MAP REDUCE ASSIGNMENT REPORT

Mahlet Nigussie Tesfaye
Addis Ababa Institute of Technology, SiTE
rigbe.rmn@gmail.com
Addis Ababa, Ethiopia.
February 10- 2024

1 INTRODUCTION

This report documents the implementation and analysis of a MapReduce-based word count application using Hadoop.

MapReduce is a programming model and distributed computing framework crucial for processing large datasets in parallel [2]. It enables efficient, scalable data analysis across a cluster of machines. The MapReduce pattern allows you to divide problems into a series of independent, equivalent tasks that can be parallelized.

The task involves writing a **word count program** using Hadoop to analyze the frequency of words across 10 books from Project Gutenberg. This assignment demonstrates the fundamental principles of MapReduce and its application to text processing [1][3].

The objectives of this assignment are to:

1. Implement a MapReduce program to count word frequencies in books from Project Gutenberg.
2. Extend the program to filter out stopwords and analyze the impact of this filtering on performance and data characteristics.
3. Answer specific questions about the data flow and performance of the MapReduce job to understand the scalability of the framework.

The experiment was conducted using **Hadoop 3.4.1** in a pseudo-distributed mode. The input dataset consisted of 10 books downloaded from Project Gutenberg, and the program was extended to handle a list of common stopwords.

2 METHODOLOGY

2.1 SETUP AND CONFIGURATION

- **Hadoop Version:** Hadoop 3.4.1 was installed and configured in pseudo-distributed mode following the official Apache Hadoop documentation. Key configurations included setting `mapreduce.framework.name` to `yarn` and specifying `yarn.resourcemanager.hostname` to `localhost`.
- **Java Setup:** Java SDK 1.8 was installed and configured as the runtime environment. The `JAVA_HOME` environment variable was set to the installation directory of the JDK, and the `PATH` variable was updated to include the Java binaries.
- **Input Dataset:** The following 10 books were downloaded from Project Gutenberg in plain text format:

- `alice_in_wonderland.txt` (160 KB)
- `dracula.txt` (850 KB)
- `frankenstein.txt` (420 KB)
- `great_expectations.txt` (1.1 MB)
- `moby_dick.txt` (2.1 MB)
- `pride_and_prejudice.txt` (700 KB)
- `sherlock_holmes.txt` (1.6 MB)
- `the_adventures_of_tom_sawyer.txt` (350 KB)
- `ulysses.txt` (1.5 MB)
- `war_and_peace.txt` (3.2 MB)

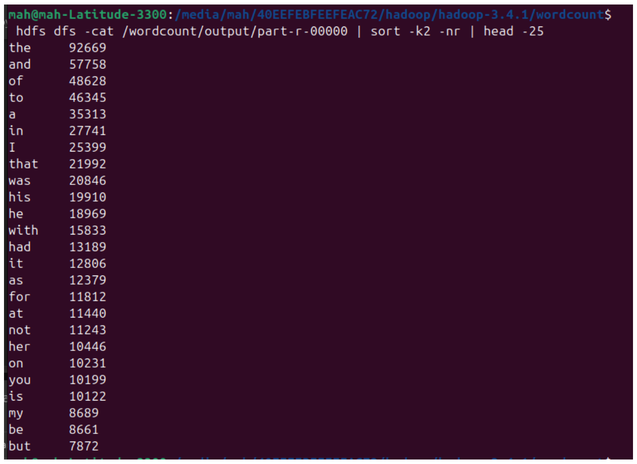
2.2 Word Count Application

- **Basic Word Count:**
  - **Mapper:** The `TokenizerMapper` class extends `MapReduceBase` and implements the `Mapper` interface. The `map` method tokenizes the input text into words using `StringTokenizer`. For each word, it emits a key-value pair where the word is the key and '1' is the value, indicating one occurrence.
  - **Reducer:** The `IntSumReducer` class extends `MapReduceBase` and implements the `Reducer` interface. The `reduce` method receives a word and a list of counts (all '1's from the mapper output). It sums these counts to determine the total number of occurrences for each word.
- **Stopwords Removal:** To filter out stopwords, the application was modified to read a list of common stopwords from a file into a HashSet. The map method checks if each tokenized word is present in this HashSet; if it is, that word is skipped [1].
- **Execution:** The program was compiled into a JAR files and executed on the Hadoop cluster using the command line interface with specified input and output paths.

3 Results and Analysis

3.1 Word Count Without Stopwords

Analysis: The results show that common words such as "the", "and", "of", "to", "a", "in", etc., dominate the top counts. This indicates that many high-frequency words are stopwords that do not contribute significant semantic meaning to text analysis [1].

