Big Data Analytics (CSCI-720)
MapReduce Code Homework Assignment

This assignment is due to the myCourses dropbox by 3:00 pm on October 29.

You may work with one other student in this class on this assignment (team of two people). Otherwise this assignment is to be completed as an *individual effort*.

Your solution MUST run on the CS department machine glados.cs.rit.edu when invoked using the command `python3`. Solutions that do not run on glados will not be given any credit. You can remotely access glados using ssh (or other similar tools).

Go back and read the previous paragraph again. If your solution does not run under python3 on glados, you will not be given any credit for the assignment.

Overview

For last week's homework, you pseudocoded the Map and Reduce solutions for counting the number of tokens in a file (#1), and then counting the occurrence of each word (#2). Below, I show you a possible MR Python code solution for each. Take a close look to be sure you understand what is happening.

Part of what you don't see happens in the MapReduce program. This program is included below – modified to support a text file. Note that unlike "real" MapReduce, this MapReduce does not execute on a cluster. It is a *simulator* designed to give you a feel for how MR works.

Be sure you understand all the code before continuing.

MapReduce.py
- - file starts here

```
#Credit:  Bill Howe at University of Washington
#Note that this assumes the file is in flat text format
#
# Modified October 2014 by tmh to run under Python 3
#
# Modified October 2015 from original version to process text files
instead of Json files
#

class MapReduce:
    def __init__(self):
        self.intermediate = {}
        self.result = []

    def emit_intermediate(self, key, value):
        self.intermediate.setdefault(key, [])
        self.intermediate[key].append(value)
    def emit(self, value):
        self.result.append(value)    ← you can emit from the reducer,
generally done if your problem requires some form of post-processing.
If you do, the results (on this simulator) are appended to this data
structure, and the data structure is dumped at the end of the
‘execute’ function.


    # pass execute an open file in 'data'
    def execute(self, data, mapper, reducer):

        inputFile = open (data);
        for eachLine in inputFile:
            mapper(eachLine)

        for key in self.intermediate:
            reducer(key, self.intermediate[key])


        for item in self.result:  ← this is the line that prints
what was emitted in the reducer.  It is empty if you do not emit in
the reducer (prints a blank line).

            print (item)
```

- - file ends here

Notice MapReduce.py *simulates* the Sort and Shuffle operations using data structures. This obviously would not work on a very large data set typically seen with a “real” MR problem, but it does give you a feel for how MR works. For this assignment, you should not modify the MapReduce.py program.

Here is a solution to count tokens in a file (#1 from last week’s homework). Notice the code is very simple – it ignores case and assumes all punctuation has been stripped.

WordCount.py

```
 - - file starts here
#Author:  Trudy Howles

# count all words, then print the total
# does not consider case or look for duplicate words - simply counts
tokens
# Requires MapReduce.py in the same working directory

import MapReduce
import sys

mr = MapReduce.MapReduce()

def mapper (data):
    tokens = data.split()
    for eachToken in tokens:
        mr.emit_intermediate ("COUNT", 1)    ← note that COUNT is a
“dummy” key so all numeric counts emitted are associated with the
same Key.

def reducer (key, list_of_values):

    print ("Total words = ", len (list_of_values))    ← here, I can
perform the output in the reducer since there will be a single
reducer (since all mappers emitted the same key)
    print ("\n")
    print ("All done")

if __name__ == '__main__':
      mr.execute(sys.argv[1], mapper, reducer)
```

- - file ends here

Here is a solution to count the occurrence of each token in a file (#2 from last week's homework). Notice this code also very simple – it ignores case and assumes all punctuation has been stripped. Notice the different behavior in the mapper to support counting the word occurrences.

