To perform the task of finding the average for temperature, dew point, and wind speed from weather data using Eclipse, follow these steps:

Step 1: Set Up Eclipse with Hadoop Plugin

Ensure that you have Eclipse installed and configured with the Hadoop plugin. The Hadoop plugin allows you to develop and run MapReduce applications within Eclipse.

Step 2: Create a New Java Project

Open Eclipse and create a new Java project by going to File -> New -> Java Project. Provide a name for your project, such as "WeatherDataProcessing," and click "Finish."

Step 3: Add Hadoop Libraries

Right-click on your project, go to Properties -> Java Build Path -> Libraries, and click on "Add External JARs." Add the required Hadoop libraries to your project. You'll need the Hadoop core library, commonly named "hadoop-core-X.X.X.jar" (replace X.X.X with the version number), and any additional dependencies you might need for your specific use case.

Step 4: Create the MapReduce Classes

Right-click on your project and select New -> Class. Create three classes: WeatherMapper, WeatherReducer, and WeatherDriver. These classes will define the Map, Reduce, and driver logic, respectively.

Step 5: Implement the WeatherMapper Class

In the WeatherMapper class, implement the org.apache.hadoop.mapreduce.Mapper interface. Override the map() method to extract the required attributes (temperature, dew point, wind speed) from the input record and emit key-value pairs.

Step 6: Implement the WeatherReducer Class

In the WeatherReducer class, implement the org.apache.hadoop.mapreduce.Reducer interface. Override the reduce() method to calculate the average for each attribute based on the values received from the mapper.

Step 7: Implement the WeatherDriver Class

In the WeatherDriver class, configure the MapReduce job. Create an instance of org.apache.hadoop.mapreduce.Job, set the input and output paths, specify the mapper and reducer classes, and any additional configurations required.

Step 8: Run the Application

In the WeatherDriver class, add a main() method. Within the main() method, create a new instance of the WeatherDriver class and call the run() method on it. This will submit the MapReduce job for execution.

Step 9: Configure Input and Output Paths

Ensure that you have a text file containing weather data to be processed. Update the WeatherDriver class to set the appropriate input path for the MapReduce job, specifying the location of the weather data file. Similarly, set the output path where the results will be stored.

Step 10: Execute the MapReduce Job

Right-click on the WeatherDriver class and select "Run As" -> "Java Application" to execute the MapReduce job. Observe the console for job progress and completion.

Step 11: Process the Output

Once the job completes successfully, you can access the output, which will contain the average values for temperature, dew point, and wind speed. You can further process this data as per your requirements, such as generating reports or performing additional analysis.

Remember to handle any exceptions and error cases appropriately, and make sure your Eclipse environment is properly configured to run MapReduce applications using Hadoop.

Please note that the above steps provide a general outline, and you'll need to adapt the implementation based on your specific requirements and the dataset format.

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WeatherMapper extends Mapper<LongWritable, Text, Text, TemperatureData> {

@Override

protected void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

// Split the input record into fields

String[] fields = value.toString().split(",");

// Extract the necessary attributes (temperature, dew point, wind speed)

double temperature = Double.parseDouble(fields[1]);

double dewPoint = Double.parseDouble(fields[2]);

double windSpeed = Double.parseDouble(fields[3]);

// Create a TemperatureData object with the extracted values

TemperatureData temperatureData = new TemperatureData(temperature, dewPoint, windSpeed);

// Emit the attribute as the key and the TemperatureData object as the value

context.write(new Text("temperature"), temperatureData);

context.write(new Text("dewPoint"), temperatureData);

context.write(new Text("windSpeed"), temperatureData);

}

}

///////////////////////

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WeatherReducer extends Reducer<Text, TemperatureData, Text, Double> {

@Override

protected void reduce(Text key, Iterable<TemperatureData> values, Context context)

throws IOException, InterruptedException {

int count = 0;

double sum = 0.0;

// Iterate through the values and calculate the sum

for (TemperatureData data : values) {

count++;

if (key.toString().equals("temperature")) {

sum += data.getTemperature();

} else if (key.toString().equals("dewPoint")) {

sum += data.getDewPoint();

} else if (key.toString().equals("windSpeed")) {

sum += data.getWindSpeed();

}

}

// Calculate the average

double average = sum / count;

// Emit the key and average as the output

context.write(key, average);

}

}

////////////////////////////////////////////

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

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;

public class WeatherDriver {

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

// Create a new job

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Weather Data Processing");

// Set the driver class

job.setJarByClass(WeatherDriver.class);

// Set the Mapper and Reducer classes

job.setMapperClass(WeatherMapper.class);

job.setReducerClass(WeatherReducer.class);

// Set the types of the output key-value pairs

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

// Set the input and output paths

FileInputFormat.addInputPath(job, new Path("input/weather\_data.txt"));

FileOutputFormat.setOutputPath(job, new Path("output"));

// Submit the job for execution and wait for completion

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

}

}

////////////////////////////////////

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class TemperatureData implements Writable {

private double temperature;

private double dewPoint;

private double windSpeed;

public TemperatureData() {

// Default constructor required for serialization

}

public TemperatureData(double temperature, double dewPoint, double windSpeed) {

this.temperature = temperature;

this.dewPoint = dewPoint;

this.windSpeed = windSpeed;

}

// Getters and setters

public double getTemperature() {

return temperature;

}

public void setTemperature(double temperature) {

this.temperature = temperature;

}

public double getDewPoint() {

return dewPoint;

}

public void setDewPoint(double dewPoint) {

this.dewPoint = dewPoint;

}

public double getWindSpeed() {

return windSpeed;

}

public void setWindSpeed(double windSpeed) {

this.windSpeed = windSpeed;

}

// Serialization and deserialization

@Override

public void write(DataOutput out) throws IOException {

out.writeDouble(temperature);

out.writeDouble(dewPoint);

out.writeDouble(windSpeed);

}

@Override

public void readFields(DataInput in) throws IOException {

temperature = in.readDouble();

dewPoint = in.readDouble();

windSpeed = in.readDouble();

}

}