TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Lab Report
On
Implementation and Analysis of Hadoop Framework for Big Data Processing

Submitted By

Aadarsha Khadka (THA077BEI001)

Submitted To

Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

February 2025

Problem

Count the frequency of words in a given plain text file using hadoop framework.

Problem Definition

The objective of this project is to count the frequency of words in a given plain text file using the Hadoop framework. The input is a text file stored in the Hadoop Distributed File System (HDFS), and the program utilizes the MapReduce model to compute word occurrences efficiently. This demonstrates how Hadoop can be used to process large datasets in a distributed manner.

Introduction

Big Data processing is a critical challenge in modern computing. Apache Hadoop, an open-source framework, provides an efficient way to store and process massive datasets through HDFS and MapReduce. This report focuses on the MapReduce programming model in distributed computing.

Hadoop MapReduce programming model facilitates the processing of big data stored on HDFS.

By using the resources of multiple interconnected machines, MapReduce effectively handles a large amount of structured and unstructured data.

Hadoop MapReduce Architecture

Hadoop MapReduce is a programming model and software framework designed for processing large datasets in parallel across distributed clusters of computers. It is a core component of the Apache Hadoop ecosystem, enabling scalable and fault-tolerant data processing. The MapReduce architecture consists of two main phases: the **Map phase** and the **Reduce phase**, which are executed in a distributed manner across the cluster.

Key Components of Hadoop MapReduce Architecture are :

1. JobTracker

The JobTracker is the master node in the MapReduce architecture. It manages the distribution of tasks (Map and Reduce) across the cluster. It monitors the progress of tasks, handles failures, and re-executes failed tasks.

2. TaskTracker

The TaskTracker runs on each worker node in the cluster. It executes the Map and Reduce tasks assigned by the JobTracker. It sends periodic progress reports to the JobTracker.

3. Map Phase

In the Map phase, input data is split into smaller chunks called input splits. Each split is processed by a Mapper function, which generates intermediate key-value pairs. The output of the Mapper is stored locally on the worker node.

4. Shuffle and Sort Phase

After the Map phase, the intermediate key-value pairs are shuffled and sorted by key. This ensures that all values associated with the same key are grouped together and sent to the same Reducer.

5. Reduce Phase

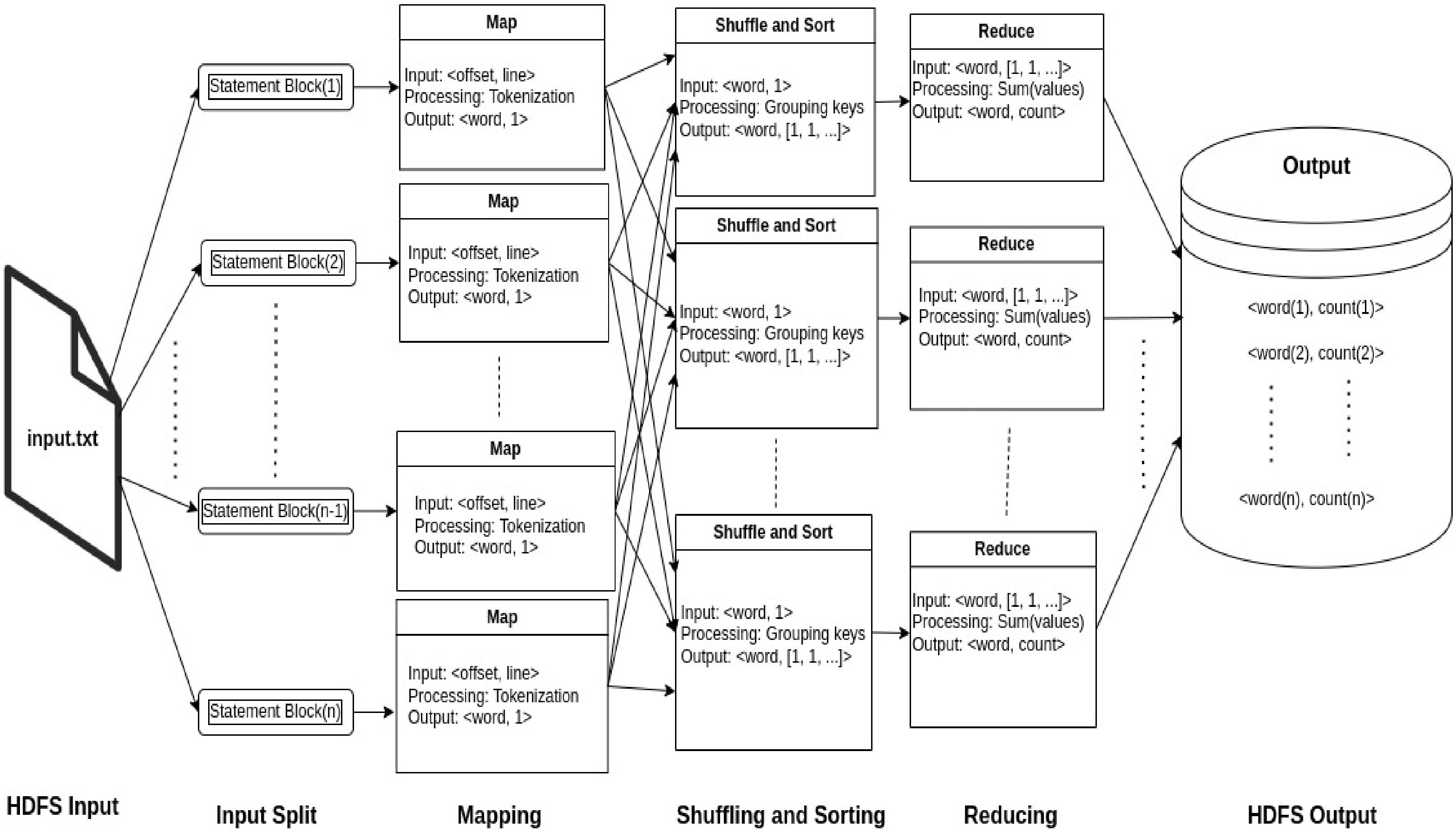
In the Reduce phase, the Reducer function processes the grouped key-value pairs. The Reducer aggregates, summarizes, or transforms the data to produce the final output. The output is stored in the Hadoop Distributed File System (HDFS).

