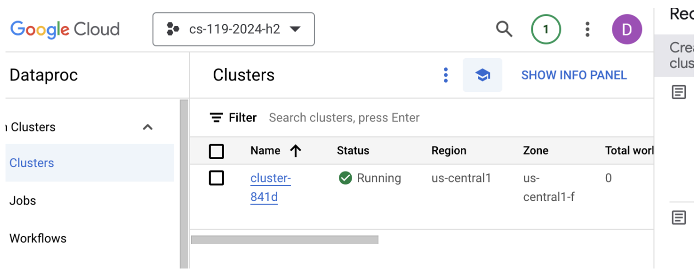
CS119 QUIZ 4 - Darcy Corson

PART 1 - ANALYZING SERVER LOGS

1. See below for confirmation that the cluster was successfully acquired.

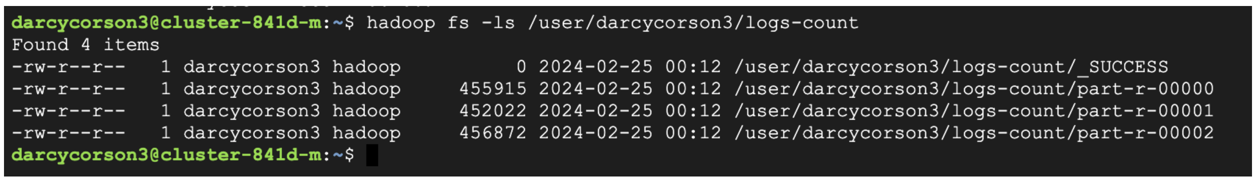


Name	Status	Region	Zone	Total worl
cluster-841d	Running	us-central1	us-central1-f	0

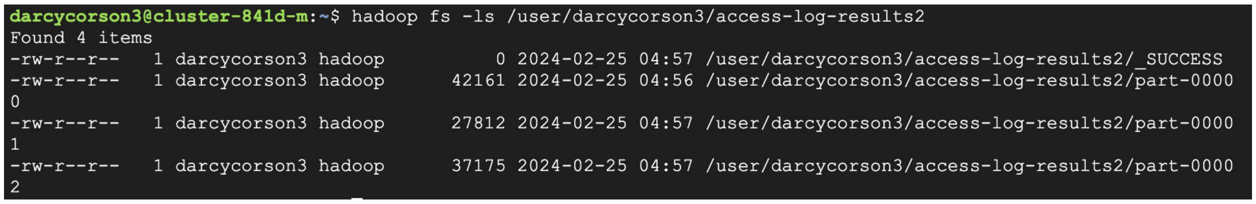
2. To load the Apache web server log data into the master node, the data was downloaded directly onto the master node of the Hadoop cluster. This was accomplished by running 'wget https://raw.githubusercontent.com/singhj/big-data-repo/main/datasets/access.log'. This command acquired the access.log file from a public repository and saved it locally to the master node.

   Then, this data was moved to the HDFS. To organize the data within HDFS and facilitate easy access to it, a new directory in HDFS named '/logs' was created using the command 'hadoop fs -mkdir /logs'. The access.log file from the local file system of the master node was moved into this newly created HDFS directory. This transfer was accomplished using the 'hadoop fs -put access.log /logs/ command'.

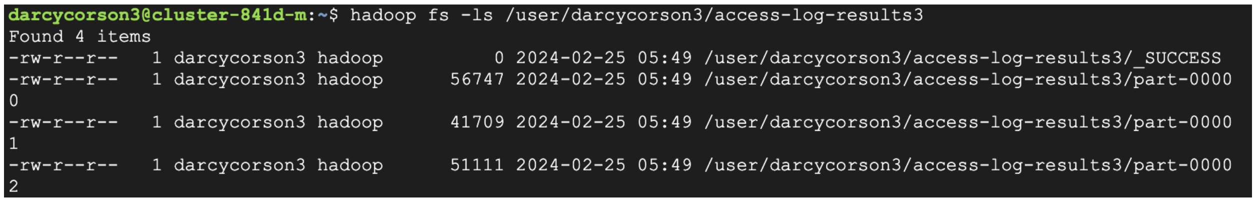
3. Confirmation that the Hadoop installation was valid was acquired by performing the following tests: the example hadoop wordcount, Hadoop streaming with Michael Noll's mapper and the default aggregate reducer, and Hadoop streaming with Michael Noll's mapper and reducer. The results of these tests are below.



```
darcycorson3@cluster-841d-m:~$ hadoop fs -ls /user/darcycorson3/logs-count
Found 4 items
-rw-r--r--   1 darcycorson3 hadoop          0 2024-02-25 00:12 /user/darcycorson3/logs-count/_SUCCESS
-rw-r--r--   1 darcycorson3 hadoop     455915 2024-02-25 00:12 /user/darcycorson3/logs-count/part-r-00000
-rw-r--r--   1 darcycorson3 hadoop     452022 2024-02-25 00:12 /user/darcycorson3/logs-count/part-r-00001
-rw-r--r--   1 darcycorson3 hadoop     456872 2024-02-25 00:12 /user/darcycorson3/logs-count/part-r-00002
darcycorson3@cluster-841d-m:~$
```



```
darcycorson3@cluster-841d-m:~$ hadoop fs -ls /user/darcycorson3/access-log-results2
Found 4 items
-rw-r--r--   1 darcycorson3 hadoop          0 2024-02-25 04:57 /user/darcycorson3/access-log-results2/_SUCCESS
-rw-r--r--   1 darcycorson3 hadoop      42161 2024-02-25 04:56 /user/darcycorson3/access-log-results2/part-0000
0
-rw-r--r--   1 darcycorson3 hadoop      27812 2024-02-25 04:57 /user/darcycorson3/access-log-results2/part-0000
1
-rw-r--r--   1 darcycorson3 hadoop      37175 2024-02-25 04:57 /user/darcycorson3/access-log-results2/part-0000
2
```



```
darcycorson3@cluster-841d-m:~$ hadoop fs -ls /user/darcycorson3/access-log-results3
Found 4 items
-rw-r--r--   1 darcycorson3 hadoop          0 2024-02-25 05:49 /user/darcycorson3/access-log-results3/_SUCCESS
-rw-r--r--   1 darcycorson3 hadoop      56747 2024-02-25 05:49 /user/darcycorson3/access-log-results3/part-0000
0
-rw-r--r--   1 darcycorson3 hadoop      41709 2024-02-25 05:49 /user/darcycorson3/access-log-results3/part-0000
1
-rw-r--r--   1 darcycorson3 hadoop      51111 2024-02-25 05:49 /user/darcycorson3/access-log-results3/part-0000
2
```

4. This Hadoop streaming job involves two main components, a mapper (ip_error_mapper.py) and a reducer (top_ip_reducer.py), along with a command to run the job on a Hadoop cluster. The goal of this job is to analyze web server log files to identify the top 5 IP addresses that generated the most client errors (HTTP status codes 400–499).

