Session 18: RDD'S CONTD. & INTRODUCTION TO DATAFRAMES

Assignment 18.3

Student Name: Abarajithan SA

Course: Big Data Hadoop & Spark Training

Start Date: 2017-09-09

End Date: 2017-11-26

Assignment 18.3– RDD'S IN SPARK, spark RDD operations and Dataframes.

Contents

Introduction

In this assignment, we are going to perform some basic Spark RDD operation functions with the given problem statement.

Dataset

Use the dataset given below:

https://drive.google.com/drive/folders/0B_P3pWagdIrrVThBaUdVSUtzbms

S18_Dataset_Holidays.txt, S18_Dataset_User_details.txt and S18_Dataset_Transport.txt.

- The above dataset has the data column wise, **userID**, **Destination**, **arrival**, **travel mode**, **travel distance** and the **year** in the file S18_Dataset_Holidays.txt.
- The dataset S18_Dataset_User_details.txt has columns **user ID, name** and the **age.**
- The dataset S18_Dataset_Transport.txt has columns as **travel mode** and the **cost** respectively.

S18_Dataset_Holidays.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_Holidays.txt
1,CHN,IND,airplane,200,1990
2,IND,CHN,airplane,200,1991
3,IND,CHN,airplane,200,1992
4,RUS,IND,airplane,200,1990
5,CHN,RUS,airplane,200,1992
6,AUS,PAK,airplane,200,1991
7,RUS,AUS,airplane,200,1990
8,IND,RUS,airplane,200,1991
9,CHN,RUS,airplane,200,1992
10,AUS,CHN,airplane,200,1993
1,AUS,CHN,airplane,200,1993
2,CHN,IND,airplane,200,1993
3,CHN,IND,airplane,200,1993
4,IND,AUS,airplane,200,1991
5,AUS,IND,airplane,200,1992
6,RUS,CHN,airplane,200,1993
7,CHN,RUS,airplane,200,1990
8,AUS,CHN,airplane,200,1990
9,IND,AUS,airplane,200,1991
10,RUS,CHN,airplane,200,1992
1,PAK,IND,airplane,200,1993
2,IND,RUS,airplane,200,1991
3,CHN,PAK,airplane,200,1991
4,CHN,PAK,airplane,200,1990
5,IND,PAK,airplane,200,1991
6,PAK,RUS,airplane,200,1991
7,CHN,IND,airplane,200,1990
8,RUS,IND,airplane,200,1992
9,RUS,IND,airplane,200,1992
10,CHN,AUS,airplane,200,1990
1,PAK,AUS,airplane,200,1993
```

S18_Dataset_User_details.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_User_details.txt
1,mark,15
2,john,16
3,luke,17
4,lisa,27
5,mark,25
6,peter,22
7,james,21
8,andrew,55
9,thomas,46
[acadgild@localhost hadoop]$
```

S18_Dataset_Transport.txt,

```
[acadgild@localhost hadoop]$ cat S18_Dataset_Transport.txt
airplane,170
car,140
train,120
ship,200[acadgild@localhost hadoop]$ ls -l
total 69060
```

Problem Statement

1. Considering age groups of **< 20 , 20-35, 35 >** ,Which age group spends the most Amount of money travelling.
2. What is the amount spent by each age-group, every year in travelling?

Before that, we are loading the dataset into the spark context,

1. ***val baseRDD1 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")***
2. ***val baseRDD2 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Transport.txt")***
3. ***val baseRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")***

Importing the singleton object for controlling the storage of an RDD,

4. ***import org.apache.spark.storage.StorageLevel***
5. ***baseRDD1.persist(StorageLevel.MEMORY_ONLY)***
6. ***baseRDD2.persist(StorageLevel.MEMORY_ONLY)***
7. ***baseRDD3.persist(StorageLevel.MEMORY_ONLY)***

```
scala> val baseRDD1 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Holidays.txt")
baseRDD1: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[58] at textFile at <console>:27

scala> val baseRDD2 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_Transport.txt")
baseRDD2: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_Transport.txt MapPartitionsRDD[60] at textFile at <console>:27

scala> val basRDD3 = sc.textFile("/home/acadgild/hadoop/S18_Dataset_User_details.txt")
basRDD3: org.apache.spark.rdd.RDD[String] = /home/acadgild/hadoop/S18_Dataset_User_details.txt MapPartitionsRDD[62] at textFile at <console>:27

scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> baseRDD1.persist(StorageLevel.MEMORY_ONLY)
res26: baseRDD1.type = /home/acadgild/hadoop/S18_Dataset_Holidays.txt MapPartitionsRDD[58] at textFile at <console>:27

scala> baseRDD2.persist(StorageLevel.MEMORY_ONLY)
res27: baseRDD2.type = /home/acadgild/hadoop/S18_Dataset_Transport.txt MapPartitionsRDD[60] at textFile at <console>:27

scala> baseRDD3.persist(StorageLevel.MEMORY_ONLY)
res28: baseRDD3.type = /home/acadgild/Assignment-18/S18_Dataset_User_details.txt MapPartitionsRDD[56] at textFile at <console>:26
```

The given dataset's have been loaded and we are creating the tuple RDD columns wise in the spark context,

We are loading the dataset's in the name of **holidays**, **transport** and user **RDD's**.

