Mei Tan Le Ping, lt528@live.mdx.ac.uk
M00724895

Task 1 : To Count Even and Odd character of words.

1.1 Create File

Firstly a text file "task1_mei" as shown below was created with a line of words "my name is Mei Tan". This text file as input will be read and count for the sum of even and odd number of characters.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ vi task1_mei.txt
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task1_mei.txt
my name is Mei Tan
```

1.2 Python MapReduce Code

Python code was used for later Hadoop Streaming API as to assist in passing data in between Map and Reduce code via the STDIN (standard input) and STDOUT (standard output). Python's *sys.stdin* was used to read input data and print out the output as *sys.stdout*.

1.3 Map Step : mapper1.py

A mapper1.py file was created as below code to read data from STDIN and split into words and out as list of lines mapping words to their counts to STDOUT. These code will output either "even" 1 or "odd" 1 tuple immediately. All the words character counts either even or odd will be list out without summation. As the case will subsequently do by reducer to final the sum of count. The execution permission of file was being updated as *chmod -R 777 mapper1.py* to avoid any execution problems.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ vi mapper1.py
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ chmod -R 777 mapper1.py
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task1.mei.txt | mapper1.py
cat: task1.mei.txt: No such file or directory
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task1_mei.txt | mapper1.py
even    1
even    1
even    1
odd     1
odd     1
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat mapper1.py
#!/usr/bin/python3
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # Output here will be Input for the Reduce step
        # word length could divide by 2 define as even else odd
        length = len(word)
        if length % 2 == 0:
            print ( '%s\t%s' % ('even', 1) )
        else:
            print ( '%s\t%s' % ('odd', 1) )
```

1.4 Reduce Step : reducer1.py

A reducer1.py code as below was saved to read the results from mapper1.py via STDIN where the output from mapper1.py and the expected input format of reducer1.py were matched. Later will be total the sum of count and output as the STDOUT.Same as in mapping stage, the access permission execution of reducer1.py was updated as as *chmod -R 777 reducer1.py* to avoid any execution problems.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ vi reducer1.py
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat reducer1.py
#!/usr/bin/python3
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print ('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print ('%s\t%s' % (current_word, current_count))
```

1.5 Code testing of mapper and reducer

Testing of the functionality of scripts *mapper1.py and reducer1.py* were done before using in Hadoop MapReduce job.Below shown the expected output of *cat task1_mei.txt | mapper1.py | sort | reducer1.py*

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task1_mei.txt | mapper1.py | sort | reducer1.py
even    3
odd     2
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$
```

1.6 Run MapReduce job

By using Python MapReduce job run on the Hadoop cluster. Thisleveraging the Hadoop Streaming API to assist passing the data between Map and Reduce code via STDIN and STDOUT Hadoop-streaming.jar. A result of output ‘part-00000’ text file will be generated.

Mei Tan Le Ping, lt528@live.mdx.ac.uk
M00724895

Task 2 : Student's Grade.

2.1 Create File

A csv file "task2_mei" as shown below was created with a lines of words separated by ',' and serve as input data for mapper and reducer jobs later. Python code was used for later Hadoop Streaming API as to assist in passing data in between Map and Reduce code via the STDIN (standard input) and STDOUT (standard output). Python's *sys.stdin* was used to read input data and print out the output as *sys.stdout*.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task2.csv
cat: task2.csv: No such file or directory
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task2_mei.csv
StudentId,Module,Grade
S001,Statistic,75
S002,Statistic,72
S001,Big Data,78
S003,Big Data,66
S001,Programming,70
S002,Programming,55
S001,Machine Learning,65
S002,Machine Learning,61
```

2.2 Map Step : mapper2.py

A mapper2.py file was created as below code to internally read (stdin) and print out (stdout) line by line. Thus, Python can read each of the lines as string and pars them by fuctions such as strip and split (","). From the input, first (StudentId) and third (Grade) elements were picked in each array and print out (stdout) them as Key-Value pair. The execution permission of file was later being updated as *chmod -R 777 mapper2.py* to avoid any execution problems.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ vi mapper2.py
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat mapper2.py
#!/usr/bin/python3
"""mapper.py"""

import sys

#input from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    if len(line) >= 2:
        studentid = line[0]
        grade = line[2]

        print ('%s\t%s' % (studentid, grade))


hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task2_mei.csv | mapper2.py
StudentId       Grade
S001    75
S002    72
S001    78
S003    66
S001    70
S002    55
S001    65
S002    61
```