   The Python script below serves as the mapper in the Hadoop job. It reads lines from standard input (sys.stdin), which are lines from a web server log file. For each line,the IP address and the HTTP status code are extracted. If the status code indicates a client error (e.g it is between 400 and 499, inclusive), the mapper produces a key-value pair with the IP address as the key and 1 as the value. This output signifies an occurrence of a client error associated with that IP address. The script for the mapper is below.

```
#!/usr/bin/env python3
import sys
import re

# Extract the IP address and status code
log_pattern = re.compile(r'(?P<ip>\S+) \S+ \S+ \[[^\]]+\] "[^"]+" (?P<status>\d+)')

for line in sys.stdin:
    match = log_pattern.search(line)
    if match:
        ip = match.group('ip')
        status = int(match.group('status'))
        # Check if the status code indicates a client error
        if 400 <= status <= 499:
            print(f"{ip}\t1")
```

   The Python script below serves as the reducer in the Hadoop job. It reads key-value pairs from its standard input, where each line contains an IP address and a count (from the mapper's output). It aggregates these counts per IP address to determine the total number of client errors per IP. After processing all input, it identifies the top 5 IP addresses with the most client errors and prints each IP address and its associated count of client errors. The script for the reducer is below.

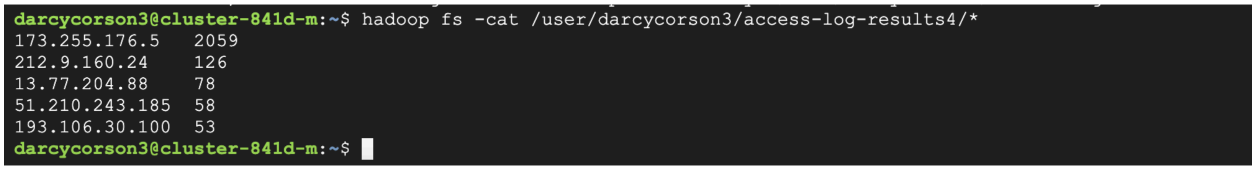
```
#!/usr/bin/env python3
import sys
from collections import defaultdict
from heapq import nlargest

# Initialize a dictionary to count occurrences of each IP
ip_counts = defaultdict(int)

# Process each line of input
for line in sys.stdin:
    ip, count = line.strip().split('\t')
    ip_counts[ip] += int(count)

# Find the top 5 IPs with the most client errors
top_ips = nlargest(5, ip_counts, key=ip_counts.get)

# Print the results
for ip in top_ips:
    print(f"{ip}\t{ip_counts[ip]}")
```

CS119 QUIZ 4 - Darcy Corson

The command below was designed to execute a Hadoop streaming job using both the 'ip_error_mapper.py' and 'top_ip_reducer.py'. The input for this job was specified as the '/user/darcycorson3/access.log file', from which the log data was read. The output destination was defined as the '/user/darcycorson3/access-log-results4' directory in the HDFS, the location where the job's results were stored. Note that the job configuration includes the '-numReduceTasks 1' parameter, which limits the number of reduce tasks to one. This setting is crucial for the accurate aggregation and identification of the top 5 IP addresses generating the most client errors across the entirety of the dataset, ensuring that the reduction phase is conducted by a single reducer to correctly compile the final results.

```
hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
  -files ip_error_mapper.py,top_ip_reducer.py \
  -mapper ip_error_mapper.py \
  -reducer top_ip_reducer.py \
  -input /user/darcycorson3/access.log \
  -output /user/darcycorson3/access-log-results4 \
  -numReduceTasks 1
```

The result of this job is below, and identifies the 5 IP addresses that generate the most client errors.



```
darcycorson3@cluster-841d-m:~$ hadoop fs -cat /user/darcycorson3/access-log-results4/*
173.255.176.5   2059
212.9.160.24    126
13.77.204.88    78
51.210.243.185  58
193.106.30.100  53
darcycorson3@cluster-841d-m:~$
```

5. This Hadoop streaming job involves two main components, a mapper (request_type_mapper.py) and a reducer (request_type_reducer.py), and a command to run the job on a Hadoop cluster. This job was designed to analyze web server logs to understand the distribution of HTTP request methods (like GET, POST, PUT, etc.) used in the traffic.

The 'request_type_mapper.py' Python script serves as the job's mapper. It processes each line of the input log files, both identifying and extracting HTTP request methods. When a request method is found, it emits a key-value pair, with the method as the key and a count of one as the value, to standard output. This step prepares the data for the reduction phase by categorizing each request by its type. The script for the mapper is below.

```
#!/usr/bin/env python3
import sys
import re

# Extract the request method
log_pattern = re.compile(r'\"(GET|POST|PUT|DELETE|HEAD|OPTIONS|PATCH|CONNECT|TRACE)\s')

for line in sys.stdin:
  match = log_pattern.search(line)
  if match:
    method = match.group(1)
    print(f"{method}\t1")
```

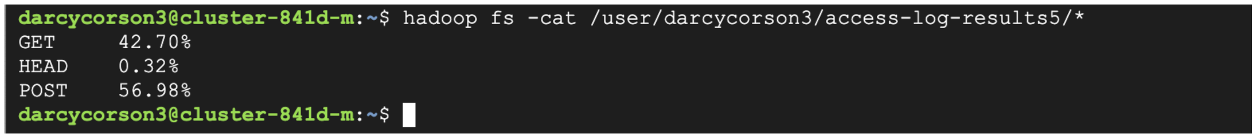
During the job’s reducing phase, the ‘request_type_reducer.py script’ aggregates the key-value pairs produced by the mapper. It reads from standard input, summing the counts for each HTTP method to determine the total number of requests made using each method. Additionally, it calculates the total count of all requests across all methods. Then, it computes the percentage of the total requests represented by each method, outputting these percentages alongside their corresponding HTTP methods. The script for the reducer is below.

```
#!/usr/bin/env python3
import sys

total_count = 0
method_counts = {}

for line in sys.stdin:
    line = line.strip()
    method, count = line.split('\t', 1)
    count = int(count)

    # Accumulate counts for each method and total count
    if method in method_counts:
        method_counts[method] += count
    else:
        method_counts[method] = count

    total_count += count

# Calculate and print percentages for each method
for method, count in method_counts.items():
    percentage = (count / total_count) * 100
    print(f"{method}\t{percentage:.2f}%")
```

The Hadoop job was configured to use a single reducer task (‘-D mapreduce.job.reduces=1’), ensuring that all data was processed together to accurately calculate percentages across the entire dataset. The ‘-files’ option specified that the scripts be copied to the cluster. The ‘-mapper’ and ‘-reducer’ options designated the specific scripts to use for the mapping and reducing phases. Input data is specified to be read from ‘/user/darcycorson3/access.log’, and the results of the job were directed to be stored in the ‘/user/darcycorson3/access-log-results5’ directory on the HDFS. The command below was used to execute the Hadoop streaming job described above.

```
hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
    -D mapreduce.job.reduces=1 \
    -files request_type_mapper.py,request_type_reducer.py \
    -mapper request_type_mapper.py \
    -reducer request_type_reducer.py \
    -input /user/darcycorson3/access.log\
    -output /user/darcycorson3/access-log-results5
```

The result of this job is below, and gives the percentages of each request type (GET, PUT, POST, etc.) found in the web server logs.



```
darcycorson3@cluster-841d-m:~$ hadoop fs -cat /user/darcycorson3/access-log-results5/*
GET     42.70%
HEAD    0.32%
POST    56.98%
darcycorson3@cluster-841d-m:~$
```

6. This Hadoop streaming job aims to further analyze web server logs, this time focusing on categorizing HTTP response status codes into predefined groups and calculating the percentage distribution of these groups. In order to perform this analysis, two Python scripts – 'status_code_mapper.py' for the mapping phase and 'status_code_reducer.py' for the reducing phase – were used to process the log data.

