

Analyzing Wikipedia Data   
Using Hadoop

Alejandro Flores   
&   
Krishna Ashok Poddar

ECE 536   
Computer Software Systems

Table of Contents

[1.0 Abstract 3](#_Toc387526072)

[2.0 Introduction 3](#_Toc387526073)

[3.0 Experiment 3](#_Toc387526074)

[3.1 Step1 - Data 4](#_Toc387526075)

[3.2 Step 2-Script(s) Development & Testing 4](#_Toc387526076)

[3.3 Step 3- MapReduce 7](#_Toc387526077)

[4.0 Experiment Results 8](#_Toc387526078)

[5.0 Conclusion 10](#_Toc387526079)

[5.1 Amazon Balance 10](#_Toc387526080)

[6.0 Works Cited 11](#_Toc387526081)

[7.0 Appendix A – Challenge Question 1 Scripts 12](#_Toc387526082)

[7.1 Appendix B – Challenge Question 3 Scripts 15](#_Toc387526083)

[7.2 Appendix C – Challenge Question 1 Output 18](#_Toc387526084)

[7.3 Appendix D – Challenge Question 3 Output 19](#_Toc387526085)

## 1.0 Abstract

In recent years, businesses have been analyzing customer data that today has become crucial to their success. Companies such as Facebook and Netflix have thrived on being able to provide smart recommendations to their customer based on complex algorithms that take into account past customer habits and or knowledge of the customers’ associations.

In this experiment, we attempt to reproduce this systematic approach to analyzing large datasets by making use of the Hadoop MapReduce software framework provided by the Amazon Web Service (AWS). Specifically, we will be looking into extracting relevant data trends from Wikipedia hourly traffic collected between October 1, 2008 and April 30, 2009.

The report will take you through the steps we followed in order to obtain key statistics that could potentially be used to understanding what online users are currently intrigued about.

## 2.0 Introduction

Analyzing large datasets can be somewhat difficult due to the high amount of computing power and scalability requirements. In this experiment, we wanted to make use of the Hadoop MapReduce innovative framework to facilitate the analysis of Wikipedia’s traffic dataset.

To effectively do this, we have established four challenge questions that we hope to answer based on our results.

1. Top 10 trending subjects based on the data?
2. Top 10 least trending subjects based on the data?
3. Top 10 subjects with the most content (size in bytes)?
4. Based on the results in step 1 and 2, attempt to identify significant events happening around the world.

## 3.0 Experiment

In order to the answer to the challenge questions set by this experiment, the following tasks had to be performed:

* *Step 1- Data* 
  + Obtain dataset and upload to Amazon S3 buckets
* *Step 2- Script development and Testing*
* Develop the necessary mapper/reducer script(s) required to answer the four questions.
* Test script(s) within Linux using a small amount of data and make adjustments as needed.
* *Step 3-MapReduce*
* Build a MapReduce cluster using the developed mapper/reducer script(s).

## 3.1 Step1 - Data

In looking for data to analyze, we wanted to use a large enough dataset that could take advantage of the power of the Amazon MapReduce platform. For this reason, we decided to use a 320GBs Wikipedia dataset containing page traffic statistics. The dataset was comprised of 5067 compressed hourly readings for seven months.

As our first step in the experiment we had to create an Amazon EC2 instance and attach it to the Amazon specified volume containing the Wikipedia dataset. Once attached, we uncompressed each of the files onto a separate volume totaling 1TB in size. Figure 1 illustrates a sample of the file contained within the dataset.

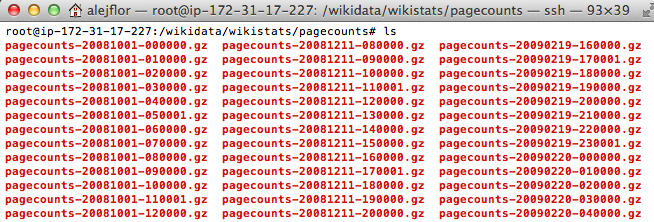


Figure 1: Sample of Wikipedia Dataset

It was here that we realized that that downloading and uploading a terabyte of data from Amazon EC2 onto Amazon S3 buckets was going to present a problem due to our Internet bandwidth limitations. To mitigate this issue, we decided to use a subset of the data by selecting the first two working weeks for each month recorder at 9:00 am; bringing the dataset size down to 13GBs.

