

**DEPARTMENT OF  
UNIVERSITY INSTITUTE OF COMPUTING  
CHANDIGARH UNIVERSITY**



**PROJECT REPORT**

**ON**

**WORD COUNT**

Subject Name: BIG DATA

Subject Code: 24CAT-683

Submitted by:

Name: Praphul Kumar

Class/Section: 24MCD-1-A

UID:24MCC20072

Submitted to:

Name: Mr. Rishabh Tomar

## 1. Introduction

In the modern era of data-driven decision-making, processing large volumes of data efficiently has become a critical challenge. Big Data refers to massive datasets that require specialized tools and techniques for storage, processing, and analysis. One of the most widely used frameworks for handling such large-scale data is **Hadoop**, an open-source distributed computing platform. Within Hadoop, **MapReduce** serves as a powerful programming model that enables parallel processing of large datasets across multiple nodes in a cluster.

This project focuses on implementing the **Word Count program**, a fundamental example of Hadoop MapReduce that showcases its ability to process text data efficiently. The Word Count program is a simple yet effective demonstration of how MapReduce works by breaking down large text files into individual words and counting their occurrences in a distributed computing environment.

The project provides a **step-by-step guide** to setting up Hadoop, writing the MapReduce program, executing it on a Hadoop cluster, and analyzing the output. The implementation is carried out in a **Linux environment**, which is commonly used for Hadoop deployments due to its stability and compatibility with distributed computing frameworks.

Through this project, we aim to:

1. **Understand the Hadoop ecosystem** and its role in Big Data processing.
2. **Learn the MapReduce programming model** and its working mechanism.
3. **Implement and execute the Word Count program** on a Hadoop cluster.
4. **Analyze the results** and gain insights into distributed data processing.

By the end of this project, we will have a **practical understanding of Hadoop MapReduce**, its advantages, and how it can be leveraged to process large-scale textual data efficiently. This knowledge serves as a foundation for more complex Big Data analytics applications in real-world scenarios.

## 2. Objective

The main objective of this project is to understand and implement Hadoop MapReduce for processing large-scale data. Hadoop MapReduce is a powerful framework that helps process big data efficiently by breaking down tasks into smaller ones that can be handled in parallel across multiple machines. The goal of this project is to use a basic example, the **Word Count program**, to demonstrate the capabilities of MapReduce for processing and analyzing text data.

First, the project aims to provide a clear understanding of the **Hadoop ecosystem** and its components. Hadoop is made up of several key parts, including **HDFS (Hadoop Distributed File System)** for storing large amounts of data and **MapReduce** for processing that data. By understanding these components, we can better grasp how data is handled in a distributed environment, which is crucial for working with big data.

The next objective is to learn how the **MapReduce programming model** works. The process is divided into two main stages: the **Map phase** and the **Reduce phase**. During the Map phase, input data is split into smaller pieces and processed in parallel. Each piece of data is turned into key-value pairs, which are then sent to the Reduce phase. In the Reduce phase, these key-value pairs are aggregated to produce the final output, such as the total count of each word in the Word Count program. Understanding how data is processed in these two phases will help in writing and running MapReduce jobs.

Another important objective of this project is to implement the **Word Count program** using Hadoop MapReduce. The Word Count program is often used as a simple example to demonstrate how MapReduce works. In this program, a large text file is read, and the goal is to count how many times each word appears. The project will guide through writing the **Map and Reduce classes** required for this task. After writing the code, the next step is to compile it into a **JAR file** that can be executed by Hadoop.

The project will also focus on setting up and running the program in a **Linux environment**. Since Hadoop is typically run on Linux-based systems, this project will demonstrate how to install and configure Hadoop on a Linux machine. The goal is to learn how to run Hadoop MapReduce jobs on a local machine or a small cluster. By doing so, we will understand how to manage files using **HDFS** and how to execute MapReduce programs using the Hadoop command-line tools.