- - file starts here

```
#Author:  Trudy Howles

# count all words, then print the occurrence count for each word
# does not consider case

import MapReduce
import sys

mr = MapReduce.MapReduce()

def mapper (data):
    tokens = data.split()
    for eachToken in tokens:
        mr.emit_intermediate (eachToken, 1)  ← here I must emit
with each specific word so all occurrences are associate with the
same Key

def reducer (key, list_of_values):
    value = len (list_of_values)
    print (key, " occurred  ", value, " times.")  ← Again, I can
print from the reducer because no two reducers will receive the same
Key
    print ("\n")
    print ("All done")

if __name__ == '__main__':
      mr.execute(sys.argv[1], mapper, reducer)
```

- - file ends here

Be sure you understand this simple code. To run a program named WordCount.py, you would issue this command (with MapReduce.py in the same working directory):

```
  python3 WordCount.py Test.txt
```

Again, note that python3 is required, and the data file name (Test.txt in this case) is expected as a command line argument.

This Week's Assignment

This week, you will implement a MR solution on your own. Two important items to remember:

> Your Python solution must run with my MapReduce.py program. It is available in the Content Area (it is the same file shown in this document). Do not modify this program. Your solution must run under Python Version 3 on glados.

I am requiring that your program run on glados because some CS lab machines have different hardware architectures and may support different versions of compilers and tools. Be sure your final solution correctly executes on this machine.

For this assignment, you will work with a large web log activity file. The file is located here:
/usr/local/pub/large_log_files/access_log_new

This file is very large so you should not attempt to copy it into your personal account or you will very likely experience disk quota issues.

You must implement a Python MapReduce solution that runs with my Python MapReduce simulator, as described below. Name your solution CountURLS.py.

Your program must process the file referenced above, and your program must expect the file name as a command line argument. For example (on glados):

```
python3 CountURLS.py /usr/local/pub/large_log_files/access_log_new
```

No credit for your submission if it requires any other user input! Your solution must be implemented in Python. You should already be experienced writing Java programs, so Python should be easy for you to pick up on your own. If you need help, an online Python tutorial is available.

All output must go to standard output. Do not modify my data file to work with your program.

Here is the problem:

The security people have determined that repeated accesses to the same URL can sometimes indicate a potential threat. **How many** URLs in the indicated file were accessed more than 20 times? In order to be a match, the full URL must match. In the samples below, both accessed ~pga/pics2000 but the *full* URLs are different.

Here are two samples showing where to locate the URL:

Raw data – Sample 1:

```
203.204.153.85 - - [30/Nov/2002:01:08:19 -0500] "GET /~pga/pics2000/img3.gif HTTP/1.1" 200 276
```

URL accessed: /~pga/pics2000/img3.gif

Raw data – Sample 2:

```
203.204.153.85 - - [30/Nov/2002:01:08:19 -0500] "GET /~pga/pics2000/img4.gif HTTP/1.1" 200 258
```

URL accessed: /~pga/pics2000/img4.gif

Your only output produced by your program should be a single count with an appropriate label. Do not include any other output, and be sure you remove all debug statements before you submit.

Deliverables

As indicated above, name your solution `CountURLS.py`.

Be sure to use this *exact* file name – *I will test your program using a script and will make deductions if I must rework my scripts to accommodate non-standard naming conventions.*

For your program, you must expect the file name as the command line argument. All output must go to standard output. Do not modify my data file to work with your program!

What to Submit to the Dropbox

> **Submit your code (CountURLS.py) and a brief writeup described below. Do not upload MapReduce.py to the dropbox.**
>
> If you work with a partner from class, only one of you should submit a solution. Be your name(s) is(are) included in the code and the writeup.

The writeup: Submit a maximum 1 page writeup to the myCourses dropbox, including the following:

1. Your name(s) and username(s)
2. The required output as described above.
2. A summary of your approach.
3. Problems or issues you encountered.

Be sure you spell and grammar check before submitting. I will make severe deductions for sloppy writing and reports that do not reflect much effort. Submit only a Word or pdf file for your writeup.

Grading

No credit for non-working programs. You must submit working code in order to get credit for any portion of the writeup.

Your project MUST run on the CS department machine glados.cs.rit.edu when invoked using the command `python3`. Solutions that do not run on glados will not be given any credit.

Hint: This is actually a very simple MR problem. My solution, including all documentation headers and generous comments is less than 40 lines of Python code. Your solution should not involve using any data structures. Be sure you are taking advantage of MR.

All submissions are subject to an originality check.

Rev. 10/22/2015