# Data Engineering

# TP2 :Introduction to Spark and RDDs

Diplôme National d'Ingénieur en Informatique

*Spécialité :*

**Génie Logiciel**

*Réalisée par:*

**Oussama Ben Slama**

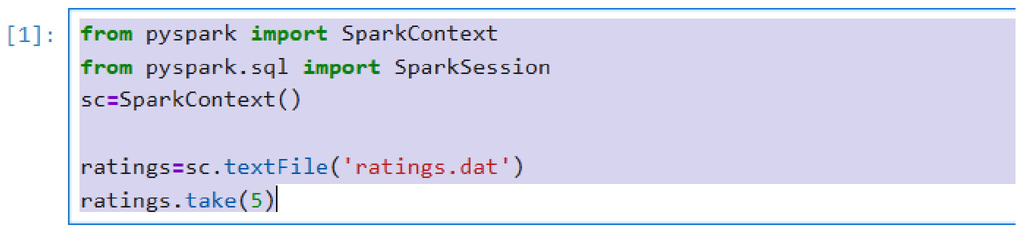
Année Universitaire 2024/2025

# Chapter 1

# Spark RDD

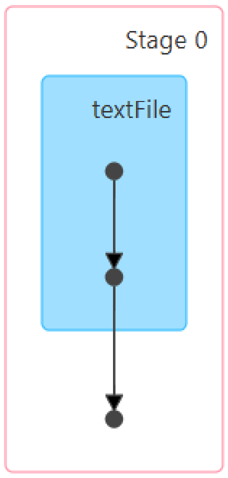
## 1.1 Examples

Reading the ratings.dat file into an RDD and displaying the first 5 lines.

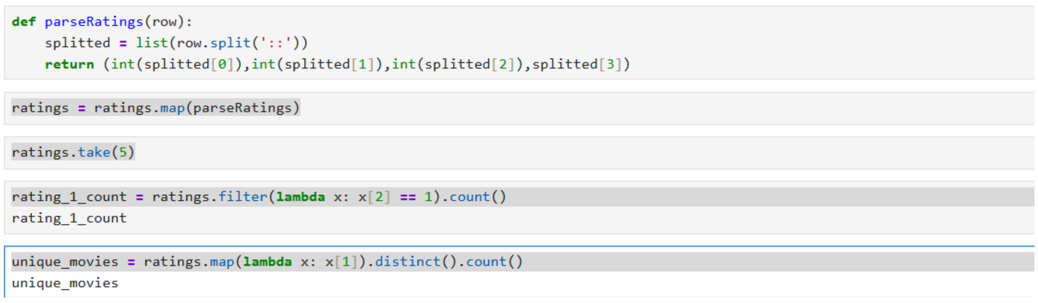


```
[1]: from pyspark import SparkContext
     from pyspark.sql import SparkSession
     sc=SparkContext()

     ratings=sc.textFile('ratings.dat')
     ratings.take(5)
```

DAG Visualization :



Parsing the ratings data using parseRatings function.
Counting how many ratings have a score of 1.
Counting the number of unique movies rated.



```
def parseRatings(row):
    splitted = list(row.split('::'))
    return (int(splitted[0]),int(splitted[1]),int(splitted[2]),splitted[3])
```

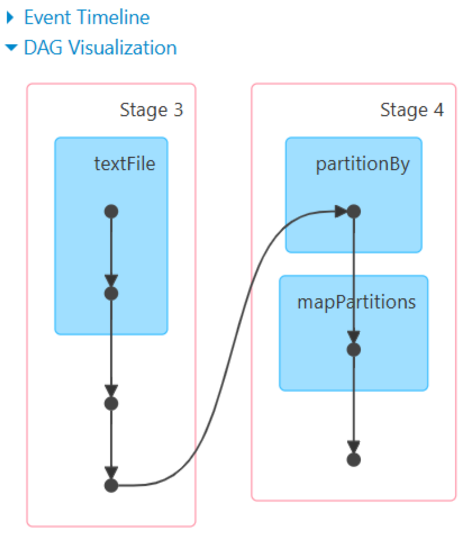
```
ratings = ratings.map(parseRatings)
```

```
ratings.take(5)
```

