Human-in-the-Loop Schema Inference for Massive JSON Datasets

Mohamed-Amine Baazizi
Sorbonne Université, LIP6 UMR 7606
baazizi@ia.lip6.fr

Clément Berti
Sorbonne Université
clement.berti.upmc@gmail.com

Dario Colazzo
Université Paris-Dauphine, PSL Research University
dario.colazzo@dauphine.fr

Giorgio Ghelli
Dipartimento di Informatica, Università di Pisa
ghelli@di.unipi.it

Carlo Sartiani
DIMIE, Università della Basilicata
carlo.sartiani@unibas.it

ABSTRACT

JSON established itself as a popular data format for representing data whose structure is irregular or unknown a priori. JSON collections are usually massive and schema-less. Inferring a schema describing the structure of these collections is crucial for formulating meaningful queries and for adopting schema-based optimizations.

In a recent work, we proposed a Map/Reduce schema inference approach that either infers a compact representation of the input collection or a precise description of every possible shape in the data. Since no level of precision is ideal, it is more appealing to give the analyst the freedom of choosing between different levels of precisions in an interactive fashion. In this paper we describe a schema inference system offering this important functionality.

1 INTRODUCTION

Borrowing flexibility from semistructured data models and simplicity from nested relational ones, JSON affirmed as a convenient and widely adopted data format for exchanging data between applications as well as for exporting data through Web API and/or public repositories. JSON datasets are usually retrieved from remote, uncontrolled sources, with partial, incomplete, or no schema information about the data. In these contexts, however, having a precise description of the structure of the data is of paramount importance, in order to design effective and efficient data processing pipelines. Schema inference, therefore, becomes a crucial operation enabling the formulation of meaningful queries and the adoption of well-known schema-based optimization techniques.

Several approaches and tools exist for inferring structural information from JSON data collections [13–15]. As pointed out in [10, 11], the common aspect of all these approaches is the extraction of some structural description with a precision that is fixed a priori, by the approach itself. While this methodology has the advantage of simplicity, it is in practice not satisfactory, since a JSON dataset can be rather (oftentimes highly) irregular in structure, and for this reason it can be typically described at different precision levels by a schema, while there exists no "best" precision level that can be fixed a priori. In general, one is interested in a description that is compact, easy to read even if it hides lots of details, typically in the first exploration steps, while in subsequent steps he/she is likely to be interested in a more precise, and therefore less succinct, schema description, where more details about the alternative shapes that can be found in the data are provided.

We believe that leaving the user the ability of tuning the level of precision of the inferred schema, by trying different possibilities and changing the level of details at different times, is an important feature, that existing techniques do not provide. With such a motivation in mind, in two recent works [9, 12], we devised, respectively, i) a Map/Reduce-based schema inference technique for massive JSON data that enables the user to choose, a priori, the level of precision of the inferred schema, and ii) a formal system which provides the user with mechanisms to interactively refine/expand the inferred schema, even locally, without the need of re-processing the data multiple times.

The goal of this demonstration is to showcase results and mechanisms provided by these two works, by means of an implementation of the parametric schema inference system [9] which is based on Spark and which interacts with a Web interface that the user can exploit to choose or submit a dataset of interest, and to play with the interactive schema inference process [12].

The user interacts with the system by choosing an existing, already analyzed, dataset, or by submitting a new one. The system initially returns to the user a succinct, but not very precise, schema, and the user then can explore it in order to decide where to get more precision, at several nesting levels: indeed, the user can choose to get a more detailed schema description at a given nesting level, while leaving the inner levels described in a more succinct fashion, hence at a lower degree of precision.

In the remainder of this article, Sections 2 and 3 introduce the parametric [9] and the interactive schema [12] inference techniques, while Section 4 details the architecture supporting our system and the demonstration scenario.

