MapReduce Program



There are a total of 10 fields of information in each line. Our programming objective uses only the first and fourth fields, which are arbitrarily called "year" and "delta" respectively. We will ignore all the other fields of data.We will need to define a Mapper class, Reducer class and a Driver class.

**Designing and Implementing the Mapper Class**

public class ReceiptMapper extends Mapper<LongWritable,Text,Text,Text>

{

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

StringTokenizer itr=new StringTokenizer(value.toString(),[\\s+](file:///\\s+)”);

String year=itr.nextToken();

itr.nextToken();

itr.nextToken();

String delta=itr.nextToken();

context.write(new Text(“summary”) , new Text (year + “\_” + delta);

}

}

**Designing and Implementing the Reducer Class**

**for (Text value : values ) {**

**compositeString=value.toString();**

**compositeStringArray=compositeString.split(“\_”);**

**tempYear=new Text(compositeStringArray[0]);**

**tempValue=new Long(compositeStringArray[1]).longValue();**

**if(tempValue < min) {**

**min=tempValue;**

**minYear=tempYear;**

**}**

**}**

**Text keyText= new Text (“min” + “(“ + minYear.toString() + “): “);**

**context.write(keyText , new FloatWritable(min));**

**}**

**}**

In the Reducer code,

* We iterate over the all values associated with the key in the for loop.
* We convert the Text value to a string (compositeString) so we can split out the year from the value (delta) for that year. We then convert that string into a string array (compositeStringArray) which splits out the compositeString variable based on the “\_” character.
* We pull out the year from the 0th element of the string array, and then we pull out the value as the “1th” element of the array.
* We determine if we’ve found a global minimum delta, and if so, assign the min and minYear accordingly.
* When we pop out of the loop, we have the global min delta and the year associated with the min. We emit the year and min delta.

**Design and Implement The Driver**

**public class ReceiptsDriver extends Configured implements Tool {**

**public int run(String[]args throws Exception {**

**if(args.length != 2) {**

**System.out.println(“usage : %s [generic options]<inputfile><outputdir>\n” , getClass().getSimpleName());**

**System.exit(1);**

**}**

**Job job=new Job(getConf() , “my receipts”);**

**job.setJarByClass(ReceiptsDriver.class);**

**job.setMapperClass(ReceiptsMapper.class);**

**job.setReducerClass(ReceiptsReducer.class);**

**job.setInputFormatClass(TextInputFormat.class);**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(FloatWritable.class);**

**job.setMapOutValueClass(Text.class);**

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

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

**return job.waitForCompletion(true) ? 0 : 1 ;**

**}**

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

**configuration conf=new Configuration();**

**System.exit(ToolRunner.run(conf , new ReceiptsDriver() , args));**

**}**

**}**

In the Driver class, we also define the types for output key and value in the job as Text and FloatWritable respectively. If the mapper and reducer classes do NOT use the same output key and value types, we must specify for the mapper. In this case, the output value type of the mapper is Text, while the output value type of the reducer is FloatWritable.

**IMPORTANT NOTE : These are the variable length equivalents of the fixed legth IntWritable and LongWritable types.**