22b1219-abhineet-1

October 31, 2023

0.1 E10-1

This Notebook illustrates the use of "MAP-REDUCE" to calculate averages from 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 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.

[ ]:
```
import findspark
findspark.init()
```

It intializes spark within the Jupyter Notebook. It is using the findspark library to locate the Spark installation and set up the necessary environment variables to work with Spark.

[ ]:
```
import pyspark
from pyspark.sql.types import *
```

In this cell, we are importing the necessary PySpark libraries. PySpark is used for distributed data processing, and this cell is setting up the environment for working with Spark.

[ ]:
```
sc = pyspark.SparkContext(appName="E10")
```

In this cell, we are creating a SparkContext, which is the entry point to the Spark cluster.

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

In this cell, we are reading the nsedata.csv file into an RDD (Resilient Distributed Dataset). RDD is the primary data abstraction in Spark for performing distributed data processing tasks. Now we have the data loaded into the rdd1 RDD, and can start applying transformations and actions to it for your MapReduce operation.

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

In this cell, we are filtering out any lines that contain the string "SYMBOL." The filter transformation is used to select specific elements from the RDD based on a given condition.

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

In this cell, we are applying a transformation to rdd1 to split each line into a list of values by splitting the line at the comma (,) delimiter.

This transformation prepares the data for the next steps, as we now have a list of values from each line that can be used for calculations in the MapReduce operation.

```
[ ]: # 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 ...
```

```
[ ]: rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
     rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
```

These transformations are preparing the data by extracting and formatting the open and close prices for each stock symbol, which will be used in the subsequent MapReduce operation to calculate averages.

```
[ ]: rdd_united = rdd_open.union(rdd_close)
```

In this cell, we are using the union transformation to combine two RDDs, rdd_open and rdd_close, into a single RDD called rdd_united.

The purpose of this step is likely to have a single RDD containing all the data needed for the MapReduce operation. rdd_united will now have key-value pairs in the format "SYMBOL_open" and "SYMBOL_close" with their respective open and close prices.

```
[ ]: reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y)
```

In this cell, we are performing a reduce operation on the rdd_united RDD using the reduceByKey transformation. This is a key step in implementing the MapReduce operation to calculate averages.

In this case, the lambda function takes two values, x and y, and returns their sum (x + y). The resulting reducedByKey RDD will have keys like "SYMBOL_open" and "SYMBOL_close" and their corresponding summed values, which represent the total open and close prices for each stock symbol.

```
[ ]: temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey()
     countOfEachSymbol = sc.parallelize(temp1.items())
```

The purpose of these operations is to count the occurrences of each stock symbol (both open and close) in the rdd_united RDD. This count information will be useful for calculating averages or performing other statistical analyses based on the data.

```
[ ]: symbol_sum_count = reducedByKey.join(countOfEachSymbol)
```

The resulting symbol_sum_count RDD will have keys that represent stock symbols (e.g., "SYMBOL_open" or "SYMBOL_close"). The values will be tuples, where the first element is the sum of open and close prices for that stock symbol, and the second element is the count of occurrences of that symbol.

This is an important step for the MapReduce operation because it pairs the aggregated sum values with the count values, which can be used to calculate the average for each stock symbol in the next steps.

[ ]:
```
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1]))
```

The result is an RDD with key-value pairs where the key represents the stock symbol, and the value represents the calculated average for that symbol.

With this step, we have completed the MapReduce operation to calculate averages for each stock symbol.

[ ]:
```
averagesSorted = averages.sortByKey()
```

In this cell, we are sorting the averages RDD based on the keys, which represent the stock symbols. We are creating a new RDD named averagesSorted that contains the averages sorted by the stock symbols.

[ ]:
```
averagesSorted.saveAsTextFile("/home/hduser/spark/averages")
```

This line uses the saveAsTextFile action to save the contents of the averagesSorted RDD to the specified directory, "/home/hduser/spark/averages." The data will be saved as text files in this directory, with each text file containing a portion of the RDD data.

[ ]:
```
ss.stop()
```

In this final cell, we are stopping the SparkContext, which is essential to release resources and shut down the Spark application.

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

[ ]:
```
'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)
```

We have the average by company now.

0.2 Task 2 - Problem Statement

0.2.1 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.

[ ]:
```
import findspark
findspark.init()
import pyspark

sc = pyspark.SparkContext(appName="HighPriceAverages")

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

def filter_by_date(record):
    desired_date_range = ['2023-01', '2023-03']
    date = record.split(",")[1]
    return any(date.startswith(month) for month in desired_date_range)
filtered_rdd = rdd.filter(filter_by_date)
high_prices_rdd = filtered_rdd.map(lambda record: (record.split(",")[0],␣
 ↪float(record.split(",")[3])))
sum_count_rdd = high_prices_rdd.combineByKey(
    lambda value: (value, 1),
    lambda accumulator, value: (accumulator[0] + value, accumulator[1] + 1),
    lambda accumulator1, accumulator2: (accumulator1[0] + accumulator2[0],␣
 ↪accumulator1[1] + accumulator2[1])
)
average_rdd = sum_count_rdd.mapValues(lambda value: value[0] / value[1])
sorted_rdd = average_rdd.sortByKey(ascending=False)
sorted_rdd.saveAsTextFile("/home/hduser/spark/high_price_averages")
sc.stop()
```