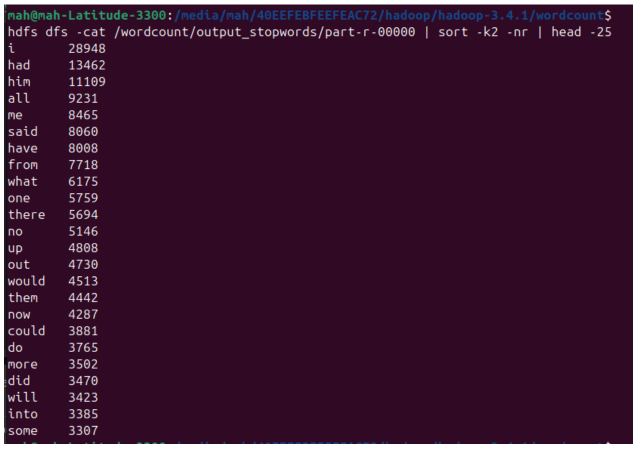


```
mah@mah-Latitude-3300:/media/mah/40EEFEBFEEFEAC72/hadoop/hadoop-3.4.1/wordcount$
 hdfs dfs -cat /wordcount/output/part-r-00000 | sort -k2 -nr | head -25
the     92669
and     57758
of      48628
to      46345
a       35313
in      27741
I       25399
that    21992
was     20846
his     19910
he      18969
with    15833
had     13189
it      12806
as      12379
for     11812
at      11440
not     11243
her     10446
on      10231
you     10199
is      10122
my      8689
be      8661
but     7872
```

Figure 1: Top 25 Words Without Stopwords

3.2 Word Count With Stopwords Removed

Analysis: After removing stopwords, the top words reflect more meaningful content, highlighting nouns, verbs, and pronouns that are central to narratives within these books. This demonstrates that filtering out stopwords improves both relevance and clarity in frequency analysis [1].



```
mah@mah-Latitude-3300:/media/mah/40EEFEBFEEFEAC72/hadoop/hadoop-3.4.1/wordcount$
hdfs dfs -cat /wordcount/output_stopwords/part-r-00000 | sort -k2 -nr | head -25
i       28948
had     13462
him     11109
all     9231
me      8465
said    8060
have    8008
from    7718
what    6175
one     5759
there   5694
no      5146
up      4808
out     4730
would   4513
them    4442
now     4287
could   3881
do      3765
more    3502
did     3470
will    3423
into    3385
some    3307
```

Figure 2: Top 25 Words With Stopwords Removed

3.3 Impact of Removing Stopwords

- The number of outputs by mappers decreased significantly after removing stopwords:
  - Without stopwords: 122,781
  - With stopwords: 56,624

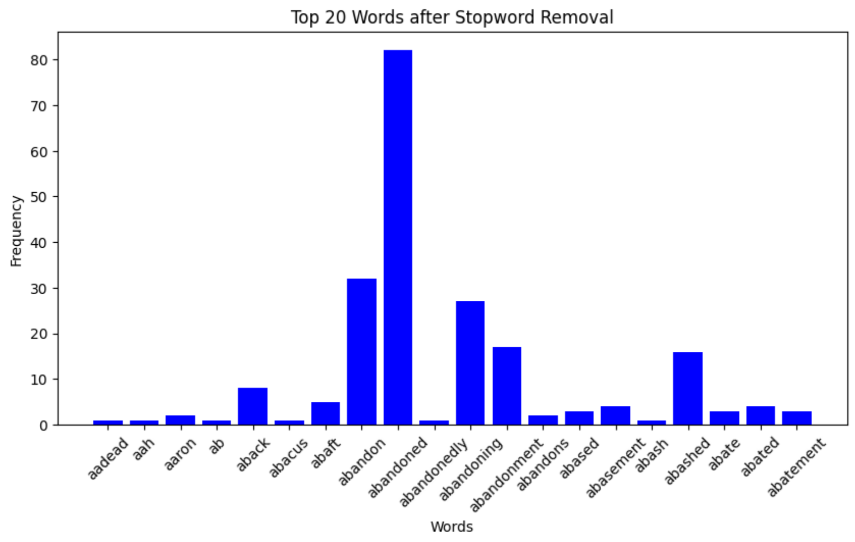
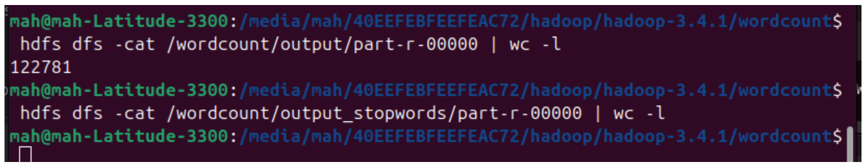


