

# Transformations and Actions In `pyspark` RDD

Students:

- 2019DMF05 - Ishita Gupta
- 2019DMF06 - Kishan R
- 2019DMF14 - Yogesh GS
- 2019DMB05 - Lilendar Rohidas
- 2019DMB07 - Rohith Krishna
- 2019DMB08 - Shashi Ranjan Mandal

# List of transformations and actions

## Actions

1. count()
2. collect()
3. take(n)
4. top()
5. countByValue()
6. reduce()
7. fold()
8. aggregate()
9. foreach()

## Transformations

1. map(func)
2. flatMap()
3. filter(func)
4. mapPartitions(func)
5. mapPartitionWithIndex()
6. union(dataset)
7. intersection(other-dataset)
8. distinct()
9. groupByKey()
10. reduceByKey(func, [numTasks])
11. sortByKey()
12. join()
13. coalesce()

# aggregate()

`aggregate()` lets you take an RDD and generate a single value that is of a different type than what was stored in the original RDD.

Parameters:

- `zeroValue`: The initialization value, for your result, in the desired format.
- `seqOp`: The operation you want to apply to RDD records. Runs once for every record in a partition.
- `combOp`: Defines how the resulted objects (one for every partition), gets combined.

```
seqOp = lambda data, item: (data[0] + [item], data[1] + item)
combOp = lambda d1, d2: (d1[0] + d2[0], d1[1] + d2[1])
```

```
x = sc.parallelize([1,2,3,4])
y = x.aggregate([], 0, seqOp, combOp)

print(y)
```



**x:** [1, 2, 3, 4]

**y:** ([1, 2, 3, 4], 10)

Cmd 18

## aggregate()

Cmd 19

```
seqOp = (lambda local_result, list_element:(local_result[0] + list_element, local_result[1] + 1))
combOp = (lambda some_local_result, another_local_result: (some_local_result[0] +
                                                            another_local_result[0],
                                                            some_local_result[1] +
                                                            another_local_result[1]) )

listRDD = sc.parallelize(acc,10)
listRDD.aggregate((0,0), seqOp, combOp)
```

► (1) Spark Jobs

Out[28]: (1568611, 777)

Command took 0.55 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:25:29 on myCluster



# fold()

`fold(zeroValue, op)`

[\[source\]](#)

Aggregate the elements of each partition, and then the results for all the partitions, using a given associative function and a neutral “zero value.”

The function `op(t1, t2)` is allowed to modify `t1` and return it as its result value to avoid object allocation; however, it should not modify `t2`.

This behaves somewhat differently from fold operations implemented for non-distributed collections in functional languages like Scala. This fold operation may be applied to partitions individually, and then fold those results into the final result, rather than apply the fold to each element sequentially in some defined ordering. For functions that are not commutative, the result may differ from that of a fold applied to a non-distributed collection.

```
>>> from operator import add
>>> sc.parallelize([1, 2, 3, 4, 5]).fold(0, add)
15
```

>>>

# fold()

## Syntax

```
def fold(zeroValue: T)(op: (T, T) => T): T
```

## Usage

Since RDD's are partitioned, the `fold()` function takes full advantage of it by first aggregating elements in each partition and then aggregating results of all partitions to get the final result. The result of this function is the same as this RDD type.

This function takes the following arguments –

`zeroValue` – Initial value to be used for each partition in folding, this value would be used to initialize the accumulator for each partition. we mostly use `0` for integer and `Nil` for collections.

