Core Spark Main Functions

Notebook:	Spark		
Created:	5/18/2018 9:03 AM	**Updated:**	6/3/2018 10:02 PM
Author:	ashu41228@gmail.com		
URL:	http://sparkhdpcd.blogspot.in/		

Core spark Functions

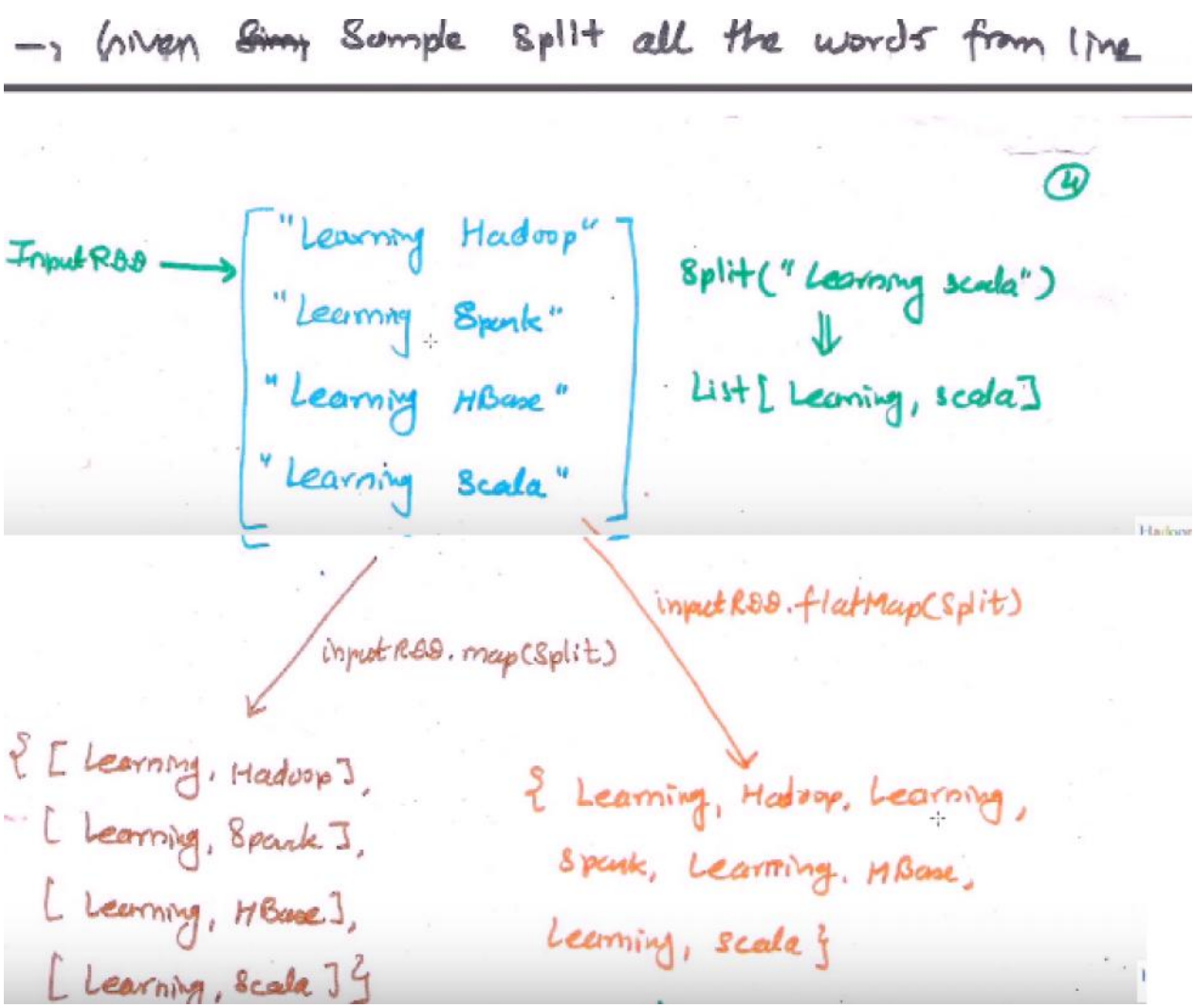
Transformation Function

1. map
2. flatMap
3. filter
4. mapPartitions
5. mapPartitionsWithIndex
6. cogroup
7. join
8. leftOuterJoin
9. rightOuterJoin
10. groupByKey
11. reduceByKey
12. combineByKey
13. foldByKey
14. aggregateByKey
15. distinct
16. intersection
17. repartition
18. coalesce
19. subtract
20. Pair RDD --> mapValues

