Lab Assignment 2 **Big Data**
Due Date: 15/01/2019 Prof. P. Boncz
Jeroen Schmidt (122808560) & Dimitris Michailidis (12325929) TA: Dean De Leo
Group 10

Part 1 - Running Map Reduce

In this section we report on the results of running the simple and extended map reduce scripts on the small and the large Enron dataset.

Running the Example Script

At first, we ran the provided `enron-etl.py` script to download, extract and convert the email archives into tuples structured as (date, from, to). It then saves the results on the HDFS of the cluster. We create a streaming job and set the following parameters:

- Mapper: the enron-etl script that extracts data from the email archives.
- Reducer: simple cat command that redirects the output to the HDFS.
- mapred.input.dir: the s3 bucket that contains the email archives.
- mapred.output.dir: the folder in the cluster that will be used as an input for the extended map reduce scripts.

The result of the script is a folder called `enron-output-1` that contains the tuples in seven different partitions. This folder is used as an input in the extended map reduce script we describe in the following subsection.

Extending the Mapper and Reducer Scripts

The goal is to find the top-20 Enron email accounts that communicated with people outside of Enron at least once in the time between 05-11-2001 and 08-11-2001. To achieve this we modify the map script as follows:

```
#!/usr/bin/env python

from datetime import datetime

import sys

def get_email_domain(email):
        return email.split('@')[1]

for line in sys.stdin:
        data = line.split('\t')
        # Exclude emails not between the given date
        date = datetime.strptime(data[0], '%Y-%m-%dT%H:%M:%S%fZ')
        keep_date =
            datetime(2001, 11, 5).date() <= date.date() <= datetime(2001, 11, 8).date()

        # Get the domain of the email addresses of the sender and the recipient
        sndr = get_email_domain(data[1])
        rcpt = get_email_domain(data[2])
```

```
        # Exclude emails from enron not sent outside of enron
        keep_sndr = 'enron' in sndr
        keep_rcpt = 'enron' not in rcpt

        if keep_date and keep_sndr and keep_rcpt:
                # Emit the sender to the reducer.
                print data[1].strip()
```

The main change here is that we filter out accounts based on three conditions: i) date, ii) sender, iii) recipient. At first, we convert the string date into a python datetime object, to ensure correct comparisons. We then create the boolean flag `keep_date` that is set to true only if the email's date is between the given one. We then create two additional boolean flags, `keep_sndr` and `keep_rcpt` that aim to keep only emails sent from Enron accounts to non-Enron accounts. We first split the email addresses by '@' so that we only get the domain(we want to exclude addresses like enron@easymatch.com). We then filter the domains by 'enron' because there are addresses that contain different extensions(e.g .com, .net, .co.uk) and even some that do not have an extension at all(e.g generalannouncement@enron). If all three conditions are met we emit the sender to the reducer(we only care about the sender in this exercise).

We also modify the reducer as follows:

```
#!/usr/bin/env python
import sys
current_count = 0
current_word = ""
for line in sys.stdin:
        line = line.strip().split('\t')
        if len(line) != 1:
                continue

        key = line[0]
        if (key != current_word):
                if (current_count > 1):
                        print current_word + '\t' + str(current_count)
                current_count = 0
                current_word = key
        current_count += 1

if (current_count > 1):
        print current_word + '\t' + str(current_count)
```

The main change here is that we print the `current_word` only if the `current_count` is greater than one, since we only care about accounts that communicated with outsiders *more than once*. We also move the `current_count` reset outside of the second if, because we want it to reset regardless of it getting printed in the output. In the following subsections we describe the results of running the updated scripts on the small and the large dataset.

Question - Executing on the Small Dataset

To run the scripts on the small dataset, we create a new streaming job with the following parameters.

- Mapper: `enron-map.py`
- Reducer: `enron-reduce.py`

- mapred.input.dir: /user/group10/enron-output-1
- mapred.output.dir: /user/group10/enron-output-2
- Archives: `enron-map.py` and `enron-reduce.py`

Running this job gives us an empty output, because none of the emails in the small dataset satisfy all of the conditions we set in the updated script. There is no Enron account that sent more than one emails to non-Enron accounts between 05-11-2001 and 08-11-2001 in the small dataset.

Question - Executing on the Large Dataset

To run the scripts on the large dataset, we modify the `enron-etl` job to receive input from the large dataset instead of the small (mapred.input.dir: s3n://enron-scripts/enron-urls.txt). We store the output on the same folder as before, and then we run the same streaming job as in the small dataset. In the output folder we now get seven partitions with tuples in the form of (email_account, total_count).

