Big Data Analysis

Practical 4: MapReduce

Objective :

Design MapReduce algorithms to take a very large file of integers and produce as output:

a)      The largest integer

b)      The average of all the integers.

c)      The same set of integers, but with each integer appearing only once. \*

d)     The count of the number of distinct integers in the input.\*

Roll No. & Name : 18bce183 & Prince Prajapati

Submitted to: Prof. Jaiprakash Verma

**What is MapReduce?**

MapReduce is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. As the processing component, MapReduce is the heart of Apache Hadoop. The term "MapReduce" refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

Mapper class:

|  |
| --- |
| import java.io.\*;  import java.util.\*;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.mapreduce.Mapper;    public class BDA\_Mapper extends Mapper<LongWritable, Text, Text, LongWritable> {    private TreeMap<String,Integer> tmap;    @Override  public void setup(Context context) throws IOException,InterruptedException  {  tmap = new TreeMap<String,Integer>();  }    @Override  public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  {  if(tmap.containsKey(value.toString().trim())) {  int count=tmap.get(value.toString().trim());  tmap.put(value.toString().trim(),count+1);  }  else  tmap.put(value.toString().trim(),1);  }    @Override  public void cleanup(Context context) throws IOException,InterruptedException  {  for (Map.Entry<String,Integer> entry : tmap.entrySet())  {    String number= entry.getKey();  int count = entry.getValue();      context.write(new Text(number), new LongWritable(count));  }  }  } |

Reducer class

|  |
| --- |
| import java.io.IOException;  import java.util.Map;  import java.util.TreeMap;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Reducer;    public class BDA\_Reducer extends Reducer<Text,LongWritable, Text,LongWritable> {    private TreeMap<String,Long> tmap2;  private int max=Integer.MIN\_VALUE,unique=0,cnt=0;  private long sum=0;  @Override  public void setup(Context context) throws IOException,InterruptedException  {  tmap2 = new TreeMap<String,Long>();  }  @Override  public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException  {    String number = key.toString();  long count = 0;    for (LongWritable val : values)  {  count += val.get();  sum+=((int)val.get())\*Integer.parseInt(number.trim());  }  tmap2.put( number,count);  cnt+=count;  if(max<Integer.parseInt(number.trim()))max=Integer.parseInt(number.trim());  unique++;    }  @Override  public void cleanup(Context context) throws IOException,InterruptedException  {    for (Map.Entry< String,Long> entry : tmap2.entrySet())  {    long count = entry.getValue();  String name = entry.getKey();  context.write(new Text(name),new LongWritable(count));  }  context.write(new Text("MAX NUMBER = "),new LongWritable(max));  context.write(new Text("AVERAGE = "),new LongWritable(sum/cnt));  context.write(new Text("Total Unique Numbers ="), new LongWritable(unique));  }  } |

Combiner class:

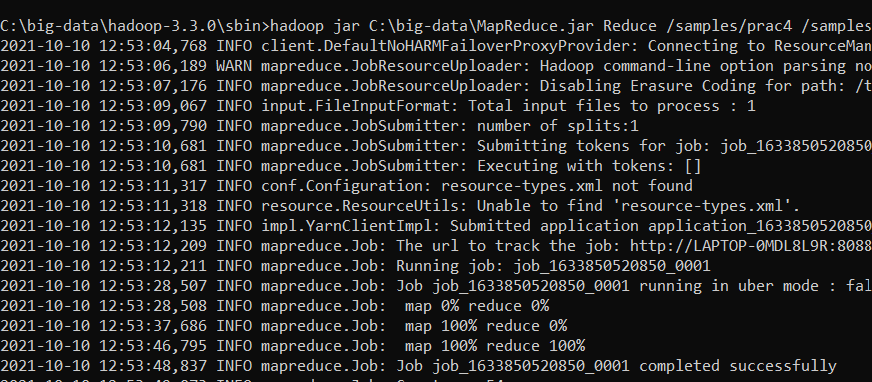
|  |
| --- |
| import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  import org.apache.hadoop.util.GenericOptionsParser;    public class Reduce {    public static void main(String[] args) throws Exception  {  Configuration conf = new Configuration();      Job job = Job.getInstance(conf, "Pratical 4");  job.setJarByClass(Reduce.class);    job.setMapperClass(BDA\_Mapper.class);  job.setReducerClass(BDA\_Reducer.class);    job.setMapOutputKeyClass(Text.class);  job.setMapOutputValueClass(LongWritable.class);    job.setOutputKeyClass(LongWritable.class);  job.setOutputValueClass(Text.class);    FileInputFormat.addInputPath(job, new Path(args[0]));  FileOutputFormat.setOutputPath(job, new Path(args[1]));    System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

**OUTPUT:**

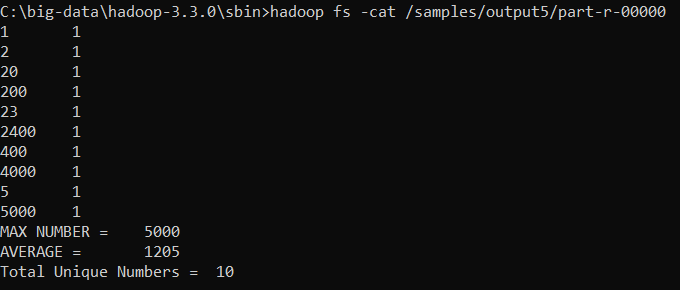
Command:

hadoop jar C:\big-data\MapReduce.jar Reduce /samples/prac4 /samples/output5

MapReduce process



OUTPUT:



Conclusion: After implementing this practical now I have clear knowledge of how Mapper, Reducer and Combiner class works and how to implement it.