Task -1 - Considering age groups of < 20, 20-35, 35 >, which age group spends the most amount of money travelling.

In Order to considering particular age groups, we are using a below if, else logic to define a RDD **AgeMap** which gives you a set of age groups,

val AgeMap = user.map(x=>x._1->

| {

| if(x._3<20)

| "20"

| else if(x._3>35)

| "35"

| else "20-35"

| })

```
scala> val AgeMap = user.map(x=>x._1->
     | {
     | if(x._3<20)
     | "20"
     | else if(x._3>35)
     | "35"
     | else "20-35"
     | })
AgeMap: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[37] at map at <console>:30
```

1. **val userMap = holidays.map(x => x._4 -> (x._1,x._5))**
2. **val transportmap = transport.map(x=> x._1 -> x._2)**
3. **val joinCost = userMap.join(transportmap)**
4. **val calCost = joinCost.map(x => x._2._1._1 -> x._2._1._2 * x._2._2)**
5. **val groupCost = calCost.groupByKey().map(x => x._1 -> x._2.sum)**
6. **val groupAgeCost = AgeMap.join(groupCost).map(x => x._2._1 -> x._2._2)**
7. **val finalCost = groupAgeCost.groupByKey().map(x => x._1 -> x._2.sum)**
8. **val maxVal = finalCost.sortBy(x => -x._2).first()**

Step-1- we are mapping the travel mode, user ID & the distance from the **holidays** RDD.

```
scala> userMap.foreach(println)
(airplane,(1,200))
(airplane,(2,200))
(airplane,(3,200))
(airplane,(4,200))
(airplane,(5,200))
(airplane,(6,200))
(airplane,(7,200))
(airplane,(8,200))
(airplane,(9,200))
(airplane,(10,200))
(airplane,(1,200))
(airplane,(2,200))
(airplane,(3,200))
(airplane,(4,200))
(airplane,(5,200))
(airplane,(6,200))
(airplane,(7,200))
(airplane,(8,200))
(airplane,(9,200))
(airplane,(10,200))
(airplane,(1,200))
(airplane,(2,200))
(airplane,(3,200))
(airplane,(4,200))
(airplane,(5,200))
(airplane,(6,200))
(airplane,(7,200))
(airplane,(8,200))
(airplane,(9,200))
(airplane,(10,200))
(airplane,(1,200))
(airplane,(5,200))
```

Step -2 – the new RDD has been created **transportmap** and the screen shot shown below,

```
scala> transportmap.foreach(println)
(airplane,170)
(car,140)
(train,120)
(ship,200)
```

Step -3 – we are joining the 2 RDD's we created in the step 1 and the step 2.

Step -4 – we are multiplying the distance and the total cost and making the **userID** as key and the multiplied value as value

Step -5 – the multiplied values are summed for each user ID.

```
scala> joinCost.foreach(println)
(airplane,((1,200),170))
(airplane,((2,200),170))
(airplane,((3,200),170))
(airplane,((4,200),170))
(airplane,((5,200),170))
(airplane,((6,200),170))
(airplane,((7,200),170))
(airplane,((8,200),170))
(airplane,((9,200),170))
(airplane,((10,200),170))
(airplane,((1,200),170))
(airplane,((2,200),170))
(airplane,((3,200),170))
(airplane,((4,200),170))
(airplane,((5,200),170))
(airplane,((6,200),170))
(airplane,((7,200),170))
(airplane,((8,200),170))
(airplane,((9,200),170))
(airplane,((10,200),170))
(airplane,((1,200),170))
(airplane,((2,200),170))
(airplane,((3,200),170))
(airplane,((4,200),170))
(airplane,((5,200),170))
(airplane,((6,200),170))
(airplane,((7,200),170))
(airplane,((8,200),170))
(airplane,((9,200),170))
(airplane,((10,200),170))
(airplane,((1,200),170))
(airplane,((5,200),170))
```

step -4

```
scala> calCost.foreach(println)
(1,34000)
(2,34000)
(3,34000)
(4,34000)
(5,34000)
(6,34000)
(7,34000)
(8,34000)
(9,34000)
(10,34000)
(1,34000)
(2,34000)
(3,34000)
(4,34000)
(5,34000)
(6,34000)
(7,34000)
(8,34000)
(9,34000)
(10,34000)
(1,34000)
(2,34000)
(3,34000)
(4,34000)
(5,34000)
(6,34000)
(7,34000)
(8,34000)
(9,34000)
(10,34000)
(1,34000)
(5,34000)
```

step -5

```
scala> groupCost.foreach(println)
(4,102000)
(1,136000)
(6,102000)
(3,102000)
(7,102000)
(9,102000)
(8,102000)
(10,102000)
(5,136000)
(2,102000)
```