After successfully uploading the 13GBs of data on to an S3 bucket, we then focused our attention on developing the key mapper/reducer script(s) that could enable us to answer the questions at hand.

## 3.2 Step 2-Script(s) Development & Testing

The key to developing the proper scripts for this experiment required that we fully understood the format of our data. As seen in figure 2, each line within the file contained four columns, each corresponding to the language indicator, page name, page view count, and size of the Wikipedia page content in bytes.

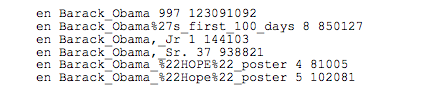


Figure 2: Wikipedia Sample Data

With this in mind, we began developing the mapper script to help find the top 10 trending subjects by using the template provided by Michael G. Noll in his web tutorial ([Michael G. Noll](http://www.michael-noll.com/)). We were particularly inclined in using his scripts because we were not only familiar with it but was also written in python; a relatively easy programming language. Using the base script, we began to make the modifications to correctly read and extract the data in a format that could facilitate our quest. Listed below are a few of the central modifications that were made:

1. Function to turn all words into lower case to ease the filtering of data and avoid the need to specifying lower and upper instances.
   1. line = line.lower()
2. A correct parsing of each column
   1. language, pagename, views, bytess = line.split(' ',3)
3. An if statement to filter out all page names that were not in English
   1. if (language == ‘en’):
4. An if statement to filter out page names that one of our sources indicated as being irrelevant results (Pete Skomoroch)
   1. if ( not (pagename == '404\_error/' or pagename == 'index.php' or pagename == 'search' or pagename == 'favicon.ico' or pagename == '404\_error' or pagename == 'main\_page' or …….)

A complete copy of this mapper script is included in Appendix A.

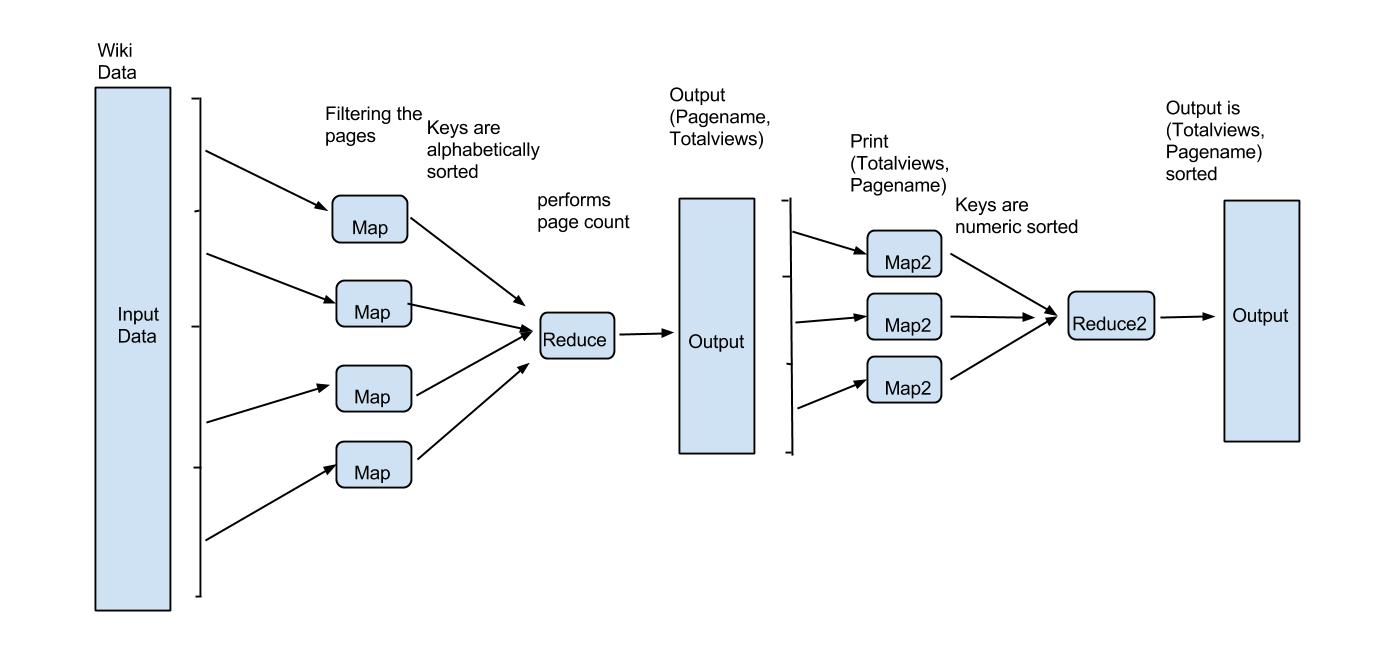
The output of the mapper script was then handed to a single reducer in which we once again began by using Michael G. Noll template. As in the mapper, we modified the reducer to accommodate the four columns in the data file. We also converted the views into an integer to allow us to sum up the different view counts from each of the file being analyzed.

