Queen Mary
University of London

BIG DATA PROCCESING

Bitcoin Analysis Using Big Data Jobs

Coursework1: "Analysis of Bitcoin Transactions"

Lecturer: Dr. Felix Cuadrado

Student Name: Frank Erasmo Cruz Felix
ID: 150684871

December 8th, 2018

Contents

Introduction

This report gives an overview of the implementation of Hadoop and Spark for the different tasks in the coursework. A subset of CVS files has been used which are: blocks.csv, transaction.csv, coinger.csv (Coin Generations/Mined coins), vin.csv and vout.csv. A brief description is given for each one as following [4]:

- **Blocks.csv:**
  When a block is completed, all the past transactions belonging to it become a permanent record and a new block in the blockchain is created.

height: The block number

hash: The unique ID for the block

time: The time at which the block was created (Unix Timestamp in seconds)

difficulty: The complexity of the math problem associated with the block

Sample entry:

0,
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f, 1231006505,1

- **Transactions.csv**
  Blocks record all transactions occurring at any time while they continue unsolved.

  Dataset schema:

tx_hash: The unique id for the transaction

blockhash: The block this transaction belongs to

time: The time when the transaction occurred

tx_in_count: The number of transactions with outputs coming into this transaction

tx_out_count: The number of wallets receiving bitcoins from this transaction

Sample entry:

0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098,00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048,1231469665,1,1

- **Coingen.csv**
  When miners solve a problem, an amount of Bitcoins (BTC – Bitcoin currency symbol) is given to the miner, which goes circulation.

  Dataset scheme:

db_id: The block the coin generation was found within (maps to block height not hash)

tx_hash: Transaction id for the Coin Generation

coinbase: Input of a generation transaction. As a generation transaction has no parent and creates new coins from nothing, the coinbase can contain any arbitrary data (or perhaps a hidden meaning).

sequence: Allows unconfirmed time-locked transactions to be updated before being finalised

Sample entry:

1, 0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098, 04ffff001d0104, 4294967295

- **Vin.csv**
  Source of transaction(s) of the coins.

  Dataset scheme:

txid: The associated transaction the coins are going into

tx_hash: The transaction the coins are coming from

vout: The ID of the output from the previous transaction - the value equals n in vout below

Sample entry:

f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16,0437cd7f8525ceed2324359c2d0ba26006d92d856a9c20fa0241106ee5a597c9,0

- **Vout.csv**
  Source of destination wallet(s)

  Dataset scheme:

  hash: The associated transaction

  value: The amount of bitcoins sent

  n: The id for this output within the transaction. This will equal the **vout** field within the vin table above, if the coins have been respent

  publicKey: The id for the wallet where the coins are being sent

  Sample entry:

```
0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098, 50, 0, {12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX}
```

Overview

Bitcoin is the first decentralized digital/ virtual currency. It an electronic payment method that uses public-key cryptography, peer-to-peer networking, and proof-of-work to process and verify payments. A person or group under the pseudonym of Satoshi Nakamoto developed it in 2018 and it was introduced since 2009[1].

The entire Bitcoin network depends on the blockchain, which is a shared public ledger. All successful transactions are added in the blockchain (see Figure 1). Cryptography enforces the integrity and historical order the blockchain [2].

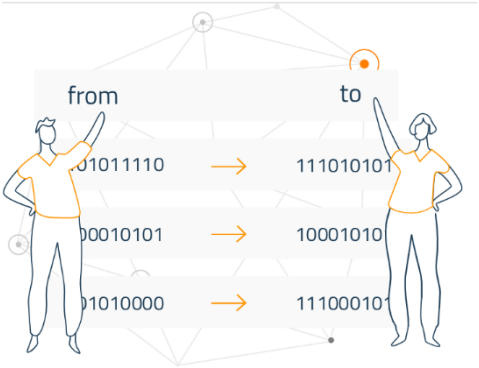


Figure 1: Balances – block chain

A transaction is a conversion of values between Bitcoins wallets that are included in the blockchain, meaning that these coins can be used twice. A private key (see Figure 2) or seed (piece of data) is kept by the bitcoins wallets, which is used to sign transactions. A mathematical proof is computed in order to check if those bitcoins have come from the owner wallet. Thus, the signature avoids the transactions to be modified by any people once it has been broadcasted. A confirmation of the transaction is generated within 10 and 20 minutes after the transaction has been broadcasted to the network, this process is called mining [2].