### 3. System Requirements

To successfully execute the Word Count program in Hadoop MapReduce, the following system requirements must be met:

- Operating System: Linux (Ubuntu, CentOS, or any other distribution)
- Java Development Kit (JDK): Java 8 or higher
- Hadoop Version: Hadoop 3.3.6
- Memory: Minimum 4GB RAM (Recommended 8GB+ for better performance)
- Storage: Minimum 20GB free disk space
- Internet Connection: Required for downloading dependencies and configuring Hadoop

### 4. Understanding Hadoop MapReduce

Hadoop is an open-source framework designed to handle large-scale data processing by distributing tasks across multiple nodes in a cluster. It has become a go-to solution for organizations dealing with massive datasets that traditional systems cannot manage efficiently. Hadoop relies on the Hadoop Distributed File System (HDFS) for storing data across various nodes and uses the MapReduce programming model for processing data. MapReduce splits the task into smaller chunks, processes them in parallel, and combines the results to generate the final output.

MapReduce consists of two main phases: the **Map phase** and the **Reduce phase**. In the Map phase, the input data is divided into chunks, each processed by a mapper function. Each chunk of data is mapped to key-value pairs, representing individual data elements. For instance, in a Word Count program, the mapper would read through a text document, split the text into words, and emit a key-value pair for each word, where the word itself is the key and the count is the value. The **Reduce phase** then takes these key-value pairs, groups them by the key (word), and aggregates the counts for each word to produce the final output, which in the case of Word Count would be the total number of occurrences for each word in the input data.

The MapReduce framework operates in a distributed manner. When a job is executed, it divides the input data into blocks, and each block is assigned to a mapper. The mappers process the data independently, creating key-value pairs. After the map phase, the framework sorts and groups the data by the key, a process

called **shuffling**. These grouped data are then passed on to the reducers, which aggregate the values for each key and produce the final output.

Hadoop MapReduce operates in a fault-tolerant manner, meaning that if any node in the cluster fails, the system will automatically replicate the data and rerun the task on another available node, ensuring no data is lost. This ensures reliability, even when processing data on large clusters of machines. The parallel execution of tasks significantly speeds up the processing time, making it an ideal framework for tasks that require heavy data computations, such as large-scale text analysis, data mining, and statistical modeling.

MapReduce is highly scalable, meaning it can process datasets ranging from gigabytes to petabytes across thousands of machines. It is particularly cost-effective because it uses commodity hardware, reducing the infrastructure cost compared to other high-performance computing frameworks. The system's flexibility allows it to process various types of data, including structured, semi-structured, and unstructured data, making it adaptable to a wide range of applications.

While MapReduce offers significant advantages, it also has limitations. The most notable limitation is the overhead caused by frequent disk reads and writes during the map and reduce phases. Since data is stored on disk and read multiple times throughout the process, the system can experience I/O bottlenecks. Additionally, MapReduce is primarily batch-oriented, which means it is not suited for real-time or streaming data processing. While newer frameworks like Apache Spark have been introduced to address these issues, MapReduce remains a foundational component in the Hadoop ecosystem for processing large-scale data.

Despite its limitations, Hadoop MapReduce remains one of the most popular tools for large-scale data processing. Its ability to distribute tasks across a cluster, combined with its fault tolerance and scalability, makes it an essential tool for Big Data analytics. By leveraging the power of Hadoop MapReduce, organizations can process vast amounts of data efficiently, enabling them to gain valuable insights and make data-driven decisions.