Action Functions

1. collect()
2. count()
3. first()
4. reduce()
5. countByKey()
6. saveAsTextFile(path)
7. takeSample()
8. takeOrdered()
9. foreach

Map and flatMap



In flatmap each element in operated first then after that flatten operation is performed which give list of all element its like we do split and then explode in HIVE.

mapPartitions(func)

The **MapPartition** converts each *partition* of the source RDD into many elements of the result (possibly none). In mapPartition(), the map() function is applied on each partitions simultaneously. MapPartition is like a map, but the difference is it runs separately on each partition(block) of the RDD.

mapPartitionWithIndex()

It is like mapPartition; Besides mapPartition it provides *func* with an integer value representing the index of the partition, and the map() is applied on partition index wise one after the other.

Creating Pair RDDs

Pair RDDs can be created by running a map() function that returns key or value pairs. The procedure to build the key-value RDDs differs by language. In Python language, for the functions on keyed data to work we need to return an RDD composed of tuples Creating a pair RDD using the first word as the key in Python programming language.

pairs = lines.map(lambda x: (x.split(" ")[0], x))

Cogroup

When two data sets of type (k, v) and (k, w) are grouped, it will result in (k, (Iterable[v], Iterable[w]))

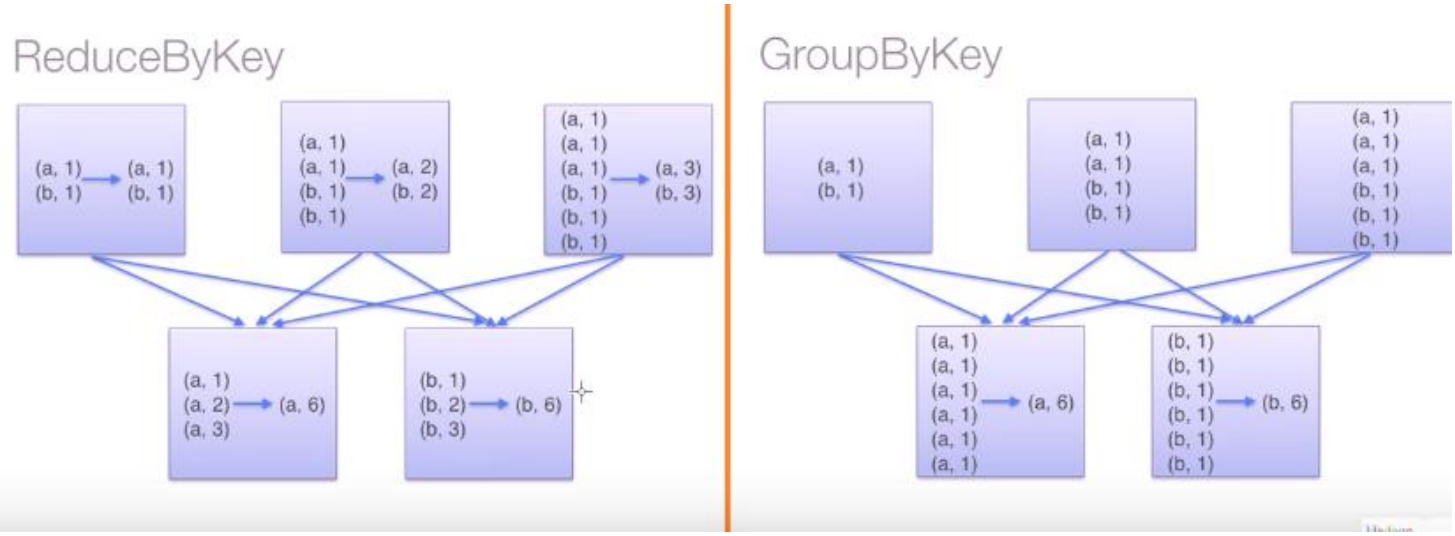
joins

When two data sets of type (k, v) and (k, w) are joined, it will result in (k, (v, w))

