BIG DATA SYSTEMS

Assignment 2

GROUP 22

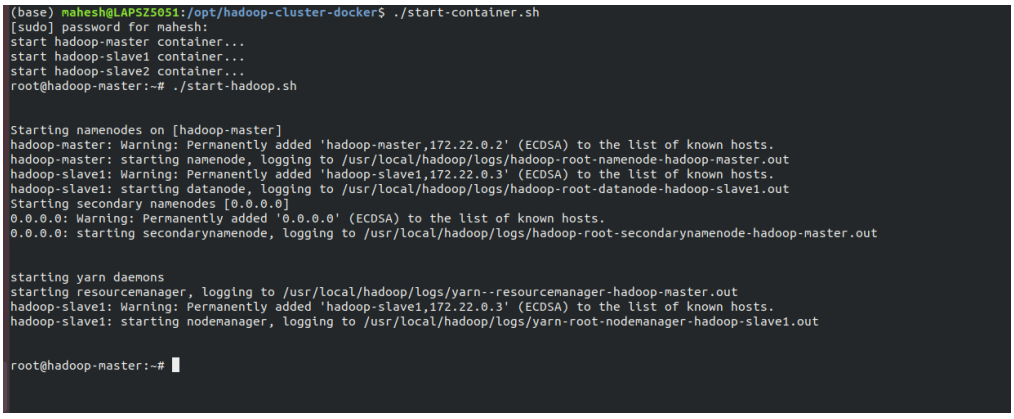
Authored by:

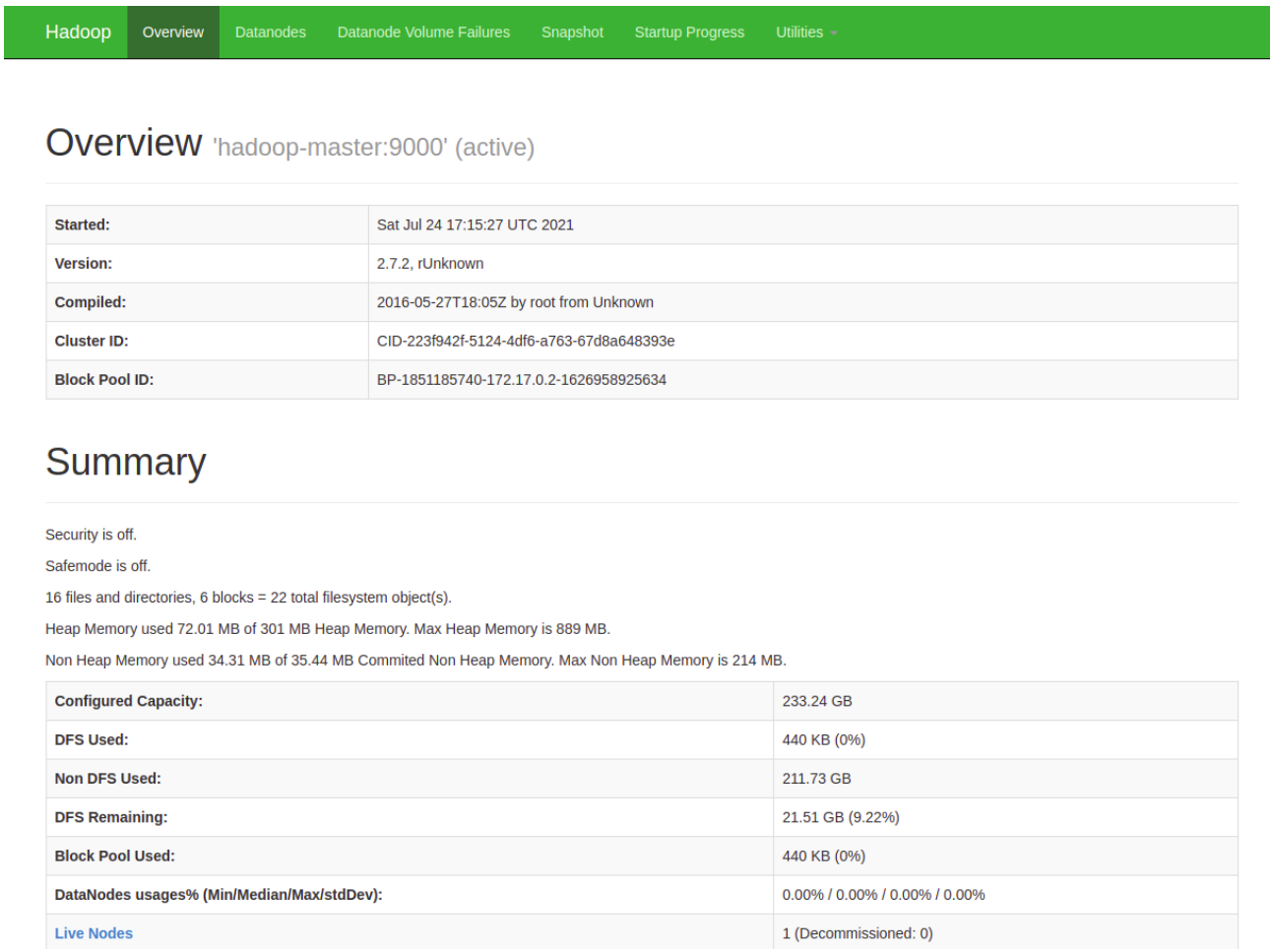
1. Chandra Sekhar Gupta – 2019ab04187
2. Sanka Mahesh Sai – 2019ab04135
3. Snigdha Tarua – 2019ab04171

