



TDSAI

Chapitre 2 : SPARK - Partie 2-

Dr. S.ZERABI

Faculté des NTIC

`Soumeya.zerabi@univ-constantine2.dz`

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	IFA	M1	SDIA

Actions sur les RDDs

Transformation	signification
<code>collect()</code>	Transfère les données du RDD dans le nœud maitre (driver) du cluster sous forme de tableau
<code>count()</code>	Compte le nombre d'éléments présents dans le RDD
<code>countByKey()</code>	Compte le nombre d'éléments pour chaque clé du RDD
<code>countByValue()</code>	Compte le nombre de valeurs dans un RDD
<code>first()</code>	Renvoie le premier enregistrement
<code>max()</code>	Renvoie l'enregistrement max
<code>min()</code>	Renvoie l'enregistrement min
<code>sum()</code>	Calcule la somme des éléments du RDD
<code>mean()</code>	Calcule la moyenne des éléments du RDD

Actions sur les RDDs

Transformation	signification
stdev()	Calcule l' écart type des éléments du RDD
reduce()	Effectue une agrégation entre les éléments du RDD. Il combine les données en utilisant une fonction associative qui produit un résultat
Lookup()	Renvoie la liste des valeurs dans le RDD pour la clé spécifiée.
getNumPartitions()	Retourne le nombre de nœuds sur le cluster
take(N)	Renvoie N premiers éléments du RDD
top(N)	Renvoie une liste de N premiers éléments du RDD (après avoir trier la liste par ordre descendant)
saveAsTextFile()	Persiste les données du RDD sur le disque dur
Coalesce()	Modifie le nombre de nœuds du cluster utilisés

count()

Calcule le nombre d'éléments du RDD.

```
rdd1 = spark.sparkContext.parallelize([2,7,1,6])  
print("le nombre d'ele du rdd1 est: ",rdd1.count())  
rdd2 = spark.sparkContext.parallelize([('a', 3), ('b', 5), ('c', 10)])  
print("le nombre d'ele du rdd2 est: ",rdd2.count())
```

```
le nombre d'ele du rdd1 est: 4  
le nombre d'ele du rdd2 est: 3
```

countByKey()

Calcule le nombre d'éléments pour chaque clé du RDD puis retourne le résultat sous forme de dictionnaire de paire (clé, nombre).

```
rdd = spark.sparkContext.parallelize([("a", 3), ("b", 5), ("c", 10), ('a', 10), ('b', 7)])  
rdd.countByKey()
```

```
defaultdict(int, {'a': 2, 'b': 2, 'c': 1})
```

countByValue()

Calcule le nombre de chaque valeur du RDD puis retourne le résultat sous forme de dictionnaire de paire (valeur, nombre).

```
rdd1 = spark.sparkContext.parallelize([1,2,1,2,3])  
rdd1.countByValue()
```

```
defaultdict(int, {1: 2, 2: 2, 3: 1})
```

```
rdd2 = spark.sparkContext.parallelize([('a', 3), ('a', 3), ('c', 4), ('a', 10), ('b', 4)])  
rdd2.countByValue()
```

```
defaultdict(int, {('a', 3): 2, ('c', 4): 1, ('a', 10): 1, ('b', 4): 1})
```

First()

Retourne le premier élément du RDD.



```
spark.sparkContext.parallelize([4, 6, 8]).first()
```



4

max(), min()

max(): Retourne l'élément maximum du RDD.

min(): Retourne l'élément minimum du RDD.



```
rdd = spark.sparkContext.parallelize([2,7,10,6])  
print("le maximum est: ",rdd.max())  
print("le minimum est: ",rdd.min())
```



```
le maximum est: 10  
le minimum est: 2
```


sum()

Retourne la somme des éléments du RDD.



```
rdd = spark.sparkContext.parallelize([2,7,1,6])  
rdd.sum()
```

16

mean()

Retourne la moyenne des éléments du RDD.



```
rdd = spark.sparkContext.parallelize([2,7,1,6])  
rdd.mean()
```

```
4.0
```

stdev()

Retourne l'écart type des éléments du RDD.



```
rdd = spark.sparkContext.parallelize([2,7,1,6])  
rdd.stdev()
```



```
2.5495097567963922
```

reduce()

Appliquer la fonction `reduce()` sur les éléments du RDD à l'aide de l'opérateur spécifié.

```
from operator import add
rdd = spark.sparkContext.parallelize([2,4,6,8,10])
print(rdd.reduce(add))
```