To test our scripts, we copied a file from the dataset onto our local computers and executed the command below:

cat pagecounts-20081006-090000 | ./**mapper.py** | sort -k1,1 | ./**reducer.py**

The command above ran the pagecounts-20081006-090000 file thought our scripts while the “sort -k1,1” simulated the task performed by MapReduce.

In executing our first set of mapper/reducer scripts, we quickly realized that at least two mappers and two reducers were going to be required to help in the sorting process. The idea was to use the first mapper and reducer for most of the work and then use a second dummy mapper to allow a second reducer to sort the data correctly. Figure 3 illustrates our overall design when making use of dual mapper/reducer scripts in an attempt to obtain the top 10 trending subject in the Wikipedia traffic dataset.

  
Figure 3: MapReduce Diagram for Top 10 Trending Subjects

A second test in Linux was conducted utilizing all four scripts in their corresponding order to determine if the correct output was being generated.

cat pagecounts-20081006-090000 | ./**mapper.py** | sort -k1,1 | ./**reducer.py** | ./**mapper2.py** | sort **-n** -k1,1 | ./**reducer2.py**

A close look of the command will make you realize that a –n parameter was added to the original command. The –n parameter was included to conduct proper sorting order from ASCII to numeric. Further in the paper we also talk about the equivalent parameters that were added to the MapReduce.

With a working code on hand, we expected to use the reverse logic to answer the next challenge question of finding the top 10 least trending topics. As we will find in the results sections, this was not necessarily the case.

Finally, we needed to modify our scripts to answer the last challenge questions of determining the top 10 subjects with the larges page content measured in bytes. To do this, we simply included the page size and reorganized our output order to achieve correct sorting. Figure 4 illustrates how we incorporated the size columns and rearranged the fields to obtain the desired sorting order after the second mapper.

