E10-1

This Notebook illustrates the use of "MAP-REDUCE" to calculate averages from the data contained in nsedata.csv.

Task 1

You are required to review the code (refer to the SPARK document where necessary), and add comments / markup explaining the code in each cell. Also explain the role of each cell in the overall context of the solution to the problem (ie. what is the cell trying to achieve in the overall scheme of things). You may create additional code in each cell to generate any debug output that you may need to complete this exercise.

Task 2

You are required to write code to solve the problem stated at the end this Notebook

Submission

Create and upload a PDF of this Notebook. BEFORE CONVERTING TO PDF and UPLOADING ENSURE THAT YOU REMOVE / TRIM LENGTHY DEBUG OUTPUTS . Short debug outputs of up to 5 lines are acceptable.

In [1]:

```
import findspark
findspark.init()

# Line1:
# Here we are importing the library findspark of Python which will help you to locate and
use  PySpark installed in your system
# Line 2:
# We are using the function init from the library findspark.The function helps you to fin
g=d the location in which Pyspark is installed in your machine
# and also initialises it to work with the current Python environment
```

In [2]:

```
import pyspark
from pyspark.sql.types import *
#Line 1:
#Here we are importing the pyspark library of Python
#Line 2:
#Here we are accessing all the modules of types from sql from the bigger library PySpark.
This module helps us to use different data types while
#using different DataFrames in PySpark.
```

In [3]:

```
sc = pyspark.SparkContext(appName="E10")
#Here we are setting up a Spark Context with the Application name E10
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLeve
l).
23/10/31 13:28:52 WARN NativeCodeLoader: Unable to load native-hadoop library for your pl
atform... using builtin-java classes where applicable
23/10/31 13:28:54 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting p
ort 4041.
```

In [4]:

```
rdd1 = sc.textFile("/home/hduser/spark/nsedata.csv")
#Here we are creating a RDD file. RDDs are the fundamental data structure in Apache Spark
, and they allow for distributed data processing.
```

```
#We are reading the text file from the folder mentioned in the code above and storing it
to rdd1 as a RDD file
#Each line from the text file becomes an element in the rdd
```

In [5]:

```
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x)
#Here we are using the Lambda function and filter transformation to remove all the elemen
ts of the RDD with the word "SYMBOL" in it.
```

In [6]:

```
rdd2 = rdd1.map(lambda x : x.split(","))
#In this code given above,we are using the map transformation and Lambda function for spl
ittting the elements of the RDD using the delimiter ",".
```

In [7]:

```
# Helper comment!: The goal is to find out the mean of the OPEN prices and the mean of th
e CLOSE price in one batch of tasks ...
```

In [8]:

```
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
#We are creating a key-value pair by using Lambda and map transformation .In rdd_open,the
key is formed by concatenating x[0] with _open
#and in rdd_close we are concatenating x[0] with _close for making the key.For the values
we are adding the corresponding
#open and close values from the csv file using x[2] and x[5] in float form
```

In [9]:

```
rdd_united = rdd_open.union(rdd_close)
#Here we are using the Union transformation to combine both rdd_open and rdd_close
```

In [10]:

```
reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y)
#We are using reduceByKey transformation to combine the values with same key
```

In [11]:

```
temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol = sc.parallelize(temp1.items())
# It maps each element (key-value pair) to a new key-value pair where the key is the stoc
k symbol x[0] and set the value to 1.
#The above step effectively converts the data to a format where each symbol is associated
with the value 1.
#sc.parallelize(temp1.items()) is used to convert the dictionary items into an RDD.
#Each of the key-value pairs represent a stock symbol and its corresponding count.
```

In [12]:

```
symbol_sum_count = reducedByKey.join(countOfEachSymbol)
#Here we are joining the two RDD's reducedByKey and countOfEachSymbol,based on their keys
by using the join transformation.
```

In [13]:

```
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1]))
#Here we are using the map transformation on the symbol_sum_count RDD to calculate the av
erages by dividing
#the sum of open and close prices by the count of occurrences for each stock symbol.
```

In [14]:

```
averagesSorted = averages.sortByKey()
```

```
#Here we are using the sortByKey transformation to sort the averages RDD by its keys.
```

In [15]:

```
averagesSorted.saveAsTextFile("/home/hduser/spark/averages")
#Here we are using the saveAsTextFile action to save the averagesSorted RDD to a text fil
e to the destination mentioned above.
```

In [ ]:

```
```

Review the output files generated in the above step and copy the first 15 lines of any one of the output files into the cell below for reference. Write your comments on the generated output

In [16]:

```
with open("/home/hduser/spark/averages/part-00000", "r") as file:
    for i, line in enumerate(file):
        if i < 15:
            print(line)
        else:
            break

#The below listed are the first 15 lines from part 00000 of the saved file
#As we can see,we can easily get to know the average profit/loss made by trading the give
n stock from the given average values of opening and closing
#stock price of the data
```

```
('20MICRONS_close', 53.004122877930484)

('20MICRONS_open', 53.32489894907032)

('3IINFOTECH_close', 18.038803556992725)

('3IINFOTECH_open', 18.17417138237672)

('3MINDIA_close', 4520.343977364591)

('3MINDIA_open', 4531.084518997574)

('3RDROCK_close', 173.2137755102041)

('3RDROCK_open', 173.18316326530612)

('8KMILES_close', 480.73622047244095)

('8KMILES_open', 481.63858267716535)

('A2ZINFRA_close', 18.609433962264156)

('A2ZINFRA_open', 18.73553459119497)

('A2ZMES_close', 89.69389505549951)

('A2ZMES_open', 90.46271442986883)

('AANJANEYA_close', 441.84030249110316)
```

Task 2 - Problem Statement

Using the MAP-REDUCE strategy, write SPARK code that will create the average of HIGH

prices for all the traded companies, but only for any 3 months of your choice. Create the appropriate (K,V) pairs so that the averages are simultaneously calculated, as in the above example. Create the output files such that the final data is sorted in descending order of the company names.

