Quiz 4: Hadoop for Fun and Profit

Casey Owen

October 11, 2024

CS119 Big Data

Spring 2024

1 Part 1 - Functional Programming

1.1 Question 1: Create Arithmetic Functional Functions

[112]:
```
from functools import reduce

def add(*args):
    return reduce(lambda x, y: x + y, args, 0)

def sub(*args):
    return reduce(lambda x, y: x - y, args)

def ra_sub(*args):
    if len(args) == 1:
        return args[0]
    else:
        # Unpack the tuple so each element is one argument
        return args[0] - ra_sub(*args[1:])
```

1.1.1 Test the outputs against known answer

[113]:
```
add_result = add(1, 2, 3)
print(add_result)
if add_result == 6:
    print("CORRECT")
else:
    print("INCORRECT")

sub_result = sub(5, 1, 2)
print(sub_result)
if sub_result == 2:
    print("CORRECT")
else:
    print("INCORRECT")
```

```
ra_sub_result = ra_sub(5, 1, 2)
print(ra_sub_result)
if ra_sub_result == 6:
    print("CORRECT")
else:
    print("INCORRECT")
```

```
6
CORRECT
2
CORRECT
6
CORRECT
```

1.2 Question 2: Create Zip Function

[114]:
```
import numpy as np

def add_to_zipped(zipped, seq):
    # Concatenates the elements of the existing zipped, and the sequence, for each element
    return list(map(lambda zipped, seq: [*zipped, seq], zipped, seq))

def my_zip(*args):
    # Starts with a series of empty lists, one for each element of a given sequence (all must be the same length), and appends to it, one sequence at a time
    return reduce(add_to_zipped, args, [[]]*len(args[0]))
```

1.2.1 Test the outputs against known answer

[115]:
```
zip_result_1 = my_zip([1,2,3],[4,5,6])
print(zip_result_1)
if zip_result_1 == [[1, 4], [2, 5], [3, 6]]:
    print("CORRECT")
else:
    print("INCORRECT")

zip_result_2 = my_zip([1,2,3],[4,5,6],[7,8,9])
print(zip_result_2)
if zip_result_2 == [[1, 4, 7], [2, 5, 8], [3, 6, 9]]:
    print("CORRECT")
else:
    print("INCORRECT")
```

```
[[1, 4], [2, 5], [3, 6]]
CORRECT
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
CORRECT
```

1.3 Question 3: Create Zipwith Function

[116]:
```
def zipwith(f, *args):
    return list(map(f, *args))
```

1.3.1 Test the outputs against known answer

[117]:
```
zipwith_result_1 = zipwith(add, [1, 2, 3], [4, 5, 6])
print(zipwith_result_1)
if zipwith_result_1 == [5, 7, 9]:
    print("CORRECT")
else:
    print("INCORRECT")

zipwith_result_2 = zipwith(add, [1, 2, 3], [4, 5, 6], [1, 1, 1])
print(zipwith_result_2)
if zipwith_result_2 == [6, 8, 10]:
    print("CORRECT")
else:
    print("INCORRECT")
```

```
[5, 7, 9]
CORRECT
[6, 8, 10]
CORRECT
```

1.4 Question 4: Create Flatten Function

[118]:
```
from functools import reduce

def concat_ints(x, y):
    '''
    Concatenates two values, x and y, that may either be ints or lists of ints,␣
 ↪into a single list of ints
    '''
    if isinstance(x, int):
        if isinstance(y, int):
            return [x, y]
        else:
            return [x, *y]
    else:
        if isinstance(y, int):
            return [*x, y]
        else:
            return [*x, *y]
```

```
def flatten(tree:list):
    # Recursion base case is if all elements of tree are ints rather than lists
    if reduce(lambda prev, x: prev and isinstance(x, int), tree, True):
        return tree
    else:
        return flatten(reduce(concat_ints, tree))
```

1.4.1 Test the outputs against known answer

[119]:
```
flatten_result = flatten([1, [2, [3, 4], [5, 6], 7], 8, [9, 10]])
print(flatten_result)
if flatten_result == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    print("CORRECT")
else:
    print("INCORRECT")
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
CORRECT
```

1.5 Question 5: Create Groupby Function

[120]:
```
from copy import deepcopy
from functools import reduce

def add_key_to_dict(d: dict, tup: tuple):
    key, val = tup
    # Need to deepcopy since dicts are mutable - not editing the existing d
    new_dict = deepcopy(d)
    if key in new_dict:
        old_val = d[key]
    else:
        old_val = []
    # This combines dicts into new, updating keys from d with new key
    return new_dict | {key: old_val + [val]}

def group_by(func, seq):
    func_outputs =list(map(func, seq))
    zipped = my_zip(func_outputs, seq)
    return reduce(add_key_to_dict, zipped, {})
```

1.5.1 Test the outputs against known answer

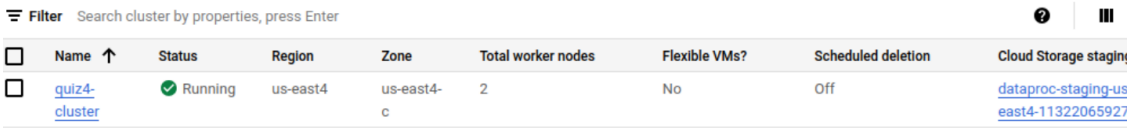
[121]:
```
grouby_result = group_by(len, ["hi", "dog", "me", "bad", "good"])
print(grouby_result)
if grouby_result == {2: ["hi", "me"], 3: ["dog", "bad"], 4: ["good"]}:
    print("CORRECT")
```

```
else:
    print("INCORRECT")
```

```
{2: ['hi', 'me'], 3: ['dog', 'bad'], 4: ['good']}
CORRECT
```

