TWITTER ANALYSIS USING MAPREDUCE AND COMPARISON WITH SERIAL EXECUTION

1. Abstract

Big Data is one of the leading streams in Computer Science. The normal database management tools have significantly failed in processing and managing the enormous and complex data. The issue becomes serious with the working of large search engines. The Web is the collection of documents and resources interlinked through dense connections known as Internet. Document Clustering is one of the important aspects of Web to analyze and cluster the documents, or in simple words group the related documents so as to reduce the size of Big Data. [1]

To efficiently retrieve data from large number of documents, we use a Parallel Algorithm paradigm known as MapReduce. The MapReduce application in document clustering will allow us to understand the parallel computation of <key, value> pairs using the Mapper function and correspondingly retrieving data using Reducer algorithm. [2]

Keywords

Big Data, Document Clustering, Parallel Computing, Distributed Memory, MapReduce.

2. Introduction

MapReduce is parallel paradigm which has 2 phases: Map and Reduce which are pipelined phases to give a touch of Parallelism. The input data is processed into independent segments. The segments are passed through functions of Text Mining (e.g. Tokenizing, Stemming, Stop Words Removal, Weight Assignment, etc.).[2]

The segments are tuned into <key, value> pairs which are parallelly processed using Mapper function. The outputs from the Mapper are also: <key, value> pairs known as intermediate pairs which are sorted and grouped based on key.[3]

The Reducer is applied on this sorted and grouped set of pairs of key-values to generate the original pairs of keys and corresponding values.

Input (after pre-	Splitting (K1,	Mapper	Shuffle and	Reducer	Result
• Let us say we have following sentences: • Cancer Dangerou sDisease • Chicken-pox not dangerous • Malaria mosquit o	• {{0: Cancer Dangerous Disease}, {1:Chicken-poxnot dangerous}, {2: Malaria mosquito disease}}	• {cancer: (1), dangerous: (1), disease: (1)} • {chicken-pox: (1), not: (1), dangerous: (1)} • {malaria: (1),	• {cancer: (1)} • {chicken-pox: (1)} • {dangerous: (1, 1)} • {disease: (1,1)} • {malaria: (1)}	• {cancer: 1} • {chicken-pox: 1} • {dangerous: 2} • {disease: 2} • {malaria: 1} • {mosquito: 1}	• {cancer: 1, chicken-pox: 1, dangerous: 2, disease: 2, malaria: 1, mosquito: 1, not: 1}

The process can be observed from this given chart (left to right)

3. Literature Survey

Data clustering algorithms become expensive and slow while dealing with large data repositories. This large volume of data repositories makes analytical operations, retrieval operations and process operations time consuming and difficult. [2]

It is thus inevitable to develop efficient, faster and effective scalable and parallel clustering algorithms. These algorithms run on distributed environments provided by software like Hadoop. Lot of work has been done in MapReduce using Hadoop.

Hadoop provides its own personalised MapReduce functions which operate on data distributed over a network of nodes. However, often people in research fields have tried to override these functions in order to get a better understanding of the algorithm.[2]

Researchers have also tried to modify existing clustering algorithms like K-means and Hierarchical clustering algorithms. These clustering algorithms are executed in parallel and distributed environment where sometimes MapReduce application is put to use for mapping data onto different nodes and reducing the output from each node into final output. [4]

4. Problem Statement

The problem is to handle Big Data for clustering process which involves building of Term Frequency table. Use the Parallel and Distributed Computing paradigm to quickly process this Big Data and build a Term Frequency table as the output. Analysis can be compared with correspondingly processing the same data in serial fashion using a different program.

5. Early References and Study

During the first phase of the project, an Economy Based System for Cluster Scheduling, that is a Utility Driven Cluster Scheduler was aimed for and had to be objectified over Apache Spark which can be implemented over SCALA.

The objective was to design a resource management system or RMS architecture which would have consisted of a server and a client, and then a set of cluster nodes which would have acted across the computing horse power required. Each cluster would have maintained the up to date information of all the nodes through a resource monitoring daemon.

This earlier version of the concept led us to make the current application of Map Reduce for document cluster, since the whole thing was not objectifiable, the whole project was designed and programmed over Python 2.7, and hence the Map Reduce algorithm was successfully implemented.

6. Motivation and Objective

Big Data is the current talk. Document Clustering is another trend in the fields of Machine Learning. Before applying Document Clustering, it is necessary to collect all distinct terms from each document and process them as tokens. [1]

Tokens are then grouped as Term-Frequency table using fast algorithms. One such algorithm is MapReduce – an application of parallel and distributed computing in building the frequencies of each document faster than before. The master node distributes data among worker nodes which process the data in a parallel fashion for each document.

Thus, our objective is to build MapReduce paradigm working on virtual distributed environment using Python and getting output as Term Frequency count for each of the documents. We will use this information to compare it with serial execution of the same data.

7. Methodology

Testing Dataset

We will be using 11 documents having health tweets extracted from Twitter stream of 11 different news channels. Attributes for the data can be various types of diseases in common. However, for our current work, which is to produce Term Frequency count, the set of attributes do not play any role.[5]

Tools

The tool that is used is Python 2. Python allows pre-processing the Text Mining algorithms in easy fashion and allows us to focus more on Parallel Algorithms of Mapper and Reducer Function. Other references include the server-client connection code (which is also written in Python).

Procedure

1. Write a python script to read data (line-by-line) from documents (input file type: .txt). The read data is processed through text mining algorithms. These include:

   a. Tokenizing: Converting the sentences into individual tokens or words.

   b. Stop Word Removal: Stop words are unnecessary words like “and”, “it”, “me”, etc. which do not contribute to properties considered for clustering.

   c. Removing special characters.

