Aggregate by key work better where we need to calculate a result from computing from all the records

And it works fast because it uses the spark distributed computing feature. For this it uses combiner and Reducer

Continuing upon last video ( 11th video) we can write

Val orderJoinMapABK= ordersJoinMap.aggregateByKey(**(0.0,0))**(

**(intAgg: (Double, Int), intVal: Double) => (intAgg.\_1+intVal, intAgg.\_2+1) ,**

**(totAgg :(Double, Int) , totVal :(Double, Int)) => (totAgg.\_1 + totVal.\_1 , totAgg.\_2 + totVal.\_2)**

**)**

------Explanation------

1. (0.0,0) is an initializer

Here initializer and intAgg are is the format you want your output is ie.. (Double , Int)

It is of type TUPLE

1. **(intAgg: (Double, Int), intVal: Double) => (intAgg.\_1+intVal, intAgg.\_2+1) ,**

is the combiner logic iie.. the calculations you want to put on each row

intAgg is the output logic

intVal is the input ie.. the values on which you are working on so it is having type Double

**(intAgg.\_1+intVal, intAgg.\_2+1)**

it is the result which will calculate the output of the type (Double, Int) which is

(totalRevenue, Count)

1. Here Combiner will do this process by deviding it in different slots….

Like For eg ..let say for 100 elements combiner devides and working in 2 slots …. One batch is 0f 60 element and other is for 40 element

For this Reducer is used which finally sums both 2 batches or slots and generates the final result

**(totAgg :(Double, Int) , totVal :(Double, Int)) => (totAgg.\_1 + totVal.\_1 , totAgg.\_2 + totVal.\_2)**

To explain how it works

Lets take an example of counting and aggregating a list value

Val L=List(100.00 ,200.00 , 270.00 , 350.00)

Var total=(0.0,0)

For(i <- L)

{

Total =(total.\_1+I , total.\_2+1)

}

AggregateByKey works in same way

Reduce by key is also works same way like aggregate by key where combiner and reducer logic is same

So to do same task with ReduceByKey we need to map the dataset from fomat (date, revenue) to

(date, (revenue,1)) then apply RBK

Ie..

.map(rec => (rec.\_1,(rec.\_2,1))).

reduceByKey((agg,value) => (agg.\_1 + value.\_1,agg.\_2+value.\_2))

So Final program result will be

(2014-06-15 00:00:00.0,(11968.36999999999,62))

(2013-10-05 00:00:00.0,(20431.799999999985,92))

(2013-09-28 00:00:00.0,(14486.109999999988,84))

(2014-05-17 00:00:00.0,(14433.509999999982,74))

(2013-08-11 00:00:00.0,(15212.209999999988,70))

(2013-08-08 00:00:00.0,(15623.399999999994,76))

(2014-07-24 00:00:00.0,(18933.169999999995,91))

(2014-04-17 00:00:00.0,(8827.009999999997,43))

(2014-05-23 00:00:00.0,(13375.529999999995,64))

And the final program with input/output directory validations to calculate total revenue for each day with number of orders

**import** com.typesafe.config.ConfigFactory  
**import** org.apache.spark.{SparkConf, SparkContext}  
**import** org.apache.hadoop.fs.\_  
  
**object** retial\_order {  
 **def** main(args: Array[String]): Unit = {  
  
 **val** props=ConfigFactory.*load*()  
 **val** conf = **new** SparkConf().  
 setMaster(props.getConfig(args(2)).getString(**"executionMode"**)).  
 setAppName(**"Total Revenue Collection"**)  
  
 **val** sc = **new** SparkContext(conf)  
  
 **val** inputBaseDir=args(0)  
 **val** outputBasedir=args(1)  
  
 **val** fs = FileSystem.*get*(sc.hadoopConfiguration)  
 **if**(!fs.exists(**new** Path(inputBaseDir)))  
 {  
 *println*(**"Base Directory Does not exists"**)  
 **return** }  
  
 **if**(fs.exists(**new** Path(outputBasedir)))  
 fs.delete(**new** Path(outputBasedir),**true**)  
  
  
  
  
 **val** ordersFilterMap = sc.textFile(inputBaseDir +**"\\orders\\part-00000"**).  
 filter(rec => (rec.split(**','**)(3) == **"CLOSED"** || rec.split(**','**)(3) == **"COMPLETE"**)).  
 map(rec => (rec.split(**','**)(0).toInt, rec.split(**','**)(1)))  
  
 **val** orderItemMap = sc.textFile(inputBaseDir+**"\\order\_items\\part-00000"**).  
 map(rec => (rec.split(**','**)(0).toInt, rec.split(**','**)(4).toDouble))  
  
 **val** totalrevenue=ordersFilterMap.join(orderItemMap).  
 map(rec=> (rec.\_2)).  
 aggregateByKey((0.0,0))(  
 (intAgg: (Double, Int), intVal: Double) => (intAgg.\_1+intVal, intAgg.\_2+1) ,  
 (totAgg :(Double, Int) , totVal :(Double, Int)) => (totAgg.\_1 + totVal.\_1 , totAgg.\_2 + totVal.\_2)).  
  
