CLAMS: Bringing Quality to Data Lakes

Mina Farid
University of Waterloo
mfathy@uwaterloo.ca

Alexandra Roatiş
University of Waterloo
aroatis@uwaterloo.ca

Ihab F. Ilyas
University of Waterloo
ilyas@uwaterloo.ca

Hella-Franziska Hoffmann[*]
Thomson Reuters
hella.hoffmann@thomsonreuters.com

Xu Chu
University of Waterloo
x4chu@uwaterloo.ca

ABSTRACT

With the increasing incentive of enterprises to ingest as much data as they can in what is commonly referred to as "data lakes", and with the recent development of multiple technologies to support this "load-first" paradigm, the new environment presents serious data management challenges. Among them, the assessment of data quality and cleaning large volumes of heterogeneous data sources become essential tasks in unveiling the value of big data.

The coveted use of unstructured and semi-structured data in large volumes makes current data cleaning tools (primarily designed for relational data) not directly adoptable.

We present CLAMS, a system to discover and enforce expressive integrity constraints from large amounts of lake data with very limited schema information (e.g., represented as RDF triples). This demonstration shows how CLAMS is able to discover the constraints and the schemas they are defined on simultaneously. CLAMS also introduces a scale-out solution to efficiently detect errors in the raw data. CLAMS interacts with human experts to both validate the discovered constraints and to suggest data repairs.

CLAMS has been deployed in a real large-scale enterprise data lake and was experimented with a real data set of 1.2 billion triples. It has been able to spot multiple obscure data inconsistencies and errors early in the data processing stack, providing huge value to the enterprise.

1. INTRODUCTION

At present, data is viewed as a major enterprise asset, and tools for analyzing big data are highly coveted. To handle the scale of available data, businesses have shifted to a "load-first" approach. This has lead to an upward trend of maintaining data in its native format and storing it in large repositories referred to as "data lakes."[1]

The effectiveness of data analysis depends on the quality of the underlying data. Current data quality techniques [5] rely on the existence of a global schema and use it to define various integrity constraints. In load-first schema-later environments, most data is available in a plethora of local schemas, schema-less RDF triple formats, and comma-separated-value files. Analytics are performed on top of application-specific schemas that are defined much later in the stack. Waiting until a global schema is defined means propagating large volumes of erroneous data through multiple data processing layers via complex Extract-Transform-Load [2] operations. This complicates repairing and tracking errors to their original sources [1]. On the other hand, attempting to clean data in the lake directly after ingesting it from the sources is complicated by the lack of schema information and the limited number of integrity constraints that can be defined without a schema.

Denial Constraints (DCs) [4] are a state-of-the-art method to express data quality rules over relational data [5]. The literature has proposed efficient discovery [3] and enforcement [4] algorithms for DCs. However, these algorithms cannot be directly applied on data lakes because of (*i*) the schema-less nature of lake data; and (*ii*) the limited scalability of these algorithms to large datasets.

We present CLAMS, an interactive system that enables business users to design and enforce complex constraints over large unstructured and semi-structured datasets. In particular, CLAMS operates on large-scale data processing frameworks (specifically, Apache Spark) to manipulate massive datasets. The features of CLAMS include:

- Providing a data ingestion module to transform unstructured and semi-structured data into RDF triples, which are then loaded into HDFS;
- Specifying *expressive integrity constraints* over graph data through (*i*) automatic discovery and ranking of plausible quality constraints which get verified by the user; and (*ii*) a friendly and interactive UI for creating and modifying constraints guided by data sampling and summarization;
- Enforcing the specified constraints through scalable, distributed algorithms to detect data inconsistencies;
- Surfacing a ranked list of possible errors along with the lineage of the violating data for user inspection.

Our solution brings the enterprise one step closer to extracting value from its dark (limited-access) data.

[*]Work done while at University of Waterloo

[1]http://www.gartner.com/newsroom/id/2809117

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2882903.2899391

2. CLAMS SYSTEM OVERVIEW

Figure 1 illustrates the system architecture of CLAMS. We give an overview of the different components, and then discuss the details of each component.