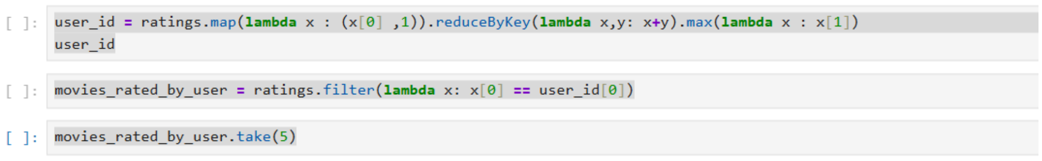
```
rating_1_count = ratings.filter(lambda x: x[2] == 1).count()
rating_1_count
```

```
unique_movies = ratings.map(lambda x: x[1]).distinct().count()
unique_movies
```

DAG Visualization :



Finding the user who rated the most movies.
Filtering and retrieving movies rated by that user.

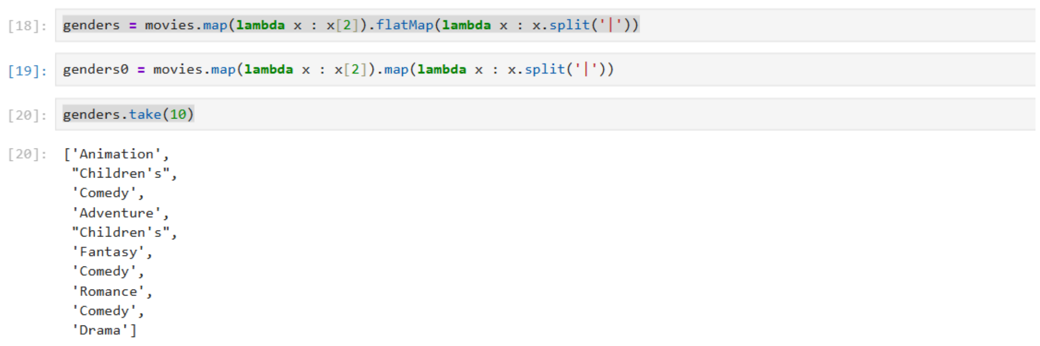


```
[ ]: user_id = ratings.map(lambda x : (x[0] ,1)).reduceByKey(lambda x,y: x+y).max(lambda x : x[1])
     user_id
```

```
[ ]: movies_rated_by_user = ratings.filter(lambda x: x[0] == user_id[0])
```

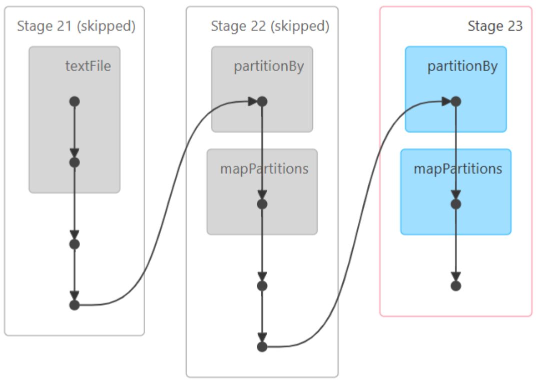
```
[ ]: movies_rated_by_user.take(5)
```

Extracting and flattening all genres from the movies dataset by splitting the genre string using |.

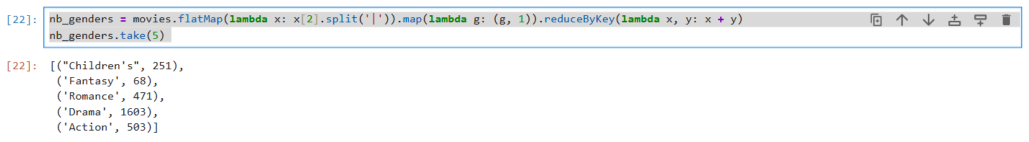


```
[18]: genders = movies.map(lambda x : x[2]).flatMap(lambda x : x.split('|'))

[19]: genders0 = movies.map(lambda x : x[2]).map(lambda x : x.split('|'))

[20]: genders.take(10)

[20]: ['Animation',
       "Children's",
       'Comedy',
       'Adventure',
       "Children's",
       'Fantasy',
       'Comedy',
       'Romance',
       'Comedy',
       'Drama']
```

