

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

Contents

1	Spark and RDDs	2
1.1	Introduction	2
1.1.1	Getting Started with Spark	2
1.1.2	Introduction to Spark RDDs	2
1.2	RDDs Operations	3
1.2.1	Transformations	3
1.2.2	Actions	8

Chapter 1

Spark and RDDs

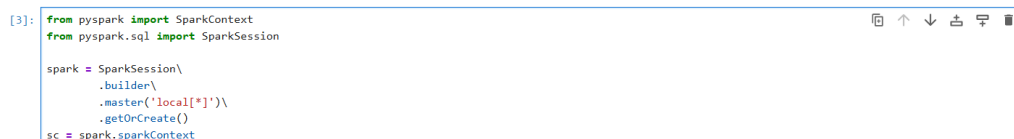
1.1 Introduction

1.1.1 Getting Started with Spark

To work with Spark, we need to initialize a `SparkContext` and a `SparkSession`.

SparkContext: It is the entry point for low-level Spark functionality, primarily used in RDD-based APIs. It enables interaction with the cluster, resource allocation, and parallel operations.

SparkSession: It is a unified entry point that encapsulates `SparkContext` and provides a higher-level API for `DataFrame` and SQL operations.



```
[3]: from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession\
    .builder\
    .master('local[*]')\
    .getOrCreate()
sc = spark.sparkContext
```

Figure 1.1: `SparkSession` and `SparkContext`

1.1.2 Introduction to Spark RDDs

Definition: A Resilient Distributed Dataset (RDD) is the fundamental abstraction in Spark. It represents an immutable, partitioned collection of elements that can be processed in parallel.

In this tutorial, we will explore the most commonly used functions for working with RDDs.

parallelize: This function distributes data across multiple partitions, enabling parallel processing.

```
[4]: data = sc.parallelize([1,2,3,4,5])
    print (data)
```

ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:274

Figure 1.2: Using the `parallelize` function

```
[5]: data = sc.parallelize([(1,2),(3,4),(3,6),(3,4)])
    data.collect()
```

[5]: [(1, 2), (3, 4), (3, 6), (3, 4)]

Printing the object created with the `parallelize` method will display the RDD object itself rather than the actual data.

```
[6]: type(data)
```

[6]: pyspark.rdd.RDD

Figure 1.3: Printing an RDD object

count: This function returns the number of elements in the RDD.

```
[7]: data.count()
```

[7]: 4

Figure 1.4: Using the `count` function

getNumPartitions: This function returns the number of partitions in an RDD.

```
[8]: num = sc.parallelize([5,5,4,3,2,9,2],5)
    num.getNumPartitions()
```

[8]: 5

Figure 1.5: Checking the number of partitions

1.2 RDDs Operations

1.2.1 Transformations

A Transformation is a function that produces new RDD from the existing RDDs. let's explore the most used transformations :

narrow transformations

map : It Applies a function to each element in the RDD and returns a new RDD with the same number of elements.

```
[9]: num.map(lambda a : a*2).collect()
[9]: [10, 10, 8, 6, 4, 18, 4]
```

```
[10]: num.map(lambda a : pow(a,3)).collect()
[10]: [125, 125, 64, 27, 8, 729, 8]
```

```
[11]: names = sc.parallelize(["Bills", "Mark", "Brain", "Mick"])
      names.map(lambda a : "Mr. "+ a).collect()
[11]: ['Mr. Bills', 'Mr. Mark', 'Mr. Brain', 'Mr. Mick']
[ ]:
```

flatMap : It Applies a function to each element in the RDD, but flattens the results into a single list.

```
[14]: rdd = sc.parallelize([ 2, 3,4])
      rdd.collect()
```

```
[14]: [2, 3, 4]
```

```
[15]: rdd.map(lambda x: range(1, x)).collect()
```

```
[15]: [range(1, 2), range(1, 3), range(1, 4)]
```

```
[16]: rdd.flatMap(lambda x: range(1, x)).collect()
```

```
[16]: [1, 1, 2, 1, 2, 3]
```

```
[17]: a = sc.parallelize([1,2,3])
      print(a.map(lambda x: (x,x*10,57)).collect())
      print(a.flatMap(lambda x: (x,x*10,57)).collect())
      [(1, 10, 57), (2, 20, 57), (3, 30, 57)]
      [1, 10, 57, 2, 20, 57, 3, 30, 57]
```

```
[ ]:
```

filter : it selects elements from an RDD based on a condition. It keeps only the elements where the function returns True.

```
[20]: num.collect()
[20]: [5, 5, 4, 3, 2, 9, 2]
[21]: num.filter(lambda x : x%3 == 1).collect()
[21]: [4]
[ ]:
```

```
[22]: names.collect()
names.filter(lambda x : "a" in x).collect()
[22]: ['Mark', 'Brain']
```

sortBy :it sorts an RDD based on a specified key .

```
[23]: print(names.sortBy(lambda x:x[0],1).collect())
print(names.sortBy(lambda x:x[0],0).collect())
['Bills', 'Brain', 'Mark', 'Mick']
['Mark', 'Mick', 'Bills', 'Brain']
```

