Big Data Systems

Assignment Spark

Polyzogopoulos Pavlos (BAPT1730)

Giorgos Alexiou

Athens University of Economics and Business

Athens, 2018

Question 1

a)

- Create a new RDD called airportRDD by loading the “airports” text file in the current directory.
  val airportRDD = sc.textFile("/user/b-analytics/assignment/airports.text")

- Filter out lines which contain the word “greek” in order to find all the airports which are located in Greece.
  val greekairportsRDD = airportRDD.filter(line=>line.contains("Greece"))

- Create a new RDD by splitting the lines of the greekairportsRDD using the comma as the delimiter.
  val greekairportsRDD2 = greekairportsRDD.map(line=>line.split(","))

- Create an RDD which holds only the required information (airport's name, the city's name and the airports , IATA/FAA code).
  val greekairportsRDD3 = greekairportsRDD2.map(vals=>(vals(1),vals(2),vals(4)))

- Use the collect action to collect the results
  greekairportsRDD3.collect().foreach(println)

b)

- Create a new RDD called “airportRDD” by loading the “airports” text file in the current directory.
  val airportRDD = sc.textFile("/user/b-analytics/assignment/airports.text")

- First split the RDD using the comma as the delimiter. Then take the sixth column which holds the latitude information and apply the limitations.
  Val airportlat=airportRDD.filter(_.split(",")(6).toDouble>37).filter(_.split(",")(6).toDouble<39)
  .map(_.split(","))

- Create the final RDD which contains only the required columns (airport's name, Main city served by airport and the airport's latitude).
  val airportlatfinal = airportlat.map(vals=>(vals(1),vals(2),vals(6)))

Question 2

a)

- First, you are going to create two RDDs for the nasa_19950701 and the nasa_19950801 tsv files in the current directory.
```
val nasajulyRDD = sc.textFile("/user/b-analytics/assignment/nasa_19950701.tsv")
val nasaaugustRDD = sc.textFile("/user/b-analytics/assignment/nasa_19950801.tsv")
```

- Then, split the files using "\t" as the delimiter and create new RDDs containing key-value pairs of the form: (word,1). The reduceByKey will group each of these pairs by key. This will help us in the next steps in order not to have the same keys appearing more than once.
```
val nasajulyRDD1  = nasajulyRDD.map(line=>line.split("\t"))
.map(word => (word(0), 1)).reduceByKey((a,b)=>a+b)

val nasaaugustRDD1 = nasaaugustRDD.map(line=>line.split("\t"))
.map(word => (word(0), 1)).reduceByKey((a,b)=>a+b)
```

- Join these two RDDs together to get a collective set. The join function combines the two datasets (K,V) and (K,W) together and get (K, (V,W)).
```
val nasajoinedRDD = nasajulyRDD1.join(nasaaugustRDD1)
```

- Filter out the keys which contain the word "host". This will filter out the header lines. The final RDD contains the hosts which are accessed on BOTH days.
```
val nasajoinedRDD1 = nasajoinedRDD.map(t=>(t._1)).filter((t)=>t !="host")
```

- Use the collect action to collect the results.
```
nasajoinedRDD1.collect().foreach(println)
```

b)

- Create a new RDD called "wordRDD" by loading the "word count" text file.
```
val wordRDD  = sc.textFile("/user/b-analytics/assignment/word_count.text")
```

- Do a WordCount on this RDD so that the results are (K,V) pairs of (word, count).
```
val wordcountRDD = wordRDD.flatMap(line=>line.split("
")).map(word=>(word,1)).reduceByKey((a,b)=>a+b)
```

- Transpose keys and values and sort by the total number of occurrences of each word in descending order. Finally, we will transpose again to get the required form of the RDD which is (word,total_number_of_occurrence_of_each_word).
```
val wordcountRDD1 = wordcountRDD.map(t=>(t._2,t._1))
val wordcountRDD2 = wordcountRDD1.sortByKey(false)
val wordcountRDD3 = wordcountRDD2.map(t=>(t._2,t._1))
```

- Use the collect action to collect the results.
```
WordcountRDD3. collect().foreach(println)
```

Question 3

a)

- First, we are going to create an RDD for the Real csv file in the current directory.
  ```
  val houseRDD = sc.textFile("/user/b-analytics/assignment/Real.csv")
  ```

- Create a new RDD called headerhouse which contains the first row (header lines) of the houseRDD and then filter out in order to remove them from the resulting RDD.
  ```
  val headerhouse = houseRDD.first()
  val houseRDD1 = houseRDD.filter(row=>row!=headerhouse)
  ```

- Then, we will split the file using the comma as the delimiter and create a houseRDD2 containing key-value pairs of the form: (number_of_bedrooms,price).
  ```
  val houseRDD2 = houseRDD1.map(r => (r.split(",")(3).toInt, r.split(",")(2).toDouble))
  ```

- Finally, create a houseaverage containing key-value pairs of the form: (number_of_bedrooms,average_price).
  ```
  val houseaverage = houseRDD2.groupByKey().map(t=>(t._1, t._2.sum/t._2.size))
  ```

- Use the collect action to collect the results.
  ```
  houseaverage.collect().foreach(println)
  ```

b)

- Create the sql context.
  val sqlContext = new org.apache.spark.sql.SQLContext(sc)

- Import a library to convert an RDD to a SchemaRDD.
  import sqlContext.implicits._

- Create the case class in Scala which defines the schema of the table.
  case class house(Location:String,Price:Double,Pricesqf:Double)

- First, we are going to create an RDD for the Real csv file in the current directory.
  val housesql = sc.textFile("/user/b-analytics/assignment/Real.csv")

- Create a new RDD called headerhouseRDD which contains the first row (header lines) of the houseRDD and then filter it out in order to remove them from the resulting RDD.
  val housesql2 = housesql.first()
  val housesql3 = housesql.filter(row=>row!=housesql2)

- Then, we will split the file using the comma as the delimiter.
  val housesql4 = housesql3.map(_.split(","))

- Create a new RDD called housesql5 of the form (Location, Price, Price /SQ.ft) for each house. We will also convert the last two values to Double in order to calculate the average and the max values.
  val housesql5 = housesql4.map(p=>house(p(1),p(2).trim.toDouble,p(6).trim.toDouble)).toDF()

- Register the RDD as a table.
  housesql5.registerTempTable("House")

- Create the final RDD called housefinal of the form (Location,average_PriceSQ/ft,max(Price). This RDD will be created by running SQL statements using the sql method provided by the SQLContext.
  val housefinal = sqlContext.sql("SELECT Location,AVG(Pricesqf), MAX(Price) FROM House GROUP BY Location")