Vicente De Leon

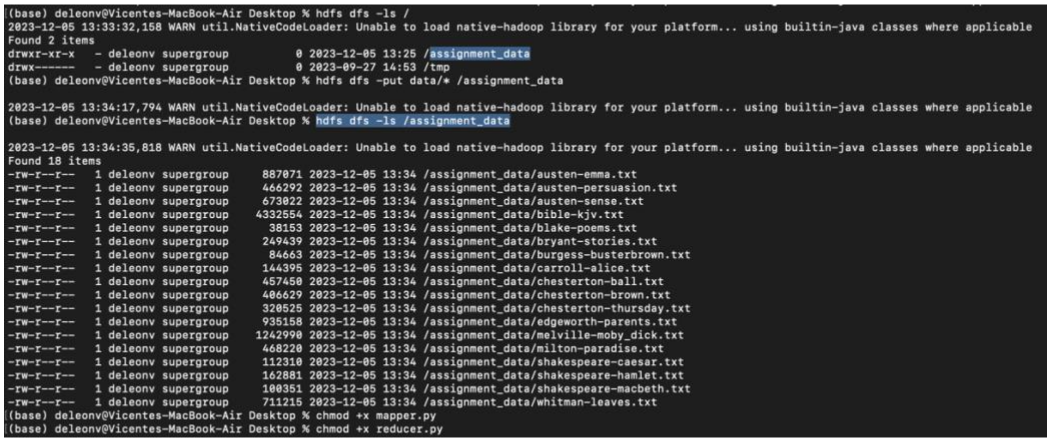
Big Data Applications

December 6, 2023

Final Project

It is important to understand what HDFS is. As we know, HDFS (Hadoop Distributed File System) is not a traditional database, but a distributed file system designed to store and process big data. It allows for storing and processing large datasets across multiple commodity servers. It also provides high-throughput access to data and is optimized for large files transfers, making it suitable solution for big data storage and processing needs. Some of the proven benefits provided by HDFS are the following: scalability (you don't have to worry about running out storage space or risk of data loss), cost effective, and it provides fast data access and processing speeds. For this project, we are going to be using Hadoop to create an inverted index.

