ECS765P - Big Data Processing - 2021/22

Course Work Ethereum Analysis

Student Name – Prajwal Bhadravathi Narayana Murthy
Student ID – 210897324
EECS ID – EC211218

Table of Contents

1) Part A – Time Analysis

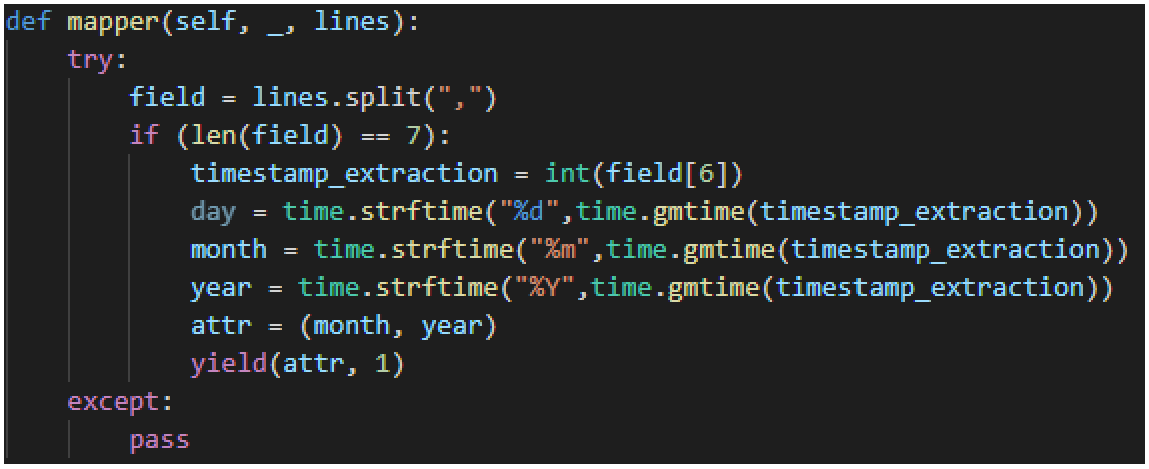
Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset. Create a bar plot showing the average value of transactions in each month between the start and end of the dataset.

1.1) Create a bar plot showing the number of transactions occurring every month between the start and end of the dataset.

Code Explanation:

For this task, MapReduce approach is used to determine the number of transactions occurring every month.

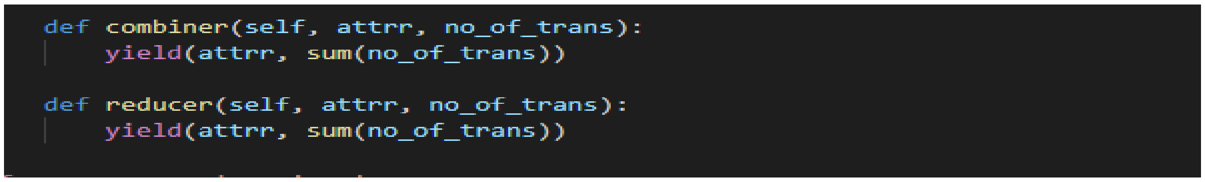
Mapper Function:



```
def mapper(self, _, lines):
    try:
        field = lines.split(",")
        if (len(field) == 7):
            timestamp_extraction = int(field[6])
            day = time.strftime("%d",time.gmtime(timestamp_extraction))
            month = time.strftime("%m",time.gmtime(timestamp_extraction))
            year = time.strftime("%Y",time.gmtime(timestamp_extraction))
            attr = (month, year)
            yield(attr, 1)
    except:
        pass
```

- The input for mapper here is the transactions file. The length of the file is checked. If the length is not equal to 7, it is considered as malformed record.
- Day Month and Year is extracted from timestamp field.
- Month and year are used as key attributes.
- For each and every transaction record, 1 is considered as value.

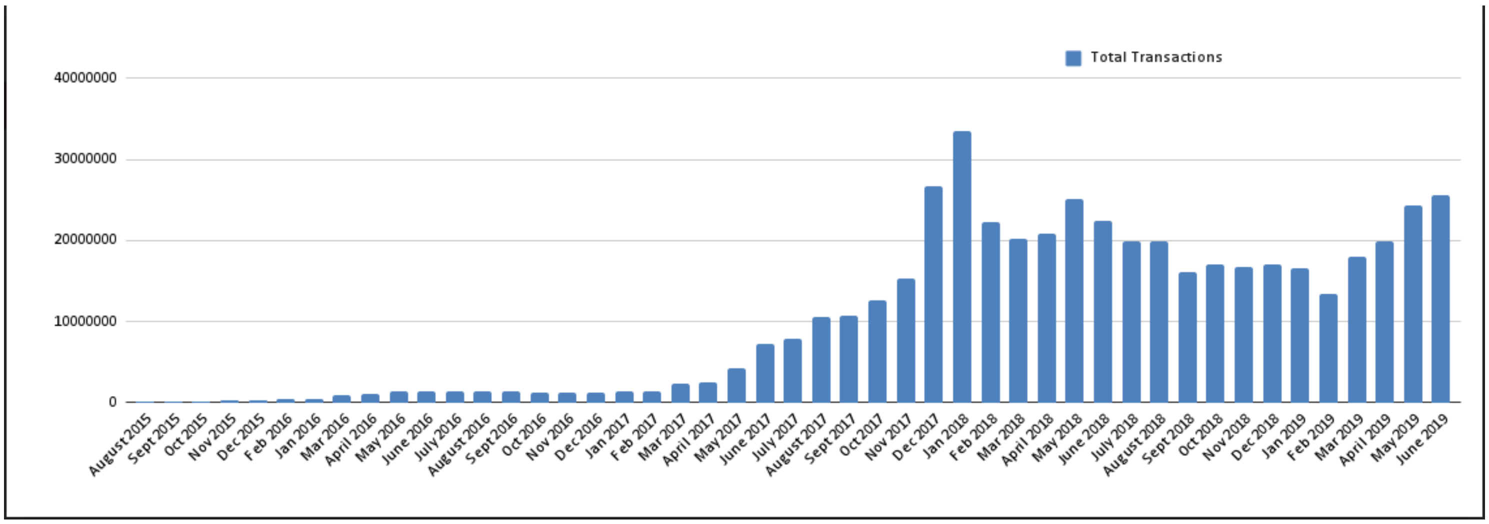
Reducer and Combiner Function:



```
def combiner(self, attrr, no_of_trans):
    yield(attrr, sum(no_of_trans))

def reducer(self, attrr, no_of_trans):
    yield(attrr, sum(no_of_trans))
```

- Combiner is used to reduce the execution time.
- Summation of every key attributes (month and year) along with all of its value is done in the reducer.
- The output file is generated and the below graph is plotted using the output with the help of Excel.