6. HDFS (Hadoop Distributed File System)

HDFS is the storage layer used by MapReduce to store input data and output results. It provides high-throughput access to data and ensures fault tolerance by replicating data across multiple nodes.

However, it is important to note that Hadoop 2.x and later versions introduced significant changes to the architecture, particularly with the introduction of YARN (Yet Another Resource Negotiator).

Workflow of MapReduce



1. Input Splitting

The input file is divided into multiple chunks called input splits, which are assigned to individual mappers for parallel processing.

2. Mapping

Each Mapper processes an input split and outputs intermediate key-value pairs. For example, in a word count program:

Input: <offset, line>

Processing: Tokenization

Output: <word, 1>

3. Shuffling and Sorting

The intermediate key-value pairs from all Mappers are shuffled and sorted before being sent to the Reducers. The same keys from different Mappers are grouped together. The output is prepared for the Reduce phase.

Example:

Input: <word, 1>

Processing: Grouping all occurrences of the same word

Output: <word, [1, 1, 1, ...]>

4. Reducing

The grouped key-value pairs are processed by Reducers. The Reducer applies an aggregation function (such as sum, average, max, etc.). The final output is produced.

Example:

Input: <word, [1, 1, 1, ...]>

Processing: Summing the occurrences

Output: <word, count>

5. Output Storage

The final output is written to HDFS in a structured format. The number of output files corresponds to the number of Reducers. The output can be further used for data analysis, visualization, or reporting.

Source Code

The full implementation of the Word Count program is provided in the attached file.
Filename: *WordCount.java*

It contains the complete Hadoop MapReduce program to count word frequencies in a given text file.

Input File

In addition to the plain text *input.txt* file, a CSV file *input_data.csv* is also included in the assignment folder. This file represents structured data.

