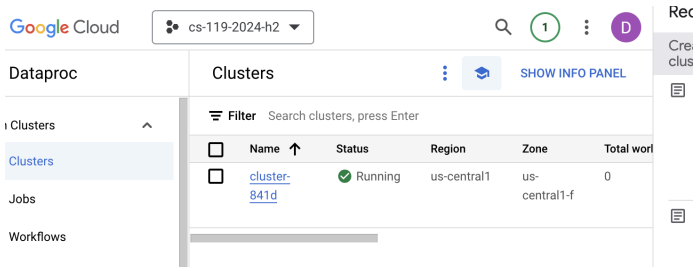
PART 1 - ANALYZING SERVER LOGS

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



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]}")
```

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)
```

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.

```
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
```