Big data Processing Assignment

PART A. TIME ANALYSIS (30%)

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

Note: As the dataset spans multiple years and you are aggregating together all transactions in the same month, make sure to include the year in your analysis.

Note: Once the raw results have been processed within Hadoop/Spark you may create your bar plot in any software of your choice (excel, python, R, etc.)

The raw result was being processed within Spark and was obtained as shown below:

```
def Transaction_data (trans):
    try:
        fields = trans.split(',')
        if len(fields)!=7:
            return False
        float(fields[6])
        return True
    except:
        return False
```

The above code snippet returns false if the length of the fields is not equal to 7, else it returns the `block_timestamp value of the transaction dataset.`

```
transaction=sc.textFile('/data/ethereum/transactions')
clean_trans=transaction.filter(is_good_trans)
timestamp=clean_trans.map(lambda t:int(t.split(',')[6]))
monthyears=timestamp.map(lambda my: (time.strftime("%B-%Y",time.gmtime(my)),1))
transactions=monthyears.reduceByKey(lambda a,b: a+b)
inmem=transactions.persist()
inmem.saveAsTextFile("/user/apk30/BDP_PartA")
```

In the above code snippet, the data is loaded to the variable and a filter is used. Time built in function is used to import the month and year and the output data is obtained is as shown below:

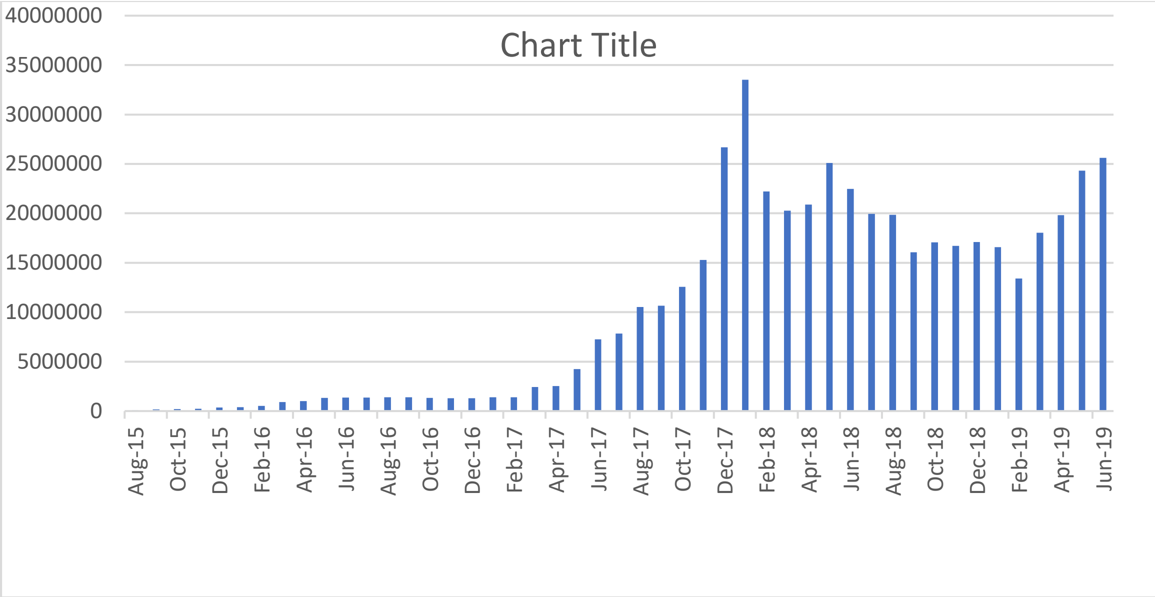
```
('October-2015', 205045)
('October-2016', 1329847)
('October-2017', 12570063)
('October-2018', 17056926)
('March-2017', 2426471)
('March-2016', 917170)
('March-2019', 18029582)
('March-2018', 20261862)
```

```
('February-2019', 13413899)
('February-2018', 22231978)
('February-2017', 1410048)
('February-2016', 520040)
('August-2016', 1405743)
('August-2017', 10523178)
('August-2015', 85609)
('August-2018', 19842059)
('July-2017', 7835875)
('July-2016', 1356907)
('July-2018', 19937033)
('May-2019', 24332475)
('May-2018', 25105717)
('April-2016', 1023096)
('May-2017', 4245516)
('April-2017', 2539966)
('May-2016', 1346796)
('April-2018', 20876642)
('April-2019', 19830158)
('January-2016', 404816)
('January-2017', 1409664)
('January-2018', 33504270)
('January-2019', 16569597)
('June-2017', 7244657)
('June-2016', 1351536)
('June-2019', 25613628)
('June-2018', 22471788)
('December-2016', 1316131)
('December-2017', 26687692)
('December-2015', 347092)
('December-2018', 17107601)
('November-2017', 15292269)
('November-2016', 1301586)
('November-2015', 234733)
('November-2018', 16713911)
('September-2015', 173805)
('September-2017', 10672734)
('September-2016', 1387412)
('September-2018', 16056742)
```