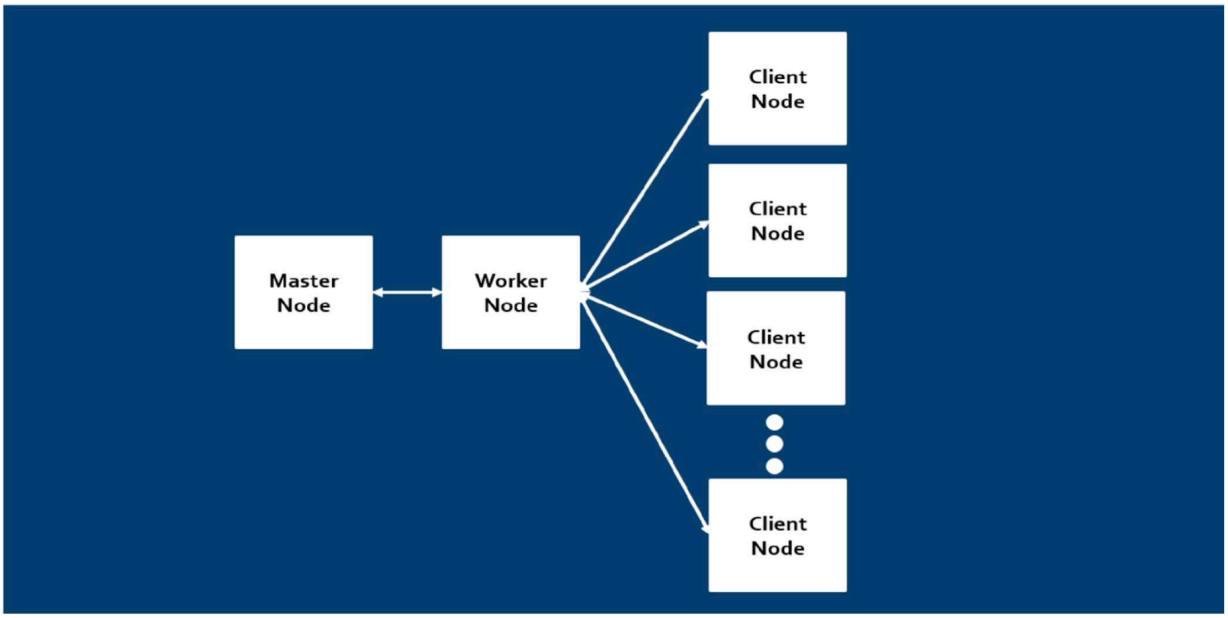
   d. Trimming out garbage characters or special characters from each token. [6]

   The processed tokens are stored (line-by-line) in JSON file corresponding to each document. JSON file stores data in the form of dictionary or as defined by key-value pairs. The keys are the sentence number and value is a list of tokens in the sentence. Here, we treat each tweet in the document as a single sentence.

2. Next step is to write python code for which will allow us to create server-client network in a stand-alone machine, allowing us to utilise a single machine as distributed environment with processors considered as nodes of the environment i.e. creating a virtual distributed environment. The server establishment code reference can be seen in References section. This code establishes connection and creates worker nodes to do the task allocated by master node.

3. Our main job is to now create our own mapper and reducer functions. This program is known as the master node which will have its own MapReduce algorithm. The master node reads the pre-processed data from each JSON file (obtained from step 1), and transfers or passes as argument: the data dictionary, map function, and reduce function to the “server hosting python program” (also known as the worker program).

4. The worker program allocates task to the number of client nodes (as specified in the command while running the code). Client nodes work as specified by the mapper function. Mapper function yields intermediate key value pairs for each document. As soon as the work is completed by the client nodes, the worker program runs the Reducer function to gather all the “intermediate key-value” pairs.



Mapper and Reducer Algorithms

- Mapper (INPUT: <key1, value1>:= <idi, sentencei> pairs) 1. FOR each id:

    a. dcti => Empty Dictionary

    b. FOR each word in sentence:

        i. Append 1 to list: dcti[word]

    c. RETURN dcti

OUTPUT: dcti for each sentence.

- Shuffle and Sort (INPUT: dcti for each sentence)

1. collect => Empty Dictionary
2. FOR each dcti:

    a. FOR each word in dcti:

        i. Add list items from: dcti[word] to the list: collect[word]

3. SORT collect with respect to keys

OUTPUT: collect => dictionary having keys as each token and value as list of ones equal to number of occurrences in entire document.

• Reducer (INPUT: collect (dictionary of intermediate key-value pairs))

1. output => Empty dictionary
2. FOR each word in collect:

    a. output[word] => SUM of the list: collect[word]

3. RETURN output

Therefore, the final output of the program is the dictionary having keys as tokens in the document and value equal to frequency count (number of occurrences) of the token in the document.

8. Code

Refining the tweet set extracted from twitter:

```
import re
from nltk.corpus import stopwords # importing nltk package import
nltk

import json # importing json package
nltk.download('stopwords')
stopWords = set(stopwords.words('english')) # storing existing stopwords in set data structure.

specialChar = "~`!@#$%^&*()_+={}[]|\\:;\'\"<>?/.,"
urlRE = re.compile("http")

fileName = "dataset/d" # location of file for i in
range(1, 12):
    dct = dict()
    c = 0
    with open(fileName + str(i) + ".txt", errors="ignore") as f: # opening files one by one

        for lines in f: # loop for removing special character, whitespaces or other
unnecessary data from the set.
            lines = lines.strip().split('|')
            #  print(lines[2])
            #  lines = lines[2].split()
            wrds = []

            for phrase in lines:
                phrase = phrase.lower()
                phrase = phrase.rstrip("'s")
                chars = list(phrase)
                phrase = ""
                for ch in chars:
                    if ch not in specialChar:
                        phrase += ch
                if urlRE.match(phrase) == None and phrase != "-" and phrase not in stopWords
and phrase not in specialChar:
                    try:
                        phrase.encode('ascii')
                        if phrase != "":
                            wrds.append(phrase)
                    except:
                        continue
            if wrds != []:
                dct.update({c: wrds})
                c += 1
        data = {"data": dct}
```

with open("processed_data/d" + str(i) + "_output.json", 'w') as outfile: # creating a new json file with key-value pairs of keys as numbers and values as stopwords of each tweet.

json.dump(data, outfile)

Serial Code

```
import json # importing necessary directories.
import time # importing directory to calculate time.

timeList = [] # storing time taken for each file in an array.
for i in range(1, 12): # using iterations opening each file.
    with open('d' + str(i) + '_output.json', 'r') as f:
        dataset = json.load(f)
        data = dataset["data"] # taking only values stored in “data” attribute lines =
        len(data) # length of data
        wrds = []
        start = time.time()
        dct = dict()
        for j in range(lines): # iterating all lines
            wrds += data[str(j)] # storing all words in an array
            s = set(wrds)
            for w in s:
                dct[w] = wrds.count(w)
        end = time.time()
        timeList.append(round(end - start, 3))
        print(len(dct))

for i in range(11):
    print("Doc:", i + 1, "==>", timeList[i]) # printing the time taken for each file
```

