NATIONAL CENTRE FOR SCIENTIFIC RESEARCH "DEMOKRITOS"

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
UNIVERSITY OF PELOPONNESE

MSc in Data Science

2021-2022

Big Data Management

Assignment 1

REPORT

Vitsas Alexandros-Konstantinos

Spiliakos Georgios

Contents

1 Task 1

1.1 Attribute types

1. *ID*: nominal, discrete
2. *Year_Birth*: numeric, interval-scaled
3. *Education*: nominal, discrete
4. *Marital_Status*: nominal, discrete
5. *Income*: numeric, interval-scaled
6. *Kidhome*: numeric, interval-scaled
7. *Teenhome*: numeric, interval-scaled
8. *Dt_Customer*: numeric, interval-scaled
9. *Recency*: numeric, interval-scaled
10. *Complain* nominal, binary
11. *MntWines*: numeric, ratio-scaled
12. *MntFruits*: numeric, ratio-scaled
13. *MntMeatProducts*: numeric, ratio-scaled
14. *MntFishProducts*: numeric, ratio-scaled
15. *MntSweetProducts*: numeric, ratio-scaled
16. *MntGoldProds*: numeric, ratio-scaled
17. *NumDealsPurchases*: numeric, ratio-scaled
18. *AcceptedCmp1*: nominal, binary
19. *AcceptedCmp2*: nominal, binary
20. *AcceptedCmp3*: nominal, binary

21. *AcceptedCmp4*: nominal, binary

22. *AcceptedCmp5*: nominal, binary

23. *Response*: nominal, binary

24. *NumWebPurchases*: numeric, ratio-scaled

25. *NumCatalogPurchases*: numeric, ratio-scaled

26. *NumStorePurchases*: numeric, ratio-scaled

27. *NumWebVisitsMonth*: numeric, ratio-scaled

Total numeric features: 16/27

Total nominal features: 10/27

1.2 Preprocessing

The preprocessing steps described below were implemented using a Python script, provided in the deliverable. The csv file produced as output of the script was used for the rest of the tasks.

Duplicates	: No duplicate rows found in the dataset.
Extra columns	: Four redundant columns found in the original .csv file with the data were dropped.
Outliers	: The columns *Year_Birth* and *Income* were used to remove outliers with respect to age and income. IQR method was used for the outlier removal.
Datetime object	: The values of the column *Dt_Customer* was turned into a Python datetime object.
Missing values	: The missing values of the *Income* column were filled with the mean of the rest of the values of this column.

NOTE: The column *Dt_Customer* was loaded as a *chararray* type in Pig and transformed to Pig Date type internally, for Task 4 only.

2 Task 2

For this task we have 1 Job.

Description: The purpose of this Job is to count how many occurrences of each educationType we have in our data and provide a total sum for each of the education types.

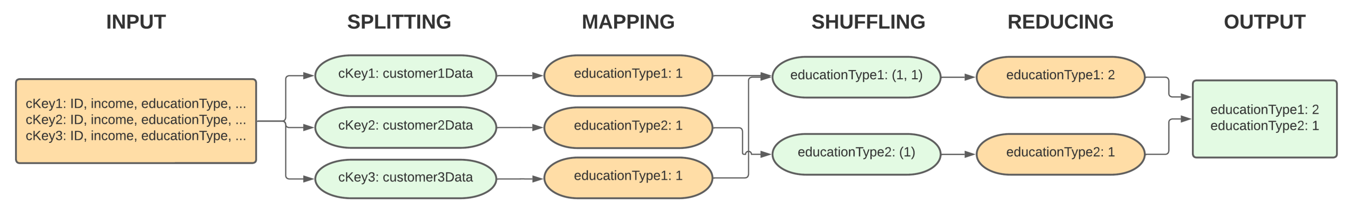
- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as shown in the .csv file.
  - KeyOUT: Returns the educationType (2nd Grade,Graduation,etc).
  - ValueOUT: Returns number "1".
- Reducer:
  - KeyIN: educationType (2nd Grade,Graduation,etc).
  - ValueIN: Array with number "1" values. (1,1,1,1,1, ...) returned from Shuffle for each education type.
  - KeyOUT: educationType (2nd Grade,Graduation,etc).
  - ValueOUT: Summed values of the ValueIN array.

```
class Mapper
    method Map(Customer cKey, Customer cValues)
        educationType = cValues[x]
        emit(educationType, count 1)


class Reducer
    method Reduce(Education educationType, Array eduCounts[1, 1,     ])
        sum=0
        for each x in eduCounts[1, 1, ...]:
            sum = sum + eduCounts[x]

        emit(Education educationType, count sum)
```

Example - Diagram :



3 Task 3

For this task we used 4 Jobs.