Graph Analysis

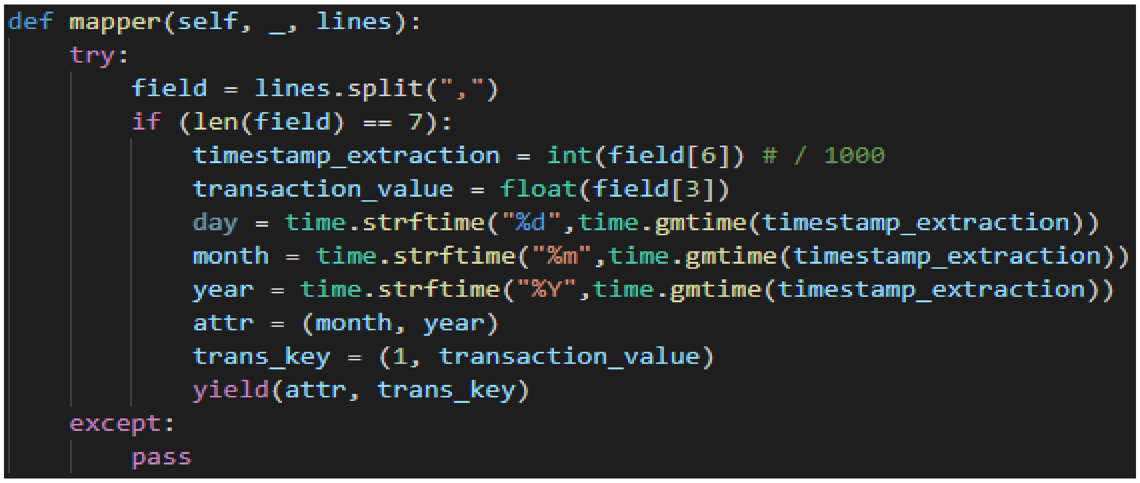


- Based on the output from the MapReduce program, above graph is plotted.
- From the plot, we could observe that the highest number of transactions were made in **January 2018**.
- On 11 December 2017, the total supply of Ether on Ethereum classic was hard capped at ETC 210700000 **by Gotham hard fork**, which **possibly could be a reason** for sudden **increase** in number of transactions in **December 2017** and **January 2018**.

The Job ID for this Task is "job_1637317090236_23434"

1.2) Create a bar plot showing the average value of transactions in each month between the start and end of the dataset

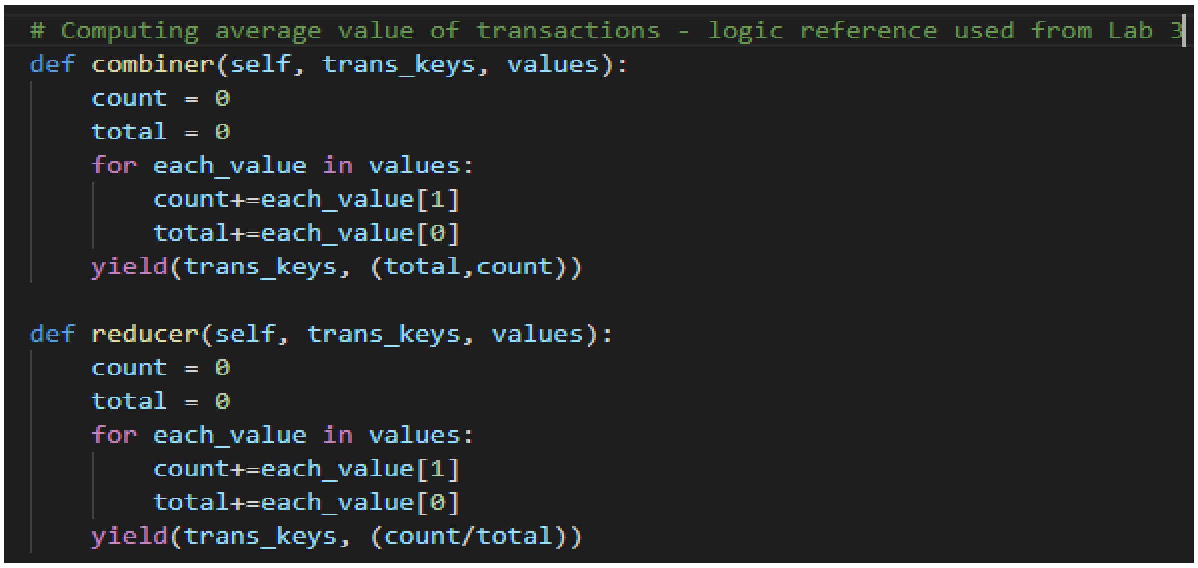
Code Explanation:



```
def mapper(self, _, lines):
    try:
        field = lines.split(",")
        if (len(field) == 7):
            timestamp_extraction = int(field[6]) # / 1000
            transaction_value = float(field[3])
            day = time.strftime("%d",time.gmtime(timestamp_extraction))
            month = time.strftime("%m",time.gmtime(timestamp_extraction))
            year = time.strftime("%Y",time.gmtime(timestamp_extraction))
            attr = (month, year)
            trans_key = (1, transaction_value)
            yield(attr, trans_key)
    except:
        pass
```

- **MapReduce approach** is used for this task to get the average value of transactions in each month.
- Similar to task 1, the mapper receives input from transactions, and length check is made on each record to identify malformed records.
- Month and Year is extracted from timestamp and yielded as key, whereas **transaction value along with 1** is yielded as value.

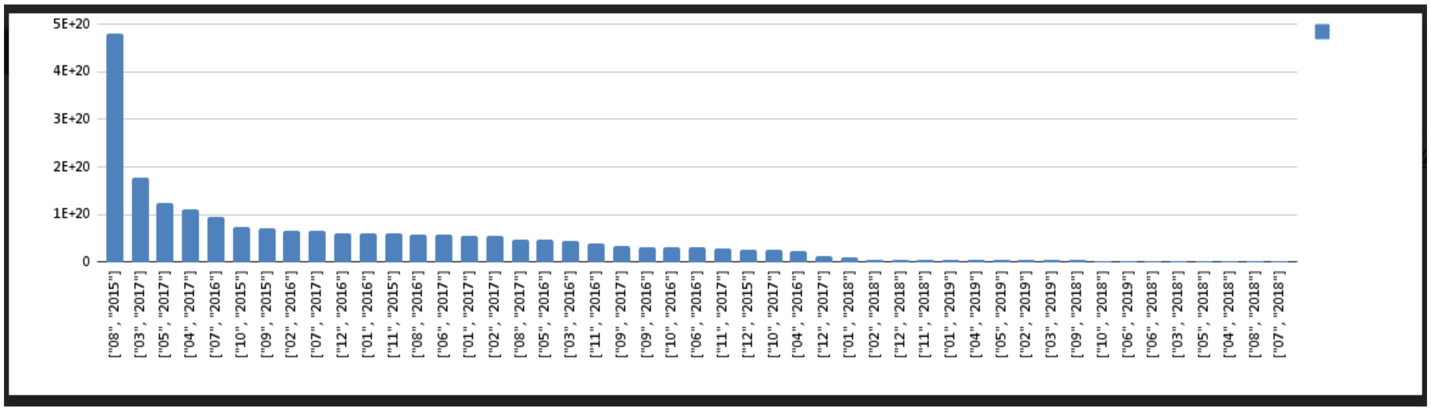
Reducer and Combiner functions:



```
# Computing average value of transactions - logic reference used from Lab 3
def combiner(self, trans_keys, values):
    count = 0
    total = 0
    for each_value in values:
        count+=each_value[1]
        total+=each_value[0]
    yield(trans_keys, (total,count))

def reducer(self, trans_keys, values):
    count = 0
    total = 0
    for each_value in values:
        count+=each_value[1]
        total+=each_value[0]
    yield(trans_keys, (count/total))
```

- Reducer function aggregates the count and value of the transactions in each month.
- Later, the count is divided by the total to compute the average value of transaction.
- Combiner is used to reduce the computing time.
- The output generated is plotted using Excel.