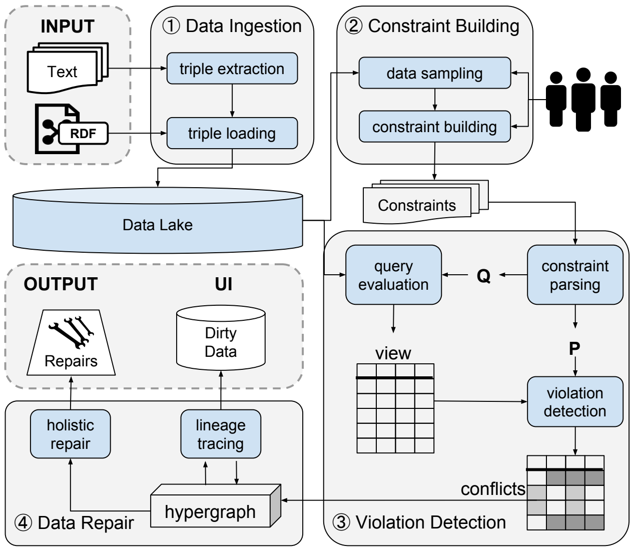


Figure 1: CLAMS System Architecture

- the *data ingestion* module ① loads datasets into HDFS, registers data sources for cleaning, and uses information extraction to produce semi-structured data (Section 2.1);
- the *constraint building* module ② offers an interactive UI to examine automatically discovered simple quality rules and guides the user in creating more complex ones (Section 2.2);
- the *violation detection* module ③ efficiently enforces a set of constraints over the registered data sources to detect data that does not conform to the specified constraints (Section 2.3);
- the *data repair* module ④ identifies the highly likely causes of errors and proposes solutions for cleaning them (Section 2.3).

2.1 Data Ingestion

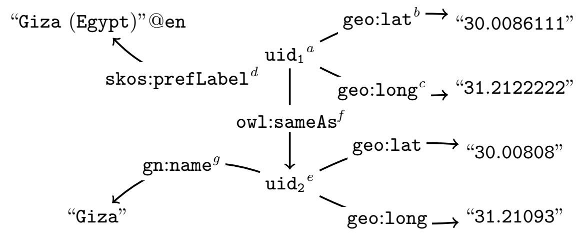
At present, CLAMS uses the RDF data model for storing heterogeneous data. RDF is a graph-based data model, where facts are represented as *triples* (directed labeled edges between labeled nodes). Figure 2 shows a sample RDF dataset in its graph representation, using data from the New York Times[2] and GeoNames[3] datasets.

Heterogeneous information can be easily represented in this format. Also, no global schema is necessary to understand the data due to its human friendly readability.

Our conversation with multiple companies confirmed that RDF is indeed a common format to represent data coming from heterogenous data sources. Information extraction tools (e.g., Open Calais[4] of Thomson Reuters and IBM's AlchemyAPI[5]) may assist in extracting RDF triples from unstructured text and linking the extracted data to popular ontologies (e.g., DBpedia).

CLAMS ingests RDF data into HDFS. If the data is in a different format, CLAMS uses information extraction tools to extract RDF triples before loading them. Users can register HDFS datasets as resources, expanding the catalog of data sources on which constraints can be applied.

[2]http://data.nytimes.com/
[3]http://www.geonames.org/
[4]http://www.opencalais.com/
[5]http://www.alchemyapi.com/



[a] http://data.nytimes.com/48075591700161033501
[b] http://www.w3.org/2003/01/geo/wgs84_pos#lat
[c] http://www.w3.org/2003/01/geo/wgs84_pos#long
[d] http://www.w3.org/2004/02/skos/core#prefLabel
[e] http://sws.geonames.org/360995/
[f] http://www.w3.org/2002/07/owl#sameAs
[g] http://www.geonames.org/ontology#name

Figure 2: Sample RDF Graph

Two identical locations must have the same latitude coordinates is represented as a CDC $c = (Q, \varphi)$, where:

```
Q = SELECT ?loc1 ?loc2 ?lat1 ?lat2
    WHERE { ?loc1 owl:sameAs ?loc2 .
            ?loc1 geo:lat ?lat1 .
            ?loc2 geo:lat ?lat2 }
```

$\varphi = \{\forall \mathtt{t} \in Q(\mathcal{D}),\ \neg(t.lat1 \neq t.lat2)\}$

Figure 3: Sample Quality Constraint

2.2 Constraint Building

Definition of Constraints. In CLAMS, we define a *conditional denial constraint (CDC)* c over data model $\mathcal{M}$ as a pair (Q, φ), where Q is a query in language $\mathcal{L}$, and φ is a denial constraint over the relation defined by the answer of Q over $\mathcal{D}$, denoted $Q(\mathcal{D})$. The query Q in a CDC defines a relational view over the unstructured input dataset. Then, the denial constraint φ over the view expresses the set of conditions that represent a forbidden (incorrect) pattern for (sets of) tuples in $Q(\mathcal{D})$. We say that a dataset $\mathcal{D} \in \mathcal{M}$ *satisfies* a CDC $c = (Q, \varphi)$, denoted $\mathcal{D} \models c$, iff the answer to $Q(\mathcal{D})$ satisfies φ, denoted $Q(\mathcal{D}) \models \varphi$.

