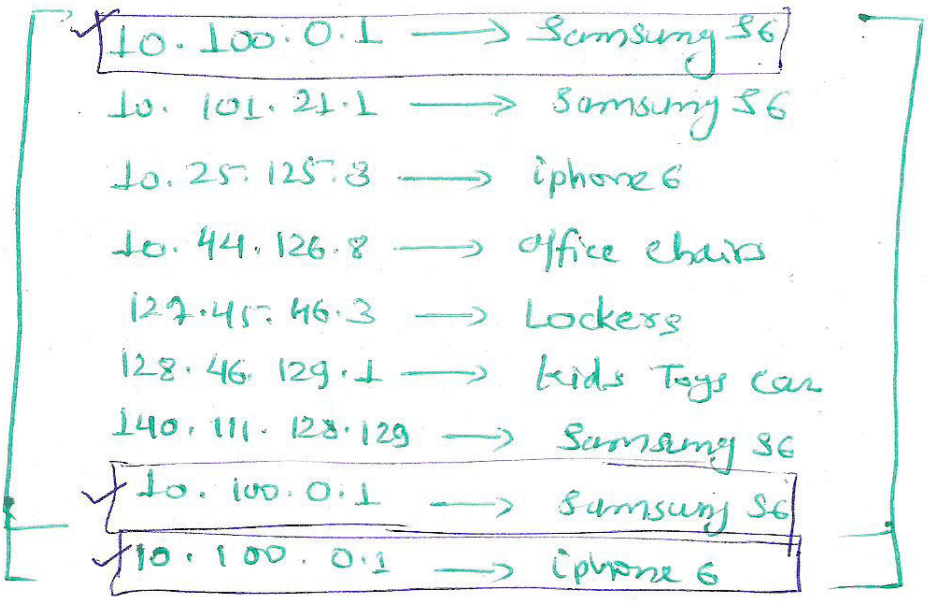
WORKING WITH SPARK PAIR RDD

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.

Working With Pair RDD

Example of Pair RDD : flipkart.com



⇒ most demanding phone — Samsung S6

⇒ Same user is comparing - iphone 6 & Samsung S6

10.100.0.1 → is likely to buy high end phone.

⇒ Flipkart will share these search with google ~~Adsense~~. Adwords.

⇒ And wherever possible google will show Samsung S6 & iphone 6 advertisement on possible Ad space on webpage. of use is browsing. (With Buy Button from flipkart.com)

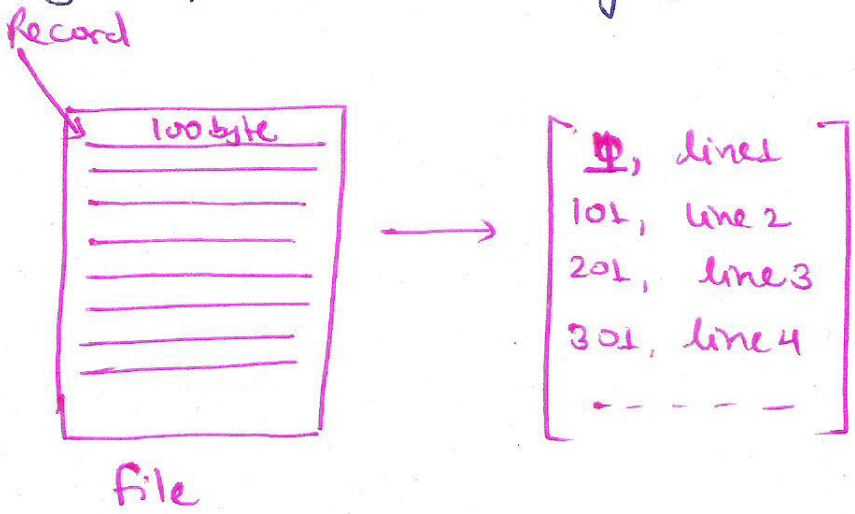
⇒ So this pair RDD has following key and value pairs.

[ip, search keyword]

⇒ Creating Pair RDD's

⇒ In Hadoop when file is loaded, each line became a value.

⇒ Byte offset as a key for that line.



⇒ But in Spark, it is created as below.

```
Val token = sc.textfile("HadoopExam.log")
Val pairs = token.map(x => x.split(' ')(0),x)
```

⇒ Regular RDD converted in a pair RDD using map() function.