30

```
rdd = spark.sparkContext.parallelize([2,4,6,8,10])
rdd.reduce(min)
```

2

```
rdd = spark.sparkContext.parallelize([2,4,6,8,10])
rdd.reduce(lambda x,y: max(x,y))
```

10

lookup(*key*)

Renvoie la liste des valeurs du RDD pour la clé *key*.

```
rdd2 = spark.sparkContext.parallelize([('a', 3), ('a', 3), ('c', 4), ('a', 10), ('b', 4)])  
rdd2.lookup('a')
```

```
[3, 3, 10]
```

getNumPartitions()

Retourne le nombre de nœuds du cluster dans lequel l'opération est lancée.

```
rdd = spark.sparkContext.parallelize([1, 2, 3, 4])  
rdd.getNumPartitions()
```

2

take(N)

Renvoie N premiers éléments du RDD.

```
listRdd = spark.sparkContext.parallelize([10, 4, 2, 12, 3])  
listRdd.take(3)
```

[10, 4, 2]

top(N)

Renvoie une liste des N premiers éléments du RDD.





```
x= spark.sparkContext.parallelize([10, 4, 2, 12, 3])  
x.top(3)
```

```
[12, 10, 4]
```









saveAsTextFile(path)

Sauvegarde le RDD sur le disque.

```
rdd=spark.sparkContext.parallelize([15,60,20,35])  
rdd.saveAsTextFile('OUTPUT')
```

- ▼  OUTPUT
 -  _SUCCESS
 -  part-00000
 -  part-00001

```
x= spark.sparkContext.parallelize(range(10))  
x.saveAsTextFile('drive/MyDrive/Colab Notebooks/File')
```


- ▼  drive
 - ▼  MyDrive
 - ▼  Colab Notebooks
 - ▼  File
 -  _SUCCESS
 -  part-00000
 -  part-00001



coalesce() ou repartition()

Utilisées pour augmenter ou diminuer les partitions.

```
# Charger à nouveau les fichiers sauvegardés en tant que RDD
rdd_output = spark.sparkContext.textFile('OUTPUT')
# Utilisation de coalesce(1) ou repartition(1) pour fusionner les fichiers en un seul
rdd_output.coalesce(1).saveAsTextFile('Final_OUTPUT')

#rdd_output.repartition(1).saveAsTextFile('Final_OUTPUT')
```

▼  Final_OUTPUT

-  _SUCCESS
-  part-00000

DataFrames

... ..

Les Dataframes

- Les RDDs ne sont pas structurés.
- **Un dataframe** est un dataset organisé en lignes et colonnes nommées.
- Les colonnes peuvent avoir différents types.
- Il est non typée.
- C'est une abstraction d'un RDD sous forme de table équivalente à une table d'une BD ou un DF de Python/R.
- Il permet de bénéficier des fonctionnalités des RDDs et du moteur d'exécution de SQL dans Spark (Spark SQL).

Les Dataframes

- `createDataFrame()` de `SpakSession`
- Il peut être créer de deux façons:
 - De multiple sources de données:
 - Les Fichiers de données (csv, txt, json, xml, dat, etc.)
 - Les SGBDs relationnelles (MySQL, etc.)
 - Les SGBDs NoSQL (HBase, Cassandra, etc.)
 - Les RDDs existants.

Création des Dataframes

● A partir d'un schéma explicite

```
data = [('James', '', 'Smith', '1991-04-01', 'M', 3000),  
        ('Michael', 'Rose', '', '2000-05-19', 'M', 4000),  
        ('Robert', '', 'Williams', '1978-09-05', 'M', 4000),  
        ('Maria', 'Anne', 'Jones', '1967-12-01', 'F', 4000),  
        ('Jen', 'Mary', 'Brown', '1980-02-17', 'F', -1)]  
  
columns = ["firstname", "middlename", "lastname", "dob", "gender", "salary"]  
df=sc.createDataFrame(data, columns)  
df.printSchema()  
df.show()
```

```
root  
|-- firstname: string (nullable = true)  
|-- middlename: string (nullable = true)  
|-- lastname: string (nullable = true)  
|-- dob: string (nullable = true)  
|-- gender: string (nullable = true)  
|-- salary: long (nullable = true)
```

