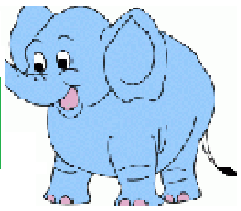
Data, BigData. Cloud.. Anaytics & Virtualizaton HadoopExam.com | QuickTechie.com

1. Apache Spark Professional Training with Hands On Lab Sessions
2. Oreilly Databricks Apache Spark Developer Certification Simulator

SPARK PAIR RDD FUNCTIONS

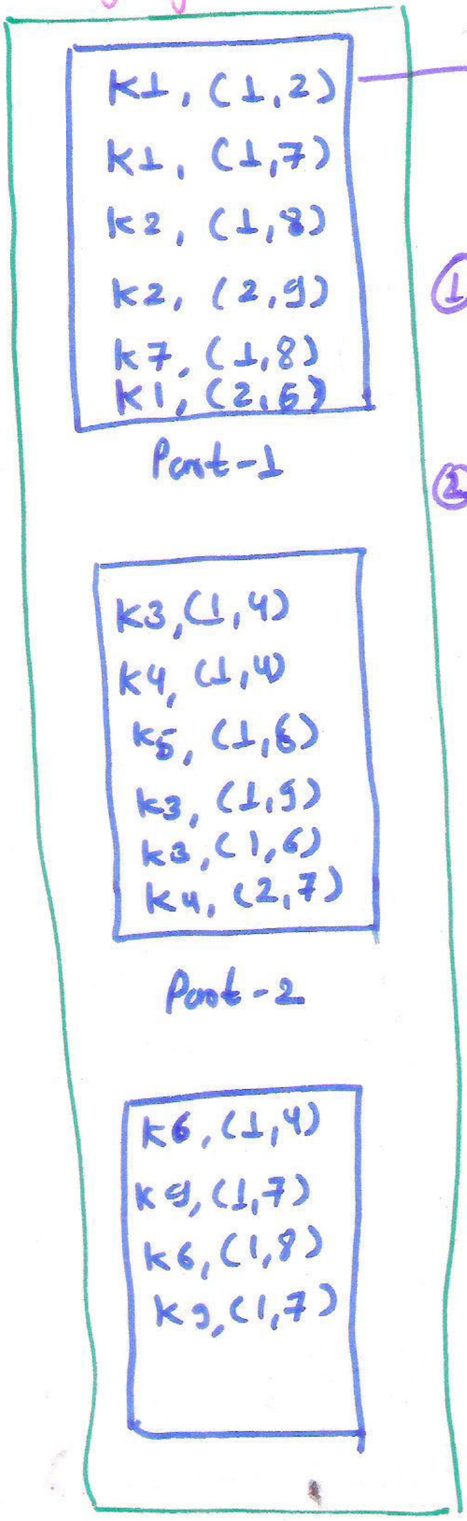
By www.HadoopExam.com

Note: These instructions should be used with the HadoopExam Apache Spark: Professional Trainings. Where it is executed and you can do hands on with trainer.



1. reduceByKey()
2. groupByKey()
3. combineByKey()
4. foldByKey()
5. aggregateByKey()
6. Comparision Between Function

reduce Bykey()



dataRDD

$\text{reduceBykey}((k1, \underset{x}{\underset{\uparrow}{(1,2)}}), (k1, \underset{y}{\underset{\uparrow}{(1,7)}}))$

① $k1 = [(1+1), (2+7)]$

$k1 = [(2,9)]$

② $k1\ \text{reduceBykey}((k1, \underset{x}{\underset{\uparrow}{(2,9)}}), (k1, \underset{y}{\underset{\uparrow}{(2,6)}}))$

$k1 = [(2+2), (9+6)]$

$= [(4,15)$

P1 => [(k1,(4,15)),(k2,(3,17)),(k7,(2,6))

P2 => [(k3,(3,19)),(k4,(3,11)),(k5,(1,6))

P3 => [(k6,(2,12)),(k9(2,14))]

Locallyeach Partition

dataRDD.reduceBykey((x,y) =>

(x._1 + y._1, x._2 + y._2))

groupBykey()

```
Val words = Array("one", "two", "two", hadoop, hadoop, hadoop)
Val generatePairRDD = sc.parallelize(words).map(word => (word, 1))
```

⇓

(one,1), (two,1), (two,1), (hadoop,1), (hadoop,1), (hadoop,1)

```
Val wordCountWithReduce
    = generatePairRDD.reduceBykey(_ + _).collect()
```

(one,1), ⟹ (one,1)

(two,1), (two,1) ⟹ two ⟹ (1+1) ⟹ (two,2)

(hadoop,1), (hadoop,1), (hadoop,1)

↳ (hadoop,(1+1)), (hadoop,1)

↳ (hadoop,3)

```
Val wordWithGroup = generatePairRDD.groupBykey()
                    .map(t => (t._1, t._2.sum)
                    .collect()
```

$$\boxed{\begin{array}{l}(one,1)\\(two,(1,1))\\(hadoop,(1,1,1))\end{array}} \Longrightarrow .map$$

(one,1)

(two => (1,1).sum)

(hadoop => (1,1,1).sum)

⇓

(one,1)

(two, 2)

(hadoop,3)

=) Both will produce same result.