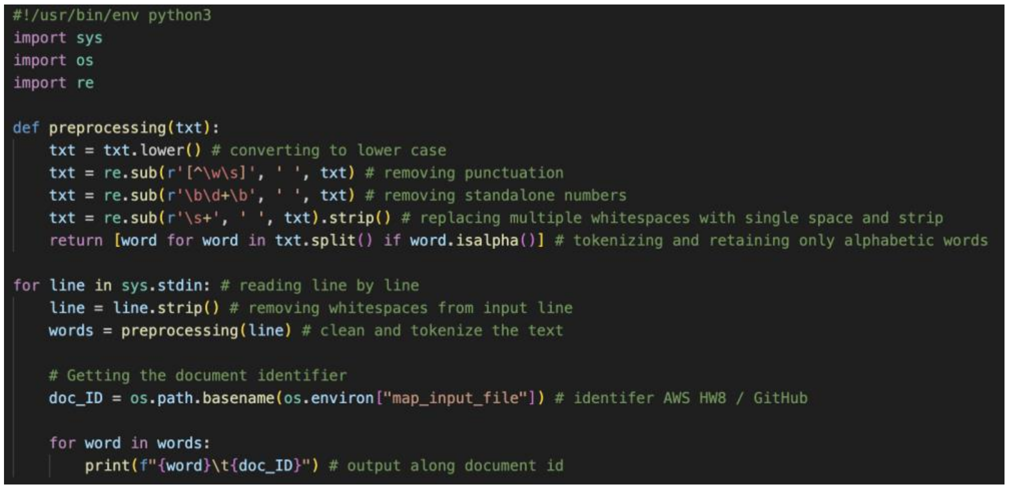
I am using the "data.rar" as the collection of text documents for this assignment. Since I am using MAC, I used "The Unarchiver" to extract the text files. Also, I the HDFS directory "assignment_data" from previous homework was used to store all files from "data.rar". The collection of data within the HDFS directory suits the Inverted index because we are going to be working with a mix of text data such as poems, novels, and plays. This large volume of data shows that the collection is large enough to benefit from distributed processing. As we know, MapReduce does a good job on handling large datasets by splitting the task into smaller pieces. The nature of the data located within the directory is extremely appealing for this kind of work. It will help us leverage the strengths of Hadoop in handling big data while ensuring scalability and processing of test data. The below image shows all data stored within assignment_data HDFS directory:



```
(base) deleonv@Vicentes-MacBook-Air Desktop % hdfs dfs -ls /
2023-12-05 13:33:32,158 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x   - deleonv supergroup          0 2023-12-05 13:25 /assignment_data
drwx------   - deleonv supergroup          0 2023-09-27 14:53 /tmp
(base) deleonv@Vicentes-MacBook-Air Desktop % hdfs dfs -put data/* /assignment_data

2023-12-05 13:34:17,794 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(base) deleonv@Vicentes-MacBook-Air Desktop % hdfs dfs -ls /assignment_data

2023-12-05 13:34:35,818 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 18 items
-rw-r--r--   1 deleonv supergroup     887071 2023-12-05 13:34 /assignment_data/austen-emma.txt
-rw-r--r--   1 deleonv supergroup     466292 2023-12-05 13:34 /assignment_data/austen-persuasion.txt
-rw-r--r--   1 deleonv supergroup     673022 2023-12-05 13:34 /assignment_data/austen-sense.txt
-rw-r--r--   1 deleonv supergroup    4332554 2023-12-05 13:34 /assignment_data/bible-kjv.txt
-rw-r--r--   1 deleonv supergroup      38153 2023-12-05 13:34 /assignment_data/blake-poems.txt
-rw-r--r--   1 deleonv supergroup     249439 2023-12-05 13:34 /assignment_data/bryant-stories.txt
-rw-r--r--   1 deleonv supergroup      84663 2023-12-05 13:34 /assignment_data/burgess-busterbrown.txt
-rw-r--r--   1 deleonv supergroup     144395 2023-12-05 13:34 /assignment_data/carroll-alice.txt
-rw-r--r--   1 deleonv supergroup     457450 2023-12-05 13:34 /assignment_data/chesterton-ball.txt
-rw-r--r--   1 deleonv supergroup     406629 2023-12-05 13:34 /assignment_data/chesterton-brown.txt
-rw-r--r--   1 deleonv supergroup     320525 2023-12-05 13:34 /assignment_data/chesterton-thursday.txt
-rw-r--r--   1 deleonv supergroup     935158 2023-12-05 13:34 /assignment_data/edgeworth-parents.txt
-rw-r--r--   1 deleonv supergroup    1242990 2023-12-05 13:34 /assignment_data/melville-moby_dick.txt
-rw-r--r--   1 deleonv supergroup     468220 2023-12-05 13:34 /assignment_data/milton-paradise.txt
-rw-r--r--   1 deleonv supergroup     112310 2023-12-05 13:34 /assignment_data/shakespeare-caesar.txt
-rw-r--r--   1 deleonv supergroup     162881 2023-12-05 13:34 /assignment_data/shakespeare-hamlet.txt
-rw-r--r--   1 deleonv supergroup     100351 2023-12-05 13:34 /assignment_data/shakespeare-macbeth.txt
-rw-r--r--   1 deleonv supergroup     711215 2023-12-05 13:34 /assignment_data/whitman-leaves.txt
(base) deleonv@Vicentes-MacBook-Air Desktop % chmod +x mapper.py
(base) deleonv@Vicentes-MacBook-Air Desktop % chmod +x reducer.py
```

Inverted index is a data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms. In this scenario, the index is organized by terms (words), and each term points to a list of documents or web pages that contain that term. The goal of this final project is to implement a MapReduce job that takes the input corpus and generates an inverted index. Our output should be a set of key-value pairs, where the key is a term (word), and the value is a list of document identifiers where the term appears.