   Serving as the job's mapper, the 'status_code_mapper.py' Python script parses each line of the input log files, employing a regular expression to identify and extract the HTTP status codes. Depending on the range in which the status code falls, it categorizes each into one of five categories: Informational Responses (100-199), Successful Responses (200-299), Redirection Messages (300-399), Client Error Responses (400-499), or Server Error Responses (500-599). For each identified status code, the script creates a key-value pair, where the key is the category and the value is one, indicating a single occurrence of that category. The script for the mapper is below.

```
!/usr/bin/env python3
import sys
import re

# Match the log line format and capture the status code
log_pattern = re.compile(r'\s(\d{3})\s')

for line in sys.stdin:
    match = log_pattern.search(line)
    if match:
        status_code = int(match.group(1))
        if 100 <= status_code <= 199:
            print("Informational\t1")
        elif 200 <= status_code <= 299:
            print("Successful\t1")
        elif 300 <= status_code <= 399:
            print("Redirection\t1")
        elif 400 <= status_code <= 499:
            print("Client Error\t1")
        elif 500 <= status_code <= 599:
            print("Server Error\t1")
```

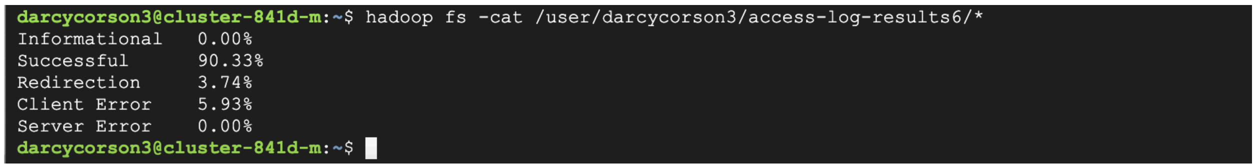
Once the mapping process is complete, the job's reducer, 'status_code_reducer.py', aggregates the key-value pairs by category. It accumulates the counts for each response category and calculates the total number of processed status codes. Using these counts, the Python script computes the percentage of the total requests represented by each category, outputting these percentages alongside their category names. The script for the reducer is below.

```
#!/usr/bin/env python3
import sys

response_counts = {
    "Informational": 0,
    "Successful": 0,
    "Redirection": 0,
    "Client Error": 0,
    "Server Error": 0
}

total_count = 0

for line in sys.stdin:
    line = line.strip()
    response_type, count = line.split('\t', 1)
    count = int(count)
    if response_type in response_counts:
        response_counts[response_type] += count
        total_count += count

for response_type, count in response_counts.items():
    percentage = (count / total_count) * 100 if total_count else 0
    print(f"{response_type}\t{percentage:.2f}%")
```

The job is configured to run with a single reduce task ('-D mapreduce.job.reduces=1'), ensuring that all data is consolidated into a single reducer for accurate calculation of percentage distributions across the entire dataset. The '-files' option specifies the mapper and reducer scripts to be copied to the cluster, while the '-mapper' and '-reducer' options designate the scripts for those respective phases. The input data is specified to come from '/user/darcycorson3/access.log', and the job's output is directed to '/user/darcycorson3/access-log-results6' on the HDFS.

```
hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
    -D mapreduce.job.reduces=1 \
    -files status_code_mapper.py,status_code_reducer.py \
    -mapper status_code_mapper.py \
    -reducer status_code_reducer.py \
    -input /user/darcycorson3/access.log\
    -output /user/darcycorson3/access-log-results6
```

The result of this job is below, and gives the percentages of each of the five responses found in the web server logs.



```
darcycorson3@cluster-841d-m:~$ hadoop fs -cat /user/darcycorson3/access-log-results6/*
Informational   0.00%
Successful      90.33%
Redirection     3.74%
Client Error    5.93%
Server Error    0.00%
darcycorson3@cluster-841d-m:~$
```

CS119 QUIZ 4 - Darcy Corson

PART 2 - PRESIDENTIAL SPEECHES

1. & 2.

In order to perform the analysis requested in this portion of the assignment spec, the presidential speech data was unzipped. Then, using a Python script, all of the presidential speech files were extracted from their .tar.gz archives, and each line of every speech was annotated with the name of the president who delivered it. The appended speech files were saved to a folder and uploaded to Hadoop, where further analysis could be performed. The modification of the speeches, by adding the president's name to each line, facilitated this analysis by associating each line of text with a specific president. The script that performed this preprocessing of the speeches is below.

```
import os
import tarfile

def prepend_presidents_name(folder_path):
   print("Starting to process .tar.gz files in:", folder_path)
   # Iterate over each .tar.gz file in the specified folder
   for filename in os.listdir(folder_path):
       if filename.endswith(".tar.gz"):
           president_name = filename[:-7]  # Get the president's name
           print(f"Processing speeches for: {president_name}")
           tar_path = os.path.join(folder_path, filename)

           # Extract the .tar.gz file
           with tarfile.open(tar_path, "r:gz") as tar:
               print(f"Extracting {filename}...")
               tar.extractall(path=folder_path)
               # Iterate over each file in the extracted folder
               for member in tar.getmembers():
                   speech_path = os.path.join(folder_path, member.name)
                   if os.path.isfile(speech_path):
                       with open(speech_path, 'r') as file:
                           lines = file.readlines()
                       # Prepend president's name to each line
                       with open(speech_path, 'w') as file:
                           for line in lines:
                               file.write(president_name + ": " + line)
               print(f"Finished processing {filename}.")

   print("All .tar.gz files have been processed.")

folder_path = '/Users/dcorson/Desktop/prez_speeches 2'
prepend_presidents_name(folder_path)
```

The ‘prez_mapper’ Python script is designed for sentiment analysis within a MapReduce framework, specifically analyzing speeches by presidents. Initially, it loads the AFINN word list, which assigns sentiment valence scores to words, using the ‘load_afinn_word_list’ function. This list provides a score ranging from -5 (very negative) to +5 (very positive) for each word on the list. Words not on the list are assigned a score of 0. The script processes each word of the input text, which are the presidents’ speeches, by converting it to lowercase and removing punctuation with the ‘clean_word function’, ensuring that words match the format of those in the AFINN list.