2 Part 2 - Confirming Hadoop Installation

2.1 Question 1: Aquire the Cluster



Filter Search cluster by properties, press Enter

Name	Status	Region	Zone	Total worker nodes	Flexible VMs?	Scheduled deletion	Cloud Storage staging
quiz4-cluster	Running	us-east4	us-east4-c	2	No	Off	dataproc-staging-us-east4-113220659277

2.2 Question 2: Load the data into the master

Make quiz4 directory on hdfs:

hadoop fs -mkdir /quiz4

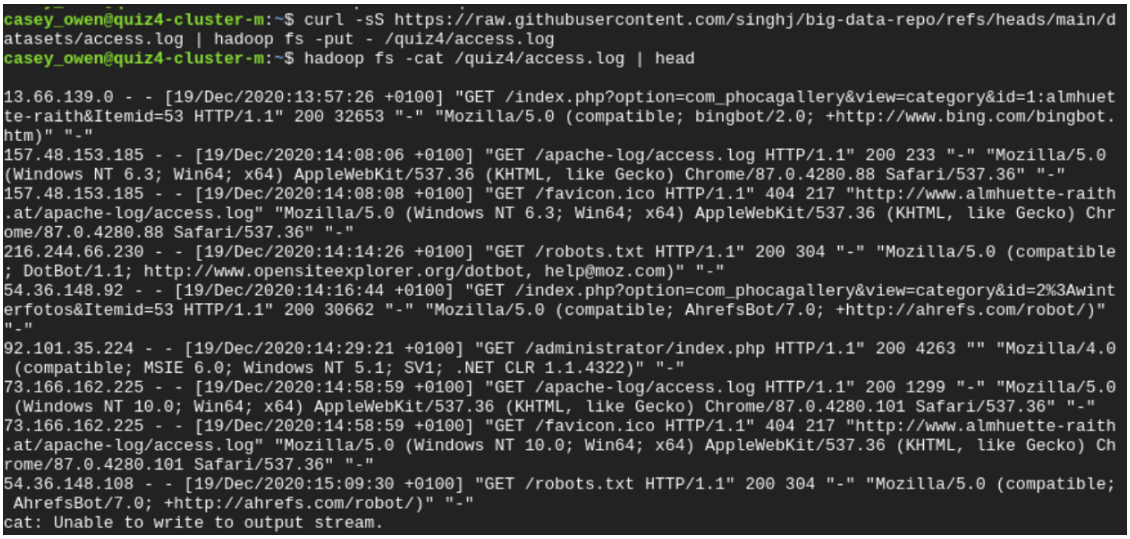
Get assignment file via curl and put it in the new folder as access.log:

curl -sS https://raw.githubusercontent.com/singhj/big-data-repo/refs/heads/main/datasets/access.log | hadoop fs -put - /quiz4/access.log

View the outputs:

hadoop fs -cat /quiz4/access.log | head

output is top 10 rows of raw file, as expected



```
casey_owen@quiz4-cluster-m:~$ curl -sS https://raw.githubusercontent.com/singhj/big-data-repo/refs/heads/main/datasets/access.log | hadoop fs -put - /quiz4/access.log
casey_owen@quiz4-cluster-m:~$ hadoop fs -cat /quiz4/access.log | head

13.66.139.0 - - [19/Dec/2020:13:57:26 +0100] "GET /index.php?option=com_phocagallery&view=category&id=1:almhuette-raith&Itemid=53 HTTP/1.1" 200 32653 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"
157.48.153.185 - - [19/Dec/2020:14:08:06 +0100] "GET /apache-log/access.log HTTP/1.1" 200 233 "-" "Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36" "-"
157.48.153.185 - - [19/Dec/2020:14:08:08 +0100] "GET /favicon.ico HTTP/1.1" 404 217 "http://www.almhuette-raith.at/apache-log/access.log" "Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36" "-"
216.244.66.230 - - [19/Dec/2020:14:14:26 +0100] "GET /robots.txt HTTP/1.1" 200 304 "-" "Mozilla/5.0 (compatible; DotBot/1.1; http://www.opensiteexplorer.org/dotbot, help@moz.com)" "-"
54.36.148.92 - - [19/Dec/2020:14:16:44 +0100] "GET /index.php?option=com_phocagallery&view=category&id=2%3Awinterfotos&Itemid=53 HTTP/1.1" 200 30662 "-" "Mozilla/5.0 (compatible; AhrefsBot/7.0; +http://ahrefs.com/robot/)" "-"
92.101.35.224 - - [19/Dec/2020:14:29:21 +0100] "GET /administrator/index.php HTTP/1.1" 200 4263 "" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)" "-"
73.166.162.225 - - [19/Dec/2020:14:58:59 +0100] "GET /apache-log/access.log HTTP/1.1" 200 1299 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.101 Safari/537.36" "-"
73.166.162.225 - - [19/Dec/2020:14:58:59 +0100] "GET /favicon.ico HTTP/1.1" 404 217 "http://www.almhuette-raith.at/apache-log/access.log" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.101 Safari/537.36" "-"
54.36.148.108 - - [19/Dec/2020:15:09:30 +0100] "GET /robots.txt HTTP/1.1" 200 304 "-" "Mozilla/5.0 (compatible; AhrefsBot/7.0; +http://ahrefs.com/robot/)" "-"
cat: Unable to write to output stream.
```

2.3 Question 3: Run Wordcount on five-books

Run hadoop jar command to create mapreduce job:

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /five-books /books-count

Get results of job once complete:

hadoop fs -get /books-count

View results:

```
casey_owen@quiz4-cluster-m:~$ ls books-count/
_SUCCESS  part-r-00000  part-r-00001  part-r-00002
```

```
casey_owen@quiz4-cluster-m:~/books-count$ cat part-r-00000 | head
"'Miss  1
"'Tis   3
"'Twas  1
"'_We   1
"(4)    1
"--I    1
"--and  3
"--but  1
"--change       1
"--so   1
```