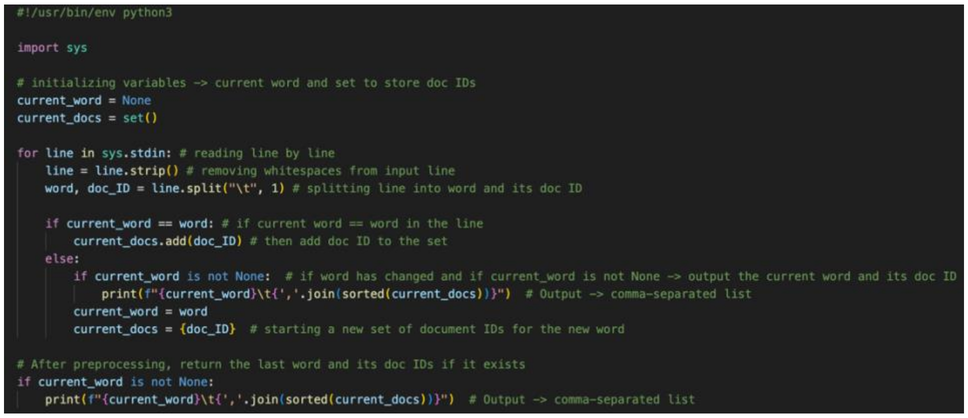
Mapper.py:



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

def preprocessing(txt):
    txt = txt.lower() # converting to lower case
    txt = re.sub(r'[^\w\s]', ' ', txt) # removing punctuation
    txt = re.sub(r'\b\d+\b', ' ', txt) # removing standalone numbers
    txt = re.sub(r'\s+', ' ', txt).strip() # replacing multiple whitespaces with single space and strip
    return [word for word in txt.split() if word.isalpha()] # tokenizing and retaining only alphabetic words

for line in sys.stdin: # reading line by line
    line = line.strip() # removing whitespaces from input line
    words = preprocessing(line) # clean and tokenize the text

    # Getting the document identifier
    doc_ID = os.path.basename(os.environ["map_input_file"]) # identifer AWS HW8 / GitHub

    for word in words:
        print(f"{word}\t{doc_ID}") # output along document id
```

In this case, the mapper.py is straight forward. The preprocessing function takes the text and converts the text to lower case, removes unwanted punctation using regex and replaces it with spaces, removes numbers (whole words that are just digits), replaces whitespaces with a single spaces and trims spaces, and splits the text into words filtering out any tokens that are not alphabetic. The mapper is just a basic combination of preprocessing steps from Intro to NLP course with Professor Olga Scrivner, GitHub resources, and code from previous homework. Regarding the document identifier (GitHub code line) the “map_input_file” suggests that the mapper script is part of the MapReduce job (extracting the base name of the file being processed). The output is just our regular key-value pair separated by a tab character.

Reducer.py:

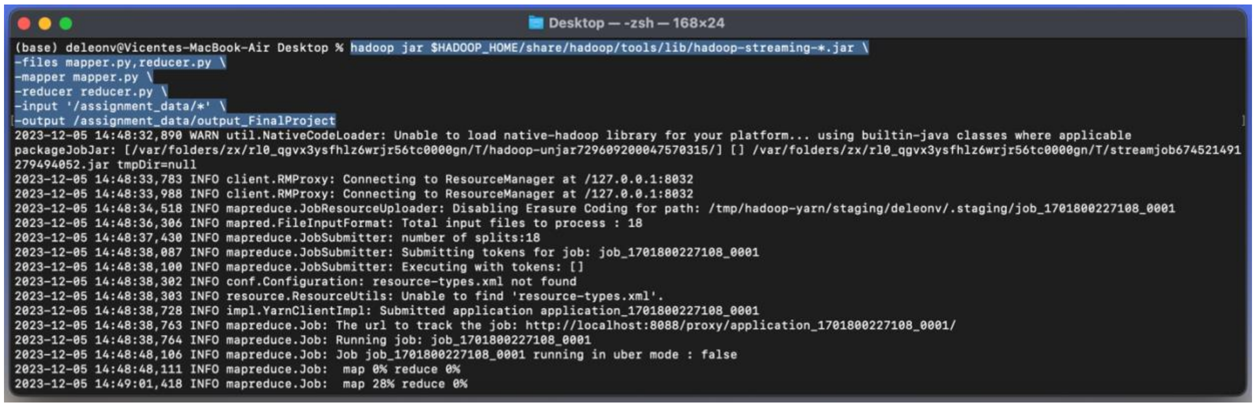