union :It combines two RDDs without removing duplicates. It returns a new RDD that contains all elements from both RDDs.

```
[24]: num2 = sc.parallelize([1,1,7,9,4,10,15])
num2.collect()
[24]: [1, 1, 7, 9, 4, 10, 15]
[25]: num3=num2.union(num)
num3.collect()
[25]: [1, 1, 7, 9, 4, 10, 15, 5, 5, 4, 3, 2, 9, 2]
```

distinct :it removes duplicate elements from an RDD.

```
[27]: num3.distinct().collect()
[27]: [1, 2, 3, 4, 5, 7, 9, 10, 15]
```

sample :it selects a random subset of elements from an RDD.

```
[28]: parallel = sc.parallelize(range(1,20))
parallel.collect()

[28]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

[29]: parallel.sample(True,2,).collect()

[29]: [1, 4, 15]
```

Wide transformations

groupBy :it groups elements in an RDD based on a specified key.

```
: names.collect()
names_gr = names.groupBy(lambda x : x[0]).collect()

: names_gr

: [('M', <pyspark.resultiterable.ResultIterable at 0x7f138021a770>),
('B', <pyspark.resultiterable.ResultIterable at 0x7f138024bd60>)]

:
```

```
[33]: for (k,v) in names_gr:
      print(k,list(v))

M ['Mark', 'Mick']
B ['Bills', 'Brain']
```

```
[35]: aa = sc.parallelize([1, 1, 2, 3, 5, 8])
result = aa.groupBy(lambda x: x % 3).collect()
for (k,v) in result:
    print(k, list(v) )

0 [3]
1 [1, 1]
2 [2, 5, 8]
```

intersection :it returns a new RDD containing only the common elements between two RDDs.

```
[36]: num.intersection(num2).collect()

[36]: [4, 9]

[37]: num2.intersection(num).collect()

[37]: [4, 9]
```

countByValue :it counts the occurrences of each unique element in an RDD.

```
[39]: dataStr = sc.parallelize([(3, 'mike'), (2, 'john'), (3, 'rambo'), (4, 'bill'), (1, 'mike')])
      dataStr.collect()

[39]: [(3, 'mike'), (2, 'john'), (3, 'rambo'), (4, 'bill'), (1, 'mike')]

[40]: dataStr.countByValue()

[40]: defaultdict(int,
                    {(3, 'mike'): 1,
                     (2, 'john'): 1,
                     (3, 'rambo'): 1,
                     (4, 'bill'): 1,
                     (1, 'mike'): 1})

[ ]:
```

sortByKey :it sorts an RDD based on keys in key-value pairs.

```
[41]: data1 = sc.parallelize([(1, 2), (3, 6), (3, 1), (2, 4)])
      data1.sortByKey().collect()

[41]: [(1, 2), (2, 4), (3, 6), (3, 1)]
```

lookup :it retrieves all values associated with a given key in a key-value RDD.

```
[42]: data1.lookup(3)

[42]: [6, 1]
```

mapValues :It applies a function only to the values while keeping the keys unchanged.

```
[44]: data.mapValues(lambda a : a*a).collect()

[44]: [(1, 4), (3, 16), (3, 36), (3, 16)]
```

reduceByKey :It applies a specified function to combine values with the same key.

```
[45]: print(data.collect())
      print(data.reduceByKey(lambda x, y : x+y).collect())

[(1, 2), (3, 4), (3, 6), (3, 4)]
[(1, 2), (3, 14)]
```



```
[46]: data.reduceByKey(max).collect()
```

```
[46]: [(1, 2), (3, 6)]
```

groupByKey : it groups values by key in a key-value RDD.

```
[47]: result = data.groupByKey().collect()
      for (k,v) in result:
          print(k, list(v))
```

```
1 [2]
3 [4, 6, 4]
```

1.2.2 Actions

An RDD Action is defined as the spark operations that return raw values.
example : collect , count , take , top ..

```
[48]: names = sc.parallelize(['Adam','Cray','Shaun','Brain','Mark','Christ','Shail','Satya','Mark','Norby','Frans','Mark','Bill'])
      print(names.collect())
```

```
['Adam', 'Cray', 'Shaun', 'Brain', 'Mark', 'Christ', 'Shail', 'Satya', 'Mark', 'Norby', 'Frans', 'Mark', 'Bill']
```

```
[49]: names.countByValue()
```

```
[49]: defaultdict(int,
      {'Adam': 1,
       'Cray': 1,
       'Shaun': 1,
       'Brain': 1,
       'Mark': 3,
       'Christ': 1,
       'Shail': 1,
       'Satya': 1,
       'Norby': 1,
       'Frans': 1,
       'Bill': 1})
```

```
-----
```

```
[50]: names.take(5)
```

```
[50]: ['Adam', 'Cray', 'Shaun', 'Brain', 'Mark']
```

```
[51]: names.count()
```

```
[51]: 13
```

```
[ ]:
```