Chapter 37
Big Data Analysis

► Part I. Preliminaries
► Part II. Tightly Coupled Multicore
► Part III. Loosely Coupled Cluster
► Part IV. GPU Acceleration
► Part V. Big Data
  Chapter 35. Basic Map-Reduce
  Chapter 36. Cluster Map-Reduce
  Chapter 37. Big Data Analysis
  Chapter 38. Map-Reduce Meets GIS
► Appendices
My next parallel programming example will be another big data analysis program using the map-reduce paradigm. The program itself is not all that different from the previous concordance program and web log analysis programs. What differs is the scale of the data.

The U.S. government’s National Oceanic and Atmospheric Administration (NOAA) runs the National Climatic Data Center (NCDC). NCDC maintains the Global Historical Climatology Network (GHCN), which collects and archives climate related data from weather stations all over the globe. NCDC publishes a data set of historical data from the GHCN going all the way back to before 1900, available on the web.* The GHCN data set consists of over 106,000 text files totaling 27.8 gigabytes. Google would consider this a puny data set, but to me that’s big data, or at least middling big data.

Here is a portion of one of the GHCN data set files:

```
AEM0004194198301TMAX  276  S  302  S  252  S  235  S-9999
AEM00041194198301TMIN  140  S  134  S  143  S  158  S  145  S
AEM00041194198301TAVG  195H S  198H S  191H S  194H S  185H S
AEM00041194198302TMAX  196  S  210  S  201  S  195  S  212  S
AEM00041194198302TMIN-9999     111  S   99  S-9999     117  S
AEM00041194198302TAVG  164H S  154H S  157H S  151H S  171H S
AEM00041194198303TMAX  192  S  199  S  208  S  222  S  297  S
AEM00041194198303TMIN-9999     128  S  123  S  120  S-9999
AEM00041194198303TAVG  162H S  168H S  162H S  168H S  232H S
```

Each line contains climate data recorded at a particular weather station for each day of a particular month and year. (The lines are too long to display in their entirety.) The first eleven characters are the weather station identifier, AEM00041194 in this file, which happens to be the Dubai International Airport. The next four characters are the year, 1983. The next two characters are the month: January (01), February (02), and so on. The next four characters indicate the kind of climate data on the line: TMAX says the line contains maximum temperatures, TMIN is minimum temperatures, TAVG is average temperatures. (Some files include other kinds of climate data too.) The rest of the line consists of 31 eight-character fields, one for each day of the month. The first five characters of the field are the temperature reading in units of 0.1 Celsius. A value of -9999 indicates a missing or nonexistent reading. The seventh character of the field is a space if the reading is valid, or is some non-space character if the reading is not valid. (The sixth and eighth characters of the field do not concern us.) Thus, at weather station AEM00041194, the maximum temperature recorded on January 1, 1983 was 27.6 C; on January 2, 30.2 C; on February 1, 19.6 C; and so on.

I want to write a program that calculates the average maximum temperature for each year in the data set. The program’s output should look like this:

package edu.rit.pjmr.example;
import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Plot;
import edu.rit.pj2.vbl.DoubleVbl;
import edu.rit.pjmr.Combiner;
import edu.rit.pjmr.Customizer;
import edu.rit.pjmr.Mapper;
import edu.rit.pjmr.PjmrJob;
import edu.rit.pjmr.Reducer;
import edu.rit.pjmr.TextDirectorySource;
import edu.rit.pjmr.TextId;
import java.io.File;
import java.io.IOException;
import java.util.Date;
public class MaxTemp01
extends PjmrJob<TextId,String,Integer,DoubleVbl>
{
   // PJMR job main program.
   public void main
   (String[] args)
   {
      // Parse command line arguments.
      if (args.length != 5) usage();
      String[] nodes = args[0].split (",");
      String directory = args[1];
      int yearlb = Integer.parseInt (args[2]);
      int yearub = Integer.parseInt (args[3]);
      String plotfile = args[4];

      // Determine number of mapper threads.
      int NT = Math.max (threads(), 1);

      // Print provenance.
      System.out.printf
      ("$ java pj2 threads=%d edu.rit.pjmr.example.MaxTemp01",
       NT);
      for (String arg : args)
         System.out.printf (" %s", arg);
      System.out.println();
      System.out.printf ("%s%n", new Date());
      System.out.flush();

      // Configure mapper tasks.
      for (String node : nodes)
         mapperTask (node)
            .source (new TextDirectorySource (directory))
            .mapper (NT, MyMapper.class, ""+yearlb, ""+yearub);

      // Configure reducer task.
      reducerTask()
         .runInJobProcess()
         .customizer (MyCustomizer.class)
         ..reducer (MyReducer.class, plotfile);
      startJob();
   }

Listing 37.1. MaxTemp01.java (part 1)
To produce the first line of output, the program considers all year-1900 TMAX records in all files of the data set; extracts all the temperature readings from these records, omitting missing and invalid readings; averages these readings; and prints the year (1900) followed by the average (17.7 C). The program does the same for 1901, 1902, and so on. In fact, I want to specify the range of years the program should analyze.