Graph Analysis



- The observation from the above is that, the **average transaction** value was the **highest during the initial days** of the dataset due to the **less number of transactions**. However, once the crypto buzzword spread the world, the Ethereum **transactions count increased** rapidly through-out the years thus resulting in the **plummeting of average transaction value**.

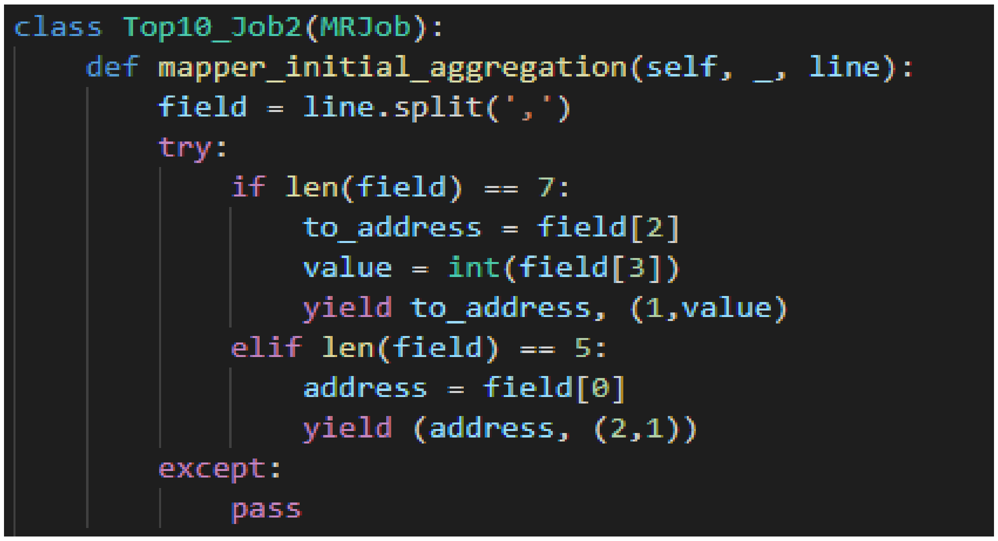
The Job ID for this Task is "job_1637317090236_23766"

2) Part B - TOP TEN MOST POPULAR SERVICES

Evaluate the top 10 smart contracts by total Ether received.

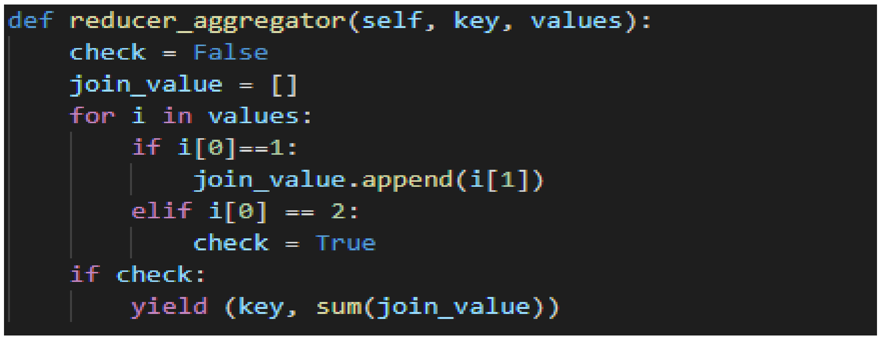
Code Explanation:

- To implement the task, MapReduce MRStep approaches is used.



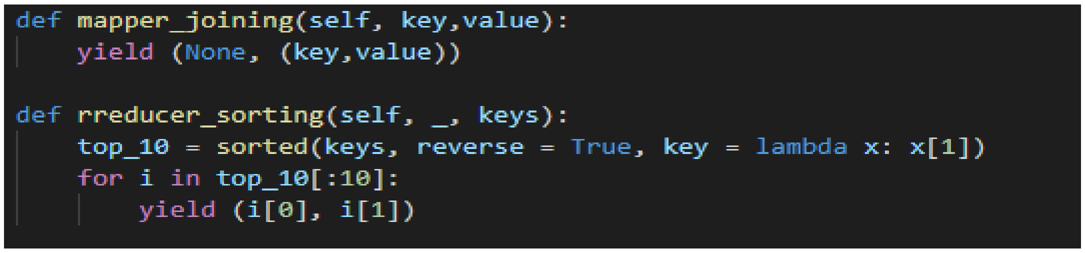
```python
class Top10_Job2(MRJob):
    def mapper_initial_aggregation(self, _, line):
        field = line.split(',')
        try:
            if len(field) == 7:
                to_address = field[2]
                value = int(field[3])
                yield to_address, (1,value)
            elif len(field) == 5:
                address = field[0]
                yield (address, (2,1))
        except:
            pass
```

- Both **Transactions** and **Contracts** dataset has been fed to the first mapper.
- Since two files are received as input, we have to differentiate the files based on the length each record.
- If a record is from Transactions dataset, **address** is considered as key and **value** is taken as value. ‘1’ is passed along with value to differentiate from contracts dataset.
- Similarly, only address field is used from Contracts and we are passing ‘2’ along with counter 1 to distinguish it from transactions.
- The first reducer verifies the values passed and checks if it’s true, which determines the existence and validity of that record. Later, the sum of all values are computed



```python
def reducer_aggregator(self, key, values):
    check = False
    join_value = []
    for i in values:
        if i[0]==1:
            join_value.append(i[1])
        elif i[0] == 2:
            check = True
    if check:
        yield (key, sum(join_value))
```

- The second mapper receives the yielded values from first reducer and combines key and values. None is considered as key.
- The second reducer sorts the values in descending to compute the top 10 values using the lambda function.



```
def mapper_joining(self, key,value):
    yield (None, (key,value))

def rreducer_sorting(self, _, keys):
    top_10 = sorted(keys, reverse = True, key = lambda x: x[1])
    for i in top_10[:10]:
        yield (i[0], i[1])
```

- Below is the output of the top10 most popular services

0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444	8.41551E+25
0xfa52274dd61e1643d2205169732f29114bc240b3	4.57875E+25
0x7727e5113d1d161373623e5f49fd568b4f543a9e	4.56206E+25
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef	4.31704E+25
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8	2.70689E+25
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd	2.11042E+25
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3	1.55624E+25
0xbb9bc244d798123fde783fcc1c72d3bb8c189413	1.19836E+25
0xabbb6bebfa05aa13e908eaa492bd7a8343760477	1.17065E+25
0x341e790174e3a4d35b65fdc067b6b5634a61caea	8.379E+24

Job ID's for this task:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_5397

http://andromeda.student.eecs.qmul.ac.uk:19888/jobhistory/job/job_1648683650522_5474

