain transaction data modelled as RDF according to the structure of the Ethereum ontology[19]. Alethio is using SANSA in general and DistQualityAssesment in particular, for performing large-scale batch quality checks, e.g. analysing the quality while merging new data, computing attack pattern frequencies and fraud detection. Alethio uses DistQualityAssesment on a cluster of 100 worker nodes to assess the quality of their ≈7 TB of data.

SLIPO – Scalable Integration and Quality Assured fusion of Big POI data   SLIPO[20] is a project which leverages semantic web technologies for scalable and quality assured integration of large Point of Interest (POI) datasets. One of the key features of the project is the fusion process. SLIPO-fusion receives two different RDF datasets containing POIs and their properties, as well as a set of links between POI entities of the two datasets. SLIPO is using DistQualityAssessment to assess the quality of both input datasets. The SLIPO-fusion produces a third, final dataset, containing consolidated descriptions of the linked POIs. This process is often data and processing intensive, therefore, it requires a scalable mechanism for data quality check. SLIPO uses DistQualityAssessment for fusion validation and quality statistics/assessment to facilitate and assure the quality of the fusion process.

---

[15] https://goo.gl/mJTkPp

[16] http://qrowd-project.eu/

[17] https://aleth.io/

[18] https://medium.com/alethio/ethereum-linked-data-b72e6283812f

[19] https://github.com/ConsenSys/EthOn

[20] http://slipo.eu/

5   Related Work

Even though quality assessment of big datasets is an important research area, it is still largely under-explored. There have been a few works discussing the challenges and issues of big data quality [3,19,8]. Only recently, a few of them have started to address the problem from a practical point of view [10], which is the focus of our work as stated in section 1. In the following, we divide the section between conceptual and practical approaches proposed in the state of the art for big data quality assessment. In [9] the authors propose a big data processing pipeline and a big data quality pipeline. For each of the phases of the processing pipeline they discuss the corresponding phase of the big data quality pipeline. Relevant quality dimensions such as accuracy, consistency and completeness are discussed for the quality assessment of RDF datasets as part of an integration scenario. Given that the quality dimensions and metrics have somehow evolved from relational to Linked Data, it is relevant to understand the evolution of quality dimensions according to the differences between the structural characteristics of the two data models [1]. This allows to manage the huge variability of methods and techniques needed to manage data quality and understand which are the quality dimensions that prevail when assessing large-scale RDF datasets.

   Most of the existing approaches can be applied to small/medium scale datasets and do not horizontally scale [10,14]. The work in [14] presents a methodology for assessing the quality of Linked Data based on a test case generation analogy used for software testing. The idea of this approach is to generate templates of the SPARQL queries (i.e., quality test case patterns) and then instantiate them by using the vocabulary or schema information, thus producing quality test case queries. Luzzu [10] is similar in spirit with our approach in that its objective is to provide a framework for quality assessment. In contrast to our approach, where data is distributed and also the evaluation of metrics is distributed, Luzzu does not provide any large-scale processing of the data. It only uses Spark streaming for loading the data which is not part of the core framework. Another approach proposed for assessing the quality of large-scale medical data implements Hadoop Map/Reduce [7]. It takes advantage of query optimization and join strategies which are tailored to the structure of the data and the SPARQL queries for that particular dataset. In addition, this work, differently from our approach, does not assess any data quality metric defined in [22]. The work in [5] propose a reasoning approach to derive inconsistency rules and implements a Spark-based implementation of the inference algorithm for capturing and cleaning inconsistencies in RDF datasets. The inference generally incurs higher complexity. Our approach is designed for scalability, and we also use Spark-based implementation for capturing inconsistencies in the data. While the approach in [5] needs manual definitions of the inconsistency rules, our approach runs automatically, not only for consistency metrics but also for other quality metrics. In addition, we test the performance of our approach on large-scale RDF datasets while their approach is not experimentally evaluated. LD-Sniffer [17], is a tool for assessing the accessibility of Linked Data resources according to the metrics defined in the Linked Data Quality Model. The limitation of this tool, besides that it is a centralized version, is that it does not provide most of the quality assessment metrics defined in [22]. In addition to above, there is a lack of unified structure to propose and develop new quality metrics that are scalable and less computationally expensive.

Based on the identified limitations of these aforementioned approaches, we have introduced DistQualityAssessment which bases its computation and evaluations mainly in-memory. As a result the computation of the quality metrics show a high performance for large-scale datasets.