Job ID : http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1575381276332_3832

Elapsed: 2mins, 55sec

The graph was plotted in excel as shown below, after the processing of raw results in spark.



PART B. TOP TEN MOST POPULAR SERVICES (40%)

Evaluate the top 10 smart contracts by total Ether received. An outline of the subtasks required to extract this information is provided below, focusing on a MRJob based approach. This is, however, only one possibility, with several other viable ways of completing this assignment.

JOB 1 - INITIAL AGGREGATION

To workout which services are the most popular, you will first have to aggregate transactions to see how much each address within the user space has been involved in. You will want to aggregate value for addresses in the to_address field. This will be similar to the wordcount that we saw in Lab 1 and Lab 2.

JOB 2 - JOINING TRANSACTIONS/CONTRACTS AND FILTERING

Once you have obtained this aggregate of the transactions, the next step is to perform a repartition join between this aggregate and contracts (example here). You will want to join the to_address field from the output of Job 1 with the address field of contracts

Secondly, in the reducer, if the address for a given aggregate from Job 1 was not present within contracts this should be filtered out as it is a user address and not a smart contract.
JOB 3 - TOP TEN

Finally, the third job will take as input the now filtered address aggregates and sort these via a top ten reducer, utilising what you have learned from lab 4.

Job 1 : initial Aggregation .

def mapper(self, _, trans):

```
    try:
      fields = trans.split(',')
      if len(fields) == 7 :
        ad=fields[2]
        val=int(fields[3])
        yield(ad,val)
    except:
      pass

  def combiner(self, ad, val):

    yield(ad,sum(val))

 def reducer(self, ad, val):

    yield(ad,sum(val))
```