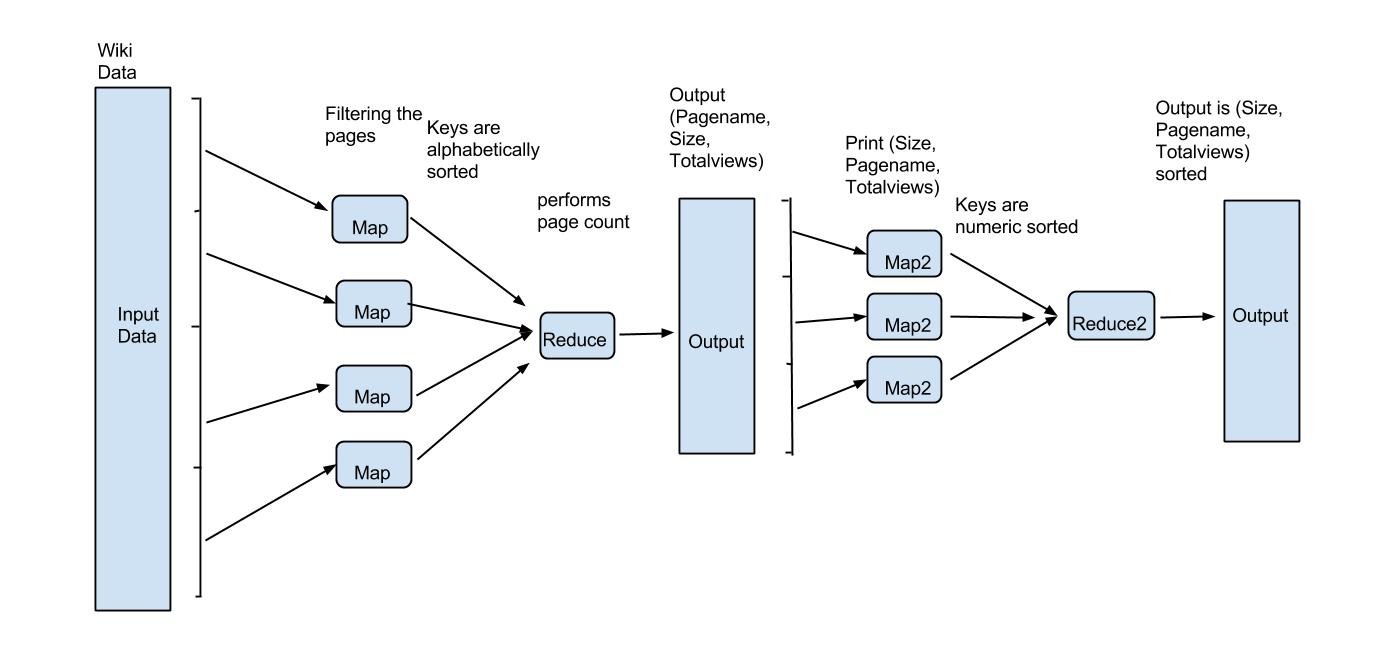


Figure 4: MapReduce Diagram for Top 10 Subject with the Most Content

These scripts were also tested in the Linux environment to address any errors and to ensure proper output before executing them within the MapReduce platform.

## 3.3 Step 3- MapReduce

The final step and the most important part of our experiment now focused on using the script we developed, the complete dataset, and the power of the MapReduce to obtain our final results. In figure 5 we show the bucket layout used for this experiment.

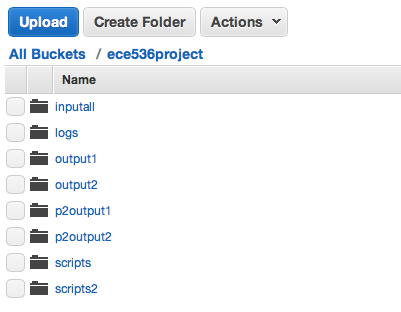


Figure 5: Amazon Buckets Setup

The inputall bucket contained all the Wikipedia traffic data, the scripts buckets the mapper/reducer scripts and the output buckets the output for each stage of the process.

With the buckets containing the necessary elements, it was then time to execute our MapReduce task. As seen in the image below, the first stream was responsible for executing the first set of mapper/reducer scripts producing the output that was then picked up by the second stream.

