Inférence de Schémas pour Données JSON Massives

Mohamed-Amine Baazizi, Houssem Ben Ben Lahmar, Dario Colazzo, Giorgio Ghelli, Carlo Sartiani

Inférence de Schémas pour Données JSON Massives

Mohamed-Amine Baazizi
LIP6
Université Pierre et Marie Curie
Mohamed-Amine.Baazizi@lip6.fr

Houssem Ben Lahmar*
IPVS
University of Stuttgart
Houssem.Ben-Lahmar@ipvs.uni-stuttgart.de

Dario Colazzo
Université Paris-Dauphine,
PSL Research University,
CNRS, LAMSADE
75016 PARIS, FRANCE
dario.colazzo@dauphine.fr

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

Carlo Sartiani
DIMIE
Università della Basilicata
sartiani@gmail.com

Résumé.
Ces dernières années ont connu une large adoption de JSON en tant que format de représentation de données massives. Les données JSON sont généralement dépourvues de schémas puisqu'elles sont produites et gérées de manière flexible. Malgré cet avantage, l'absence de schéma présente de nombreux inconvénients : la correction des requêtes et des programmes ne peut être vérifiée de manière statique comme c'est la cas traditionnellement, les utilisateurs ne disposent d'aucun moyen le permettant de découvrir la structure des données sous-jacentes et, de manière plus générale, les techniques d'optimisations basées sur les schémas ne peuvent être appliquées.

Dans ce travail nous nous intéressons à l'inférence de schémas pour des données JSON massives. Notre première contribution consiste à proposer un langage de types pour JSON permettant de représenter la structure complexe des données analysées. Notre seconde contribution concerne le développement d'un algorithme d'inférence distribué et de son implantation dans Spark afin de garantir une exécution efficace sur des données volumineuses. Les résultats obtenus suite à une première étude expérimentale permettent de conclure que notre approche est satisfaisant en terme de temps d'exécution et de concision de schémas inférés.

1. INTRODUCTION

Les applications modernes sont amenées à analyser de larges volumes de données semi-strucutrées. Dans la plupart de ces applications et dans les applications communément appelées NoSQL, les données sont représentées en JSON, ce dernier étant un format particulièrement apprécié pour sa flexibilité et sa simplicité. Dans ce contexte, l'utilisation de schémas n'est pas obligatoire contrairement au cas relationnel. Ceci a pour avantage de permettre de déployer des applications de manière rapide sans avoir à se soucier du format des données échangées. Bien entendu, l'absence d'un schéma complique la plupart des opérations d'accès aux données à commencer par l'interrogation puisque l'utilisateur ne dispose d'aucune vue synthétique lui permettant de comprendre la structure sous-jacente des données. L'absence de schéma prive également les SGBD de tirer profit de techniques d'optimisation basées sur le schéma tels que la réécriture de requêtes ou la projection.

Dans ce travail nous proposons une approche permettant d'inférer, à partir d'une large collection de documents JSON, un schéma avec comme premier objectif de refléter la structure des données en entrée en tenant compte de l'imbrication des objets et du caractère obligatoire ou facultatif des attributs. Afin de permettre que ce schéma soit exploitable, notre technique d'inférence doit produire des schéma concis en évitant la redondance d'information. Enfin, afin de pouvoir manipuler de larges jeux de données, notre technique doit passer à l'échelle.

La technique que nous proposons s'appuie sur deux étapes qui se greffent dans un programme Map-Reduce de façon assez naturelle. Dans un premier temps, chaque document de la collection en entrée est analysé séparément dans le but d'en inférer un schéma. Cette étape se déroule de façon parallèle et est donc intégrée à la phase *Map*. Dans un second temps, les schémas produits dans la phase initiale sont fusionnés dans le but de produire le schéma global capturant la structure de tous les documents en entrée. Cette étape est intégrée à la phase *Reduce*.

La suite du document est consacrée à l'illustration de notre technique d'inférence. Faute d'espace, la présentation formelle est reléguée à l'article long [2].

2. APERÇU SUR L'INFÉRENCE DE SCHÉMAS

Afin d'illustrer notre technique, rappelons d'abord la syntaxe et la sémantique de JSON. Comme dans la plupart des modèles semi-structurés, JSON utilise des valeurs bases (null, booléens, entiers et chaînes de caractères) servant à stocker des informations atomiques et des valeurs complexes permettant de structurer les données soit dans un *object* ou

*Houssem Ben Lahmar has been partially supported by the project D03 of SFB/Transregio 161 http://www.trr161.de/interfak/forschergruppen/sfbtrr161.

BDA 2016, 15 au 18 Novembre, Poitiers, France.

une *séquence*. Un object est un ensemble d'attributs, chaque attribut associe à une clé une valeur donnée. Une séquence est une liste (ordonnée) de valeurs quelconques.

