ASSIGNMENT 1 - UE22CS343AB2 - BIG DATA

Introduction

This assignment introduces an alternative input method - JSON and multi-stage Map-Reduce.

Assignment Objectives and Outcomes

- Get familiar with map reduce
- Get familiar with aggregation using map reduce
- Working with hadoop

Ethical Practices

Please submit original code only. All solutions must be submitted through the portal. We will perform a plagiarism check on the code and you will be penalized if your code is found to be plagiarized.

Submission Deadline

9th September, 2024.

The portal uptimes will be communicated to you in due course. Please do not wait till the last minute to submit your code since wait times may be long.

Submission Links

Assignments need to be submitted on portal: https://www.bigdata-pesu-2024.tech (https://www.bigdata-pesu-2024.tech)

(the site is offline right now, the online timings will be communicated via groups and mails).

Submission Guidelines

You will need to make the following changes to your Python scripts before submitting them.

Include the following shebang at the top of your Python scripts.

```
#!/usr/bin/env python3
```

Convert your files to executable using the following command.

```
chmod +x *.py
```

Convert line breaks in DOS format to Unix format (this is necessary if you are coding on Windows without which your code will not run on our portal).

```
dos2unix *.py
```

Software/Languages to be used:

- `Python 3.10.x`
- `Hadoop v3.3.3+`

NOTE : STRICTLY NOT ALLOWED TO IMPORT ANY AND ALL PACKAGES IN MAPPER AND REDUCER. The packages that can be imported are mentioned below.

Specification

Task 1: Store Profit and Loss

Story Background:

You are given data from a retail chain in India that sells a wide range of products, including groceries, home appliances, beauty items, apparel and many more goods in multiple cities across the country. However, not all stores in the chain sell every product category. Each store has its own top-selling categories based on consumer demand. Only these top-selling categories will determine the profit gained or the loss incurred by a store.

Given the input, your task is to determine:

1. The number of stores that are profitable for each city.
2. The number of stores operating at a loss for each city.

Points to be Considered:

1. Sales data (Revenue and COGS) for a product category (top-selling or not) may or may not be recorded.
2. Net results (Profit or Loss) for each store is only calculated if there exists Sales data (Revenue and COGS) for atleast some top-selling category.
3. If Net results > 0, it is a profitable store, otherwise it's incurring a loss.

Mapper

Input : Array of JSON Objects

Here's a prettified version of an example JSON object:

```
{
  "city": "Bangalore",
  "store_id": "ST01293",
  "categories": [
    "Electronics",
    "Groceries"
  ],
  "sales_data": {
    "Electronics": {
      "revenue": 600000,
      "cogs": 500000
    },
    "Groceries": {
      "revenue": 250000,
      "cogs": 270000
    }
  }
}
```

Example Input :

```
[
{"city": "Bangalore", "store_id": "ST01293", "categories": ["Electronics", "Groceries"]
{"city": "Chennai", "store_id": "ST04567", "categories": ["Pharmacy and Health", "Kitch
{"city": "Mumbai", "store_id": "ST05432", "categories": ["Books and Magazines", "Pharma
{"city": "Mumbai", "store_id": "ST08345", "categories": ["Groceries"], "sales_data": {"
{"city": "Chennai", "store_id": "ST06789", "categories": ["Home Decor", "Apparel"], "sa
{"city": "Bangalore", "store_id": "ST09874", "categories": ["Apparel"], "sales_data": {
]
```

NOTE : **Do not** load the entire dataset as the test case we use will be extremely large and will crash the server. You will be blacklisted in that case.

Net Returns Formula = Revenue - COGS
COGS = Cost of Goods and Services

Reducer

Input : Same format as Mapper output

Output : Independent JSON Objects

- output of the reducer has the following keys : name of the city, number of profitable stores and number of stores at a loss.

