Big Data Algorithms, Techniques and Platforms
CentraleSupelec - Fall 2018 - Assignment 1

Ayush Kumar Rai – ayush.rai2512@student-cs.fr

December 4, 2018

By turning in this assignment, I declare that all of this is my own work.

I would also like to mention that the code that I have worked on has been successfully tested on **Ubuntu 18.04** and **Hadoop 3.0.3**. Therefore some commands that I am using might be slightly different than students using Hadoop 2.6.

Inside the zip file that I am submitting there would be 4 folders i.e one corresponding to the four problems of the assignment. Each of these folder has bash script **run_script.sh** to run the entire pipeline.

I. The Join1 Problem

Solution:

The code for mapper function is (already provided in the Assignment)

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


for line in sys.stdin:
    line       = line.strip()   #strip out carriage return
    key_value  = line.split(",") #split line, into key
    #and value, returns a list
    key_in     = key_value[0].split(" ")   #key is first item in list
```

```
    value_in    = key_value[1]    #value is 2nd item

    #print key_in
    if len(key_in)>=2: #if this entry has <date word> in key
        date = key_in[0] #now get date from key field
        word = key_in[1]
        value_out = date+" "+value_in #concatenate date
        #, blank, and value_in
        print( '%s\t%s' % (word, value_out) )  #print a string, tab, and string
    else:   #key is only <word> so just pass it through
        print( '%s\t%s' % (key_in[0], value_in) )  #print a string tab and string
```

The output of the mapper and sort module is :
able 991
able Apr-04 13
able Dec-15 100
about 11
about Feb-02 3
about Mar-03 8
actor 22Jan-01 able
actor Feb-22 3
burger 15
burger Feb-23 5
burger Mar-08 2

The reducer code is (already provided in the assignment):

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

prev_word = " "
months = ['Jan','Feb','Mar','Apr','Jun','Jul','Aug','Sep','Nov','Dec']
dates_to_output = [] #an empty list to hold dates for a given word

day_cnts_to_output = [] #an empty list of day counts for a given word

line_cnt = 0

#count input lines
for line in sys.stdin:
        line = line.strip()
        key_value = line.split('\t')
        line_cnt = line_cnt+1
```

```
        curr_word = key_value[0]
        value_in = key_value[1]

        if curr_word != prev_word:
        # -----------------------
        #now write out the join result, but not for the first line input
        # -----------------------
                if line_cnt>1:
                        for i in range(len(dates_to_output)): #loop thru dates, indexes
                                print('{0} {1} {2} {3}'.format(dates_to_output[i],prev_w
                #now reset lists
                        dates_to_output=[]
                        day_cnts_to_output=[]

                prev_word=curr_word #set up previous word for the next set of input lin


        if (value_in[0:3] in months):
                date_day =value_in.split() #split the value field into a date and day-c
                #add date to lists of the value fields we are building
                dates_to_output.append(date_day[0])
                day_cnts_to_output.append(date_day[1])
        else:
                curr_word_total_cnt = value_in #if the value field was just the total c

for i in range(len(dates_to_output)): #loop thru dates, indexes start at 0
        print('{0} {1} {2} {3}'.format(dates_to_output[i],prev_word,day_cnts_to_output[i
```

hadoop fs -cat part-00000 (**Output of the reducer function**)

Dec-15 able 100 991
Apr-04 able 13 991
Jan-01 able 5 991
Mar-03 about 8 11
Feb-02 about 3 11
Feb-22 actor 3 22
Mar-08 burger 2 15
Feb-23 burger 5 15

All the commands to run the code are saved in run_script.sh which is in the same folder as mapper and reducer function

bash run_scrip.sh

Comments

- This problem belongs to **Structural Map Reduce Design Pattern.**

II. The Join2 Problem

Solution:

The Code for Join2 mapper is:

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

# Mapper Code
import sys

for line in sys.stdin:
    print line
    line = line.strip()   #strip out carriage return
    key_value = line.split(",")   #split line, into key and value, returns a list
    key_in = key_value[0]#.split(" ")   #key is first item in list
    value_in = key_value[1]   #value is 2nd item
    test_Num=[int(x) for x in value_in.split() if x.isdigit()] #Loop over value_
    #in elements #and check if they are digits

    if len(test_Num)>0: #Checking if the size of list
    # test_Num is greater than zero
        print( '%s\t%s' % (key_in, value_in))
    else:
        if value_in == 'ABC': #Checking if the value_in is equal to ABC
            print( '%s\t%s' % ( value_in, key_in))
```

The sample output from mapper and sort function is:

Surreal_Talking,981
Surreal_Talking 981
Surreal_Talking,993
Surreal_Talking 993
Surreal_Talking,994
Surreal_Talking 994
Surreal_Talking,995
Surreal_Talking 995
Surreal_Talking,998
Surreal_Talking 998
Surreal_Talking,999
Surreal_Talking 999
Surreal_Talking,BAT
Surreal_Talking,BAT

Surreal_Talking,BAT
Surreal_Talking,BOB
Surreal_Talking,CAB
Surreal_Talking,CAB

Here I am only displaying certain parts of the output because the overall output is very large.

The code for Join2 reducer is:

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

#Reducer Code
import sys

#Create an empty ABC dictionaryionary
ABC_dictionary={}

#Create an empty list for key values
key_value_list=[]

#For loop to read lines from console
for line in sys.stdin:
        line = line.strip()       #strip out whitespaces
        key_value = line.split('\t')   #split line, into
        # key and value, returns a list
        key_value_list.append(key_value) #Append the
        #key_value to the key_value_list list

        if key_value[0]=="ABC": #Checking if the zeroth index
        # of the key_value is string ABC
                if ABC_dictionary.has_key(key_value[1])==False:  # Checking if ABC_dict
                # has the same key in the first index of key_value
                        ABC_dictionary.update({key_value[1]:0}) #updating the value
                        # of a key in ABC_dictionary

for key_value in key_value_list: #Looping over key_value_list
        if ABC_dictionary.has_key(key_value[0]): #Checking if
        #ABC_dictionary has a key stored on zero index of key_value
                ABC_dictionary[key_value[0]]+=int(key_value[1]) #Incrementing the value
                # corresponding to a key in ABC Dictionary
#Prnting the value of the ABC_dictionary
for key, value in ABC_dictionary.iteritems():
```

```
        print( '%s %s' % (key, value))
```

hadoop fs -cat part-00000 (**Output of the reducer function**)

Dumb_Talking 103894
Almost_News 46592
Surreal_News 50420
Hourly_Cooking 54208
Surreal_Sports 46834
PostModern_Games 50644
Baked_Games 51604
Loud_Show 50820
Hourly_Talking 108163
Almost_Show 50202
Hot_Show 54378
Baked_News 47211
PostModern_News 50021
Dumb_Show 53824
Cold_Sports 52005
Hot_Games 50228
Hourly_Show 48283
Almost_Games 49237
Loud_Games 49482
Cold_News 47924

All the commands to run the code are saved in run_script.sh which is in the same folder as mapper and reducer function

bash run_scrip.sh

Comments

- This problem belongs to **Structural Map Reduce Design Pattern**.