CS119 QUIZ 4 - Darcy Corson

The ‘emit_president_word_valence function’ iterates through the words in a speech, cleans them, and checks for their presence in the AFINN list. If a word is found, the script emits a key-value pair consisting of the president's name and the word's sentiment score. This output is formatted as the president's name followed by a tab character and the score, suitable for processing in the reduce phase of a MapReduce job. The script reads from ‘sys.stdin’, allowing it to handle input streamed through standard input. It expects each line of input to follow a specific format: the president's name followed by a colon and then the speech text. The script splits these components, trims any whitespace, and applies the ‘emit_president_word_valence function’ to analyze the sentiment of each word in the speech. This approach enables the mapper to systematically evaluate and emit the sentiment scores of words in presidential speeches, setting the foundation for aggregate sentiment analysis during the reduce phase. The script for the mapper is below.

```
#!/usr/bin/env python3
import sys
import re
import string

def load_afinn_word_list(afinn_path):
    afinn_word_list = {}
    with open(afinn_path, 'r', encoding='utf-8') as file:
        for line in file:
            parts = line.strip().split('\t')
            if len(parts) == 2:
                word, score = parts
                afinn_word_list[word] = int(score)
    return afinn_word_list

afinn_path = "/home/darcycorson3/AFINN-en-165.txt"
afinn_word_list = load_afinn_word_list(afinn_path)

def clean_word(word):
    word = word.lower()
    word = word.strip(string.punctuation)
    return word

def emit_president_word_valence(president, text, afinn):
    words = text.split()
    for word in words:
        cleaned_word = clean_word(word)
        if cleaned_word and cleaned_word in afinn:
            print(f"{president}\t{afinn[cleaned_word]}")

for line in sys.stdin:
    line = line.strip()
    if ':' in line:
        president, speech = line.split(':', 1)
        emit_president_word_valence(president.strip(), speech.strip(), afinn_word_list)
```

CS119 QUIZ 4 - Darcy Corson

The Python script below, ‘prez_reducer.py’ is the reducer for this job and is designed to compute the average sentiment valence for each president’s language based on their speeches. The ‘reduce_president_valence’ method reads input lines from sys.stdin, where each line consists of a president's name and a sentiment valence score, separated by a tab. This function works by maintaining a running total of valence scores and a count of words for the given president being processed. As it reads each line, the script checks to see if the president associated with that line matches the current president in context. If it does, the valence score is added to the total for that president, and the word count is incremented. If the line is associated with a new president, the function calculates the average valence for the previous president by dividing the total valence by the word count, outputs this average, and resets counters and totals for the new president. This process repeats until all lines are processed. The script for the reducer is below.

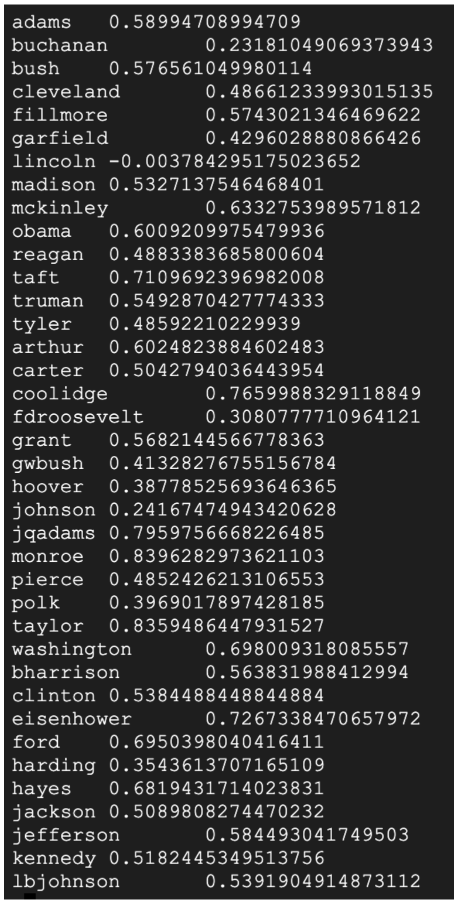
```
#!/usr/bin/env python3
import sys

def reduce_president_valence():
    current_president = None
    total_valence = 0
    word_count = 0

    for line in sys.stdin:
        line = line.strip()
        president, valence = line.split('\t', 1)
        valence = float(valence)

        if current_president == president:
            total_valence += valence
            word_count += 1
        else:
            if current_president is not None:
                # Emit the average valence for the current president
                print(f"{current_president}\t{total_valence / word_count}")
            current_president = president
            total_valence = valence
            word_count = 1

    if current_president is not None:
        print(f"{current_president}\t{total_valence / word_count}")

if __name__ == "__main__":
    reduce_president_valence()
```

The job begins by specifying the Hadoop streaming JAR file to launch it, followed by the ‘-file’ option to upload the ‘prez_mapper.py’ and ‘prez_reducer.py’ scripts, as well as the ‘AFINN-en-165.txt’ file, from the local file system to the Hadoop cluster. The mapper and reducer scripts are designated through the ‘-mapper’ and ‘-reducer’ options to process the input data. The mapper script (‘prez_mapper.py’) analyzes each line of the input files to perform sentiment analysis based on the AFINN word list, outputting key-value pairs. The reducer script (‘prez_reducer.py’) aggregates these results by president, calculating the average sentiment score for each one's speeches. The ‘-input’ option specifies the HDFS directory containing the pre-processed speeches, with each line prepended by the respective president's name, as the source of data for the job. The ‘-output’ option defines the HDFS directory where the results of the job are stored.

CS119 QUIZ 4 - Darcy Corson

```
hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
    -file /home/darcycorson3/prez_mapper.py    \
    -mapper /home/darcycorson3/prez_mapper.py  \
    -file /home/darcycorson3/prez_reducer.py   \
    -reducer /home/darcycorson3/prez_reducer.py \
    -file /home/darcycorson3/AFINN-en-165.txt \
    -input /user/darcycorson3/prepended_speeches/* \
    -output /user/darcycorson3/prez_analysis
```

The result of this job is below, and gives the average valence of each president's speeches.



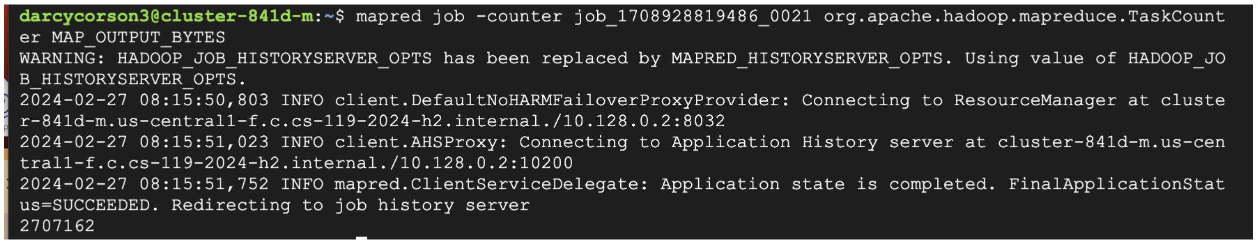
```
adams   0.58994708994709
buchanan        0.23181049069373943
bush    0.576561049980114
cleveland       0.48661233993015135
fillmore        0.5743021346469622
garfield        0.4296028880866426
lincoln -0.003784295175023652
madison 0.5327137546468401
mckinley        0.6332753989571812
obama   0.6009209975479936
reagan  0.4883383685800604
taft    0.7109692396982008
truman  0.5492870427774333
tyler   0.48592210229939
arthur  0.6024823884602483
carter  0.5042794036443954
coolidge        0.7659988329118849
fdroosevelt     0.3080777710964121
grant   0.5682144566778363
gwbush  0.41328276755156784
hoover  0.38778525693646365
johnson 0.24167474943420628
jqadams 0.7959756668226485
monroe  0.8396282973621103
pierce  0.4852426213106553
polk    0.3969017897428185
taylor  0.8359486447931527
washington      0.698009318085557
bharrison       0.563831988412994
clinton 0.5384488448844884
eisenhower      0.7267338470657972
ford    0.6950398040416411
harding 0.3543613707165109
hayes   0.6819431714023831
jackson 0.5089808274470232
jefferson       0.584493041749503
kennedy 0.5182445349513756
lbjohnson       0.5391904914873112
```