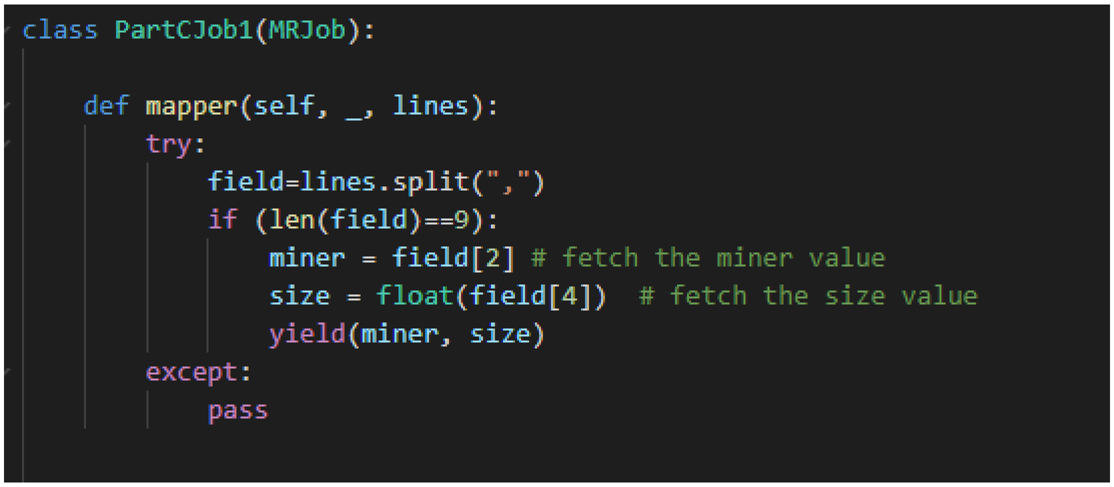
3. Part C – Top Ten Most Active Miners.

Evaluate the top 10 miners by the size of the blocks mined

Code Explanation:

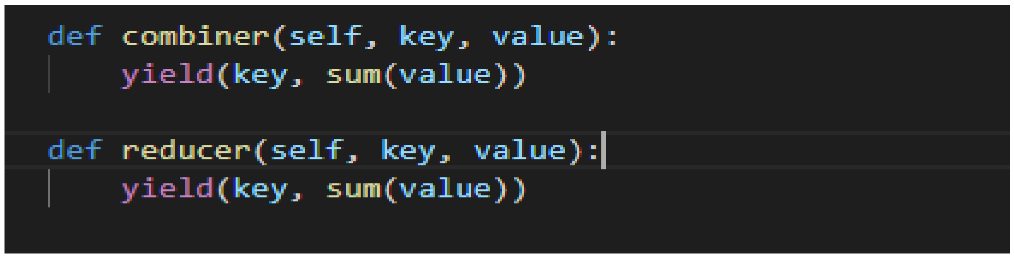
- MapReduce approach is used for this task.
- In the first Job,

- The blocks dataset has been used for the mapper to fetch the miner and size.



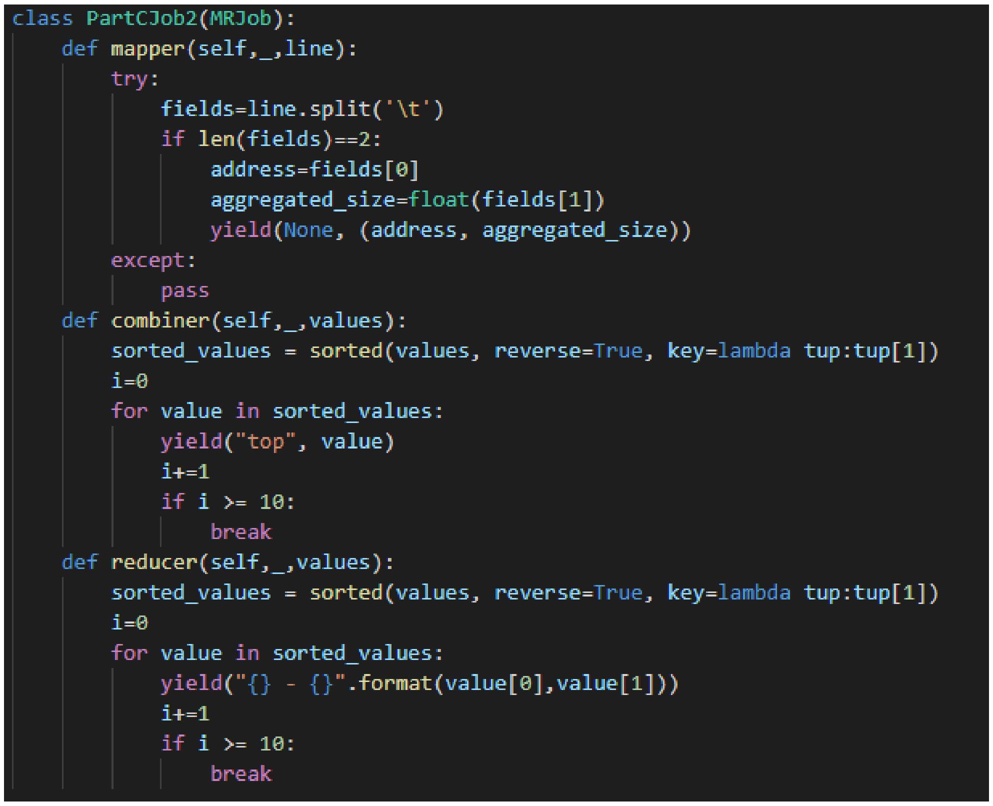
```python
class PartCJob1(MRJob):

    def mapper(self, _, lines):
        try:
            field=lines.split(",")
            if (len(field)==9):
                miner = field[2] # fetch the miner value
                size = float(field[4])  # fetch the size value
                yield(miner, size)
        except:
            pass
```

- Miner is taken as key and size as value.
- In the reducer, summation of all the value is done and yielded as value and key being the same.
- Combiner is used to reduce the computation time.



```python
def combiner(self, key, value):
    yield(key, sum(value))

def reducer(self, key, value):
    yield(key, sum(value))
```

- Another MapReduce code has been written which takes the input of first Job and computes the top10 most active miners.



```
class PartCJob2(MRJob):
    def mapper(self,_,line):
        try:
            fields=line.split('\t')
            if len(fields)==2:
                address=fields[0]
                aggregated_size=float(fields[1])
                yield(None, (address, aggregated_size))
        except:
            pass
    def combiner(self,_,values):
        sorted_values = sorted(values, reverse=True, key=lambda tup:tup[1])
        i=0
        for value in sorted_values:
            yield("top", value)
            i+=1
            if i >= 10:
                break
    def reducer(self,_,values):
        sorted_values = sorted(values, reverse=True, key=lambda tup:tup[1])
        i=0
        for value in sorted_values:
            yield("{} - {}".format(value[0],value[1]))
            i+=1
            if i >= 10:
                break
```

- The mapper of second job takes the output from first job which is tab separated and verifies the length of the record.
- Address and Aggregated size from Job1 output is yielded as value with key being None.
- In the reducer, lambda function is used to sort all the pairs in descending order and iterated throughout for 10 times to yield the top 10 miners.
- Below are the Job ID's for the task and output of top 10 miners.