 */\*map(rec => (rec.\_1,(rec.\_2,1))).  
 reduceByKey((agg,value) => (agg.\_1 + value.\_1,agg.\_2+value.\_2)).\*/* saveAsTextFile(outputBasedir)  
 }  
}

**Now Practice this exercise for elections in UP**

<http://discuss.itversity.com/t/exercise-09-scala-and-spark-political-analysis-for-the-state-of-up/2907>

Download dataset

Not go to retail project directory and type sbt console

We will read data

Val fileContents= sc .textFile(“ ”)

Now we move data using mapPartitionsWithIndex will read data and map each record

As per their column name and map instances with partitions

**mapPartitionsWithIndex()**, which apart from similar to mapPartitions() also provides an index to track the Partition No

val data=fileContents.mapPartitionsWithIndex( (idx , iter) => if(idx==0) iter.drop(1) else iter)

We only want UP data

Val dataUP=data.filter(rec => rec.split(“\t”)(0) == “Uttar Pradesh”)

**Now we want to get the results if SP and BSP would have fought together …what would have been our results**

**For this we will map our data accordingly as per data required**

Val dataPerConstituencyAndParty = dataUP.map(rec => {

val r=rec.split(“\t”)

((r(0),r(1)) , (r(6),r(10)))

})

val dataPerConstituency = dataPerConstituencyAndParty.groupByKey()

**Now the data would look like**

((Uttar Pradesh,Baghpat),CompactBuffer((RLD,199516), (SP,213609), (BSP,141743), (BJP,423475), (PECP,3025), (NAP,601), (NNP,441), (AAAP,5828), (IND,440), (IND,992), (IND,3915), (IND,2708), (IND,2092), (IND,1967), (NOTA,3911)))

((Uttar Pradesh,Jaunpur),CompactBuffer((BJP,367149), (SP,180003), (INC,42759), (BSP,220839), (BMUP,4026), (SBSP,2204), (SHS,1751), (SPP,7206), (PMSP,6814), (AAAP,43471), (SUCI,2608), (SaDa,2329), (RUC,19636), (PECP,2064), (IND,2694), (IND,7281), (IND,64137), (IND,2773), (IND,20832), (IND,2109), (IND,1863), (NOTA,2595)))

**Now we will create a fumction to ADD the votes from SP and INC and make them as ALLY**

scala> def func1(rec: Iterable[(String, Int)]) :Iterable[(String, Int)] ={

rec.map(r => {

if(r.\_1 == "INC" || r.\_1 == "SP")

("ALLY",r.\_2)

else

r

}).groupBy(r=>r.\_1).

map(r => (r.\_1 ,r.\_2.map(\_.\_2).sum))

}

func1: (rec: Iterable[(String, Int)])Iterable[(String, Int)]

**Using this function**

val dataWith3Way = dataPerConstituency.map(rec => (rec.\_1 , func1(rec.\_2)))

((Uttar Pradesh,Shrawasti),Map(PECP -> 101817, BJP -> 345964, BSP -> 194890, ALLY -> 280057, JSEP -> 2961, JSK -> 4489, NOTA -> 14587, AAAP -> 2578, IND -> 29107, SPP -> 3188))

((Uttar Pradesh,Mohanlalganj),Map(PECP -> 1427, BJP -> 455274, BSP -> 309858, NAP -> 1782, SARP -> 1632, MwSP -> 1494, ALLY -> 294964, RPI(A) -> 1826, INL -> 4280, ASaP -> 1617, MaSPa -> 2425, RRP -> 2225, AITC -> 5115, NOTA -> 4708, AAAP -> 10031, IND -> 17930))

**Now the final result can be generated as**

dataWith3Way.map(rec => (rec.\_1 , rec.\_2.toList.sortBy(k => -k.\_2))).

map(rec => (rec.\_1.\_2, rec.\_2(0)).

foreach(println)

Here we are first sorting the elements in descending order that’s why we have used –k .

Then we have used the first element of sorted list …which will give the ruling party information with maximum votes.

**OUTPUT**

(Banda,(BJP,342066))

(Mathura,(BJP,574633))

(Ballia,(BJP,359758))

(Akbarpur,(BJP,481584))

(Meerut,(BJP,532981))

(Kaushambi,(BJP,331724))

(Azamgarh,(ALLY,358256))

(Moradabad,(BJP,485224))

(Etawah,(BJP,439646))

(Firozabad,(ALLY,542030))

(Sitapur,(BJP,417546))

(Hardoi,(BJP,360501))

(Hathras,(BJP,544277))

(Jalaun,(BJP,548631))

(Bijnor,(BJP,486913))

(Bansgaon,(BJP,417959))

(Ghosi,(BJP,379797))

(Shrawasti,(BJP,345964))

(Mohanlalganj,(BJP,455274))

Now to get the final UP results ie.. no of seats won by parties will be shown with

**dataWith3Way.map(rec => (rec.\_1 , rec.\_2.toList.sortBy(k => -k.\_2))).map(rec => ((rec.\_1.\_1, rec.\_2(0).\_1),1)).countByKey()**

we could have also used reduceByKey() for the same

**output**

scala.collection.Map[(String, String),Long] = Map((Uttar Pradesh,ALLY) -> 12, (Uttar Pradesh,AD) -> 2, (Uttar Pradesh,BJP) -> 66)