CS119 QUIZ 4 - Darcy Corson

The following command was executed to determine the amount of data, in bytes emitted by the mappers during the job execution.

```
mapred job -counter job_1708928819486_0021 0rg.apache.hadoop.mapreduce.TaskCounter
MAP_OUTPUT_BYTES
```

Below, is the result. The mappers emitted a total of 2,707,162 bytes of data



```
darcycorson3@cluster-841d-m:~$ mapred job -counter job_1708928819486_0021 org.apache.hadoop.mapreduce.TaskCount
er MAP_OUTPUT_BYTES
WARNING: HADOOP_JOB_HISTORYSERVER_OPTS has been replaced by MAPRED_HISTORYSERVER_OPTS. Using value of HADOOP_JO
B_HISTORYSERVER_OPTS.
2024-02-27 08:15:50,803 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at cluste
r-841d-m.us-central1-f.c.cs-119-2024-h2.internal./10.128.0.2:8032
2024-02-27 08:15:51,023 INFO client.AHSProxy: Connecting to Application History server at cluster-841d-m.us-cen
tral1-f.c.cs-119-2024-h2.internal./10.128.0.2:10200
2024-02-27 08:15:51,752 INFO mapred.ClientServiceDelegate: Application state is completed. FinalApplicationStat
us=SUCCEEDED. Redirecting to job history server
2707162
```

CS119 QUIZ 5 - Darcy Corson

PART 1 - ABOUT TF-IDF

1. The TF value for a word in a document is a measure of how frequently that word occurs in a given document. The IDF is a measure of how important a word in a document is to a collection of documents, or corpus. That is, it is a measure of how common or rare a word is in an entire document set. Thus, while TF can be calculated for a word in a single document, the calculation of IDF can only be performed for a set of documents, because the IDF value means to express something about a set of documents.

2. Scikit-Learn's IDF calculation methodology is different from the 'standard' IDF methodology in two ways. First, and unlike for the 'standard' IDF equation, some constant 1 is added to both the numerator and denominator of the Scikit-Learn's IDF equation as though an extra document was seen that contains every term in the collection exactly once. This has the effect of eliminating the possibility of the denominator being 0, wholly preventing division by 0. Division by 0 is undesirable because it necessarily results in an IDF value that is undefined. Second, and also unlike the 'standard' IDF equation, the Scikit-Learn's IDF equation requires the addition of 1 to the final result, as an act of 'smoothing.' The result of adding one ensures that no word ends up with a 0 IDF score just because it is common. When looking for information or sorting documents into categories, it is important that every word has the opportunity to help in that process, because common words can play a role in defining the content and context of the documents. By ensuring that no word has a zero weight, the Scikit-Learn's IDF equation may make better use of all the words, potentially leading to improved performance in tasks like search and document classification.

PART 2 - PRESIDENTIAL SPEECHES

3. For each speech, the mapper receives raw text from one of the 10 inaugural speeches as input. The input is the entire speech document, and it is cleaned before being analyzed; cleaning includes the provided actions and also the removal of single letter words. The mapper will tokenize the speech into words. For each word in a given inaugural speech, the mapper will emit a key-value pair where the key is a tuple of (word, document_id) and the value is the count of 1. The output will be key-value pairs of the format: ((word, document_id), 1). Before the reducer, Hadoop will sort and group these keys. Note that the mapper calculates the raw frequency (which is then normalized in the reducer to calculate TF). I assume that this is what the assignment instructions mean when they say that the mapper should be used for TF; the mapper should not calculate the actual TF part of the TF-IDF because it does not have information about the total number of words in the document to normalize the term frequency. Thus, it makes the most sense for the mapper's role to be to process records independently and prepare them for the reducer, which then aggregates these intermediate results.

   The reducer receives (word, document_id), [1, 1, 1, ...]. It sums the counts for each word to find the total term frequency (TF) for that word in the document. Then, it calculates the document frequency (DF) for each word, which is the number of documents in the corpus that contain that word. The IDF is calculated using the standard approach: IDF = log((Total number of documents) / (DF). The TF-IDF score is found by multiplying the TF by the IDF. The reducer maintains a local priority queue that keeps track of the top 20 highest TF-IDF scores for the words that it processes. The reducer's final output will be the top-20 words with the highest TF-IDF scores for each document.

4. To begin the analysis, the preprocessing script below is designed and executed to preliminarily access and normalize the text data of the last 10 inaugural speeches.

```
#!/usr/bin/env python
import nltk
import nltk.corpus
from nltk.corpus import inaugural
nltk.download('inaugural')

REPLACEMENTS = [
    (". ",   ".\n"),
    ("Mr.\n", "Mr. "),
    ("Ms.\n", "Ms. "),
    ("Mrs.\n", "Mrs. "),
]

speech_names = nltk.corpus.inaugural.fileids()[-10:]

for name in speech_names:
    text = inaugural.raw(name)
    text = ''.join([i if ord(i) < 128 else ' ' for i in text])
    text = ' '.join(text.split())

    for old, new in REPLACEMENTS:
        text = text.replace(old, new)

    # Write the cleaned text to a new file
    with open('inaug_' + name, 'w') as writer:
        writer.write(text + '\n')
```

The script begins by importing the necessary modules from the Natural Language Toolkit (NLTK). Then, it downloads the inaugural corpus from NLTK, which contains a collection of inaugural speeches from various presidents of the United States. A list of REPLACEMENTS is defined for standardizing certain patterns in the text. For example, after a period, a newline character is added to make the text more readable. Corrections for abbreviations like "Mr.", "Ms.", and "Mrs." are also employed to avoid breaking lines incorrectly after these terms. Though not necessary for this analysis per se, such standardization of text is good practice and was thus performed. Specifically, the script fetches the file IDs of the 10 inaugural speeches from the inaugural corpus and, for each of these speeches, performs the following preprocessing steps: it retrieves the raw text of the speech, it removes any non-ASCII characters and replaces them with a space to avoid encoding issues that could complicate text analysis, it reduces any sequences of multiple spaces to a single space thus leading to a more consistent and clean text, and it applies the previously defined replacements to handle periods and certain abbreviations appropriately. Finally, the script writes the processed text to a new file. Each file is named by appending 'inaug_' to the original file name, and the content is written with an additional newline character at the end for better formatting. The speech files are now ready to pass to the mapper.

