Analyzing email reply chains using mrjob and Hive on Enron email data set

Tomasz Gliniecki and Denys Chechelnytskyy

University of Stavanger

Abstract. This project concentrates on processing information from large data set. The data set used was Enron Corporation email data set [4]. In todays world of big data and big sets of data, a new way of data processing had to be found. For some years ago, processing the data on a distributed system was introduced. As it is not efficient to process huge amounts of data on a single drive/server because of big difference in Read/Write (R/W) speed of hard drives vs processors. The hard drive on a single server system will always be a bottleneck as it can never supply the same speed of data delivery to the processor as the speed the processor is processing the data. There are special frameworks for distributed storage and processing of large data sets [5]. Apache Hadoop being on example takes leading positions. It is open source and consists of different modules such as Hadoop Common, Hadoop Distributed File System (HDFS), YARN and MapReduce [6]. Hadoop stores data in HDFS (Hadoop Distributed File System), all files are split and distributed amongs nodes in the system, and also replicated for failure recovery. MapReduce is a programming paradigm to access and process large data spread across HDFS.

Keywords: enron, data analysis, email, sorting, mapreduce, hadoop

1 Overview of our approach

In this section we aim to describe our approach and draw the outline of information we are looking for. We will connect data-intensive approach with meaningful insights.

The goal of the project is to retrieve the relevant information from Enron email data set. We know that Enron data set was retrieved after Enron corporation went bankrupt. There are many possible ways to analyze and retrieve information from the data set. What we are trying to explore is how could MapReduce be used to analyze this type of data. Given the circumstances of how our testing environment looks like, since its mostly one local machine, there is hardly any meaningful data as far as the running time of different algorithms goes. So we decided to not focus that much on optimizing our work, since it would not make any difference in our case.

As for the goal of our MapReduce, we are interested in getting to know the key figures and employees in the data set. From the research about the structure of the data set [3] we know the general structure of the data set. With this in mind we try to gather some information about the structure of the data set. One approach is to gather information about the email folders that are assigned to each message in a messy comma separated value form. Another interesting information is the people that are in the data set. Each folder has a owner, so it would be interesting to get a list of people who were involved in most conversations based on the ”content” of the folder. Most likely those people that repeat most often in any given folder, e.g ”this folder has most emails where ”from:” is person X” thus person X is assumed to be the owner.

Word analysis is always an approach of great significance. By counting certain words (bankrupt, fraud, etc.) in conversations it is possible to draw the time line and follow the development and continuation of conflict inside company. The reason behind existence of this data set is obviously bankruptcy and shady practices, so looking for words in this topic could bring some nice looking statistics, and depending on implementation clever use of MapReduce.

As for the email messages themselves, we assume that messages which were never replied to are not significant and most likely contain spam, some general instructions or information about meetings and appointments. They have low information content factor [3]. For research purposes we will retrieve conversations with medium and high information content factors. Saying in other words we are interested in conversations, because they are more likely to contain important and meaningful information.

Data-intensive approach is different from data mining. Therefore we are not interested in data we get out of the set, but more about the way we can retrieve information, knowing something about the structure beforehand. For example, we know who was the CTO of the company, how can we retrieve his emails, and do some analysis on that.

It is hard to work with raw data. We consider cleaning the data set and keeping only relevant to us information on each step of MapReduce.

2 Data set overview

In this section we provide a short overview of data set. Structure of data set and its format will be discussed.

The Enron data set includes about 500 0000 emails from Enron Corporation's employees. It was obtained after Enron Corporation collapsed in 2001. The current version of data set is from 2015 [4].

All data in data set is divided on file and message itself separated by comma. File contains information about the original directory and file name of the email. The root level directory points to the owner of directory [4].

