DATABRICKS

Easy JSON Data Manipulation in Spark

Yin Huai – Spark Summit 2014

About me

PhD Student at The Ohio State University

Research

- Previous work includes studies on file formats (e.g. RCFile) and query optimization (Hive Correlation Optimizer)
- Interested in distributed systems, database systems, and storage systems

Open source

- Hive (committer)
- Spark SQL (current focus)

Research intern at Databricks

Prevalence of JSON

Simple, compact and easy to read

Flexible on the schema

- Every JSON object is self-describing

De facto data-interchange format among web services

- e.g. Facebook and Twitter APIs

Heavily used in mobile and web application development

- Large volume of JSON datasets

The flexibility of JSON makes it **easy to generate** JSON datasets.

However, the flexibility of JSON makes it **hard to analyze** JSON datasets.

Let’s see a random selected tweet…

{"filter_level":"medium","retweeted_status":{"contributors":null,"text":"【特別警報について】「特別警報が発表されるまでは大丈夫」ということではありません。特別警報の基準以下の雨でも被害が発生する場合があります。本市は今後も、河川氾濫等の恐れがある時は警報段階でも避難勧告等を発表していきます。注意報、警報の段階から本市やテレビ等の情報に注意して下さい。","geo":null,"retweeted":false,"in_reply_to_screen_name":null,"truncated":false,"lang":"ja","entities":{"symbols":[],"urls":[],"hashtags":[],"user_mentions":[]},"in_reply_to_status_id_str":null,"id":380226410293387264,"source":"web","in_reply_to_user_id_str":null,"favorited":false,"in_reply_to_status_id":null,"retweet_count":26,"created_at":"Wed Sep 18 07:06:55 +0000 2013","in_reply_to_user_id":null,"favorite_count":4,"id_str":"380226410293387264","place":null,"user":{"location":"神奈川県横浜市中区港町1丁目1番地","default_profile":true,"profile_background_tile":false,"statuses_count":248,"lang":"ja","profile_link_color":"0084B4","id":264104099,"following":null,"protected":false,"favourites_count":0,"profile_text_color":"333333","description":"横浜市役所の公式アカウントです。次の情報を発信します。\r\n「市全域、もしくは複数区にまたがる広域的な災害が予測される場合の避難等に関する情報。災害対策本部体制下における災害等に関する情報」 返信やリツイート、フォローは行いませんので、御了承ください。\r\nhttp://t.co/PJAdGCkaCI","verified":true,"contributors_enabled":false,"profile_sidebar_border_color":"C0DEED","name":"横浜市総務局危機管理室","profile_background_color":"C0DEED","created_at":"Fri Mar 11 10:07:23 +0000 2011","default_profile_image":false,"followers_count":43970,"profile_image_url_https":"https://si0.twimg.com/profile_images/1277621196/Twitter___48px_normal.gif","geo_enabled":false,"profile_background_image_url":"http://abs.twimg.com/images/themes/theme1/bg.png","profile_background_image_url_https":"https://abs.twimg.com/images/themes/theme1/bg.png","follow_request_sent":null,"url":"http://www.city.yokohama.lg.jp/","utc_offset":32400,"time_zone":"Tokyo","notifications":null,"profile_use_background_image":true,"friends_count":0,"profile_sidebar_fill_color":"DDEEF6","screen_name":"yokohama_saigai","id_str":"264104099","profile_image_url":"http://a0.twimg.com/profile_images/1277621196/Twitter___48px_normal.gif","listed_count":2463,"is_translator":false},"coordinates":null},"contributors":null,"text":"RT @yokohama_saigai: 【特別警報について】「特別警報が発表されるまでは大丈夫」ということではありません。特別警報の基準以下の雨でも被害が発生する場合があります。本市は今後も、河川氾濫等の恐れがある時は警報段階でも避難勧告等を発表していきます。注意報、警報の段\u2026","geo":null,"retweeted":false,"in_reply_to_screen_name":null,"truncated":false,"lang":"ja","entities":{"symbols":[],"urls":[],"hashtags":[],"user_mentions":[{"id":264104099,"name":"横浜市総務局危機管理室","indices":[3,19],"screen_name":"yokohama_saigai","id_str":"264104099"}]},"in_reply_to_status_id_str":null,"id":380232351646633984,"source":"<a href=\"http://twicca.r246.jp/\" rel=\"nofollow\">twicca<\/a>","in_reply_to_user_id_str":null,"favorited":false,"in_reply_to_status_id":null,"retweet_count":0,"created_at":"Wed Sep 18 07:30:31 +0000 2013","in_reply_to_user_id":null,"favorite_count":0,"id_str":"380232351646633984","place":null,"user":{"location":"概ね横浜","default_profile":false,"profile_background_tile":false,"statuses_count":13472,"lang":"ja","profile_link_color":"0099B9","id":357550930,"following":null,"protected":false,"favourites_count":1,"profile_text_color":"3C3940","description":"横浜在住の石川県人。妻と息子の4人暮らし。そろそろ石川に戻れないか画策中だが\u2026","verified":false,"contributors_enabled":false,"profile_sidebar_border_color":"5ED4DC","name":"sugila","profile_background_color":"0099B9","created_at":"Thu Aug 18 15:00:59 +0000 2011","default_profile_image":false,"followers_count":106,"profile_image_url_https":"https://si0.twimg.com/profile_images/1623184922/tw_pro3_normal.jpg","geo_enabled":false,"profile_background_image_url":"http://a0.twimg.com/profile_background_images/364769005/e0143a40.jpg","profile_background_image_url_https":"https://si0.twimg.com/profile_background_images/364769005/e0143a40.jpg","follow_request_sent":null,"url":null,"utc_offset":32400,"time_zone":"Tokyo","notifications":null,"profile_use_background_image":true,"friends_count":87,"profile_sidebar_fill_color":"95E8EC","screen_name":"tw_sugila","id_str":"357550930","profile_image_url":"http://a0.twimg.com/profile_images/1623184922/tw_pro3_normal.jpg","listed_count":2,"is_translator":false},"coordinates":null}