```python
#!/usr/bin/env python3

import sys

# initializing variables -> current word and set to store doc IDs
current_word = None
current_docs = set()

for line in sys.stdin: # reading line by line
    line = line.strip() # removing whitespaces from input line
    word, doc_ID = line.split("\t", 1) # splitting line into word and its doc ID

    if current_word == word: # if current word == word in the line
        current_docs.add(doc_ID) # then add doc ID to the set
    else:
        if current_word is not None:  # if word has changed and if current_word is not None -> output the current word and its doc ID
            print(f"{current_word}\t{','.join(sorted(current_docs))}")  # Output -> comma-separated list
        current_word = word
        current_docs = {doc_ID}  # starting a new set of document IDs for the new word

# After preprocessing, return the last word and its doc IDs if it exists
if current_word is not None:
    print(f"{current_word}\t{','.join(sorted(current_docs))}")  # Output -> comma-separated list
```

As we know, the above reducer is used to summarize and aggregate results from the mapper.py. this script combines the occurrences of each unique word across different documents, resulting in a list of all documents where each word appears. Within the code, the variable “current_word” tracks the word currently being processed and the set “current_docs” stores the doc IDs associated with “current_word”. (It is expected that each line contains a word and a doc ID separated with a tab characters.) Also, if the incoming word is the same as “current_word” then it adds the doc ID to “current_docs”.

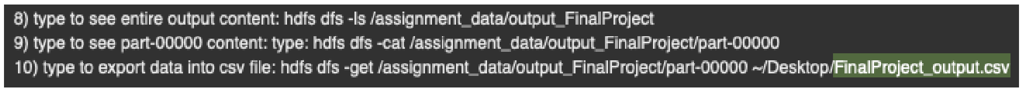
If the incoming word id different, then then it outputs the “current_word” along with the sorted list of collected doc IDs and it resets “currentz_word” and “current_docs” for the new word. As final output, for each unique word the reducer outputs the word followed by a list of doc IDs where the word appears.

Running the MapReduce Job:



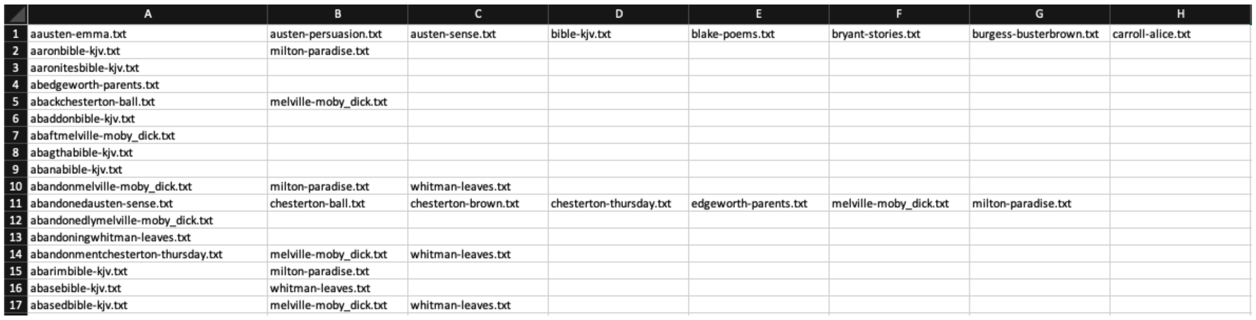
```
(base) deleonv@Vicentes-MacBook-Air Desktop % hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input '/assignment_data/*' \
-output /assignment_data/output_FinalProject
2023-12-05 14:48:32,890 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
packageJobJar: [/var/folders/zx/rl0_qgvx3ysfhlz6wrjr56tc0000gn/T/hadoop-unjar7296092000475703l5/] [] /var/folders/zx/rl0_qgvx3ysfhlz6wrjr56tc0000gn/T/streamjob674521491
279494052.jar tmpDir=null
2023-12-05 14:48:33,783 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2023-12-05 14:48:33,988 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2023-12-05 14:48:34,518 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/deleonv/.staging/job_1701800227108_0001
2023-12-05 14:48:36,306 INFO mapred.FileInputFormat: Total input files to process : 18
2023-12-05 14:48:37,430 INFO mapreduce.JobSubmitter: number of splits:18
2023-12-05 14:48:38,087 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1701800227108_0001
2023-12-05 14:48:38,100 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-12-05 14:48:38,302 INFO conf.Configuration: resource-types.xml not found
2023-12-05 14:48:38,303 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-12-05 14:48:38,728 INFO impl.YarnClientImpl: Submitted application application_1701800227108_0001
2023-12-05 14:48:38,763 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1701800227108_0001/
2023-12-05 14:48:38,764 INFO mapreduce.Job: Running job: job_1701800227108_0001
2023-12-05 14:48:48,106 INFO mapreduce.Job: Job job_1701800227108_0001 running in uber mode : false
2023-12-05 14:48:48,111 INFO mapreduce.Job:  map 0% reduce 0%
2023-12-05 14:49:01,418 INFO mapreduce.Job:  map 28% reduce 0%
```

Just like previous homework, I used the following command to store results as csv file:



```
8) type to see entire output content: hdfs dfs -ls /assignment_data/output_FinalProject
9) type to see part-00000 content: type: hdfs dfs -cat /assignment_data/output_FinalProject/part-00000
10) type to export data into csv file: hdfs dfs -get /assignment_data/output_FinalProject/part-00000 ~/Desktop/FinalProject_output.csv
```

Some random lines out of the FinalProject_output.csv:



	A	B	C	D	E	F	G	H
1	aausten-emma.txt	austen-persuasion.txt	austen-sense.txt	bible-kjv.txt	blake-poems.txt	bryant-stories.txt	burgess-busterbrown.txt	carroll-alice.txt
2	aaronbible-kjv.txt	milton-paradise.txt						
3	aaronitesbible-kjv.txt							
4	abedgeworth-parents.txt							
5	abackchesterton-ball.txt	melville-moby_dick.txt						
6	abaddonbible-kjv.txt							
7	abaftmelville-moby_dick.txt							
8	abagthabible-kjv.txt							
9	abanabible-kjv.txt							
10	abandonmelville-moby_dick.txt	milton-paradise.txt	whitman-leaves.txt					
11	abandonedausten-sense.txt	chesterton-ball.txt	chesterton-brown.txt	chesterton-thursday.txt	edgeworth-parents.txt	melville-moby_dick.txt	milton-paradise.txt	
12	abandonedlymelville-moby_dick.txt							
13	abandoningwhitman-leaves.txt							
14	abandonmentchesterton-thursday.txt	melville-moby_dick.txt	whitman-leaves.txt					
15	abarimbible-kjv.txt	milton-paradise.txt						
16	abasebible-kjv.txt	whitman-leaves.txt						
17	abasedbible-kjv.txt	melville-moby_dick.txt	whitman-leaves.txt					

The above csv file will be included within the final submission.

The results show that the preprocessing steps of tokenization, lower case, and removing unwanted punctation worked. The words are lowercase, there’s no punctuation, and the words are separated and identifiable. Within the file, each term is followed by a tab character and then a list of document identifiers (which are separated by commas). The key represents a term from the corpus, and the value is a list of document identifiers where the term was found.

Exploring ways to optimize data storage and retrieval performance in HDFS:

The following are some ways to optimize data storage and retrieval performance in HDFS:

- Cluster Sizing: we know that clusters can boost the processing speed of many big data jobs. However, size should be determine based on the expected data size, the number of concurrent users, and the data access patterns. It is important to determine the right cluster size based on the above features. If a cluster is too big this can be cost-ineffective. In the other hand, if the cluster is too small, it may not handle data very well leading to more bigger problems.
- Block Size: This scenario can also impact performance (the default block size is 128 MB). Larger block sizes allow for more efficient data retrieval but in higher overhead for data replication and management. If the blocks are small, there will be too many blocks in HDFS and thus too much metadata to store. Managing huge numbers can lead to traffic in a network.
- Data replication: this basically determines how many copies of each block of data are stored within the cluster. The balance of data availability and performance. Basically, increasing replication improves fault tolerance but consumes more storage space and network.
- Disk I/O: performance can be impacted by this area. It is important to use fast disks and to ensure that disks are not overutilized. Y buffer in data into memory of disk, I/O operations can be aggregated into larger chunks, which can improve throughput and reduce network overhead.