American mathematician Richard W. Hamming (1915–1998) famously said, “The purpose of computing is insight, not numbers.”* A list of numbers, like the program output above, doesn’t give much insight. To get better insight into climate trends, I also want the analysis program to generate a plot of average maximum temperature versus year.

The climate data analysis program is edu.rit.pjmr.example.MaxTemp01 (Listing 37.1). The program is run with this command:

```bash
$ java pj2 [threads=NT] edu.rit.pjmr.example.MaxTemp01 nodes \ directory yearlb yearub plotfile
```

The program’s command line arguments are as follows:

- **NT** is the number of mapper threads in each mapper task (default: 1).
- **nodes** is a comma-separated list of one or more backend node names in the cluster. The program will run a mapper task on each node.
- **directory** is the name of the directory containing the climate data files to be analyzed. The program assumes that each node has a directory with this name, containing a portion of the climate data files.
- **yearlb** is the first year to be analyzed.
- **yearub** is the last year to be analyzed.
- **plotfile** is the name of the file in which to store the plot of average maximum temperature versus year.

The main program class (line 15) extends class PjmrJob (line 16). As in the previous chapters, the source’s record ID and record contents data types are TextId and String. The program will be mapping years (integers) to temperatures in units of 0.1 Celsius (floating point numbers). So for the mapper’s (K,V) data types, I will use class Integer (in the Java platform) for the

Listing 37.1. MaxTemp01.java (part 2)
key $K$ and class DoubleVbl (in the Parallel Java 2 Library) for the value $V$.  
Class Integer wraps a value of type int. Class DoubleVbl provides a reduction variable that wraps a value of type double.

After parsing the command line arguments and printing the provenance, the program configures one mapper task to run on each specified backend node of the cluster (lines 44–47). The mapper task’s source is an instance of class edu.rit.pjmr.TextDirectorySource (line 46). Whereas class TextFileSource (in the previous chapters) obtains records from just one file, class TextDirectorySource obtains records from all the files in the given directory; each record is one line from one of the files; the order in which the files are read is not specified. That’s appropriate for the GHCN data set, which consists of a bunch of files rather than one file. The mapper task is configured with $NT$ mapper objects, each an instance of class MyMapper (line 47). The lower and upper bounds of the range of years to be analyzed are passed as arguments to each mapper object.

The program configures one reducer task with a customizer object and a reducer object (lines 50–53). The plot file name is passed as an argument to the reducer object. The reducer task is run in the job process on the cluster’s frontend node (line 51) so the reducer task can write the plot file in the user’s account.

The mapper class’s `start()` method (line 75) stores the lower and upper bound years for later use. The `map()` method (line 84) processes one record (line) from a climate data set file. Only $T_{MAX}$ records whose year lies in the specified range are analyzed. For each temperature reading that is not missing and that is valid, the `map()` method adds a $(K,V)$ pair to the combiner. The key $K$ is the year. The value $V$ is the temperature reading, as an instance of class DoubleVbl.Mean. This class’s reduction operation computes the average (mean) of all the values that are fed into a reduction variable.

The customizer class’s `comesBefore()` method (line 135) causes the reducer task to sort the $(K,V)$ pairs in the overall combiner into ascending order of year. This is the order I want for the program’s printout and plot.

The MaxTemp01 program’s reducer class (line 148) does a bit more than the reducer classes in the previous chapters. Besides printing the results, the reducer task creates a plot of the results. To do so, the reducer class uses two classes from the Parallel Java 2 Library: class edu.rit.numeric.ListXYSeries and class edu.rit.numeric.plot.Plot.

Class ListXYSeries encapsulates a series of two-dimensional points $(x,y)$. The $x$ and $y$ coordinates are type double. The class has methods to add a point to the series, retrieve a point from the series, and so on. The points stored in the series specify the points to appear on the plot.