DAG Visualization :

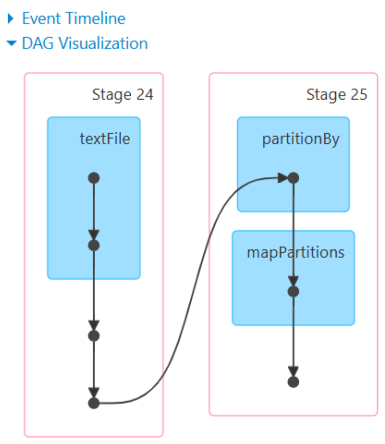


Counting the number of movies for each genre.



```
[22]: nb_genders = movies.flatMap(lambda x: x[2].split('|')).map(lambda g: (g, 1)).reduceByKey(lambda x, y: x + y)
      nb_genders.take(5)

[22]: [("Children's", 251),
       ('Fantasy', 68),
       ('Romance', 471),
       ('Drama', 1603),
       ('Action', 503)]
```

DAG Visualization :



## Get the most rated genres

```
[23]: valid_users = users.filter(lambda x : x[1] == 'M' and x[2] > 45 ).map(lambda x : (x[0],1))
      valid_ratings = ratings.filter(lambda x : x[2] >= 4).map(lambda x : (x[0] , x[1]))

      valid_rating_by_users = valid_users.join(valid_ratings).map(lambda x: (x[1][1],1))
```

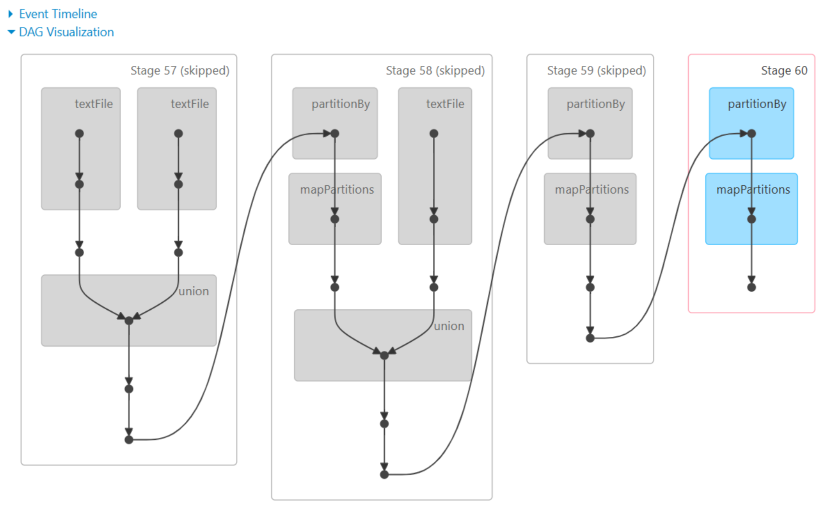
```
[24]: movies_genres = valid_rating_by_users.join(movies.map(lambda x: (x[0], x[2]))).flatMap(lambda x: x[1][1].split('|'))
      movies_genres.distinct().sortBy(lambda x:x[0],1).take(10)
```

```
[24]: ['Action',
       'Animation',
       'Adventure',
       "Children's",
       'Comedy',
       'Crime',
       'Drama',
       'Documentary',
       'Film-Noir',
       'Fantasy']
```

```
[25]: genre_counts = movies_genres.map(lambda genre: (genre, 1)).reduceByKey(lambda x, y: x + y)
      genre_counts.take(5)
```

```
[25]: [('Action', 11988),
       ('Sci-Fi', 7137),
       ('Western', 2024),
       ('Animation', 1361),
       ('Thriller', 9586)]
```

## DAG Visualization :



## Counting the number of movies per genre for each year

```
26]: import re
     def get_year(title) :
         match = re.search(r"\((\d{4})\)", title)
         return int(match[1]) if match else None
```