CS119 QUIZ 5 - Darcy Corson

The mapper script plays an important initial role in analyzing the inaugural speeches to identify the top-20 TF-IDF values for each of the last 10 presidential speeches.

```
!/usr/bin/env python3
import sys
import os
import re
import string

def clean_text(text):
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\s+', ' ', text)
    return text.strip()

file_path = os.environ['map_input_file']
document_id = os.path.basename(file_path)

for content in sys.stdin:
    words = clean_text(content).split()

    # Emit each word with the document_id and a count of 1, filtering out single-character words
    for word in words:
        if len(word) > 1:  # Only emit words longer than 1 character
            print(f'{word}\t{document_id}\t1')
```

Specifically, it processes the text of each speech to prepare it for TF-IDF analysis. The script begins by defining a clean_text function, which standardizes the raw text by converting it to lowercase, stripping annotations, punctuation, numbers, and excessive whitespace, thereby ensuring that the analysis is focused on meaningful content. After obtaining the name of the input file, which serves as the document identifier, the script reads the content from standard input. Then, it cleans and tokenizes the text, excluding single-character tokens which are generally not significant for TF-IDF analysis.

For each meaningful word, the mapper emits a key-value pair where the key is the word itself concatenated with its document identifier, and the value is the integer one. These pairs are ready for use by the reducer, where the actual calculation of TF and subsequent IDF is performed – please see the response to prompt 3) for a short discussion of why the decision to handle the calculation of TF-IDF this way was made. The output of the mapper is designed to ensure accurate TF calculation for the reducer, setting the stage for the final TF-IDF computation.

CS119 QUIZ 5 - Darcy Corson

The reducer computes the TF-IDF values for words across the 10 presidential inaugural speeches.

```
#!/usr/bin/env python
import sys
import math
from collections import defaultdict

# Number of documents in the corpus
total_documents = 10

# Data structures for calculations
word_counts = defaultdict(lambda: defaultdict(int))  # word -> doc -> count
doc_word_counts = defaultdict(int)  # doc -> total words
doc_count = defaultdict(int)  # word -> number of documents containing word

for line in sys.stdin:
    line = line.strip()
    word, document_id, count = line.split('\t', 2)
    count = int(count)
    word_counts[word][document_id] += count
    doc_word_counts[document_id] += count

# Compute DF for each word
for word in word_counts:
    doc_count[word] = len(word_counts[word])

# Calculate TF-IDF for each word in each document and keep DF information
tf_idf_scores = defaultdict(list)  # doc -> [(tf_idf, word, df), ...]

for word, docs in word_counts.items():
    idf = math.log(total_documents / doc_count[word])
    for document_id, count in docs.items():
        tf = count / float(doc_word_counts[document_id])
        tf_idf = tf * idf
        df = doc_count[word]
        tf_idf_scores[document_id].append((tf_idf, word, df))

# Output the top 20 words by TF-IDF for each document along with their DF for error checking
for document_id, scores in tf_idf_scores.items():
    top_20 = sorted(scores, key=lambda x: x[0], reverse=True)[:20]
    for tf_idf, word, df in top_20:
        print(f"{document_id}\t{word}\t{tf_idf}\t{df}")
```

After the mapper processes and emits key-value pairs representing words and their occurrences in each document, the reducer takes over to aggregate these counts and calculate the TF-IDF scores. The reducer’s script begins by initializing data structures to store word counts within each document, total word counts per document, and document frequency counts for each word across all documents. It processes input from the mapper line by line, parsing the word, document identifier, and count, and updating its data structures accordingly. With this data, the reducer calculates the DF for each word, which is the number of documents the word appears in. This DF is used to compute the IDF score. The script calculates TF by dividing the count of a word in a document by the total word count of that document, then multiplies TF by IDF to get the TF-IDF score for each word in each document. Finally, for each document, the reducer sorts the words by their TF-IDF scores in descending order and outputs the top 20 words along with their TF-IDF scores and document frequency for manual checking. This output highlights the words that are uniquely significant in individual speeches relative to the entire corpus of the last 10 presidential inauguration speeches.

The Hadoop Streaming command below is a critical step that orchestrates the entire MapReduce process described above.

```
.hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
    -files inaug_mapper.py,inaug_reducer.py \
    -mapper "python3 inaug_mapper.py" \
    -reducer "python3 inaug_reducer.py" \
    -input /user/hadoop/inaug_speeches/* \
    -output /user/hadoop/output/inaug_results \
    -numReduceTasks 1
```

The command explicitly defines the input directory '/user/hadoop/inaug_speeches/*', where all speeches are stored, and the output directory '/user/hadoop/output/inaug_results', where the results, including the top-20 TF-IDF scores per speech, are written. Setting '-numReduceTasks 1' ensures that a single reducer is used for this job, which is suitable for this assignment's scale and helps to maintain the order of results as required by the assignment spec. Executing this Hadoop Streaming command initiates the MapReduce job.

As discussed, the output of the MapReduce job are the top-20 TF-IDF words for each of the 10 inaugural speeches. These are listed below.

```
inaug_2005-Bush.txt    xand    0.00564912927623662    1
inaug_2005-Bush.txt    seen    0.00354457155346203    3
inaug_2005-Bush.txt    excuse  0.003389477565741971    1
inaug_2005-Bush.txt    questions       0.003389477565741971    1
inaug_2005-Bush.txt    fire    0.003158857531764671 5   2
inaug_2005-Bush.txt    freedom 0.002956268834879128 2   8
inaug_2005-Bush.txt    tyranny 0.00295380962788502 5    3
inaug_2005-Bush.txt    human   0.002697617463810074    4
inaug_2005-Bush.txt    defended        0.0023691431488235036   2
inaug_2005-Bush.txt    goal    0.0023691431488235036   2
inaug_2005-Bush.txt    institutions    0.0023691431488235036   2
inaug_2005-Bush.txt    permanent       0.0023691431488235036   2
inaug_2005-Bush.txt    abandon 0.002259651710494647 6   1
inaug_2005-Bush.txt    accepted        0.0022596517104946476   1
inaug_2005-Bush.txt    events  0.0022596517104946476   1
inaug_2005-Bush.txt    fulfill 0.0022596517104946476   1
inaug_2005-Bush.txt    influence       0.0022596517104946476   1
inaug_2005-Bush.txt    meant   0.0022596517104946476   1
inaug_2005-Bush.txt    mercy   0.0022596517104946476   1
inaug_2005-Bush.txt    ownership       0.0022596517104946476   1

inaug_2009-Obama.txt   carried 0.0029394703314817605   1
inaug_2009-Obama.txt   father  0.0020546015903414043   2
inaug_2009-Obama.txt   calls   0.001959646887654507    1
inaug_2009-Obama.txt   charter 0.001959646887654507    1
inaug_2009-Obama.txt   conflict        0.001959646887654507    1
inaug_2009-Obama.txt   cooperation     0.001959646887654507    1
inaug_2009-Obama.txt   faced   0.001959646887654507    1
inaug_2009-Obama.txt   false   0.001959646887654507    1
inaug_2009-Obama.txt   icy     0.001959646887654507    1
inaug_2009-Obama.txt   short   0.001959646887654507    1
inaug_2009-Obama.txt   virtue  0.001959646887654507    1
inaug_2009-Obama.txt   waters  0.001959646887654507    1
```