=) reduceBykey works better (local Aggregation)

=) groupBykey => first shuffle and then aggregate
(Hence, huge nw traffic involved, which will cause performance issue)

=> ~~Ih~~ Explaination to below image: —

$\boxed{\text{ReduceBykey V/S GroupBykey.}}$

Here are more functions to prefer over groupBykey()

combineBykey(): — Can be used when you are combining elements but your return type is different from your input.

foldBykey(): — merges the values for each key using an associative function and a neutral "zero value"

foldByKey() & fold()

fold() : Example

```
sc.parallelize(1 to 10)
            .fold(0){(acc, element) =>
                              acc + element)
```

first pass → [(0,1) ⇒ (0+1)] ⇒ 1

2nd pass → [(1,2) ⇒ (1+2)] ⇒ 3

3rd pass → [(3,3) ⇒ (3+3)] ⇒ 6

4th pass → [(6,4) ⇒ (6+4)] ⇒ 10

⋮

10th pass → [(45,10) ⇒ (45+10)] ⇒ 55

Another Example : Count element in list

```
sc.parallelize(1 to 10).fold(0){(acc, element) =>
                              acc+1}
```

$\boxed{\text{output would be} = 10}$

foldBykey() :-

```
Val depEmployee = List(
                    (CS, (Amit, 1000)),
                    (CS, (Rahul, 1200)),
                    (ECE, (Rakesh, 1500)),
                    (ECE, (Ankit, 1200))
                  )

Val empRDD = sc.makeRDD(depEmployee)

// find max score by dept

Val maxdept = empRDD.foldBykey(("dummy", 0.0))
        ((acc, element) => if (acc._2 > element._2)
                              acc
                            else
                              element)
```

for cs

~~max~~ firstPass

CS → [(dummy, 0.0), (Amit, 1000)]

=> (Amit, 1000)

CS → [(Amit, 1000), (Rahul, 1200)]

=> (Rahul, 1200)

ECE → [(dummy, 0.0), (Rakesh, 1500)]

=> (Rakesh, 1500)

ECE → [(Rakesh, 1500), (Ankit, 1200)]

=> (Rakesh, 1500)

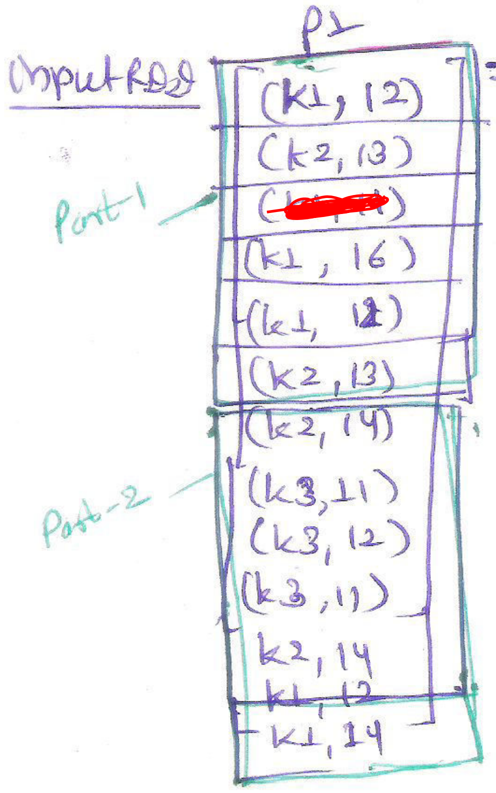
maxdept => [(CS, (Rahul, 1200)), (ECE, (Rakesh, 1500))]

CombineByKey():-

CombineByKey(CreateCombiner,
                          mergeValue,
                          mergeCombiners,
                          partitioner)

Example Code: Calculate Running Average

```
val result = inputRDD.combineByKey(
    ① ——> (v) => (v,1)
  ② <—— (acc: (Int, Int), v) => (acc._1 + v, acc._2+1)
  ③ <—— (acc1:(Int,Int), acc2: (Int,Int)
              => (acc1._1 + acc2._1 , acc1._2 + acc2._2))
         .map{ case(key,value) => (key, value._1/value._2.toFloat)
```



=> ~~for k1~~

Part-1 first time k1 → (12,1)
first time k2 → (13,1)
2nd time k1 → ((12,1),16)
→ (28,2)
3rd time k1 → ((28,2),12)
→ (~~36,3~~) (40,3)

Part-2 for k1
first time k1 → (12,1)
2nd time k1 → ((12,1),14)
→ (26,2)

finally operation 3 (Using all Accumulator)
across partition ((36,3),(26,2)) => (62,5)

Similarly it applies to all keys (k1, k2, k3 ---etc.) across all partitions.