2.4 Question 4: Run Wordcount using mapper_noll and aggregate

Run mapred streaming command to create mapreduce job:

mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -mapper mapper_noll.py -input /five-books -reducer aggregate -output /books-stream-count

Get results of job onto master once complete:

hadoop fs -get /books-stream-count

View results:

```
casey_owen@quiz4-cluster-m:~$ cd books-stream-count/
casey_owen@quiz4-cluster-m:~/books-stream-count$ ls
_SUCCESS  part-00000  part-00001  part-00002
casey_owen@quiz4-cluster-m:~/books-stream-count$ cat part-00000 | head
a       4736
abandoned       2
abate   1
abbreviated     1
abc     5
abenteuer       1
abiding 1
able    42
abolished       2
abounds 1
```

2.5 Question 5: Run wordcount using mapper_noll and reducer_noll

Run mapred streaming command to create mapreduce job:

mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -file ~/big-data-repo/hadoop/reducer_noll.py -mapper mapper_noll.py -reducer reducer_noll.py -input /five-books -output /books-my-own-counts

Get results of job onto master once complete:

hadoop fs -get /books-my-own-counts

View results, which are word counts formatted accoring to custom reducer, as expected

```
casey_owen@quiz4-cluster-m:~/books-stream-count$ cd books-my-own-counts/
casey_owen@quiz4-cluster-m:~/books-stream-count/books-my-own-counts$ cat part-00000 | head
LongValueSum:a  4736
LongValueSum:abandoned  2
LongValueSum:abate      1
LongValueSum:abbreviated        1
LongValueSum:abc        5
LongValueSum:abenteuer  1
LongValueSum:abiding    1
LongValueSum:able       42
LongValueSum:abolished  2
LongValueSum:abounds    1
casey_owen@quiz4-cluster-m:~/books-stream-count/books-my-own-counts$
```

3 Part 3 - Analyzing Server Logs

3.0.1 Question 1: Get the percentage of each request type (GET, PUT, POST, etc)

Command and Results: Put files from master onto hdfs after uploading

hadoop fs -put quiz4/Part3Question1_reducer.py quiz4/Part3Question1_mapper.py /quiz4/

MapReduce Job

mapred streaming -file ~/quiz4/Part3Question1_mapper.py -file ~/quiz4/Part3Question1_reducer.py -mapper Part3Question1_mapper.py -reducer Part3Question1_reducer.py -input /quiz4/access.log -output /quiz4/Part3Question1

Get Results from hdfs to master:

hadoop fs -get /quiz4/Part3Question1 \quiz4

Post-Processing to turn counts into Percentages. Bash command that loops through concatenated reducer results twice - first to sum up total occurences, second time to divide by the sum

awk 'NR==FNR{sum+= $2; next}{$2/=sum; print $0}' <(cat quiz4/Part3Question1/*) <(cat quiz4/Part3Question1/*) > quiz4/Part3Question1Results.txt

Results:

```
casey_owen@quiz4-cluster-m:~$ cat quiz4/Part3Question1Results.txt
RequestType:HEAD 0.00323319
RequestType:POST 0.569756
RequestType:GET 0.427011
```

I did this as one mapreduce job to simply create the counts of each unique term, then a simple bash command to transform the counts into percentages. This could not have been done as one single job to calculate the percents with multiple reduce tasks, since we can't know the overall count of all keys until all reducers are finished, and therefore can't calculate the percentage within any given reducer. So I framed the problem as a word counting, and a postprocessing command. This method is generalizeable to extremely large data since there were only 3 possible requests, so the bash computation to convert to percentages was trivial and did not necessitate another mapreduce job.

Part3Question1_mapper.py

[ ]:
```
#!/usr/bin/env python
import sys, shlex


def main(argv):
    line = sys.stdin.readline()
    try:
        while line:
            # Using shlex to split since it more naturally splits on the file - it uses spaces for delimeters, with quoted fields that sometimes contain spaces
            linelist = shlex.split(line)
            # Only consider input if we get full row exactly - may be slightly different if delimiter changes
            if len(linelist) == 11:
                # The command with the parameter is the 6th column
                request = linelist[5]
                # The request type itself is space seperated from the rest of the command
                request_type = request.split(' ')[0]
                print("RequestType:" + request_type.upper() + "\t" + "1")
            line = sys.stdin.readline()
    except EOFError as error:
        return None

if __name__ == "__main__":
    main(sys.argv)
```

Part3Question1_reducer.py (same as reducer_noll.py)

[ ]:
```
#!/usr/bin/env python
"""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))
```

3.0.2 Question 2: Get the percentage of each response type (100-199, 200-299, etc)

Very similar process to Question 1 - the main difference is the mapper file

Command and Results: Put files from master onto hdfs

hadoop fs -put quiz4/Part3Question2_reducer.py quiz4/Part3Question2_mapper.py /quiz4/

MapReduce Job

mapred streaming -file ~/quiz4/Part3Question2_mapper.py -file ~/quiz4/Part3Question2_reducer.py -mapper Part3Question2_mapper.py -reducer Part3Question2_reducer.py -input /quiz4/access.log -output /quiz4/Part3Question2

Get Results from hdfs to master:

hadoop fs -get /quiz4/Part3Question2 \quiz4

Post-Processing to turn counts into Percentages, same command as before with different files

awk 'NR==FNR{sum+= $2; next}{$2/=sum; print $0}' <(cat quiz4/Part3Question2/*) <(cat quiz4/Part3Question2/*) > quiz4/Part3Question2Results.txt

Results:

```
casey_owen@quiz4-cluster-m:~$ cat quiz4/Part3Question2Results.txt
ResponseType:400-499 0.0592708
ResponseType:200-299 0.903298
ResponseType:300-399 0.0374308
casey_owen@quiz4-cluster-m:~$ ls
```