III. Multiplication Operations

Solution: Vector-Vector Multiplication

First we describe data generation

The script **generate_data_vv.py** contains the code to generate random vectors between size 1 to 10 and their values can be between 0 to 10. These vectors are named as Vector A and Vector B and stored in a file called input_file.txt The input file contains two vectors in the following form:

$Vector\_Variable\_Name, < Index >, < Value >$

The following is an illustration:
A,0,1
A,1,0
A,2,1
A,3,5
A,4,7
A,5,2
B,0,4
B,1,8
B,2,3
B,3,1
B,4,9
B,5,7

```
# Generate Two Random Vectors of any size and save #
#them in a file, which will be an
#input to Mapreduce Vector Vector Multiplication Task
# Two Vectors are A and B
import numpy as np

#Randomly Select Size of Vector between 1-10
vector_size = np.random.randint(1,10)
#Writing Result to input file
with open('input.txt', 'w') as the_file:

        for i in range(vector_size):
                the_file.write('A')
                the_file.write(',')
                the_file.write(str(i))
```

```
                the_file.write(',')
                #Writing values of Vector A between 0-10
                the_file.write(str(np.random.randint(0,10)))
                the_file.write('\n')

        for i in range(vector_size):
                the_file.write('B')
                the_file.write(',')
                the_file.write(str(i))
                the_file.write(',')
                #Writing values of Vector B between 0-10
                the_file.write(str(np.random.randint(0,10)))
                the_file.write('\n')
```

The Code for mapper is:

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

import sys
import string
import numpy

# input comes from STDIN (stream data that goes to the program)
for line in sys.stdin:

  #Remove leading and trailing whitespace
  line = line.strip()

  #Split line into array of entry data
  entry = line.split(",")
  index = entry[1] #Storing the index
  number = entry[2] #Storing the value

  print '%s\t%s' % (index,number)
```

The output of the mapper function and sort module is:

0 1
0 4
1 0
1 8

2 1
2 3
3 1
3 5
4 7
4 9
5 2
5 7

The Code for reducer is:

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

import sys
import string
import numpy

#Initialising the Variables
last_key=""
product=1
count=0
total = 0

for line in sys.stdin:


  line = line.strip() #removing whitespaces


  key, value = line.split('\t',1)
  #Checking if last is empty and current key is
  #not equal to last key

  if last_key != "" and key != last_key:
    if count==2:
      #print '%s\t%s' % (last_key,product)
      #Accumulating the Sum
      total=total+int(product)

    product = int(value) #typecasting value into integer
    last_key = key #Setting last key to current key
    count = 1
```

```
  else:
    product = product*int(value)#Multiplying prervious product value with value
    last_key=key #Set last key equal to current key
    count=count+1

#the last key
if count==2:
  #print '%s\t%s' % (last_key,product)
  total = total + int(product)

print total
```

hadoop fs -cat part-00000 (**Output of the reducer function**)
89

All the commands to run the code are saved in **run_script.sh** which is in the same folder as mapper and reducer function

bash run_scrip.sh

Comments

- **This implementation of Vector Vector Multiplication is not very efficient as N (Size of Vector) groups of numbers are shuffled to reducer which gives rise to very high computational cost. The shuffling can be reduced by combining map indices in mapper or using index ranges of certain length R.**

Solution: Matrix-Vector Multiplication

For this question we generate two input files **input_vector.txt** and **input_matrix.txt**

The following is the code for data generation: Overall the idea is to generate random matrices with known sizes. The vector can be generated using the similar approach in the previous part. Here we are assuming that the vector is small enough to be stored in the memory.

```
# Generate Two Random Vectors of any size and save them in a
#file, which will be an input to Mapreduce
#Vector Vector Multiplication Task

# Two Vectors are A and B
import numpy as np

#Randomly Choosing the Matrix Row Size between 2-10
matrix_row = np.random.randint(2,10)
#Randomly Choosing the Matrix Column Size between 2-10
matrix_col = np.random.randint(2,10)
#Matrix Col will be equal to Vector Size
vector_size = matrix_col

# Code to Generate the Vector and write it to a file
with open('input_vector.txt', 'w') as the_file:

        for i in range(vector_size):
                #the_file.write(str(i))
                #the_file.write(',')
                #Randomly Selecting Vector Values between 0-10
                the_file.write(str(np.random.randint(0,10)))
                the_file.write('\n')

# Code to Generate the Matrix and write it to a file
M = np.matrix(np.random.randint(0,10, size=(matrix_row, matrix_col)))

#Writing Matrix into Input_Matrix.txt file
with open('input_matrix.txt', 'w') as the_file:
        for (i,j), value in np.ndenumerate(M):
                if value==0:
                        continue
                else:
                        the_file.write(str(i))
```

```
the_file.write(',')
the_file.write(str(j))
the_file.write(',')
the_file.write(str(value))
the_file.write('\n')
```