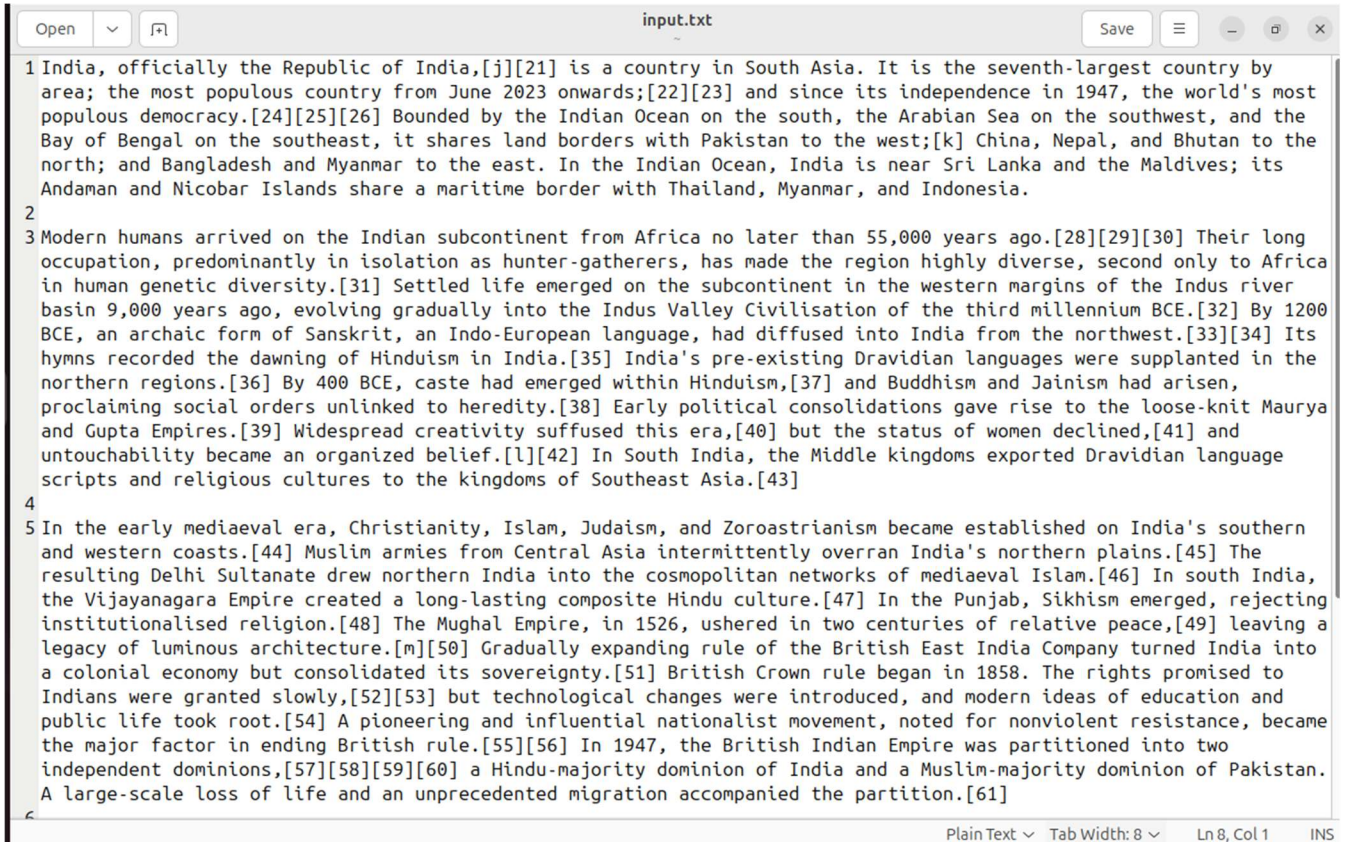


## 4. Steps to word count in mapreduce

### 1. Start-all.sh

```
praphul@praphul-VirtualBox:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as praphul in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [praphul-VirtualBox]
Starting resourcemanager
Starting nodemanagers
```

### 2. gedit input.txt (write input text)



1 India, officially the Republic of India,[j][21] is a country in South Asia. It is the seventh-largest country by area; the most populous country from June 2023 onwards;[22][23] and since its independence in 1947, the world's most populous democracy.[24][25][26] Bounded by the Indian Ocean on the south, the Arabian Sea on the southwest, and the Bay of Bengal on the southeast, it shares land borders with Pakistan to the west;[k] China, Nepal, and Bhutan to the north; and Bangladesh and Myanmar to the east. In the Indian Ocean, India is near Sri Lanka and the Maldives; its Andaman and Nicobar Islands share a maritime border with Thailand, Myanmar, and Indonesia.

2

3 Modern humans arrived on the Indian subcontinent from Africa no later than 55,000 years ago.[28][29][30] Their long occupation, predominantly in isolation as hunter-gatherers, has made the region highly diverse, second only to Africa in human genetic diversity.[31] Settled life emerged on the subcontinent in the western margins of the Indus river basin 9,000 years ago, evolving gradually into the Indus Valley Civilisation of the third millennium BCE.[32] By 1200 BCE, an archaic form of Sanskrit, an Indo-European language, had diffused into India from the northwest.[33][34] Its hymns recorded the dawning of Hinduism in India.[35] India's pre-existing Dravidian languages were supplanted in the northern regions.[36] By 400 BCE, caste had emerged within Hinduism,[37] and Buddhism and Jainism had arisen, proclaiming social orders unlinked to heredity.[38] Early political consolidations gave rise to the loose-knit Maurya and Gupta Empires.[39] Widespread creativity suffused this era,[40] but the status of women declined,[41] and untouchability became an organized belief.[l][42] In South India, the Middle kingdoms exported Dravidian language scripts and religious cultures to the kingdoms of Southeast Asia.[43]