Aggregation In Pair RDDs :

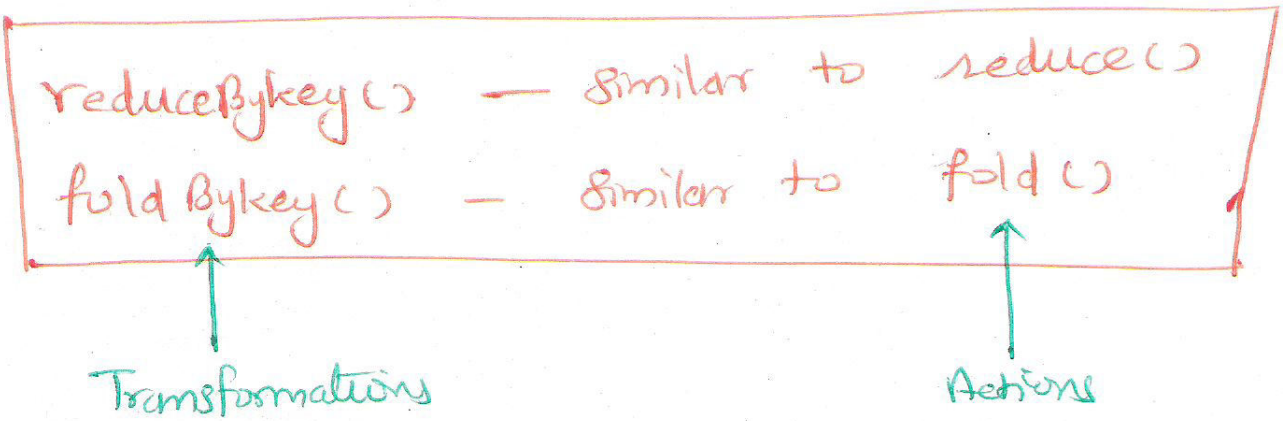
⇒ It is common to aggregate all values with the same key together.

⇒ On Regular RDD : .

fold(), reduce(), combine()

All Are Actions

=> But in pair RDD, we have similar functions for transformations.



=> reduceBykey(): take a function as input and use it to combine value for same key.

e.g. (+)

=> However, it will reduce together only for same key.

=> Generally (key, value) pairs file are bigger as stored in HDFS. (1GB to TB)

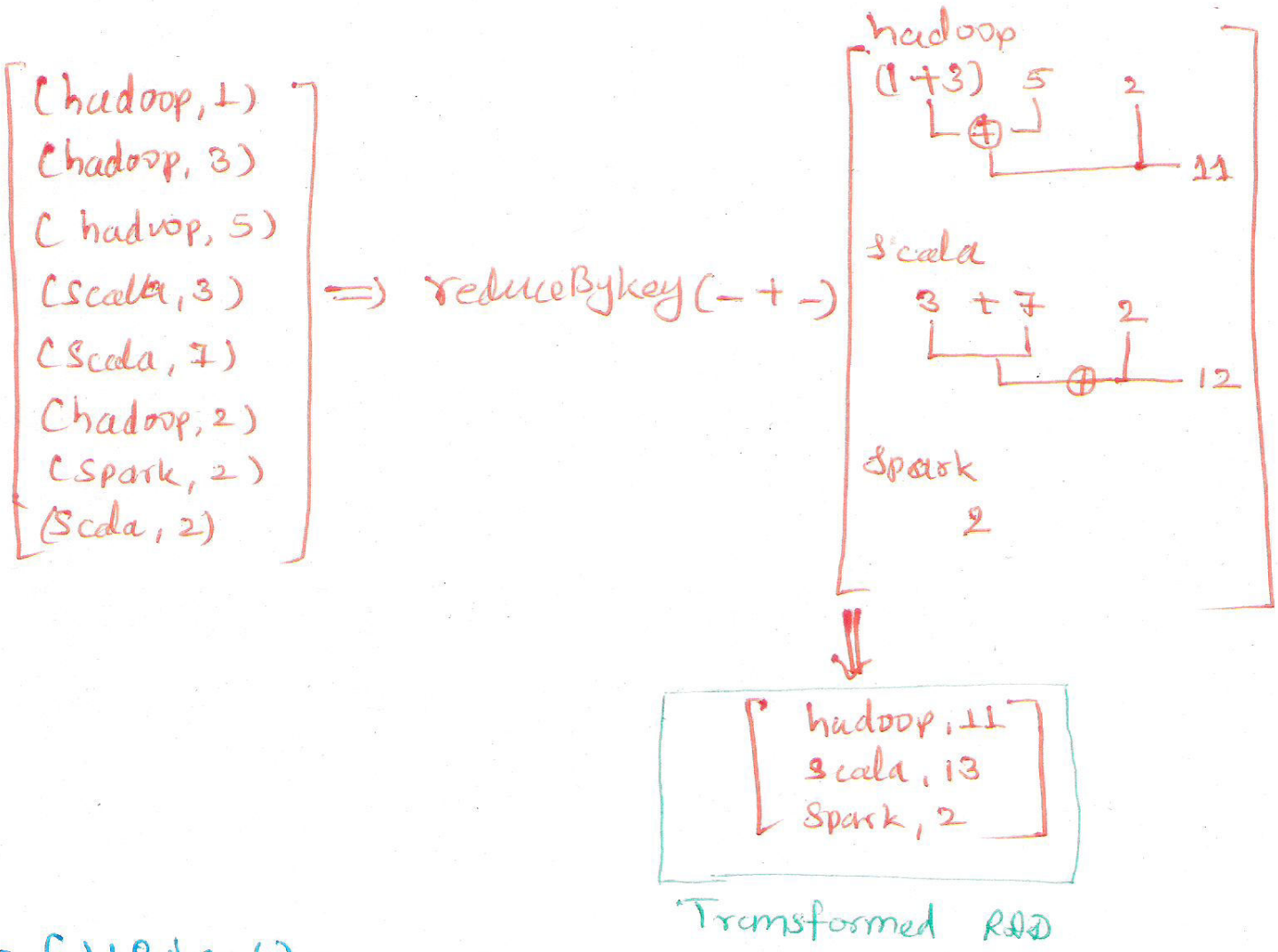
=> Hence, reduceBykey() is not implemented as an action.