Message part contains such fields as ”Message-ID”, ”Date”, ”From”, ”To”, ”Subject”, ”Cc”, ”Mime-Version”, ”Content-Type”, ”Content-Transfer-Encoding”, ”Bcc”, ”X-From”, ”X-To”, ”X-cc”, ”X-bcc”, ”X-Folder”, ”X-Origin” and ”X-FileName”. Some of these fields are optional, for example, ”To”, ”Cc” and ”Bcc”. The body of email may contain header and signature, header is separared by an empty line. ”X-...” values are from original Enron Corporation email data set. ”From”, ”To”, ”Cc” and ”Bcc” fields were retrieved by SRI researchers from ”X-...” lines [3].
Example:

File:

```
"taylor-m/all_documents/404."
```

Message:

```
"Message-ID: <6201256.1075859887091.JavaMail.evans@thyme>
Date: Fri, 2 Jul 1999 08:47:00 -0700 (PDT)
From: elena.kapralova@enron.com
To: mark.taylor@enron.com
Subject: Examples of European Long Descriptions
Cc: justin.boyd@enron.com
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Bcc: justin.boyd@enron.com
X-From: Elena Kapralova
X-To: Mark - ECT Legal Taylor
X-cc: Justin Boyd
X-bcc:
X-Folder: \Mark_Taylor_Dec_2000\Notes Folders\All documents
X-Origin: Taylor-M
X-FileName: mtaylor.nsf
Mark,

please have a look in the ""Long Descriptions"" worksheet in
     the file attached.
Descriptions are combined from different components defined
    on other pages
per trading desk. Product lines in blue colour font on
    yellow background have
been already reviewed twice, blue on white - once, black on
    white - haven't
been reviewed yet.

Please e-mail/call if you have any questions or would like
    to have a fax as
well.
Regards,
Elena

"
```

Listing 1.1. Example showing structure of the data set that we used as a guide

We decided to work with data set as it is, no modifications were applied. So initial MapReduce takes in the whole data set line by line.

We have no need for additional data sets for our case, we were only focused on what we can get out of what we are given. Data set of 500000 emails mostly from top managers and sales clerks is more than enough to get out some structure of the data set. The emails cover period from November 1998 to June 2002. There are 150 folders in data set which belong to 156 employees. Usually employee owns one folder, but there are cases when one folder belongs to two different employees and one employee owns two folders. Some employees have same first and last name, they can be recognized by middle name [3]. Each file/email will consist of the original email and all the appended emails that this email was replying to, this is a source of repeating information that has to be accounted for where it's relevant.

3 Hadoop setup usage

In this section we will present the setup we used for our work. Pros and cons of using different approaches and the reasons behind our setup.

This project focuses also on the way we implement MapReduce jobs, for this purpose it would've been nice to have a real cluster to be able to compare the running time of different approaches and try to optimize it better. In our case, we have done the majority of our work and testing on local machines in pseudo-distributed mode with the use of *Hortonworks Sandbox with HDP* [1].

This was not the most optimal solution, so in the beginning we tried to use *Amazon Web Services, AWS* [2], but we had trouble getting free usage on their servers so we simply moved back to the local machine solution. For learning purposes there is not much drawback from using the setup on local machines, but as mentioned earlier the CPU power is limited and the core idea distributed system comes down to theory and virtualization. The good thing is there is almost zero work involved in setting it up, so one might set it up and start working immediately without worrying about much.

The benefits of running jobs on a real cluster like AWS, is that it would then be possible to monitor the CPU and disk usage, to better optimize the jobs, or modify the cluster. Identifying bottlenecks would help the performance of out tasks.

For testing purposes on local machines, we used a subset of the whole data set to reduce the waiting time for different algorithms. In multistage MapReduce jobs, because steps have to run one after another, testing it on a big file might waste a lot of time because MapReduce will try to execute as much as possible with assumption that your code is bug free, so usually before the algorithm is bug free we would just run on the subset of data created beforehand.

4 Description of MapReduce algorithms

This section describes all the MapReduce jobs that we have written, and discusses what could be improved, what are the alternative ways of solving the problems that we came up with, and what additions could be made to make the MapReduce job more complicated.