- Namenode Memory: this is the master node that manages metadata about the file system. It is important to allocate enough memory to the Namenode to ensure efficient data management.
- Data partitioning: This is also a useful technique for optimizing performance. splitting large datasets into smaller pieces can help improve parallelism and reduce processing times for large datasets.
- We can also add nodes to the clusters, which can help ensure that resources are available to handle increasing demand. Also, replicating data effectively can improve performance and ensure that data is available even if a node fails.

Besides the above tunning techniques, we can also mention the data locality scenarios. Data locality means moving computation close to data rather than moving data towards computation (this helps in high throughput and faster execution of data). Since MapReduce jobs work on blocks from HDFS and are data-parallel, data locality is important for better performance and faster execution of the MapReduce jobs. Data compression might be another great technique that reduces spaces for storing files and speeds up data transfer across network or disk. The compression scenario is very high, hence we save a lot of I/O.

Conclusion:

We can safely say that the data was extremely useful for inverted index due to its own nature. Having different text files, with different sizes and genres, makes it an ideal candidate for this kind of job. The results show that the processing, mapper, and reducer worked without any trouble. Things I wish to potentially see in the future regarding the subject matter is perhaps implementing some kind of machine learning algorithms within the Hadoop environment to enhance maybe search accuracy. Also, extending the MapReduce job to support more complex and technical jobs.

References:

Hadoop: https://www.geeksforgeeks.org/hadoop-an-introduction/

Reading all files from HDFS directory: https://stackoverflow.com/questions/42484606/how-to-read-files-in-hdfs-directory-using-python

Inverted index: https://www.geeksforgeeks.org/create-inverted-index-for-file-using-python/

Inverted index: https://pmatigakis.wordpress.com/2011/09/14/using-python-and-hadoop-streaming-to-build-an-inverted-index/

Inverted index: https://www.geeksforgeeks.org/inverted-index/

Inverted_inder_mapper.py: https://gist.github.com/tori-takashi/a5431dde0df603dc260da8d2b35e0ac1

Regular expressions: https://docs.python.org/3/library/re.html

Isalpha(): https://www.geeksforgeeks.org/python-string-isalpha-method/

Regex: https://pynative.com/python-regex-replace-re-sub/

Regex: https://www.geeksforgeeks.org/python-substituting-patterns-in-text-using-regex/

Add(): https://www.geeksforgeeks.org/set-add-python/

HDFS Big Data: https://www.analyticsvidhya.com/blog/2023/02/a-dive-into-the-basics-of-big-data-storage-with-hdfs/

Cluster Size: https://www.databricks.com/glossary/hadoop-cluster

Block size: https://data-flair.training/blogs/data-block/#

HDFS performance: https://www.linkedin.com/pulse/how-optimize-hdfs-performance-large-scale-data-soumyadeep-mandal/

Data locality: https://data-flair.training/forums/topic/what-is-data-locality-in-hadoop/#:~:text=Data%20Locality%20in%20Hadoop%20means,map%20jobs%20and%20reduce%20jobs
.

Data compression: http://www.hadoopadmin.co.in/hdfs/compression-in-hadoop/