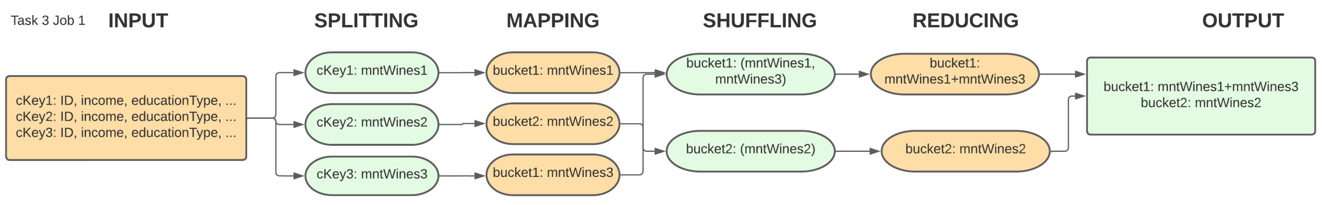
3.1 Job 1

Description: The purpose of first Job is to provide counters that have the total number of customers as well as the sums of the amount of money spent on wines. For this Job we used 4 buckets to distribute the task. The result will be 4 records that each will have the size(number of customers),sum(money spent on wines) of each bucket.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as shown in the .csv file.
  - KeyOUT: Returns the random bucket number (1,2,3,4).
  - ValueOUT: Returns the amount of money spent for wines for one customer.
- Reducer:
  - KeyIN: Bucket number (1,2,3,4).
  - ValueIN: Array with amounts spent for wine (19,391,3818, ...) returned from Shuffle for each bucket number.
  - KeyOUT: Total number of customers of the bucket.
  - ValueOUT: Summed values of the ValueIN array.

```
class SumCountMapper
    method Map(Customer cKey, Customer cValues)
        mntWines = cValues[x]
        bucket = Random (1-4)
        emit(bucket, mntWines)


class SumCountReducer
    method Reduce(Bucket b, Array mntWinesCounts [amount1, amount2, ...])
        sum=0
        size=0
        for each x in mntWinesCounts[amount1, amount2,      ]:
            sum = sum + mntWinesCounts[x]
            size++
        emit(Count size, Count sum)
```



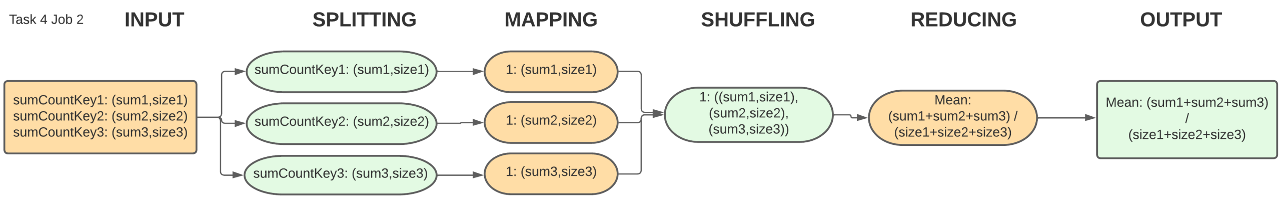
3.2 Job 2

Description: The purpose of second Job is to find the mean of amount of money spent on wines, based on the total number of customers and sums of mntWines provided by the 4 records of the first Job.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: The whole record of Job 1 (size,sum).
  - KeyOUT: Returns the number 1.
  - ValueOUT: Returns the whole record (size,sum) of Job 1.
- Reducer:
  - KeyIN: Number 1.
  - ValueIN: Array with the 4 (size,sum) records.
  - KeyOUT: Returns string word "Mean".
  - ValueOUT: Returns the sum/size for all the 4 records, which is the total Mean value.

```
class meanMapper
    method Map(SumCount key, SumCount values)
        emit(count 1, values)


class meanReducer
    method Reduce(count 1, values[(size1,sum1), (size2,sum2), ...])
        sum=0
        count=0
        for each v in values[(size1,sum1), (size2,sum2), ...]
            sum = sum + v.get(sum)
            size = count + v.get(size)
        emit("Mean", (sum/size))
```


Task 4 Job 2
INPUT
SPLITTING
MAPPING
SHUFFLING
REDUCING
OUTPUT
sumCountKey1: (sum1,size1)
sumCountKey2: (sum2,size2)
sumCountKey3: (sum3,size3)
sumCountKey1: (sum1,size1)
sumCountKey2: (sum2,size2)
sumCountKey3: (sum3,size3)
1: (sum1,size1)
1: (sum2,size2)
1: (sum3,size3)
1: ((sum1,size1), (sum2,size2), (sum3,size3))
Mean: (sum1+sum2+sum3) / (size1+size2+size3)
Mean: (sum1+sum2+sum3) / (size1+size2+size3)