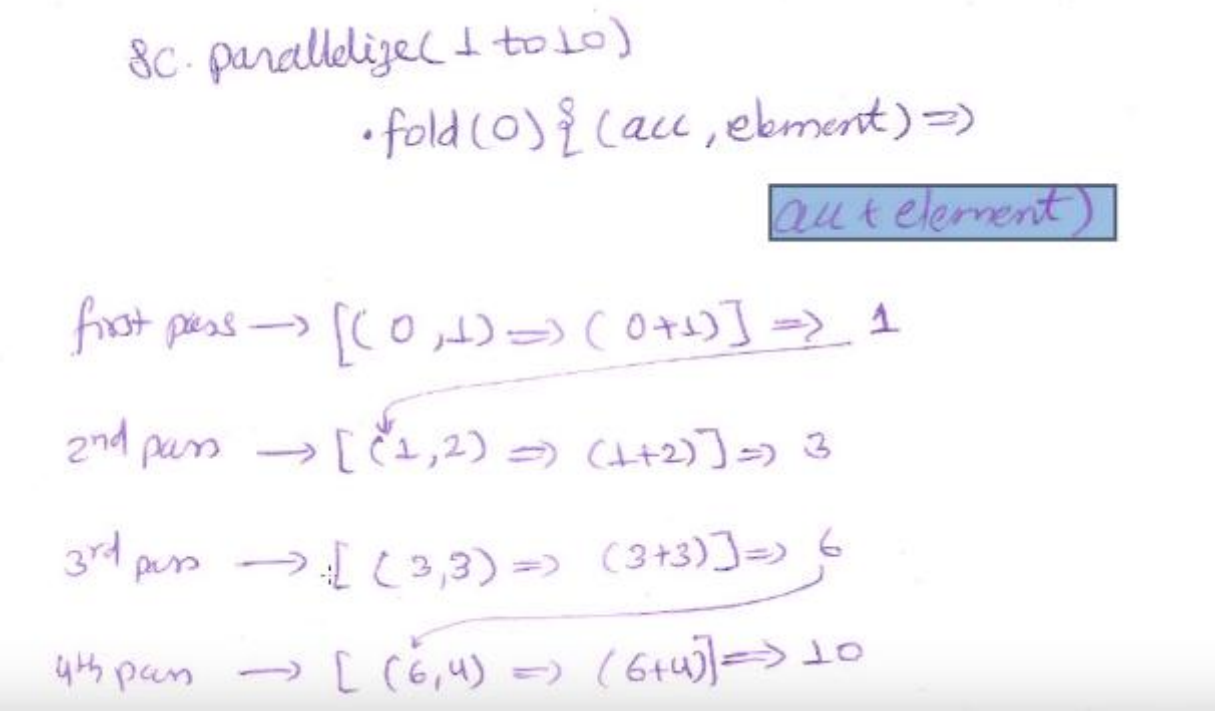
GroupByKey, ReduceByKey, combineBykey, AggregateByKey

reduceByKey will aggregate y key before shuffling,it means (K,V) pair will be aggregated at their respective partion/nodes.hence good in performance.

groupbyKey will aggregate y key after shuffling,it means (K,V) pair will be aggregated at single partition/node.hence bad in performance.
combineBykey.



Fold -->same as reduce but needs an initial zero value



ReduceBykey

```
pairs.reduceByKey((accumulatedValue: Int, currentValue: Int) => accumulatedValue + currentVal
```

