

# Traitement des données (semi-) structurées dans Spark

Master DAC – Bases de Données Large Echelle  
Mohamed-Amine Baazizi  
[baazizi@ia.lip6.fr](mailto:baazizi@ia.lip6.fr)  
Octobre 2017

## Introduction

- Les 3V (Volume, Variety, Velocity)
  - **Volume** : plusieurs tera-octets
  - **Variety** : plusieurs types de données
  - **Velocity** : données en flux, réponses quasi temps réel
- Traiter les 3V
  - **Volume** : distribuer les traitements
  - **Variety** : solution ad-hoc (mouvance NoSQL)
  - **Velocity** : traiter en continu et de manière distribuée

2

## Data Variety dans Spark

- Efficacité** ↑  
**Flexibilité** ↓
- Plateforme unifiée pour différents formats
    - Données structurées :
      - Avro, Parquet, ORC, mysql
    - Données semi-structurées :
      - JSON
    - Données faiblement structurées :
      - Texte, CSV
- Compromis à trouver entre efficacité et flexibilité!



3

## RDD et données structurées

- Constat sur l'utilisation des RDD
  - Pas d'exploitation du schéma par défaut
    - code peu lisible, programmation fastidieuse
  - Lorsque structure homogène, encapsuler chaque n-uplet dans un objet reflétant la structure
    - Performances dégradées (sérialisation d'objets, GC)
  - Absence d'optimisation logique (comme dans les SGBD)
- Pallier aux limites des RDD : Dataset
  - Utiliser les schéma pour exprimer requêtes (à la SQL)
  - meilleure organisation des données : performances ++

4

# RDD et données structurées

Films(MovieID,Title,Genres)  
Notes(UserID,MovieID,Rating,Timestamp)  
Users(UserID,Gender,Age,Occupation,Zip-code)

Requête : Le titre des films noté plus de 10 par des utilisateur de moins de 25?

5

# Dataset

- **Description**
  - Collection d'objets ayant un schéma homogène
  - Manipulées avec opérations *fonctionnelles* (ex. map) ou *relationnelles* (ex. join)
  - Distinction entre actions et transformations comme RDD
  - Possibilité d'optimisation logique (comme pour SQL)
- **Création**
  - Définir une *case class* avec schéma cible
  - Parcourir et encoder les données en utilisant cette classe

6

# RDD vs Dataset

```
val films =  
sc.textFile().map(_.split(",")).map(...)  
films.filter(x=>x._1==2)
```

1	Toy Story (1995)	Animation Children
2	Jumanji (1995)	Adventure Children
..		

```
case Class Film(MovieID:Str, Title:Str, Genres:Str)  
val films = sc.textFile().map(_.split(",")).map(...)  
films.filter(x=>x.MovieID==2)
```

<b>Film</b> (1, Toy Story (1995), Animation Children)
<b>Film</b> (2,Jumanji (1995),Adventure Children)
...

RDD

```
val films = spark.read.format("csv"). ...  
films.filter("MovieID=2")
```

MovieID	Title	Genres
1	Toy Story (1995)	Animation Children
2	Jumanji (1995)	Adventure Children

Dataset

7

# Création d'un Dataset

```
MovieID,Title,Genres  
1,Toy Story (1995),Animation|Children...  
2,Jumanji (1995),Adventure|Children...  
3,Grumpier Old Men (1995),Comedy....  
...
```

**movies.csv**

```
scala> case class Movie(MovieID:String,Title:String,Genres:String)
```

```
scala> val films = spark.read.format("csv").  
    option("header",true).  
    load(path+ "movies.csv").as[Movie]
```

```
films: org.apache.spark.sql.Dataset[Movie] = [MovieID: string, Title: string ... 1  
more field]
```

8

## Quelques opérations Dataset

- Actions
  - count
  - describe
  - reduce
  - show
- Transformations
  - distinct
  - except
  - filter
  - flatMap
  - groupByKey
  - map
  - orderBy
- agg
- groupBy
- join
- select
- rdd
- dtypes
- printSchema

9

## Dataset par l'exemple

- Actions et fonctions de base

```
scala> films.show
```

MovieID	Title	Genres
1	Toy Story (1995)	Animation Childre...
2	Jumanji (1995)	Adventure Childre...
3	Grumpier Old Men ...	Comedy Romance
4	Waiting to Exhale...	Comedy Drama
5	Father of the Bri...	Comedy

```
scala> films.printSchema
```

```
root
|-- MovieID: string (nullable = true)
|-- Title: string (nullable = true)
|-- Genres: string (nullable = true)
```

10

## Dataset par l'exemple

- Transformations

```
scala> films.map(x=>x.Genres.split("\\|")).show
```

value
[Animation, Childre...]
[Adventure, Childre...]
[Comedy, Romance]
[Comedy, Drama]

```
scala> films.orderBy("Title").show
```