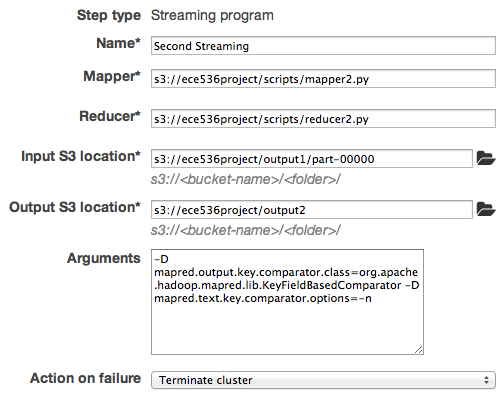
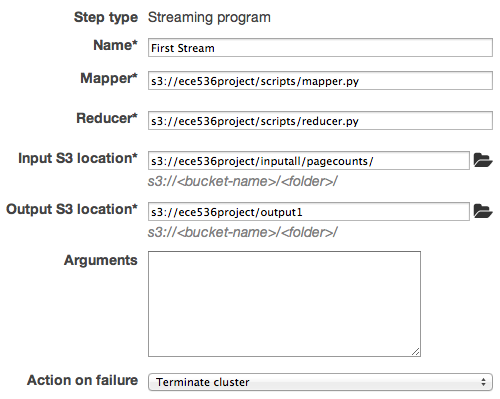


Figure 6: Map Reducer Cluster Stream Components

As previously stated, we needed a way for the MapReduce to sort the key numerically rather than by ASCII; therefore this argument was added to the last stream (Owen O’ Malley):

-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib. KeyFieldBasedComparator -D mapred.text.key.comparator.options=-n

This process was then repeated for the second set of scripts that answered the third challenge question of determining the top 10 subjects with the most content. The eight scripts used in this experiment are listed in their entirety in Appendix A and B.

## 4.0 Experiment Results

In this section, we will go over the results obtained for each of the challenge questions posed as part of the project. Additionally, we will briefly fact check some of the results obtained in our experiment by conducting a quick Google search. A brief table with results will be presented, however a more comprehensive output will be included in Appendix C and D.

*Questions 1: Top 10 trending subjects*

In this first question, we found that the items listed below were the top 10 trending subjects in Wikipedia during the seven months.

|  |  |  |
| --- | --- | --- |
| Position | # views | Subject |
| 1 | 222541 | the\_beatles |
| 2 | 219558 | barack\_obama |
| 3 | 138795 | youtube |
| 4 | 115466 | wikipedia |
| 5 | 106041 | united\_states |
| 6 | 83485 | facebook |
| 7 | 82138 | beatles |
| 8 | 76865 | sex |
| 9 | 71557 | india |
| 10 | 69255 | heroes\_(tv\_series) |

In doing some fact checking, we found that during April 7, 2009, the Beatles released a remastered catalogue of albums that possibly made them the most visited page between October 1, 2008 and April 30, 2009. Furthermore, we surely all recall the November 2008 elections where then candidate Barack Obama was elected the 44th president of the United States. With such overwhelming confirmation of our results, we move to answering the next questions.

*Question 2: Top 10 least trending subjects*

Finding the answer to this question became somewhat difficult to answer. Initially, we had planned to look at the opposite end of the top 10 trending topics to find the answer to this question. Only after we had started is when we realized that attempting to answer this question was somewhat foolish. The amount of data that we were analyzing made it virtually impossible to determine the least trending subjects. As expected, we had hundreds of subjects that had a views count of 1 and therefore the list of top 10 least visited turned into the top 100; if not more.

*Question 3: Top 10 subjects with the most content in bytes*

Question three was generated out of pure curiosity on our part. We wanted to find if there was any correlation between the page size and the page views count. As proven by the table below we can see that this was not the case.

|  |  |  |  |
| --- | --- | --- | --- |
| Position | Size in Bytes | Views | subject |
| 1 | 1150033241 | 1 | united\_states\_presidential\_election,\_2007 |
| 2 | 926588696 | 3 | barack\_obama |
| 3 | 483016351 | 7 | dr.\_seus |
| 4 | 207741368 | 1 | obam\_compend\_manager |
| 5 | 178776748 | 1 | list\_of\_presidents\_of\_the\_united\_state |
| 6 | 176431628 | 1 | watchmen\_(film |
| 7 | 155171090 | 2 | microsoflt |
| 8 | 132084315 | 3 | indi\_script\_records |
| 9 | 127327774 | 1 | the\_new\_living\_heart |
| 10 | 123717217 | 4 | united\_kingdo1 |

Perhaps the Barack Obama subject is the only one that in fact scored high in the top 10 trending subjects and also had the most content in bytes.