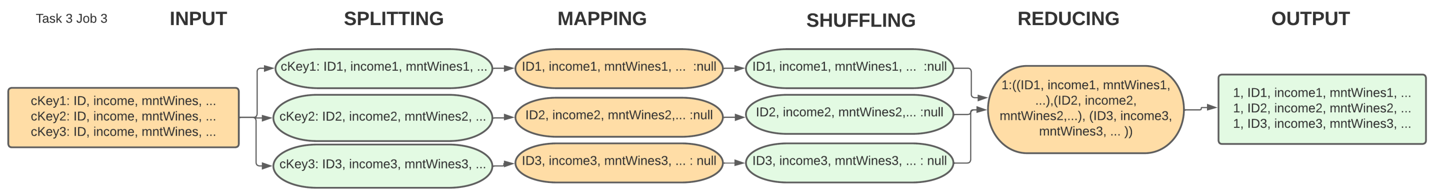
3.3 Job 3

Description: The purpose of third Job is to find which of the customers have spent more than (1.5 * Mean mntWines) and keep their information. By using the customer data as KeyOUT in our Mapper we can Sort and Shuffle with their values. The reducer is used to only write those customers to HDFS.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as shown in the .csv file.
  - KeyOUT: Customer who has spent > (1.5* Mean mntWines).
  - ValueOUT: Returns null.
- Reducer:
  - KeyIN: Customer with his data (ID,income,mntWines, etc).
  - ValueIN: Null.
  - KeyOUT: Returns number 1.
  - ValueOUT: Returns the customer data.

```
class CustomersMapper
    method Map(Customer cKey, Customer cValues[id, yearBirth, education, maritalStatus,
                income, mntWines])
        if (cValues.mntWines > (1.5 * TotalMean))
            emit(cValues, null)

class CustomersReducer
    method Reduce(Customer c, null values)
        emit(count 1, c)
```


Task 3 Job 3
INPUT
SPLITTING
MAPPING
SHUFFLING
REDUCING
OUTPUT
cKey1: ID, income, mntWines, ...
cKey2: ID, income, mntWines, ...
cKey3: ID, income, mntWines, ...
cKey1: ID1, income1, mntWines1, ...
cKey2: ID2, income2, mntWines2, ...
cKey3: ID3, income3, mntWines3, ...
ID1, income1, mntWines1, ... :null
ID2, income2, mntWines2,... :null
ID3, income3, mntWines3, ... : null
ID1, income1, mntWines1, ... :null
ID2, income2, mntWines2,... :null
ID3, income3, mntWines3, ... : null
1:((ID1, income1, mntWines1, ...),(ID2, income2, mntWines2,...), (ID3, income3, mntWines3, ...))
1, ID1, income1, mntWines1, ...
1, ID2, income2, mntWines2, ...
1, ID3, income3, mntWines3, ...

3.4 Job 4

Description: The purpose of fourth Job is keep only the desired data and provide ranking numbers to the customers.

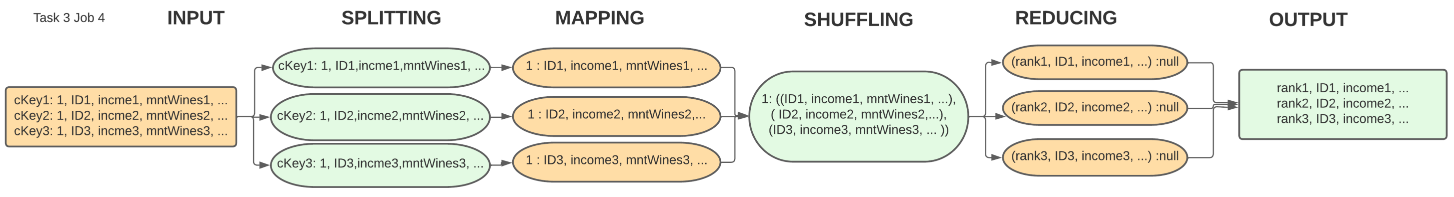
- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as they were returned from Job 3.

  - KeyOUT: Number 1. In order for Shuffle process to keep all customers in one array with the unique key Number 1.
  - ValueOUT: Customer information as they were returned from Job 3.