Figure 2 – Transactions - private keys

Mining is a distributed consensus system, which is used to approve incomplete transactions by adding them to the blockchain (see Figure 3). The neutrality of the network is protected by enforcing a chronological order in the blockchain and this allows distinct computers to acknowledge in the state of the system. Transactions must be stored in a block to be confirmed. This block has very strict cryptographic rules that are verified by the network. These rules ensure that nobody modifies, adds blocks chain. It also restricts individuals or group manage what is added in the blockchain or change parts of them [2].

Figure 3 – Processing - Mining

Assignment

Specifications:

Write a set of jobs using either Map/Reduce or Spark that process the given input and generate the data require answering the following questions:

Run instructions: Run in the terminal

module load python/3.7.0

Part A: Time Analysis

In this part, it is asked to create a bar plot showing the number of transactions which occurred every month between the start and end of the dataset.

For this task, it was used Map/Reduce. The full code of this part has been compressed and put along with this report.

Mapper function:

In the mapper function, a line read from the transactions.csv file is passed here. First, the line is split using the split() function each element a comma is found and an array fields stores these split values. In the try block, the length of the array is checked to ensure that the line contains 5 elements in it, otherwise, this is sent to the exception block. The third element is selected for this purpose (fields[2]), which contains the time when the transaction occurred. This field is converted into a fixed date. The values are stored in the array date (**the year - %Y** - in decimal, **month -%B** -full month name, number **-%m - month** as a decimal number). These values are yielded to the combiner function as a Key and it is given one as a value (see Figure 4).

```
def mapper(self, _, line):
    fields = line.split(",")# Split lines
    try:
        if (len(fields)== 5):#only carry on if fields len is equal to 5
            time_epoch = int(fields[2])#get the third field of fields array
            date = time.strftime("%Y-%B-%m",time.gmtime(time_epoch))#convert time_epoch into fix date_fields
            date_fields = date.split("-")
            year = date_fields[0]# %Y --> Year with century as a decimal number
            month = date_fields[1]# %B --> Full month name
            number = date_fields[2]# %m --> Month as a decimal number [0,12]
            yield((year,number,month),1)
    except:
        pass
         #no need to do anything, just ignore the line, as it was malformed
```

Figure 4 – Mapper Function

Combiner function:

The combiner function created has only one single job, which is to sort the values according to the value of the second element. Then, these sorted key values are sent to the reducer by adding their values if they have the same key.

```
def combiner(self, line, sums):
    sorted_values = sorted(line, reverse= True,key=lambda x: x[1])
    yield(line,sum(sums))
```

Figure 5 – Combiner function

Reducer function:

The reducer gets the values and then print them in the PartAtransacations file. The number of reducers has been configured (PartA.JOBCONF = {'mapreduce.job.reduces': '1'}() , having a single see text file saved in Hadoop (see Figure 6)

```
def reducer(self, line, sums):
    print("{}, {}, {}".format(line[0],line[2],sum(sums)))#Print lines in the format "{}, {}, {}"
    #yield(",{}, {}, {},".format(line[0],line[2],sum(sums)),1)
```

Figure 6 – Reducer Function

The next link shows how this python code was run:

../ecs640/Coursework/PartA> python partA.py -r hadoop --output-dir PartAtransacations --no-output hdfs://studoop.eecs.qmul.ac.uk/data/bitcoin/**transactions.csv**

What do you notice about the overall trend in the utilisation of bitcoin?