Part3Question2_mapper.py

[ ]:
```
#!/usr/bin/env python
import sys, shlex

def main(argv):
    line = sys.stdin.readline()
    try:
        while line:
            linelist = shlex.split(line)
            # Only consider input if we get full row exactly - may be slightly different if delimiter changes
            if len(linelist) == 11:
                response = linelist[6]
                # Only need to consider leading digit of the response code for grouping
                response_type = response[0] + "00-" + response[0] + "99"
                print("ResponseType:" + response_type + "\t" + "1")
            line = sys.stdin.readline()
    except EOFError as error:
        return None

if __name__ == "__main__":
    main(sys.argv)
```

Part3Question2_reducer.py (again, same as reducer_noll.py)

[ ]:
```
#!/usr/bin/env python
"""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))
```

3.0.3 Question 3: Get the 5 IP addresses that return the most client errors

Very similar process to Question 1 and 2 - the main differences are the mapper file, and the command at the end

Command and Results: Put files from master onto hdfs

hadoop fs -put quiz4/Part3Question3_reducer.py quiz4/Part3Question3_mapper.py /quiz4/

MapReduce Job

mapred streaming -file ~/quiz4/Part3Question3_mapper.py -file ~/quiz4/Part3Question3_reducer.py -mapper Part3Question3_mapper.py -reducer Part3Question3_reducer.py -input /quiz4/access.log -output /quiz4/Part3Question3

Get Results from hdfs to master:

hadoop fs -get /quiz4/Part3Question3 \quiz4

Post-Processing to sort the counts, and take the top five

cat quiz4/Part3Question3/* | sort -r -n -k 2 | head -5 > quiz4/Part3Question3Results.txt

This command requires the -r, -n, and -k 2 flags to sort the results in descending order, numerically instead of alphabetically, and by the second column which is the count.

Results:

```
casey_owen@quiz4-cluster-m:~$ cat quiz4/Part3Question3Results.txt
IPAddress:173.255.176.5 2059
IPAddress:212.9.160.24  126
IPAddress:13.77.204.88  78
IPAddress:51.210.243.185        58
IPAddress:193.106.30.100        53
```

This approach of using a post-processing command is somewhat less scalable than the question 1 and 2 approaches, since there may be a significant amount of unique IP addresses in extremely large data. The Mapreduce job did reduce the size of the file that needs to be processed from 78,252 lines, 12 columns down to 803 lines, 2 columns of the reducer output. If working with data that contains millions or billions of unique IP addresses, extra thought will be required. This could be a simple change, though - for example, you could pass the results into another mapreduce job that filtered the IP address counts by counts greater than some threshold (maybe 10), which would greatly reduce the number of rows and thus the sorting workload.

Part3Question3_mapper.py

[ ]:
```
#!/usr/bin/env python
import sys, shlex

def main(argv):
    line = sys.stdin.readline()
    try:
        while line:
            linelist = shlex.split(line)
            # Only consider input if we get full row exactly - may be slightly different if delimeter changes
            if len(linelist) == 11:
                response = linelist[6]
                # Is a client error if the response code starts with a 4
                if response[0] == '4':
                    ip_address = linelist[0]
```

```
                    print("IPAddress:" + ip_address + "\t" + "1")
            line = sys.stdin.readline()
    except EOFError as error:
        return None

if __name__ == "__main__":
    main(sys.argv)
```

Part3Question3_reducer.py (again, same as reducer_noll.py)

[ ]:
```
#!/usr/bin/env python
"""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))
```

4 Part 4 - Presidential Speeches

Similar process again as each Question in Part 3, with custom made mapper and reducer files

Command and Results: Put files from master onto hdfs

hadoop fs -put quiz4/Part4_reducer.py quiz4/Part4_mapper.py /quiz4/

MapReduce Job

mapred streaming -file ~/quiz4/Part4_mapper.py -file ~/quiz4/Part4_reducer.py -mapper Part4_mapper.py -reducer Part4_reducer.py -input /quiz4/prez_speeches/* -output /quiz4/Part4

