Architecture of Big Data System

MapReduce Assignment

Find the count of each word from a given input file.

Count the number of words based of character length in a given file

Create dataset with fields like 'Student Name', 'Institute', 'Program Name', and 'Gender' and solve following questions.

1. Compute number of students from each Institute.
2. Number of students enrolled to any program.
3. Number of 'boy' and 'girl' students.
4. Number of 'boy' and 'girl' students from selected Institute.

Dataset: Temperature of Indian Cities. Fields of dataset are Date, Average Temperature, City, Country, Latitude and Longitude (Dataset is attached). Solve following questions

1. Find maximum and minimum temperature of all cities from the given dataset.
2. Count number of data point for each city
3. Find the maximum and minimum temperature for city Bangalore from the given dataset.
4. Find the maximum and minimum temperature for any given city from the given dataset. City name should be passed through command line argument.

###Total Character Count

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.LongWritable; import org.apache.hadoop.io.Text; 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 characterCount { public static class testCharacterCountMapper extends Mapper<LongWritable, Text,IntWritable,IntWritable> { public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException { String line=value.toString(); StringTokenizer tokenizer=new StringTokenizer(line); while(tokenizer.hasMoreTokens()) { String str=tokenizer.nextToken(); value.set(str); context.write(new IntWritable(str.length()), new IntWritable(1)); }

```
    }

}

public static class testCharacterCountReducer extends Reducer <IntWritable,IntWritable,IntWritable,IntWritable > 
{
   public void reduce(IntWritable key,Iterable <IntWritable> value,Context context)
       throws IOException, InterruptedException {
       int sum=0;
   for(IntWritable x:value) {
       sum+=x.get();
   }
       context.write(key, new IntWritable(sum));
   }
}

public static void main(String[] args) throws Exception{
   // TODO Auto-generated method stub
   Configuration conf = new Configuration();

   Job job = Job.getInstance(conf, "character count");

   job.setJarByClass(characterCount.class);
   job.setMapperClass(testCharacterCountMapper.class);
   job.setReducerClass(testCharacterReducer.class);

   job.setOutputKeyClass(IntWritable.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);
}
```

}

###Total COunt 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.LongWritable; import org.apache.hadoop.io.Text; 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 characterCount { public static class testCharacterCountMapper extends Mapper<LongWritable, Text,IntWritable,IntWritable> { public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException { String line=value.toString(); StringTokenizer tokenizer=new StringTokenizer(line); while(tokenizer.hasMoreTokens()) { String str=tokenizer.nextToken(); value.set(str); context.write(new IntWritable(str.length()), new IntWritable(1)); }

```
    }

}

public static class testCharacterCountReducer extends Reducer <IntWritable,IntWritable,IntWritable,IntWritable > 
{
   public void reduce(IntWritable key,Iterable <IntWritable> value,Context context)
       throws IOException, InterruptedException {
       int sum=0;
   for(IntWritable x:value) {
       sum+=x.get();
   }
       context.write(key, new IntWritable(sum));
   }
}

public static void main(String[] args) throws Exception{
   // TODO Auto-generated method stub
   Configuration conf = new Configuration();

   Job job = Job.getInstance(conf, "character count");

   job.setJarByClass(characterCount.class);
   job.setMapperClass(testCharacterCountMapper.class);
   job.setReducerClass(testCharacterCountReducer.class);

   job.setOutputKeyClass(IntWritable.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);
}
```

}

Create dataset with fields like 'Student Name', 'Institute', 'Program Name', and 'Gender' and solve following questions.