CS119 QUIZ 5 - Darcy Corson

```
inaug_2009-Obama.txt	why	0.001959646887654507	1
inaug_2009-Obama.txt	willingness	0.001959646887654507	1
inaug_2009-Obama.txt	crisis	0.001559643798934732	4
inaug_2009-Obama.txt	whether	0.001559643798934732	4
inaug_2009-Obama.txt	off	0.001536986558713961	3
inaug_2009-Obama.txt	rather	0.001536986558713961	3
inaug_2009-Obama.txt	less	0.0015216082410050788	6
inaug_2009-Obama.txt	ambitions	0.0013697343935609365	2

inaug_2001-Bush.txt	civility	0.00603165708708329	1
inaug_2001-Bush.txt	story	0.0060005941838517035	4
inaug_2001-Bush.txt	affirm	0.0031619605352339883	2
inaug_2001-Bush.txt	commitment	0.0031619605352339883	2
inaug_2001-Bush.txt	compassion	0.0031619605352339883	2
inaug_2001-Bush.txt	sometimes	0.0031538252896553664	3
inaug_2001-Bush.txt	angel	0.003015828543541645	1
inaug_2001-Bush.txt	directs	0.003015828543541645	1
inaug_2001-Bush.txt	grand	0.003015828543541645	1
inaug_2001-Bush.txt	laws	0.003015828543541645	1
inaug_2001-Bush.txt	rides	0.003015828543541645	1
inaug_2001-Bush.txt	stakes	0.003015828543541645	1
inaug_2001-Bush.txt	whirlwind	0.003015828543541645	1
inaug_2001-Bush.txt	everyone	0.002365368967241525	3
inaug_2001-Bush.txt	principles	0.002365368967241525	3
inaug_2001-Bush.txt	generous	0.0021079736901559926	2
inaug_2001-Bush.txt	honored	0.0021079736901559926	2
inaug_2001-Bush.txt	humanity	0.0021079736901559926	2
inaug_2001-Bush.txt	share	0.0021079736901559926	2
inaug_2001-Bush.txt	beyond	0.001815709706771304	5

inaug_1993-Clinton.txt	season	0.005847835156810276	1
inaug_1993-Clinton.txt	renewal	0.0040874613649120005	2
inaug_1993-Clinton.txt	whom	0.0040874613649120005	2
inaug_1993-Clinton.txt	posterity	0.0030655960236840004	2
inaug_1993-Clinton.txt	raised	0.0030655960236840004	2
inaug_1993-Clinton.txt	sake	0.0030655960236840004	2
inaug_1993-Clinton.txt	spring	0.0030655960236840004	2
inaug_1993-Clinton.txt	compete	0.002923917578405138	1
inaug_1993-Clinton.txt	serving	0.002923917578405138	1
inaug_1993-Clinton.txt	change	0.0029190035643770896	6
inaug_1993-Clinton.txt	idea	0.002908859466267159	4
inaug_1993-Clinton.txt	capitol	0.0022932815320494022	3
inaug_1993-Clinton.txt	powerful	0.0017453156797602955	4
inaug_1993-Clinton.txt	done	0.001621668646876161	6
inaug_1993-Clinton.txt	ceremony	0.0015288543546996014	3
inaug_1993-Clinton.txt	depression	0.0015288543546996014	3
inaug_1993-Clinton.txt	forth	0.0015288543546996014	3
inaug_1993-Clinton.txt	founders	0.0015288543546996014	3
inaug_1993-Clinton.txt	mission	0.0015288543546996014	3
inaug_1993-Clinton.txt	preserve	0.0015288543546996014	3

inaug_1997-Clinton.txt	enough	0.0034596919664538392	3
inaug_1997-Clinton.txt	century	0.0034164266660798117	7
inaug_1997-Clinton.txt	awful	0.00220554127681422	1
inaug_1997-Clinton.txt	coast	0.00220554127681422	1
inaug_1997-Clinton.txt	doors	0.00220554127681422	1
inaug_1997-Clinton.txt	information	0.00220554127681422	1
inaug_1997-Clinton.txt	labors	0.00220554127681422	1
inaug_1997-Clinton.txt	moved	0.00220554127681422	1
inaug_1997-Clinton.txt	quest	0.00220554127681422	1
inaug_1997-Clinton.txt	religious	0.00220554127681422	1
inaug_1997-Clinton.txt	saved	0.00220554127681422	1
```

CS119 QUIZ 5 - Darcy Corson

```
inaug_1997-Clinton.txt  seemed  0.00220554127681422     1
inaug_1997-Clinton.txt  solve   0.00220554127681422     1
inaug_1997-Clinton.txt  sustain 0.00220554127681422     1
inaug_1997-Clinton.txt  waste   0.00220554127681422     1
inaug_1997-Clinton.txt  human   0.002194182787054968   4
inaug_1997-Clinton.txt  dreams  0.0017298459832269196   3
inaug_1997-Clinton.txt  streets 0.0017298459832269196   3
inaug_1997-Clinton.txt  bridge  0.0015416071958181037   2
inaug_1997-Clinton.txt  bright  0.0015416071958181037   2

inaug_2021-Biden.txt    re      0.00562291842000988     1
inaug_2021-Biden.txt    me      0.004102237708838301    4
inaug_2021-Biden.txt    virus   0.003748612280006586   1
inaug_2021-Biden.txt    days    0.00343012194964654 2   3
inaug_2021-Biden.txt    story   0.003356376307231337 5  4
inaug_2021-Biden.txt    truth   0.003275209427012820 8  2
inaug_2021-Biden.txt    lies    0.00281145921000494     1
inaug_2021-Biden.txt    objects 0.00281145921000494     1
inaug_2021-Biden.txt    prevailed       0.00281145921000494     1
inaug_2021-Biden.txt    uniting 0.00281145921000494     1
inaug_2021-Biden.txt    cry     0.0019651256562076926   2
inaug_2021-Biden.txt    get     0.0019651256562076926   2
inaug_2021-Biden.txt    violence        0.0019651256562076926   2
inaug_2021-Biden.txt    ve      0.0019600696855123093   3
inaug_2021-Biden.txt    disagree        0.0018743061400032933   1
inaug_2021-Biden.txt    disagreement    0.0018743061400032933   1
inaug_2021-Biden.txt    doesn   0.0018743061400032933   1
inaug_2021-Biden.txt    everything      0.0018743061400032933   1
inaug_2021-Biden.txt    extremism       0.0018743061400032933   1
inaug_2021-Biden.txt    folks   0.0018743061400032933   1

inaug_2017-Trump.txt    protected       0.005584447995954546 5  2
inaug_2017-Trump.txt    countries       0.004793723302555265    1
inaug_2017-Trump.txt    obama   0.004793723302555265    1
inaug_2017-Trump.txt    dreams  0.004177560042768689    3
inaug_2017-Trump.txt    capital 0.0033506687975727277   2
inaug_2017-Trump.txt    everyone        0.003342048034214951    3
inaug_2017-Trump.txt    breath  0.0031958155350368437   1
inaug_2017-Trump.txt    exists  0.0031958155350368437   1
inaug_2017-Trump.txt    glorious        0.0031958155350368437   1
inaug_2017-Trump.txt    industry        0.0031958155350368437   1
inaug_2017-Trump.txt    mountain        0.0031958155350368437   1
inaug_2017-Trump.txt    politicians     0.0031958155350368437   1
inaug_2017-Trump.txt    righteous       0.0031958155350368437   1
inaug_2017-Trump.txt    shine   0.0031958155350368437   1
inaug_2017-Trump.txt    stops   0.0031958155350368437   1
inaug_2017-Trump.txt    transferring    0.0031958155350368437   1
inaug_2017-Trump.txt    trillions       0.0031958155350368437   1
inaug_2017-Trump.txt    winning 0.0031958155350368437   1
inaug_2017-Trump.txt    again   0.003190444562036028    6
inaug_2017-Trump.txt    while   0.002886109009965074    5

inaug_2013-Obama.txt    complete        0.003891290890798115    2
inaug_2013-Obama.txt    until   0.003493151269804457    3
inaug_2013-Obama.txt    evident 0.0033403071948656373   1
inaug_2013-Obama.txt    knowing 0.0033403071948656373   1
inaug_2013-Obama.txt    she     0.003113032712638492    2
inaug_2013-Obama.txt    requires        0.0029109593915037143   3
inaug_2013-Obama.txt    happiness       0.0023287675132029713   3
inaug_2013-Obama.txt    compel  0.002226871463243758    1
inaug_2013-Obama.txt    harm    0.002226871463243758    1
inaug_2013-Obama.txt    initiative      0.002226871463243758    1
```