Example Output:

```
{"city": "Bangalore", "profit_stores": 2, "loss_stores": 0}
{"city": "Chennai", "profit_stores": 1, "loss_stores": 1}
{"city": "Mumbai", "profit_stores": 2, "loss_stores": 0}
```

Sample Input

1. small_data.json (https://drive.google.com/file/d/1OdAIdUVKWQb1ndU_m_x6iWuAUiTS6ksG/view?usp=sharing)
2. expected_output_small_data.txt for the above input

```
{"city": "Ahmedabad", "profit_stores": 3, "loss_stores": 5}
{"city": "Bangalore", "profit_stores": 5, "loss_stores": 3}
{"city": "Chennai", "profit_stores": 3, "loss_stores": 5}
{"city": "Delhi", "profit_stores": 3, "loss_stores": 4}
{"city": "Hyderabad", "profit_stores": 3, "loss_stores": 5}
{"city": "Jaipur", "profit_stores": 3, "loss_stores": 5}
{"city": "Kanpur", "profit_stores": 0, "loss_stores": 8}
{"city": "Kolkata", "profit_stores": 8, "loss_stores": 1}
{"city": "Lucknow", "profit_stores": 8, "loss_stores": 0}
{"city": "Mumbai", "profit_stores": 4, "loss_stores": 4}
{"city": "Pune", "profit_stores": 2, "loss_stores": 5}
{"city": "Surat", "profit_stores": 3, "loss_stores": 5}
```

Test Dataset

Test your code with the following dataset once it passes the sample input dataset

1. **Input**: large_data.json (https://drive.google.com/file/d/1Eq3BtsX6rNwr2ho3LbnXWAirw1QGVe6v/view?usp=sharing)
2. **Output**: expected_output_large_data.txt (https://drive.google.com/file/d/1qQSd5WHha60TBm31WUPilsAdp63yZ9WH/view?usp=sharing)

All the files can be found here (https://drive.google.com/drive/folders/1273OnlOzqQTJ14AKAY_6TN-Jy_eogfJM?usp=sharing).

Instructions

1. Write a python mapper
   - Name : `mapper.py`
   - Read the specification for input and output as mentioned above
   - Only packages that can be imported are : json and sys
2. Write a python reducer to perform the aggregation

- Name : `reducer.py`
- Read the specification for input and output as mentioned above
- Only packages that can be imported are : json and sys

3. Test it out with the sample dataset given and check the expected output
4. Adhere to the submission guidelines

Testing instructions

Local testing

```
cat <path_to_dataset>.json | ./mapper.py | sort -k 1,1 | ./reducer.py
```

Hadoop testing

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-<hadoop_version>.jar \
-mapper "$PWD/mapper.py" \
-reducer "$PWD/reducer.py" \
-input  <path_to_input_in_hdfs> \
-output <path_to_output_folder_in_hdfs>
```

TASK 2: Load Balancer

Story Background:

A Company's team was assigned to predict the status codes that clients would receieve while hitting the company's endpoints, while taking into consideration the traffic received on a certain endpoint at a given time, as well as the reliability of the servers handling the requests. You are given a dataset with the information regarding all the requests made by clients during the day. The predicted status codes by the team are also provided. You are to validate the dataset given and evaluate each client's success metric.

Metadata for the Dataset:

Request ID (rX):

A unique identifier for each request. This is used to differentiate between different requests in the dataset.

Client ID (cX):

Identifies the client making the request. Each client is associated with a unique ID, which can be used to track the activity or behavior of individual clients.

Endpoint (endpoint):

Represents the specific service or function the request is accessing.

user/profile: Access or update user profile information.
user/settings: Modify user settings, such as privacy or notification preferences.
order/history: Retrieve the history of past orders.
order/checkout: Complete the checkout process for an order.
product/details: View detailed information about a specific product.
product/search: Search for products based on certain criteria.
cart/add: Add an item to the shopping cart.
cart/remove: Remove an item from the shopping cart.
payment/submit: Submit payment information to complete a purchase.
support/ticket: Create or view support tickets for customer service.

Each endpoint is associated with a price, which is used to calculate the total_price for each client. The price for all the endpoints is given below.