Master Code

```
import worker # importing necessary directories
import json
import time

# The data source can be any dictionary-like object

#datasource = dict(enumerate(data))

def mapfn(k, v): # function for mapping

    for w in v:

        yield w, 1

    def reducefn(k, vs): # function for reducing

        result = sum(vs)

        return result

    s = worker.Server() # Connecting the Server
```

```
    s.mapfn = mapfn
    s.reducefn = reducefn
    timeList = []
    for i in range(1, 12):
        with open('d'+str(i)+'_output.json', 'r') as f:
            dataset = json.load(f)
            datasource = dataset['data']
            s.datasource = datasource
            start = time.time()
            results = s.run_server(password="scorpion11")
            # print(results) end = time.time()
            timeList.append(round(end-start, 3))
        print(len(results))
        with open('result/d'+str(i)+'_ansOfMapReduce.txt', 'w') as f:
            for keys in sorted(results):
                line = keys+"\t\t\t"+str(results[keys])+"\n"
                f.write(line)

    for i in range(1, 12):
print("Doc:", i, timeList[i-1])
```

Worker code

```
import asynchat
import asyncore
import pickle
import hashlib
import hmac
import logging
import marshal
import optparse
import os
import random
import socket
import sys
import types
import binascii

VERSION = "0.1.2"
```

```
DEFAULT_PORT = 11235

class Protocol(asynchat.async_chat):
   def    init (self, conn=None):
      if conn:
         asynchat.async_chat. init (self, conn)
      else:
         asynchat.async_chat.  init (self)

      self.set_terminator(b"\n")
      self.buffer = []
      self.auth = None
      self.mid_command = False

   def collect_incoming_data(self, data):
      self.buffer.append(data)

   def send_command(self, command, data=None):
      if not b":" in command:
         command += b":"
      if data:
         pdata = pickle.dumps(data)
         command += bytes(str(len(pdata)), 'utf-8')
         logging.debug( "<- %s" % command)
         self.push(command + b"\n" + pdata)
      else:
         logging.debug( "<- %s" % command)
         self.push(command + b"\n")

   def found_terminator(self):
      if not self.auth == b"Done":
         command, data = (b".join(self.buffer).split(b":", 1))

      self.process_unauthed_command(command, data)
   elif not self.mid_command:
      logging.debug("-> %s" % b".join(self.buffer))
      command, length = (b".join(self.buffer)).split(b":", 1) if
      command == b"challenge":
         self.process_command(command, length)
      elif length:
         self.set_terminator(int(length))
         self.mid_command = command
      else:
         self.process_command(command)
   else: # Read the data segment from the previous command
      if not self.auth == b"Done":
         logging.fatal("Recieved pickled data from unauthed source")
         sys.exit(1)
      data = pickle.loads(b".join(self.buffer))
      self.set_terminator(b"\n")
      command = self.mid_command
      self.mid_command = None
      self.process_command(command, data)
```

```
    self.buffer = []

def send_challenge(self):
    self.auth = binascii.hexlify(os.urandom(20))
    self.send_command(b":".join([b"challenge", self.auth]))

def respond_to_challenge(self, command, data):
    mac = hmac.new(self.password, data, hashlib.sha1)
    self.send_command(b":".join([b"auth", binascii.hexlify(mac.digest())]))
    self.post_auth_init()

def verify_auth(self, command, data):
    mac = hmac.new(self.password, self.auth, hashlib.sha1)
    if data == binascii.hexlify(mac.digest()):
        self.auth = b"Done"
        logging.info("Authenticated other end")
    else:
        self.handle_close()

def process_command(self, command, data=None):
    commands = {
        b'challenge': self.respond_to_challenge,
        b'disconnect': lambda x, y: self.handle_close(),
        }

    if command in commands:
        commands[command](command, data)
    else:
        logging.critical("Unknown command received: %s" % (command,))
        self.handle_close()

def process_unauthed_command(self, command, data=None):
    commands = {
        b'challenge': self.respond_to_challenge,

            b'auth': self.verify_auth,
            b'disconnect': lambda x, y: self.handle_close(),
            }

        if command in commands:
            commands[command](command, data)
        else:
            logging.critical("Unknown unauthed command received: %s" % (command,))
            self.handle_close()


class Client(Protocol):
    def    init (self):
        Protocol.  init  (self)
        self.mapfn = self.reducefn = self.collectfn = None

    def conn(self, server, port):
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect((server, port))
        asyncore.loop()
```

```
    def handle_connect(self):
        pass

    def handle_close(self):
        self.close()

    def set_mapfn(self, command, mapfn):
        self.mapfn = types.FunctionType(marshal.loads(mapfn), globals(), 'mapfn')

    def set_collectfn(self, command, collectfn):
        self.collectfn = types.FunctionType(marshal.loads(collectfn), globals(), 'collectfn')

    def set_reducefn(self, command, reducefn):
        self.reducefn = types.FunctionType(marshal.loads(reducefn), globals(), 'reducefn')

    def call_mapfn(self, command, data):
        logging.info("Mapping %s" % str(data[0]))
        results = {}
        for k, v in self.mapfn(data[0], data[1]):
            if k not in results:
                results[k] = []
            results[k].append(v)
        if self.collectfn:
            for k in results:
                results[k] = [self.collectfn(k, results[k])]
        self.send_command(b'mapdone', (data[0], results))

    def call_reducefn(self, command, data):
        logging.info("Reducing %s" % str(data[0]))
        results = self.reducefn(data[0], data[1])
        self.send_command(b'reducedone', (data[0], results))

    def process_command(self, command, data=None):


        commands = {
            b'mapfn': self.set_mapfn,
            b'collectfn': self.set_collectfn,
            b'reducefn': self.set_reducefn,
            b'map': self.call_mapfn,
            b'reduce': self.call_reducefn,
            }

        if command in commands:
            commands[command](command, data)
        else:
            Protocol.process_command(self, command, data)

    def post_auth_init(self):
        if not self.auth:
            self.send_challenge()


class Server(asyncore.dispatcher, object):
```

```
    def    init (self):
        asyncore.dispatcher. init (self)
        self.mapfn = None
        self.reducefn = None
        self.collectfn = None
        self.datasource = None
        self.password = None

    def run_server(self, password=b"", port=DEFAULT_PORT):
        if (type(password) == str):
            password = bytes(password, "utf-8")
        self.password = password
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.bind(("", port))
        self.listen(1)
        try:
            asyncore.loop()
        except:
            self.close_all()
            raise

        return self.taskmanager.results

    # def handle_accepted(self):
    def handle_accepted(self, conn, addr):
        # conn, addr = self.accept()
        sc = ServerChannel(conn, self)
        sc.password = self.password

    def handle_close(self):
        self.close()

    def set_datasource(self, ds):
