**BIG DATA LAB 4**

**Experiment name: Map Reduce Algorithm using Hadoop**

Name: Gouri Ramachandran

Register No: 22MIA1074

**Aim**

The aim of this project is to implement a **Word Count** program using **Hadoop MapReduce**. The program processes a large text file, tokenizes the words, and counts their occurrences, demonstrating the distributed computing capabilities of Hadoop.

**Algorithm**

**Algorithm:**

**Step 1: Initialize Hadoop Configuration**

* Configure Hadoop environment and set up a **MapReduce job**.

**Step 2: Implement the Mapper Class**

* Read input text line by line.
* Tokenize the text into words.
* Emit each word as a key with a value of **1**.

**Step 3: Implement the Reducer Class**

* Receive word and count pairs from the Mapper.
* Sum the occurrences of each word.
* Output the word and its total count.

**Step 4: Set Up the Driver Class**

* Define the job configuration.
* Specify the **Mapper, Reducer, and Combiner**.
* Set input and output paths.
* Submit the job for execution.

**Step 5: Execute and Retrieve Results**

* Run the program with Hadoop.
* View the final word count results from the output directory.

**Program**

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

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

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

public class WordCount {

// Mapper Class

public static class TokenizerMapper

extends Mapper<LongWritable, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(LongWritable key, Text value, Context context

) throws IOException, InterruptedException {

String[] words = value.toString().split("\\s+");

for (String str: words) {

word.set(str);

context.write(word, one);

}

}

}

// Reducer Class

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);

}

}

// Driver Class

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

Configuration conf = new Configuration();

Job job = new Job(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);

FileInputFormat.addInputPath(job, new Path(args[0]));

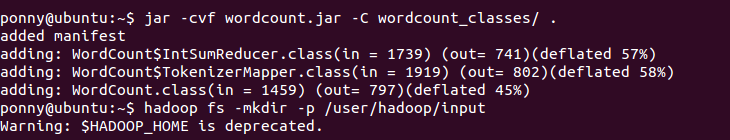
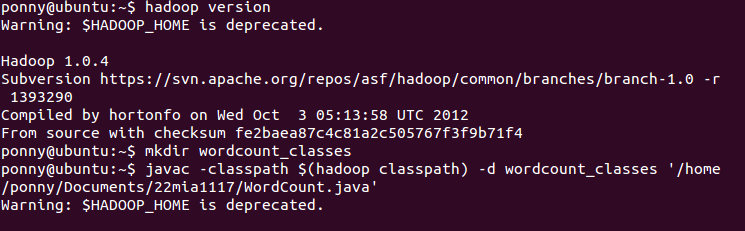
FileOutputFormat.setOutputPath(job, new Path(args[1]));

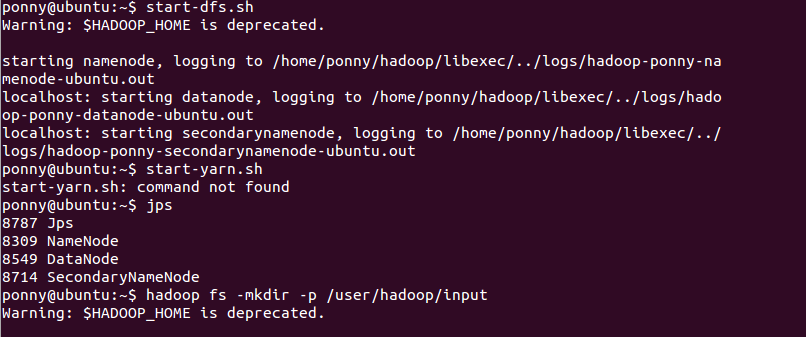
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Execution**

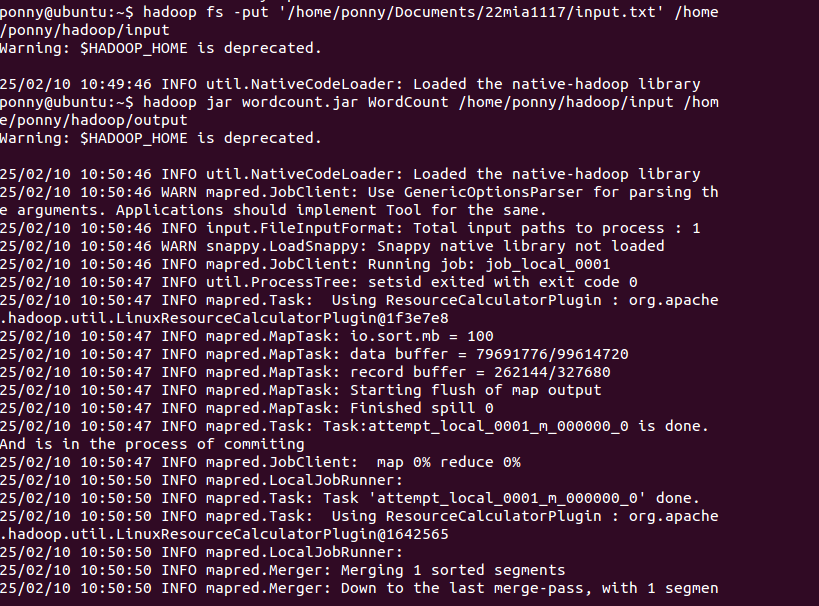




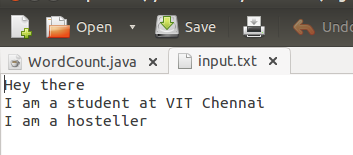


A screenshot of a computer

AI-generated content may be incorrect.



**Input**

****

**Output**

A computer screen with white text

AI-generated content may be incorrect.

**Result**

The Word Count program was successfully implemented using **Hadoop MapReduce**. The program reads a text file, processes it in a distributed manner, and provides an accurate count of word occurrences. The **Mapper** tokenizes the input and assigns an initial count to each word, while the **Reducer** aggregates these counts to produce the final word frequencies.

Upon execution, the program correctly processes the given input dataset and stores the results in the specified output directory. The experiment demonstrates the efficiency of **Hadoop's parallel processing framework**, making it suitable for handling large-scale text data.