Difficulties of analyzing JSON datasets

```
{"field1":1}
{"field1":"row21",
 "field2":"row22"}
```

Difficulty of defining a schema

- Complex structures
- Non-uniform schemas
- Multi-type fields

Difficulty on maintaining the schema

- Frequent changes of the schema (e.g. applications have been evolved)

Difficulty on accessing fields in a JSON dataset

- Lots of nested structures
- Complex structures

Users demand an easy way to process JSON datasets

Existing approaches

ETL

- Pro: Easy to access fields
- Cons: (1) Defining and maintaining schemas and (2) ETL process can take a long time

Custom JSON SerDes

- Pros: (1) Can work on fresh data and (2) Easy to access fields
- Con: Defining schemas

Storing JSON objects in a LOB column

- Pros: (1) Can work on fresh data and (2) Schema on read
- Con: Lots of UDFs needed in a query

An example of using UDFs

```
{
 "name":"Yin",
 "age":null,
 "address":
 {
  "city":"Columbus",
  "state":"Ohio"
 }
}
```

Goal

```
SELECT
  name, age,
  address.city, address.state
FROM jsonTable
```

With UDFs

```
SELECT
  v1.name, v1.age, v2.city, v2.state
FROM jsonTable jt
  LATERAL VIEW json_tuple(
  jt.json, 'name', 'age', 'address')
v1 as name, age, address
  LATERAL VIEW json_tuple(
  v1.address, 'city', 'state') v2 as
city, state;
```

JSON support in Spark

Demands and what we provide

Demands	JSON support in Spark
Work with fresh data	No mandatory ETL process
No need to define the schema for a JSON dataset	The schema is automatically inferred
Easy to access fields in complex structures	No need to use UDFs and easy to write queries

