YCBS-257 - Data at Scale

Workshop 9

In Memory Data Processing

General Instructions:

Apache Spark is an open source parallel processing framework for running large-scale data analytics applications across clustered computers. It can handle both batch and real-time analytics and data processing workloads. Spark was designed for executing complex multi-stage applications, like machine learning algorithms, and interactive ad hoc queries.

This series of exercises in Scala and PySpark are designed to familiarize you with Apache Spark ecosystem.

To perform the exercises you will need a Scala and / or PySpark command interpreter (Python for Spark). You can use either the command line or Apache Zeppelin (recommended) to perform the workshop.

Start your Cloudera QuickStart VM to complete the workshop

Online resources:

https://zeppelin.apache.org/
https://spark.apache.org/

Exercise 1: WordCount in Scala

In this exercise, we will approach the word count problem by using Scala with Spark. You might use Zeppelin to enter, test and run your code or use the command line interface.

1. Create a new directory `/sparklab` on HDFS
2. Put the file `5000-8.txt` inside this directory
3. Create a new Zeppelin note `wordcount`

Zeppelin Interpreter	`%spark`
Load the file from HDFS	`val textFile = sc.textFile("/sparklab/5000-8.txt")`
Count the number of elements	`textFile.count()`
Show the 1st line	`textFile.first()`

Find how many 'The' occurrences in the document	`val linesWithSpark = textFile.filter(line => line.contains("The")).count()`
Get the longest line	`textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)`

The same using the Max function Import Java Math library to be able to use the Max function	`import java.lang.Math`
	
	`textFile.map(line => line.split(" ").size).reduce((a, b) => Math.max(a, b))`

Create a flat structure using flatMap() function and run a ReduceByKey to get the final result	`val wordCounts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)` `wordCounts.collect()`

Exercise 2: Babies names analysis using Spark

In this exercise you will analyze the babies' names in the state of New York given between 2007 and 2016.

1. Copy the `Baby_Names__Beginning_2007.csv` file into `/sparklab` HDFS directory
2. Create a new Zeppelin note: `babies`

Spark Interpreter	`%spark`

RDD creation	`val babyNames = sc.textFile("/sparklab/Baby_Names__Beginning_2007.csv")`
	
	`babyNames.count()`

Print the header line	`babyNames.first()`

Split the columns and do a distinct count (First Name column)	`val rows = babyNames.map(line => line.split(","))`
	
	`rows.map(row => row(2)).distinct.count`

Count all occurrences of the first name "DAVID"	`val davidRows = rows.filter(row => row(1).contains("DAVID"))`
	
	`davidRows.count()`

Limit results to departments where first name has been assigned more than 10 times	`davidRows.filter(row => row(4).toInt > 10).count()`
Perform a distinct count of departments where 'David' appear more than 10 times	`davidRows.filter(row => row(4).toInt > 10).map( r => r(2) ).distinct.count`

Display the 100 most common first names	`val names = rows.map(name => (name(1),1)).reduceByKey((a,b) => a + b).sortBy(-_._2) //.foreach ( println _)`
	
	`names.take(100).foreach (println _)`

Remove the header line and Count all rows and print the 20 first names	`val filteredRows = babyNames.filter(line => !line.contains("Count")).map(line => line.split(","))` `filteredRows.map ( n => (n(1), n(4).toInt)).reduceByKey((a,b) => a + b).sortBy(-_._2).take(20).foreach (println _)`

Exercise 3: Explore Spark RDDs

In this exercise you will count negative comments from a public survey. As a data source we will use *Rotten Tomatoes Kaggle Dataset* which is a sentiments analysis about movies.

1. Copy the train.tsv file into /sparklab HDFS directory
2. Create a new Zeppellin note: rddSpark

%spark
sc

1. Load data into a new RDD

```
val data = sc.textFile("/sparklab/train.tsv")
```

2. Run Action method to 'force' Spark to evaluate the previous operation

```
data.getClass
```

3. Split the loaded data according to the separator character

```
val reviews = data.map(_.split("\t"))
```

4. Count the loaded data in the RDDs

```
reviews.count()
```

```
// Long: 156061
```

5. Count the words of the loaded document

```
val wordCounts = reviews.flatMap(x => x(2).split(" ")).map((_,
1)).reduceByKey((a, b) => a + b).sortBy(_._2, false)
```

```
wordCounts.take(5) // get top 5
```

```
reviews.flatMap(x => x(2).split(" ")).take(5)
```