- Reducer:
  - KeyIN: Number 1.
  - ValueIN: Customer information as they were returned from Job 3.
  - KeyOUT: Customer information as they were returned from Job 3 and added the rank.
  - ValueOUT: Returns Null.

```
class RanksMapper
    method Map(Customer cKey, Customer cValues[id, yearBirth, education, maritalStatus,
                income, mntWines])
            emit(count 1, cValues)

class RankReducer
    method Reduce(Key 1, Customer cValues[id, yearBirth, education, maritalStatus,
                income, mntWines])
        ranker = 0
        for each c in cValues[id, yearBirth, education, maritalStatus, income, mntWines]
            ranker++
            c.rank = ranker
            emit(c, null)
```



Task 3 Job 4
INPUT
SPLITTING
MAPPING
SHUFFLING
REDUCING
OUTPUT
cKey1: 1, ID1, incme1, mntWines1, ...
cKey2: 1, ID2, incme2, mntWines2, ...
cKey3: 1, ID3, incme3, mntWines3, ...
cKey1: 1, ID1,incme1,mntWines1, ...
cKey2: 1, ID2,incme2,mntWines2, ...
cKey3: 1, ID3,incme3,mntWines3, ...
1 : ID1, income1, mntWines1, ...
1 : ID2, income2, mntWines2,...
1 : ID3, income3, mntWines3, ...
1: ((ID1, income1, mntWines1, ...),
(ID2, income2, mntWines2,...),
(ID3, income3, mntWines3, ...))
(rank1, ID1, income1, ...) :null
(rank2, ID2, income2, ...) :null
(rank3, ID3, income3, ...) :null
rank1, ID1, income1, ...
rank2, ID2, income2, ...
rank3, ID3, income3, ...

4 Task 4

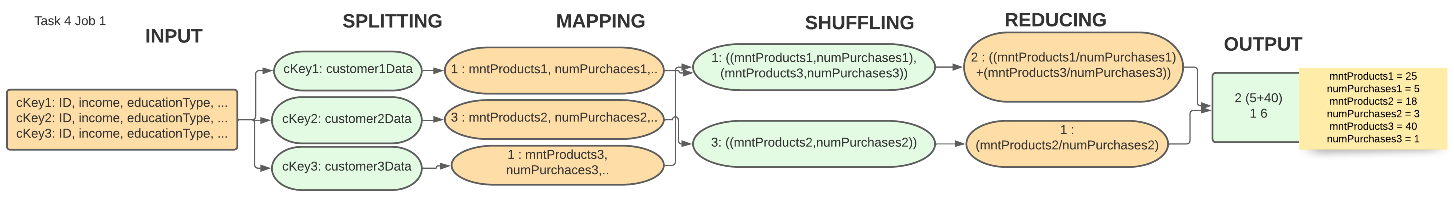
4.1 Job 1

Description: The purpose of first Job is to provide counters that have the total number of customers as well total average amount spent per purchase for each customer. For this Job we used 4 buckets to distribute the task. The result will be 4 records that each will have the size(number of customers), sum(Total averages per customer) of each bucket.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as shown in the .csv file.
  - KeyOUT: Returns the random bucket number (1,2,3,4).
  - ValueOUT: Returns the amount of money spent on products and number of purchases.
- Reducer:
  - KeyIN: Bucket number (1,2,3,4).
  - ValueIN: Array with amounts spent for wine ((moneySpent1,numPurchases1),(moneySpent1,numPurchases1) ...) returned from Shuffle for each bucket number.
  - KeyOUT: Total number of customers of the bucket.
  - ValueOUT: Total sum of money spent per purchase (moneySpent/numPurchases) of all customers of the bucket.

```
class MoneySumCountMapper
    method Map(Customer cKey, Customer cValues)
        mntProducts = cValues.MntWines + cValues.MntFruits + cValues.MntMeat + ...
        numPurchases = cValues.numWeb + cValues.numStore + ...
        bucket = Random (1-4)
        emit(bucket, (mntProducts,numPurchases))

class MoneySumCountReducer
    method Reduce(Bucket b, Values [(mntProducts1,numPurchases1), (mntProducts2,numPurchases2)
        sum=0
        size=0
        for each Value v in Values [(mntProducts1,numPurchases1), (mntProducts2,numPurchases2)
            sum = sum + (v.mntProducts/v.numPurchases)
            size++
        emit(Count size, Count sum)
```


