JUNAID GIRKAR

60004190057

TE COMPS A4

**EXPERIMENT - 3**

**Aim**

Execution of MapReduce program for counting word occurrences in a text file

**Theory**

Working of Wordcount Mapreduce:

The main method invokes the ToolRunner to run the job based on the con guration information. The map method processes one line at a time, splitting the line on regular expression word boundaries. It emits key/value pairs in the format <word, 1>.

For File0, the map method emits these key/value pairs:

<Hadoop, 1> <is, 1> <an, 1> <elephant, 1>

For File1, the map method emits:

<Hadoop, 1> <is, 1> <as, 1> <yellow, 1> <as, 1> <can, 1> <be, 1>

For File2, the map method emits:

<Oh, 1> <what, 1> <a, 1> <yellow, 1> <fellow, 1> <is, 1> <Hadoop, 1>

The reduce method adds up the number of instances for each key, and then emits them sorted in UTF-8 alphabetical order (all uppercase words, and then all lowercase words). Note that the WordCount code speci es key/value pairs. The Mapper and Reducer classes handle the rest of the processing for us.

<Hadoop, 3> <Oh, 1> <a, 1> <an, 1> <as, 2> <be, 1> <can, 1> <elephant, 1> <fellow, 1>

<is, 3> <what, 1> <yellow, 2>

**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 {   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();  Job job = new Job(conf, "word count");  //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);  } }** |
| --- |

Output

Graphical user interface, text, application, email

Description automatically generated

**Conclusion**

Understood and implemented word count of a document by MapReduce method