MapReduce using mrjob

This is to be run in the Cloudera Quickstart VM. Make sure you have started Cloudera express and the services (by clicking Launch Cloudera Express, or writing sudo /home/cloudera/cloudera-manager --force --express in a terminal) waiting for the script to finish, and then logging in into the Cloudera manager using the web browser (username and password are both cloudera).

A. The ADULTS dataset from the UCI repository

(a) Read about the dataset (particularly the attribute information) in:
https://archive.ics.uci.edu/ml/datasets/Adult

(b) Download the file adult.data from:

https://archive.ics.uci.edu/ml/machine-learning-databases/adult/

This needs to click on “Data folder” (under the title Adult Data Set and just above the main table of the page) and then viewing the page titled Index of /ml/machine-learning-databases/adult and downloading the file adult.data (with right click and save link as).

By default, the file adult.data will be in the folder ~/Downloads/.

(c) Create a dataset, *adult_3.data*, with the first 3 lines of adult.data, by executing:

```
head -3 adult.data > adult_3.data
```

from the directory that contains adult.data This is useful for testing your programs. The parameter “3” in head denotes the number of lines that we want to include in the output file and the name after > denotes the output file.

(d) Create a dataset, adult_10.data, with the first 10 lines of adult.data

B. MapReduce in Python with mrjob : Getting started

(a) Cloudera contains python version 2.6. To be able to use mrjob, we need to install python 3. We will choose the version python 3.5.0.

To install python 3.5.0, type the following commands one after another:

```
sudo yum install yum-utils
sudo curl -O https://www.python.org/ftp/python/3.5.0/Python-3.5.0.tgz

sudo tar xf Python-3.5.0.tgz
cd Python-3.5.0
sudo ./configure
sudo make
sudo make install
```

(b) We now install some programs needed for installing mrjob. This is done by the following commands, which you should execute one after another.

```
sudo yum install python34-setuptools
sudo python3 -m pip install --upgrade pip
```

(c) Last, we install mrjob by typing.

```
sudo python3 -m pip install mrjob
```

Main website of mrjob:

https://pythonhosted.org/mrjob/index.html

Reference manual for mrjob (in pdf):

https://media.readthedocs.org/pdf/mrjob/latest/mrjob.pdf

(d) Create a file *wordcount.py* with the following code. Be careful to ident the code correctly:

```
from mrjob.job import MRJob

import re

WORD_RE = re.compile(r"[\w']+")

class wordcount(MRJob):

    def mapper(self, _, line):

        for word in WORD_RE.findall(line):

            yield (word.lower(), 1)

    def reducer(self, word, counts):

        yield (word, sum(counts))

if __name__ == '__main__':

     wordcount.run()
```

(g) Execute *wordcount.py* **in the local mode** on adult3_data with

python3 wordcount.py adult_3.data

(h) What is the output? Observe carefully the functions for the mapper and reducer in the code and the output in the file, to understand how the output is generated.

(i) Execute *wordcount.py* **in your local hadoop cluster** (that runs inside Cloudera) with

python3 wordcount.py adult_3.data -r hadoop

Note: All services need to be running (green buttons in the Cloudera Manager menu - see lab 2). If they are not running, you will get error messages and the task will not complete. For example, if HDFS is not running, the last line of the error message is

OSError: Could not mkdir hdfs:///user/cloudera/tmp/mrjob/wordcount.cloudera….

This is because our mrjob program needs to create a directory in HDFS, which will store the results, and if the HDFS service is not running, then this is not possible.