The first clever use of MapReduce in our project was to figure out the number of employees that are in the data set. As described earlier we know the structure of the data set. Every message has its own subfolder, which is the first value in the comma separated values list. The root folder in the path corresponds to the owner of the folder and has a unique name.

The simple and straightforward way is to simply map all the subfolders, extract the root folder that will end on the first occurrence of single '/' and in reducer sum all these occurrences.

```
def mapper_init_count(self):
  self.message_id = ''
  self.in_body = False
  self.body = []
```

```
    self.after_key = False
    self.beginning = False
    self.key = False

  def mapper_count(self, _, line):
    line = line.strip()

    if (line.find('.","Message-ID: <') > 0) and self.in_body
        and not self.beginning:
      yield self.message_id, self.body
      self.message_id = ''
      self.body = []
      self.in_body = False
      self.after_key = False
      self.beginning = False
      self.key = False

    if self.in_body and not self.after_key:
      self.beginning = False
      self.body.append(line)

    if line.find('.","Message-ID: <') > 0 and not self.key:
      if not self.in_body:
        self.in_body = True
        self.beginning = True
        self.after_key = True
        self.key = True
      start = line.find("Message-ID")+13
      i=0
      for char in line[start:]:
        i=i+1
        if (not (char.isdigit() or (char == '.'))):
          stop = i+start-2
          break
      self.message_id = line[start:stop]
    self.after_key = False

# STEP 2
  def mapper_child(self, message_id, values):
    clean_body = ''
    clean_date = ''
    clean_from = ''
    clean_to = ''
    clean_values = []
    start = 0
    for idx, line in enumerate(values):
      if "Date:" in line:
        clean_date = line[5:].strip()
      if line.find("From:") == 0:
        clean_from = line[5:].strip()
      if line.find("To:") == 0:
        clean_to = line[3:].strip()
      if "X-FileName:" in line:
        start = idx+1
        break
    for i in range(start,len(values)):
      if "-Original Message-" in values[i]:
        break
      clean_body=clean_body + values[i] + " "

    clean_values.append(clean_date)
    clean_values.append(clean_from)
    #clean_values.append(clean_to)
    #clean_values.append(clean_body.strip())
    clean_values.append("TEST BODY")
    newval = values
    for element in values:
      if "subject:" in element.lower():
        subject = element
        break
    if "re:" in subject.lower():
      newval.append("child")
    elif "fw:" not in subject.lower():
      newval.append("parent")
    for element in newval:
      if "Subject:" in element:
        subject = element
        break
    relation = values[-1]
    i = 0
    colon = 0
    if "<" not in subject:
      for char in subject:
        i=i+1
        if char == ":":
          colon = i
      sub = subject[colon+1:].strip()
      sub_relation = []
      sub_relation.append(sub)
      sub_relation.append(relation)
      yield sub_relation, (message_id,clean_values)
```
Listing 1.2. First two step MapReduce that sorts and combines data into single row

If on the other hand this structure would not be present, we tried a different approach, that could not only count the number of employees in the data set, but also give much more information. This is done by keeping more information between each map-reduce stage. The end goal of this was not clear in the beginning, but we wanted to keep as much information that could later on be used for different reducers. The things that we worked with were, for example, composite keys, different techniques of summation of values.

Our first step focuses more on cleaning in the data. We know that the raw structure is:

```
"filepath", "file_contents"
```

each file is individual email, to get the data structure of the type

```
"message_id" , "email_body"
```

We would first have to locate where each individual file begins and where it ends. This can be done relatively easily with the use of ”mapper_init” function. This function is run once on each node before each mapper phase. In that function we can store all the information that we are interested in from the data set. e.g mapping multiple lines of the file to an array of values.

At this stage the data structure does not provide us with more information, but we are now able to easily get the email body, because each key - value pair from this mapper contains the whole file, we will call the value of this tuple a file because it contains all the information of a single file from the folder in the data set.