We will demonstrate the application of CDCs over heterogeneous datasets using the RDF data model and its query language, SPARQL. Figure 3 shows a simple constraint example over the RDF graph in Figure 2.

Note that while the set of predicates is limited by the DC formalism, the query Q allows the definition of constraints beyond DCs. Take for instance a graph reachability constraint stating a conflict of interest, e.g., a bank employee cannot be in charge of the account of a first-degree relative. In order to express that constraint, the reachability predicates are pushed into the query. Therefore, the query will represent the forbidden pattern while φ is void, signalling that every query answer is a violation of the CDC.

Constraint Discovery. The separation of a CDC into query and DC significantly complicates the discovery process. The space of possible DCs for a given schema is already massive [3]. Performing such a search for all possible schemas is unfeasible. We therefore limit our search to basic graph pattern (i.e., conjunctive) queries. This limitation allows for a manageable scope of potential schemas, with many repeating sub-patterns. Furthermore, CDC discovery is optimized by intertwining the search for relation schemas with the search for DC predicates.

The automatic constraint discovery uses multiple techniques to detect potential schemas in the data and discover constraints that may apply on them. The schema detection spans from using schema languages such as RDFS and OWL, to graph summaries and entity anchoring. DC predicate detection is based on state-of-the-art techniques [3], extended with attribute type inference and on-the-fly schema detection. CLAMS generates friendly natural English descriptions for the discovered constraints.

Constraint Building. CLAMS also provides an interactive interface to manually build more complex constraints. The system guides the user in the process of creating constraints by providing statistics about common types and properties in the dataset and their co-occurrence to assist in constructing the constraint queries.

2.3 Violation Detection and Holistic Repair

A violation of a CDC c is defined as a minimal set of triples that cannot coexist, and removing any triple from that set will resolve the violation. For example, the following triples from Figure 2 violate the CDC c from Figure 3. The set of triples $\{t_1, t_2, t_3\}$ forms a *single* violation of c.

t_1 : uid$_1$ owl:sameAs uid$_2$.
t_2 : uid$_1$ geo:lat "30.0086111" .
t_3 : uid$_2$ geo:lat "30.00808" .

Once we have detected all violations, possibly from multiple constraints, a *violation hypergraph* G is built. Each node in the hypergraph represents a triple, and each hyperedge represents a violation. Figure 4 shows a violation hypergraph with three violations V_1, V_2, V_3, where $V_1 = \{t_1, t_2, t_3\}$, $V_2 = \{t_1, t_2, t_4, t_5\}$, and $V_3 = \{t_1, t_6\}$.

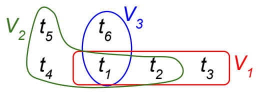
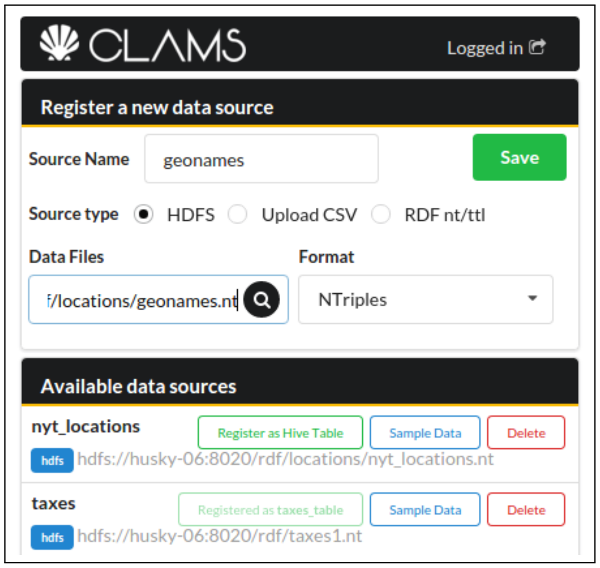