Task 4 Job 1
INPUT
SPLITTING
MAPPING
SHUFFLING
REDUCING
OUTPUT
cKey1: ID, income, educationType, ...
cKey2: ID, income, educationType, ...
cKey3: ID, income, educationType, ...
cKey1: customer1Data
cKey2: customer2Data
cKey3: customer3Data
1 : mntProducts1, numPurchases1,..
3 : mntProducts2, numPurchaces2,..
1 : mntProducts3, numPurchaces3,..
1: ((mntProducts1,numPurchases1), (mntProducts3,numPurchases3))
3: ((mntProducts2,numPurchases2))
2 : ((mntProducts1/numPurchases1) +(mntProducts3/numPurchases3))
1 : (mntProducts2/numPurchases2)
2 (5+40)
1 6
mntProducts1 = 25
numPurchases1 = 5
mntProducts2 = 18
numPurchases2 = 3
mntProducts3 = 40
numPurchases3 = 1

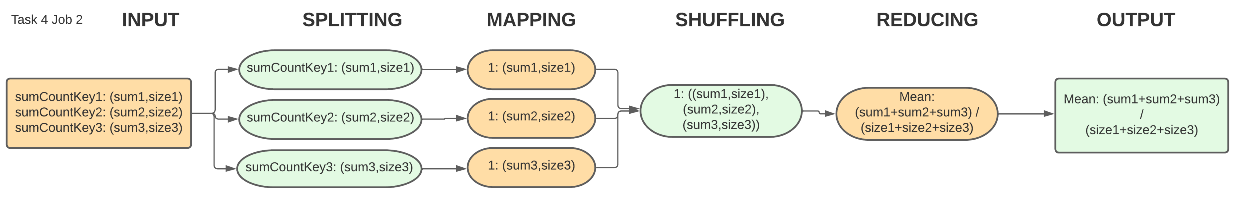
4.2 Job 2

Description: The purpose of second Job is to find the mean value of money spent per purchase according to the output of Job 1.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: The whole record of Job 1 (size,sum) .
  - KeyOUT: Returns number 1.
  - ValueOUT: Returns the whole record (size,sum) of Job 1 .
- Reducer:
  - KeyIN: Number 1..
  - ValueIN: Array with the 4 (size,sum) records.
  - KeyOUT: Returns string word "Mean".
  - ValueOUT: Returns the sum/size for all the 4 records, which is the total Mean value.

```
class meanMapper
    method Map(MoneySumCount Key, MoneySumCount Values)
        emit(count 1, Values)

class meanReducer
    method Reduce(count 1, values[(size1,sum1), (size2,sum2), ...])
        sum=0
        count=0
        for each v in values[(size1,sum1), (size2,sum2), ...]
            sum = sum + v.get(sum)
            size = count + v.get(size)
        emit("Mean", (sum/size))
```


Task 4 Job 2
INPUT
SPLITTING
MAPPING
SHUFFLING
REDUCING
OUTPUT
sumCountKey1: (sum1,size1)
sumCountKey2: (sum2,size2)
sumCountKey3: (sum3,size3)
sumCountKey1: (sum1,size1)
sumCountKey2: (sum2,size2)
sumCountKey3: (sum3,size3)
1: (sum1,size1)
1: (sum2,size2)
1: (sum3,size3)
1: ((sum1,size1), (sum2,size2), (sum3,size3))
Mean: (sum1+sum2+sum3) / (size1+size2+size3)
Mean: (sum1+sum2+sum3) / (size1+size2+size3)

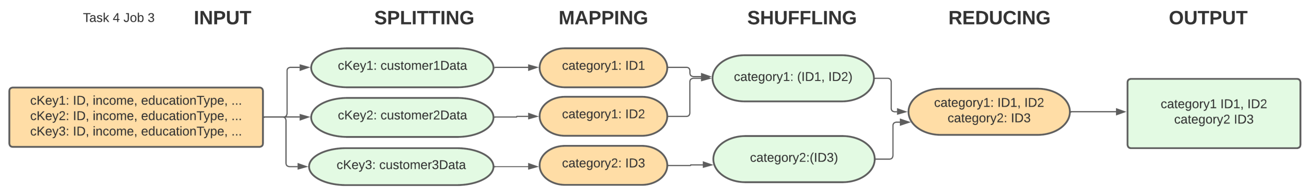
4.3 Job 3

Description: The purpose of third Job is to find and return the customer ID's grouped by the category they belong to.