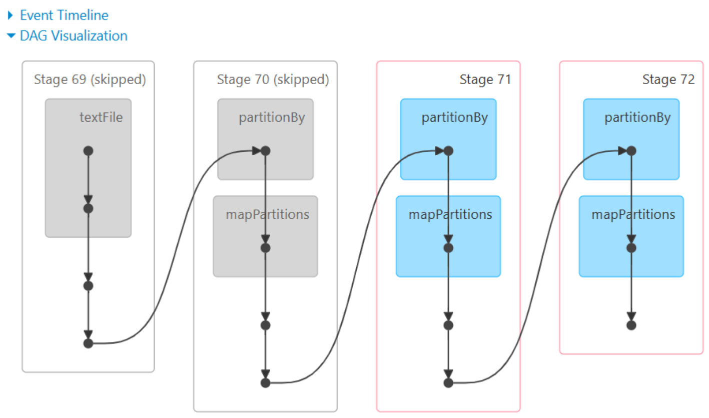
```
27]: movies_selected = movies.map(lambda x: (get_year(x[1]), x[2])) \
                               .flatMap(lambda x: [((x[0], genre),1) for genre in x[1].split('|')])
```

```
28]: movies_selected = movies_selected.reduceByKey(lambda x,y : x+y).map(lambda x: (x[0][0], (x[0][1], x[1])))
```

```
29]: movies_selected.take(10)
```

```
29]: [(1995, ('Animation', 8)),
      (1995, ('Comedy', 89)),
      (1995, ('Adventure', 25)),
      (1995, ('Crime', 18)),
      (1995, ('War', 12)),
      (1994, ('Action', 32)),
      (1994, ('Drama', 121)),
      (1994, ('Thriller', 31)),
      (1994, ('Romance', 37)),
      (1995, ('Mystery', 8))]
```

## DAG Visualization :



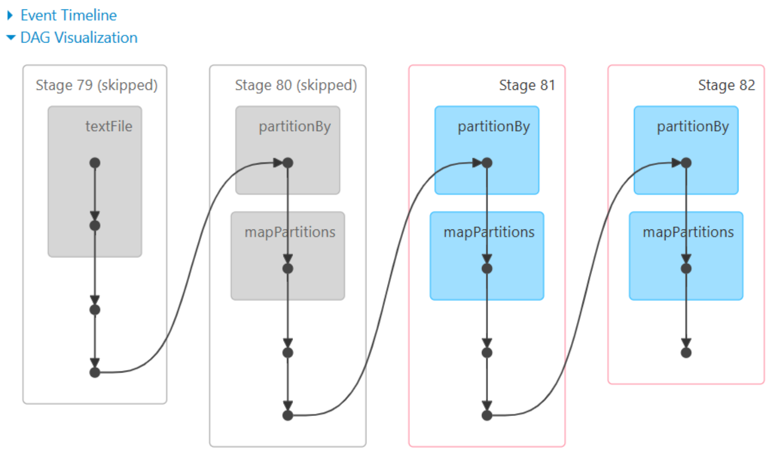
## Finding the most frequent genre for each year

```
[30]: movies_selected = movies_selected.reduceByKey(lambda a, b: a if a[1] > b[1] else b)
```

```
[31]: movies_selected.sortBy(lambda x : x[0] , 0 ).take(10)
```

```
[31]: [(2000, ('Comedy', 69)),
       (1999, ('Drama', 130)),
       (1998, ('Drama', 166)),
       (1997, ('Drama', 139)),
       (1996, ('Drama', 150)),
       (1995, ('Drama', 158)),
       (1994, ('Drama', 121)),
       (1993, ('Drama', 81)),
       (1992, ('Drama', 38)),
       (1991, ('Drama', 26))]
```

## DAG Visualization :


Event Timeline
DAG Visualization
Stage 79 (skipped)
textFile
Stage 80 (skipped)
partitionBy
mapPartitions
Stage 81
partitionBy
mapPartitions
Stage 82
partitionBy
mapPartitions