        self._datasource = ds
        self.taskmanager = TaskManager(self._datasource, self)

        def get_datasource(self):
            return self._datasource

    datasource = property(get_datasource, set_datasource)


class ServerChannel(Protocol):
    def    init (self, conn, server):
        Protocol. init (self, conn)
        self.server = server

        self.start_auth()

    def handle_close(self):
        logging.info("Client disconnected")
        self.close()

    def start_auth(self):
        self.send_challenge()
```

```
    def start_new_task(self):
        command, data = self.server.taskmanager.next_task(self)
        if command == None:
            return
        self.send_command(command, data)

    def map_done(self, command, data):
        self.server.taskmanager.map_done(data)
        self.start_new_task()

    def reduce_done(self, command, data):
        self.server.taskmanager.reduce_done(data)
        self.start_new_task()

    def process_command(self, command, data=None):
        commands = {
            b'mapdone': self.map_done,
            b'reducedone': self.reduce_done,
            }

        if command in commands:
            commands[command](command, data)
        else:
            Protocol.process_command(self, command, data)

    def post_auth_init(self):
        if self.server.mapfn:
            self.send_command(b'mapfn', marshal.dumps(self.server.mapfn.  code        ))
        if self.server.reducefn:
            self.send_command(b'reducefn', marshal.dumps(self.server.reducefn.  code ))
        if self.server.collectfn:
            self.send_command(b'collectfn', marshal.dumps(self.server.collectfn. code ))
        self.start_new_task()

class TaskManager:

    START = 0
    MAPPING = 1
    REDUCING = 2
    FINISHED = 3

    def    init (self, datasource, server):
        self.datasource = datasource
        self.server = server
        self.state = TaskManager.START

    def next_task(self, channel):
        if self.state == TaskManager.START:
            self.map_iter = iter(self.datasource)
            self.working_maps = {}
            self.map_results = {}
            #self.waiting_for_maps = []
            self.state = TaskManager.MAPPING
        if self.state == TaskManager.MAPPING:
```

```
            try:
                map_key = next(self.map_iter)
                map_item = map_key, self.datasource[map_key]
                self.working_maps[map_item[0]] = map_item[1]
                return (b'map', map_item)
            except StopIteration:
                if len(self.working_maps) > 0:
                    key = random.choice(list(self.working_maps.keys()))
                    return (b'map', (key, self.working_maps[key]))
                self.state = TaskManager.REDUCING
                self.reduce_iter = iter(self.map_results.items())
                self.working_reduces = {}
                self.results = {}
        if self.state == TaskManager.REDUCING:
            try:
                reduce_item = next(self.reduce_iter)
                self.working_reduces[reduce_item[0]] =
                reduce_item[1] return (b'reduce', reduce_item)
            except StopIteration:
                if len(self.working_reduces) > 0:
                    key = random.choice(list(self.working_reduces.keys()))
                    return (b'reduce', (key, self.working_reduces[key]))
                self.state = TaskManager.FINISHED
        if self.state == TaskManager.FINISHED:
            self.server.handle_close()
            return (b'disconnect', None)

    def map_done(self, data):
        # Don't use the results if they've already been counted
        if not data[0] in self.working_maps:
            return

         for (key, values) in data[1].items():
             if key not in self.map_results:


                self.map_results[key] = []
            self.map_results[key].extend(values)
        del self.working_maps[data[0]]

    def reduce_done(self, data):
        # Don't use the results if they've already been counted
        if not data[0] in self.working_reduces:
            return

        self.results[data[0]] = data[1]
        del self.working_reduces[data[0]]

def run_client():
    parser = optparse.OptionParser(usage="%prog [options]", version="%%prog %s"%VERSION)
    parser.add_option("-p", "--password", dest="password", default="", help="password")
    parser.add_option("-P", "--port", dest="port", type="int", default=DEFAULT_PORT, help="port")
    parser.add_option("-v", "--verbose", dest="verbose", action="store_true")
    parser.add_option("-V", "--loud", dest="loud", action="store_true")
```

```
   (options, args) = parser.parse_args()

   if options.verbose:
      logging.basicConfig(level=logging.INFO)
   if options.loud:
      logging.basicConfig(level=logging.DEBUG)

   client = Client()
   if (type(options.password) == str):
      options.password = bytes(options.password, "utf-8")
   client.password = options.password
   client.conn(args[0], options.port)


if   name    == ' main ':
   run_client()

if not self.auth == "Done":
            logging.fatal("Recieved pickled data from unauthed source")
            sys.exit(1)
         data = pickle.loads(''.join(self.buffer))
         self.set_terminator("\n")
         command = self.mid_command
         self.mid_command = None
         self.process_command(command, data)
      self.buffer = []

def send_challenge(self):
   self.auth = os.urandom(20).encode("hex")
   self.send_command(":".join(["challenge", self.auth]))

def respond_to_challenge(self, command, data):
   mac = hmac.new(self.password, data, hashlib.sha1)
   self.send_command(":".join(["auth", mac.digest().encode("hex")]))
   self.post_auth_init()

def verify_auth(self, command, data):
   mac = hmac.new(self.password, self.auth, hashlib.sha1)
   if data == mac.digest().encode("hex"):
      self.auth = "Done"
      logging.info("Authenticated other end")
   else:
      self.handle_close()

def process_command(self, command, data=None):
   commands = {
      'challenge': self.respond_to_challenge,
      'disconnect': lambda x, y: self.handle_close(),
      }


   if command in commands:
      commands[command](command, data)
   else:
```

```
            logging.critical("Unknown command received: %s" % (command,))
            self.handle_close()

    def process_unauthed_command(self, command, data=None):
        commands = {
            'challenge': self.respond_to_challenge,
            'auth': self.verify_auth,
            'disconnect': lambda x, y: self.handle_close(),
            }

        if command in commands:
            commands[command](command, data)
        else:
            logging.critical("Unknown unauthed command received: %s" % (command,))
            self.handle_close()


class Client(Protocol): //Client side connection def init (self):

        Protocol. init (self)
        self.mapfn = self.reducefn = self.collectfn = None

    def conn(self, server, port):
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect((server, port))
        asyncore.loop()

    def handle_connect(self):
        pass

    def handle_close(self):
        self.close()
    def set_mapfn(self, command, mapfn):
        self.mapfn = types.FunctionType(marshal.loads(mapfn), globals(), 'mapfn')
    def set_collectfn(self, command, collectfn):
        self.collectfn = types.FunctionType(marshal.loads(collectfn), globals(), 'collectfn')