4

5 In the early mediaeval era, Christianity, Islam, Judaism, and Zoroastrianism became established on India's southern and western coasts.[44] Muslim armies from Central Asia intermittently overran India's northern plains.[45] The resulting Delhi Sultanate drew northern India into the cosmopolitan networks of mediaeval Islam.[46] In south India, the Vijayanagara Empire created a long-lasting composite Hindu culture.[47] In the Punjab, Sikhism emerged, rejecting institutionalised religion.[48] The Mughal Empire, in 1526, ushered in two centuries of relative peace,[49] leaving a legacy of luminous architecture.[m][50] Gradually expanding rule of the British East India Company turned India into a colonial economy but consolidated its sovereignty.[51] British Crown rule began in 1858. The rights promised to Indians were granted slowly,[52][53] but technological changes were introduced, and modern ideas of education and public life took root.[54] A pioneering and influential nationalist movement, noted for nonviolent resistance, became the major factor in ending British rule.[55][56] In 1947, the British Indian Empire was partitioned into two independent dominions,[57][58][59][60] a Hindu-majority dominion of India and a Muslim-majority dominion of Pakistan. A large-scale loss of life and an unprecedented migration accompanied the partition.[61]

### 3. hadoop fs -mkdir /inputwc

```
praphul@praphul-VirtualBox:~$ hdfs dfs -mkdir /inputwc
```

### 4. hadoop fs -ls /

```
praphul@praphul-VirtualBox:~$ hadoop fs -ls /inputwc
Found 1 items
-rw-r--r--  1 praphul supergroup      220 2025-02-11 23:48 /inputwc/input.txt
```

5. `hadoop fs -put input.txt /inputwc`

```
praphul@praphul-VirtualBox:~$ hadoop fs -put input.txt /inputwc
```

6. `hadoop fs -ls /inputwc`

```
praphul@praphul-VirtualBox:~$ hadoop fs -ls /inputwc
Found 1 items
-rw-r--r--  1 praphul supergroup      220 2025-02-11 23:48 /inputwc/input.txt
```

7. `gedit WordCount.java` (Paste the Java Program given below in this file)

```
praphul@praphul-VirtualBox:~$ gedit WordCount.java
```

JAVA CODE ->

```
import java.io.IOException;
```

```
import java.util.StringTokenizer;
```

```
import org.apache.hadoop.conf.Configuration;
```

```
import org.apache.hadoop.fs.Path; import
```

```
org.apache.hadoop.io.IntWritable; import
```

```
org.apache.hadoop.io.Text; import
```

```
org.apache.hadoop.mapreduce.Job; import
```

```
org.apache.hadoop.mapreduce.Mapper; import
```

```
org.apache.hadoop.mapreduce.Reducer;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
```

```
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import
```

```
org.apache.hadoop.util.GenericOptionsParser;
```

```
public class WordCount {
```

```
public static class TokenizerMapper extends
```

```
Mapper<Object, Text, Text, IntWritable>{
```

```
private final static IntWritable one = new IntWritable(1); private
Text word = new Text();
```

```
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) { word.set(itr.nextToken());
context.write(word, one);
}
}
}
```

```
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> { private
IntWritable result = new IntWritable();
```

```
public void reduce(Text key, Iterable<IntWritable> values,
Context context
) throws IOException, InterruptedException { int sum = 0;
for (IntWritable val : values) { sum
+= val.get();
}
result.set(sum); context.write(key,
result);
}
}
```

```
public static void main(String[] args) throws Exception {
```



```
Configuration conf = new Configuration();
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  if
(otherArgs.length < 2) {
System.err.println("Usage: wordcount <in> [<in>...] <out>");
System.exit(2);
}
Job job = Job.getInstance(conf, "word count");  job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);      job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);    for (int i = 0; i < otherArgs.length - 1;
++i) {
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}
FileOutputFormat.setOutputPath(job,      new
Path(otherArgs[otherArgs.length - 1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

(Go to path -> cd hadoop-3.3.6/etc/hadoop -> sudo nano hadoop-env.sh and write  
(HADOOP\_CLASSPATH=\${JAVA\_HOME}/lib/tools.jar) save it and type cd again).