Afin d'illustrer la syntaxe de JSON considérons le document d_1 ci-dessous.

```
{"A": 123,  "B": "The ...",  "C": false,
    "D": ["abc", "cde", "fr12"] }
```

Ce document est construit à partir d'un objet délimité par des accolades. Cet objet est composé des clés "A", "B", "C" et "D" auxquelles sont associés respectivement un entier, une chaîne de caractères, un booléen et une séquence. La séquence associée à "D" est délimitée par des crochets et est composée de trois chaînes de caractères.

Il est important de noter qu'il n'existe pas de langage de schéma standard pour représenter un schéma JSON. Néanmoins le langage Json-Schema [1] semble être le plus adopté dans la pratique et a d'ailleurs fait l'objet d'une étude formelle dans [3] visant à révéler ses différentes caractéristiques. Grosso modo, Json-Schema permet de représenter la structure d'un document JSON en exhibant sa structure, abstrait les valeurs atomiques en des types pré-définis, précise si les attributs sont facultatifs ou obligatoires et indique la cardinalité des éléments des séquences. Il a été démontré dans [3] que le problème de validation d'une instance par rapport à un schéma de ce langage est PTIME-hard. La problématique d'inférence de schémas n'a pas été considéré dans la littérature.

Dans notre étude où l'intérêt est d'inférer un schéma succinct à partir d'une collection de documents, nous proposons un langage de schéma permettant, de la même manière que Json-Schema, d'abstraire les valeurs atomiques vers des types atomiques, de représenter la structure des données en entrée et de savoir pour chaque attribut s'il est facultatif ou obligatoire. Le schéma s_1 correspondant au document d_1 est donc

```
{"A": Number,  "B": String,  "C": Boolean,
      "D": [String, String, String] }
```

Rappelons que l'un des objectifs de notre approche est de construire des schémas succincts. Ceci passe par l'encodage compact du contenu des séquences en introduisant l'étoile de Kleene. Ainsi, il sera possible de ne renseigner que les types distincts qui apparaissent dans une séquence. Dans notre exemple, le schéma s_1 devient

```
{"A": Number,  "B": String,  "C": Boolean,
      "D": [String*] }
```

Nous allons maintenant nous intéresser à l'opération de fusion qui prend en entrée deux schémas s et s' et doit produire un schéma capturant toutes les caractéristiques de s et s' tout en étant compact. Intuitivement, cette opération consiste à combiner les parties de s et s' qui sont identiques et à retourner les parties restantes telles quelles. Afin d'illustrer cette opération considérons le schéma s_2 ci-dessous qui sera fusionné avec s_1 décrit précédemment.

```
{"A": Number,  "B": Number,  "C": Boolean,
      "D": [Number*] }
```

Il est aisé de constater que les attributs identifiés par "A", "C" sont identiques dans les deux schémas et que, par conséquent, ils seront fusionnés dans le résultat. Les attributs identifiés par "B" et "D" sont, quant à eux, différents et devront subir un traitement particulier permettant de garder l'information sur la variation de leur structure dans s_1 et s_2. Pour ce faire, nous utilisons l'opérateur d'union (+) qui indique toutes les possibilités rencontrées lors de la fusion. Le résultat de la fusion de s_1 et s_2 est donc

```
{"A": Number,  "B": String+Number,  "C":
    Boolean,  "D": [(String+Number)*] }
```

Il est important de noter que le schéma retourné est un super-type des schémas s_1 et s_2 en entrée. Il est aussi important de souligner que pour permettre l'exécution de la fusion en Map-Reduce, la fusion est commutative et associative.

3. CONCLUSIONS ET PERSPECTIVES

L'approche décrite dans cet article est une première étape vers le développement d'une technique d'inférence de schémas pour une large collection de documents JSON. Le but principal est de fournir une vue succincte des données sous-jacentes afin de faciliter leur manipulation (stockage, interrogation, transformation). Bien que relativement simple, le langage de schéma utilisé permet de refléter les variations structurelles des données tout en offrant une vue succincte de celles-ci. Une étude expérimentale (consultable en [2]) a pu démontré que l'approche proposée remplit les objectifs fixés en amont : la concision et le passage à l'échelle. En guise de perspective, nous étudions la possibilité d'enrichir le langage de schéma d'information sur la cardinalité des données afin de mieux cerner les caractéristiques statistiques des données. Nous envisageons également d'étudier le compromis entre concision et précision des schémas produits en nous appuyant sur des cas d'études réels.

4. REFERENCES

[1] The json schema language. `http://json-schema.org/`.

[2] M.-A. Baazizi, H. B. Lahmar, D. Colazzo, G. Ghelli, and C. Sartiano. Schema inference for massive json datasets. *EDBT*, 2017 (to appear).

[3] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoc. Foundations of JSON schema. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 263–273. ACM, 2016.