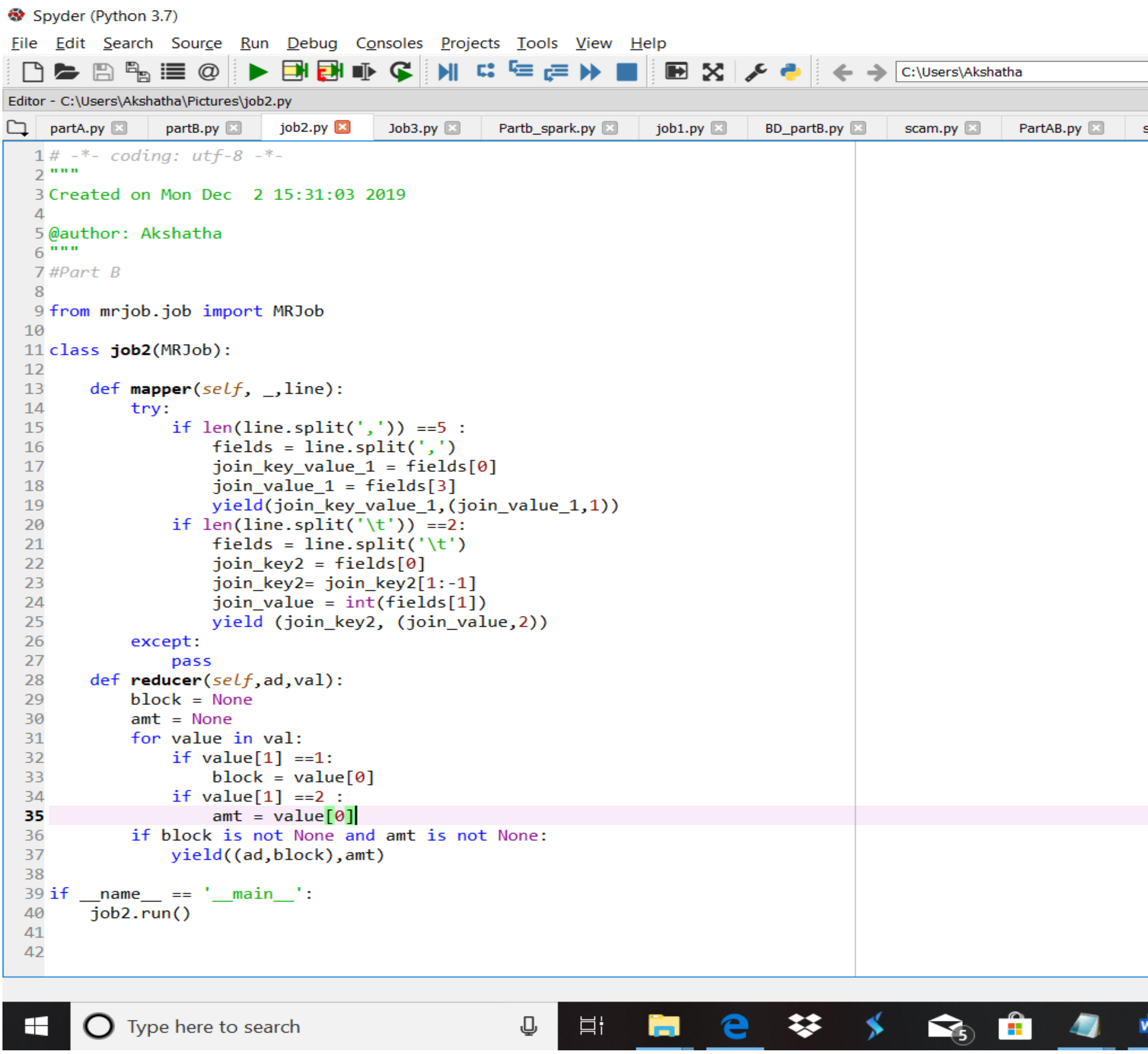
The above code snippet is used aggregate the to_address and the value of the transaction data set. The ouput of this is taken to the second job for joining