Demo

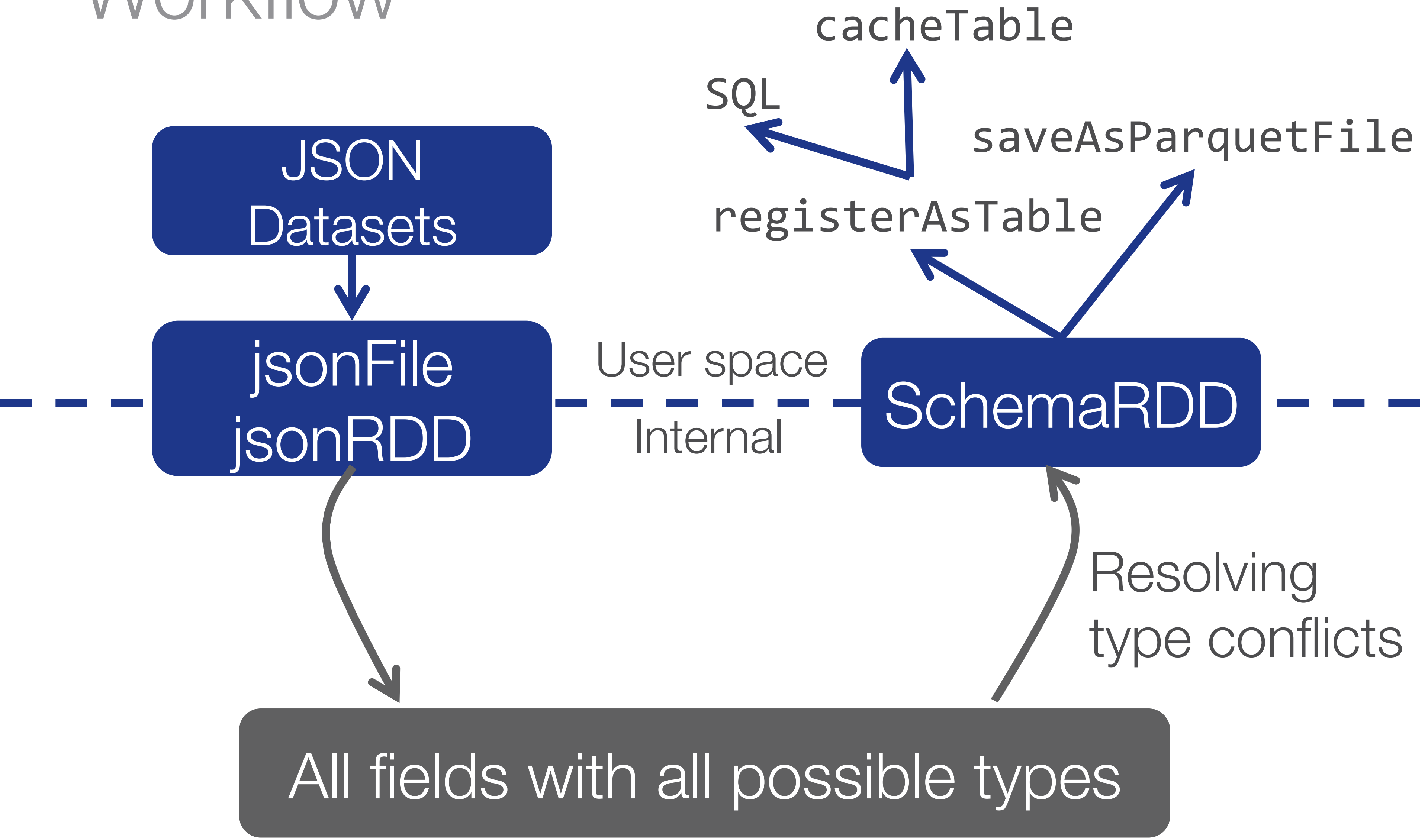
Getting started

Step 1:
1 line of code to load the dataset (the schema is automatically inferred)

Step 2:
1 line of code to register the dataset as a table

Start to process the dataset:
Write your queries in a natural way without using UDFs