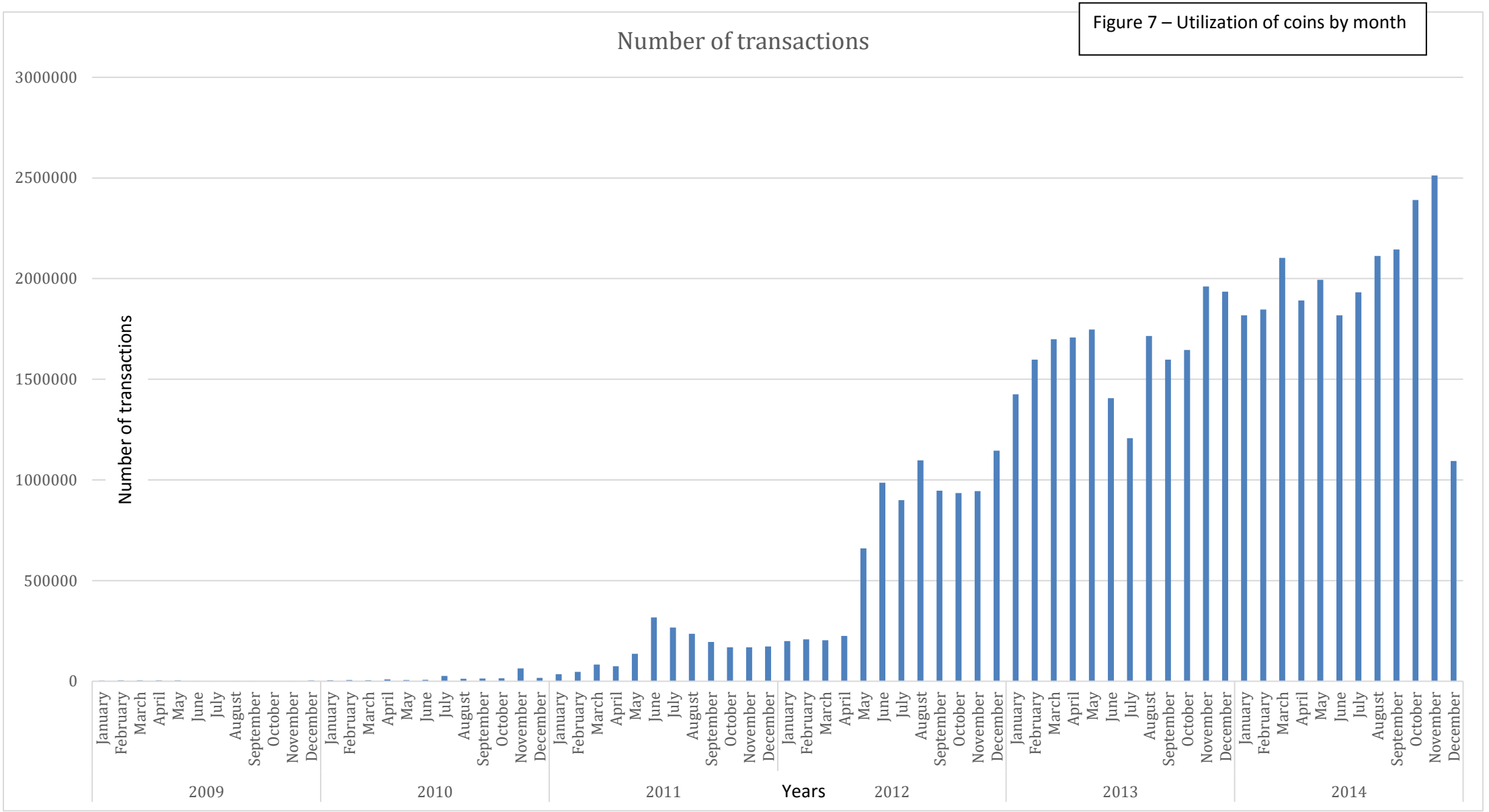
The bar chart below shows the number of transactions made between 2009 and 2014. As it can be seen in the diagram, during the first two years there are not too many transactions made as compared with the other half of this total period. During the first half of the period, the first highest peak happened in June of 2011 with 317482 transactions.

After this first half, there is a slightly decreased in transactions made from the following month July 2017 to April 2012. However, the number of transactions rose rapidly from May 2012 to November 2014, having as the highest peak in November 2014 with 2 512 559 transactions made.

To conclude, the trend from data filtered shows that the number of transactions increased fast over the period and about 2 years after, bitcoins become popular and were used more frequently to do online payments or exchange. The bar chart was made in excel from the data filtered and stored in the folder with name Part A. A date from 1970 with a value were removed as it is believed that this was produced due to import time header. Furthermore, transactions started in 2009.