Top 10 miners:

0xea674fdde714fd979de3edf0f56aa9716b898ec8	23989401188
0x829bd824b016326a401d083b33d092293333a830	15010222714
0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c	13978859941
0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5	10998145387
0xb2930b35844a230f00e51431acae96fe543a0347	7842595276
0x2a65aca4d5fc5b5c859090a6c34d164135398226	3628875680
0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01	1221833144
0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb	1152472379
0x1e9939daaad6924ad004c2560e90804164900341	1080301927
0x61c808d82a3ac53231750dadc13c777b59310bd9	692942577

The Job ID's for this tasks are:

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_2287

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_2295

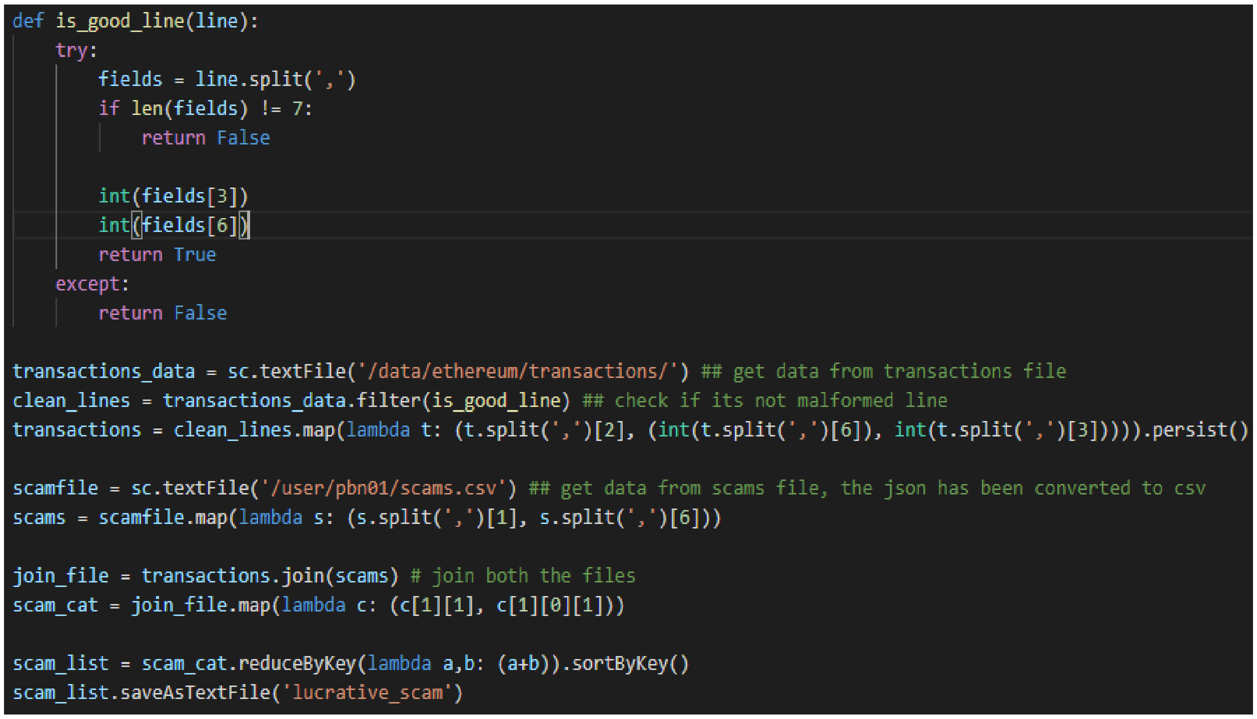
4) Part D - Data Exploration

4.1) Scam Analysis

Utilizing the provided scam dataset, what is the most lucrative form of scam? Does this correlate with certainly known scams going offline /inactive?

Code Explanation:

- The First task is to identify the most lucrative form of scam, for which Spark is used.
- The scams.json file is converted to scams.csv files using python's pandas framework and used as input.



```python
def is_good_line(line):
    try:
        fields = line.split(',')
        if len(fields) != 7:
            return False

        int(fields[3])
        int(fields[6])
        return True
    except:
        return False

transactions_data = sc.textFile('/data/ethereum/transactions/') ## get data from transactions file
clean_lines = transactions_data.filter(is_good_line) ## check if its not malformed line
transactions = clean_lines.map(lambda t: (t.split(',')[2], (int(t.split(',')[6]), int(t.split(',')[3])))).persist()

scamfile = sc.textFile('/user/pbn01/scams.csv') ## get data from scams file, the json has been converted to csv
scams = scamfile.map(lambda s: (s.split(',')[1], s.split(',')[6]))

join_file = transactions.join(scams) # join both the files
scam_cat = join_file.map(lambda c: (c[1][1], c[1][0][1]))

scam_list = scam_cat.reduceByKey(lambda a,b: (a+b)).sortByKey()
scam_list.saveAsTextFile('lucrative_scam')
```

- Function is_good_line() is defined to check the length of transactions dataset.

- Using the map transformation (address, timestamp, and value) fields are obtained.
- persist() is used to save the result of RDD in cache memory, which makes further computation faster.
- Scams.csv files is read and mapped using map transformation which produces (address, category)
- The two obtained files are joined using join transformation.
- The values from join transformation are mapped to obtain (category, value).
- The value field is aggregated using a lambda function based on category and later sorted to obtain the most lucrative scams.
- From the output below, we could observe that the Scamming is the most lucrative scam type.

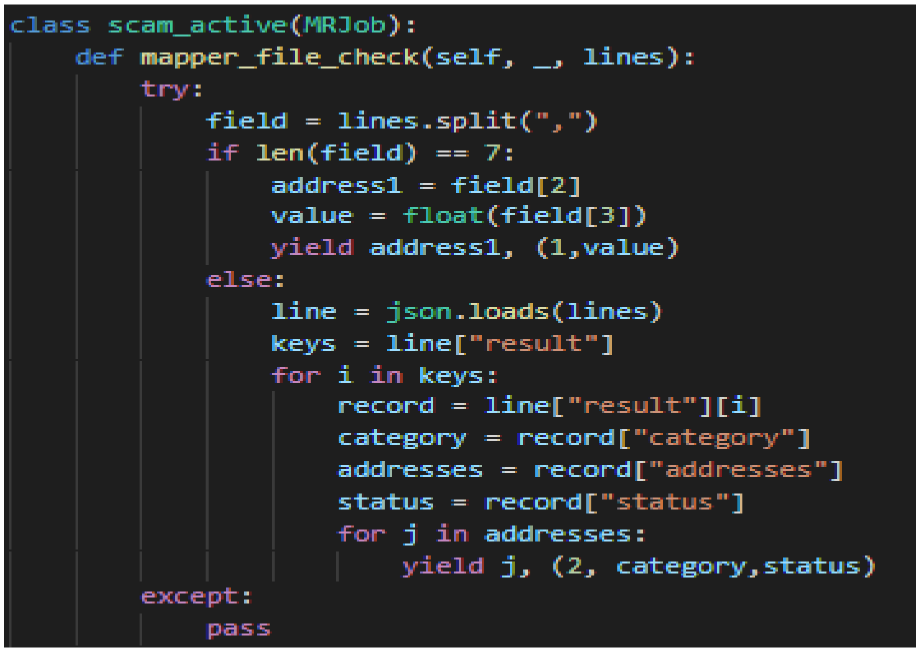
(u'Phishing', 26927757396110618476458L)
(u'Fake ICO', 1356457566889629979678L)
(u'Scamming', 38407781260421703730344L)

Task 2 : Does this correlate with certainly known scams going offline /inactive?