2 PARAMETRIC SCHEMA INFERENCE

The schema inference technique proposed in [9] is based on a Map/Reduce algorithm to ensure scalability. During the *map* phase, an input collection of JSON objects is processed by inferring a schema for each object in the collection. The *reduce* phase produces the final schema by invoking a commutative and associative function whose role is to *merge* the object schemas that are *equivalent*. Deciding whether two schemas are equivalent is a crucial aspect of our approach, as this allows one to choose between different precision levels. We rely on two main equivalence relations (kind equivalence and label equivalence), which we identified to be useful in practice, but our system, which is parametric, allows for using other equivalences defined by the user (see [9] for details).

When using the *kind* equivalence (K), every record type is equivalent to any other record type, and every array type is equivalent to any array type. Hence, this equivalence leads to merging all record types into a single one while indicating for each field whether it is optional or mandatory.

To illustrate, consider the following heterogeneous collection containing three JSON records and one array.

o_1 $\{a : 1, b : 2, d : \{e : 3, f : 4\}\}$
o_2 $\{a : 1, c : 2, d : \{g : 3, h : 4\}\}$
o_3 $\{a : 1, c : 2, d : \{e : 3, f : 4\}\}$
o_4 $[123, "abc", \{a : 10, b : 20\}]$

The *map* phase yields for each value a corresponding schema. Essentially, atomic values are mapped to their corresponding atomic types (numbers to Num, etc), while complex constructs are processed recursively. The potentially heterogeneous content of arrays is concisely represented using the union (+) operator.

$o_1 \rightarrow s_1 = \{a : \mathsf{Num}, b : \mathsf{Num}, d : \{e : \mathsf{Num}, f : \mathsf{Num}\}\}$
$o_2 \rightarrow s_2 = \{a : \mathsf{Num}, c : \mathsf{Num}, d : \{g : \mathsf{Num}, h : \mathsf{Num}\}\}$
$o_3 \rightarrow s_3 = \{a : \mathsf{Num}, c : \mathsf{Num}, d : \{e : \mathsf{Num}, f : \mathsf{Num}\}\}$
$o_4 \rightarrow s_4 = [\mathsf{Num} + \mathsf{Str} + \{a : \mathsf{Num}, b : \mathsf{Num}\}]$

During the *reduce* phase, equivalent types are merged based on the chosen equivalence relation. The K equivalence merges all record types and yields a union of a record and array type, as follows:

$$S_3 = \{\ a : \mathsf{Num}, b : \mathsf{Num}?, c : \mathsf{Num}?, d : \{e : \mathsf{Num}?, f : \mathsf{Num}?, g : \mathsf{Num}?, h : \mathsf{Num}?\}\ \} + [\ \mathsf{Num} + \mathsf{Str} + \{a : \mathsf{Num}, b : \mathsf{Num}\}]$$

The record type reports all fields appearing in the merged record types from the *map* phase, while indicating whether they are mandatory or optional (this latter fact being indicated by decorating the fields with ?). For instance, a is a mandatory field of type Num, while b, c, and d are optional fields of type Num; furthermore, d values are objects whose fields are all optional.

Notation 2.1 *In the following, when a union schema $s_1 + \ldots + s_n$ is inferred by means of an equivalence $\mathcal{E}$, we will use the prefix notation $+_{\mathcal{E}}(s_1, \ldots, s_n)$, so that the inferred schema indicates which equivalence has been used in order to decide what schemas to merge in the inference process. For readability, we omit the $+_{\mathcal{E}}$ prefix for atomic types when they appear as a singleton.*

So for instance, we note S_3 as

$$S_3 = +_K(\ \{\ a : \mathsf{Num}, b : \mathsf{Num}?, c : \mathsf{Num}?, d : +_K(\{e : \mathsf{Num}?, f : \mathsf{Num}?, g : \mathsf{Num}?, h : \mathsf{Num}?\})\ \}, [\ +_K(\mathsf{Num}, \mathsf{Str}, \{a : \mathsf{Num}, b : \mathsf{Num}\})]\ )$$

Concerning precision, it is worth observing that the above schema hides important correlation information like the fact that b and c never co-occur or the fact that fields e and f always occur together.