- Mapper:
  - KeyIN: Automatic generated key by InputSplit Hadoop.
  - ValueIN: Customer information as shown in the .csv file.
  - KeyOUT: Categories (Gold,Silver). Gold: Customers with income > 69500 AND AverageMoneySpent > 1.5 * Mean AND registrationDate < 365. Silver: Gold: Customers with income > 69500 AND AverageMoneySpent > 1.5 * Mean AND registrationDate > 365.
  - ValueOUT: CustomerID.
- Reducer:
  - KeyIN: Category (Gold,Silver).
  - ValueIN: Array with CustomerID's for each category.
  - KeyOUT: Category.
  - ValueOUT: Array of CustomerID's for each category sorted.

```
class CustomersMapper
    method Map(Customer cKey, Customer cValues)
        Customer c = cValues[category, ID,income ,registrationDate, averageMoneySpent]
        if (cValues.income > 69500 && (cValues.averageMoneySpent > 1.5 * Mean))
            if (cValues.registrationDate < 365)
                c.category = "Gold"
                emit(c.category, cValues.ID)
            else
                c.category = "Silver"
                emit(c.category, cValues.ID)

class CustomersReducer
    method Reduce(Category cat, Customer cValues[ID1,ID2,ID3, ...])
        emit(cat, cValues)
```



5 User Manual

5.1 Contents

In the BigDataManagement directory you will find:

- "big_data_management.py" which is the Python script we used to pre-process the original data and created the clean_data.csv
- "dataFolder" directory which contains the clean_data.csv
- "config" directory which contains all the .xml configurations required from Hadoop during the installation.
- "Dockerfile" which creates our docker image. It contains Ubuntu 16, Java 8, Apache Hadoop, Apache PIG and openssh-server.
- "start-cluster.sh" which is a bash script used to set up our cluster on Docker containers. It takes one argument which is the number of workers you want to have on your cluster. e.g. "./start-cluster.sh 3".
- "stop-cluster.sh" which is used to stop and remove the Docker containers and Docker network of our cluster.
- "src" directory which contains the source code of Java implementation.
- "program" directory which contains the JAR of the Java implementation.
- "pigScripts" directory which contains the (3) .pig scripts, one for each task.
- "run_all_tasks.sh" which is used in order to run all queries for both Java and Pig implementations automatically.
- "export_results.sh" which is used to export the results of the Java and Pig implementation on our local machine.
- "results" directory which contains all the results files from both implementations.

5.2 Requirements

- Having Docker installed on your local machine.
- Provide Docker with at least 4 CPU, 8GB Memory, 1GB Swap Memory, 32GB Disk image size. Note: In case you have more containers running on your local machine you might need to upscale the mentioned resources.
- Be able to run bash scripts from your local machine.
- Have a good Internet connection since it is needed to download 4 GB during the creation of the Docker image.

5.3 Steps

1. Open a terminal on your local machine.
2. Move to BigDataManagement directory.
3. With open Docker type " `./start-cluster.sh 2` " to initiate a cluster with 2 worker Datanodes + 1 master Namenode. This step will require several minutes to set up the image and the containers.
4. Once the script is finished type " `docker ps` " and validate that you have 3 Containers running: "hadoop-master", "hadoop-worker1", "hadoop-worker2".
5. Open a CLI on the hadoop-master container using the following command on your local machine terminal: " `docker exec -it hadoop-master bash` ".
6. On the hadoop-master CLI you can execute the PIG implementation by using the following commands:
   (a) `pig /pigScripts/query_2.pig`
   (b) `pig /pigScripts/query_3.pig`
   (c) `pig /pigScripts/query_4.pig`
7. On the hadoop-master CLI you can execute the Java implementation by using the following commands:
   (a) `hadoop jar /program/BDM_Project1-1.0.60.jar Vitsas_Spiliakos.Question2 input JavaResults/task2`
   (b) `hadoop jar /program/BDM_Project1-1.0.60.jar Vitsas_Spiliakos.Question3 input output1 output2 output3 JavaResults/task3`
   (c) `hadoop jar /program/BDM_Project1-1.0.60.jar Vitsas_Spiliakos.Question4 input output1 output2 JavaResults/task4`
8. To run the steps 6, 7 automatically instead of executing the commands individually you can run the `./run_all_tasks.sh` on your local machine CLI. Note: You can run either the commands or the run_all_tasks.sh script. If you want to try both please reload the cluster by using stop-cluster.sh and then start-cluster.sh scripts.

9. To view the Pig results you on hadoop-master container CLI you can use the following commands:
   (a) `hdfs dfs -cat /user/root/PigResults/task2/part-r-00000`
   (b) `hdfs dfs -cat /user/root/PigResults/task3/part-m-00000`
   (c) `hdfs dfs -cat /user/root/PigResults/task4/part-r-00000`
10. To view the Java results you on hadoop-master container CLI you can use the following commands:
    (a) `hdfs dfs -cat /user/root/JavaResults/task2/part-r-00000`
    (b) `hdfs dfs -cat /user/root/JavaResults/task3/part-r-00000`
    (c) `hdfs dfs -cat /user/root/JavaResults/task4/part-r-00000`
11. To parse the result files in your local machine you can execute the `export_results.sh` script in your local machine terminal after you have finished the execution of the Java/Pig implementations by using the following command: `./export_results.sh -j true -p true -d /path/for/results/on/local/` . If you have ran only one of the implementations please do not use the -p,-j flags. -p flag is for Pig results, -j flag is for Java results.