Création des Dataframes

● A partir d'un schéma explicite

```
data = [('James','', 'Smith', '1991-04-01', 'M', 3000),
        ('Michael', 'Rose', '', '2000-05-19', 'M', 4000),
        ('Robert', '', 'Williams', '1978-09-05', 'M', 4000),
        ('Maria', 'Anne', 'Jones', '1967-12-01', 'F', 4000),
        ('Jen', 'Mary', 'Brown', '1980-02-17', 'F', -1)]

columns = ["firstname", "middlename", "lastname", "dob", "gender", "salary"]
df=sc.createDataFrame(data, columns)
df.printSchema()
df.show()
```

```
+-----+-----+-----+-----+-----+-----+
|firstname|middlename|lastname|      dob|gender|salary|
+-----+-----+-----+-----+-----+-----+
|   James|          |   Smith|1991-04-01|    M|   3000|
| Michael|      Rose|          |2000-05-19|    M|   4000|
|  Robert|          |Williams|1978-09-05|    M|   4000|
|   Maria|      Anne|   Jones|1967-12-01|    F|   4000|
|     Jen|      Mary|   Brown|1980-02-17|    F|     -1|
+-----+-----+-----+-----+-----+-----+
```

Création des Dataframes

● A partir d'un dataframe pandas



```
import pandas as pd
df1 = pd.DataFrame({
    'a': [1, 2, 4],
    'b': [1., 3., 5.],
    'c': ['string1', 'string2', 'string3'],
    'd': [date(2000, 1, 1), date(2000, 2, 1), date(2000, 3, 1)]})
df2 = sc.createDataFrame(df1)
df2.show()
```



```
+---+---+---+---+
|  a|  b|      c|      d|
+---+---+---+---+
|  1|1.0|string1|2000-01-01|
|  2|3.0|string2|2000-02-01|
|  4|5.0|string3|2000-03-01|
+---+---+---+---+
```


Création des Dataframes

● A partir d'un RDD (avec createDataFrame())

```
▶ rdd = sc.sparkContext.parallelize([
    (1, 2., 'string1', date(2000, 1, 1)),
    (2, 3., 'string2', date(2000, 2, 1)),
    (3, 4., 'string3', date(2000, 3, 1))
])
df = sc.createDataFrame(rdd, schema=['a', 'b', 'c', 'd'])
df.printSchema()
df.show()
```

```
↳ root
  |-- a: long (nullable = true)
  |-- b: double (nullable = true)
  |-- c: string (nullable = true)
  |-- d: date (nullable = true)
```

```
+---+---+-----+-----+
|  a|  b|      c|      d|
+---+---+-----+-----+
|  1|2.0|string1|2000-01-01|
|  2|3.0|string2|2000-02-01|
|  3|4.0|string3|2000-03-01|
+---+---+-----+-----+
```

Création des Dataframes

● A partir d'un RDD (avec toDF())

```
df = rdd.toDF()  
df.printSchema()  
df.show()
```

```
root  
|-- _1: long (nullable = true)  
|-- _2: double (nullable = true)  
|-- _3: string (nullable = true)  
|-- _4: date (nullable = true)
```

```
+---+---+-----+-----+  
| _1| _2|      _3|      _4|  
+---+---+-----+-----+  
|  1|1.0|string1|2000-01-01|  
|  2|3.0|string2|2000-02-01|  
|  3|5.0|string3|2000-03-01|  
+---+---+-----+-----+
```

● En spécifiant le nom des colonnes

```
columns=['a', 'b', 'c', 'd']  
df = sc.createDataFrame(rdd).toDF(*columns)  
df.show()
```

```
+---+---+-----+-----+  
|  a|  b|      c|      d|  
+---+---+-----+-----+  
|  1|1.0|string1|2000-01-01|  
|  2|3.0|string2|2000-02-01|  
|  3|5.0|string3|2000-03-01|  
+---+---+-----+-----+
```

Création des Dataframes

● A partir d'un fichier csv

```
df=sc.read.csv('file.csv').show(3)
```

```
┌───┬───┬───┬───┬───┬───┐
│_c0│   │_c1│_c2│
├───┴───┴───┴───┴───┴───┤
│nom│   │prenom│age│
│ali│   │benali│ 20│
│med│benmohamed│ 30│
└───┴───┴───┴───┴───┴───┘
```

● Using the options

```
df=sc.read.options(header= 'True', delimiter=',').csv('file.csv' ).show()
```

```
┌───┬───┬───┬───┬───┬───┐
│nom│   │prenom│age│
├───┴───┴───┴───┴───┴───┤
│ali│   │benali│ 20│
│med│benmohamed│ 30│
└───┴───┴───┴───┴───┴───┘
```