transaction/contracts and filtering.

Job ID : http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1575381276332_3275

Elapsed:33mins, 16sec

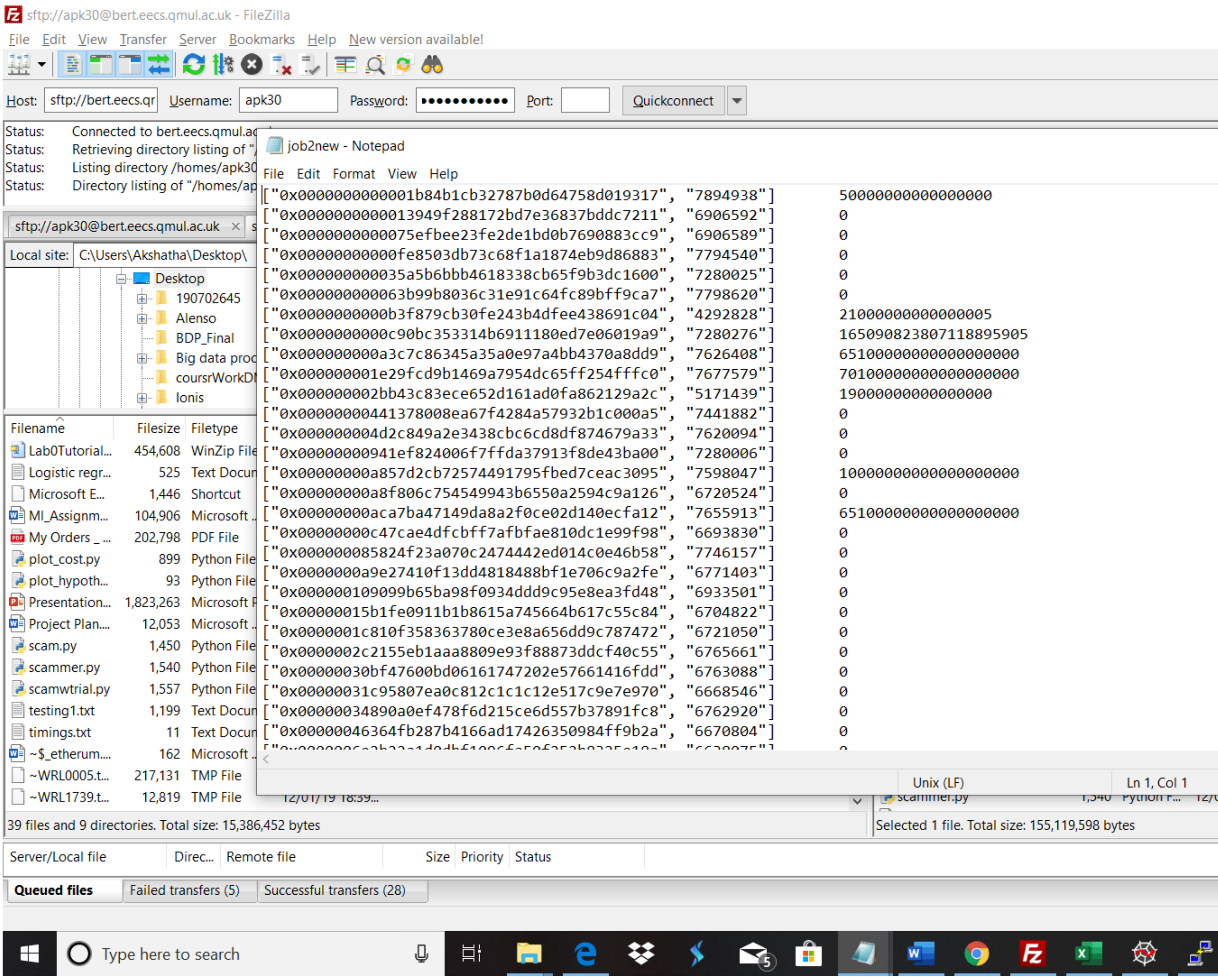
Job 2 :