To derive a more precise schema, where records having different labels are kept separated, we use the *label* equivalence (L), according to which record types are equivalent only if they share the same top-level field labels. So, by means of the L equivalence only s_2 and s_3 are merged, thus obtaining:

$$S_4 = +_L(\ \{\ a : \mathsf{Num},\ b : \mathsf{Num},\ d : \{e : \mathsf{Num}, f : \mathsf{Num}\}\}, \{\ a : \mathsf{Num}, c : \mathsf{Num}, d : +_L(\{e : \mathsf{Num}, f : \mathsf{Num}\},\ \{g : \mathsf{Num}, h : \mathsf{Num}\}\ )\ \}, [\ +_L(\mathsf{Num}, \mathsf{Str}, \{a : \mathsf{Num}, b : \mathsf{Num}\}\ )]\ )$$

The resulting inferred schema S_4 gives now a very detailed description of the records in the data by sacrificing conciseness.

However, in general, schemas are much larger than those in the above example, and this could be a complication for an analyst who wants a precise description of a specific part of the dataset/schema (for instance one specific record type) without being overwhelmed by a too large schema describing all the rest. In order to overcome this limitation, we show in the next section how inferred schemas can be interactively manipulated by the analyst, by preserving soundness (schemas obtained in the interaction all describe the dataset at hand).

3 INTERACTIVE SCHEMA INFERENCE

The interactive schema inference is very useful when it allows for describing the same data with different levels of precision-succinctness, so that parts of greater interest to the user are described with the finest precision, while parts with lower interest are described in a succinct way. The interactive schema inference proposed in [12] goes into this direction, and to show its effectiveness we illustrate below a possible interaction that the user can perform on the schema inferred from a real-life dataset, crawled from the official NYTimes API [5] and consisting in metadata about articles of the newspaper. This dataset is interesting for our problem since it features many irregularities at several levels, and discovering these irregularities, using a traditional type inference mechanism, may simply become unpractical.

A simplified version of the K type inferred from this dataset is depicted in Figure 1. This schema focuses on the *byline* part which describes the authors of the articles. By examining this schema, the user realizes that almost all the fields are optional and hence, she/he may want to dig deeper to investigate for a potential correlation between the different fields.

```
+K({  docs :
    +K({  byline :
            +K({  contributor : Str?
                  organization : Str?
                  original : Str?
                  person : [+K({fn : Str?,
                                ln : Str?,
                                mn : Str?,
                                org : Str?})  ]
                })
            })
    })
```

Figure 1: The NYTimes K type.

The type resulting from refining the content of *byline* is depicted in Figure 2 and shows four possible situations which correspond to different combinations of the *contributor*, *organization*, and *original* fields, but, more interestingly, it reveals that the occurrence of *organization* implies that *person* has an empty array, while its absence coincides with the case where *person* contains an array with a record type. This observation can be explained by the fact that, when an article is written by an organization, the *person* field is not relevant and hence it contains an empty array and, conversely, when it is written by persons, the *organization* field is irrelevant and, hence, it just does not appear in the *byline* field.

Now that the user has gained some knowledge about the structure of the *byline* field, she/he may want to explore the *person*

```
+K({  docs :
      +K({  byline :
            +L({  contributor : Str
                  organization : Str
                  original : Str
                  person : [ ]
                  },
               {  contributor : Str
                  original : Str?
                  person : [+K({fn : Str?, ...
                              ..., org : Str?  }) ]
                  },
               {  organization : Str
                  original : Str
                  person : [ ]
                  },
               {  original : Str
                  person : [+K({fn : Str?, ...
                              ..., org : Str?  }) ]
                  })
            })
      })
```

Figure 2: The L refinement of the content of *byline*.

field which also contains records with optional fields. The user can recover the original type depicted in Figure 1, then expand the record inside the array resulting in the type that is partially depicted in Figure 3. This type shows different situations whose relevance is left to the discretion of the user.

```
[+L(
      {fn : Str, ln : Str, mn : Str, org : Str},
      {fn : Str, ln : Str, org : Str},
      {fn : Str, org : Str}
   ) ]
```

Figure 3: The L refinement of the content of *person*.