1. Compute number of students from each Institute.
2. Number of students enrolled to any program.
3. Number of 'boy' and 'girl' students.
4. Number of 'boy' and 'girl' students from selected Institute.

```
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.LongWritable;
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 Student {

    public static class testMap_institute extends Mapper<LongWritable,Text, Text, IntWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] insti = line.split(",");
            Configuration conf=context.getConfiguration();
            String par = context.getConfiguration().get("INSTITUTE");
            if(insti[1].equals(par))
            {
            context.write(values,new IntWritable(1));
            }
        }
    }
    public static class testMap_program extends Mapper<LongWritable,Text, Text, IntWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] program = line.split(",");
            String par = context.getConfiguration().get("PROGRAM");
            if(program[2].equals(par))
            context.write(new Text(program[2]),new IntWritable(1));
            }
    }
    public static class testMap_gender extends Mapper<LongWritable,Text, Text, IntWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] gender = line.split(",");
            String par = context.getConfiguration().get("GENDER");
            if(gender[3].equals(par))
            context.write(new Text(gender[3]),new IntWritable(1));
            }
    }
    public static class testMap_institgender extends Mapper<LongWritable,Text, Text, IntWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] insti = line.split(",");
            String par = context.getConfiguration().get("INSTITUTE");
            if(insti[1].equals(par))
            context.write(new Text(insti[3]),new IntWritable(1));
            }
    }
    public static class testReduce extends Reducer<Text,IntWritable,Text,IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException, Interrupte
dException{
            int totalCount = 0;
            for(IntWritable x: values) {
                totalCount += x.get();
            }
            context.write(key, new IntWritable(totalCount));
        }

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        conf.set("INSTITUTE", args[1]);
        conf.set("PROGRAM", args[1]);
        Job job = Job.getInstance(conf,"student_institute");
        job.setJarByClass(studentDetails2.class);
        job.setMapperClass(testMap_institute.class);
        job.setReducerClass(testReduce.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job,new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        job.waitForCompletion(true);

        Configuration conf1 = new Configuration();
        conf1.set("PROGRAM", args[1]);
        Job job1 = Job.getInstance(conf1,"student_program");
        job1.setJarByClass(studentDetails2.class);
        job1.setMapperClass(testMap_program.class);
        job1.setReducerClass(testReduce.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job1,new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[2]));

        job1.waitForCompletion(true);

        Configuration conf2 = new Configuration();
        conf2.set("GENDER", args[1]);
        Job job2 = Job.getInstance(conf2,"student_gender");
        job2.setJarByClass(studentDetails2.class);
        job2.setMapperClass(testMap_gender.class);
        job2.setReducerClass(testReduce.class);

        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job2,new Path(args[0]));
        FileOutputFormat.setOutputPath(job2, new Path(args[2]));

        job2.waitForCompletion(true);
        Configuration conf3 = new Configuration();
        conf3.set("INSTITUTE", args[1]);
        Job job3 = Job.getInstance(conf3,"student_institgender");
        job3.setJarByClass(studentDetails2.class);
        job3.setMapperClass(testMap_institgender.class);
        job3.setReducerClass(testReduce.class);

        job3.setOutputKeyClass(Text.class);
        job3.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job3,new Path(args[0]));
        FileOutputFormat.setOutputPath(job3, new Path(args[4]));

        job3.waitForCompletion(true);
        System.exit(job3.waitForCompletion(true)?0:1);

    }
}
```

Dataset: Temperature of Indian Cities. Fields of dataset are Date, Average Temperature, City, Country, Latitude and Longitude (Dataset is attached). Solve following questions

1. Find maximum and minimum temperature of all cities from the given dataset.
2. Count number of data point for each city
3. Find the maximum and minimum temperature for city Bangalore from the given dataset.
4. Find the maximum and minimum temperature for any given city from the given dataset. City name should be passed through command line argument.