Code Explanation:

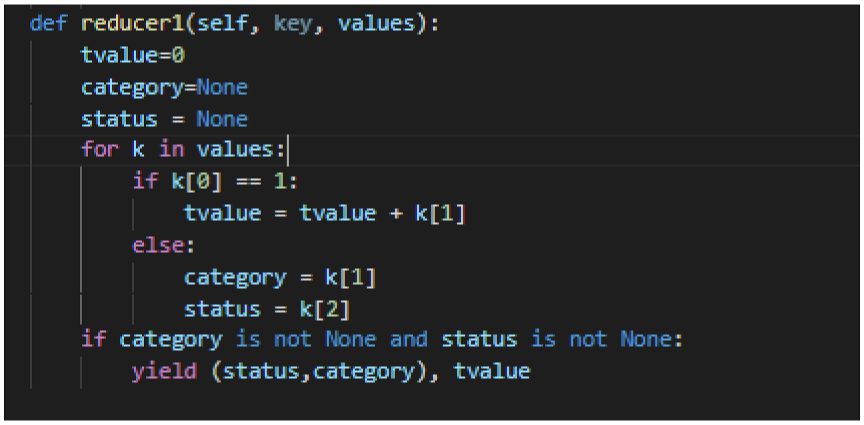
- In this code also, we will be using the scams.json files to evaluate the correlation.
- As implemented in previous code, the records are checked if they are clean or malformed.

- Address is taken as key and value is taken as value and constant 1 is sent to distinguish the record.
- From scams.json file, category and status as sent as value for reducer and address as key along with 2 as constant to distinguish the record.



```python
class scam_active(MRJob):
    def mapper_file_check(self, _, lines):
        try:
            field = lines.split(",")
            if len(field) == 7:
                address1 = field[2]
                value = float(field[3])
                yield address1, (1,value)
            else:
                line = json.loads(lines)
                keys = line["result"]
                for i in keys:
                    record = line["result"][i]
                    category = record["category"]
                    addresses = record["addresses"]
                    status = record["status"]
                    for j in addresses:
                        yield j, (2, category,status)
        except:
            pass
```

- In reducer, we will check the record mapping. If it’s one, it’s been added to values. Else, its been added to categories and status as this pair is from scams.json file.



```python
def reducer1(self, key, values):
    tvalue=0
    category=None
    status = None
    for k in values:
        if k[0] == 1:
            tvalue = tvalue + k[1]
        else:
            category = k[1]
            status = k[2]
    if category is not None and status is not None:
        yield (status,category), tvalue
```

- The second mapper just receives the values and forwards that to second reducer where the summation of total number of scams and its category is calculated.
- Below is the output from the MapReduce program.

Scam Type	State	Volume
Scamming	Active	2.2096952356679193e+22
Phishing	Inactive	1.4886777707995027e+19
Fake ICO	Offline	1.35645756688963e+21
Phishing	Offline	2.2451251236751494e+22
Scam	Offline	0
Phishing	Suspended	1.63990813e+18
Phishing	Active	4.5315978714979395e+21
Scamming	Offline	1.6235500337815102e+22
Scamming	Suspended	3.71016795e+18

The Job ID's for this tasks are below:

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7702

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_0635

4.2 Miscellaneous Analysis.

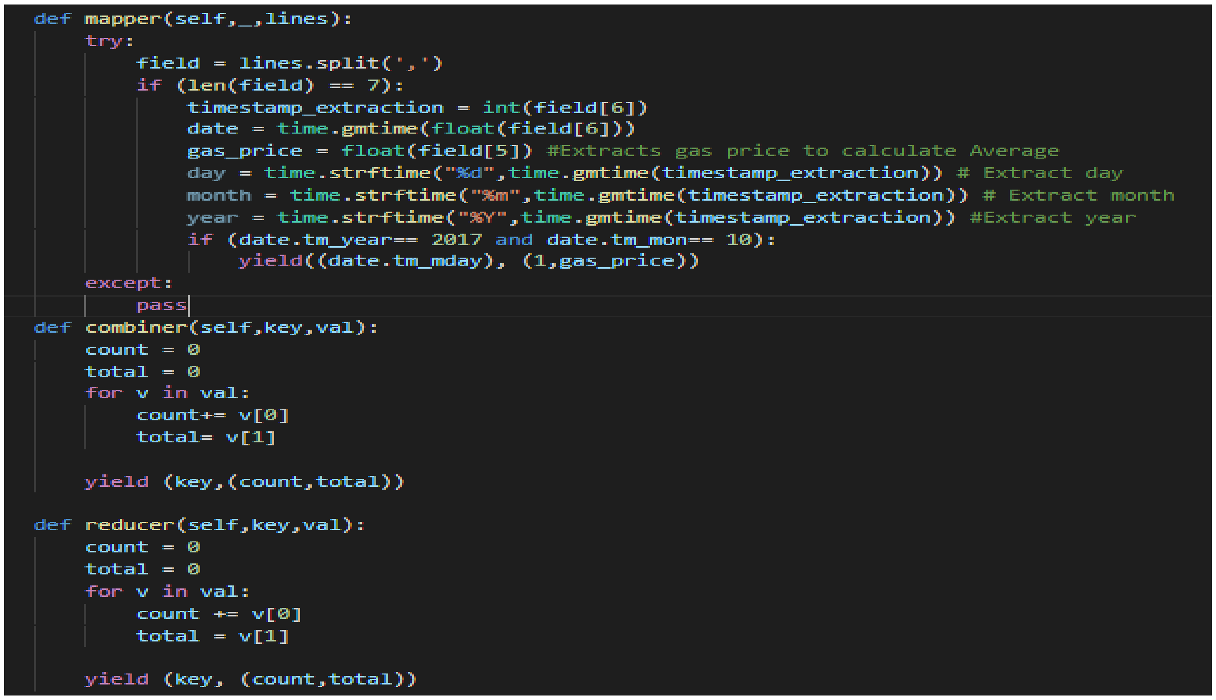
4.2.1 Fork The Chain.

We will observer Forks on below two different timelines.