    def set_reducefn(self, command, reducefn):
        self.reducefn = types.FunctionType(marshal.loads(reducefn), globals(), 'reducefn')
    def call_mapfn(self, command, data):
        logging.info("Mapping %s" % str(data[0]))
        results = {}
        for k, v in self.mapfn(data[0], data[1]):
            if k not in results:
                results[k] = []
            results[k].append(v)
        if self.collectfn:
            for k in results:
                results[k] = [self.collectfn(k, results[k])]
        self.send_command('mapdone', (data[0], results))

    def call_reducefn(self, command, data):
        logging.info("Reducing %s" % str(data[0]))
        results = self.reducefn(data[0], data[1])
```

```
        self.send_command('reducedone', (data[0], results))

    def process_command(self, command, data=None):

            commands = {
                'mapfn': self.set_mapfn,
                'collectfn': self.set_collectfn,
                'reducefn': self.set_reducefn,
                'map': self.call_mapfn,
                'reduce': self.call_reducefn,
                }

            if command in commands:
                commands[command](command, data)
            else:
                Protocol.process_command(self, command, data)

        def post_auth_init(self):
            if not self.auth:
                self.send_challenge()


    class Server(asyncore.dispatcher, object):
        def     init (self):
            self.socket_map = {}
            asyncore.dispatcher. init (self, map=self.socket_map)
            self.mapfn = None
            self.reducefn = None
            self.collectfn = None
            self.datasource = None
            self.password = None

        def run_server(self, password="", port=DEFAULT_PORT):
            self.password = password
            self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
            self.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.bind(("", port))
            self.listen(1)
            try:
                asyncore.loop(map=self.socket_map)
            except:
                self.close_all()
                raise

            return self.taskmanager.results

        def handle_accept(self):
            conn, addr = self.accept()
            sc = ServerChannel(conn, self.socket_map, self)
            sc.password = self.password

        def handle_close(self):
            self.close()
```

```
    def set_datasource(self, ds):
        self._datasource = ds
        self.taskmanager = TaskManager(self._datasource, self)

            def get_datasource(self):
                return self._datasource

    datasource = property(get_datasource, set_datasource)


class ServerChannel(Protocol):
    def     init (self, conn, map, server):
        Protocol. init (self, conn, map=map)
        self.server = server

        self.start_auth()

    def handle_close(self):
        logging.info("Client disconnected")
        self.close()

    def start_auth(self):
        self.send_challenge()

    def start_new_task(self):
        command, data = self.server.taskmanager.next_task(self)
        if command == None:
            return
        self.send_command(command, data)
    def map_done(self, command, data):
        self.server.taskmanager.map_done(data)
        self.start_new_task()
    def reduce_done(self, command, data):
        self.server.taskmanager.reduce_done(data)
        self.start_new_task()
   def process_command(self, command, data=None):
        commands = {
            'mapdone': self.map_done,
            'reducedone': self.reduce_done,
            }
        if command in commands:
            commands[command](command, data)

        else:
            Protocol.process_command(self, command, data)
    def post_auth_init(self):
        if self.server.mapfn:
            self.send_command('mapfn', marshal.dumps(self.server.mapfn.func_code))
        if self.server.reducefn:
            self.send_command('reducefn', marshal.dumps(self.server.reducefn.func_code))
        if self.server.collectfn:
            self.send_command('collectfn', marshal.dumps(self.server.collectfn.func_code))
        self.start_new_task()
```

```
class TaskManager:
    START = 0
    MAPPING = 1
    REDUCING = 2
    FINISHED = 3
    def    init (self, datasource, server):
        self.datasource = datasource
        self.server = server
        self.state = TaskManager.START

    def next_task(self, channel):
        if self.state == TaskManager.START:
            self.map_iter = iter(self.datasource)
            self.working_maps = {}
            self.map_results = {}
            #self.waiting_for_maps = []
            self.state = TaskManager.MAPPING
        if self.state == TaskManager.MAPPING:
            try:
                map_key = self.map_iter.next()
                map_item   =   map_key,   self.datasource[map_key]
                self.working_maps[map_item[0]]    =    map_item[1]
                return ('map', map_item)
            except StopIteration:
                if len(self.working_maps) > 0:
                    key = random.choice(self.working_maps.keys())
                    return ('map', (key, self.working_maps[key]))
                self.state = TaskManager.REDUCING
                self.reduce_iter = self.map_results.iteritems()
                self.working_reduces = {}
                self.results = {}
        if self.state == TaskManager.REDUCING:
            try:
                reduce_item = self.reduce_iter.next()
                self.working_reduces[reduce_item[0]] = reduce_item[1]
                return ('reduce', reduce_item)

            except StopIteration:
                if len(self.working_reduces) > 0:
                    key = random.choice(self.working_reduces.keys())
                    return ('reduce', (key, self.working_reduces[key]))
                self.state = TaskManager.FINISHED
        if self.state == TaskManager.FINISHED:
            self.server.handle_close()
            return ('disconnect', None)
    def map_done(self, data):
        # Don't use the results if they've already been counted if
        not data[0] in self.working_maps:
            return
        for (key, values) in data[1].iteritems(): if
            key not in self.map_results:
                self.map_results[key] = []
            self.map_results[key].extend(values)
```

```
        del self.working_maps[data[0]]
    def reduce_done(self, data):
        # Don't use the results if they've already been counted if
        not data[0] in self.working_reduces:
            return self.results[data[0]] =
        data[1]
        del self.working_reduces[data[0]]
def run_client():
    parser = optparse.OptionParser(usage="%prog [options]", version="%%prog %s"%VERSION)
    parser.add_option("-p", "--password", dest="password", default="", help="password")
    parser.add_option("-P", "--port", dest="port", type="int", default=DEFAULT_PORT,
help="port")
    parser.add_option("-v", "--verbose", dest="verbose", action="store_true")
    parser.add_option("-V", "--loud", dest="loud", action="store_true")

    (options, args) = parser.parse_args()

    if options.verbose:
        logging.basicConfig(level=logging.INFO)
    if options.loud:
        logging.basicConfig(level=logging.DEBUG)

    client = Client()
    client.password = options.password
    client.conn(args[0], options.port)

if    name    == ' main ':
    run_client()
```