```
'user/profile': 100
'user/settings': 200
'order/history': 300
'order/checkout': 400
'product/details': 500
'product/search': 600
'cart/add': 700
'cart/remove': 800
'payment/submit': 900
'support/ticket': 1000
```

Timestamp (HH:MM:SS):

Indicates the time at which the request was made in hours, minutes, and seconds. This can be used for analyzing patterns in request timing or for sequencing events.

Downtime (No.of servers):

Reflects the no.of servers down. A value of 0.0 indicates no servers are down at that second.

Predicted Status Code:

Represents the status codes predicted by the Company's team for each client request. These predictions may or may not be correct.

500 - Internal Server Error
200 - Successful Request

Objective

Direct each incoming request to an available server assigned to the respective endpoint, given that each endpoint has 3 servers which may or may not be down at different timestamps. The goal is to determine the number of successfully predicted requests by the team for each client and calculate each client's total_price, based on the fixed price associated with each endpoint.

Points to be considered :

1. The requests in the same timestamp must be processed in the lexicographical order.
2. A Single client is only allowed to hit one endpoint at a given timestamp. Other requests are not processed and are not taken into consideration for evaluation.
3. The time taken to handle a client request by any server is 1 second.

Example

Input

The input file is a .txt file which is space seperated for request_id, client_id, endpoint, timestamp and no_of_servers_down. Space seperated for request_id and predicted_status_code.

Sample Input

```
request_id client_id endpoint timestamp no_of_servers_down

request_id predicted_status_code
```

```
r001 c01 user/profile 00:00:00 2.0        #200
r002 c02 user/profile 00:00:00 2.0        #500
r003 c03 user/settings 00:01:02           #200
r004 c01 order/history 00:01:05 2.0       #200
r005 c05 order/history 00:05:03 3.0       #500
r006 c03 order/history 00:05:03 3.0       #500
r007 c02 user/settings 00:07:00 1.0       #200
r008 c03 user/settings 00:07:05           #200
r009 c01 user/profile 00:08:00            #200
r010 c04 user/profile 00:08:05 3.0        #500

r001 500
r002 500
r003 200
r004 200
r005 500
r006 500
r007 200
r008 200
r009 500
r010 200
```

Note : The status codes given in '#' in the sample dataset above are the correctly evaluated status codes and are ONLY given for your reference. These values are NOT part of the actual datasets.

Output

```
client_id  successful_predictions/total_requests  total_price



c01 1/3 500
c02 2/2 200
c03 3/3 400
c04 0/1 0
c05 1/1 0
```

Metrics :

successful_predictions/total_requests: The ratio of status codes successfully predicted by the Company's team to the total number of processed requests for that client.

total_price: The total cost incurred by a client successfully hitting the endpoints.

Workflow Constraints :

1. You are obligated to use exactly three stages to account to this solution. (i.e. three pairs of Mappers and Reducers)

2. Do not import any external modules. Use modules only available in the default python package.
3. The output should be in space separated format.

Test Dataset

Test your code with the following datasets.
Dataset: Data Folder (https://drive.google.com/drive/folders/1T6ni23cq7JA23h7JAIMVqwI3XgYapDGK?usp=sharing)
The folder includes small and large dataset with expected outputs for both the files.

Testing your code

Here is a sample script on testing your multi-stage map reduce.

```
#!/usr/bin/env bash

# Clean up the intermediate HDFS directories which could have been created as a part of
hdfs dfs -ls /intermediate-1
if [[ $? == 0 ]]
then
        echo "Deleteing Intermediate-1 HDFS directory before starting job.."
        hdfs dfs -rm -r /intermediate-1
fi

hdfs dfs -ls /intermediate-2
if [[ $? == 0 ]]
then
        echo "Deleting Intermediate-2 HDFS directory before starting job.."
        hdfs dfs -rm -r /intermediate-2
fi

# Clean up the previously used output directory.
hdfs dfs -ls /task2/output
if [[ $? == 0 ]]
then
    echo "Deleting previous output directory"
    hdfs dfs -rm -r /task2/output
fi


echo "Initiating stage-1"
echo "============================================================"

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_1.py" \
-reducer "$PWD/reducer_1.py" \
-input /task2/dataset.txt \
-output /intermediate-1

echo "============================================================"
echo "Stage-1 done"
echo "Initiating stage-2"
echo "============================================================"
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_2.py" \
-reducer "$PWD/reducer_2.py" \
-input /intermediate-1/part-00000 \
-output /intermediate-2
echo "============================================================"
echo "Stage-2 done"
echo "Initializing stage-3"
echo "============================================================"

hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-mapper "$PWD/mapper_3.py" \
-reducer "$PWD/reducer_3.py" \
-input /intermediate-2/part-00000 \
-output /task2/output
```

```
echo "============================================================"
echo "Stage-3 done"
```

Note: Change Hadoop version as required