Input Data (Table for CSV format)

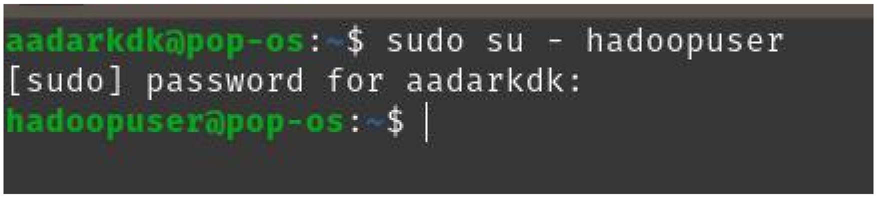
Sentence ID	Sentence
1	The term “big data” refers to data that is so large, fast or complex that it’s difficult or impossible to process using traditional methods.
2	The act of accessing and storing large amounts of information for analytics has been around a long time.
3	But the concept of big data gained momentum in the early 2000s when industry analyst Doug Laney articulated the now-mainstream definition of big data as the three V’s:
4	Volume: Organizations collect data from a variety of sources, including business transactions, smart (IoT) devices, industrial equipment, videos, social media and more.
5	In the past, storing it would have been a problem – but cheaper storage on platforms like data lakes and Hadoop have eased the burden.
6	Velocity: With the growth in the Internet of Things, data streams in to businesses at an unprecedented speed and must be handled in a timely manner.
7	RFID tags, sensors and smart meters are driving the need to deal with these torrents of data in near-real time.
8	Variety: Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, emails, videos, audios, stock ticker data and financial transactions.
9	In addition to the increasing velocities and varieties of data, data flows are unpredictable – changing often and varying greatly.
10	It’s challenging, but businesses need to know when something is trending in social media, and how to manage daily, seasonal and event-triggered peak data loads.
11	Veracity refers to the quality of data.
12	Because data comes from so many different sources, it’s difficult to link, match, cleanse and transform data across systems.
13	Businesses need to connect and correlate relationships, hierarchies and multiple data linkages.
14	Otherwise, their data can quickly spiral out of control.

Steps To Run Program

1. Change user

Command: `sudo su - hadoopuser`

Output:



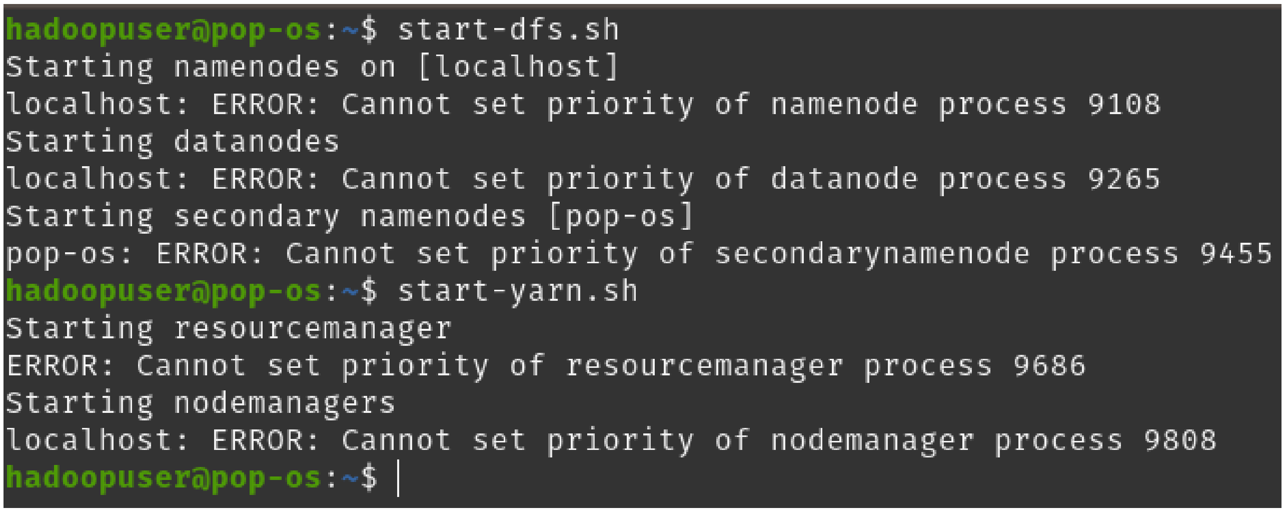
```
aadarkdk@pop-os:~$ sudo su - hadoopuser
[sudo] password for aadarkdk:
hadoopuser@pop-os:~$ 
```

2. Start Hadoop Services

Command:

```
start-dfs.sh
start-yarn.sh
```

Output:



```
hadoopuser@pop-os:~$ start-dfs.sh
Starting namenodes on [localhost]
localhost: ERROR: Cannot set priority of namenode process 9108
Starting datanodes
localhost: ERROR: Cannot set priority of datanode process 9265
Starting secondary namenodes [pop-os]
pop-os: ERROR: Cannot set priority of secondarynamenode process 9455
hadoopuser@pop-os:~$ start-yarn.sh
Starting resourcemanager
ERROR: Cannot set priority of resourcemanager process 9686
Starting nodemanagers
localhost: ERROR: Cannot set priority of nodemanager process 9808
hadoopuser@pop-os:~$ 
```