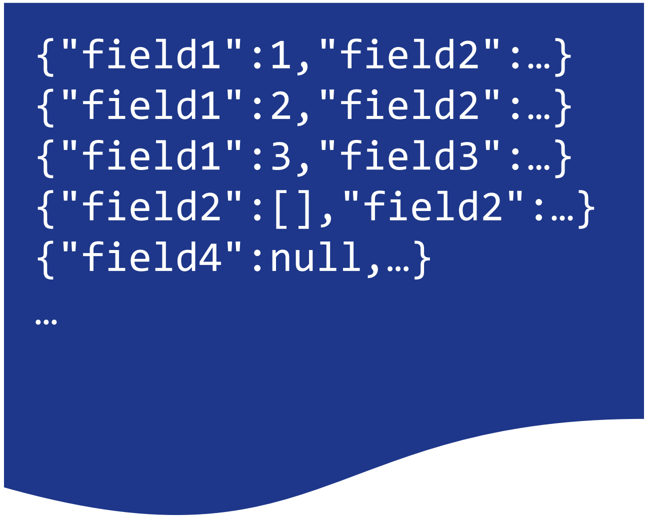
Workflow



DATABRICKS

Interfaces

data.json (text file)

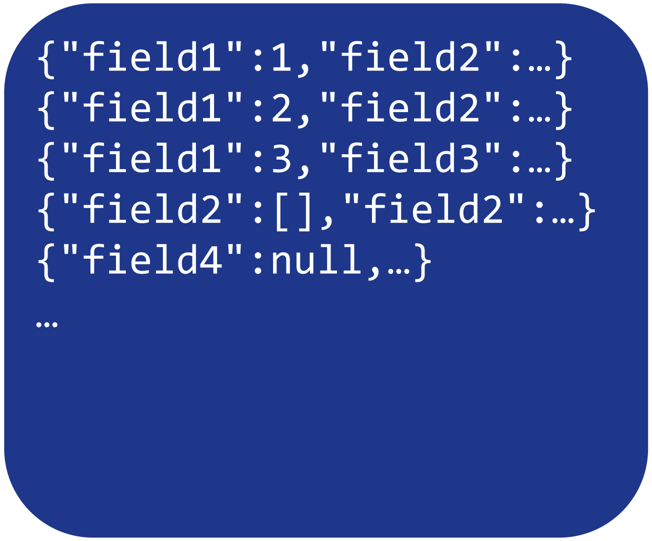


```
{"field1":1,"field2":…}
{"field1":2,"field2":…}
{"field1":3,"field3":…}
{"field2":[],"field2":…}
{"field4":null,…}
…
```

`sqlContext.jsonFile("data.json")`

One JSON object per line

data:RDD[String]