Following is the code snippet for mapper and reducer that is used to join the key and value as shown below.




```python
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  2 15:31:03 2019

@author: Akshatha
"""
#Part B

from mrjob.job import MRJob

class job2(MRJob):

    def mapper(self, _,line):
        try:
            if len(line.split(',')) ==5 :
                fields = line.split(',')
                join_key_value_1 = fields[0]
                join_value_1 = fields[3]
                yield(join_key_value_1,(join_value_1,1))
            if len(line.split('\t')) ==2:
                fields = line.split('\t')
                join_key2 = fields[0]
                join_key2= join_key2[1:-1]
                join_value = int(fields[1])
                yield (join_key2, (join_value,2))
        except:
            pass
    def reducer(self,ad,val):
        block = None
        amt = None
        for value in val:
            if value[1] ==1:
                block = value[0]
            if value[1] ==2 :
                amt = value[0]
        if block is not None and amt is not None:
            yield((ad,block),amt)

if __name__ == '__main__':
    job2.run()
```

The output of this code is obtained as a .txt file , as shown below :

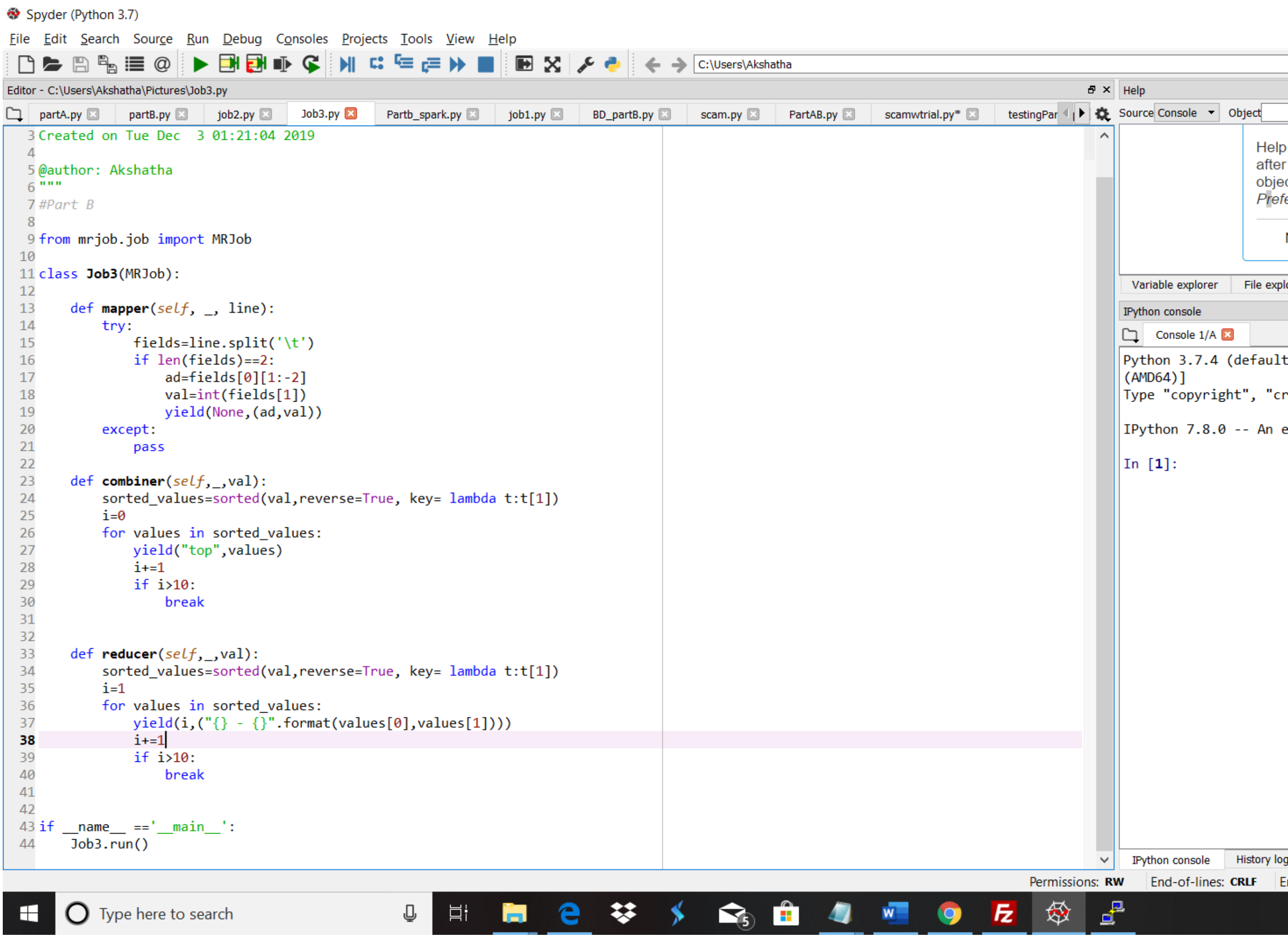


Job ID : http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1575381276332_3445