**Sorting and Rating**  
  
Sorting can be of two kinds   
  
1. global sorting -- sorting on baseline ..with a single processor doing the work  
2. sortingByKey -- it is much better as it distributes the processing time in spark ....it takes less time than global sorting   
  
  
  
  
We can use regular expression to read the data from logs file  
\*\*\*\*\*\*\*\*\*\*\*Regular expression can be VERY IMPORTANT\*\*\*\*\*\*\*\*\*  
  
  
1. To sort data we can use top()  
  
so whatever data passed to top...it would be converted from RDD to an array and sorted ...... it can be used as  .top(10) will give top 10 values in decreasing order.  
  
Drawback of top() -- It first convert RDD to Array  
  
  
  
2.The better way is using sortByKey  
  
It works in (key,value) fashion and sortby the key  
  
for eg... productFilter.map(rec => (rec.split(',')(4).toFloat ,rec )).sortByKey(false).take(5).foreach(println)  
  
It is much faster as sortByKey doesn't convert RDD to array ...it gives RDD as output  
  
  
  
3. takeOrdered(n , ordering logic)  
  
eg..    
productFilter.takeOrdered(5).foreach(println)  
It will sort each record alphabatically .....so to sort by the value...we need to provide ordering logic  
  
It can be written as   
productFilter.takeOrder(5)(Ordering[Float].on(x => -x.split(",")(4).toFloat)).foreach(println)  
  
here -x is used for sorting in reverse order ....and Ordering is scala code  
  
  
--------------  
sortByKey  
--------------  
  
The key can be a composite key ...  
ie..  productFilter.map(rec => { r=rec.split(",") ((r(1).toInt,r(4).toFloat ), rec )}).sortByKey(false).take(5).foreach(println)  
  
You can also do sorting on the basis of two different postive and negative  
  
ie..  productFilter.map(rec => { r=rec.split(",") ((r(1).toInt,-r(4).toFloat ), rec )}).sortByKey().take(5).foreach(println)  
  
  
  
  
  
  
  
----------------------------------------  
Scala Based List sorting APIs  
  
1. sortBy  
  
For a list l  
  
l.sortBy(k => k.split(",")(4).toFloat)  
  
  
  
  
  
  
FlatMap vs Map  
  
  
For eg after groupBy result is given a list  
so if we want output as list we use Map ---- one input one output  
if we want individual elements from each list...we use FlatMap ----one input multiple output

**Exercise**

Q- get the top n products in each category

Val categories = sc.textFile(“E:\\STUDY\\CCA175\\spark\\data\\retail\_db\\categories\\part-00000”)

val l =("Bike & Skate Shop", Iterable("933,42,Nike VR\_S Covert Driver,,179.99,http://images.acmesports.sports/Nike+VR\_S+Covert+Driver",

"934,42,Callaway X Hot Driver,,0.0,http://images.acmesports.sports/Callaway+X+Hot+Driver",

"935,42,TaylorMade RocketBallz Stage 2 Driver,,169.99,http://images.acmesports.sports/TaylorMade+RocketBallz+Stage+2+Driver",

"936,42,Cleveland Golf Classic XL Driver,,119.99,http://images.acmesports.sports/Cleveland+Golf+Classic+XL+Driver",

"937,42,Cobra AMP Cell Driver - Orange,,169.99,http://images.acmesports.sports/Cobra+AMP+Cell+Driver+-+Orange"))

**def topNProducts(rec: (String, Iterable[String]), topN: Int): Iterable[(String, String)] = {**

**rec.\_2.toList.sortBy(k => -k.split(",”)(4).toFloat).take(topN).map(r => (rec.\_1, r))**

**}**

*IF you want to get top N priced products*

*You have to first store top n distinct priced products and its function will be*

**def topNPricedProducts(rec: (String, Iterable[String]), topN: Int): Iterable[(String, String)] = {**

**val list=rec.\_2.toList**

**val prices= list.map(r => r.split(",")(4).toFloat)**

**val topNPrice=prices.sortBy(k => -k).distinct.take(topN)**

**val sortedList= list.sortBy(k => -k.split(",")(4).toFloat)**

**sortedList.filter(rec => topNPrice.contains(rec.split(",")(4).toFloat)).take(topN).map(r => (rec.\_1, r))**

**}**

val products = sc.textFile("/public/retail\_db/products")

val productsFiltered = products.filter(rec => rec.split(",")(4) != "")

val productsMap = productsFiltered.map(rec => (rec.split(",")(1).toInt, rec))

val categories = sc.textFile("/public/retail\_db/categories").

map(rec => (rec.split(",")(0).toInt, rec.split(",")(2)))

val productsJoin = productsMap.

join(categories).

map(rec => (rec.\_2.\_2, rec.\_2.\_1))

val productsGroupByCategory = productsJoin.groupByKey()

productsGroupByCategory.

flatMap(rec => topNProducts(rec, 3)).

collect().foreach(println)