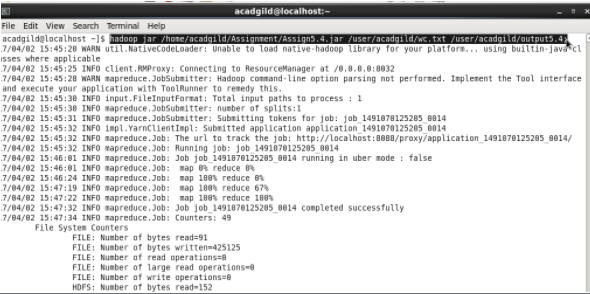
|  |
| --- |
|  |
|  | **Assignment 5.4**  **Problem Statement :**  Write a word count program using partitioner and implement the following logic in the partitioner words with length 5 should go into reducer 1 and words with length 6 should go into reduer 2 and the rest of the words should go into reducer 3.  **STEP 1:**  **Write the following map reduce program in the java platform that performs the mapreduce operation as per the problem statement:**  Driver class :  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.Text; |
|  | import org.apache.hadoop.mapreduce.Job; |
|  | import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; |
|  | import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; |
|  | import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; |
|  | import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; |
|  |  |
|  | public class Assign { |
|  |  |
|  | public static void main(String[] args) throws IllegalArgumentException, IOException, ClassNotFoundException, InterruptedException { |
|  | // TODO Auto-generated method stub |
|  | Configuration conf = new Configuration(); |
|  | Job job = new Job(conf, "Demo"); |
|  | job.setJarByClass(Assign.class); |
|  |  |
|  | job.setMapOutputKeyClass(Text.class); |
|  | job.setMapOutputValueClass(IntWritable.class); |
|  | job.setMapperClass(AssignMapper.class); |
|  |  |
|  | job.setPartitionerClass(AssignPartitioner.class); |
|  |  |
|  | job.setNumReduceTasks(3);//setting the number of reducers |
|  | job.setReducerClass(AssignReducer.class); |
|  | job.setOutputKeyClass(Text.class); |
|  | job.setOutputValueClass(IntWritable.class); |
|  |  |
|  | job.setInputFormatClass(TextInputFormat.class); |
|  | job.setOutputFormatClass(TextOutputFormat.class); |
|  |  |
|  | FileInputFormat.addInputPath(job, new Path(args[0])); |
|  | FileOutputFormat.setOutputPath(job, new Path(args[1])); |
|  |  |
|  | job.waitForCompletion(true); |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | Mapper Class: |
|  | import java.io.IOException; |
|  | import java.util.regex.Pattern; |
|  |  |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.LongWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | import org.apache.hadoop.mapreduce.Mapper; |
|  |  |
|  | public class AssignMapper extends Mapper<LongWritable, Text, Text, IntWritable> { |
|  | public void map(LongWritable Key, Text Value, Context context) throws IOException, InterruptedException { |
|  | String[] data = Value.toString().split(","); |
|  | IntWritable one=new IntWritable(1); |
|  | Text t=new Text(data[0]); |
|  | context.write(t, one); |
|  | } |
|  | } |
|  |  |
|  | Reducer Class: |
|  | import java.io.IOException; |
|  |  |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | import org.apache.hadoop.mapreduce.Reducer; |
|  |  |
|  | public class AssignReducer extends Reducer<Text, IntWritable, Text, IntWritable> |
|  | { |
|  | private IntWritable tot = new IntWritable(); |
|  | private int minValue = Integer.MIN\_VALUE; |
|  |  |
|  | @Override |
|  | public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException, InterruptedException |
|  | { |
|  | int count = 0; |
|  | for ( IntWritable value : values ) |
|  | { if(value.get()>minValue){ |
|  | count+=value.get(); |
|  | } |
|  | } |
|  | tot.set(count); |
|  | context.write(key, tot); |
|  | } |
|  | } |
|  |  |
|  | Partitioner class: |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | import org.apache.hadoop.mapreduce.Partitioner; |
|  |  |
|  | public class AssignPartitioner extends Partitioner<Text, IntWritable> { |
|  |  |
|  |  |
|  | @Override |
|  | public int getPartition(Text key, IntWritable value, int arg2) { |
|  |  |
|  | int k = key.toString().length(); |
|  | if (k==5) |
|  | { |
|  | return 0; |
|  | } |
|  | else if (k==6) |
|  | { |
|  | return 1; |
|  | } |
|  | else |
|  | { |
|  | return 2; |
|  | } |
|  |  |
|  | } |

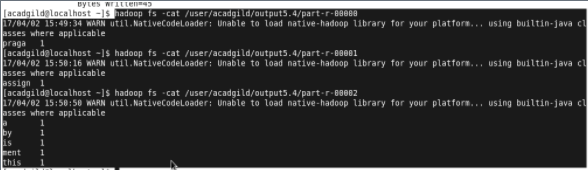
**STEP 2:**

**The hadoop environment that establishes the relation with that of the mapreduce program is represented below:**

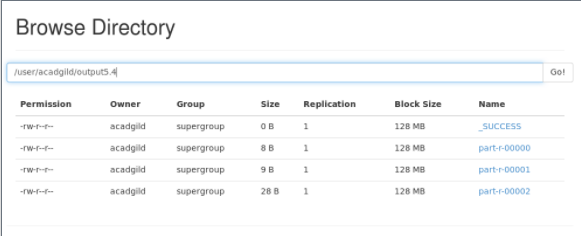


**STEP 3:**

**The output that represents the frequency of the words that are produced within the hadoop environment is shown below**



**The browsing directory showing the success status is represented in the below screenshot:**



OUTPUT :

The required output for the mapreduce program to find the frequency of the words is described below :