- Byzantium – October 16 2017

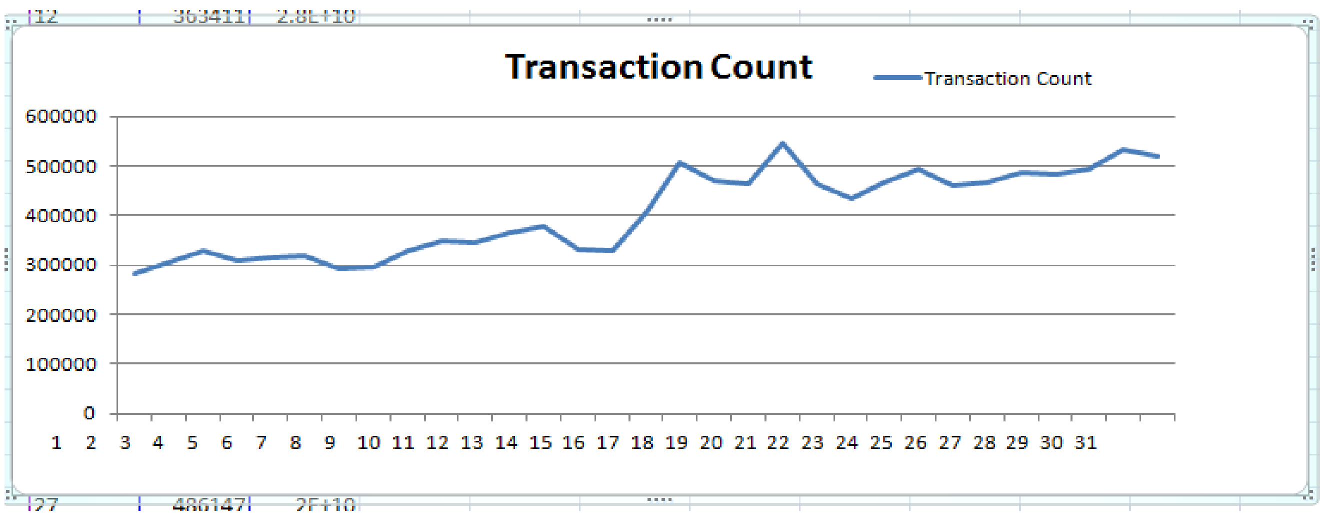
Code Explanation:

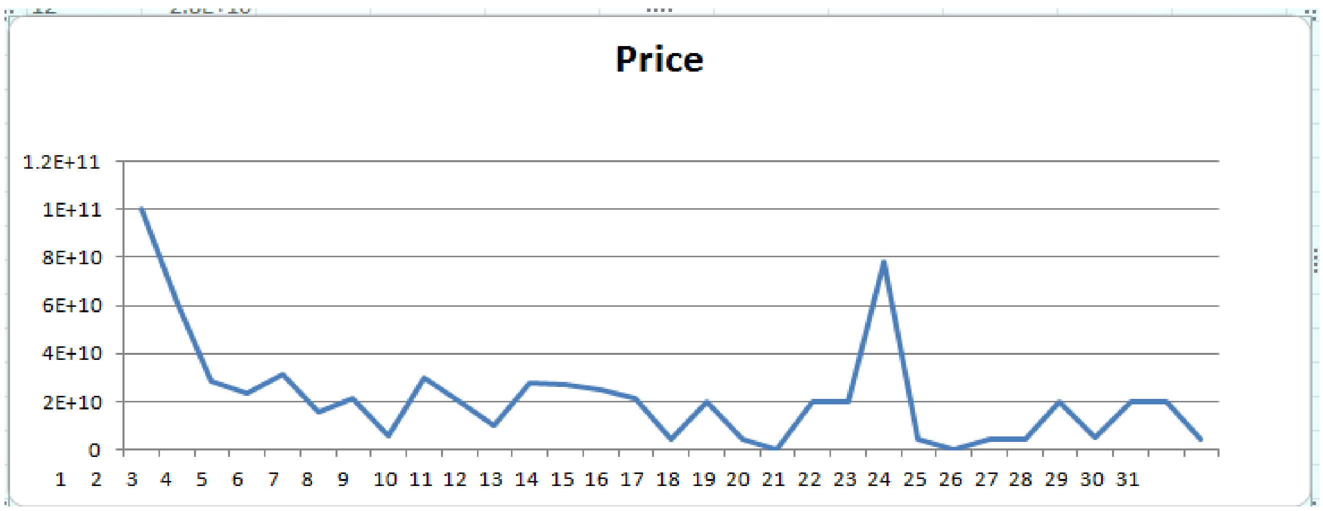
- MapReducer is used for this task to analyses the fork.
- The transaction dataset is given as input.
- In mapper, only specific month and year when fork occurred is taken as key and gas price is yielded as value.
- Aggregation of count of transaction and gas price is done by the reducer.
- Combiner is used to reduce the time taken to compute.
- The output generated is explained with plot



```
def mapper(self,_,lines):
    try:
        field = lines.split(',')
        if (len(field) == 7):
            timestamp_extraction = int(field[6])
            date = time.gmtime(float(field[6]))
            gas_price = float(field[5]) #Extracts gas price to calculate Average
            day = time.strftime("%d",time.gmtime(timestamp_extraction)) # Extract day
            month = time.strftime("%m",time.gmtime(timestamp_extraction)) # Extract month
            year = time.strftime("%Y",time.gmtime(timestamp_extraction)) #Extract year
            if (date.tm_year== 2017 and date.tm_mon== 10):
                yield((date.tm_mday), (1,gas_price))
    except:
        pass
def combiner(self,key,val):
    count = 0
    total = 0
    for v in val:
        count+= v[0]
        total= v[1]

    yield (key,(count,total))

def reducer(self,key,val):
    count = 0
    total = 0
    for v in val:
        count += v[0]
        total = v[1]

    yield (key, (count,total))
```

Graph Analysis





- The **Byzantium fork** which happened on October 16 2017 has been considered for our Fork analysis.
- From the plots we could see that the price dropped slightly after the fork and went to the lowest on 18th before rising again.
- However, the number of transaction which was steadily increasing till 15th of October, took a small dip on the day of fork(16th October). Furthermore, after fork due to decrease in price, we could see that the transaction increased exponentially for a week.

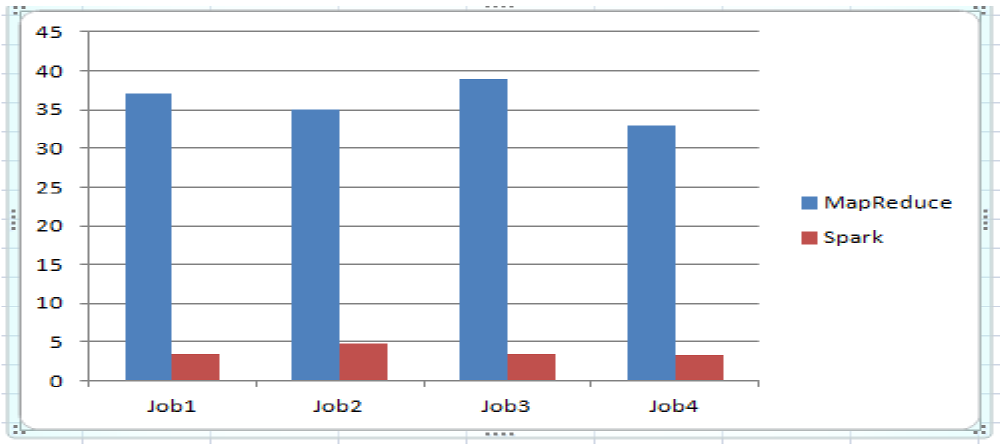
- From the above analysis, we could conclude that the due to fork on 16thOctober, the price of ETH decreased with resulted in increase in transactions.

Job ID for this task is below:
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1648683650522_7828

4.2.2 Comparative Analysis

I have executed the Part B in MapReduce, for the analysis I have executed the same program in Spark.

The time is calculated using 4 runs each of both Spark and MapReduce.



Job	MapReduce	Spark
Job1	37	3.4
Job2	35	4.8
Job3	39	3.53
Job4	33	3.3
Average	36	3.7575

Looking at the average we could see that the Spark programming is faster than MapReduce.

The reason for this is Spark processes and retains data in memory for subsequent steps, whereas MapReduce processes data on disk. As a result, for smaller workloads, Spark's data processing speeds are up to 100x faster than MapReduce.