One thing worth noticing at this stage, is that the data set might be unfortunately split across all the nodes, e.g. in the middle of the email, thus the mapper might not locate where the file ends (because it is located on another node) and we will miss some of the information. But the data loss if very small compared to the whole data set, we might miss one email in hundreds of emails, this is negligible in our case. see Listing 1.3

```
  def mapper_init_count(self):
    self.message_id = ''
    self.in_body = False
    self.body = []
    self.after_key = False
    self.beginning = False
    self.key = False

  def mapper_count(self, _, line):
    line = line.strip()

    if (line.find('.","Message-ID: <') > 0) and self.in_body
         and not self.beginning:
      yield self.message_id, self.body
      self.message_id = ''
      self.body = []
      self.in_body = False
      self.after_key = False
      self.beginning = False
      self.key = False

    if self.in_body and not self.after_key:
      self.beginning = False
      self.body.append(line)

    if line.find('.","Message-ID: <') > 0 and not self.key:
      if not self.in_body:
        self.in_body = True
        self.beginning = True
        self.after_key = True
        self.key = True
      start = line.find("Message-ID")+13
      i=0
      for char in line[start:]:
        i=i+1
        if (not (char.isdigit() or (char == '.'))):
          stop = i+start-2
          break
      self.message_id = line[start:stop]
    self.after_key = False
```
Listing 1.3. First mapper that combines relevant lines into one big value

The reducer in this case is not needed since we are getting unique rows anyway, all the keys from mapper are unique.

4.1 Counting threads of messages

Now that we have somehow modified the data, we can easier map the information we are looking for and reduce it further. At this stage the value we get has such data structure that we can easily access the ”Subject” of the email that was sent, who it was sent to and who sent the message. Mapping by subject as a key will allow us to reduce each row to single subject + whatever information about the particular sender or recipient we want. see Listing 1.4.

```python
def mapper(self, key, value)
  [....]
  for element in values:
    if "subject:" in element.lower():
      subject = element
      break
  if "re:" in subject.lower():
    newval.append("child")
  elif "fw:" not in subject.lower():
    newval.append("parent")
  for element in newval:
    if "Subject:" in element:
      subject = element
      break
  relation = values[-1]
  i = 0
  colon = 0
  if "<" not in subject:
    for char in subject:
      i=i+1
      if char == ":":
        colon = i
    sub = subject[colon+1:].strip()
    yield (sub, relation), (message_id,clean_values)
```

Listing 1.4. mapper that creates composite key in order to preserve the creator of the topic and conversation partner(s)

```python
def reducer_child(self, key, values):
  lista = []

  for mid in values:
    lista.append(mid)

  from_who = []
  for val in values:
    if not val[1][1] in from_who:
      from_who.append(val[1][1])
  num = len(from_who)
  lista.append(num)
```

```
    yield key[0], (key[1], lista)
```
Listing 1.5. reducer that will reduce *subject relation* composite key and some array of values

By reducing the composite key to just the "Subject" element, we will map both children and parent to the same key. Listing 1.5.

To count the number of threads in the data set, we can simply count the number of subject, by yielding "subject", 1, in the previous mapper, then reduce it by summing all the values in reducer by key like so: see Listing 1.6

```
def mapper_count(self, key, value):
  if "<" not in subject:
    for char in subject:
      i=i+1
      if char == ":":
        colon = i
    sub = subject[colon+1:].strip()
    yield sub, 1

def combiner_count(self, key, values):
  yield key, sum(values)

def reducer_count(self, key, values):
  yield key, sum(values)

def mapper_tot(self, key, values):
  yield 'Number of threads:::::', 1

def reducer_tot(self, key, values):
  yield key, sum(values)
```
Listing 1.6. reducer that will reduce *subject relation* composite key and some array of values

but this is even possible without doing all that previous work with collection the body of the email to a single row. What the first step enables us to do, is to, for example, count the number of replies there has been in a single conversation/subject.

