

## **EXPERIMENT-2**

### **Aim:**

Develop a MapReduce program to calculate the frequency of a given word in a given file.

### **Theory:**

Word frequency counting is a classic MapReduce problem that demonstrates the "divide and conquer" approach. The Map phase processes input data by emitting each word with a count of 1, while the Reduce phase sums these counts for each unique word.

### **Description:**

The program reads a text file, splits it into words, and counts how many times a specific target word appears. The mapper emits the word as a key with value 1 whenever it encounters the target word. The reducer sums these values to get the total count.

### **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;
public class WordCount {
    // 1. Mapper Class
    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);
            }
        }
    }
    // 2. 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);
}
}

// 3. Driver / Main Method
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class); // Optional optimization
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);
}
}

```

**Input:**

A text file named input.txt containing the following text:

```
root@DESKTOP-0MNJIUH:~/wordfreq# hdfs dfs -cat /user/root/input/input.txt  
big data is powerful  
big data needs map reduce  
big data big future
```

**Output:**

The content of the part-r-00000 file generated by Hadoop:

```
Bytes Written=6  
root@DESKTOP-0MNJIUH:~/wordfreq# hdfs dfs -cat /user/root/output/part-r-00000  
big 4  
root@DESKTOP-0MNJIUH:~/wordfreq#
```

**Commands:**

Create input directory in HDFS:	hdfs dfs -mkdir /input_dir
Upload the local text file to HDFS:	hdfs dfs -put input.txt /input_dir
Compile the Java program:	javac -classpath \$(hadoop classpath) -d . WordCount.java
Create the JAR file:	jar -cvf wordcount.jar *.class
Run the Hadoop Job:	hadoop jar wordcount.jar WordCount /input_dir /output_dir
View the Output:	hdfs dfs -cat /output_dir/part-r-00000

**Conclusion:**

In this experiment, we successfully developed and executed a MapReduce program to calculate word frequency. We understood how the Mapper tokenizes the text and emits key-value pairs (Word, 1), and how the Reducer aggregates these values to produce the final count for every word in the input file.