3. Upload Input File to HDFS

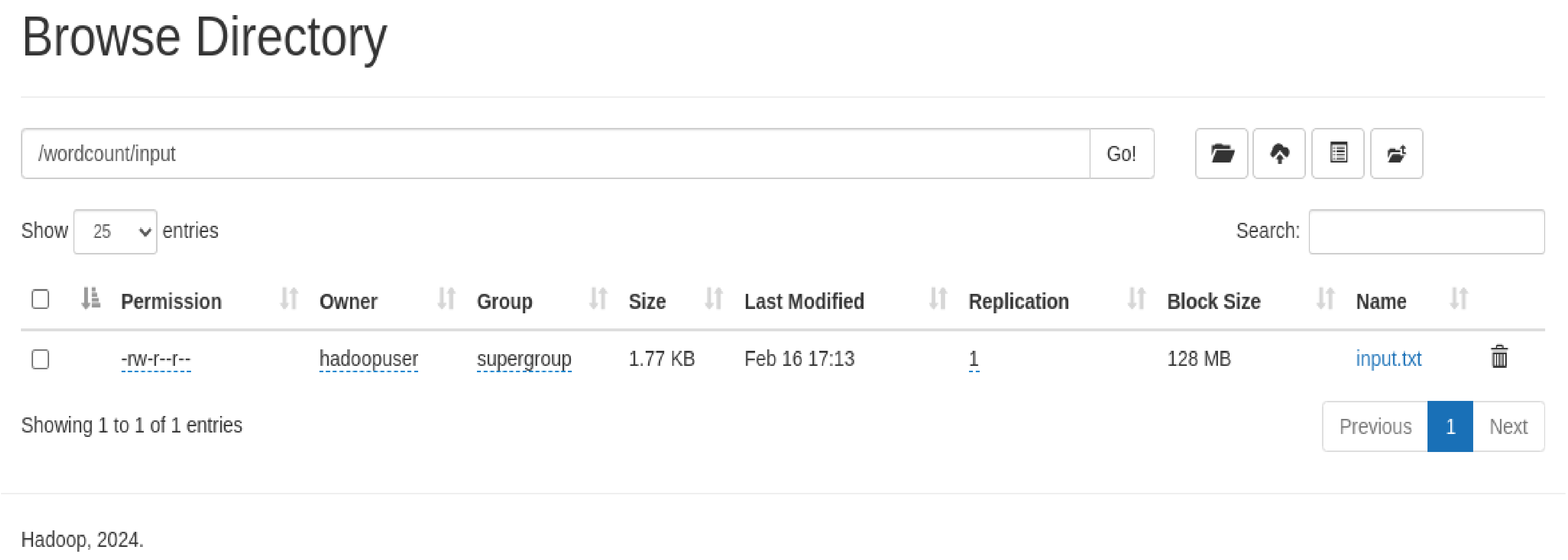
Create Input directory in HDFS.

Command:
```
hdfs dfs -mkdir /wordcount/input
```

Change directory to the directory where input.txt file is located. Then upload input.txt file to HDFS.

Command:
```
hdfs dfs -put input.txt /wordcount/input
```

Output:



Browse Directory

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	hadoopuser	supergroup	1.77 KB	Feb 16 17:13	1	128 MB	input.txt	🗑

Showing 1 to 1 of 1 entries

Hadoop, 2024.

4. Compile the Java Program

Command:
```
javac -classpath `hadoop classpath` -d .    WordCount.java
```

5. Create a JAR file

Command:
```
jar -cvf wordcount.jar -C . .
```

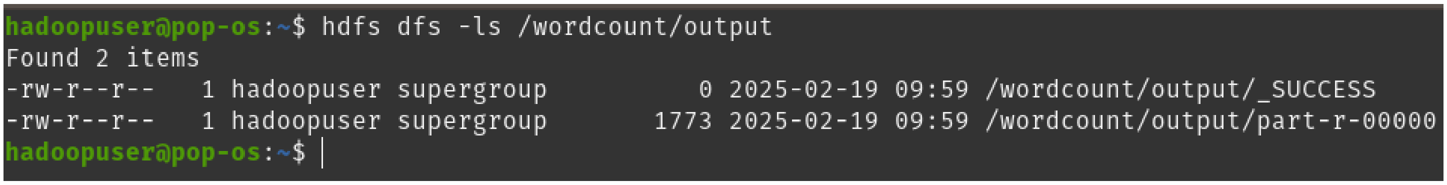
6. Run the WordCount Program

Command:
```
hadoop jar wordcount.jar WordCount /wordcount/input
/wordcount/output
```