`op` – an operator used to both accumulate results within a partition and combine results from all partitions,

## fold()

Cmd 16

```
from operator import add
acc = list(df.select('Accept').toPandas()['Accept'])
sc.parallelize(acc).fold(0, add)
```

► (2) Spark Jobs

Out[11]: 1568611

Command took 0.72 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:18:19 on myCluster

Cmd 17

```
df.select('Accept').groupBy().sum().collect()
```

► (2) Spark Jobs

Out[19]: [Row(sum(Accept)=1568611)]

Command took 0.37 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:20:37 on myCluster

# foreach()

In Spark, `foreach()` is an action operation that is available in RDD, DataFrame, and Dataset to **iterate/loop over each element in the dataset**. It is similar to `for` with advance concepts. This is different than other actions as `foreach()` function **doesn't return a value instead it executes input function on each element of an RDD, DataFrame, and Dataset**. This is particularly useful in you have to call perform some calculation on an RDD and log the result somewhere else, for example a database or call a REST API with each element in the RDD.

**`foreach(f)`**

[\[source\]](#)

Applies a function to all elements of this RDD.

```
>>> def f(x): print(x)
>>> sc.parallelize([1, 2, 3, 4, 5]).foreach(f)
```

>>>



Cmd 20

## foreach()

Cmd 21

```
schools = list(df.select('School').toPandas()['School'])
names = sc.parallelize(schools)
def f(x):
    print(x)
scl = names.foreach(f)
print(scl)
```

▸ (2) Spark Jobs

None

Command took 0.86 seconds -- by pgdm19rohith@mse.ac.in at 27/10/2020,

Cmd 22

```
def f(x):
    print(x)
sc.parallelize(acc).foreach(f)
```

▸ (1) Spark Jobs

Command took 0.51 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020,

Cmd 23

```
def printing(x):
    print(x)
```

```
def div_two(n):
    return n/2
```

```
numbersRDD = sc.parallelize(acc)
numbersRDD.map(div_two).foreach(printing)
```

▼ (1) Spark Jobs

▸ Job 183 [View](#) (Stages: 1/1)

Command took 0.47 seconds -- by pgdm19rohith@mse.ac.in at

## groupByKey()

```
def listOfTuples(list1, list2):  
    return list(map(lambda x, y:(x,y), list1, list2))  
  
school = list(df.select('School').toPandas()['School'])  
private = list(df.select('Private').toPandas()['Private'])  
  
pvt = listOfTuples(private, school)
```

► (2) Spark Jobs

Command took 0.65 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:30:07 on myCluster

```
pairRDD1 = sc.parallelize(pvt)  
pairRDD1.groupByKey().mapValues(len).collect()
```

► (1) Spark Jobs

Out[32]: [('No', 212), ('Yes', 565)]

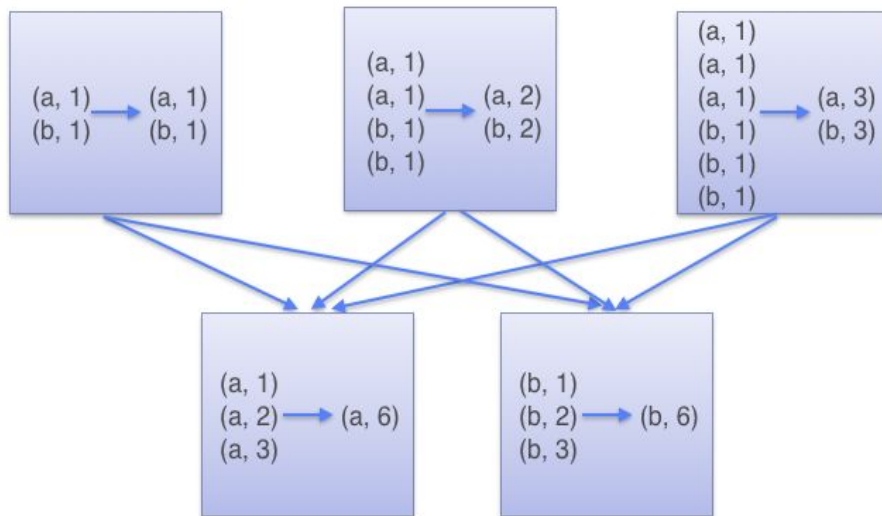
# reduceByKey()

`reduceByKey(func)` runs several parallel reduce operations, one for each key in the dataset, where each operation combines values that have the same key.