9. Result

As results we will be tabulating the time for formulating term frequency table using both MapReduce and Serial execution.

Document Id	Number of Terms	Time for execution in Serial Fashion (sec)	Time for execution in MapReduce (sec)
11	3081	0.667	1.138
01	4005	1.235	1.741
05	4033	1.032	0.899
04	4293	2.011	1.258
08	4307	1.654	1.845
06	4836	2.52	1.485
09	5761	2.941	1.446
02	6431	2.947	1.549
03	7549	4.591	1.278
07	8173	5.546	1.397
10	8594	7.42	1.712

**Results may vary based on the device specification.*

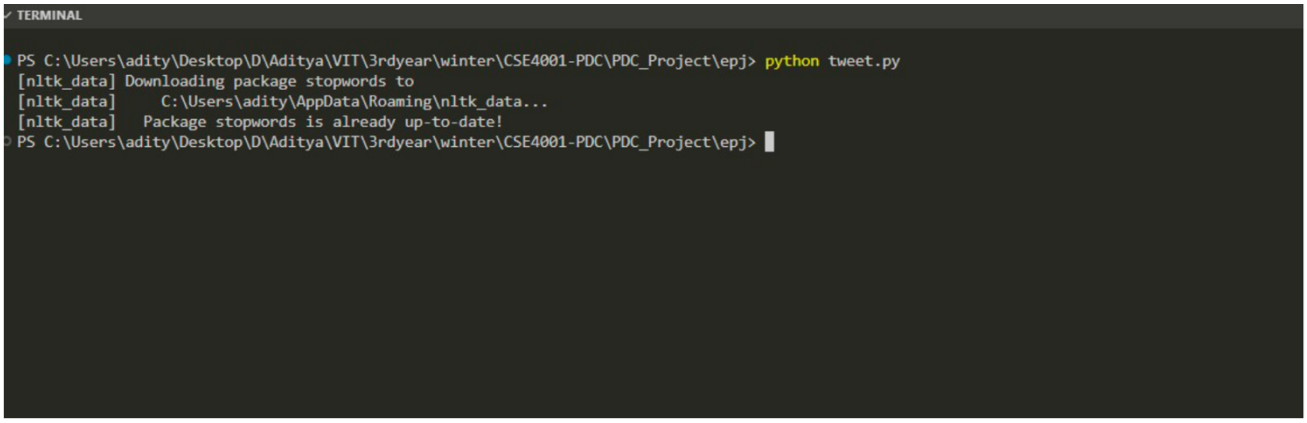
10. Analysis of Result

- We can see that for small number of terms (distinct set of words/phrases) in documents, the time for serial execution is either better or comparable to distributed execution through MapReduce.
- We assume that this happens because serial fashion does not involve sending and receiving of data between different nodes, while for small data this time becomes effectively visible in parallel execution.
- While in cases of document id: 3, 7, and 10 we can clearly see that as the distinct number of tokens increases, serial time increases linearly, while parallel execution time is much less as compared to the former.
- Even though document size has increased, after each node holds the data, the computation time is parallelly divided and hence it stays much lower than the execution time through serial fashion.

12. Output

```
d1.txt
585978391360221184|Thu Apr 09 01:31:50 +0000 2015|Breast cancer risk test devised http://bbc.in/1CimpJF
585947808772960257|Wed Apr 08 23:30:18 +0000 2015|GP workload harming care - BMA poll http://bbc.in/1ChTBRv
585947807816650752|Wed Apr 08 23:30:18 +0000 2015|Short people's 'heart risk greater' http://bbc.in/1ChTANp
585866060991078401|Wed Apr 08 18:05:28 +0000 2015|New approach against HIV 'promising' http://bbc.in/1E6jAjt
585794106170839041|Wed Apr 08 13:19:33 +0000 2015|Coalition 'undermined NHS' - doctors http://bbc.in/1CnLwK7
585733482413891584|Wed Apr 08 09:18:39 +0000 2015|Review of case against NHS manager http://bbc.in/1Ffj6ci
585733481608646657|Wed Apr 08 09:18:39 +0000 2015|VIDEO: 'All day is empty, what am I going to do?' http://bbc.in/1N7wSSz
585701601131765761|Wed Apr 08 07:11:58 +0000 2015|VIDEO: 'Overhaul needed' for end-of-life care http://bbc.in/1CmrRu3
585620828110397440|Wed Apr 08 01:51:00 +0000 2015|Care for dying 'needs overhaul' http://bbc.in/1FdSGrl
585437294120677376|Tue Apr 07 13:41:42 +0000 2015|VIDEO: NHS: Labour and Tory key policies http://bbc.in/1Ci5eqD
585437293399252992|Tue Apr 07 13:41:42 +0000 2015|Have GP services got worse? http://bbc.in/1Ci5c22
585376127931129857|Tue Apr 07 09:38:39 +0000 2015|A&amp;amp;E waiting hits new worst level http://bbc.in/1Fa4Xgz
585283574888206336|Tue Apr 07 03:30:52 +0000 2015|Parties row over GP opening hours http://bbc.in/1CfcVOb
585231549819330560|Tue Apr 07 00:04:09 +0000 2015|Why strenuous runs may not be so bad after all http://bbc.in/1Ceq0Y7
584985578074050560|Mon Apr 06 07:46:44 +0000 2015|VIDEO: Health surcharge for non-EU patients http://bbc.in/1C5M1bk
584985576962592768|Mon Apr 06 07:46:44 +0000 2015|VIDEO: Skin cancer spike 'from 60s holidays' http://bbc.in/1C9Gy3o
584901557218480129|Mon Apr 06 02:12:52 +0000 2015|80,000 'might die' in future outbreak http://bbc.in/1F3nif2
584893680747585537|Mon Apr 06 01:41:34 +0000 2015|Skin cancer 'linked to holiday boom' http://bbc.in/1Pb4Xjb
584140478720811008|Fri Apr 03 23:48:37 +0000 2015|Public 'back tax rises to fund NHS' http://bbc.in/1ETauYE
584140477718339584|Fri Apr 03 23:48:37 +0000 2015|VIDEO: Welcome to the designer asylum http://bbc.in/1CD3Dip
583959459589824513|Fri Apr 03 11:49:19 +0000 2015|VIDEO: Why are we having less sex? http://bbc.in/19RKqiY
583837541373059073|Fri Apr 03 03:44:51 +0000 2015|Five ideas to transform the NHS http://bbc.in/1BTQU7k
583784740064337920|Fri Apr 03 00:15:02 +0000 2015|Personal cancer vaccines 'exciting' http://bbc.in/1ENwsMD
583770728090112000|Thu Apr 02 23:19:22 +0000 2015|Child heart surgery deaths 'halved' http://bbc.in/1BT4s2U
```