4 DEMONSTRATION OVERVIEW

The objective of this demonstration is to help the attendees understand the features and the goals of our schema inference system. To this aim, attendees will be able to:

(i) infer schemas for real-life JSON datasets according to K and L;

(ii) explore the inferred schemas and interactively fine-tune their precision;

(iii) get a concrete representation of the inferred schemas in JSON Schema.

We first describe the architecture of our system as well as the setup of the demo, and then illustrate the demonstration scenarios we propose.

4.1 System Architecture and Setup Details

Our system is based on a web application implemented following the client-server architecture depicted in Figure 4. The web client is used for loading the JSON collection and for performing the interactive schema inference, while the web server is dedicated to storing the collection and to inferring the initial schema. The storage is supported by HDFS while the computation is ensured by Spark, as presented in [9]. The web client and the remote inference engine communicate through a REST API implemented in Python 3 using the Flask [2] library. The API requests from the client are processed by an orchestrator that coordinates between the storage and the inference modules in the server side. This coordination is ensured by API calls using two open source libraries: webHDFS [8] for communicating with the HDFS storage system, and livy [4] for submitting jobs to the Spark engine.

Upon receiving the input collection in JSONLines format [3], through the client, the server will store the collection on the HDFS then infers the L schema, using the Spark engine. The L schema is then sent to the client and used for inferring the K schema. The *schema visualizer* displays the K schema and translates the user actions into corresponding schema operations that are processed by the *schema manager*. These two modules coordinate during all the interaction session to fulfill the user requests.

The web client is implemented in Typescript [7] using the Angular 6 platform [1], which offers many advantages like modularity and the clean separation between the content of a web page and the program modifying its content; the schema inference module of the server is implemented in Scala and is fully described in [9].

A lightweight system showcasing the core features of the full-system is available online at [6]. Differently from the full-system that will be demonstrated at the conference, the lightweight system performs schema inference on the client side and hence, it is limited to processing small size documents.

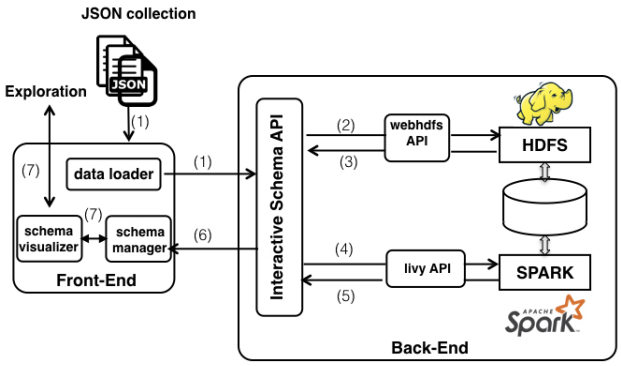

Figure 4: System architecture.

4.2 Demonstration Scenario

Our demonstration enables attendees to infer schemas for preloaded JSON datasets, to provide their own datasets, to explore the extracted schemas, and to fine-tune their precision, as well as to convert them in more popular schema languages like JSON Schema. The datasets we plan to use in our demo are described below.

The GitHub dataset corresponds to metadata generated upon pull requests issued by users willing to commit a new version of code. It takes 14GB of storage and contains 1 million JSON objects sharing the same top-level schema and only varying in their lower-level schema. All objects of this dataset consist exclusively of records nested up to four levels of nesting. Arrays are not used at all.

The Twitter dataset corresponds to metadata that are attached to the tweets shared by Twitter users. It takes 23 GB of storage and

contains nearly 10 million records corresponding, in most cases, to tweet entities. A tiny fraction of these records corresponds to a specific API call meant to delete tweets using their ids.