STEP 1. Hadoop cluster Setup :

Setup using docker container : In the command prompt :

1. sudo docker pull kiwenlau/hadoop:1.0
2. git clone https://github.com/kiwenlau/hadoop-cluster-docker
3. sudo docker network create --driver=bridge hadoop
4. cd hadoop-cluster-docker
5. sudo ./start-container.sh
6. ./start-hadoop.sh
    - Sample o/p





STEP 2. Updating respective files in the Hadoop:

Now add input geosales.csv in to the hadoop filesystem

1. Create a temporary *input* folder with *geosales.csv* file in it (Remove the column names in the csv file)
2. Create *input* folder in hdfs file system (hadoop fs -mkdir -p input)
3. Put the file in current input folder to hdfs file system input folder (hdfs dfs -put ./input/* input)
4. Create mapper.py and reducer.py for respective operations
5. Use the following command to run python mapper and reducer files to get the necessary output
    1. hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar -Dmapred.reduce.tasks=1 -file mapper.py reducer.py -mapper "python3 mapper.py" -reducer "python3 reduce.py" -input input -output output

1. Average unit_price by country for a given item type in a certain year

que1_avg_unit_price_mapper.py

```python
#!/usr/bin/env python
"""mapper.py"""

#Importing necessary libraries
import sys
import pandas as pd


# Operations for every line in the stdin
for line in sys.stdin:

    #using the column name
    columns = ["index",'region','country','item_type','sales_channel',
    'order_priority','order_date', 'order_id',    'ship_date',
    'units_sold','unit_price',  'unit_cost',    'total_revenue',
    'total_cost',   'total_profit']

    #removing tailing spaces
    line = line.strip()

    colvalues = line

    #splitting input line to column values
    cols = colvalues.split(',')
    rowJson = dict()

    #mapping input line to column names using dict for easier access
    for i in range(0,len(columns)):
        rowJson[columns[i]] = cols[i]

    #acessing required data
    order_date = pd.to_datetime(rowJson['order_date'])
    ship_date = pd.to_datetime(rowJson['ship_date'])
    year = order_date.year

    #creating the tab '\t' seperated key,value pair and prting - in
    order to stdin of reducer to pick
    key = (rowJson['country'],rowJson['item_type'],year)
    value = rowJson['unit_price']
    print('%s\t%s' % (key,value))
```

que1_avg_unit_price_reducer.py

```python
#!/usr/lib/python
"""reducer.py"""

import sys

unit_price = {}

#Partitoner
for line in sys.stdin:
    line = line.strip()
    key, price = line.split('\t')

    if key in unit_price:
        unit_price[key].append(float(price))
    else:
        unit_price[key] = []
        unit_price[key].append(float(price))

#Reducer
for key in unit_price.keys():
    avg_price = sum(unit_price[key]) / len(unit_price[key])
    print('%s\t%s'% (key, avg_price))
```