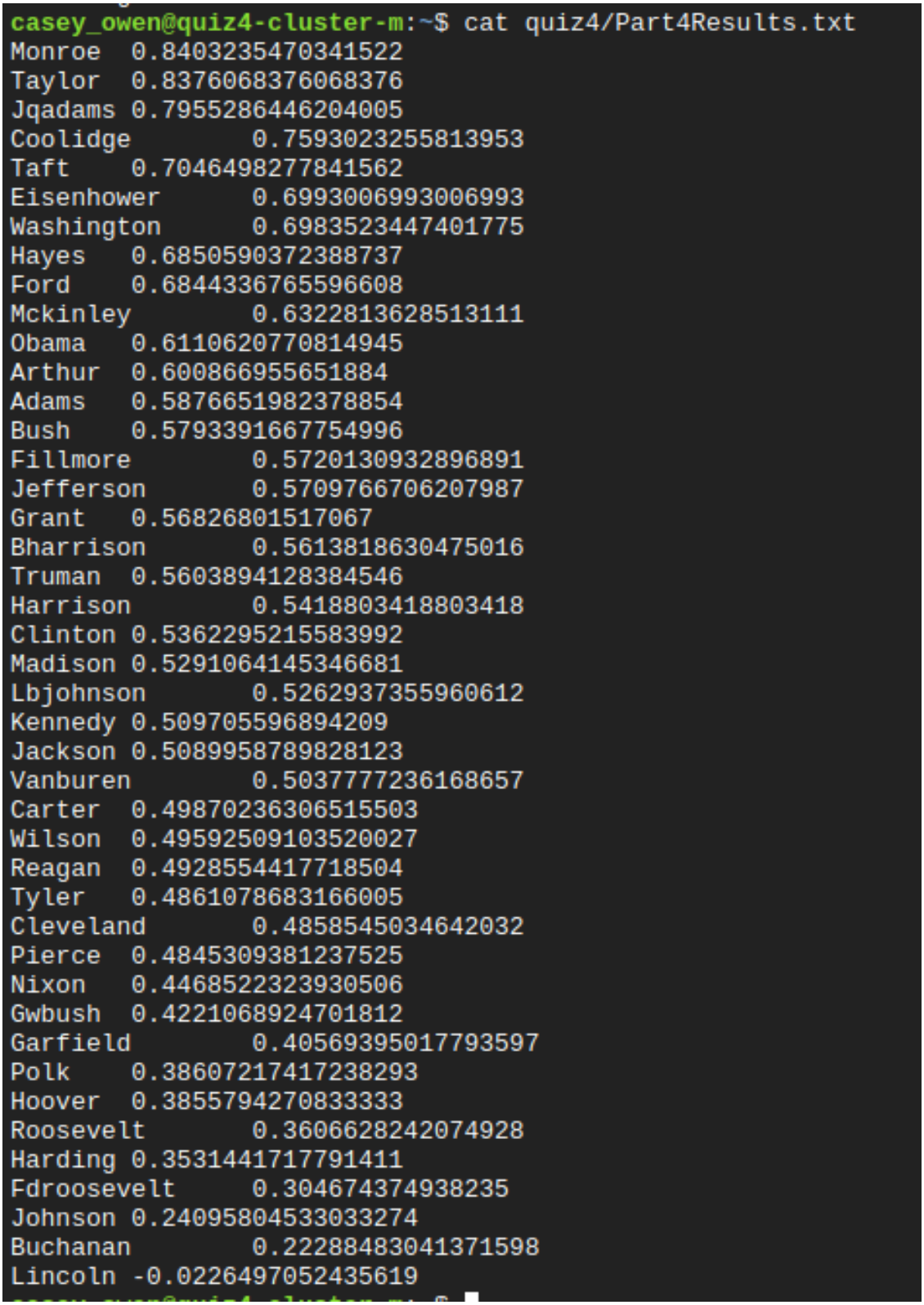
Get Results from hdfs to master:

hadoop fs -get /quiz4/Part4 \quiz4

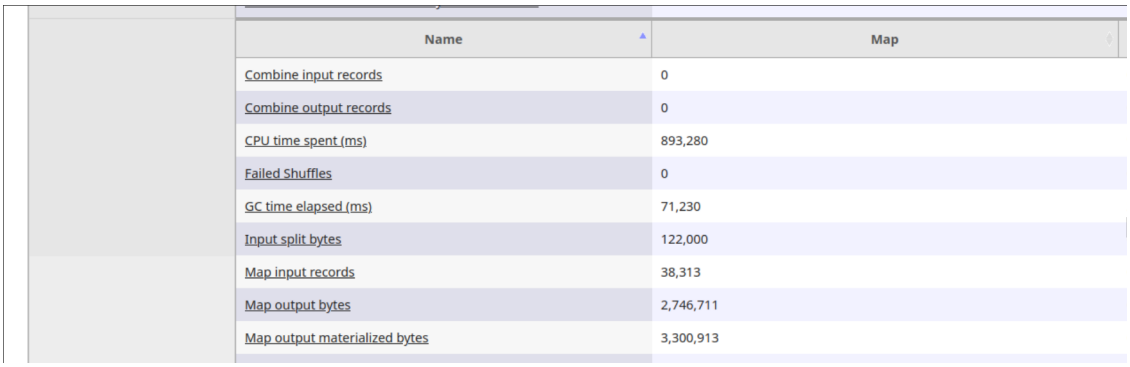
Post-Processing to sort the counts in descending order by the second column, which is valence

cat quiz4/Part4/* | sort -r -n -k 2 > quiz4/Part4Results.txt

Results:



No pattern here between jumps out to me - about either predidential dispoisition or current events at the time of their presidency.



Name	Map
Combine input records	0
Combine output records	0
CPU time spent (ms)	893,280
Failed Shuffles	0
GC time elapsed (ms)	71,230
Input split bytes	122,000
Map input records	38,313
Map output bytes	2,746,711
Map output materialized bytes	3,300,913

Map Wrote 3,300,913 total bytes

4.1 Valence Function (for testing)

[ ]:
```
import requests
import re
import string

def remove_stopwords(stopwords, words):
    list_ = re.sub(r"[^a-zA-Z0-9]", " ", words.lower()).split()
    return [itm for itm in list_ if itm not in stopwords]

def clean_text(stopwords, text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
    text = re.sub('[\d\n]', ' ', text)
    return ' '.join(remove_stopwords(stopwords,  text))

def calc_word_valence(word, afinn_dict):
    if word in afinn_dict:
        return int(afinn_dict[word])
    else:
        return None

def calc_valence(text, afinn_dict):
    '''
    Gets the valence of a line of cleaned text, returned as a list of valences␣
 ↪at each word
    '''
    # At this point they will have been cleaned, so we assume a space seperator
    word_valences = list(map(lambda word: calc_word_valence(word, afinn_dict),␣
 ↪text.split(' ')))
    return list(filter(lambda valence: valence is not None, word_valences))

def valence(text):
```

```
    '''
    Gets the valence of a line of raw text
    '''
    # Using afinn_dict and stopwords as inputs so I don't have to load them␣
↪anew for every line - just once at beginning of mapper
    stopwords_list = requests.get("https://gist.githubusercontent.com/rg089/
↪35e00abf8941d72d419224cfd5b5925d/raw/
↪12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt").content
    stopwords = list(set(stopwords_list.decode().splitlines()))

    afinn = requests.get('https://raw.githubusercontent.com/fnielsen/afinn/
↪master/afinn/data/AFINN-en-165.txt').content.decode().splitlines()
    afinn_dict = dict(map(lambda x: (x.split('\t')), afinn))

    if type(text) != str:
        text = text.decode()
    return calc_valence(clean_text(stopwords, text), afinn_dict)
```

4.2 Part4_mapper_tests.py

Testing a large variety of edge cases using the mapper form of the function, each described in function comment. All tests were passed in the .py file.