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function.

Because datasets can have very large numbers of keys, `reduceByKey()` is not implemented as an action that returns a value to the user program.

Instead, it returns a new RDD consisting of each key and the reduced value for that key.



## reduceByKey()

Cmd 29

```
private = list(df.select('Private').toPandas()['Private'])
accept = list(df.select('Accept').toPandas()['Accept'])
pvtacc = listOfTuples(private, accept)
```

► (2) Spark Jobs

Command took 0.53 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:37:45 on myCluster

Cmd 30

```
pairRDD2 = sc.parallelize(pvtacc)
pairRDD2_red = pairRDD2.reduceByKey(lambda x,y: x+y)
```

Command took 0.07 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:39:14 on myCluster

Cmd 31

```
[print(num) for num in pairRDD2_red.collect()];
```

► (1) Spark Jobs

```
('No', 830889)
('Yes', 737722)
```

Command took 0.64 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:40:33 on myCluster

# sortByKey()

`sortByKey()` is a transformation which returns an Resilient Distributed Datasets (RDD) sorted by Key.

Sorting can be done in :

- Ascending
- Descending
- Custom sorting

When we apply the `sortByKey()` function on a dataset of (K, V) pairs, the data is sorted according to the key K in another RDD.

For example: In our `college.csv` data using `sortByKey()` we can find that as our enrollment increases the graduation rate also increases.

## sortByKey()

Cmd 33

```
pairRDD2_red_sort = pairRDD2_red.sortByKey(ascending = False)
```

► (2) Spark Jobs

Command took 0.39 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:44:54 on myCluster

Cmd 34

```
[print(num) for num in pairRDD2_red_sort.collect()];
```

► (1) Spark Jobs

('Yes', 737722)

('No', 830889)

Command took 0.27 seconds -- by pgdm19rohith@mse.ac.in at 26/10/2020, 23:44:55 on myCluster

# join()

When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through `leftOuterJoin`, `rightOuterJoin`, and `fullOuterJoin`.

Transformations on two pair RDDs:

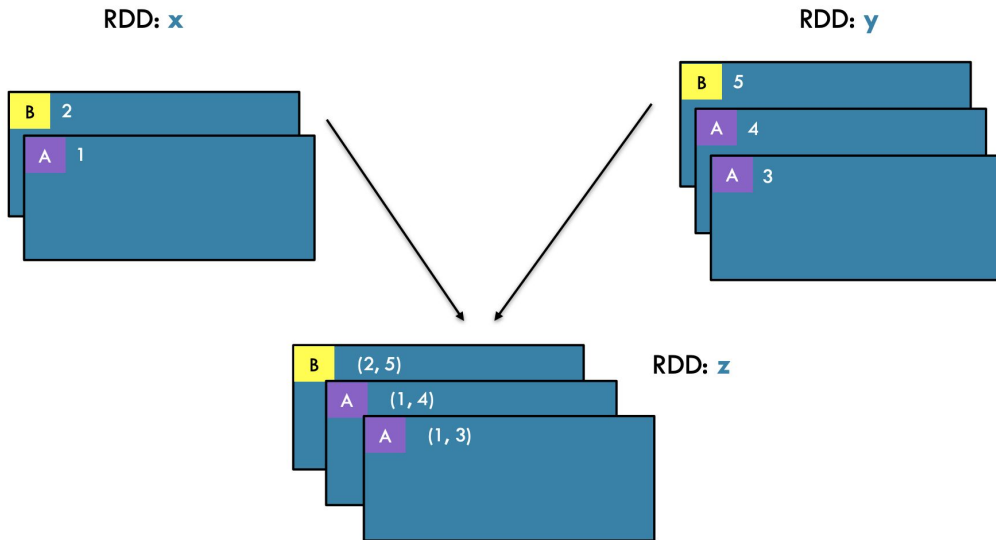
```
rdd = {(1, 2), (3, 4), (3, 6)}
```

```
other = {(3, 9)}
```

```
rdd.join(other)
```