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Weather {

    public static class testMapper extends Mapper<LongWritable,Text, Text, FloatWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] list = line.split(",");


            context.write(new Text(list[2]),new FloatWritable(Float.parseFloat(list[1])));
        }
    }
    public static class testMapper_1 extends Mapper<LongWritable,Text, Text, IntWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] entry2 = line.split(",");


            context.write(new Text(entry2[2]),new IntWritable(1));

        }
    }

    public static class testMapper_2 extends Mapper<LongWritable,Text, Text, FloatWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String line = values.toString();
            String[] list = line.split(",");

            if(list[2].equals("Bangalore"))
            {

                context.write(new Text(list[2]),new FloatWritable(Float.parseFloat(list[1])));
            }
    }}

    public static class testMapper_3 extends Mapper<LongWritable,Text, Text, FloatWritable> {
        public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException{
            String par = context.getConfiguration().get("CITY");
            String line = values.toString();
            String[] city2 = line.split(",");
            if(city2[2].equals(par))
            {
            context.write(new Text(city2[2]),new FloatWritable(Float.parseFloat(city2[1])));
            }
        }
    }

    public static class testReducer_2 extends Reducer<Text,IntWritable,Text,IntWritable>
    {
        public void reduce(Text key, Iterable <IntWritable> values,Context context) throws IOException,Interrupte
dException
        {
            int total_count = 0;
            for(IntWritable x:values) {
                total_count +=x.get();
            }
            context.write(key, new IntWritable(total_count));
        }

    }

    public static class testReduce extends Reducer<Text,FloatWritable,Text,FloatWritable> {
        public void reduce(Text key, Iterable<FloatWritable> values,Context context) throws IOException, Interrup
tedException{
            float Max = 0,Min = 999;


            for(FloatWritable x: values) {
                if(Max < x.get())
                {
                    Max = x.get();
                }
                if(Min > x.get())
                {
                    Min = x.get();
                }
            }
            context.write(key, new FloatWritable(Max));
            context.write(key, new FloatWritable(Min));
        }


    }

    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
                Configuration conf = new Configuration();
                conf.set("CITY", args[1]);
                Job job1 = Job.getInstance(conf,"city-count1");
                job61.setJarByClass(Weather.class);
                job61.setMapperClass(testMapper_1.class);
                job61.setReducerClass(testReduce.class);
                job61.setOutputKeyClass(Text.class);
                job61.setOutputValueClass(FloatWritable.class);

                FileInputFormat.addInputPath(job61,new Path(args[0]));
                FileOutputFormat.setOutputPath(job61, new Path(args[1]));
                job61.waitForCompletion(true);

                Configuration conf2 = new Configuration();
                Job job62 = Job.getInstance(conf2,"city-count2");
                job62.setJarByClass(Weather.class);
                job62.setMapperClass(testMapper_2.class);
                job62.setReducerClass(testReducer_2.class);
                job62.setOutputKeyClass(Text.class);
                job62.setOutputValueClass(IntWritable.class);

                FileInputFormat.addInputPath(job62,new Path(args[0]));
                FileOutputFormat.setOutputPath(job62, new Path(args[2]));
                job62.waitForCompletion(true);

                Configuration conf3 = new Configuration();
                job63.setJarByClass(Weather.class);
                job63.setMapperClass(testMapper_2.class);
                job63.setReducerClass(testReduce.class);
                job63.setOutputKeyClass(Text.class);
                job63.setOutputValueClass(FloatWritable.class);

                FileInputFormat.addInputPath(job63,new Path(args[0]));
                FileOutputFormat.setOutputPath(job63, new Path(args[3]));
                job63.waitForCompletion(true);


                Configuration conf4 = new Configuration();
                Job job64 = Job.getInstance(conf3,"city-count3");
                job64.setJarByClass(Weather.class);
                job64.setMapperClass(testMapper_3.class);
                job64.setReducerClass(testReduce.class);
                job64.setOutputKeyClass(Text.class);
                job64.setOutputValueClass(FloatWritable.class);

                FileInputFormat.addInputPath(job64,new Path(args[0]));
                FileOutputFormat.setOutputPath(job64, new Path(args[4]));
                job64.waitForCompletion(true);


                System.exit(job64.waitForCompletion(true)?0:1);

    }
}
```