Step – 6 – we have joined the AgeMap RDD with the RDD created in the previous step. Now, the age group will be taken from the AgeMap RDD and grouped according to the ages using the user ID. **Step – 7 –** the Value is grouped based on the age group.

```
scala> groupAgeCost.foreach(println)
(20-35,102000)
(20,136000)
(20-35,102000)
(20,102000)
(20-35,102000)
(35,102000)
(35,102000)
(35,102000)
(20-35,136000)
(20,102000)
```

step 6 & 7 respectively

```
scala> finalCost.foreach(println)
(20,340000)
(20-35,442000)
(35,306000)
```

Step-8 – at this step, we achieved the required result by using the function first() by sorting the data in the previous step.

Expected output

```
scala> val maxVal = finalCost.sortBy(x => -x._2).first()
maxVal: (String, Int) = (20-35,442000)
```

Task -2 - What is the amount spent by each age-group, every year in travelling?

The codes used,

1. **val UserHolidays = holidays.map(x => x._4 -> (x._1,x._5,x._6))**
2. **val usertransport = transport.map(x=>x._1->x._2)**
3. **val join1 = UserHolidays.join(usertransport)**
4. **val cost_dist_amt = join1.map(x=>x._2._1._1->(x._2._1._3,x._2._1._2*x._2._2))**
5. **val join2 = AgeMap.join(cost_dist_amt).map(x=>(x._2._1,x._2._2._1)->x._2._2._2)**
6. **val ExpEachAgeGroup = join2.groupByKey().map(x=>x._1->x._2.sum)**

Step -1 – we are creating a tuple RDD where travel mode as Key, userID,distance & year as value by mapping the **holidays** RDD.

```
scala> val UserHolidays = holidays.map(x=>x._4->(x._1,x._5,x._6))
UserHolidays: org.apache.spark.rdd.RDD[(String, (Int, Int, Int))] = MapPartitionsRDD[69] at map at <console>:30

scala> UserHolidays.foreach(println)
(airplane,(1,200,1990))
(airplane,(2,200,1991))
(airplane,(3,200,1992))
(airplane,(4,200,1990))
(airplane,(5,200,1992))
(airplane,(6,200,1991))
(airplane,(7,200,1990))
(airplane,(8,200,1991))
(airplane,(9,200,1992))
(airplane,(10,200,1993))
(airplane,(1,200,1993))
(airplane,(2,200,1993))
(airplane,(3,200,1993))
(airplane,(4,200,1991))
(airplane,(5,200,1992))
(airplane,(6,200,1993))
(airplane,(7,200,1990))
(airplane,(8,200,1990))
(airplane,(9,200,1991))
(airplane,(10,200,1992))
(airplane,(1,200,1993))
(airplane,(2,200,1991))
(airplane,(3,200,1991))
(airplane,(4,200,1990))
(airplane,(5,200,1991))
(airplane,(6,200,1991))
(airplane,(7,200,1990))
(airplane,(8,200,1992))
(airplane,(9,200,1992))
(airplane,(10,200,1990))
(airplane,(1,200,1993))
(airplane,(5,200,1994))
```

Step -2 – creating a tuple RDD **usertransport** by mapping the previously created RDD **transport.**

```
scala> val usertransport = transport.map(x=>x._1->x._2)
usertransport: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[70] at map at <console>:30

scala> usertransport.foreach(println)
(airplane,170)
(car,140)
(train,120)
(ship,200)
```

Step-3- joining the 2 RDD's created previously.

```
scala> val join1 = UserHolidays.join(usertransport)
join1: org.apache.spark.rdd.RDD[(String, ((Int, Int, Int), Int))] = MapPartitionsRDD[73] at join at <console>:38

scala> join1.foreach(println)
(airplane,((1,200,1990),170))
(airplane,((2,200,1991),170))
(airplane,((3,200,1992),170))
(airplane,((4,200,1990),170))
(airplane,((5,200,1992),170))
(airplane,((6,200,1991),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1991),170))
(airplane,((9,200,1992),170))
(airplane,((10,200,1993),170))
(airplane,((1,200,1993),170))
(airplane,((2,200,1993),170))
(airplane,((3,200,1993),170))
(airplane,((4,200,1991),170))
(airplane,((5,200,1992),170))
(airplane,((6,200,1993),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1990),170))
(airplane,((9,200,1991),170))
(airplane,((10,200,1992),170))
(airplane,((1,200,1993),170))
(airplane,((2,200,1991),170))
(airplane,((3,200,1991),170))
(airplane,((4,200,1990),170))
(airplane,((5,200,1991),170))
(airplane,((6,200,1991),170))
(airplane,((7,200,1990),170))
(airplane,((8,200,1992),170))
(airplane,((9,200,1992),170))
(airplane,((10,200,1990),170))
(airplane,((1,200,1993),170))
(airplane,((5,200,1994),170))
```

Step -4 – In this step, we are eliminating the travel mode and mapping the user ID with the year and the multiplied value of the distance and the cost.