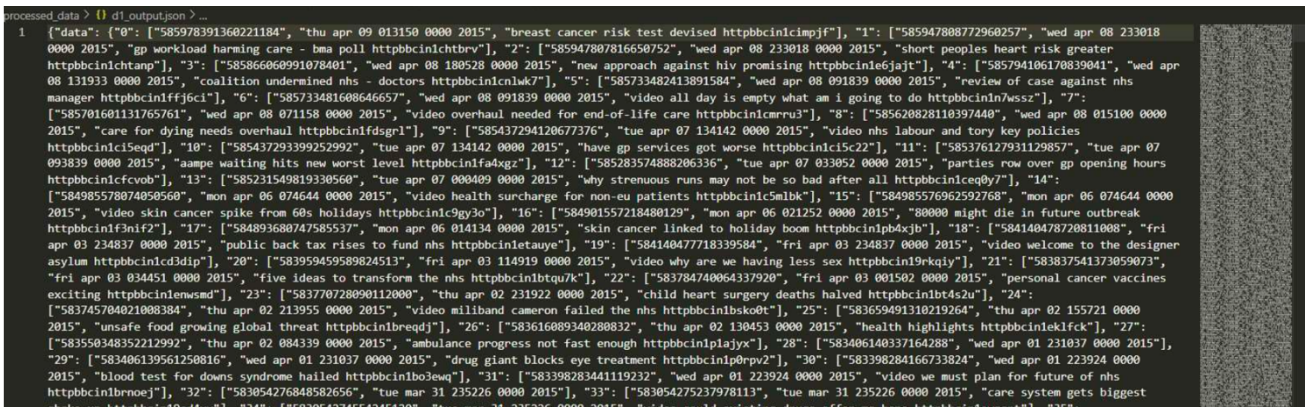
Initial Sample data (d1.txt)



```
TERMINAL

PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python tweet.py
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\adity\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj>
```