A correct output is as follows:

```
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Found hadoop binary: /usr/bin/hadoop
…
Running step 1 of 1
  …
  number of splits:2
  Submitted tokens for job: …
  …
  The url to track the job: …
  Running job: …
  Output directory: hdfs:///user/cloudera/tmp/mrjob/wordcount.cloudera. ….
…
  Removing HDFS temp directory …
  Removing temp directory /tmp/wordcount.cloudera …
```

Why does it take longer? See the File System Counters, Job Counters, and Map-Reduce Framework statistics in the output on the screen. For detailed explanation of each counter see https://www.mapr.com/blog/managing-monitoring-and-testing-mapreduce-jobs-how-work-counters

(j) A combiner takes a key and a subset of the values for that key as input and returns zero or more (key, value) pairs. Combiners are optimizations that run immediately after each mapper and can be used to decrease total data transfer. Combiners should be idempotent (produce the same output if run multiple times in the job pipeline).

(1) Add a combiner to wordcount.py that sums the count of each word and save the file as wordcount2.py

(2) Execute wordcount2.py (in the local or hadoop mode) on adult3_data.

You can use redirection (i.e., add > output_file) where output_file is the name of a file that will contain the output of executing the program that is displayed on the screen if you don't use redirection.

(3) Is the output different from the output of wordcount.py?

(4) Does executing wordcount2.py take longer? Why?

C. Write your own Map Reduce programs using mrjob

Unless stated otherwise, you can execute each of the programs you will write in the local mode, which is more efficient.

(a) Write a MapReduce program, called count_age.py, using mrjob that finds the frequency (i.e., number of occurrences, or count) of each value in age (the first attribute) of adult.data. Think what a mapper should output as (key, value) pair and what a reducer should do with a pair (key, list of values) to get the count.
HINT: The mapper and reducer are similar to that of wordcount.py. However, the mapper yields a pair (a,1) where a is a value of age.

(b) Execute *count_age.py* on your local Hadoop cluster. How many mappers and reducers are used?

(c) Execute *count_age.py* on your local Hadoop cluster with 1 mapper and 1 reducer. This is done by

```
python count_age.py adult.data -r hadoop --jobconf mapreduce.job.maps=1
 --jobconf mapreduce.job.reduces=1
```

How much time it takes? How does this compare to the time it took for the program in (b)?

(d) The following MapReduce program computes the maximum value of age. For example, when applied to adult.data, it outputs "sum:" 1256257. If you are not familiar with generator expressions (the first line in the reducer) see https://realpython.com/blog/python/introduction-to-python-generators/

```
from mrjob.job import MRJob

class MRmyjob(MRJob):
    def mapper(self, _, line):
        data=line.split(", ")
        if len(data)>=2:
            age=data[0].strip()
            yield None, age

    def reducer(self, _, list_of_values):
        a=[int(i) for i in list_of_values]
        sum_a=sum(a)
        yield "sum:", sum_a

if __name__== "__main__":
       MRmyjob.run()
```

(1) Modify the code in (d) to compute the average age. Save the result in a new file and execute it on adult.data.

(2) Modify the code in (d) to compute the maximum age. Save the result in a new file and execute it on adult.data

(3) Modify the code in (d) to compute the average age per work-class value. Save the result as *avg_age_per_wc.py* and execute it on adult.data. Assume that the symbol **?** in work-class is a valid value, for simplicity.

Example run for adult_10.data:

python avg_age_per_wc.py adult_10.data

Output for adult_10.data:

```
"Private"           39.714285714285715
"Self-emp-not-inc"           51.0
"State-gov"         39.0
```

(e) Write a MapReduce program, called *max_age_per_wc.py*, using mrjob that finds the maximum age and its frequency (i.e., number of occurrences, or count), per work-class value. Some of the code has been written below

HINT: This program needs to be written as a multi-step job. Read pages 15-16 from https://pythonhosted.org/mrjob/guides/writing-mrjobs.html#writing-multi-step-jobs
for an example.

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRmyjob(MRJob):
    def mapper(self, _, line):
        # complete the mapper

    def reducer1(self, key, list_of_values):
        # complete reducer1

    def reducer2(self, key, list_of_values):
        # complete reducer 2

    def steps(self):
      return [MRStep(mapper=self.mapper, reducer=self.reducer1),
              MRStep(reducer=self.reducer2)]
if __name__== "__main__":
       MRmyjob.run()
```