Number of transactions per year:

Figure 7 – Utilization of coins by month



Part B: Top ten donors

In this part of the coursework, it is asked to obtain the top ten donors from the **WikiLeaks** using its public key (wallet - 1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v)

For this task, it was used spark. The full code of this part has been compressed and put along with this report.

During this part, the files filtered, first join and second join have been saved in Hadoop to reduce the number of partitions used when running the program in the cluster.

Initial Filtering

The dataset vout.csv is read as a collection of lines. These lines are sent to the wallet_key function by using the filter function. In this function, data is filtered by using the public key (wallet - 1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v). The lines of code matching this publickey are kept, split after a comma and then saved in the array variable filterwallet.

Filtered lines are mapped in way that allows us to can be used with ease. Lastly, these mapped lines are saved in Hadoop as a text file with the name FilteredPartB. The first element of this filtered data f[0], contains the transaction number in which this wallet received the bitcoins and will be used in our second join.

First join

For this part, The filtered file saved in Hadoop is used (**FilteredPartB**). It is read and mapped as a **key l[0]** (contains the transaction number - hash), and **value l[1] (**contains the value converted into a float by using the function getFloat). The vin.csv dataset is read and mapped as **a key l[0]** (contains the associated transaction the coins are going into - txid) and **values l[1]** (tx_hash) and **l[2] (vout).** The join matches the keys of these two datasets and if they are equals, it kepts the lines which are then stored in the first_join variable. Thus, this new collection is saved in Hadoop as a text file with the name **FirstJoinPartB**. The main values that will be used for the second join, are **second** and **third** elements l[1] (contains the transaction where the money is sent) and l[2] (the vout value) respectively.

```
first_join = vin_f_join.join(vout_split)
## New Data is formatted and saved in Hadoop as FirstJoinPartB
first_hadoop = first_join.map(lambda l: "{},{},{},{}".format(l[0],l[1][0][0],l[1][0][1],l[1][1]))
first_hadoop.repartition(1).saveAsTextFile("FirstJoinPartB")
```

Figure 8 – First Join

Second Join

For this part, the **first join** file is used. It is read and mapped as **a key l[1]** (contains the transaction number -**tx_hash**), **l[2]** (contains the value- **vout**) and a **value l[3]** (contains the amount of bitcoins which is converted into float).

```
#Mapping values as a key and value form
vin_f_join = vin_function.map(lambda l: ((l[1],l[2]),(l[0],float(l[3]))))
```

Figure 9 – FirstJoinPartB

The **vou.csv dataset** is read and mapped as a **key l[0]** (contains the associated transaction the coins are going into - hash), l[2] (contains the output within the transaction - n); and **values l[1]** (amount of bitcoins sent which is converted into a float) and **l[3]** (**the wallet id**). See Figure 10.

```
## Mapping lines as key and value
vout_f_join = vout_function.map(lambda l: ((l[0],l[2]),(float(l[1]),l[3])))
```

Figure 10 – vout.csv map

The **second join** is made as it can be seen in Figure 11. The join returns lines that have the same key. These new lines from the second join are mapped and saved in Hadoop as **SecondJoinPartB** text file. This new file contains the amount of bitcoins sent per block from the publikey f[1][1][1], f[1][0][1] the publickey, f[1][0][0] the amount received by block transfer and f[1][1][0] the transaction number.

```
## Second Join
second_join = vout_f_join.join(vin_f_join)
# join data is mapped as follows:
money_from = second_join.map(lambda f: "{}, {}, {}, {}".format(f[1][1][1], f[1][0][1], f[1][0][0], f[1][1][0]))
## This second join is saved in Hadoop
money_from.repartition(1).saveAsTextFile("SecondJoinPartB")
```

Figure 11 – Second Join

After successfully creating the second join and obtaining the data, it is now possible to find the top ten donors. Following the previous steps to read files and map them to do in the SecondJoinPartB text file. The function takeOrdered is used for this purpose, having as parameters 10 (top ten) and key – x[1] (descending order). See Figure 12.

```
top10 = vout_total_donate.takeOrdered(10, key = lambda x: -x[1])
##Print Top ten to the screen
for record in top10:
    print("{} - {}".format(record[0],record[1]))
```