Mei Tan Le Ping, lt528@live.mdx.ac.uk
M00724895

2.3 Reduce Step : reducer2.py

Upon mapper code, reducer code was used to parse the string from mapper2.py as Key-Value pair, and kept in dictionary form such as {StudentId: [Grade,…]}. This dictionary were used for further computation of min grade, max grade and count of the modules taken according to StudentId as show in figure below. Again the access permission execution of reducer2.py was updated as as *chmod -R 777 reducer2.py* to avoid any execution problems.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat reducer2.py
#!/usr/bin/python3
"""reducer.py"""

from operator import itemgetter
import sys

StudentId_Grade = {}

# Partitioner
for line in sys.stdin:
    line = line.strip()
    StudentId, Grade  = line.split('\t')

    # convert count (currently a string) to int
    try:
        Grade = int(Grade)
    except ValueError:
        # if count not a number,ignore/discard this line
        continue

    # IF-switch only works as Hadoop sorts map output
    # by key (here: word) before pass to the reducer
    if StudentId in StudentId_Grade:
        StudentId_Grade[StudentId].append(int(Grade))
    else:
        StudentId_Grade[StudentId] = []
        StudentId_Grade[StudentId].append(int(Grade))

# Reducer
print ('StudentId ', 'MinGrade ','MaxGrade ', 'Modules')
for StudentId in StudentId_Grade.keys():
    MinGrade = min(StudentId_Grade[StudentId])
    MaxGrade = max(StudentId_Grade[StudentId])
    Modules = len(StudentId_Grade[StudentId])
    print ('%s\t%s\t%s\t%s' % (StudentId, MinGrade, MaxGrade, Modules))
```

2.4 Code testing of mapper and reducer

Cat command was used to stimulate the process flow to test the code of mapper and reducer and produced the output as show below from command of *cat task2_mei.csv | mapper2.py | sort | reducer2.py*

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task2_mei.csv | mapper2.py | sort | reducer2.py
StudentId  MinGrade  MaxGrade  Modules
S001    65      78      4
S002    55      72      3
S003    66      66      1
```

2.5 Run MapReduce job

Lastly to execute the codes of mapper and reducer on the csv files (task2_mei_csv) in Hadoop should produce the same output as above figure.

Mei Tan Le Ping, lt528@live.mdx.ac.uk
M00724895

Task 3 : User follower

3.1 Create Files

Two csv input files were created with given data as show below, task3a_mei.csv and task3b_mei.csv. Both files were combined and executed in a mapper reducer job.

```
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task3a_mei.csv
UserId,Name,DOB
U001,Alice,2005-01-05
U002,Tom,2001-02-07
U003,John,1998-06-02
U004,Alex,2006-02-01
hduser@stack:/usr/local/hadoop/hadoop-2.6.5/sbin$ cat task3b_mei.csv
UserIdFollower,UserIdFollowing
U001,U002
U001,U003
U002,U001
U002,U004
U003,U001
U004,U001
```

3.2 Map and Reduce Step : mapreduce3.py

A mapreduce3.py file was created as below code to read the two csv files and merge for further mapping and reducing execution line by line. Python was used to read each of the lines as string and pars them by functions such as strip and split (“,”). From the input, combine UserId and second dataset of follower elements were picked in each array and print out (stdout) them as Key-Value pair. The execution permission of file was later being updated as *chmod -R 777 mapreduce3.py* to avoid any execution problems.

```
import sys

#input from STDIN (standard input)
for line in sys.stdin:
    line = line.strip()
    line = line.split(",")

    UserId = "-1"
    Name = "-1"
    DOB = "-1"
    UserIdFollower = "-1"
    UserIdFollowing = "-1"

    if len(line) == 4:
        UserId = line[0]
        Name = line[1]
        DOB = line[2]

    else:
        UserIdFollower = line[0]
        UserIdFollowing = line[1]

    print ('%s\t%s\t%s\t%s\t%s' % (UserId, Name, DOB, UserIdFollower, UserIdFollowing))

    U_dict ={}
    F_dict =[]
    for line in sys.stdin:
        line = line.strip()
        UserId, Name,DOB, UserIdFollower, UserFollowing = line.split('\t'

                UserId = int(UserId)
                NameIdFollower = str(UserIdFollower)
                NameIdFollowing = str(UserIfFollowing)

                if UserId == -1:
                U_dict[UserId] = UserIdFollower
                U_dict[UserId] = UserIdFollowing

                print ( '%s\t%s\t%s' % (UserId, UserIdFollower, UserIdFollowing))
```