$k1 \rightarrow 62/5$

$k2 \rightarrow ((26,2),(28,2)) \Rightarrow (54,4)$

$\Rightarrow 54/4$

$k3 \rightarrow ((34,3)) \Rightarrow 34/3$

~~result = [k1, 12.5~~

result = [(k1, 12.4), (k2, 13.5), (k3, 11.33)]

Running Average for all the keys.

Aggregate ByKey() - This function also requires three parameters.

① An initial zero value, which will not affect the total values to be collected.

e.g. Summation = 0+5 = 5 etc
0+6 = 6

Collecting unique elements from set. - Empty set

emptySet.add("hadoop") ⇒ emptySet[hadoop]

② A combiner function, which accept two parameters

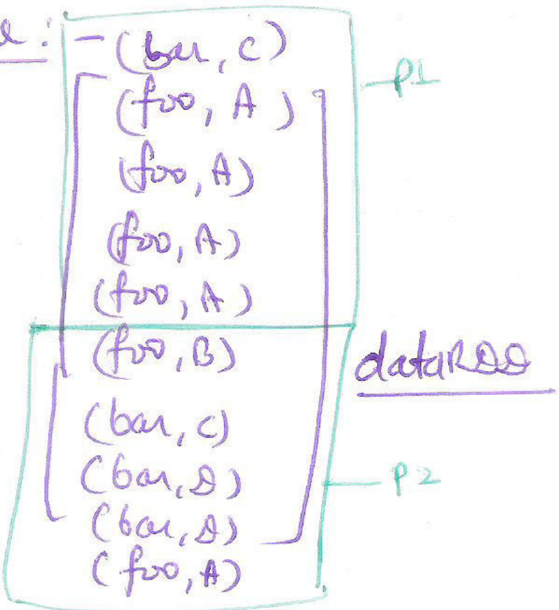
⇒ Second parameter will be merged into the first parameter

=> Combining function will work on local partition only.

③ Merging function = Works across the partition.

It also accepts two parameters.

Example:



~~dataRDD.aggregateByKey((0),((n:Int, V:String) => n+1)~~,

~~(p1~~

dataRDD.aggregateByKey(0)((acc:Int, V:String) => acc+1),

((acc:Int, acc:Int) => acc+acc))

①

②

③

P1

foo → (0+1) => 1
foo → (1+1) => 2
foo → (2+1) => 3
foo → (3+1) => 4
~~foo →~~ (
bar → (0+1) => 1

P2

foo → (0+1) => 1
foo → (1+1) => 2
bar → (0+1) => 1
bar → (1+1) => 2
bar → (2+1) => 3

final Result

foo → (4+2) => (foo → 6)
bar → (1+3) => bar → 4

function Comparison =>

① You can replace groupBykey() with reduceBykey() to improve performance.

② reduceBykey() performs map side combine which can reduce n/w IO and shuffle size.

③ groupBykey() will not perform map side combine.

④ CombineBykey() is more general than aggregateBykey()

⑤ Implementation of aggregateBykey, reduceBykey, and groupBykey is achieved by combineBykey().

⑥ AggregateBykey() is similar to reduceBykey(), but you can provide initial values when performing aggregation.

=> As name suggests, aggregateBykey is suitable for compute aggregations for keys such as sum, avg. etc.

=> CombineBykey(): is more general and you have the flexibility to specify whether you would like to perform map side combine

=> However, eg combineBykey() is more complex, at the minimum you need to implement three functions,

① create Combiner
② merge Value
③ merge Combiners.

cloudera CCA175 Is Now Available , with Hands On Sessions

CCA Spark and Hadoop Developer Certification

HadoopExam Learning Resource provides the following material for the Advanced Technologies.
Please visit www.HadoopExam.com for more detail this is just a few products from portfolio.

Price start for training with Just $79/3500INR

Spark Apache Spark Professional Training with HandsOn Session + Certification Material	hadoop Hadoop Professional Training with HandsOn Sessioon + Certification Material	APACHE HBASE HBase Professional Training with HandsOn Session + Certification Material	android Certification Material	ORACLE Java Certification Material
amazon web services Certification Material	sas Certification Material	EMC² Certification Material	Microsoft Azure Certification Material	EMC² Data Scientist Certification Material