Spark Assignment 18.3

Given the below 3 dataset

1) S18_Dataset_Holidays.txt as a text file (user_id, src, dest, travel_mode, distance, year_of_travel)
2) S18_Dataset_Transport.txt as a text file (travel_mode, cost_per_unit)
3) S18_Dataset_User_details.txt as a text file (user_id, src, dest, travel_mode, distance, year_of_travel)

Solve the below mentioned problem statement in spark rdd.

This assignment is done in the spark shell within the acadgildVM.

Steps Followed:

To Solve the problems first I've combined the datas from the above 3 files to one, using the common keys. Used travel_mode as a key to combine transport and holidays data. Used user_id to combine user_details and holidays.

1) Copied the dataset file in the path /home/acadgild/spark/S18_Dataset_Holidays.txt Then read the text file by using sc.textfile as below.

   ```
   val holidays = sc.textFile("/home/acadgild/spark/S18_Dataset_Holidays.txt")
   ```

2) Then created a holiday_tuple rdd by using map function over the holidays rdd with travel_mode as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

   ```
   val holiday_tuple = sc.parallelize(holidays.map(line => line.split(",")).map(x => ((x(3)), (x(0).toInt,x(1), x(2),x(4).toInt,x(5).toInt))).collect)
   ```

Spark-shell output

```
scala> val holidays = sc.textFile("/home/acadgild/spark/S18_Dataset_Holidays.txt")
holidays: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_Holidays.txt MapPartitionsRDD[112] at textFi
le at <console>:24

scala> val holiday_tuple =  sc.parallelize(holidays.map(line => line.split(",")).map(x => ((x(3)), (x(0).toInt,x(1), x(2),
x(4).toInt,x(5).toInt))).collect)
holiday_tuple: org.apache.spark.rdd.RDD[(String, (Int, String, String, Int, Int))] = ParallelCollectionRDD[115] at paralle
lize at <console>:26
```

3) Copied the dataset file in the path /home/acadgild/spark/S18_Dataset_ Transport.txt Then read the text file by using sc.textfile as below.

val transport = sc.textFile("/home/acadgild/spark/S18_Dataset_Transport.txt")

4) Then created a transport_tuple rdd by using map function over the transport rdd with travel_mode as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

val transport_tuple = sc.parallelize(transport.map(line => line.split(",")).map(x => ((x(0)), (x(1).toInt))).collect)

Spark-shell output

```
scala> val transport = sc.textFile("/home/acadgild/spark/S18_Dataset_Transport.txt")
transport: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_Transport.txt MapPartitionsRDD[117] at text
File at <console>:24

scala> val transport_tuple =  sc.parallelize(transport.map(line => line.split(",")).map(x => ((x(0)), (x(1).toInt))).colle
ct)
transport_tuple: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[120] at parallelize at <console>:26
```

---

5) Then using join function joined the holiday_tuple and transport_tuple as below as both had travel_mode as key.From the transport_tuple taken the cost_per unit and multiplied with distance, to calculate the expenses for each of the travel.

val holiday_trans = holiday_tuple.join(transport_tuple).map {

case ((travel_mode),((user_id,src,dest,distance,year_of_travel),cost_per_unit)) => (user_id,src,dest,travel_mode,distance,year_of_travel,cost_per_unit *distance)

}

Spark-shell output

```
scala> val holiday_trans = holiday_tuple.join(transport_tuple).map {
     |   case ((travel_mode),((user_id,src,dest,distance,year_of_travel),cost_per_unit)) => (user_id,src,dest,travel_mode,
distance,year_of_travel,cost_per_unit *distance  )
     | }
holiday_trans: org.apache.spark.rdd.RDD[(Int, String, String, String, Int, Int, Int)] = MapPartitionsRDD[124] at map at <c
onsole>:32

scala> val travel_details = sc.parallelize(holiday_trans.map(x =>(((x._1)), ((x._2),(x._3),(x._4),(x._5),(x._6),(x._7))))).
collect)
travel_details: org.apache.spark.rdd.RDD[(Int, (String, String, String, Int, Int, Int))] = ParallelCollectionRDD[126] at p
arallelize at <console>:34
```

6) Next,To combine the holiday_trans with user data,I've modified the holiday_trans and kept the key as user_id

```
val travel_details = sc.parallelize(holiday_trans.map(x =>((x._1)), ((x._2),(x._3),(x._4),(x._5),(x._6),(x._7)))).collect)
```

Spark-shell output

```
scala> val travel_details = sc.parallelize(holiday_trans.map(x =>(((x._1)), ((x._2),(x._3),(x._4),(x._5),(x._6),(x._7))))).
collect)
travel_details: org.apache.spark.rdd.RDD[(Int, (String, String, String, Int, Int, Int))] = ParallelCollectionRDD[126] at p
arallelize at <console>:34
```

7) Copied the dataset file in the path /home/acadgild/spark/ S18_Dataset_User_details .Then read the text file by using sc.textfile as below.

```
val user_details = sc.textFile("/home/acadgild/spark/S18_Dataset_User_details.txt")
```

8) Then created a user_tuple rdd by using map function over the user_details rdd with user_id as key. Then created a parallelized collection by using parallelize method over the mapped tuple.

```
val user_tuple = sc.parallelize(user_details.map(line => line.split(",")).map(x => ((x(0).toInt), (x(1), x(2).toInt))).collect)
```