The following is an illustration:

An example of the input matrix:

$$M = \begin{bmatrix} 0 & 4 & 1 & 6 & 9 \\ 2 & 3 & 0 & 7 & 0 \\ 1 & 0 & 8 & 1 & 7 \\ 6 & 1 & 2 & 1 & 8 \\ 9 & 9 & 2 & 2 & 5 \\ 5 & 6 & 4 & 0 & 9 \end{bmatrix}$$

This matrix is represented as following in the input file:

0,1,4
0,2,1
0,3,6
0,4,9
1,0,2
1,1,3
1,3,7
2,0,1
2,2,8
2,3,1
2,4,7
3,0,6
3,1,1
3,2,2
3,3,1
3,4,8
4,0,9
4,1,9
4,2,2
4,3,2
4,4,5
5,0,5
5,1,6
5,2,4

5,4,9

Note that we do not represent 0 values in the matrix.

The vector in the input_vector.txt has been represented as a column vector. **Here we are assuming that the vector can be stored in the memory without any problems.**

2
9
4
8
9

The mapper code is:

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

import sys
import string
import numpy

# In this implementation we assume that the  vector can be stored in the memory

#Initializing a empty vector list
v = []

#opens file with name of "input_vector.txt"
f = open('input_vector.txt','r')

while True:
  #Reading lines from input vector file
  #and appending it to vector v
        line = f.readline()
        line = line.strip()
        if not line:
                break
        else:
                v.append(int(line))
f.close()

# input comes from STDIN (stream data that goes to the program)
for line in sys.stdin:
```

```
#Remove leading and trailing whitespace
line = line.strip()

#Split line into array of entry data
entry = line.split(",")
row = int(entry[0])
col = int(entry[1])
value = int(entry[2])
temp = v[col]*value #Multiplying Vector Column value
#with Matrix value of a row but with the same column
print '%s\t%s' % (row,temp) #printing row and temp
```

The output of the mapper function and sort module is:

0 36
0 4
0 48
0 81
1 27
1 4
1 56
2 2
2 32
2 63
2 8
3 12
3 72
3 8
3 8
3 9
4 16
4 18
4 45
4 8
4 81
5 10
5 16
5 54
5 81

The reducer code is:

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

```
import sys
import string
import numpy

#Initialization of the Value
current_key = None
current_value = 0
key = None

#Looping over values on Standard Output
for line in sys.stdin:
    line = line.strip() #Remove Whitespaces
    key, value = line.split('\t',1)

    # Convert the value from string into Integer
    #Introducing Exception Handling
    try:
        value = int(value)
    except ValueError:
        continue

    # if the current key is equal to key
    if current_key == key:
        current_value += value # Incrementing Current Value by Value
    else:
        if current_key:
        # Printing the current key and current value
            print '%s\t%s' % (current_key, current_value)
        current_value = value #Set current_value to value
        current_key = key #Set current_key to key

# Printing Last Value
if current_key == key:
    print '%s\t%s' % (current_key, current_value)
```

The solution of Matrix-Vector Multiplication

All the commands to run the code are saved in run_script.sh
bash run_scrip.sh
hadoop fs -cat part-00000 (**Output of the reducer function**)

0 169
1 87

2 105
3 109
4 168
5 161

Comments

- **Matrix Vector multiplication is a building block in Google's PageRank Mapreduce implementation. It can also be used for computing cosine similarity to compute similarity between a document and all other documents.**