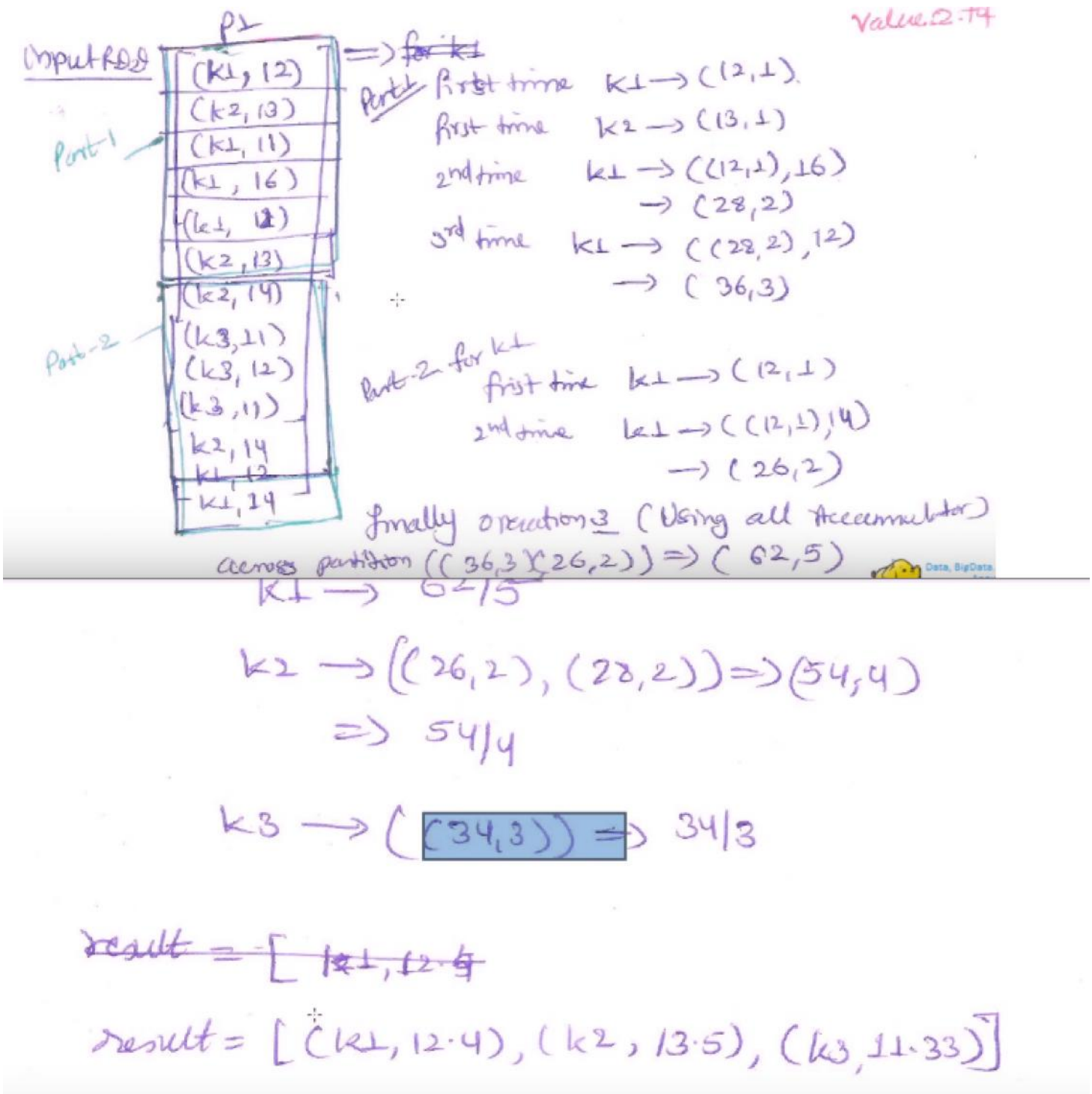
Combinebykey

CombineByKey used 4 main parameters
1)create combiner --> this will be used to initiliaze a value when key is occured first time at each partitioner/node
2)merger value --> this will be used to perform operation on (K,V) when same Key k is occured more than once at each partitoner/node
3)merge combiners --> this will be used to perform operation on different (K,V) generated by different partions output in order to perform operion on same key emitted by different partitioners. combines the final result
4)partitioner optional

Find average

```
data = sc.parallelize( [(0, 2.), (0, 4.), (1, 0.), (1, 10.), (1, 20.)] )

sumCount = data.combineByKey(lambda value: (value, 1),

lambda x, value: (x[0] + value, x[1] + 1),

lambda x, y: (x[0] + y[0], x[1] + y[1]))
```

```
averageByKey = sumCount.map(lambda (label, (value_sum, count)): (label, value_sum / count))

print averageByKey.collectAsMap()
```



AggregateByKey

Aggregate By key() :- 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, 0+6=6 etc

Collecting unique elements from set. - Empty set

emptySet.add("hadoop") ⇒ emptySet[

② A Combiner function, which accept two parameters

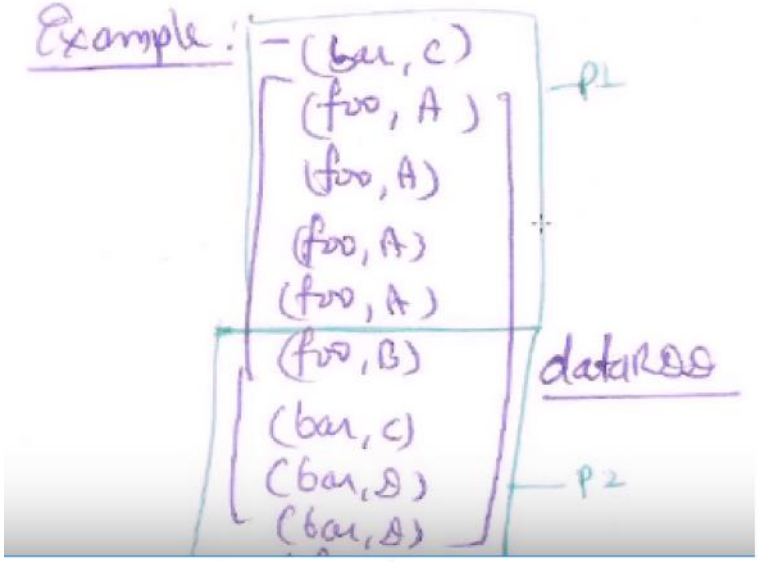
⇒ Second parameter will be merged into the first parameter

⇒ Combining function will work on local partition only. ⑧

③ Merging function = Works across the partition.