Spark-shell output

```
scala> val user_details = sc.textFile("/home/acadgild/spark/S18_Dataset_User_details.txt")
user_details: org.apache.spark.rdd.RDD[String] = /home/acadgild/spark/S18_Dataset_User_details.txt MapPartitionsRDD[128] a
t textFile at <console>:24

scala> val user_tuple = sc.parallelize(user_details.map(line => line.split(",")).map(x => ((x(0).toInt), (x(1), x(2).toInt
))).collect)
user_tuple: org.apache.spark.rdd.RDD[(Int, (String, Int))] = ParallelCollectionRDD[131] at parallelize at <console>:26
```

---

9) Then using join function joined the user_tuple and travel_details as below as both had user_id as key.

```
val user_travel = user_tuple.join(travel_details).map {

 case (user_id,((name,age),(src,dest,travel_mode,distance,year_of_travel,charges)))
=> (user_id,name,age,src,dest,travel_mode,distance,year_of_travel,charges )

}
```

The user_travel tuple, will have items in the following order

user_id,name,age,src,dest,travel_mode,distance,year_of_travel,charges

user_id ->1,

name -> 2,

age ->3,

src ->4,

dest -> 5,

travel_mode -> 6,

distance -> 7,

year_of_travel -> 8,

charges -> 9

10) The above Rdd, user_travel is the final tuple, which is a combination of user_details, transport and holidays. It will be used to solve multiple problems, so we are storing it in the memory.

```
import org.apache.spark.storage.StorageLevel

user_travel.persist(StorageLevel.MEMORY_ONLY)
```

Spark-shell output

```
scala> val user_travel = user_tuple.join(travel_details).map {
     |   case (user_id,((name,age),(src,dest,travel_mode,distance,year_of_travel,charges))) => (user_id,name,age,src,dest,
travel_mode,distance,year_of_travel,charges )
     | }
user_travel: org.apache.spark.rdd.RDD[(Int, String, Int, String, String, String, Int, Int, Int)] = MapPartitionsRDD[135] a
t map at <console>:40

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

scala> user_travel.persist(StorageLevel.MEMORY_ONLY)
res36: user_travel.type = MapPartitionsRDD[135] at map at <console>:40
```

Problem Statement:

1) ***Considering age groups of < 20 , 20-35, 35 > ,Which age group spends the most amount of money travelling..***

For this problem, first I've calculated the age group, using the below function.

```
scala>  def ageRangeCalculate(x: Int) :String={
     |  if (x < 20)
     |   "<20"
     |  else if (x > 35)
     |   ">35"
     |  else "20-35";
     |  }
ageRangeCalculate: (x: Int)String
```

```
def ageRangeCalculate(x: Int) :String={
 if (x < 20)
  "<20"
 else if (x > 35)
  ">35"
 else "20-35";
}
```

To find which age-group has travelled the most every year., we are calling the map function over the user_travel rdds 3rd element which refers to age . We are passing the age value to the function ageRangeCalculate, calculating the age group , then setting the agegroup as key and setting user_travel rdds 9th element, expenses as value for each item.

Then calling the reduceBykey, where the values for each key are aggregated using the given reduce function. Here we are adding the values for each key and finally calling the collect action.

```
val ag5=user_travel.map (x=> ageRangeCalculate(x._3) ->(x._9)).reduceByKey((x,y) => (x +
y)).collect
```

In group_most_spent , we are finding the maximum value in the value column of ag5.., which will return which age-group has spent maximum amount

```
val group_most_spent = sc.parallelize(ag5).collect.maxBy(_._2)
```

Output

```
group_most_spent: (String, Int) = (20-35,442000)
```

Spark-shell ouput

```
scala> val ag5=user_travel.map (x=> ageRangeCalculate(x._3) ->(x._9)).reduceByKey((x,y) => (x + y)).collect
ag5: Array[(String, Int)] = Array((20-35,442000), (>35,306000), (<20,340000))

scala> val group_most_spent = sc.parallelize(ag5).collect.maxBy(_._2)
group_most_spent: (String, Int) = (20-35,442000)
```

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

For this problem, first I've calculated the age group, using the below function.

```
scala>  def ageRangeCalculate(x: Int) :String={
     |  if (x < 20)
     |   "<20"
     |  else if (x > 35)
     |   ">35"
     |  else "20-35";
     |  }
ageRangeCalculate: (x: Int)String
```

```
def ageRangeCalculate(x: Int) :String={
 if (x < 20)
  "<20"
 else if (x > 35)
  ">35"
 else "20-35";
}
```

To find which age-group has spent the most every year, we are calling the map function over the user_travel rdds 8th and 3rd element which refers to the travel_year, age . We are passing the age value to the function ageRangeCalculate, calculating the age group , then setting the year and agegroup as key and setting user_travel rdds 9th element, expenses as value for each item.

Then calling the reduceBykey, where the values for each key are aggregated using the given reduce function. Here we are adding the values for each key and finally calling the collect action.

```
val ag6=user_travel.map (x=> x._8 ->({
ageRangeCalculate(x._3)
 }) ->(x._9) ).reduceByKey((x,y) => (x + y)).collect
```

```
val amount_spent_every_year = ag6.map {
  case ((year,age_range),expenses) => ((year),(age_range,expenses) )
}
```

In amount_spent_every_year, we are calling the map function over the ag6, and creating a key with year, age_range and expenses as value for the key, since we need to find the amount spent by each age group for every year.

Output

amount_spent_every_year: Array[(Int, (String, Int))] = Array((1993,(<20,170000)), (1990,(>35,68000)), (1994,(20-35,34000)), (1991,(<20,102000)), (1992,(<20,34000)), (1991,(20-35,136000)), (1990,(<20,34000)), (1992,(>35,136000)), (1990,(20-35,170000)), (1993,(>35,34000)), (1992,(20-35,68000)), (1993,(20-35,34000)), (1991,(>35,68000)))

Spark-shell ouput