# GCD Assignments week 5 - MapReduce

## PART 1: Installation and introduction

1. Create a folder for exercises
   1. E.g.: ..\My Documents\GCD\MapReduce
2. Download Mincemeat.py and example.py from [the intranet](https://portal.fhict.nl/AppliedDataScience/Shared Documents/GCD-Week-5-mincemeat3.4.zip) (for python 3.4)
3. Put all files in the folder

Run the Humpty Dumpty example

1. Open 2 cmd windows
2. in both windows navigate to your MapReduce folder (with example.py in it)
3. In window 1 run the server

python example.py

1. In window 2 run a client

python mincemeat.py -p changeme localhost

The result in the server window:

Sanders-MacBook-Pro:code\_python3.4 sandervanlaar$ python3.4 example.py

{'wall': 1, "King's": 2, 'fall': 1, 'sat': 1, 'again': 1, 'great': 1, 'on': 1, 'horses': 1, 'a': 2, 'the': 2, 'put': 1, 'and': 1, 'men': 1, 'Humpty': 3, 'All': 1, 'had': 1, "Couldn't": 1, 'all': 1, 'Dumpty': 2, 'together': 1}

Links:

The sources of the mincemeat frameworks:

* <https://github.com/ziyuang/mincemeatpy> (Python 3.4)
* <https://github.com/michaelfairley/mincemeatpy> (Python 2.7)

## PART 2: Wordcount

This ‘Hello World’ example in MapReduce is called ‘Wordcount’ The question is: which words are most common in the English language. To calculate this we will use MapReduce on a dataset of a subset of the Gutenberg books (see also [http://www.gutenberg.org](http://www.gutenberg.org/)).

You will use the Humpty Dumpty example as a starting point to program your own wordcount program. If you have a working program, you will experiment with the performance if you allocate more processors/processes.

*Activity 2.1: Introduction*

* Program wordcount for the Gutenberg project (see [Gutenberg.zip](http://www.fhict.nl/docent/downloads/BGDT/Gutenberg.zip)on intranet)
* Develop/test with ‘Gutenberg Small’
* Then run on ‘Gutenberg SF’
* What is the top 10 of words most used?

Tip:

To read through all files in a folder, the following code could be used:

import glob

all\_files = glob.glob('Gutenberg Small/\*.\*')

def file\_contents(file\_name):

f = open(file\_name)

try:

return f.read()

finally:

f.close()

datasource = dict((file\_name, file\_contents(file\_name)) for file\_name in all\_files)

To write the result sorted to file, the following code could be used:

#sort results from high to low

results = sorted(results.items(), key=lambda x: x[1], reverse=True)

#print results

i = open('outfile','w')

i.write(str(results))

i.close()

Add your code and a screenshot of the top 10 to your portfolio.

Activity 2.2: Optimization

* **Optimize** your wordcount program for the Gutenberg project, by omitting the following:
  + Stopwords (use stopwords.py which was downloaded in part 1 of this workshop)
  + One-letter words, like ‘a’
  + Hyphens, periods, commas, other punctuation
* What is the top 10 of words most used?

Add your code and a screenshot of the top 10 to your portfolio.

*#!/usr/bin/env python*

import glob

from functools import reduce

import operator

import mincemeat

def file\_contents(*filename*):

with open(filename, "rb") as f:

return f.read()

def mapfn(*key*, *value*):

from stopwords import allStopWords

import re

words = re.findall(r"[\w']+", value)

for word in words:

word = word.lower()

if (word in allStopWords):

continue

elif len(word) == 1:

continue

else:

yield word, 1

def reducefn(*key*, *values*):

result = 0

for value in values:

result += value

return result

files = glob.glob('small/\*.\*')

data = dict((file\_contents(f).decode('latin1'), f) for f in files)

s = mincemeat.Server()

*# The data source can be any dictionary-like object*

s.datasource = dict(enumerate(data))

s.mapfn = mapfn

s.reducefn = reducefn

results = s.run\_server(*password*="changeme")

results = sorted(results.items(), *key*=operator.itemgetter(1), *reverse*=True)

results = results[:10]

with open('results.txt', 'w') as f:

f.write(str(results))

**Top 10 most used words**

[('us', 2694), ('one', 2593), ('said', 2285), ('upon', 2161), ('now', 1751), ('will', 1626), ('like', 1582), ('time', 1465), ('see', 1153), ('came', 1152)]

Activity 2.3: Performance

* Run the wordcount program again and calculate the execution time for the execution on
  + One processor/process
  + More processors/processes (e.g. 4 clients, open 4 client Terminal/Command windows)
* Give an overview of the execution time for at least 3 different number of processors.

Add a table of your findings to your portfolio.

|  |  |
| --- | --- |
| # of processors | Execution time (s) |
| 1 | 45.66 |
| 2 | 27.64 |
| 3 | 25.68 |
| 4 | 23.92 |

On a octo-core processor, the benefits of multi-threading this problem seem to quickly downgrade with multiple processes.

Activity 2.4 (Optional): Clean the dataset

As you might have noticed, the Gutenberg set is not clean. The set contains other languages than only English and each of the books contains a header with Gutenberg information. Counting these words influences the final result of the word count. Due to the large dataset the actual influence on the final top 10 is marginal, still if the Gutenberg dataset is increased, this might lead to unwanted results.

Clean the dataset before importing or during the MapReduce process to enhance the final word count.

Add your code to your portfolio.