*Question 4: Identify significant events happening during this time.*

As our final challenge question, we wanted to identify any significant event happening in the world between October 1, 2008 and April 30, 2009. However, we understood that answering this question could not be based on simply looking at the top 10 trending subjects. For this reason, we looked at a more comprehensive list of trending subjects. Based on this list, we identified 3 significant events that were worth mentioning.

To start with, we considered the election of the first African America president in the United States, Barack Obama, to be the most significant event during this period. Secondly, in the area of technology, Microsoft’s release of its Windows 7 Operating System on October 22, 2009 takes the second spot (Wikipedia 2014). Lastly, the 8 Oscar winning movie Slumdog Millionaire was in theaters on January 2009 (Internet Movies Database).

## 5.0 Conclusion

Making use of the MapReduce platform within Amazon Web Services is a great resource when needing to analyze large datasets. As seen by this experiment, good data can be extrapolated from building simple mapper/reducer scripts in combination with the power of the Hadoop framework.

Without a doubt, this project gave us tremendous exposure to the Amazon Web Services and allowed us to learn more about the Hadoop Framework. We feel that we have learned an important lesson in processing and analyzing big chunks of data.

Most importantly, we feel that we have learned an important skill that might be crucial to know for future employment.

## 5.1 Amazon Balance

We were unable to determine our Amazon balance due to the current billing cycle. However, based on amount of resources we used from EC2 and Elastic MapReduce, we expect our Amazon balance will be approximately less than $50 per member.

## 

## 6.0 Works Cited

Amazon Public Datasets (2009). Wikipedia Page Traffic Statistics. Retrieved from [http://aws.amazon.com/datasets/259](http://aws.amazon.com/datasets/2596)6

Amazon Elastic MapReduce Developer Guide (March 2009). Retrieved from h[ttp://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html](http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html)

[D. Thiebaut](http://cs.smith.edu/dftwiki/index.php/User:Thiebaut) (Nov, 2013).  Running a Python version of Word Count on an AWS cluster.  Retrieved from Tutorial Online Website <http://cs.smith.edu/dftwiki/index.php/Tutorial:_Running_a_Python_version_of_WorkCount_on_an_AWS_cluster>

George Coulouris, Jean Dollimore, Tim Kindberg and Gordon Blair (2011). Distributed Systems: Concepts and Design, 5/E. Addison Wesley.

Internet Movies Database. Slumdog Millionaire. Retrieved from <http://www.imdb.com/title/tt1010048/>

[Michael G. Noll](http://www.michael-noll.com/) (2007). Writing an Hadoop MapReduce Program in Python. Retrieved from <http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>

Owen O'Malley ( Mar, 2009 ). Sorting data numerically. Retrieved from <http://mail-archives.apache.org/mod_mbox/hadoop-common-user/200903.mbox/%3C40BF6593-0B81-487E-B307-F905C436E7A0@apache.org%3E>

[Pete Skomoroch](http://www.linkedin.com/in/peterskomoroch) (2009). Tracking Trends with Hadoop and Hive on EC2. Retrived from h[ttps://github.com/datawrangling/trendingtopics/blob/master/README.textile](https://github.com/datawrangling/trendingtopics/blob/master/README.textile)

Stackoverflow (May, 2013). [Force integer sort in Hadoop on Elastic Map Reduce (AWS).](http://stackoverflow.com/questions/16317435/force-integer-sort-in-hadoop-on-elastic-map-reduce-aws) Retrieved from

[http://stackoverflow.com/questions/16317435/force-integer-sort-in-hadoop-on-elastic-map-reduce-aws](http://stackoverflow.com/questions/13331722/how-to-sort-numerically-in-hadoops-shuffle-sort-phase)

Stackoverflow (Nov, 2012). H[ow to sort numerically in hadoop's shuffle/sort phase.](http://stackoverflow.com/questions/13331722/how-to-sort-numerically-in-hadoops-shuffle-sort-phase) Retrieved from <http://stackoverflow.com/questions/13331722/how-to-sort-numerically-in-hadoops-shuffle-sort-phase>

Tutorials Point. Python Strings. Retrieved from <http://www.tutorialspoint.com/python/python_strings.htm>

Windows 7. In Wikipedia online. Retrieved from <http://en.wikipedia.org/wiki/Windows_7>

## 7.0 Appendix A – Challenge Question 1 Scripts

The next four scripts were used to answer the first challenge questions of the top 10 trending subjects.