=> Because action always returns a value to the user program.

⇒ If values are very large, then it would be a memory issue. or space issue on single machine.

⇒ Instead, it returns a new RDD consisting of each key and reduced values of that key.

⇒ And RDD is partitioned across cluster hence will not cause space issue.



⇒ foldBykey():

It is also quite similar to fold() Action.

⇒ It requires the zerovalue similar to fold() function, for initialization value.

⇒ If you know the Hadoop framework, then their was a concept of combiner.

⇒ Combiner is like mini reducer, which work independently on each partition (on individual node)

⇒ And combines the data locally.

⇒ Similarly reduceBykey() and foldBykey() will automatically perform combining locally on each node/machine before computing global totals for each key.

⇒ In Spark user do not have to specify combiner.

⇒ However, you can create a custom combiner.

⇒ Let's see word count example

```
val input = sc.textFile("hdfs://hadoopExam.log")
val words = input.flatMap(x => x.split(' '))
val results = words.map(x => x,1)
              .reduceByKey((x,y) => x+y)
```

⇒ CombineBykey():-

⇒ This is the underline function of many other function.

⇒ It is most general of the per-key aggregation function.

⇒ Like aggregate(), combineBykey() allows the user to return values that are not same type as our input data.

```
val result = input.combineBykey(
    (v) => (v,1)
    (acc:(Int,Int),v) =>
        acc._1+v, acc._2+1)
    (acc1:(Int,Int), acc2:(Int,Int)
    => (acc1._1 + acc2._1, acc1._2+acc2._2))
.map{ case(key,value) => (key, value._1/value._2)}
```

result.collectMap().map(println(_))

Example :- Average search count for each keyword during last 3 days.

(Two Partitions)

day1

day2

day3

Partition P1 using combineByKey()

① Create Combiner on P1 on P1

hadoop ↔ (70,1)
Spark ↔ (50,1)
Sqoop ↔ (30,1)
hive ↔ (40,1)

② mergeValue on P1

hadoop → [(70,1), 107] ⇒ (177, 2)
Spark → [(50,1), 50] ⇒ [100, 2]

① Create combiner on P2

Sqoop — (28,1)
hive — (90,1)
hadoop — (190,1)
Spark — (200,1)
Scala — (30,1)

② mergeValue() on P2

Sqoop [(28,1), 20] — (48, 2)
hive [(90,1), 40] — (130, 2)
~~Scala~~

③ mergeCombiner (Apply 3rd function)

(Work on both P1 + P2)

hadoop (177 + 190, 2+1)
Spark (100 + 200, 2+1)
Sqoop (30 + 48, 1+2)
hive (40 + 130, 1+2)
Scala (30, 1)

⇒

hadoop (367, 3)
Spark (300, 3)
Sqoop (78, 3)
hive (170, 3)
Scala (30, 1)

⇒ Then do Avrge calculation

Value / count

⇒ There are many options for combining our data by key.

⇒ Most of them are implemented on top of

combineByKey()

but provide a simpler interface.

WWW.HadoopExam.Com

WWW.QuickTechie.com