It also accepts two parameters.

Wordcount example pleas note we are ignore A,B,C in below example only considering words



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 → 6~~

P2

foo → (0+1) ⇒ 1
foo → (1+1) ⇒ 2
bar → (0+1) ⇒ 1
bar → (1+1) ⇒ 2
bar → (2+1) ⇒ 3

finalResult

foo → (4+2) ⇒ (foo → 6)
bar → (1+3) ⇒ bar → 4

function Comparision =>

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().

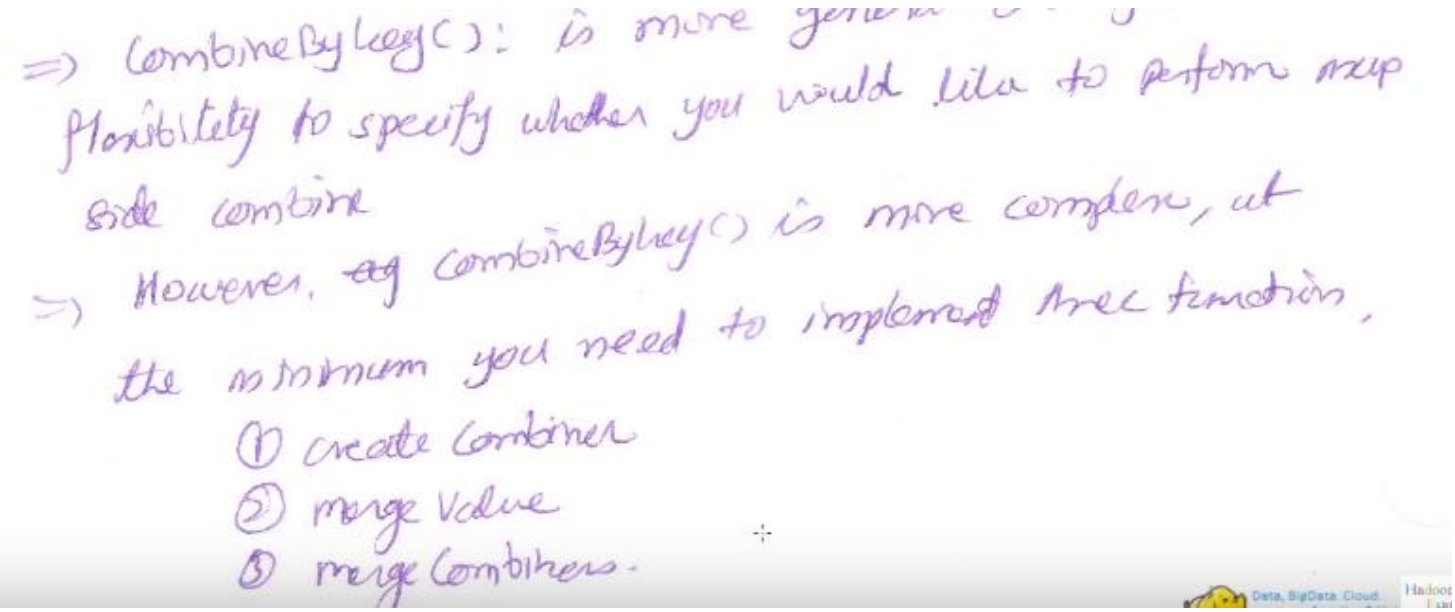
⑤ Implementation of aggregateByKey, reduceByKey and groupByKey is achieved by combineByKey()

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,



CountbyKey

It counts the value of **RDD** consisting of two components tuple for each distinct key. It actually counts the number of elements for each key and return the result to the master as lists of (key, count) pairs.

```
val rdd1 = sc.parallelize(Seq(("Spark",78),("Hive",95),("spark",15)
rdd1.countByKey
```

Output:

scala.collection.Map[String,Long] = Map(Hive -> 1, BigData -> 1, HBase -> 1, spark -> 3, Spark -> 1)

foreach()

foreach() is an action. Unlike other actions, foreach do not return any value. It simply operates on all the elements in the RDD. foreach() can be used in situations, where we do not want to return any result, but want to initiate a computation. A good example is ; inserting elements in RDD into database.