Running Tweet.py

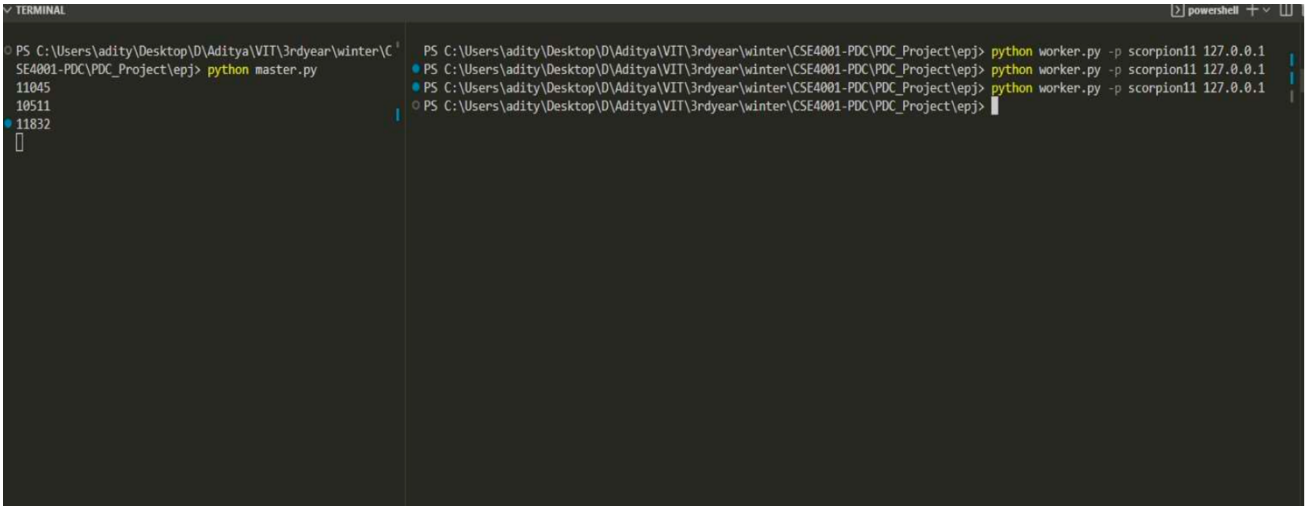


```
processed_data > {} d1_output.json > ...
1  {"data": {"0": ["585978391360221184", "thu apr 09 013150 0000 2015", "breast cancer risk test devised httpbbcin1cimpjf"], "1": ["585947808772960257", "wed apr 08 233018
   0000 2015", "gp workload harming care - bma poll httpbbcin1chtbrv"], "2": ["585947807816650752", "wed apr 08 233018 0000 2015", "short peoples heart risk greater
   httpbbcin1chtanp"], "3": ["585866060991078401", "wed apr 08 180528 0000 2015", "new approach against hiv promising httpbbcin1e6jajt"], "4": ["585794106170839041", "wed apr
   08 131933 0000 2015", "coalition undermined nhs - doctors httpbbcin1cnlwk7"], "5": ["585733482413891584", "wed apr 08 091839 0000 2015", "review of case against nhs
   manager httpbbcin1ffj6ci"], "6": ["585733481608646657", "wed apr 08 091839 0000 2015", "video all day is empty what am i going to do httpbbcin1n7wssz"], "7":
   ["585701601131765761", "wed apr 08 071158 0000 2015", "video overhaul needed for end-of-life care httpbbcin1cmrru3"], "8": ["585620828110397440", "wed apr 08 015100 0000
   2015", "care for dying needs overhaul httpbbcin1fdsgrl"], "9": ["585437294120677376", "tue apr 07 134142 0000 2015", "video nhs labour and tory key policies
   httpbbcin1ci5eqd"], "10": ["585437293399252992", "tue apr 07 134142 0000 2015", "have gp services got worse httpbbcin1ci5c22"], "11": ["585376127931129857", "tue apr 07
   093839 0000 2015", "aampe waiting hits new worst level httpbbcin1fa4xgz"], "12": ["585283574888206336", "tue apr 07 033052 0000 2015", "parties row over gp opening hours
   httpbbcin1cfcvob"], "13": ["585231549819330560", "tue apr 07 000409 0000 2015", "why strenuous runs may not be so bad after all httpbbcin1ceq0y7"], "14":
   ["584985578074050560", "mon apr 06 074644 0000 2015", "video health surcharge for non-eu patients httpbbcin1c5mlbk"], "15": ["584985576962592768", "mon apr 06 074644 0000
   2015", "video skin cancer spike from 60s holidays httpbbcin1c9gy3o"], "16": ["584901557218480129", "mon apr 06 021252 0000 2015", "80000 might die in future outbreak
   httpbbcin1f3nif2"], "17": ["584893680747585537", "mon apr 06 014134 0000 2015", "skin cancer linked to holiday boom httpbbcin1pb4xjb"], "18": ["584140478720811008", "fri
   apr 03 234837 0000 2015", "public back tax rises to fund nhs httpbbcin1etauye"], "19": ["584140477718339584", "fri apr 03 234837 0000 2015", "video welcome to the designer
   asylum httpbbcin1cd3dip"], "20": ["583959459589824513", "fri apr 03 114919 0000 2015", "video why are we having less sex httpbbcin19rkqiy"], "21": ["583837541373059073",
   "fri apr 03 034451 0000 2015", "five ideas to transform the nhs httpbbcin1btqu7k"], "22": ["583784740064337920", "fri apr 03 001502 0000 2015", "personal cancer vaccines
   exciting httpbbcin1enwsmd"], "23": ["583770728090112000", "thu apr 02 231922 0000 2015", "child heart surgery deaths halved httpbbcin1bt4s2u"], "24":
   ["583745704021008384", "thu apr 02 213955 0000 2015", "video miliband cameron failed the nhs httpbbcin1bsko0t"], "25": ["583659491310219264", "thu apr 02 155721 0000
   2015", "unsafe food growing global threat httpbbcin1breqdj"], "26": ["583616089340280832", "thu apr 02 130453 0000 2015", "health highlights httpbbcin1eklfck"], "27":
   ["583550348352212992", "thu apr 02 084339 0000 2015", "ambulance progress not fast enough httpbbcin1p1ajyx"], "28": ["583406140337164288", "wed apr 01 231037 0000 2015"],
   "29": ["583406139561250816", "wed apr 01 231037 0000 2015", "drug giant blocks eye treatment httpbbcin1p0rpv2"], "30": ["583398284166733824", "wed apr 01 223924 0000
   2015", "blood test for downs syndrome hailed httpbbcin1bo3ewq"], "31": ["583398283441119232", "wed apr 01 223924 0000 2015", "video we must plan for future of nhs
   httpbbcin1brnoej"], "32": ["583054276848582656", "tue mar 31 235226 0000 2015"], "33": ["583054275237978113", "tue mar 31 235226 0000 2015", "care system gets biggest
```

Processed data in JSON format after running tweet.py

```
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python serial.py
11045
10511
11832
9110
5820
7970
22392
10282
12206
8421
11779
Doc: 1 8.08
Doc: 2 3.948
Doc: 3 3.050
Doc: 4 2.904
Doc: 5 1.947
Doc: 6 4.775
Doc: 7 5.579
Doc: 8 3.982
Doc: 9 3.833
Doc: 10 15.211
Doc: 11 4.271
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj>
```

Running Sequential code (Serial.py)



Running Parallel Code:
When all documents are done the master code displays the counted frequencies followed by time taken by each
document.
Running Master thread in one terminal and the worker process separately in another terminal.

```
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python master.py
11045
10511
11832
9110
5820
7970
22392
10282
12206
8421
11779
Doc: 1 5.08
Doc: 2 2.768
Doc: 3 2.253
Doc: 4 2.281
Doc: 5 1.473
Doc: 6 2.103
Doc: 7 4.085
Doc: 8 2.451
Doc: 9 2.66
Doc: 10 10.211
Doc: 11 2.609
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj>
```

Final master.py output with the time stamps.

```
powershell
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
```

Worker.py

```
TERMINAL
powershell
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python master.py
11045
10511
11832
9110
5820
7970
22392
10282
12206
8421
11779
Doc: 1 5.08
Doc: 2 2.768
Doc: 3 2.253
Doc: 4 2.281
Doc: 5 1.473
Doc: 6 2.103
Doc: 7 4.085
Doc: 8 2.451
Doc: 9 2.66
Doc: 10 10.211
Doc: 11 2.609
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj>

PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj> python worker.py -p scorpion11 127.0.0.1
PS C:\Users\adity\Desktop\D\Aditya\VIT\3rdyear\winter\CSE4001-PDC\PDC_Project\epj>
```

Master and worker terminals together.

13. References

[1] Base paper: Document Clustering with Map Reduce using Hadoop Framework, Satish Muppidi, M. Ramakrishna Murty, International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 3 Issue: 1, ISSN: 2321-8169 409 - 413.

[2] https://archive.ics.uci.edu/ml/datasets/Health+News+in+Twitter

[3] https://github.com/michaelfairley/mincemeatpy

[4] https://github.com/jpmec/shepherdpy

[5] An analysis of MapReduce efficiency in document clustering using parallel K-means algorithm, Tanvir Habib Sardar, Zahid Ansari, Future Computing and Informatics Journal xx (2018) 1-10.

[6] Sentiment Analysis Using Machine Learning and Deep Learning on Covid 19 Vaccine Twitter Data with Hadoop MapReduce, S Kul, A Sayar.

[7] Sentiment analysis on Twitter data using Apache Hadoop and performance evaluation on Hadoop MapReduce and Apache Spark, K Garg, D Kaur.