    You can also use `./export_results.sh -h` for help option.
12. To kill and remove the containers and docker network you can use the `./stop-cluster.sh` command on your local CLI.

6 Pig vs Java

6.1 Consistency

- Task 2: For this task there were no differences observed between the two implementations
- Task 3: The only difference in this task is the sorting. It is sorted by the amount spend on wines and the income correctly, but on Java we had also explicitly sorted the results by ID. Otherwise the results would be identical.
- Task 4: For this task we cannot compare the results as the two implementations were different in the handling of the DT_Customer. Pig implementation handles the DT_Customer by calendar year (if Year= 2022 then customer aquired his code during last year) while in Java implementation we handle the "during last year" requirement as 365 days before today's date. A different rationale was adopted in the Java and Pig implementations for this task, too. For the Java implementation the filtering used for Gold and Silver customers was made taking into account the average of the money spent *per* Purchase. Gold and Silver customers were considered to be those spending 50% more than the above average. For the Pig implementation only the averages for each product type were taken into account. Hence, Gold and Silver customers were considered to be those spending 50% more than all of those averages.

6.2 Lines of code

- Pig: The Pig commands were totally 73 lines of code (with new lines removed)
- Java: The Java commands were totally 615 lines of code (with new lines removed) for all the classes in the project.

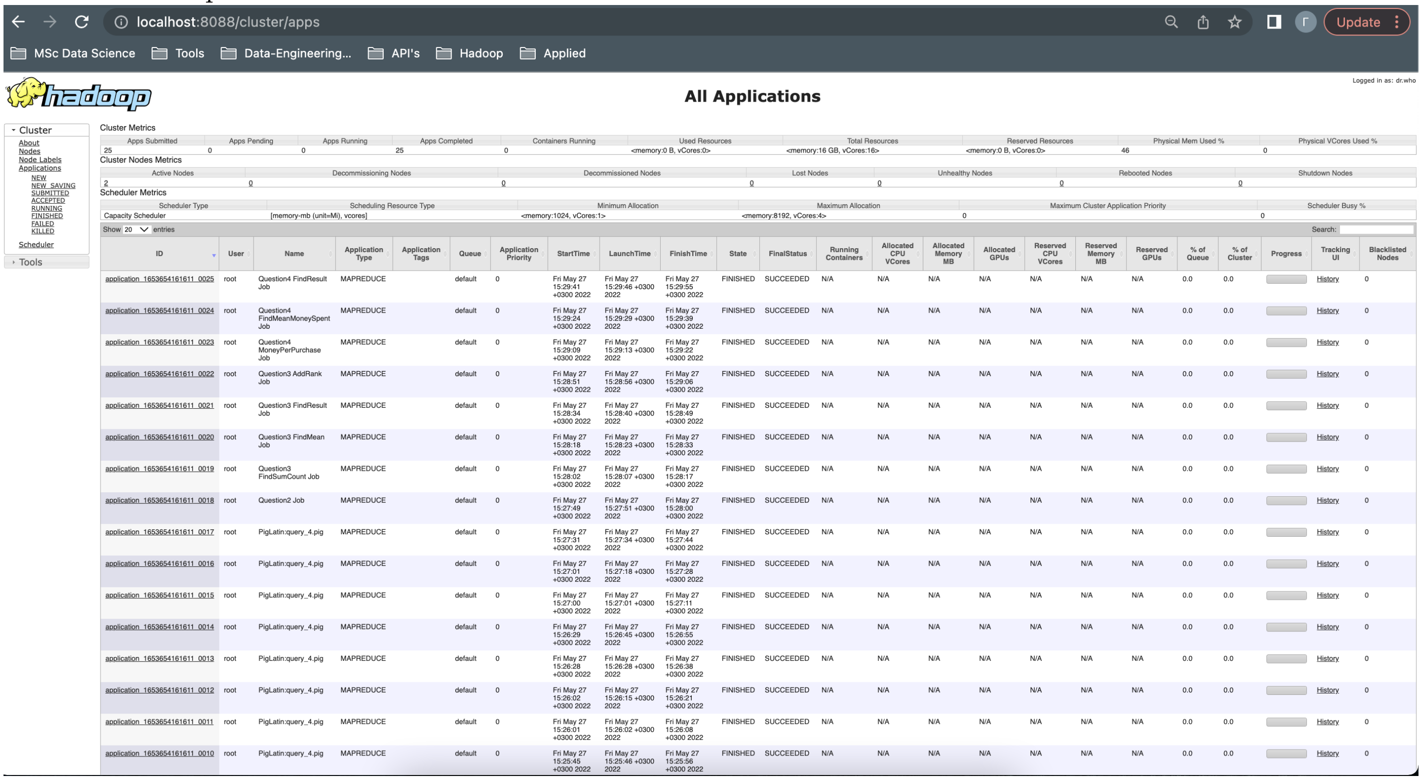
  As we are able to see Java code requires 8x more commands in contrast to the Pig scripts.

