23B3307-E10-1

November 19, 2024

0.1 E10-1

Nirav Bhattad (23B3307)

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

0.1.1 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 what each cell is trying to achieve in the overall scheme of things. You may create additional code in each cell to generate 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.

[1]:
```
import findspark # import findspark module to locate spark in the system
findspark.init() # initialize spark
```

[2]:
```
import pyspark # import pyspark module
from pyspark.sql.types import * # import all the functions from pyspark.sql.
 ↪types module
```

[ ]:
```
sc = pyspark.SparkContext(appName="E10") # create a spark context
```

[4]:
```
rdd1 = sc.textFile("nsedata.csv") # read the csv file into an rdd
```

[5]:
```
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x) # here we are using the Lambda␣
 ↪function and filter transformation to remove all the elements of the RDD␣
 ↪with the word "SYMBOL" in it.
```

[6]:
```
rdd2 = rdd1.map(lambda x : x.split(",")) # split the rdd by commas using a␣
 ↪lambda function and the map function
```

[7]:
```
# Helper comment!: The goal is to find out the mean of the OPEN prices and the␣
 ↪mean of the CLOSE price in one batch of tasks ...
```

[8]:
```
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2]))) # here we are using␣
↪the map function and a lambda function to create a new rdd with the␣
↪symbol_open and the open price as the key value pair.
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5]))) # here we are␣
↪using the map function and a lambda function to create a new rdd with the␣
↪symbol_close and the close price as the key value pair.
```

[9]:
```
rdd_united = rdd_open.union(rdd_close) # here we are using the union␣
↪transformation to combine the two rdds into one rdd.
```

[10]:
```
reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y) # here we are using the␣
↪reduceByKey transformation to reduce the rdd by key and sum the values.
```

[11]:
```
temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey() # here we are using the␣
↪map function and a lambda function to create a new rdd with the symbol and␣
↪the value 1 as the key value pair. We then use the countByKey function to␣
↪count the number of times each symbol appears in the rdd.
countOfEachSymbol = sc.parallelize(temp1.items()) # here we are using the␣
↪parallelize function to create an rdd from the dictionary items of the temp1␣
↪rdd.
```

[12]:
```
symbol_sum_count = reducedByKey.join(countOfEachSymbol) # here we are using the␣
↪join transformation to join the reducedByKey rdd and the countOfEachSymbol␣
↪rdd.
```

[13]:
```
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1])) # here we␣
↪are using the map function and a lambda function to create a new rdd with␣
↪the symbol and the average price as the key value pair.
```

[14]:
```
averagesSorted = averages.sortByKey() # here we are using the sortByKey␣
↪transformation to sort the rdd by key.
```

[15]:
```
averagesSorted.saveAsTextFile("./averages") # save the rdd to a text file
```

```
24/11/19 19:52:35 WARN GarbageCollectionMetrics: To enable non-built-in garbage
collector(s) List(G1 Concurrent GC), users should configure it(them) to
spark.eventLog.gcMetrics.youngGenerationGarbageCollectors or
spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
```

[16]:
```
sc.stop() # stop the spark context
```

0.1.2 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

[17]:
```
with open("./averages/part-00024", "r") as file:
    for i, line in enumerate(file):
        if i < 15:
            print(line, end='')
        else:
            break
```

```
('SHREYANIND_open', 25.703465346534657)
('SHREYAS_close', 102.53706030150751)
('SHREYAS_open', 102.4324539363484)
('SHRIASTER_close', 8.112228260869564)
('SHRIASTER_open', 8.002853260869566)
('SHRINATRAJ_close', 75.10040650406505)
('SHRINATRAJ_open', 75.13536585365853)
('SHRIRAMCIT_close', 1046.2627501012555)
('SHRIRAMCIT_open', 1046.2623572296482)
('SHRIRAMEPC_close', 81.96172190784152)
('SHRIRAMEPC_open', 82.73399353274051)
('SHYAMCENT_close', 8.672)
('SHYAMCENT_open', 8.718)
('SHYAMTEL_close', 32.51556184316896)
('SHYAMTEL_open', 32.732659660468876)
```

0.2 Task 2 - Problem Statement

Using the MAP-REDUCE method, write SPARK code that will create the average of HIGH prices for every traded company traded within any 3 continuous months of your choice. Create the appropriate (K,V) pairs so that the averages are simultaneously calculated for each company, as in the above example. Create the output files such that the final data is sorted in descending order of the company names.

[18]:
```
sc = pyspark.SparkContext(appName="E10") # create a spark context
```

[19]:
```
rdd1=sc.textFile("./nsedata.csv")
```

[20]:
```
rdd1=rdd1.filter(lambda x:"SYMBOL" not in x)
```

[21]:
```
rdd2=rdd1.map(lambda x:x.split(","))
```

[22]:
```
rdd_high=rdd2.map(lambda x: (x[0]+"_high_average",float(x[3])))
elements = rdd_high.take(3)

print(elements)
```

```
[Stage 0:>                                                          (0 + 1) / 1]
```

```
[('20MICRONS_high_average', 37.75), ('3IINFOTECH_high_average', 45.3),
('3MINDIA_high_average', 3439.95)]
```

[23]:
```
reducedByKey_2 = rdd_high.reduceByKey(lambda x,y: x+y)
elements = reducedByKey_2.take(3)

print(elements)
```

```
[Stage 1:>                                                          (0 + 5) / 5]
```

```
[('ABBOTINDIA_high_average', 2425757.700000001), ('ABCIL_high_average',
166873.4000000001), ('ACKRUTI_high_average', 87689.35)]
```

[24]:
```
temp1_2 = rdd_high.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol_2 = sc.parallelize(temp1_2.items())
elements = countOfEachSymbol_2.take(3)

print(elements)
```

```
[Stage 3:=============================================>           (4 + 1) / 5]
```

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

[25]:
```
symbol_sum_count_2 = reducedByKey_2.join(countOfEachSymbol_2)
temporary_2 = symbol_sum_count_2.sortByKey()
elements = temporary_2.take(3)

print(elements)
```

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

[26]:
```
averages_2 = symbol_sum_count_2.map(lambda x : (x[0], x[1][0]/x[1][1]))
elements = averages_2.take(3)

print(elements)
```

```
[('AFTEK_high_average', 8.742150170648463), ('APOLLOTYRE_high_average',
111.61240905416336), ('ASHOKLEY_high_average', 39.14737479806137)]
```

[27]:
```
averagesSorted_2 = averages_2.sortByKey()
elements = averagesSorted_2.take(3)

print(elements)
```

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

[28]:
```
averagesSorted_2.saveAsTextFile("./averages-2")
```

[29]:
```
rdd_2=sc.textFile("./nsedata.csv")
rdd_2=rdd_2.filter(lambda x:"SYMBOL" not in x)
```

[30]:
```
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(elements)
```

```
[Stage 32:>                                                         (0 + 5) / 5]
```

```
[('3MINDIA', 111974.0), ('8KMILES', 12297.65), ('ABBOTINDIA',
57560.149999999994)]
```

[31]:
```
temp_2 = temp_by_key.map(lambda x : (x[0],1)).countByKey()
counts = sc.parallelize(temp_2.items())
```

[32]:
```
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)
```

[33]:
```
elements = avgs_desc.take(3)
for element in elements:
    print(element)
```

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

[34]:
```
avgs_desc.saveAsTextFile("./averages-3")
```