Figure 4: Example Hypergraph of three Violations

The hypergraph immediately reveals the number of violations a triple participates in, for example, t_1 is involved in three violations, while t_2 is involved in two violations. All nodes (triples) in the hypergraph are potential erroneous triples. We rank the potential erroneous triples by the number of violations each triple participates in. The intuition behind such ranking is that the more violations a triple is involved in, the more likely that triple is incorrect [4]. A visual interface is offered to the user, color coding the problematic triples by their participation in violations, as well as providing reasons to explain why these triples are erroneous [1]. The user gradually edits or remove triples from the graph until all the violations are solved. For example, triple t_1 in Figure 4 is first presented to the user. After examining the violations t_1 is involved in, the user might decide that t_1 is an erroneous triple and thus should be removed. Removing t_1 causes CLAMS to update the hypergraph. In this example, deleting t_1 causes all the hyperedges (i.e., violations)



Register a new data source

Source Name: geonames

Source type: ● HDFS ○ Upload CSV ○ RDF nt/ttl

Data Files: f/locations/geonames.nt

Format: NTriples

Available data sources

nyt_locations
hdfs hdfs://husky-06:8020/rdf/locations/nyt_locations.nt

taxes
hdfs hdfs://husky-06:8020/rdf/taxes1.nt

Figure 5: Data Source Registration

to be removed, eliminating the need for the user to further examine the rest of the triples.

2.4 Implementation Details

We designed and implemented the backend algorithms for constraint discovery and enforcement on the Apache Spark cluster computing framework to benefit from its efficient in-memory processing capabilities. Spark reads RDF data from HDFS, which represents our data lake, and distributes the algorithmic work among 10 worker nodes with a total of 832 GB RAM memory. We implemented a RESTful web services layer using Dropwizard[6] to communicate with Spark and HDFS. Dropwizard exposes our data quality algorithms as web services. When a request is received, we formulate and submit an appropriate Spark job to the cluster, and the results are written to HDFS. A user web application is built in JavaScript on top of Node.js[7] framework and it is responsible for providing an interactive UI to call the Dropwizard web services and manipulate catalog information.

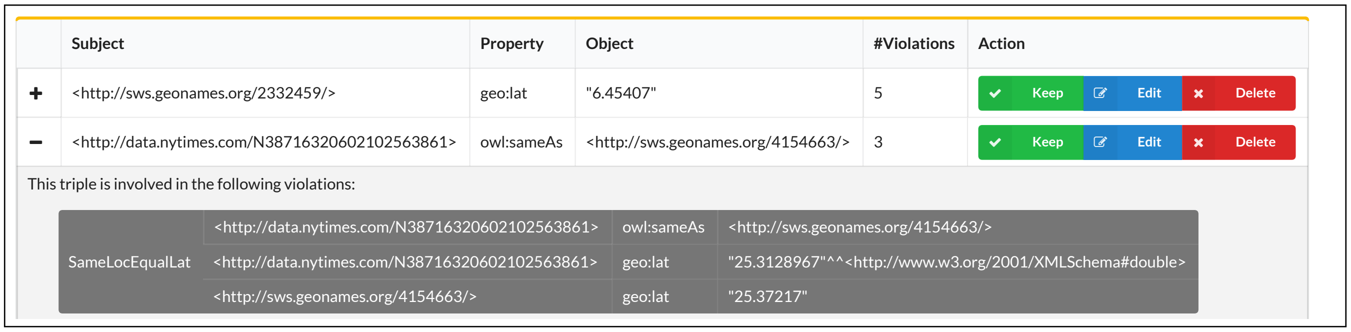
3. DEMONSTRATION SCENARIO

The demonstration's audience will be able to use CLAMS to enforce quality constraints on RDF data. Participants may assess automatically discovered constraints or manually build constraints with the help of CLAMS's data exploration tool. After defining the constraints, the constraint enforcement algorithms of CLAMS run to detect violations and suggest repairs. Users can investigate the detected violations and provide repair feedback to the system.

In this section, we present a demonstration scenario to show the functionality of CLAMS in one possible application to enforce data quality constraints on the data lake.

Data Ingestion and Source Registration. The first step in our demonstration is to register the data sources that we will operate on.