Elapsed: 7mins, 42sec

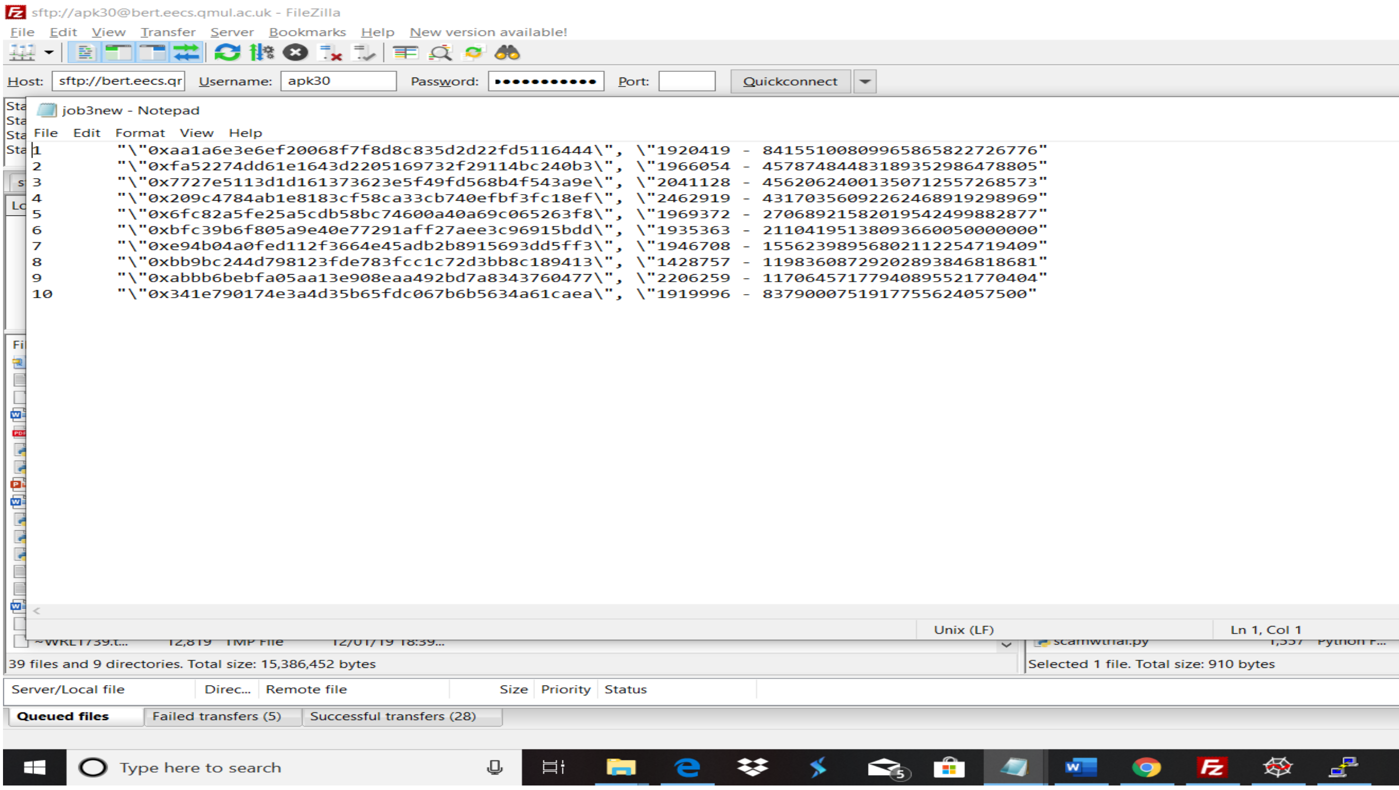
Job 3 :

The Mapper reads all the fields by splitting the line and yielding /emitting it to combiner. Split function is used eliminate '\' and ','.Combiner is used for sorting the values from each mapper, also yields the top 10 values . Reducer sorts the values reieved from the cobiner and yields the top 10 global values.



```python
Created on Tue Dec  3 01:21:04 2019

@author: Akshatha
"""
#Part B

from mrjob.job import MRJob

class Job3(MRJob):

    def mapper(self, _, line):
        try:
            fields=line.split('\t')
            if len(fields)==2:
                ad=fields[0][1:-2]
                val=int(fields[1])
                yield(None,(ad,val))
        except:
            pass

    def combiner(self,_,val):
        sorted_values=sorted(val,reverse=True, key= lambda t:t[1])
        i=0
        for values in sorted_values:
            yield("top",values)
            i+=1
            if i>10:
                break


    def reducer(self,_,val):
        sorted_values=sorted(val,reverse=True, key= lambda t:t[1])
        i=1
        for values in sorted_values:
            yield(i,("{} - {}".format(values[0],values[1])))
            i+=1
            if i>10:
                break


if __name__ =='__main__':
    Job3.run()
```

The output of this code is obtained as shown below:



Job ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1575381276332_3529

Elapsed:35sec

PART C. DATA EXPLORATION (30%)

SCAM ANALYSIS

1. **Popular Scams**: Utilising the provided scam dataset, what is the most lucrative form of scam? How does this change throughout time, and does this correlate with certain known scams going offline/inactive? (20/30)

MISCELLANEOUS ANALYSIS

Comparative Evaluation

Reimplement Part B in Spark (if your original was MRJob, or vice versa). How does it run in comparison? Keep in mind that to get representative results you will have to run the job multiple times, and report median/average results. Can you explain the reason for these results? What framework seems more appropriate for this task? (10/30)