I repeated these tests for the form of the function that will be tested by the grader (where valence()'s only input is the text), and also passed all tests

[ ]:
```
import unittest
from Part4_mapper import valence, get_afinn_dict
import dis
import requests

class TestValence(unittest.TestCase):
    def setUp(self):
        self.afinn_dict = get_afinn_dict()
        stopwords_list = requests.get("https://gist.githubusercontent.com/rg089/
↪35e00abf8941d72d419224cfd5b5925d/raw/
↪12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt").content
        self.stopwords = list(set(stopwords_list.splitlines()))

    def test_normal(self):
        '''
        A typical sequence of three words
        '''
        self.assertEqual(valence('yeah winner worst', self.afinn_dict, self.
↪stopwords),[1, 4, -3])

    def test_empty(self):
```

```
        '''
        Empty string input
        '''
        self.assertEqual(valence('', self.afinn_dict, self.stopwords),[])

    def test_nonword(self):
        '''
        Words not in afinn dictionary should be skipped
        '''
        self.assertEqual(valence('qqqqqq', self.afinn_dict, self.stopwords),[])

    def test_quotes(self):
        '''
        Words in quotes should still parse correctly
        '''
        self.assertEqual(valence('"yeah" "winner worst"', self.afinn_dict, self.
↪stopwords),[1, 4, -3])

    def test_seperators(self):
        '''
        Testing that various seperators are removed, and special characters␣
↪ignored
        '''
        self.assertEqual(valence('yeah\twinner\tworst', self.afinn_dict, self.
↪stopwords),[1, 4, -3])
        self.assertEqual(valence('yeah\t\twinner\t\tworst', self.afinn_dict,␣
↪self.stopwords),[1, 4, -3])
        self.assertEqual(valence('yeah\nwinner\nworst\t\n', self.afinn_dict,␣
↪self.stopwords),[1, 4, -3])
        self.assertEqual(valence('yeah! *winner[\n]worst$%^&', self.afinn_dict,␣
↪self.stopwords),[1, 4, -3])

    def test_nonprintable(self):
        '''
        Only nonprintable characters are removed
        '''
        self.assertEqual(valence('\n', self.afinn_dict, self.stopwords),[])
        self.assertEqual(valence('\n*@$%&($\n', self.afinn_dict, self.
↪stopwords),[])

    def ex_function():
        '''
        Function to get bytecode of in below test - clean and true both have␣
↪valences of 2 - no other words in bytecode are present
        '''
        clean = True
```

```
    def test_bytecode_string(self):
        '''
        Bytecode string should interpret the given instructions
        '''
        bc_string = dis.Bytecode(self.ex_function).dis()
        self.assertEqual(valence(bc_string, self.afinn_dict, self.
 ↪stopwords),[2, 2])

    def test_bytestring(self):
        '''
        Byte strings should be decoded first
        '''
        self.assertEqual(valence(b'yeah winner worst', self.afinn_dict, self.
 ↪stopwords),[1, 4, -3])

if __name__ == '__main__':
    unittest.main()
```

4.3 Part4_mapper.py

This valence function is not exactly the same as the form that I changed so it could be used for testing - I refactored the function inputs/outputs slightly so that the stopwords and afinn dictionary did not have to be reloaded on every call to valence(), which happends on every line in the map

[ ]:
```
#!/usr/bin/env python

import sys
from pathlib import Path
import os
import requests
import re
import string

def remove_stopwords(stopwords, words):
    list_ = re.sub(r"[^a-zA-Z0-9]", " ", words.lower()).split()
    return [itm for itm in list_ if itm not in stopwords]

def clean_text(stopwords, text:str):
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub(r'[%s]' % re.escape(string.punctuation), ' ', text)
    text = re.sub(r'[\d\n]', ' ', text)
    return ' '.join(remove_stopwords(stopwords, text))

def get_afinn_dict():
    '''
```

```
    Create a dict from the afinn data, for easier lookup of each word
    '''
    afinn = requests.get('https://raw.githubusercontent.com/fnielsen/afinn/
 ↪master/afinn/data/AFINN-en-165.txt').content.decode().splitlines()
    return dict(map(lambda x: (x.split('\t')), afinn))

def calc_word_valence(word, afinn_dict):
    if word in afinn_dict:
        return int(afinn_dict[word])
    else:
        return None

def calc_valence(text, afinn_dict):
    '''
    Gets the valence of a line of cleaned text, returned as a list of valences␣
 ↪at each word
    '''
    # At this point they will have been cleaned, so we assume a space seperator
    word_valences = list(map(lambda word: calc_word_valence(word, afinn_dict),␣
 ↪text.split(' ')))
    return list(filter(lambda valence: valence is not None, word_valences))

def valence(text, afinn_dict, stopwords):
    '''
    Gets the valence of a line of raw text
    '''
    # Using afinn_dict and stopwords as inputs so I don't have to load them␣
 ↪anew for every line - just once at beginning of mapper
    if type(text) != str:
        text = text.decode()
    return calc_valence(clean_text(stopwords, text), afinn_dict)

def main(argv):
    stopwords_list = requests.get("https://gist.githubusercontent.com/rg089/
 ↪35e00abf8941d72d419224cfd5b5925d/raw/
 ↪12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt").content
    stopwords = list(set(stopwords_list.splitlines()))
    afinn_dict = get_afinn_dict()
    line = sys.stdin.readline()
    filename = Path(os.environ['mapreduce_map_input_file']).stem
    pres = filename.split('_')[0]
    try:
        while line:
            valencelist = valence(line, afinn_dict, stopwords)
            for v in valencelist: print(pres.title() + "\t" + str(v))
            line = sys.stdin.readline()
    except EOFError as error:
```

```
        return None

if __name__ == "__main__":
    main(sys.argv)
```

