AIM

To implement and execute a **Word Count program using MapReduce in Hadoop** to count the frequency of each word in a text file stored in **HDFS**, using **Java**.

DESCRIPTION

MapReduce is a distributed programming model used in Hadoop for processing large datasets.
In this experiment:

- **Mapper** reads input text and emits (word, 1)
- **Reducer** sums the values for each word
- Input is stored in **HDFS**
- Output is written back to **HDFS**
- The program is packaged as a **JAR file** and executed using Hadoop command

STEP–BY–STEP EXECUTION

STEP 1: Check Java and Hadoop Installation

java -version

hadoop version

Output:

- Java version displayed
- Hadoop version displayed
  ✓ Confirms environment setup

STEP 2: Start Hadoop Services

Open **Command Prompt as Administrator**

cd \

cd hadoop

cd sbin

start-all.cmd

jps

Output (jps):

NameNode

DataNode

ResourceManager

NodeManager

✓ Hadoop services started successfully

STEP 3: Verify Hadoop Web Interfaces

Open browser:

- HDFS UI → http://localhost:9870
- YARN UI → http://localhost:8088

✓ Web UI opens successfully

STEP 4: Create Input Directory in HDFS

hadoop fs -mkdir /input

Output:

No error → directory created

Check in browser: open localhost:9870

- Go to **Utilities → Browse**
- /input directory visible (empty)

STEP 5: Create Input Text File (Local System)

Path:

Documents → Create folder → FILES

Create file: **page1.txt**

Input File Content

Annamacharya University

AIML

CSE(AI)

Branch

Thirdyear

AIML

CSE(AI)

Save and close the file.

STEP 6: Upload File into HDFS

hadoop fs -put C:\Users\admin\Documents\FILES\page1.txt /input

Output:

No error → upload successful

Verify:

hadoop fs -ls /input

Output:

page1.txt

✓ Input file successfully stored in HDFS

STEP 7: Create MapReduce Project in Eclipse

1. Open **Eclipse**
2. File → New → Java Project
    - Project Name: MapReduceWordCount
    - Execution Environment: **JavaSE-1.8**
3. Create package:

com.mapreduce.wc

STEP 8: Add Hadoop External JARs

Right-click project → Build Path → Configure Build Path → Libraries → Add External JARs

Add JARs from:

C:\hadoop\share\hadoop\

├– common

├– common\lib

├– mapreduce

├– mapreduce\lib

├– yarn

├– hdfs

Click **Apply → Close**

✓ Errors removed

STEP 9: Create WordCount Java Class

Package: com.mapreduce.wc
Class name: **WordCount**

MapReduce Java Program

```
package com.mapreduce.wc;


import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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 void main(String[] args) throws Exception {
      Configuration c = new Configuration();
      String[] files = new GenericOptionsParser(c, args).getRemainingArgs();


      Path input = new Path(files[0]);
      Path output = new Path(files[1]);


      Job j = Job.getInstance(c, "wordcount");
      j.setJarByClass(WordCount.class);
      j.setMapperClass(MapForWordCount.class);
```

```
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(j, input);
    FileOutputFormat.setOutputPath(j, output);

    System.exit(j.waitForCompletion(true) ? 0 : 1);
}

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text wordText = new Text();

    public void map(LongWritable key, Text value, Context con)
            throws IOException, InterruptedException {
        String[] words = value.toString().split("\\s+");
        for (String word : words) {
            wordText.set(word.toUpperCase());
            con.write(wordText, one);
        }
    }
}

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text word, Iterable<IntWritable> values, Context con)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable v : values) {
            sum += v.get();
        }
```

```
            con.write(word, new IntWritable(sum));
        }
    }
}
```

STEP 10: Export JAR File

Right-click project → Export → Java → JAR file

- Folder: Documents → JARFILES
- File name: WordCountMapReduce.jar

✓ JAR file created

STEP 11: Run MapReduce Job

hadoop jar C:\Users\admin\Documents\JARFILES\WordCountMapReduce.jar \

com.mapreduce.wc.WordCount /input /output

Output:

Running job: wordcount

Job completed successfully

✓ Program executed successfully

STEP 12: View Output

hadoop fs -cat /output/part-r-00000

Output (Word Count Result)

AIML 2

ANNAMACHARYA 1

BRANCH 1

CSE(AI) 2

THIRDYEAR 1

UNIVERSITY 1

STEP 13: View Output via Browser

- Go to http://localhost:9870

- Browse → /output
- Open part-r-00000

✓ Output visible

STEP 14: Copy Output to Local Machine

hadoop fs -get /output/part-r-00000 C:\Users\admin\Documents\FILES\output.txt

✓ Output copied to local system

STEP 15: Delete Input / Output (Optional)

hadoop fs -rm -r /input/page1.txt

hadoop fs -rm -r /output

STEP 16: Stop Hadoop Services

stop-all.cmd

RESULT:

The **Word Count MapReduce program** was successfully implemented using **Java and Hadoop**.
The program correctly counted the frequency of each word from the input file stored in HDFS and produced the output in HDFS.