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\Akshatha

Editor - C:\Users\Akshatha\Pictures\PartC_spark.py

PartC_spark.py | job1.py | BD_partB.py | scam.py | scamwtrial.py | scammer.py

```python
@author: Akshatha
"""

import pyspark

sc = pyspark.SparkContext()

def transactions_data(trans):
    try:
        fields = trans.split(',')
        if len(fields)!=7:
            return False
        int(fields[3])
        return True
    except:
        return False


def contracts_data(contract):
    try:
        fields = contract.split(',')
        if len(fields)!=5:
            return False
        return True
    except:
        return False

transaction = sc.textFile("/data/ethereum/transactions")
trans_filter = transaction.filter(transactions_data)
address=trans_filter.map(lambda l: (l.split(',')[2], int(l.split(',')[3]))).persist()
partbjob1 = address.reduceByKey(lambda a,b:(a+b))
partbjob1_join=partbjob1.map(lambda f:(f[0], f[1]))

contracts = sc.textFile("/data/ethereum/contracts")
contrts_filter = contracts.filter(contracts_data)
contracts_join = contrts_filter.map(lambda f: (f.split(',')[0],f.split(',')[3]))

partbjob2 = partbjob1_join.join(contracts_join)

t10=partbjob2.takeOrdered(10, key = lambda x:-x[1][0])
for record in t10:
    print("[]: []".format(record[0],record[1][0]))
```

Type here to search

The output obtained for the code is as shown below:

bert.eecs.qmul.ac.uk - PuTTY

```
##############################################################
-bash-4.1$ ssh itl001
apk30@itl001's password:
Last login: Thu Dec  5 04:02:06 2019 from bert.student.eecs.qmul.ac.uk
Kickstarted on 2019-09-09
apk30@itl001 ~> module purge
apk30@itl001 ~> module load hadoop/cdh-6.3.0
apk30@itl001 ~> pyspark
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
19/12/05 09:25:34 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attem
pted to request executors before the AM has registered!
19/12/05 09:25:34 WARN lineage.LineageWriter: Lineage directory /var/log/spark/l
ineage doesn't exist or is not writable. Lineage for this application will be di
sabled.
19/12/05 09:25:34 WARN lineage.LineageWriter: Lineage directory /var/log/spark/l
ineage doesn't exist or is not writable. Lineage for this application will be di
sabled.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.0-cdh6.3.0
      /_/

Using Python version 2.7.5 (default, Apr 11 2018 07:36:10)
SparkSession available as 'spark'.
>>> exit()
apk30@itl001 ~> pwd
/homes/apk30
apk30@itl001 ~> cd CourseWork
apk30@itl001 ~/CourseWork> spark-submit PartC_spark.py
19/12/05 09:26:50 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attem
pted to request executors before the AM has registered!
19/12/05 09:26:50 WARN lineage.LineageWriter: Lineage directory /var/log/spark/l
ineage doesn't exist or is not writable. Lineage for this application will be di
sabled.
19/12/05 09:27:07 WARN cluster.YarnScheduler: Initial job has not accepted any resources; check your cluster UI to ensure that workers are regist
0xaa1a6e3e6ef20068f7f8d8c835d2d22fd5116444: 84155100809965865822726776
0xfa52274dd61e1643d2205169732f29114bc240b3: 45787484483189352986478805
0x7727e5113d1d161373623e5f49fd568b4f543a9e: 45620624001350712557268573
0x209c4784ab1e8183cf58ca33cb740efbf3fc18ef: 43170356092262468919298969
0x6fc82a5fe25a5cdb58bc74600a40a69c065263f8: 27068921582019542499882877
0xbfc39b6f805a9e40e77291aff27aee3c96915bdd: 21104195138093660050000000
0xe94b04a0fed112f3664e45adb2b8915693dd5ff3: 15562398956802112254719409
0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 11983608729202893846818681
0xabbb6bebfa05aa13e908eaa492bd7a8343760477: 11706457177940895521770404
0x341e790174e3a4d35b65fdc067b6b5634a61caea: 8379000751917755624057500
apk30@itl001 ~/CourseWork>
```

Type here to search

Job ID :

http://andromeda.student.eecs.qmul.ac.uk:8088/cluster/app/application_1575381276332_4230

Elapsed:5mins, 10sec

Spark executes jobs faster than when compared to MR job, as per the elapsed time as well spark is quicker. Spark uses in memory processing which is better than map reduce codes which gets stored in a disk, this is a disadvantage for map reduce since the next job depends on the previous job output , which will consume a lot a time.