4.4 Part4_reducer.py

[ ]:
```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_pres = None
current_pres_count = 0
current_pres_valence_sum = 0
pres = 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
    pres, valence = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        valence = int(valence)
    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_pres == pres:
        current_pres_count += 1
        current_pres_valence_sum += valence
    else:
        if current_pres:
            avg_valence = current_pres_valence_sum/current_pres_count
            # write result to STDOUT
            print ('%s\t%s' % (current_pres, avg_valence))
        current_pres_count = 1
        current_pres_valence_sum = valence
        current_pres = pres
```

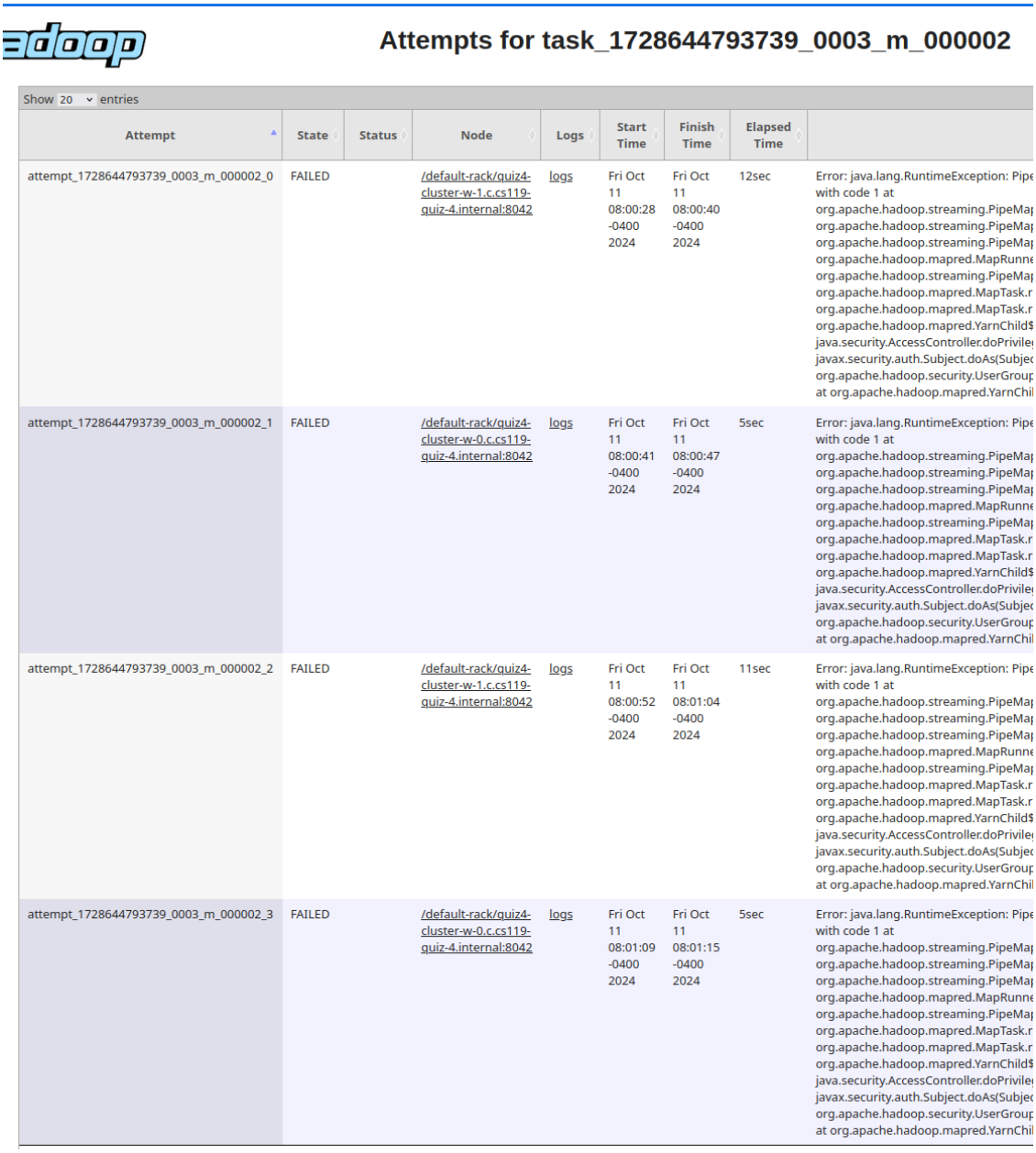
```
# do not forget to output the last word if needed!
# Avoid divide by zero error
if current_pres == pres and current_pres_count != 0:
    avg_valence = current_pres_valence_sum/current_pres_count
    print ('%s\t%s' % (current_pres, avg_valence))
```

5 Part 5 - Hadoop Errors

When using the modified mapper_noll, all 13 Map task attempts that ran failed, across 5 unique tasks (several had retries that also failed).