Sample Output

Format: (Country, Item_type, Year) Average Unit Price

question1_ans.txt

```
('Tonga', 'Snacks', 2017)   152.58
('Czech Republic', 'Baby Food', 2015)   255.28
('Ukraine', 'Baby Food', 2018)  255.28000000000003
('Romania', 'Personal Care', 2018)  81.73
('Tanzania', 'Personal Care', 2020) 81.73
('France', 'Beverages', 2017)   47.449999999999996
('Senegal', 'Clothes', 2017)    109.27999999999999
('Sudan', 'Household', 2020)    668.27
('Papua New Guinea', 'Meat', 2020)  421.88999999999993
('Liechtenstein', 'Cereal', 2017)   205.70000000000002
('Belize', 'Cereal', 2016)  205.70000000000002
('Cambodia', 'Household', 2016) 668.27
('United States of America', 'Personal Care', 2020) 81.73
('North Korea', 'Cosmetics', 2016)  437.1999999999999
('Niger', 'Fruits', 2016)   9.33
('Cambodia', 'Snacks', 2016)    152.58
('Angola', 'Beverages', 2016)   47.449999999999996
```

2. Total units_sold by year for a given country and a given item type

que2_total_unit_sold_mapper.py

```python
#!/usr/bin/env python
"""mapper.py"""

#Importing necessary libraries
import sys
import pandas as pd


# Operations for every line in the stdin
for line in sys.stdin:

    #using the column name
    columns = ["index","region","country","item_type","sales_channel",
    'order_priority','order_date', 'order_id',    'ship_date',
    'units_sold','unit_price',  'unit_cost',    'total_revenue',
    'total_cost',   'total_profit']

    #removing tailing spaces
    line = line.strip()

    colvalues = line

    #splitting input line to column values
    cols = colvalues.split(',')
    rowJson = dict()

    #mapping input line to column names using dict for easier access
    for i in range(0,len(columns)):
        rowJson[columns[i]] = cols[i]

    #acessing required data
    country = rowJson['country']
    order_date = pd.to_datetime(rowJson['order_date'])
    year = order_date.year
    item_type = rowJson['item_type']

    #creating the tab '\t' seperated key,value pair and prting - in
    order to stdin of reducer to pick
    key = (country,item_type,year)
    value = rowJson['units_sold']
    print('%s\t%s' % (key,value))
```

que2_total_unit_sold_reducer.py

```python
#!/usr/lib/python
"""reducer.py"""

import sys

unit_sold = {}

#Partitoner
for line in sys.stdin:
    line = line.strip()
    key, value = line.split('\t')

    if key in unit_sold:
        unit_sold[key].append(float(value))
    else:
        unit_sold[key] = []
        unit_sold[key].append(float(value))

#Reducer
for key in unit_sold.keys():
    total = sum(unit_sold[key])
    print('%s\t%s'% (key, total))
```

Sample Output

Format: (Country, Item_type, Year) Units Sold

question2_ans.txt

```
('Tonga', 'Snacks', 2017)   30518.0
('Czech Republic', 'Baby Food', 2015)   33437.0
('Ukraine', 'Baby Food', 2018)  17614.0
('Romania', 'Personal Care', 2018)  22494.0
('Tanzania', 'Personal Care', 2020) 39001.0
('France', 'Beverages', 2017)   32949.0
('Senegal', 'Clothes', 2017)    19564.0
('Sudan', 'Household', 2020)    31315.0
('Papua New Guinea', 'Meat', 2020)  41451.0
('Liechtenstein', 'Cereal', 2017)   36851.0
('Belize', 'Cereal', 2016)  33411.0
('Cambodia', 'Household', 2016) 20508.0
('United States of America', 'Personal Care', 2020) 48754.0
('North Korea', 'Cosmetics', 2016)  54378.0
('Niger', 'Fruits', 2016)   45530.0
```