[6]http://www.dropwizard.io/
[7]https://nodejs.org

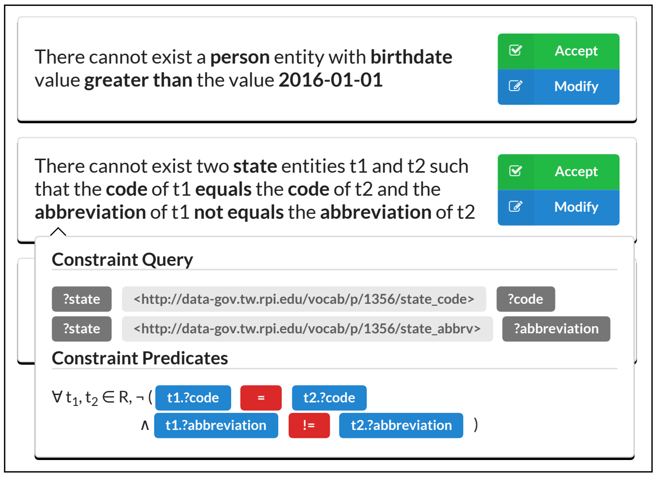


	Subject	Property	Object	#Violations	Action
+	<http://sws.geonames.org/2332459/>	geo:lat	"6.45407"	5	Keep Edit Delete
−	<http://data.nytimes.com/N38716320602102563861>	owl:sameAs	<http://sws.geonames.org/4154663/>	3	Keep Edit Delete

This triple is involved in the following violations:

SameLocEqualLat	<http://data.nytimes.com/N38716320602102563861>	owl:sameAs	<http://sws.geonames.org/4154663/>
	<http://data.nytimes.com/N38716320602102563861>	geo:lat	"25.3128967"^^<http://www.w3.org/2001/XMLSchema#double>
	<http://sws.geonames.org/4154663/>	geo:lat	"25.37217"

Figure 7: Detected Violations and User Feedback for Repair



There cannot exist a **person** entity with **birthdate** value **greater than** the value **2016-01-01**

There cannot exist two **state** entities t1 and t2 such that the **code** of t1 **equals** the **code** of t2 and the **abbreviation** of t1 **not equals** the **abbreviation** of t2

Constraint Query

?state	<http://data-gov.tw.rpi.edu/vocab/p/1356/state_code>	?code
?state	<http://data-gov.tw.rpi.edu/vocab/p/1356/state_abbrv>	?abbreviation

Constraint Predicates

$\forall t_1, t_2 \in R, \neg($ t1.?code = t2.?code $\wedge$ t1.?abbreviation != t2.?abbreviation $)$

Figure 6: Discovered Constraints and User Verification

- Users can register RDF data (Figure 5) that is already loaded on HDFS (large datasets), upload RDF data files or select extractors for other data formats.
- CLAMS shows the registered datasets and allows users to see samples of them.

Discovering and Building Constraints. The next step is to define the constraints that apply on the dataset. CLAMS supports two methods to create constraints.
Discovered Constraints. First, users can trigger the automatic constraint discovery from the underlying datasets as explained in Section 2.2.

- The discovered constraints are presented to the user with simple English translations (Figure 6).
- Users can see the details of a constraint (its query and predicates) by hovering over it, and can modify it or directly add it (accept) to the system catalog.

User-Defined Constraints. The second approach is to use the interactive constraint building tool of CLAMS to explore a sample of the data and define manual constraints.

- Users interact with data to build the constraint query.
- Users define the constraint predicates based on the variables in the query. The predicates can be defined on a single tuple or on pairs of tuples.

Constraint Enforcement. Given a set of defined constraints and a set of registered data sources, users can run our constraint enforcement algorithm to detect violations.

- Users select the constraints to enforce and the data sources to enfoce them on, then start the constriant enforcement process.
- CLAMS runs the enforcement Spark jobs on the cluster and returns the detected violations (Figure 7).
- The system displays the violating RDF triples, ranked by their accumulated violations (cf. Section 2.3).
- Upon clicking on a triple, all the violations that this triple is involved in are displayed to the user.
- The user decides whether to keep, edit, or delete a triple from the data. If deleted, CLAMS excludes that triple when performing further constraint enforcement.

4. CONCLUSION

We presented CLAMS—a system to discover and to enforce quality constraints on large RDF datasets that reside in data lakes. CLAMS uses a new integrity constraint formalism to capture both relational model-like constraints and more expressive quality rules based on graph patterns. We showed how CLAMS holistically combines the signals from diverse constraints spanning over multiple datasets and utilize user feedback to obtain accurate repairs.

5. ACKNOWLEDGEMENTS

This work is generously funded by Thomson Reuters. The company has also provided us with valuable data that was used to validate CLAMS and feedback about the implemented prototype.

6. REFERENCES

[1] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and Prescriptive Data Cleaning. In *SIGMOD*, 2014.

[2] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Rec.*, 26(1):65–74, Mar. 1997.

[3] X. Chu, I. F. Ilyas, and P. Papotti. Discovering Denial Constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, Aug. 2013.

[4] X. Chu, I. F. Ilyas, and P. Papotti. Holistic Data Cleaning: Put Violations Into Context. In *ICDE*, 2013.

[5] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.