6   Conclusions and Future Work

The data quality assessment becomes challenging with the increasing sizes of data. Many existing tools mostly contain a customized data quality functionality to detect and analyze data quality issues within their own domain. However, this process is both data-intensive and computing-intensive and it is a challenge to develop fast and efficient algorithms that can handle large scale RDF datasets.

In this paper, we have introduced DistQualityAssessment, a novel approach for distributed in-memory evaluation of RDF quality assessment metrics implemented on top of the Spark framework. The presented approach offers generic features to solve common data quality checks. As a consequence, this can enable further applications to build trusted data utilities.

We have demonstrated empirically that our approach improves upon previous centralized approach that we have compared against. The benefit of using Spark is that its core concepts (RDDs) are designed to scale horizontally. Users can adapt the cluster sizes corresponding to the data sizes, by dropping when it is not needed and adding more when there is a need for it.

Although we have achieved reasonable results in terms of scalability, we plan to further improve time efficiency by applying intelligent partitioning strategies and persist the data to an even higher extent in memory and perform dependency analysis in order to evaluate multiple metrics simultaneously. We also plan to explore near real-time interactive quality assessment of large-scale RDF data using Spark Streaming. Finally, in the future we intend to develop a declarative plugin for the current work using Quality Metric Language (QML) [10], which gives users the ability to express, customize and enhance quality metrics.

Acknowledgment

References

1. C. Batini, A. Rula, M. Scannapieco, and G. Viscusi. From data quality to big data quality. *J. Database Manag.*, 26(1):60–82, 2015.
2. C. Batini and M. Scannapieco. *Data and Information Quality - Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer, 2016.
3. D. Becker, T. D. King, and B. McMullen. Big data, big data quality problem. In *International Conference on Big Data*, pages 2644–2653. IEEE, 2015.

4. W. Beek, F. Ilievski, J. Debattista, S. Schlobach, and J. Wielemaker. Literally better: Analyzing and improving the quality of literals. *Semantic Web*, 9(1), 2018.
5. S. Benbernou and M. Ouziri. Enhancing data quality by cleaning inconsistent big RDF data. In *International Conference on Big Data*, pages 74–79. IEEE, 2017.
6. C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5:1–24, 2009.
7. S. Bonner, A. S. McGough, I. Kureshi, J. Brennan, G. Theodoropoulos, L. Moss, D. Corsar, and G. Antoniou. Data quality assessment and anomaly detection via map/reduce and linked data: A case study in the medical domain. In *International Conference on Big Data*. IEEE, 2015.
8. L. Cai and Y. Zhu. The challenges of data quality and data quality assessment in the big data era. *Data Science Journal*, 14, 2015.
9. T. Catarci, M. Scannapieco, M. Console, and C. Demetrescu. My (fair) big data. In *International Conference on Big Data*, pages 2974–2979. IEEE, 2017.
10. J. Debattista, S. Auer, and C. Lange. Luzzua methodology and framework for linked data quality assessment. *Journal of Data and Information Quality (JDIQ)*, 8(1):4, 2016.
11. J. Debattista, C. Lange, S. Auer, and D. Cortis. Evaluating the quality of the LOD cloud: An empirical investigation. *Semantic Web*, 9(6):859–901, 2018.
12. I. Ermilov, J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, S. Bin, N. Chakraborty, H. Petzka, M. Saleem, A.-C. N. Ngonga, and H. Jabeen. The Tale of Sansa Spark. In *16th International Semantic Web Conference, Poster & Demos*, 2017.
13. M. Färber, F. Bartscherer, C. Menne, and A. Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018.
14. D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 747–758, 2014.
15. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.
16. J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C. Ngonga Ngomo, and H. Jabeen. Distributed Semantic Analytics using the SANSA Stack. In *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, 2017.
17. N. Mihindukulasooriya, R. García-Castro, and A. Gómez-Pérez. Ld sniffer: A quality assessment tool for measuring the accessibility of linked data. In *Knowledge Engineering and Knowledge Management*, pages 149–152, Cham, 2016. Springer International Publishing.
18. A.-C. Ngonga Ngomo, S. Auer, J. Lehmann, and A. Zaveri. Introduction to linked data and its lifecycle on the web. In *Reasoning Web*, 2014.
19. D. Rao, V. N. Gudivada, and V. V. Raghavan. Data quality issues in big data. In *International Conference on Big Data*, pages 2654–2660. IEEE, 2015.
20. C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, 3(4):333–354, 2012.
21. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX, 2012.
22. A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2015.