7. Output file

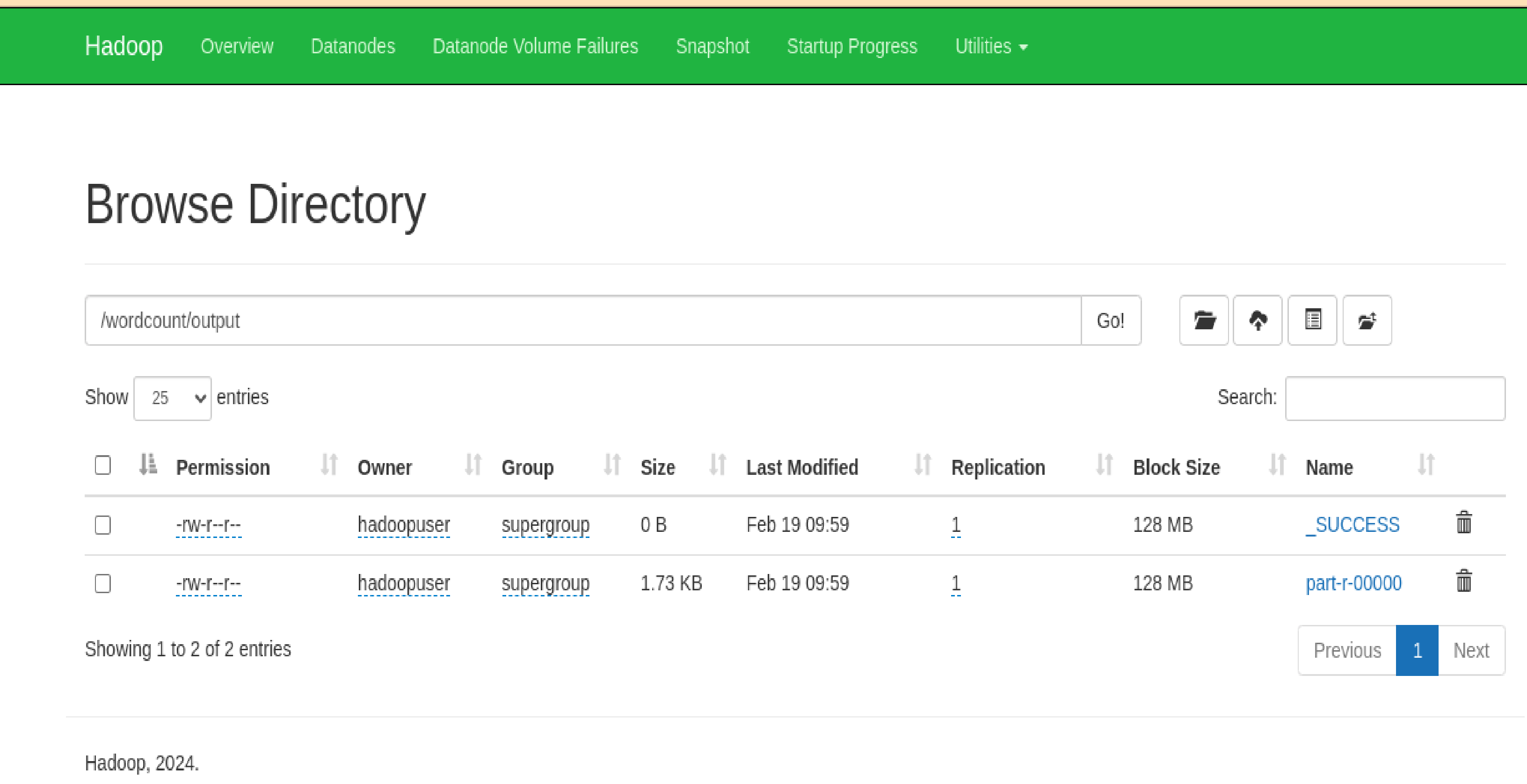
The output file is named as *part-r-00000*.