Since the contents of the file/email are concentrated in a single row, we can now distinguish the original message from whatever has been appended

to it when the recipient pressed ”reply - send”, whatever was appended to the end thats not part of the ”current” message is always separated with the line ”- - - -OriginalMessage - - - - -”
with this we can use the technique from the first mapper at Step 1, to combine just the relevant body of the email into a new value. Each of those will then have one single ”Subject” line, single ”From:” line and single ”To:” line. By examining the size of the array of each ”Subject” row, we are given the number of individual emails sent in that conversation. see Listing 1.7

```
def reducer_child(self, key, values):
  lista = []

  from_who = []
  for val in values:
    if not val[1][1] in from_who:
      from_who.append(val[1][1])
  num = len(from_who)
  lista.append(num)
      yield key[0], num
```

Listing 1.7. topics with the most participants

Each subject has a root message, the message that started the conversation. We thought it would also be interesting to look at the statistics of who initialized the most conversations.

```

def reducer_child(self, key, values):
  relation = key[1]
  from_mail = values[1][1]

  if relation.find("parent") > -1:
    yield from_mail, relation

def mapper_child_2(self, mail, relation):
  yield mail, 1
def reducer_child_2(self, mail, values):
  yield mail, sum(values)
```

Listing 1.8. This reducer is using mapper from step 1

The key in this case consists of subject and relation of particular value to this subject. If we detect that the relation is parent, we yield the mail as the key, and relation as value, this is done for each key, that is original subject.

Listing 1.5, also provides possibility to count the number of unique emails in the conversation based on the ”From:” value of each original email. If the value is 2, it means that only two people were conversing, if the number is bigger, it means that there were more active participants in the subject.

5 Description of Hive usage

Apache Hive is a data warehouse which works directly with HDFS and Apache HBase, and allows to store data of different types. Hive data management system uses SQL style for queries which makes it easy to use for people already familiar with this syntax. More information about Hive and its features can be found in source [7].

This project is mainly about Hadoop MapReduce, therefore we didn't spend much time on Hive. We performed simple queries to compare it with MapReduce. It was also difficult to compare any run times since we mainly tested on local machines. Advantages of Hive include easiness of usage and storage. The biggest disadvantage of using Hive is that the input data should be well structured otherwise the queries are very complicated. So in our experience, it is still best to at least prepare the data in standard MapReduce and then use that data to perform fast queries with different values, or on different columns/rows. We had to preprocess email data set to next format, with the data that we were interested in for particular query: see Listing 1.9

```
"14387418.1075862107058"^ "Mon 26 Nov 2001 22:04:36"^ "
    mcausholli@hotmail.com"^ "Happy Thanksgiving"
"16098759.1075862107080"^ "Tue 27 Nov 2001 18:42:53 -0800 (
    PST)"^ "announcements.enron@enron.com"^ "Enron/Dynegy
    Merger; Antitrust Issues"
"22468189.1075862107107"^ "Tue 27 Nov 2001 12:13:26 -0800 (
    PST)"^ "ayesha.kanji@enron.com"^ "newsprint update,
    11/27"
```

Listing 1.9. What data looks like prepared for hive

This data is a comma separated list with message ID, date, from:email, subject. First we create a table where we will store all the data:

```
CREATE EXTERNAL TABLE enron(id STRING, dates STRING,
    fromaddr STRING, subject STRING) ROW FORMAT DELIMITED
    FIELDS TERMINATED BY "^" LINES TERMINATED BY "\n"
    STORED AS TEXTFILE;
```

Listing 1.10. Hive query for creating table

Let us check the table we created:

```
hive> show tables;
OK
enron
```

Next step is to append data from file to table:

```
LOAD DATA INPATH "/test/hive_data.csv" OVERWRITE INTO TABLE
    enron;
```