6.3 Execution time

- Pig: Pig implementation spawned 17 total Jobs and total time spent was 4 minutes and 7 seconds.
  - Task 2: 3 Jobs - 0 minute 46 seconds
  - Task 3: 6 Jobs - 1 minute 22 seconds
  - Task 4: 8 Jobs - 1 minute 59 seconds

- Java: Java implementation spawned 8 total Jobs and total time spent was 2 minutes and 1 second
  - Task 2: 1 Job - 0 minute 11 seconds
  - Task 3: 4 Jobs - 1 minute 04 seconds
  - Task 4: 3 Jobs - 0 minute 46 seconds

We are able to see that the Pig implementation spent almost 2x more time. The results were parsed from the Hadoop UI:



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %	Physical VCores Used %
25	0	0	25	0	<memory:0 B, vCores:0>	<memory:16 GB, vCores:16>	<memory:0 B, vCores:0>	46	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
2	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority	Scheduler Busy %
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0	0

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Allocated GPUs	Reserved CPU VCores	Reserved Memory MB	Reserved GPUs	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1653654161611_0025	root	Question4 FindResult Job	MAPREDUCE		default	0	Fri May 27 15:29:41 +0300 2022	Fri May 27 15:29:46 +0300 2022	Fri May 27 15:29:55 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0024	root	Question4 FindMeanMoneySpent Job	MAPREDUCE		default	0	Fri May 27 15:29:24 +0300 2022	Fri May 27 15:29:29 +0300 2022	Fri May 27 15:29:39 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0023	root	Question4 MoneyPerPurchase Job	MAPREDUCE		default	0	Fri May 27 15:29:09 +0300 2022	Fri May 27 15:29:13 +0300 2022	Fri May 27 15:29:22 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0022	root	Question3 AddRank Job	MAPREDUCE		default	0	Fri May 27 15:28:51 +0300 2022	Fri May 27 15:28:56 +0300 2022	Fri May 27 15:29:06 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0021	root	Question3 FindResult Job	MAPREDUCE		default	0	Fri May 27 15:28:34 +0300 2022	Fri May 27 15:28:40 +0300 2022	Fri May 27 15:28:49 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0020	root	Question3 FindMean Job	MAPREDUCE		default	0	Fri May 27 15:28:18 +0300 2022	Fri May 27 15:28:23 +0300 2022	Fri May 27 15:28:33 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0019	root	Question3 FindSumCount Job	MAPREDUCE		default	0	Fri May 27 15:28:02 +0300 2022	Fri May 27 15:28:07 +0300 2022	Fri May 27 15:28:17 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0018	root	Question2 Job	MAPREDUCE		default	0	Fri May 27 15:27:49 +0300 2022	Fri May 27 15:27:51 +0300 2022	Fri May 27 15:28:00 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0017	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:27:31 +0300 2022	Fri May 27 15:27:34 +0300 2022	Fri May 27 15:27:44 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0016	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:27:01 +0300 2022	Fri May 27 15:27:18 +0300 2022	Fri May 27 15:27:28 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0015	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:27:00 +0300 2022	Fri May 27 15:27:01 +0300 2022	Fri May 27 15:27:11 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0014	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:26:29 +0300 2022	Fri May 27 15:26:45 +0300 2022	Fri May 27 15:26:55 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0013	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:26:28 +0300 2022	Fri May 27 15:26:28 +0300 2022	Fri May 27 15:26:38 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0012	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:26:02 +0300 2022	Fri May 27 15:26:15 +0300 2022	Fri May 27 15:26:21 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0011	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:26:01 +0300 2022	Fri May 27 15:26:02 +0300 2022	Fri May 27 15:26:08 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1653654161611_0010	root	PigLatin:query_4.pig	MAPREDUCE		default	0	Fri May 27 15:25:45 +0300 2022	Fri May 27 15:25:46 +0300 2022	Fri May 27 15:25:56 +0300 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0