CS119 QUIZ 5 - Darcy Corson

```
inaug_2013-Obama.txt	lessons 0.002226871463243758	1
inaug_2013-Obama.txt	thrives 0.002226871463243758	1
inaug_2013-Obama.txt	train	0.002226871463243758	1
inaug_2013-Obama.txt	enduring	0.0017465756349022284	3
inaug_2013-Obama.txt	principles	0.0017465756349022284	3
inaug_2013-Obama.txt	require 0.0017465756349022284	3
inaug_2013-Obama.txt	self	0.0017465756349022284	3
inaug_2013-Obama.txt	creed	0.0016758877673112799	5
inaug_2013-Obama.txt	capacity	0.001556516356319246	2
inaug_2013-Obama.txt	debates 0.001556516356319246	2

inaug_1989-Bush.txt	breeze	0.0051930200563690705	1
inaug_1989-Bush.txt	door	0.004154416045095257	1
inaug_1989-Bush.txt	fact	0.003629765251317321	2
inaug_1989-Bush.txt	word	0.003629765251317321	2
inaug_1989-Bush.txt	don	0.003258383773547865	3
inaug_1989-Bush.txt	mr	0.003258383773547865	3
inaug_1989-Bush.txt	blowing 0.0031158120338214425	1
inaug_1989-Bush.txt	expression	0.0031158120338214425	1
inaug_1989-Bush.txt	loyal	0.0031158120338214425	1
inaug_1989-Bush.txt	crucial 0.002177859150790393	2
inaug_1989-Bush.txt	engagement	0.002177859150790393	2
inaug_1989-Bush.txt	seems	0.002177859150790393	2
inaug_1989-Bush.txt	mean	0.00217225584903191	3
inaug_1989-Bush.txt	begins	0.0020772080225476284	1
inaug_1989-Bush.txt	bow	0.0020772080225476284	1
inaug_1989-Bush.txt	executive	0.0020772080225476284	1
inaug_1989-Bush.txt	heads	0.0020772080225476284	1
inaug_1989-Bush.txt	involved	0.0020772080225476284	1
inaug_1989-Bush.txt	leadership	0.0020772080225476284	1
inaug_1989-Bush.txt	majority	0.0020772080225476284	1

inaug_1985-Reagan.txt	increase	0.0037019052942026463	1
inaug_1985-Reagan.txt	human	0.0033145565059756413	4
inaug_1985-Reagan.txt	federal 0.0032344009494254425	2
inaug_1985-Reagan.txt	nuclear 0.002903471393068978	3
inaug_1985-Reagan.txt	weapons 0.002903471393068978	3
inaug_1985-Reagan.txt	number	0.0027764289706519846	1
inaug_1985-Reagan.txt	senator 0.002587520759540354	2
inaug_1985-Reagan.txt	tax	0.002587520759540354	2
inaug_1985-Reagan.txt	reduce	0.002419559494224148	3
inaug_1985-Reagan.txt	beginning	0.0019406405696552659	2
inaug_1985-Reagan.txt	destroy 0.0019406405696552659	2
inaug_1985-Reagan.txt	program 0.0019406405696552659	2
inaug_1985-Reagan.txt	song	0.0019406405696552659	2
inaug_1985-Reagan.txt	spend	0.0019406405696552659	2
inaug_1985-Reagan.txt	economic	0.0019356475953793185	3
inaug_1985-Reagan.txt	self	0.0019356475953793185	3
inaug_1985-Reagan.txt	aimed	0.0018509526471013232	1
inaug_1985-Reagan.txt	arm	0.0018509526471013232	1
inaug_1985-Reagan.txt	arsenals	0.0018509526471013232	1
inaug_1985-Reagan.txt	blow	0.0018509526471013232	1
```

CS119 QUIZ 5 - Darcy Corson

The top-20 TF-IDF words from George W. Bush's inaugural speeches in 2001 and 2005 reveal significant insights into the main issues and concerns that occupied the United States during his presidency. In the 2001 speech, terms such as "civility," "compassion," "commitment," "principles," and "humanity" indicate a focus on unity, ethical governance, and social values, reflecting the context of a country seeking direction after a contentious election. Notably, "civility" and "compassion" suggest an appeal for national harmony and understanding, aligning with Bush's initial approach to domestic policy and social welfare. In contrast, the 2005 speech, occurring in the context of post-9/11 America, emphasizes terms like "freedom," "tyranny," "liberty," and "ownership." These words reflect the global challenges that the US faced at the time, particularly in terms of terrorism and the wars in Afghanistan and Iraq. Notably, "freedom" and the fight against "tyranny" highlight Bush's focus on spreading democracy and combating terrorism as central themes of his foreign policy. Additionally, "ownership" and "liberty" resonate with Bush's domestic agenda, promoting individual rights and economic freedom.

Similarly, the top-20 TF-IDF words from Barack Obama's inaugural speeches in 2009 and 2013 reflect the themes and concerns that were prevalent in the United States during his presidency. In the 2009 speech, words like "crisis," "ambitions," "conflict," and "cooperation" reflect Obama’s focus on the significant economic challenges facing the country at the time as well as his appeal for unity and collective effort to overcome these obstacles. Notably, Obama’s mention of "crisis" underscores the gravity of the economic downturn at the time, while "cooperation" and "ambitions" reflect Obama's call for national solidarity to rebuild the economy. Additionally, words like "father," "charter," and "virtue" suggest an emphasis on returning to foundational values and principles as a way to guide the nation through difficult times. In Obama’s 2013 speech, terms such as "complete," "enduring," "principles," and "requires" point towards a narrative of continuity, resilience, and the ongoing work required to sustain democracy and progress. Notably, the words "complete" and "enduring" suggest a long-term perspective on the challenges and achievements of the nation, while "principles" and "requires" underscore the importance of adhering to fundamental democratic values and the effort needed to maintain them. Words like "happiness," "capacity," and "debates" highlight Obama's focus on the well-being of citizens and the importance of dialogue and discussion in a healthy democracy. Further, the presence of words related to personal responsibility and moral action, such as "compel," "harm," "initiative," and "self," in both speeches, reflects Obama's emphasis on the importance of individual responsibility in contributing to the collective good and addressing societal challenges. This narrative aligns with the broader issues the US was grappling with at the time, including economic recovery, healthcare reform, and climate change.