Command: `hdfs dfs -ls /wordcount/output`



```
hadoopuser@pop-os:~$ hdfs dfs -ls /wordcount/output
Found 2 items
-rw-r--r--   1 hadoopuser supergroup          0 2025-02-19 09:59 /wordcount/output/_SUCCESS
-rw-r--r--   1 hadoopuser supergroup       1773 2025-02-19 09:59 /wordcount/output/part-r-00000
hadoopuser@pop-os:~$ 
```

Output:



Browse Directory

/wordcount/output Go!

Show 25 entries

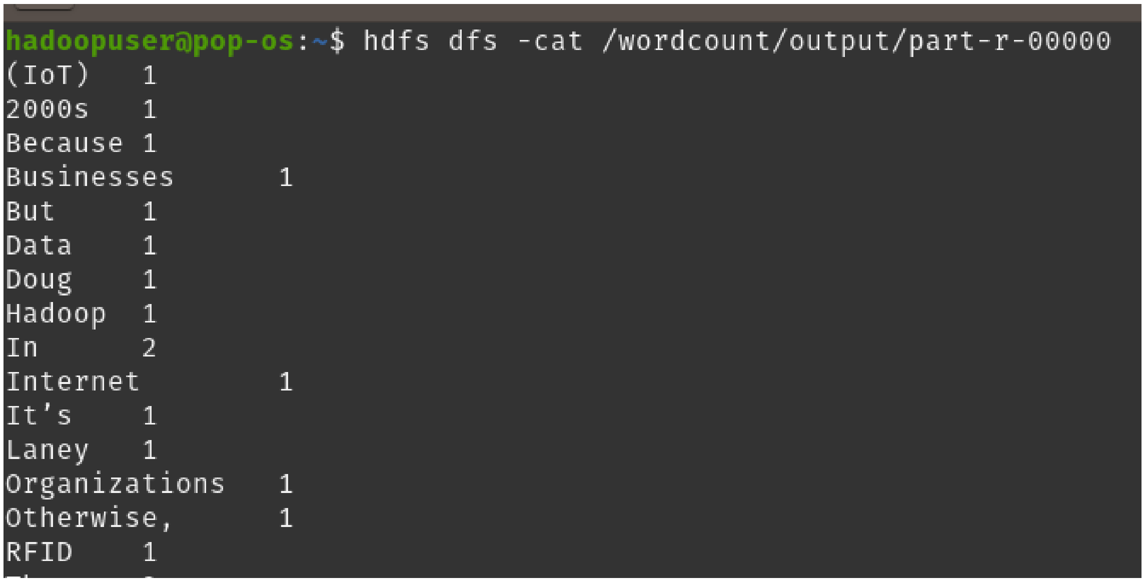
Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
□	-rw-r--r--	hadoopuser	supergroup	0 B	Feb 19 09:59	1	128 MB	_SUCCESS	🗑
□	-rw-r--r--	hadoopuser	supergroup	1.73 KB	Feb 19 09:59	1	128 MB	part-r-00000	🗑

Showing 1 to 2 of 2 entries

Hadoop, 2024.

Sample output:



```
hadoopuser@pop-os:~$ hdfs dfs -cat /wordcount/output/part-r-00000
(IoT)   1
2000s   1
Because 1
Businesses      1
But     1
Data    1
Doug    1
Hadoop  1
In      2
Internet        1
It’s    1
Laney   1
Organizations   1
Otherwise,      1
RFID    1
```

Discussion and Conclusion

The experiment successfully demonstrated the execution of the MapReduce framework for word frequency counting. The implementation showed how data is split, processed, and aggregated efficiently in a distributed manner. The use of Hadoop’s HDFS and MapReduce model proved effective for handling large datasets. Additionally, the parallel execution in a cluster environment improved performance over traditional single-node processing.

The results indicate that Hadoop is a powerful tool for processing large-scale data with minimal effort. However, some challenges were encountered, such as managing Hadoop services and ensuring correct file permissions in HDFS. These issues highlight the need for proper Hadoop cluster configuration and job monitoring to achieve optimal performance.

Conclusion

This lab demonstrated the implementation of a word count program using the Hadoop MapReduce framework. The results highlight how distributed computing effectively processes large datasets by splitting, mapping, and reducing data in parallel. The Hadoop ecosystem provides an efficient and scalable solution for handling big data problems in real-world applications.