Now when we have table with all clean values, it is quite easy to perform simple queries on this data set, for example:

```
hive> select subject from enron limit 5;
OK
 "Happy Thanksgiving"
 "Enron/Dynegy Merger; Antitrust Issues"
 "newsprint update, 11/27"
 "PPW THIS WEEK!"
 "Your enrollment status has been modified."
```

We will find out how many unique subjects are in Enron email data set:

```
Status: Running (Executing on YARN cluster with App id
    application_1478284412550_0080)

        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING
            PENDING  FAILED  KILLED
```

```
Map 1 ..........   SUCCEEDED      4          4         0
           0        0        0
Reducer 2 ......   SUCCEEDED      1          1         0
           0        0        0
Reducer 3 ......   SUCCEEDED      1          1         0
           0        0        0
VERTICES: 03/03  [==========================>>] 100%
    ELAPSED TIME: 32.69 s
OK
125005
Time taken: 44.296 seconds, Fetched: 1 row(s)
```

Listing 1.11. Hive output showing number of threads

The number we get is 125005 unique subjects, this is very trivial task when data is prepared.

Let us count unique email addresses from "From:" field and group them by subject:

```
hive> select subject, count(distinct fromaddr) AS theCount
    from enron group by subject order by theCount DESC limit
     20
     > ;
Query ID = root_20161105181929_6a97b786-b815-48fd-9e48-0
    fed5e59302f
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id
    application_1478284412550_0080)

        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING
            PENDING  FAILED  KILLED
Map 1 ..........   SUCCEEDED      4          4         0
           0        0        0
Reducer 2 ......   SUCCEEDED      1          1         0
           0        0        0
Reducer 3 ......   SUCCEEDED      1          1
0        0        0        0

VERTICES: 03/03  [==========================>>] 100%
    ELAPSED TIME: 32.94 s

OK
 "Demand Ken Lay Donate Proceeds from Enron Stock Sales"
     1116
 "(no subject)" 132
 "Hi" 112
 "Meeting"  106
 "Hello"  99
 "0"  98
 "Lunch"  91
 "FYI"  87
 "Thank You"  78
 "test" 72
 "Congratulations"  72
 "Confidentiality Agreement"  63
 "Enron"  61
 "Hey"  58
 "Test" 56
 "Resume" 56
 "Question" 55
 "Update" 53
 "Organizational Announcement"  52
Time taken: 36.871 seconds, Fetched: 20 row(s)
```
Listing 1.12. This reducer is using mapper from step 1

Hive works very fast and it is relatively easy to get results without having programming skills, but it is hard to get processed data presented in nice way in order for hive to be easy to use.

6 Summary of key findings

MRJob results of finding the most interesting topics, based on the number of unique *senders* in the same topic, that is active participants in the same topic. see Listing 1.13

```
"(no subject)"  77
"0" 68
"Demand Ken Lay Donate Proceeds from Enron Stock Sales" 1116
"FYI" 69
"Hello" 79
"Hi"  56
"Hi"  54
"Lunch" 56
"Meeting" 52
"Meeting" 57
"Thank You" 57
```
Listing 1.13. Output from similar task performed in Hive in Listing 1.12

As we can see, there is one topic that stands out, quick Google search reviled some information about it, and the numbers from sources [8] are quite similar. But to further investigate the contents of this Subject we would have to go deeper into data mining. Compared to the same job run in hive from Listing 1.12 also gave similar results.

As mentioned earlier, the topics that got high scores are pretty generic, and most probably do not belong to the same thread, at the same time there is very low chance that more complex topic would be repeated multiple times. Having said that, it would be nice to expand the composite key, by giving it another value like a date the subject was updated. This could give us opportunity to reduce output to a range of dates, subjects exceeding this range would be treated as a separate key. For example in Listing 1.5 , before we split the composite key, we should add a date checker that compared existing date in *values[1][0]*.