3. Find the max and min units_sold in any order for each year by country for a given item type

que3_min_max_units_sold_mapper.py

```python
#!/usr/bin/env python
"""mapper.py"""

#Importing necessary libraries
import sys
import pandas as pd


# Operations for every line in the stdin
for line in sys.stdin:

    #using the column name
    columns = ["index",'region','country','item_type','sales_channel',
    'order_priority','order_date', 'order_id',    'ship_date',
    'units_sold','unit_price',  'unit_cost',    'total_revenue',
    'total_cost',   'total_profit']

    #removing tailing spaces
    line = line.strip()

    colvalues = line

    #splitting input line to column values
    cols = colvalues.split(',')
    rowJson = dict()

    #mapping input line to column names using dict for easier access
    for i in range(0,len(columns)):
        rowJson[columns[i]] = cols[i]

    #acessing required data
    country = rowJson['country']
    item_type = rowJson['item_type']
    order_date = pd.to_datetime(rowJson['order_date'])
    year = order_date.year
    unit_sold =rowJson['units_sold']

    #creating the tab '\t' seperated key,value pair and prting - in
    order to stdin of reducer to pick
    key = (country,item_type,year)
    value = unit_sold

    print('%s\t%s' % (key,value))
```

que3_min_max_units_sold_reducer.py

```python
#!/usr/lib/python
"""reducer.py"""

import sys

# Custom Partitoner Class and reducer
class MinMaxPartitionClass:

    def __init__(self):
        self.unit_sold = {}


    def update_min_max(self,key,val):
        value = float(val)
        if key not in self.unit_sold.keys():
            self.unit_sold[key] ={'max': value, 'min': value}
        else:
            self.unit_sold[key]['min'] = value if self.unit_sold[key]
            ['min'] < value else self.unit_sold[key]['min']
            self.unit_sold[key]['max'] = value if self.unit_sold[key]
            ['max'] > value else self.unit_sold[key]['max']

        return self.unit_sold


# Object for paritioner class
mmObject = MinMaxPartitionClass()
updated_units_sold = {}


# for each line in input update min max value in object
for line in sys.stdin:
    line = line.strip()
    key, value = line.split('\t')
    updated_units_sold = mmObject.update_min_max(key,value)


# printing min, max values for every key in the data setd
for key in updated_units_sold.keys():
    max_unit = updated_units_sold[key]['min']
    min_unit = updated_units_sold[key]['max']
    print('%s\t%s'% (key, (min_unit,max_unit)))
```

Sample Output

Format: (Country, Item_type, Year) (min_units, max_units)

question3_ans.txt

```
('Tonga', 'Snacks', 2017)   (165.0, 8601.0)
('Czech Republic', 'Baby Food', 2015)   (1713.0, 9057.0)
('Ukraine', 'Baby Food', 2018)  (1627.0, 7293.0)
('Romania', 'Personal Care', 2018)  (1587.0, 7030.0)
('Tanzania', 'Personal Care', 2020) (607.0, 9747.0)
('France', 'Beverages', 2017)   (706.0, 9158.0)
('Senegal', 'Clothes', 2017)    (560.0, 4733.0)
('Sudan', 'Household', 2020)    (1253.0, 9896.0)
('Papua New Guinea', 'Meat', 2020)  (1400.0, 8478.0)
('Liechtenstein', 'Cereal', 2017)   (3538.0, 9389.0)
('Belize', 'Cereal', 2016)  (2104.0, 7665.0)
```

3. What are the top 10 order id for a given year by the total_profit

que4_top_order_id_mapper.py

```python
#!/usr/bin/env python
"""mapper.py"""

#Importing necessary libraries
import sys
import pandas as pd


# Operations for every line in the stdin
for line in sys.stdin:

    #using the column name
    columns = ["index","region","country","item_type","sales_channel",
    "order_priority","order_date", "order_id",    "ship_date",
    "units_sold","unit_price",  "unit_cost",    "total_revenue",
    "total_cost",   "total_profit"]

    #removing tailing spaces
    line = line.strip()

    colvalues = line

    #splitting input line to column values
    cols = colvalues.split(',')
    rowJson = dict()

    #mapping input line to column names using dict for easier access
    for i in range(0,len(columns)):
        rowJson[columns[i]] = cols[i]

    #acessing required data
    order_id = rowJson['order_id']
    order_date = pd.to_datetime(rowJson['order_date'])
    year = order_date.year
    total_profit = rowJson['total_profit']

    #creating the tab '\t' seperated key,value pair and prting - in
    order to stdin of reducer to pick
    print('%s\t%s' % (year,(float(total_profit),order_id )))
```