Attempt Type	Failed	Killed	Successful
Maps	13	2	0

2 more that were in progress received a kill command, which appears to trigger after one task fails four times - in this case it was task_1728644793739_0003_m_000002

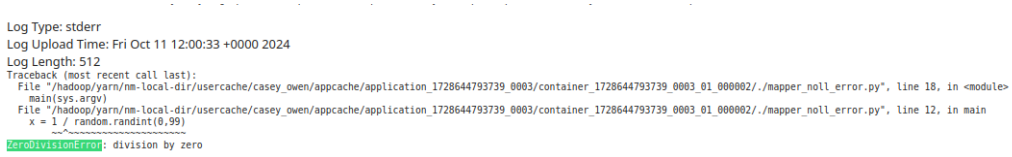


Attempts for task_1728644793739_0003_m_000002

Show 20 entries

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	
attempt_1728644793739_0003_m_000002_0	FAILED		/default-rack/quiz4-cluster-w-1.c.cs119-quiz-4.internal:8042	logs	Fri Oct 11 08:00:28 -0400 2024	Fri Oct 11 08:00:40 -0400 2024	12sec	Error: java.lang.RuntimeException: Pipe with code 1 at org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapRunne org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.YarnChild$ java.security.AccessController.doPrivile javax.security.auth.Subject.doAs(Subjec org.apache.hadoop.security.UserGroup at org.apache.hadoop.mapred.YarnChil
attempt_1728644793739_0003_m_000002_1	FAILED		/default-rack/quiz4-cluster-w-0.c.cs119-quiz-4.internal:8042	logs	Fri Oct 11 08:00:41 -0400 2024	Fri Oct 11 08:00:47 -0400 2024	5sec	Error: java.lang.RuntimeException: Pipe with code 1 at org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapRunne org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.YarnChild$ java.security.AccessController.doPrivile javax.security.auth.Subject.doAs(Subjec org.apache.hadoop.security.UserGroup at org.apache.hadoop.mapred.YarnChil
attempt_1728644793739_0003_m_000002_2	FAILED		/default-rack/quiz4-cluster-w-1.c.cs119-quiz-4.internal:8042	logs	Fri Oct 11 08:00:52 -0400 2024	Fri Oct 11 08:01:04 -0400 2024	11sec	Error: java.lang.RuntimeException: Pipe with code 1 at org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapRunne org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.YarnChild$ java.security.AccessController.doPrivile javax.security.auth.Subject.doAs(Subjec org.apache.hadoop.security.UserGroup at org.apache.hadoop.mapred.YarnChil
attempt_1728644793739_0003_m_000002_3	FAILED		/default-rack/quiz4-cluster-w-0.c.cs119-quiz-4.internal:8042	logs	Fri Oct 11 08:01:09 -0400 2024	Fri Oct 11 08:01:15 -0400 2024	5sec	Error: java.lang.RuntimeException: Pipe with code 1 at org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapRunne org.apache.hadoop.streaming.PipeMap org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.MapTask.r org.apache.hadoop.mapred.YarnChild$ java.security.AccessController.doPrivile javax.security.auth.Subject.doAs(Subjec org.apache.hadoop.security.UserGroup at org.apache.hadoop.mapred.YarnChil

When navigating to the logs of each of these 13 failed tasks, the log files all show the same error printed to stderr, the divide by zero error, 13 total error messages. Here is one example:



```
Log Type: stderr
Log Upload Time: Fri Oct 11 12:00:33 +0000 2024
Log Length: 512
Traceback (most recent call last):
  File "/hadoop/yarn/nm-local-dir/usercache/casey_owen/appcache/application_1728644793739_0003/container_1728644793739_0003_01_000002/./mapper_noll_error.py", line 18, in <module>
    main(sys.argv)
  File "/hadoop/yarn/nm-local-dir/usercache/casey_owen/appcache/application_1728644793739_0003/container_1728644793739_0003_01_000002/./mapper_noll_error.py", line 12, in main
    x = 1 / random.randint(0,99)
        ~~^~~~~~~~~~~~~~~~~~~~~
ZeroDivisionError: division by zero
```

These failed tasks were spread across both worker nodes: /default-rack/quiz4-cluster-w-0.c.cs119-quiz-4.internal:8042 and /default-rack/quiz4-cluster-w-1.c.cs119-quiz-4.internal:8042


Node
/default-rack/quiz4-cluster-w-1.c.cs119-quiz-4.internal:8042
/default-rack/quiz4-cluster-w-0.c.cs119-quiz-4.internal:8042
/default-rack/quiz4-cluster-w-1.c.cs119-quiz-4.internal:8042
/default-rack/quiz4-cluster-w-0.c.cs119-quiz-4.internal:8042

Comparing this to the logs of the successfull version of this task (From Part 2 Question 5) we can see that each task is charged with analyzing anywhere from 500 to 4,800 lines:

Show 20 entries

Attempt	State	Status	
attempt_1728644793739_0002_m_000000_0	SUCCEEDED	Records R/W=3978/1	/defa w-0.c quiz-4
attempt_1728644793739_0002_m_000001_0	SUCCEEDED	Records R/W=3058/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000002_0	SUCCEEDED	Records R/W=3086/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000003_0	SUCCEEDED	Records R/W=2977/1	/defa w-0.c quiz-4
attempt_1728644793739_0002_m_000004_0	SUCCEEDED	Records R/W=2915/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000005_0	SUCCEEDED	Records R/W=3667/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000006_0	SUCCEEDED	Records R/W=4882/1	/defa w-0.c quiz-4
attempt_1728644793739_0002_m_000007_0	SUCCEEDED	Records R/W=3947/1	/defa w-0.c quiz-4
attempt_1728644793739_0002_m_000008_0	SUCCEEDED	Records R/W=1948/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000009_0	SUCCEEDED	Records R/W=657/1	/defa w-1.c quiz-4
attempt_1728644793739_0002_m_000010_0	SUCCEEDED	Records R/W=503/1	/defa w-0.c quiz-4
Attempt	State	Status	Nod

We can verify that records refers to lines of text here, since in this example there were 11 tasks and 35,119 total lines of text across the five books - an average of around 3,000 per task is about right. The chance of failure at each line is 1/100, since this is the chance the randomly generated number is 0. Therefore, even the smallest task has only a .99^500=0.65% chance of completing successfully, and the larger tasks are much more unlikely. It is no suprise to see every one of them fail.