Figure 12 – Top ten donors

Top ten donors

Top ten donors in dollars sorted according to bitcoins:
BTC/GBP rate was taken in 09/12/12 11:30

Rank	Public Key	Bitcoins (BTC)	Amount in Pounds(£)
1	17B6mtZr14VnCKaHkvzqpkuxMYKTvezDcp	46515.1894803	123,593,182.88
2	19TCgtx62HQmaaGy8WNhLvoLXLr7LvaDYn	5770.0	15,331,151.93
3	14dQGpcUhejZ6QhAQ9UGVh7an78xoDnfap	1931.482	5,274,548.99
4	1LNWw6yCxkUmkhArb2Nf2MPw6vG7u5WG7q	1005.30353679	2,743,593.99
5	1L8MdMLrgkCQJ1htiGRAcP11eJs662pYSS	806.13402728	2,199,543.39
6	1ECHwzKtRebkymjSnRKLqhQPkHCdDn6NeK	648.5199788	1,770,360.90
7	18pcznb96bbVE1mR7Di3hK7oWKsA1fDqhJ	637.04365574	1,739,033.62
8	19eXS2pE5f1yBggdwhPjauqCjS8YQCmnXa	576.835	1,575,099.88
9	1B9q5KG69tzjhqq3WSz3H7PAxDVTAwNdbV	556.7	1,520,119.45
10	1AUGSxE5e8yPPLGd7BM2aUxfzbokT6ZYSq	500.0	1,364,475.00

Part C: Data Exploration

For this part, it has been chosen the Ransomware case. Ransomware often gets victims to pay via bitcoin.

It is using **Crytolocker ransom** public key (wallet - 1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc)

The code was written as to make use of spark. The full code of this part has been compressed and put along with this report. Part C has was divided into 2 parts: Looking at keys which sent more bitcoins to the Ransome wallet and the other where this bitcoins are sent.

Bitcoins coming from:

Initial Filtering:

For this part **(PART I),** data is filtered according to the ransomware wallet by using the fucntion wallet_key. The filtered data is mapped and then saved in Hadoop as a text file as RansomwareFiltered. Doing this part, it was possible to obtain the list of transactions from where Cryptolocker received the bitcoins.

First Join

For the next part **(PART II),** the RansomwareFiltered file saved in Hadoop is read and mapped as a key **l[0]** (contains the transaction associated - **hash**) and a value **l[1]** (amount of bitcoins received by per block Criptolocker – **value**, which is converted into a float number).

```
vout_split = vout_function.map(lambda l: ((l[0]),(float(l[1]))))
```

Figure 14 – RamsomwareFiltered

Vin.csv file is read and then map as a **key l[0]** (transaction number where bitcoins are going- **hash**) and **two values l[1]** (contains transaction serie where coins are coming from -**txid**) and **l[2**] (contains the vout value- **vout**). The first join is made if both keys match; the values are kept and saved in the array first_join. The new collection of lines are map and put in a format that it can be used with easy, then they are saved in Hadoop as a text file RansomwareFirstJoin.

```
first_hadoop = first_join.map(lambda l: "{},{},{},{}".format(l[0],l[1][0][0],l[1][0][1],l[1][1]))
first_hadoop.repartition(1).saveAsTextFile("RansomwareFirstJoin")
```

Figure 15 – First join

Second Join

In this section **(PART III)**, the second join is made. Firstly, the first join is read, mapped as **a key l[1]** (contains the transaction number -**tx_hash**), **l[2]** (contains the value- **vout**) and a **value l[3]** (contains the amount converted into float- **value**). See Figure 16.

```
vin_f_join = vin_function.map(lambda l: ((l[1],l[2]),(l[0],float(l[3]))))
```

Figure 16 – Second join

The full vou.csv dataset is read and mapped as **a key l[0]** (contains the associated transaction the coins are going into - **hash**), **l[2]** (contains the output within the transaction - **n**) and **values l[1]** (**amount** of bitcoins sent which is converted as a float number) and **l[3]** (the **wallet id**). See Figure 17.

```
## Lines are mapped as key and value
vout_f_join = vout_function.map(lambda l: ((l[0],l[2]),(float(l[1]),l[3])))
```

Figure 17- Full vout dataset.