MRJob result of finding usage of work "fraud" by date, extracted from unique emails.

```
["2000", 10]   11
["2000", 11]   85
["2000", 12]   8
["2000", 3]    10
["2000", 5]    4
["2000", 6]    3
["2000", 7]    9
["2000", 8]    2
["2000", 9]    11
["2001", 10]   102
["2001", 11]   44
["2001", 12]   5
["2001", 1]    10
["2001", 2]    3
["2001", 3]    13
["2001", 4]    10
["2001", 5]    8
["2001", 6]    18
["2001", 8]    1
["2001", 9]    3
["2002", 1]    4
```

```
["2002", 3]    3
```

Listing 1.14. Output of counting all frequency of usage of word ”fraud” in distinct emails by date

As we can see the phrase ”fraud” was most used around the end of 2001, this corresponds pretty well with the time around which the company went bankrupt.

7 Conclusion

In this project we proccessed large amount of data using different MapReduce algorithms, located, sorted, combined, and displayed different combinations of data and gathered statistics. We also used Hive to some extent to try and get similar results. We improved our knowledge greatly in building multistep MapReducers, experimented and combined different data types with combination of key value structure.

We experienced that it is hard to perform extensive tests and play with MapReduce jobs on our local machines, in future work we would like to move the whole work environment to some cloud service like AWS [2], that we tried, but because of lack of time we had to prioritize algorithm development more instead of setting up free clusters and trying to get free access. Nevertheless we managed to get a feeling and compare the running time of complex MapReducers to some extent.

8 Future improvements

Even though we managed to reach out goals when it comes to the results, while working with MapReduce we figured out the potential it has and its applications. Unfortenalty there was not enough time to explore every detail of it. One of the main things that we would like to improve is reconstructing the conversation tree in emails. As of now, we are able to count the size of the tree, how many people were involved in certain topics, what would be interesting to find, is all the different connections between users. This could be achieved by looking at the ”From:” and ”To:” positions in the value array from Listing 1.2 together with the ”Subject:” line but also keeping track of the ”Re:” and ”FW:” in the ”Subject:”. As it is now, all the children are stored in one array, if we assume that every time an email is forwarded - ”FW:” exists in ”Subject:” new branch is being created, and nested array is added to the existing one, and then repeating the process for every new ”Subject: FW:”, ”from_email” pair. It would also be possible to ignore the emails that have ”from_email” as one of the existing ”parents”.

As presented in Listing 1.5 and the description, we found out that there are not that many *interesting* Subject titles that have more than 3 participants. It seems like most of the conversations were of type one to many, or one to one, with occasional responses to the root message. We started with that simple statistics just to see what we could expect as the result if we invested time into reconstruction of the graphs.

References

1. Hortonworks hadoop Available: http://hortonworks.com/hadoop-tutorial/hortonworks-sandbox-guide/#section_1 [5 November 2016]
2. Amazon Web Services Available: https://aws.amazon.com/ [Accesed 5 November 2016]
3. Mining organizational emails for social networks with application to Enron corpus by Zhou, Yingjie, Ph.D., RENSSELAER POLYTECHNIC INSTITUTE, 2008, 179 pages.
4. The Enron Email data set. Kaggle. [Online] Available: https://www.kaggle.com/wcukierski/enron-email-dataset. – Name from screen. [Accessed 5 November 2016]
5. Apache Hadoop. From Wikipedia, the free encyclopedia. [Online] Available: https://en.wikipedia.org/wiki/Apache_Hadoop. – Name from screen. [Accessed 5 November 2016]
6. Welcome to Apache™ Hadoop®. Apache hadoop. [Online] Available: http://hadoop.apache.org. – Name from screen. [Accessed 5 November 2016]
7. Apache Hive. From Wikipedia, the free encyclopedia. [Online] Available: https://en.wikipedia.org/wiki/Apache_Hive. – Name from screen. [Accessed 5 November 2016]
8. Machine Learning: Enron Corpus Available: http://www.enron-mail.com/email/lay-k/deleted_items/Demand_Ken_Lay_Donate_Proceeds_from_Enron_Stock_Sa_661.html [Accessed 5 November 2016]