In [7]:

```
rdd1=sc.textFile("/home/hduser/spark/nsedata.csv")
```

In [8]:

```
rdd1=rdd1.filter(lambda x:"SYMBOL" not in x)
```

In [9]:

```
rdd2=rdd1.map(lambda x:x.split(","))
```

In [10]:

```
rdd_high=rdd2.map(lambda x: (x[0]+"_high_average",float(x[3])))
elements = rdd_high.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
[Stage 0:>                                                          (0 + 1) / 1]
('20MICRONS_high_average', 37.75)
('3IINFOTECH_high_average', 45.3)
('3MINDIA_high_average', 3439.95)
```

In [11]:

```
reducedByKey_2 = rdd_high.reduceByKey(lambda x,y: x+y)

elements = reducedByKey_2.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
[Stage 1:=============================================================(5 + 0) / 5]
('AARTIDRUGS_high_average', 396832.7999999998)
('ABB_high_average', 1063958.7500000002)
('ABBOTINDIA_high_average', 2425757.700000001)
```

In [12]:

```
temp1_2 = rdd_high.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol_2 = sc.parallelize(temp1_2.items())
elements = countOfEachSymbol_2.take(3)

# Print or examine the elements
for element in elements :
    print(element)
```

```
('20MICRONS_high_average', 1237)
('3IINFOTECH_high_average', 1237)
('3MINDIA_high_average', 1237)
```

In [13]:

```
symbol_sum_count_2 = reducedByKey_2.join(countOfEachSymbol_2)
temporary_2 = symbol_sum_count_2.sortByKey()
# elements = symbol_sum_count_2.take(3)
# # Print or examine the elements
```

```
# for element in elements:
#     print(element)

elements = temporary_2.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
[Stage 13:=================================================>        (6 + 1) / 7]
```

```
('20MICRONS_high_average', (67564.34999999998, 1237))
('3IINFOTECH_high_average', (22960.199999999997, 1237))
('3MINDIA_high_average', (5694089.6499999985, 1237))
```

In [14]:

```
averages_2 = symbol_sum_count_2.map(lambda x : (x[0], x[1][0]/x[1][1]))

elements = averages_2.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
('ABBOTINDIA_high_average', 1961.0005658852072)
('ACC_high_average', 1257.7121665319323)
('ANGIND_high_average', 26.880166821130672)
```

In [15]:

```
averagesSorted_2 = averages_2.sortByKey()
# Assuming you have an RDD named 'my_rdd'
elements = averagesSorted_2.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
[Stage 26:=================================================>        (6 + 1) / 7]
```

```
('20MICRONS_high_average', 54.61952303961195)
('3IINFOTECH_high_average', 18.561196443007272)
('3MINDIA_high_average', 4603.144421988681)
```

In [59]:

```
averagesSorted_2.saveAsTextFile("/home/hduser/spark/averages_high_all_months")
```

In [23]:

```
#The above file contains the average of high values of all the companies from all the dat
es
```

In [16]:

```
rdd_2=sc.textFile("/home/hduser/spark/nsedata.csv")
rdd_2=rdd_2.filter(lambda x:"SYMBOL" not in x)
# rdd_high_2=rdd_21.map(lambda x: (x[0]+"_high",float(x[3])))
```

In [29]:

```
temp_sample=rdd_2.filter(lambda x:("OCT-2014" or "NOV-2014" or "DEC-2014") in x)
temp=temp_sample.map(lambda x:x.split(","))
temp_high=temp.map(lambda x : (x[0],float(x[3])))
temp_by_key=temp_high.reduceByKey(lambda x,y : x+y)

elements = temp_by_key.take(3)
# Print or examine the elements
```

```
for element in elements:
    print(element)
```

```
[Stage 31:=============================================>           (4 + 1) / 5]
```

```
('ABAN', 10900.05)
('ABGSHIP', 4123.4)
('ACE', 581.9)
```

In [30]:

```
temp_2 = temp_by_key.map(lambda x : (x[0],1)).countByKey()
counts = sc.parallelize(temp_2.items())
```

In [39]:

```
symbol_highsum_count = temp_by_key.join(counts)
avg_high = symbol_highsum_count.map(lambda x : (x[0] , x[1][0]/x[1][1]))
avgs_desc = avg_high.sortByKey(False)
#Setting the sortByKey to false will sort it by keys in the descending order
```

In [40]:

```
elements = avgs_desc.take(3)
# Print or examine the elements
for element in elements:
    print(element)
```

```
[Stage 63:==================================>                    (4 + 2) / 7]
```

```
('ZYLOG', 140.15)
('ZYDUSWELL', 11556.099999999999)
('ZUARIGLOB', 1681.6000000000001)
```

In [41]:

```
avgs_desc.saveAsTextFile("/home/hduser/spark/average_highest_3months")
```

In [ ]:

```
```