The second join is performed. If the key values of both files match, then the lines are kept and saved in the second_join variables (see Figure 18). The second join is mapped and saved in Hadoop, which is later used to find the top ten. From the second join, following data was obtained: the amount of value sent by publickey f[1][1][1] by transaction, the public key f[1][0][1] which sent bitcoins to Cryptolocker, the amount received by the Ransomeware wallet f[1][0][0] and the transaction number in which the Ransomware wallet received the money.

```
## Second join performed
second_join = vout_f_join.join(vin_f_join)
## Second join data is mapped as follows:
money_from = second_join.map(lambda f: "{}, {}, {}, {}".format(f[1][1][1], f[1][0][1], f[1][0][0], f[1][1][0]))

##(transaccion receiveing money, publick key, amount sent in the transacaction, total receive by C)
## Second Joined is saved in Hadoop
money_from.repartition(1).saveAsTextFile("RansomwareSecondJoin")
```

Figure 18 - Second Join: RansomwareSecondJoin

Top 10 wallets who sent more bitcoins to Crytolocker ransom

The second join saved in Hadoop is read, mapped as previously. For this part, the lines are mapped as a **key l1]** (contains the **publickey** which sent bitcoins to Cryptolocker) and **value l[2]** (containing the bitcoins sent to the **Ransomware wallet**). The function used in order to find the top ten wallets who sent a big amount of bitcoins to the Ransomware wallet is **reduceKey().** This function matches the key values of each line and if they are equal, keeps them and stores them. Then these new lines are sorted according to the second element. The top ten wallets who sent a big amount of bitcoins are printed to the screen.

```
## They are reduced by key and if they have the same key
vout_total_receive = trans_function.map(lambda l: (l[1],float(l[2]))).reduceByKey(lambda v1,v2: v1+v2)
#Order and print the 10 wallets which sent more bitcoins to Cryptolocker ransom
top10 = vout_total_receive.takeOrdered(10, key = lambda x: -x[1])
```

Figure 19 – Top 10 wallets who sent more bitcoins to the Ransom wallet.

Exchange rate bitcoins to pounds (1 bitcoin → 2,654.55 Pound sterling (8 Dec, 18:54 UTC)))

Public key	Bitcoins (BTC)	Pounds (£)
19z4mf9RF2R2ybjaLXsCB5EmMgKUWxDjAo	1256.4770329500002	3 335 381.108
1ACKcumkx4M3aQisMMLq32EubPkUNiUfTC	652.0	1 730 766.60
1CKYJwA5THG8nCZWvHs2RbjuVf9GfKMHkM	370.0	982 183.5
1AtFozpzXv6Zec8gBJSfdrksDKtJj1dr9G	270.5214228	718 112.6429
1Nrms8HxFqiRWT7qVeZqercMddhUZqrqHq	258.63	686 546.2665
1BEG3QKfoC1DwwQ2fNZgjNMNPr9P2R3zhb	175.64	466 245.162
1DCcmXAMpZMd2FWM7kwMcvs2ErwVbNtoSa	171.47707095000004	455 194.4586
131Hyktbz2zV7iDUKunMrJCkgkDiSy1k3p	159.32	422 922.906
1BZtUXz3bBAtbVQjyEWpPyVNSmpbBYwukG	118.13814286	313 603.6071
15Ncjs5uSL4CXUVjPdsYEdamRUauqgcjAU	117.25725346	311265.2422

Bitcoins going to:

First Join

The first join is performed between the vin.csv and vout.csv datasets. The vout dataset is mapped as a key **l[0]** (contains the transaction associated - **hash**), **l[2]** (contains the output within the transaction -**n**) and **a value l[1]** (contains an amount of bitcoins - **value**), **l[3]** (contains the **publickey** which received the coins). In the same case for vin, it is mapped as a **key l[1]** (contains the transaction bitcoins are coming from), **l[2]** (contains the output from previous transactions- **vout**) and a **value l[0**] (contains the associated transaction the coins are going into). The first join is performed between these two datasets. If the keys are equal, the lines are **saved in** first_join variable. These new lines are mapped in form to be used with ease.