que4_top_order_id_reducer.py

```python
#!/usr/lib/python
"""reducer.py"""

import sys

#creating hashmap to store top ten values
top_ten = {}

#Combiner
for line in sys.stdin:
    line = line.strip()
    keys, value = line.split('\t')

    if keys in top_ten :
        value = eval(value)
        top_ten[keys].append(value)
    else:
        top_ten[keys] = []
        value = eval(value)
        top_ten[keys].append(value)

#Reducer
for keys in top_ten.keys():
    total = sorted(top_ten[keys],key=lambda x: x[0],reverse=True)
    print('%s\t%s'% (keys, total[0:10]))
```

Sample Output

Format: Year [(Profit, OrderID1), (Profit, OrderID2), (Profit, OrderID3), ……..(Profit, OrderID10)]
Profit of OrderID1 > Profit of OrderID2 > … >Profit of OrderID10

question4_ans.txt

```
2017    [(1736439.69, '654263579'), (1736439.69, '610806214'), (1733483.9, '647944295'), (1733483.9, '359760463'), (1731223.59, '305523192'), (1729311.02, '692820870'), (1729311.02, '404637038'), (1727572.32, '957755315'), (1724616.53, '829072797'), (1724616.53, '663252198')]
2015    [(1737135.17, '520613610'), (1737135.17, '477156245'), (1734874.86, '300555741'), (1734353.25, '103747260'), (1730180.37, '314444434'), (1730180.37, '969717967'), (1730180.37, '890812098'), (1730180.37, '357901799'), (1723051.7, '972826755'), (1720617.52, '219858586')]
2018    [(1737482.91, '803154075'), (1735222.6, '460732972'), (1734700.99, '307381856'), (1726355.23, '807682073'), (1721660.74, '209754598'), (1721486.87, '454481065'), (1718704.95, '159977948'), (1716792.38, '634190356'), (1716270.77, '271561276'), (1714010.46, '383144605')]
2020    [(1738700.0, '261322534'), (1736265.82, '329448497'), (1734527.12, '428562343'), (1734527.12, '839109408'), (1732092.94, '208504474'), (1732092.94, '986141240'), (1727920.06, '995569312'), (1727398.45, '842218196'), (1725485.88, '775511443'), (1724964.27, '211613261')]
2016    [(1737656.78, '638516223'), (1737656.78, '149063289'), (1737656.78, '839785325'), (1735918.08, '859993302'), (1730701.98, '842894113'), (1730701.98, '921799981'), (1728789.41, '207828557'), (1726529.1, '564138352'), (1724094.92, '221717250'), (1723573.31, '514361703')]
2019    [(1738178.39, '344689023'), (1735744.21, '377366483'), (1733831.64, '310659730'), (1731571.33, '378785693'), (1731049.72, '470161044'), (1729137.15, '490369021'), (1728615.54, '783013474'), (1726876.84, '558494985'), (1726876.84, '680858218'), (1726355.23, '606412971')]
2014    [(1738004.52, '473878371'), (1736091.95, '205902516'), (1735570.34, '743273437'), (1735570.34, '131457269'), (1733657.77, '641118180'), (1731397.46, '918522107'), (1726702.97, '399500501'), (1726702.97, '277137267'), (1726702.97, '967859303'), (1726181.36, '692144954')]
2021    [(1737309.04, '434881627'), (1727920.06, '261389911'), (1723747.18, '716813552'), (1721313.0, '619118916'), (1721313.0, '662576282'), (1719052.69, '564881646'), (1719052.69, '485975778'), (1709663.71, '924300229'), (1707229.53, '747699725'), (1703056.65, '914939534')]
2013    [(1730875.85, '757162129'), (1726181.36, '893414056'), (1726181.36, '648687589'), (1707751.14, '411597907'), (1707751.14, '123414075'), (1696449.59, '538489139'), (1689494.79, '908687627'), (1689494.79, '663961160'), (1687060.61, '854450356'), (1679931.94, '612832677')]
```