Figure 3: Top 25 Words With Stopwords Removed



```
mah@mah-Latitude-3300:/media/mah/40EEFEBFEEFEAC72/hadoop/hadoop-3.4.1/wordcount$
 hdfs dfs -cat /wordcount/output/part-r-00000 | wc -l
122781
mah@mah-Latitude-3300:/media/mah/40EEFEBFEEFEAC72/hadoop/hadoop-3.4.1/wordcount$
 hdfs dfs -cat /wordcount/output_stopwords/part-r-00000 | wc -l
mah@mah-Latitude-3300:/media/mah/40EEFEBFEEFEAC72/hadoop/hadoop-3.4.1/wordcount$
```

Figure 4: number of key values

This represents approximately a 54% reduction in key-value pairs produced, Removing stopwords reduces the amount of data shuffled between mappers and reducers. This lowers network overhead and speeds up the job, as fewer key-value pairs need to be processed and transferred.

3.4 Keyspace Analysis

- The unique keys were also significantly reduced:
  - Without Stopwords: 122,781 unique keys.
  - With Stopwords: 56,624 unique keys.

This reduction indicates that many common variants of words were also removed along with explicit stopword filtering, demonstrating an efficient reduction in keyspace size due to case insensitivity in counting.

3.5 Scaling Word Count for Project Gutenberg

Assuming we are processing the entirety of Project Gutenberg with the given constraints:

- 100TB of input data
- Data is spread over 10 sites

- Each site has 20 mappers
- We track only the top 25 most common words
- Combiners are optimally reducing data, outputting at most 1 key-value pair per key
- One reducer per site

Mapper Workload

Each of the 10 sites has 20 mappers, giving a total of:

$$\text{Total mappers} = 10 \times 20 = 200 \tag{1}$$

The data is evenly distributed across these mappers, so each mapper processes:

$$\text{Data per mapper} = \frac{100TB}{200} = 0.5TB \tag{2}$$

Size of Keyspace

Since we are only tracking the 25 most common words, the total keyspace consists of:

$$25 \text{ unique keys} \tag{3}$$

Maximum Key-Value Pairs Communicated

Each mapper outputs at most one key-value pair per word (after combining). With 200 mappers:

$$\text{Total key-value pairs} = 200 \times 25 = 5000 \tag{4}$$

Thus, at most, 5000 key-value pairs are communicated between the mapping and reducing phases.

Reducer Workload

Since there is one reducer per site and there are 10 sites, we have 10 reducers. The total number of key-value pairs is distributed across these reducers:

$$\text{Key-value pairs per reducer} = \frac{5000}{10} = 500 \tag{5}$$

Each reducer, on average, will process 500 key-value pairs, Given the assumptions, our MapReduce job will distribute the workload efficiently across the 200 mappers and 10 reducers, ensuring scalability for processing large-scale text datasets like Project Gutenberg. Here is the the data flow diagram:

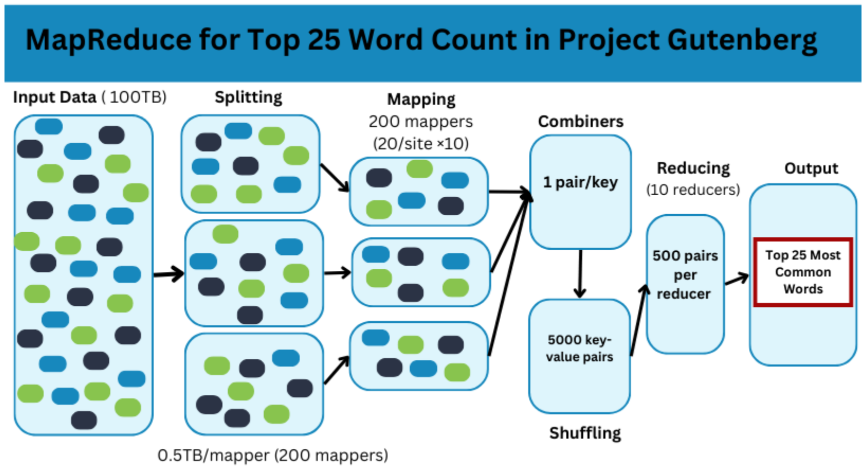


Figure 5: Data Flow Diagram

4 Conclusion

This experiment demonstrated the effectiveness of **MapReduce** for large-scale text processing tasks like word counting. By filtering out stopwords, we significantly reduced both keyspace size and overall output volume while improving performance metrics associated with data transfer between mappers and reducers. The results highlight how preprocessing steps such as stopword removal can enhance analytical quality in natural language processing tasks.

5 References

1. Apache Hadoop Documentation: `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html`
2. MapReduce Tutorial: `https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`
3. MapReduce. Assignment 1. - HackMD: `https://hackmd.io/s/H1LM2fR5m`