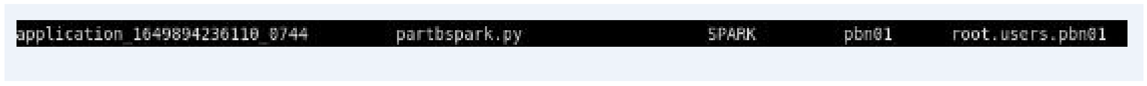
Job ID's for this tasks are

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1126

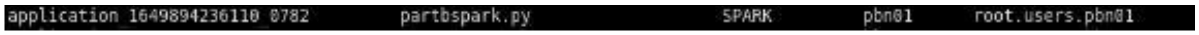
http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1292

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_1777

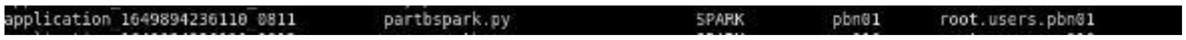
http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_1901



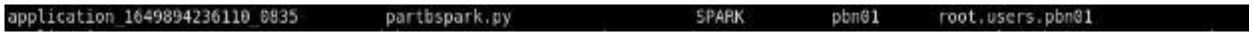
```
application_1649894236110_0744        partbspark.py              SPARK        pbn01      root.users.pbn01
```



```
application_1649894236110_0782        partbspark.py              SPARK        pbn01      root.users.pbn01
```



```
application_1649894236110_0811        partbspark.py              SPARK        pbn01      root.users.pbn01
```



```
application_1649894236110_0835        partbspark.py              SPARK        pbn01      root.users.pbn01
```

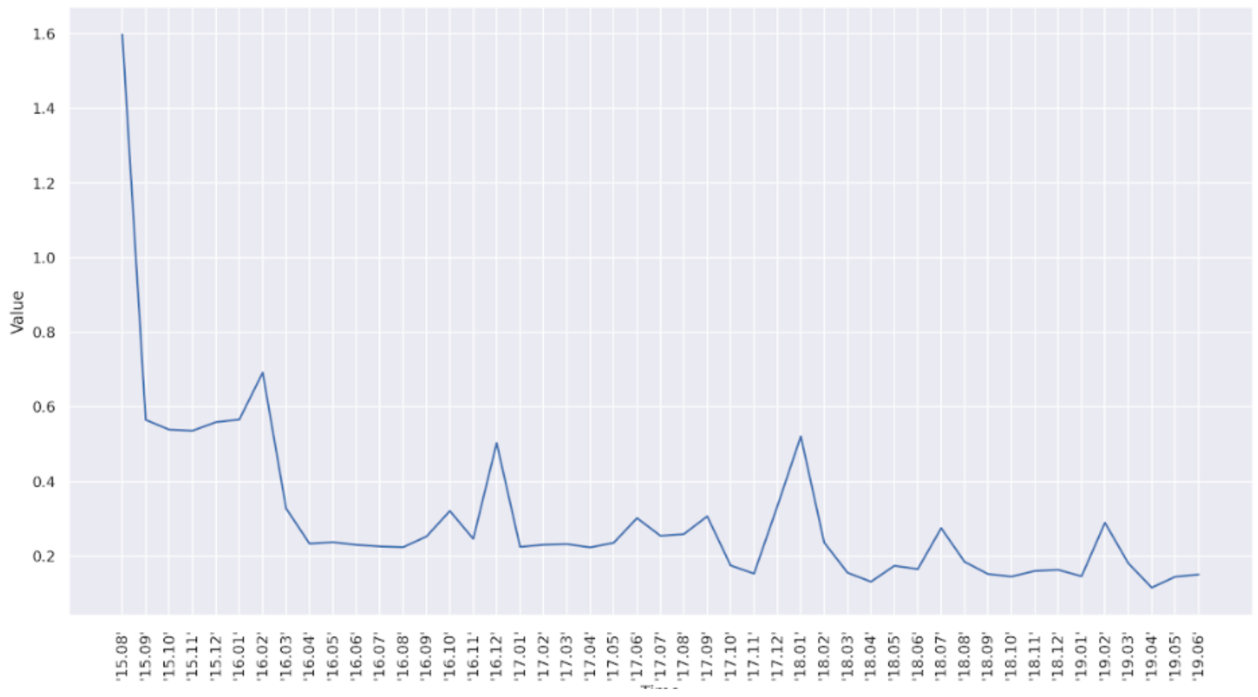
4.2.3 Gas Guzzlers

For any transaction on Ethereum a user must supply gas. How has gas price changed over time? Have contracts become more complicated, requiring more gas, or less so? Also, could you correlate the complexity for some of the top-10 contracts found in Part-B by observing the change over their transactions.

Task 1 - How has gas price changed over time?

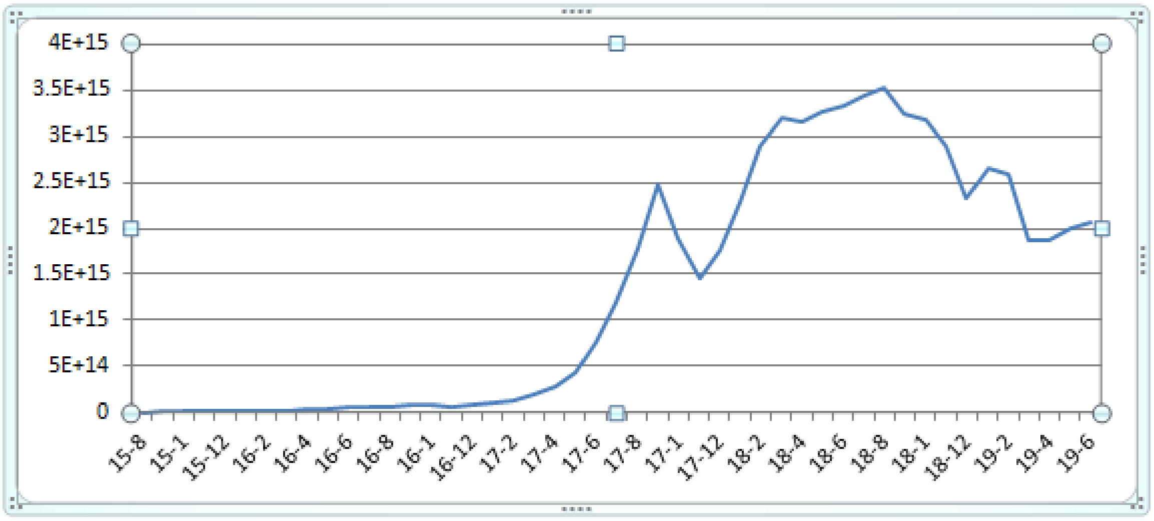
Code Explanation:

- MapReduce approach is used to compute the gas price change.
- Timestamp is extracted from the transactions dataset and considered as key whereas; gas price is taken an value.
- The average of price of each month is computed in the reducer.
- Combiner is added to reduce computation time.
- The output is shown in below plot.



Task 2: Have contracts become more complicated, requiring more gas, or less so?

Contract Complexity over time



JobID's for this task

http://andromeda.student.eecs.qmul.ac.uk:8088/proxy/application_1649894236110_4431

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1649894236110_4789

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1648683650522_7622