Step -5 – we are creating the **AgeMap** RDD using the if, else logic in order to categories the Age Group.

val AgeMap = user.map(x=>x._1->

| {

| if(x._3<20)

| "20"

| else if(x._3>35)

| "35"

| else "20-35"

| })

```
scala> val AgeMap = user.map(x=>x._1->
     | {
     | if(x._3<20)
     | "20"
     | else if(x._3>35)
     | "35"
     | else "20-35"
     | })
```

```
scala> val cost_dist_amt = join1.map(x=>x._2._1._1->(x._2._1._3,x._2._1._2*x._2._2))
cost_dist_amt: org.apache.spark.rdd.RDD[(Int, (Int, Int))] = MapPartitionsRDD[74] at map at <console>:40

scala> cost_dist_amt.foreach(println)
(1,(1990,34000))
(2,(1991,34000))
(3,(1992,34000))
(4,(1990,34000))
(5,(1992,34000))
(6,(1991,34000))
(7,(1990,34000))
(8,(1991,34000))
(9,(1992,34000))
(10,(1993,34000))
(1,(1993,34000))
(2,(1993,34000))
(3,(1993,34000))
(4,(1991,34000))
(5,(1992,34000))
(6,(1993,34000))
(7,(1990,34000))
(8,(1990,34000))
(9,(1991,34000))
(10,(1992,34000))
(1,(1993,34000))
(2,(1991,34000))
(3,(1991,34000))
(4,(1990,34000))
(5,(1991,34000))
(6,(1991,34000))
(7,(1990,34000))
(8,(1992,34000))
(9,(1992,34000))
(10,(1990,34000))
(1,(1993,34000))
(5,(1994,34000))
```

AgeMap output,

```
scala> AgeMap.foreach(println)
(1,20)
(2,20)
(3,20)
(4,20-35)
(5,20-35)
(6,20-35)
(7,20-35)
(8,35)
(9,35)
(10,35)
```

Step – 6 – now we are joining the **AgeMap** & and the **cost_dist_amt** and mapping in such a way that we need to get the **userID** gets its own age group categorization.

Step -7 – in the step 7, we achieve the expected output by using the function **groupByKey** and summing the value we have got in the previous steps year, age group wise.

```
scala> val join2 = AgeMap.join(cost_dist_amt).map(x=>(x._2._1,x._2._2._1)->x._2._2._2)
join2: org.apache.spark.rdd.RDD[((String, Int), Int)] = MapPartitionsRDD[78] at map at <console>:48

scala> join2.foreach(println)
((20-35,1990),34000)
((20-35,1991),34000)
((20-35,1990),34000)
((20,1990),34000)
((20,1993),34000)
((20,1993),34000)
((20,1993),34000)
((20-35,1991),34000)
((20-35,1993),34000)
((20-35,1991),34000)
((20,1992),34000)
((20,1993),34000)
((20,1991),34000)
((20-35,1990),34000)
((20-35,1990),34000)
((20-35,1990),34000)
((35,1992),34000)
((35,1991),34000)
((35,1992),34000)
((35,1991),34000)
((35,1990),34000)
((35,1992),34000)
((35,1993),34000)
((35,1992),34000)
((35,1990),34000)
((20-35,1992),34000)
((20-35,1992),34000)
((20-35,1991),34000)
((20-35,1994),34000)
((20,1991),34000)
((20,1993),34000)
((20,1991),34000)
```

Expected output,

```
scala> val ExpEachAgeGroup = join2.groupByKey().map(x=>x._1->x._2.sum)
ExpEachAgeGroup: org.apache.spark.rdd.RDD[((String, Int), Int)] = MapPartitionsRDD[80] at map at <console>:50

scala> ExpEachAgeGroup.foreach(println)
((35,1992),136000)
((20,1991),102000)
((20,1993),170000)
((20-35,1990),170000)
((35,1993),34000)
((20-35,1991),136000)
((20-35,1994),34000)
((20,1990),34000)
((20-35,1993),34000)
((20,1992),34000)
((20-35,1992),68000)
((35,1990),68000)
((35,1991),68000)
```