A function was created in order to filter and return all lines from the first join only if their publickey is equal to the Ransomware wallet (Cryptolocker's wallet - 1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc). These filtered lines are mapped and stored in Hadoop as RansomwarePartB text file.

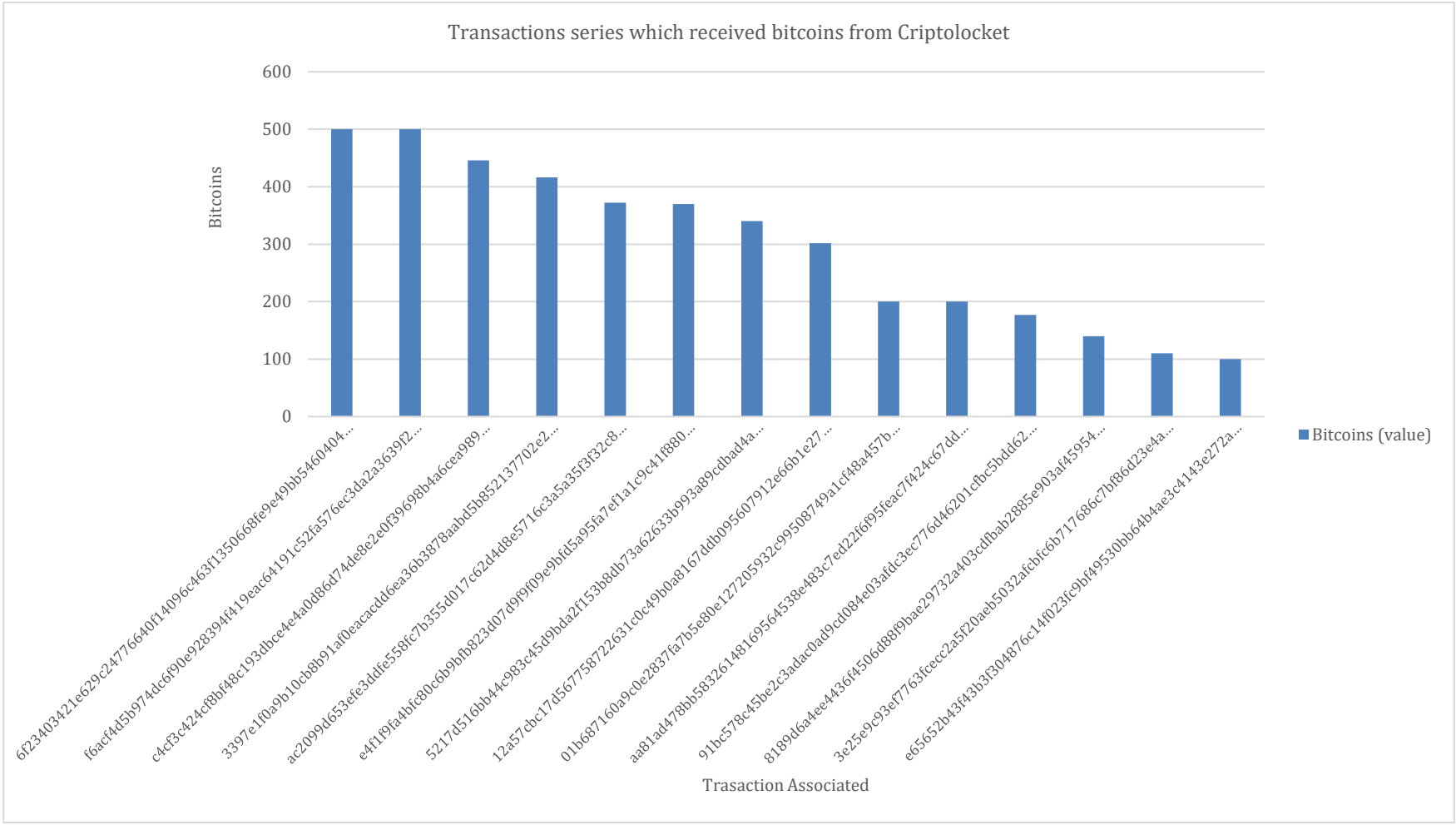
For this last part, the total number of bitcoins sent out from this wallet was calculated (**SUM TOTAL**). Firstly, the text file RansomwaredPartB saved in Hadoop is read and mapped as a key l[3] (contains the publickey **Crypyoclocker**) and a value l[4](contains the value converted into float). The function redceByKey is used in order to match all keys and sum theirs values. The total number of bitcoins sent out from this wallet were:

Exchange rate bitcoins to pounds (1 bitcoin → 2,654.55 Pound sterling (8 Dec, 18:54 UTC)))

Public key	Bitcoins (BTC)	Pounds (£)
1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc	5338.98288731	14 172 597.02350

Transactions Associate graph

As it can be seen, Ransome sent all bitcoins out of it wallet to these wallets. A diagram showing the top fifteen transaction series which received bitcoins has been plotted.



A better graph can be seen in the excel file inside of the folder called “TransactionNumbersSent”. This diagram shows the top fifteen transactions series, which received the bitcoins after the crime was committed. It can been that the highest amount of bitcoins sent were of two different transactions numbers with the amount of 500 bitcoins. Thus, it can also see from the excel file “TranferedCoins” that the least amount of bitcoins sent were 1.00E-08.

It has been used a map/reduce. The file stored in Hadoop RansomeDates is been used. The map/reduce code is called “transID.py”.The lines are passed and split in the mapper function. The only elements taken are fields[2] (the amount sent by Cryptolocker converted into a float value) and fields[0] (contains the transaction associated). A combiner function has been added which add the values if keys are equal.

Furthermore, a bar chart has been plot according to the date in which these transactions were made. Map/reduce was used. The file stored in Hadoop RansomeDates has been used. The map/reduce code is called “transID.py”.The lines are passed and split in the mapper function. The only elements taken are fields[2] (the amount sent by Cryptolocker converted into a float value) and fields[0] (contains the transaction associated). A combiner function has been added which add the values if keys are equal.

Furthermore, a bar chart has been plot according to the date in which these transactions were made. It has been so far interesting how data is filtered and I wanted to show the next diagram as an extra part which was not consulted with the lecturer.

A map/reduce has been implemented for this part. The file python has been named getTime and it has been attacked to this coursework.

The file stored in Hadoop RansomeDates has been used. In the mapper function the elements considered in the lines are fields[1] (contains the date in which the transaction was made, which is converted to a readable form) and field[2] (contains the number of bitcoins converted into a float number). The combiner sorts them according to number (month as a decimal number[0,12]) and then lines having the same key, their values are sum up. The reducer function prints the lines of code and the folder DatesbyDay is created.

Year	Month	Day	Amount
2013	October	15	206.1543
		16	332.4097
		17	398.13
		18	57.969
		19	37
		20	267
		21	112.359
		22	644.2353
		23	36.55368
		24	29.4399
		25	55.16056
		26	20.21
		28	108.788
		29	12.70075
		30	237.558

		31	365.1
	November	1	63.35664
		4	517.33
		5	20.92
		6	652.1132
		7	29.49
		8	393.26
		9	622.64
		11	97.735
		12	21.3699
		12	21.3699

References

1. Bitcoin.org. (2018). [online] Available at: https://bitcoin.org/bitcoin.pdf [Accessed 8 Dec. 2018].
2. Bitcoin.org. (2018). *How does Bitcoin work? - Bitcoin*. [online] Available at: https://bitcoin.org/en/how-it-works [Accessed 8 Dec. 2018].
3. Qmplus.qmul.ac.uk.(2018). *Assignment*.[online]Available at:Qmplus.qmul.ac.uk. (2018). *Assignment*. [online] Available at: https://qmplus.qmul.ac.uk/mod/assign/view.php?id=846563 [Accessed 8 Dec. 2018]. [Accessed 8 Dec. 2018].
4. Blockchain.com. (2018). *Bitcoin Address 1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc*. [online] Available at: https://www.blockchain.com/btc/address/1AEoiHY23fbBn8QiJ5y6oAjrhRY1Fb85uc [Accessed 8 Dec. 2018].