#!/usr/bin/env python

#mapper.py

import sys

import re

#Code used to remove undesired subjects

namespace\_titles\_regex = re.compile('(media|special' +

'|talk|user|user\_talk|project|project\_talk|file' +

'|file\_talk|mediaWiki|mediaWiki\_talk|template' +

'|template\_talk|help|help\_talk|category' +

'|category\_talk|portal|wikipedia|wikipedia\_talk)\:(.\*)')

image\_file\_regex = re.compile('(.\*).(jpg|gif|png|JPG|GIF|PNG|txt|ico)')

#input comes from STDIN (standard input)

for line in sys.stdin:

#ADDED FUNCTION TO CONVERT LETTERS TO LOWERCASE

line = line.lower()

# remove leading and trailing whitespace

line = line.strip()

#split the line into words

language, pagename, views, bytess = line.split(' ',3)

if (language == 'en'):

if namespace\_titles\_regex.match(pagename) is None:

if (not (image\_file\_regex.match(pagename))):

#Code used to remove undesired subjects

if ( not (pagename == '404\_error/' or pagename == 'index.php' or pagename == 'search' or pagename == 'favicon.ico' or pagename == '404\_error' or pagename == 'main\_page' or pagename == 'search' or pagename == 'wiki' or pagename == 'special:random' or pagename == 'special:randompage' or pagename == 'index.html' or pagename == 'special:search' or pagename == 'portal:contents' or pagename == 'portal:current\_events')):

# tab-delimited;

print '%s\t%s' % (pagename,views)

#!/usr/bin/env python

#reducer.py

from operator import itemgetter

import sys

current\_pagename = None

current\_views = 0

pagename = 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

pagename, views = line.split('\t', 1)

# convert views (currently a string) to int

try:

views = int(views)

except ValueError:

# views was not a number, so silently

# ignore/discard this line

continue

# this IF-switch only works because Hadoop sorts map output

# by key (here: pagename) before it is passed to the reducer

if current\_pagename == pagename:

current\_views += views

else:

if current\_pagename:

# write result to STDOUT

print '%s\t%s' % (current\_pagename,current\_views)

current\_views = views

current\_pagename = pagename

#do not forget to output the last word if needed!

if current\_pagename == pagename:

print '%s\t%s' % (current\_pagename,current\_views)

#!/usr/bin/env python

#mapper2.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

pagename,views = line.split('\t',1)

print '%s\t%s' % (views, pagename)

#!/usr/bin/env python

#reducer2.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

views, pagename = line.split('\t',1)

print '%s\t%s' % (views, pagename)

## 7.1 Appendix B – Challenge Question 3 Scripts

The next four scripts were used to answer the second challenge questions of the top 10 subject with the most content (in bytes).

#!/usr/bin/env python

#smapper.py

import sys

import re

#Code used to remove undesired subjects

namespace\_titles\_regex = re.compile('(media|special' +

'|talk|user|user\_talk|project|project\_talk|file' +

'|file\_talk|mediaWiki|mediaWiki\_talk|template' +

'|template\_talk|help|help\_talk|category' +

'|category\_talk|portal|wikipedia|wikipedia\_talk)\:(.\*)')

image\_file\_regex = re.compile('(.\*).(jpg|gif|png|JPG|GIF|PNG|txt|ico)')

#input comes from STDIN (standard input)

for line in sys.stdin:

#ADDED FUNCTION TO CONVERT LETTERS TO LOWERCASE

line = line.lower()