In order to sort these partitions and save them to s3 we create a new streaming job with the following parameters:

- Mapper: cat
- Reducer: sort -k2nr
- mapred.reduce.tasks: 1
- mapred.input.dir: /user/group10/enron-output-2
- mapred.output.dir: s3n://bd-group10/emails-sorted

Now the mapper is a `cat` command that emits everything to the reducer, which is also a bash command `-k2nr` which sorts the input by the second parameter in a descending order. Note that we set the reducer to be only 1, so that we get the output in one partition. The top-20 accounts are therefore presented in Figure 1.

```
amy.felling@enron.com   256
mary.cook@enron.com     83
marie.heard@enron.com   73
darran.binns@enron.com  26
kathy.dodgen@enron.com  18
michael.green@enron.com 18
alan.aronowitz@enron.com        15
julissa.marron@enron.com        14
stacey.richardson@enron.com     10
jackson.logan@enron.com 9
kim.ward@enron.com      9
fraisy.george@enron.com 8
sladana.kulic@enron.com 8
susan.j.mara@enron.com  8
david.portz@enron.com   7
patricia.villalobos@enron.com   6
phillip.m.love@enron.com        6
lara.leibman@enron.com  5
grace.rodriguez@enron.com       4
jad.doan@enron.com      4
```

Figure 1: Top 20 Enron email accounts that sent more than one email to non Enron accounts and the amount of emails they sent.

Part 2 - Answering Questions

In this section we answer theoretical questions about Map Reduce and Streaming Jobs, as well as explain the details of the `enron-etl.py` script.

Question - How does Map Reduce Work

MapReduce tasks are distributed across a cluster. MapReduce consists of 3 stages, Map, Shuffle and Reduce. They are explained in more details in the following three points;

1. The map stage receives data in the form of key value pairs which are read from partitions. The map stage performs a one to one functional map on the data, e.g. something of interest is calculated/extracted from the data.

2. The shuffle stage occurs on at the output of the map and input of the reduce stages. The data outputted by the mapper processes gos into a circular round robin (memory) buffer of a predefined size, it then spills to disk once that buffer has been exceeded, the data is sorted at the end of the mapper write phase. The partitions are then copied and merged according to the key for the reducer to work on.

3. The reduce stage reads in the data that was outputted by the mapper processes and it can start running concurrently to the map phase, as long as some merged partitions from map processes are ready. The role of the reducer phase is often to perform a many to one function map across the partitions.

Each stage of the mapreduce program spawns multiple map and reduce tasks that are independent from one another and run in parallel, which is one of the reasons why MapReduce is fault tolerant. The mapreduce process are also run on the machine where the data reside. This is done inorder to prevent the data from having to be transferred onto independent machines (in the traditional case). The cluster consists of a master and worker nodes that span across a shared file system (S3, HDFS). Resources are allocated and containers are create for each map and reduce processes which are run on each of the worker nodes when a mapreduce job is submitted to the master.

The base case is illustrated in figure 2 but it can be extended such that you have several mapreduce operations chained one after the other.

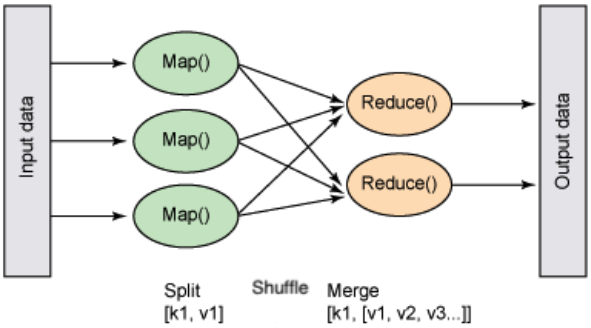

Figure 2: MapReduce process [1]

Question - Advantages of Streaming Jobs

Hadoop is a framework written in Java and runs map reduce using .jar executables[1]. However, it also exposes a streaming API that allows mappers and reducers written in different languages to be configured and executed as part of a streaming process. This is enabled by the fact that streaming jobs distribute the data input to the cluster's machines using standard input (`stdin`) and the output is written using standard output (`stdout`)[2]. Therefore, any language that has commands to interact with `stdin` and `stdout` can be used to write map and reduce scripts, without needed to compile .jar executives.

Question - Explaining the URL Download Script

The `enron-etl.py` script gets as input a .txt file(from the s3 bucket `enron-scripts`) with urls corresponding to the following files:

- edrm-enron-v2_south-s_xml.zip
- edrm-enron-v2_pereira-s_xml.zip
- edrm-enron-v2_rapp-b_xml.zip

In order to create the tuple files and enable the analysis, the script gets these urls as input and uses the `urrlib` to download the zip files, saving them to the temp folder. It then uses the `zipfile` library to extract their contents and create the tuples.

References

[1] Saliha Keskin and Atilla Erguzen. Hadoop cluster analysis and assessment. *International Journal on Pavement Engineering Asphalt Technology*, 06 2018.

[1] https://hadoop.apache.org/docs/r1.2.1/streaming.html
[2] http://manishbarnwal.com/blog/2016/08/29/hadoop_streaming/