MovieID	Title	Genres
5	Father of the Bri...	Comedy
10	GoldenEye (1995)	Action Adventure...
3	Grumpier Old Men ...	Comedy Romance
6	Heat (1995)	Action Crime Thri...
2	Jumanji (1995)	Adventure Childre...
7	Sabrina (1995)	Comedy Romance
9	Sudden Death (1995)	Action

11

## Dataset par l'exemple

- Agrégation simple – agrégation avec groupement

```
scala> case class Note(userID: Int, MovieID: Int, Rating: Int, Timestamp: Int)

scala> val notes = sc.textFile(path+"sub-ratings.csv").map(x=>x.split(",")).
  map(x=>Note(x(0).toInt, x(1).toInt, x(2).toInt, x(3).toInt)).toDS

scala> notes.agg(min("Rating"), max("Rating"), avg("Rating"))

scala> notes.describe("Rating").show //montre count, stddev en plus

scala> notes.groupBy("MovieID").agg(count("*")).sort("count(1)").show
```

12

# Dataset

## Sélection

```
scala> films.filter("MovieID=1")
```

```
scala> films.filter("MovieID=1 or Title='Toy Story (1995)'")
```

## Projection

```
scala> films.select("MovieID", "Genres")
```

```
scala> films("MovieID") // une colonne à la fois
```

## Equi-jointure

```
scala> films.join(notes, "MovieID")
```

13

# Dataset vs RDD

- Les Dataset facilitent l'interrogation à la SQL
  - Ex. pour exprimer une jointure, il suffit d'expliquer le(s) attribut(s) de jointure seulement vs partitionner sur les(s) attributs(s) de jointure si RDD
  - Possibilité d'exprimer des requêtes SQL (consulter doc)

Films(MovieID, Title, Genres)

Notes(UserID, MovieID, Rating, Timestamp)

Users(UserID, Gender, Age, Occupation, Zip-code)

Requête : Le titre des films noté plus de 10 par des utilisateur de moins de 25?

14

# Présentation du mini-projet

- Dataset Yago :
  - Extrait de Yago (base de connaissance extraite du web)
  - triplets (sujet, propriétés, objet)

```
<A._Latham_Staples> <isCitizenOf> <United_States>
<A._Latham_Staples> <livesIn> <California>
<A._Latham_Staples> <livesIn> <La_Jolla>
<A._Latham_Staples> <wasBornIn> <Fort_Worth,_Texas>
```

....

- Familiarisation en TME
- Travail demandé
  - Statistiques sur le dataset
  - Liberté d'utiliser API RDD ou Dataset
  - Bonus. Faire de même pour DBPedia

15

# Data Variety dans Spark

- Plateforme unifiée pour différents formats
  - Données structurées :
    - Avro, Parquet, ORC, mysql
  - Données semi-structurées :
    - JSON
  - Données faiblement structurées :
    - Texte, CSV
- Compromis à trouver entre efficacité et flexibilité!

↑ Efficacité  
Flexibilité ↓



16

# Traitement de JSON en Spark

- Modèle JSON
  - Valeur atomique : int, string, boolean, null
  - Record : { 'att':val, ... } – Array : [val, ...]
- Utiliser Dataset pour :
  - Extraire la structure arborescente à partir d'une collection d'objets JSON
  - Interrogation (notation pointée pour traverser l'arborescence)

17

# Exemple

```
{
  "_id": "57cea1d1",
  "abstract": null,
  "blog": [],
  "byline": {
    "original": "By MELENA RYZIK",
    "person": [
      {
        "firstname": "Melena",
        "lastname": "RYZIK",
        "organization": ""
      }
    ]
  }
}
```

```
{ "entities": {
  "hashtags": [
    {
      "indices": [
        95,
        118
      ],
      "text": "Infrastruk"
    }
  ]
}
```

small.json

18

# Dataset et JSON

```
scala> val jcoll = spark.read.json(path+"small.json")
```

```
scala> jcoll.printSchema
```

```
root
|-- _id: string (nullable = true)
|-- abstract: string (nullable = true)
|-- blog: array (nullable = true)
|   |-- element: string (containsNull = true)
|-- byline: struct (nullable = true)
|   |-- original: string (nullable = true)
|   |-- person: array (nullable = true)
|       |-- element: struct (containsNull = true)
|           |-- firstname: string (nullable = true)
|           |-- lastname: string (nullable = true)
|           |-- organization: string (nullable = true)
```

```
scala> jcoll.count //retourne 3
```

19

# Dataset et JSON

- Problèmes
  - Structure extraite imprécise : impossible de distinguer des champs optionnels ou la variation des types
  - Impact sur la cohérence des requêtes

```
scala> jcoll.select("_id")
```

```
res116: org.apache.spark.sql.DataFrame = [_id: string]
```

```
scala> jcoll.select("entities").count
```

```
//retourne 3 alors qu'il en existe qu'un seul!
```

```
+-----+
| entities |
+-----+
| null |
| null |
| WrappedArray([Wr... |
```

20