```
praphul@praphul-VirtualBox:~$ cd hadoop-3.3.6/etc/hadoop
praphul@praphul-VirtualBox:~/hadoop-3.3.6/etc/hadoop$ sudo nano hadoop-env.sh
[sudo] password for praphul:
praphul@praphul-VirtualBox:~/hadoop-3.3.6/etc/hadoop$ cd
```

9. hadoop com.sun.tools.javac.Main WordCount.java

```
praphul@praphul-VirtualBox:~$ hadoop com.sun.tools.javac.Main WordCount.java
```

10. jar -cf wc.jar WordCount\*.class

```
praphul@praphul-VirtualBox:~$ jar -cf wc.jar WordCount*.class
```

11. `hadoop jar wc.jar WordCount /inputwc/input.txt /outputwordcount`

(Open localhost:9870 and open outputwordcount -> part-r-00000)

```
praphul@praphul-VirtualBox:~$ hadoop jar wc.jar WordCount /inputwc/input.txt /outputwordcount
2025-02-11 23:53:40,088 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to Res
2025-02-11 23:53:43,291 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for
praphul/.staging/job_1739297668251_0001
2025-02-11 23:53:44,746 INFO input.FileInputFormat: Total input files to process : 1
2025-02-11 23:53:46,080 INFO mapreduce.JobSubmitter: number of splits:1
2025-02-11 23:53:47,128 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_17392
2025-02-11 23:53:47,132 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-02-11 23:53:47,880 INFO conf.Configuration: resource-types.xml not found
2025-02-11 23:53:47,881 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-02-11 23:53:56,133 INFO impl.YarnClientImpl: Submitted application application_17392
2025-02-11 23:53:56,336 INFO mapreduce.Job: The url to track the job: http://praphul-Virt
739297668251_0001/
```

12. `hadoop fs -ls /outputwordcount`

```
praphul@praphul-VirtualBox:~$ hadoop fs -ls /outputwordcount
Found 2 items
-rw-r--r--  1 praphul supergroup          0 2025-02-11 23:55 /outputwordcount/_SUCCESS
-rw-r--r--  1 praphul supergroup       63 2025-02-11 23:55 /outputwordcount/part-r-00000
```

13. `hadoop fs -cat /outputwordcount/part-r-00000`

```
43 the
26 and
23 in
21 india
18 of
15 a
11 to
9 its
7 is
7 from
```

## 5. Conclusion

In conclusion, this project successfully demonstrated the power and capabilities of **Hadoop MapReduce** in handling large-scale data processing tasks, specifically through the implementation of the **Word Count program**. By using this basic yet effective example, the project provided a hands-on understanding of the core concepts behind Hadoop and MapReduce, such as data distribution, parallel processing, and fault tolerance.

Through the implementation process, we gained a deeper understanding of how **MapReduce** divides a task into two phases: the **Map phase**, where data is processed into key-value pairs, and the **Reduce phase**, where those pairs are aggregated to produce meaningful results. This step-by-step breakdown revealed the efficiency of MapReduce in processing vast amounts of data by dividing it into smaller, manageable parts and executing them in parallel across a distributed system.

Setting up Hadoop on a Linux environment and running the Word Count program reinforced the practical aspects of deploying and managing a Hadoop cluster. We learned the process of configuring **HDFS (Hadoop Distributed File System)** for storing data and how to execute MapReduce jobs using Hadoop commands. This experience is essential for anyone working with Big Data frameworks in real-world environments, as it highlights the setup, execution, and management of data processing tasks.

Additionally, the project showcased the scalability and fault tolerance of Hadoop, emphasizing how it can handle large datasets across multiple machines and recover from node failures without data loss. These features make Hadoop a robust and reliable tool for Big Data applications, though the project also highlighted some limitations, such as the I/O overhead and batch processing nature of MapReduce.

Overall, this project provided valuable insights into the **Hadoop ecosystem**, the **MapReduce programming model**, and the practical steps involved in setting up and running a Hadoop-based data processing job. The skills and knowledge gained here will serve as a solid foundation for tackling more complex Big Data challenges in the future.