Function Result:



```
{(3, (4, 9)), (3, (6, 9))}
```



# join()

Cmd 36

```
dfA = sqlContext.sql("SELECT School, Accept, Enroll FROM College")
dfB = sqlContext.sql("SELECT School, Books, PhD FROM College")
```

-  dfA: pyspark.sql.dataframe.DataFrame = [School: string, Accept: integer ... 1 more fields]
-  dfB: pyspark.sql.dataframe.DataFrame = [School: string, Books: integer ... 1 more fields]

Command took 0.20 seconds -- by pgdm19rohith@mse.ac.in at 27/10/2020, 00:00:55 on myCluster

Cmd 37

```
dfCombined=dfA.join(dfB,dfA.School == dfB.School, "inner")
display(dfCombined)
```

- (1) Spark Jobs
-  dfCombined: pyspark.sql.dataframe.DataFrame = [School: string, Accept: integer ... 4 more fields]

	School ▲	Accept ▲	Enroll ▲	School ▲	Books ▲	PhD ▲
1	Abilene Christian University	1232	721	Abilene Christian University	450	70
2	Adelphi University	1924	512	Adelphi University	750	29
3	Adrian College	1097	336	Adrian College	400	53
4	Agnes Scott College	349	137	Agnes Scott College	450	92
5	Alaska Pacific University	146	55	Alaska Pacific University	800	76
6	Albertson College	479	158	Albertson College	500	67
7	Albertus Magnus College	340	103	Albertus Magnus College	500	90

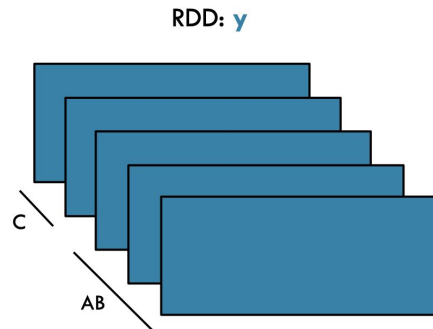
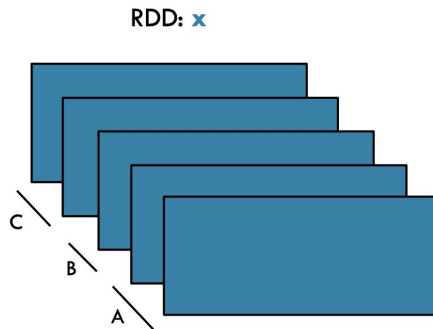
Showing all 777 rows.



# coalesce()

On specifying the value for numPartitions `coalesce(numPartitions)` returns a new DataFrame that has exactly numPartitions partitions.

It results in a narrow dependency, e.g. if you go from 1000 partitions to 100 partitions, there will not be a shuffle, instead each of the 100 new partitions will claim 10 of the current partitions.



```
x = sc.parallelize([1, 2, 3, 4, 5], 3)
y = x.coalesce(2)
print(x.glom().collect())
print(y.glom().collect())
```



**x:** [[1], [2, 3], [4, 5]]

**y:** [[1], [2, 3, 4, 5]]

## coalesce()

Cmd 39

```
x = sc.parallelize(acc, 7)
y = x.coalesce(3)
print('Number of Partitions in x: ', len(x.glom().collect()))
print('Length of each partition in x')
for i in (x.glom().collect()):
    print(len(i))

print('Number of Partitions in y: ', len(y.glom().collect()))
print('Length of each partition in y')
for i in (y.glom().collect()):
    print(len(i))

#print(x.glom().collect())
#print(y.glom().collect())
```

### ► (4) Spark Jobs

```
Number of Partitions in x: 7
Length of each partition in x
111
111
111
111
111
111
111
Number of Partitions in y: 3
Length of each partition in y
222
222
333
```



# TRANSFORMATIONS

+



# ACTIONS

Thank you!