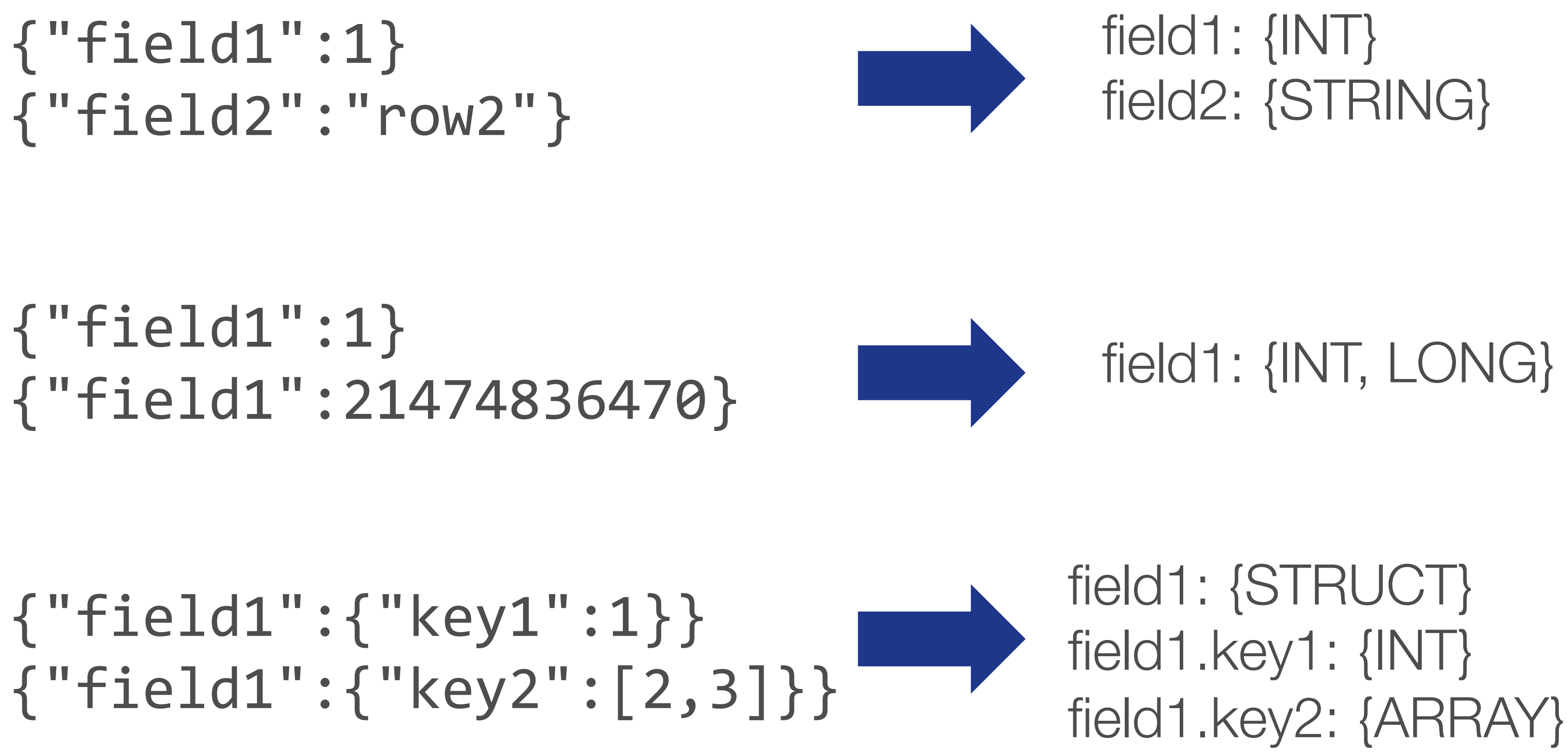


```
{"field1":1,"field2":…}
{"field1":2,"field2":…}
{"field1":3,"field3":…}
{"field2":[],"field2":…}
{"field4":null,…}
…
```

`sqlContext.jsonRDD(data)`

One JSON object per record

Gathering all fields with all possible types



```
{"field1":1}
{"field2":"row2"}
```

field1: {INT}
field2: {STRING}

```
{"field1":1}
{"field1":21474836470}
```

field1: {INT, LONG}

```
{"field1":{"key1":1}}
{"field1":{"key2":[2,3]}}
```

field1: {STRUCT}
field1.key1: {INT}
field1.key2: {ARRAY}

Resolving type conflicts (primitive types)

Widening the type: Conflicts between two numeric types

- NULL => INT => LONG => DOUBLE => DECIMAL

Downcasting to string

- Conflicts between string type and numeric types
- Conflicts between string type and boolean type
- Conflicts between boolean type and numeric types

Resolving type conflicts (primitive types)

```
{
 "name": "Alice",
 "age": 25
}
{
 "name": "Bob",
 "age": "50s"
}
```

name: {STRING}
age: {INT, STRING}

name: STRING
age: STRING

```
SELECT name, age
FROM table
WHERE age > 20
```

Values of `age` are promoted to numeric values; null when we cannot promote

Resolving type conflicts (complex types)

```
{
 "name": "Alice",
 "address": "somewhere"
}
{
 "name": "Bob",
 "address":
   {
    "city": "Columbus",
    "state": "Ohio"
   }
}
```

Conflicts involving complex types
=> Downcasting to string

```
SELECT address
FROM table
```

```
Somewhere
{"city": "Columbus", "state": "Ohio"}
```

Future work

Easily handling corrupted data

JSON column

SQL DDL commands for defining JSON data sources

Support for new semi-structured data formats such as CSV files

APIs for manipulating data types and schemas

DATABRICKS

Thank You!