Finally, the NYTimes dataset, which was partly described in Section 3, contains approximately 1.2 million records and reaches the size of 22GB. Its records feature both nested records and arrays, and are nested up to 7 levels. Most of the fields in records are associated to text data which explains the large size of this dataset compared to the previous ones.

Schema Inference. The attendee will start the demo by connecting the web interface to the remote engine, and by selecting a pre-loaded dataset for schema inference; alternatively, the attendee will request the system to load an external dataset by providing a URI. Once selected a dataset, the attendee will choose an inference algorithm to be used by the remote engine. While the focus of this demonstration is on the interactive refinement of K-based schemas, the attendee will also have the opportunity to directly infer a schema according to the L-based approach as well as to get basic statistics about the data (average object size, AST height, etc). Once the inference system has completed the schema extraction process, it will upload the inferred schema to the web interface.

Schema Exploration. After the inference of the K schema, the attendee will likely start to explore this schema through the web interface which allows the user to change the precision level of the schema without any further intervention of the remote inference engine.

Schema Translation. After having explored the inferred schema and (possibly) fine-tuned its precision level, the attendee will be able to translate the schema in a JSON Schema representation; by relying on this feature, the attendee will be able to exploit the schema in any system or application supporting this language, without the need to manually rephrase the schema.

5 RELATED WORK

The problem of inferring structural information from JSON received some attention as reviewed in our recent paper [9], where we outlined the improvements of our approach over state-of-the-art approaches for JSON schema inference, while the topic of interactive JSON schema inference was only recently addressed [12].

In the context of XML, the only work about interactive inference we are aware of relies on user intervention for recognizing regular expressions that are similar enough to be merged and for deriving sophisticated XML schemas expressing complex constructs like inheritance and derivation [16].

REFERENCES

[1] Angular. Available at https://angular.io.
[2] Flask. https://www.flaskapi.org.
[3] JSONLines. http://jsonlines.org.
[4] livy REST API. Available at https://livy.incubator.apache.org.
[5] NYTimes API. https://developer.nytimes.com/.
[6] Online demo. http://132.227.204.195:4200/host.
[7] Typescript. Available at https://www.typescriptlang.org.
[8] webHDFS REST API. Available at https://hadoop.apache.org/.
[9] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Parametric schema inference for massive JSON datasets. *VLDB J.* 28, 4 (2019), 497–521. https://doi.org/10.1007/s00778-018-0532-7
[10] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Schemas And Types For JSON Data. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019.* 437–439. https://doi.org/10.5441/002/edbt.2019.39
[11] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Schemas and Types for JSON Data: From Theory to Practice. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.* 2060–2063. https://doi.org/10.1145/3299869.3314032
[12] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. A Type System for Interactive JSON Schema Inference (Extended Abstract). In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (LIPIcs)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.), Vol. 132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 101:1–101:13. https://doi.org/10.4230/LIPIcs.ICALP.2019.101
[13] Stefanie Scherzinger, Eduardo Cunha de Almeida, Thomas Cerqueus, Leandro Batista de Almeida, and Pedro Holanda. 2016. Finding and Fixing Type Mismatches in the Evolution of Object-NoSQL Mappings. In *Proceedings of the Workshops of the EDBT/ICDT 2016.* http://ceur-ws.org/Vol-1558/paper10.pdf
[14] Peter Schmidt. 2017. mongodb-schema. Available at `https://github.com/mongodb-js/mongodb-schema`.
[15] scrapinghub. 2015. Skinfer. Available at `https://github.com/scrapinghub/skinfer`.
[16] Julie Vyhnanovska and Irena Mlynkova. 2010. Interactive Inference of XML Schemas. In *Proceedings of the Fourth IEEE International Conference on Research Challenges in Information Science, RCIS 2010, Nice, France, May 19-21, 2010.* 191–202.