6. Let's look at the third column

```
reviews.flatMap(x => x(2).split(" ")).map((_, 1)).take(5)
```

```
// MapReduce Style
reviews.flatMap(x => x(2).split(" ")).map((_, 1)).reduceByKey((a, b) => a +
b).take(5)
```

7. Remove headers and count negative comments

```
val trueReviews = reviews.filter(x => x(0) != "PhraseId")
```

```
val negativeReviews = trueReviews.filter(x => x(3).toInt == 0)
```

```
val negWordCounts = negativeReviews.flatMap(x => x(2).split(" ")).map((_,
1)).reduceByKey((a, b) => a + b).sortBy(_._2, false)
```

```
negWordCounts.take(50)
```

Exercise 4: Querying data using Spark SQL

In this exercise you will write a few lines of code in Scala to query a Spark RDD with SQL language. As a data source you will use the Stations_2017.csv file

1. Copy the customers.txt file into /sparklab HDFS directory
2. Create a new Zeppelin note: sparksql

Zeppelin Interpreter	`%spark`

Create a DataFrame of Station objects from the dataset text file	`// Create a DataFrame of Station objects from the dataset text file.` `val StationsDF = sqlContext.read.format("csv").option("header","true").option("inferSchema","true").load("/user/cloudera/spark/Stations_2017.csv")`
Register DataFrame as a table	`// Register DataFrame as a table.` `dfStations.registerTempTable("Stations")`
Display the content of DataFrame	`// Display the content of DataFrame` `dfStations.show()`
Print the DF schema	`// Print the DF schema` `dfStations.printSchema()`
Select Station name column	`// Select Station name column` `dfStations.select("name").show()`
Select Station code and name columns	`// Select Station code and name columns` `dfStations.select("code", "name").show()`
Select a Station by id	`// Select a Station by id` `dfStations.filter(dfStations("code").equalTo(6080)).show()`
Switch to SQL interpreter	`%sql`

Print all rows in the table	`select * from Stations`
	`or`
	`spark.sql("Select * from Stations Limit 10").show`

Print Stations count	`select count(*) from Stations`
	`or`
	`spark.sql("Select count(*) from Stations").show`

Exercise 5: Rating movies

In this exercise we will analyze the ratings attributed by the audience to movies. The dataset has 4 columns:

Column 1:	User ID
Column 2:	Movie ID
Column 3:	Rating
Column 4:	Timestamp

1. Copy the `u.data` file into /sparklab HDFS directory
2. Create a new Zeppellin note: ratings

Spark Interpreter	`%pyspark`

RDD creation	`import collections`
	
	`lines = sc.textFile('/user/cloudera/spark/u.data')`

Extract the 3rd column and count ratings (from 1 à 5)	`ratings = lines.map(lambda x : x.split()[2])`
	
	`res = ratings.countByValue()`

Sort the result and print it	`sortedres = collections.OrderedDict(sorted(res.items()))`
	`for key,value in sortedres.items():` `  print ("%s %i" %(key,value))`
	1 6110 2 11370 3 27145 4 34174 5 21201

Extracting the movie ID and creating an Pair-RDD that contains the movie ID and value = 1	`movies = lines.map(lambda x : (int(x.split()[1]), 1))`
	
	`movies.take(10)`

	[(242, 1), (302, 1), (377, 1), (51, 1), (346, 1), (474, 1), (265, 1), (465, 1), (451, 1), (86, 1)]

Count movies	`moviesCount = movies.reduceByKey(lambda x , y : x + y )`
	
	`moviesCount.take(10)`

	[(2, 131), (4, 209), (6, 26), (8, 219), (10, 89), (12, 267), (14, 183), (16, 39), (18, 10), (20, 72)]

To sort the RDD on 'value' we flip the RDD	`flippedMovies = moviesCount.map(lambda x : (x[1], x[0])).sortByKey()`
	
	`flippedMovies.take(10)`

Print the length of the final result	`finalResult = sortedMovies.collect()`
	
	`len(finalResult)`

	1682

Transform the RDD into a data frame and register the dataframe as an in-memory table	`resultDF = sc.parallelize(finalResult).toDF()`
	
	`sqlContext.registerDataFrameAsTable(resultDF, "MoviesRating")`

Switch to SQL interpreter and query the table	`%sql`
	
	`select * from MoviesRating order by _1 desc limit 15`