1. **b. Find the number of occurrence of a word appearing in the input file(s).**

**Aim:**

To write a java map-reduce program for counting the number of occurrences of a word in a text file.

**Procedure:**

1. Prepare the input file: Create a text file that contains the input data
2. Package the Java program: Compile the WordCount Java program and package it into a JAR file.
3. Upload the input file to HDFS: Use the hadoop fs -put command to upload the input file to HDFS.
4. Submit the MapReduce job: Use the hadoop jar command to submit the WordCount program JAR file and specify the input and output paths.
5. Monitor the job execution: The MapReduce job will start executing, and user can monitor its progress and logs through the Hadoop JobTracker interface or the command line.
6. Check the output: Once the job is completed, user can check the output by using the hadoop fs -cat command to display the contents of the output file.

**Program:**

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;

public class WordSCount {

public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

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

private Text word = new Text();

private String searchWord;

protected void setup(Context context) throws IOException, InterruptedException {

// Get the word to search for from the configuration

Configuration conf = context.getConfiguration();

searchWord = conf.get("searchWord");

}

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

// Tokenize the input line

StringTokenizer tokenizer = new StringTokenizer(value.toString());

// Iterate through tokens

while (tokenizer.hasMoreTokens()) {

// Get next token

String token = tokenizer.nextToken();

// Emit key-value pair only if token matches the search word

if (token.equals(searchWord)) {

word.set(token);

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;

// Sum up the counts for the word

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

if (args.length != 3) {

System.err.println("Usage: WordCount <inputFile> <outputDir> <searchWord>");

System.exit(1);

}

Configuration conf = new Configuration();

conf.set("searchWord", args[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);

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

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

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

}

}

**Procedure**

**Setting the path to run mapreduce**

1.Type the following command in your hadoop login. (in hdoop) to set path

*$ export HADOOP\_CLASSPATH=$(hadoop classpath)*

Make sure it has been set correctly by using

*$ echo $HADOOP\_CLASSPATH*

2. Create a directory on HDFS to store your input / output for mapreduce

Check whether the directory created or not by

*$ hadoop fs -ls*

otherwise check using web browser type localhost:9870 🡪 then click on Utilities -🡪 Browse the file system.

3. Create a directory in hdoop home and create a input text file with some sentences

4. Create a java file for running mapreduce tasks

5. Create another directory to store java class files

6. Move the input text file into HDFS

$ hdfs dfs -put <PATH OF SOURCE INPUT FILE> <HDFS PATH>

7. Compile the program and create a directory to store java classfiles

Syntax

$ javac -classpath ${HADOOP\_CLASSPATH} -d <Path Classes folder> <Path of wordcount java file>

8. Create a jar file from the classfiles that we created in previous step

$ jar -cvf <path of newjar file> <path of classfile folder>

9. Run the mapreduce task

$ hadoop jar <path of jar file> <name of class> <input directory> <output\_directory> word\_to\_search

10.Check the mapreduce output by

$ hadoop fs -cat <HDFS OUTPUT FILE PATH>

**Result :**

Thus the java program to count a particular word in the file is executed in Hadoop environment successfully.