# remove leading and trailing whitespace

line = line.strip()

#split the line into words

language, pagename, views, bytess = line.split(' ',3)

if (language == 'en'):

if namespace\_titles\_regex.match(pagename) is None:

if (not (image\_file\_regex.match(pagename))):

#Code used to remove undesired subjects

if ( not (pagename == '404\_error/' or pagename == 'index.php' or pagename == 'search' or pagename == 'favicon.ico' or pagename == '404\_error' or pagename == 'main\_page' or pagename == 'search' or pagename == 'wiki' or pagename == 'special:random' or pagename == 'special:randompage' or pagename == 'index.html' or pagename == 'special:search' or pagename == 'portal:contents' or pagename == 'portal:current\_events')):

# tab-delimited;

print '%s\t%s\t%s' % (pagename,bytess,views)

#!/usr/bin/env python

#sreducer.py

from operator import itemgetter

import sys

current\_pagename = None

current\_views = 0

pagename = 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

pagename, bytess,views = line.split('\t', 2)

# convert views (currently a string) to int

try:

views = int(views)

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: pagename) before it is passed to the reducer

if current\_pagename == pagename:

current\_views += views

else:

if current\_pagename:

# write result to STDOUT

print '%s\t%s\t%s' % (current\_pagename,bytess,current\_views)

current\_views = views

current\_pagename = pagename

#do not forget to output the last word if needed!

if current\_pagename == pagename:

print '%s\t%s\t%s' % (current\_pagename,bytess,current\_views)

#!/usr/bin/env python

#smapper2.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

pagename,bytess,views = line.split('\t',2)

#Change the order to bytess the pagename

print '%s\t%s\t%s' % (bytess, pagename,views)

#!/usr/bin/env python

#sreducer2.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

bytess, pagename,views = line.split('\t',2)

# convert views (currently a string) to int

try:

bytess = int(bytess)

except ValueError:

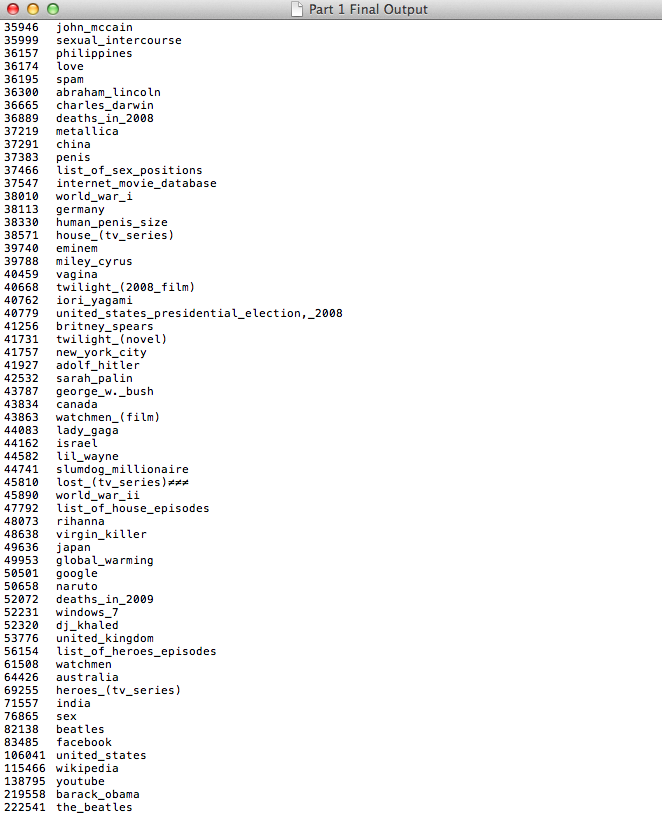
# bytess was not a number, so silently

# ignore/discard this line

continue

print '%d\t%s\t%s' % (bytess, pagename,views)

## 7.2 Appendix C – Challenge Question 1 Output



## 7.3 Appendix D – Challenge Question 3 Output