Class Plot encapsulates a plot. The class has methods to set a title for the plot; set the length, title, lower and upper bounds, and so on of the X axis and the Y axis; as well as other methods too numerous to mention here. (Refer to
catch (NumberFormatException exc) {
    continue;
}
char qflag = data.charAt (i + 6);
if (tmax != -9999 && qflag == ' ')
    combiner.add (year, new DoubleVbl.Mean (tmax));
}

// Parse an integer, ignoring leading and trailing whitespace.
private static int parseInt (String s, int from, int to) {
    return Integer.parseInt (s.substring (from, to) .trim());
}

// Reducer task customizer class.
private static class MyCustomizer extends Customizer<Integer,DoubleVbl> {
    // Sort into ascending order of year (key).
    public boolean comesBefore (Integer key_1, DoubleVbl value_1, 
                                Integer key_2, DoubleVbl value_2) {
        return key_1 < key_2;
    }
}

// Reducer class.
private static class MyReducer extends Reducer<Integer,DoubleVbl> {
    File plotfile;
    ListXYSeries data;
    // Initialize data series.
    public void start (String[] args) {
        plotfile = new File (args[0]);
        data = new ListXYSeries();
    }
    // Print the year (key) and the average maximum temperature (value).
    public void reduce (Integer key, 
                        DoubleVbl value) {
        int year = key.intValue();
        double mean = value.doubleValue()/10.0;
        data.add (year, mean);
        System.out.printf ("%d\t%.1f\n", year, mean);
        System.out.flush();
    }
}

Listing 37.1. MaxTemp01.java (part 3)
the Javadoc for class Plot.) The class also has a method to add a data series to
the plot; the data is stored in a ListXYSeries object. You can specify the
shape and color of the dots that are plotted; you can specify the width and
color of the lines that connect the dots; you can specify to omit the dots or
the lines. The default settings usually suffice.

Once you have set up a Plot object, you can display it in a window on the
screen. You can also store the plot in a file. Later, you can view the plot with
this command, which runs the View program in the Parallel Java 2 Library:

$ java View plotfile

The View program reads the plot from the given file and displays it in a win-
dow. The window has menus that let you alter the plot’s appearance, store the
plot as a PNG image file, store the plot as a PostScript file, and so on. (I’ve
used class Plot to create all the plots in this book.)

Returning to the code, the reducer task calls the reducer’s start() method
(line 155) after all the mappers’ (K,V) pairs have been absorbed into
the overall combiner. The start() method stores the plot file name passed in
as an argument and creates a ListXYSeries object to hold the data for the av-
erage maximum temperature versus year plot.

The reducer task calls the reducer’s reduce() method (line 163) for each
(K,V) pair from the overall combiner, K being the year and V being the aver-
age maximum temperature for that year in units of 0.1 Celsius. The reducer
task presents the (K,V) pairs to the reduce() method in sorted order, that is,
in ascending order of year. To convert the temperature value from units of 0.1
Celsius to units of Celsius, the reduce() method divides the temperature
value by 10. The reduce() method prints the year and the temperature. The
reduce() method also adds an (x,y) point to the plot series, with x being the
year and y being the temperature.

After that, the reducer task calls the reducer’s finish() method (line
174). The finish() method creates a Plot object; sets the plot’s X and Y axis
titles; turns off plotting dots for the data points, so that only the connecting
lines will be plotted; and adds the data accumulated in the ListXYSeries ob-
ject to the plot. The finish() method then stores the plot in the given file.

I downloaded the GHCN data set from the NCDC web site and stored
one-tenth of the climate data files on the disk drives of each of the ten back-
end nodes of the tardis cluster. Then I ran the following command to find
the average maximum temperature for each year from 1900 through 2017. I
stored the plot in a “drawing file” named MaxTemp01.dwg. I configured the
mapper tasks with two mapper objects running in two threads (cores) on each
node. The program’s running time was 192 seconds.

$ java pj2 threads=2 edu.rit.pjmr.example.MaxTemp01 \ 
dr00,dr01,dr02,dr03,dr04,dr05,dr06,dr07,dr08,dr09 \ /
/var/tmp/arch/NOAA/ghcnd_all 1900 2017 MaxTemp01.dwg
Figure 37.1 shows the plot the program produced. Despite year-to-year fluctuations, some overall trends are apparent. The worldwide average maximum temperature trended downwards until about 1970, flattened out until about 1990, then started trending upwards at a faster rate. Recalling Hamming’s maxim, this is an insight that would be difficult to see in a list of numbers. I personally believe the GHCN climate data shows evidence of global warming; you may disagree. Still, this example shows how a parallel map-reduce program can extract meaningful insights from a large data set.

Listing 37.1. MaxTemp01.java (part 4)
Points to Remember

• “The purpose of computing is insight, not numbers.”
  —Richard W. Hamming
• Consider whether presenting analysis results in the form of plots might yield better insights. However, do present analysis results as tables of numbers also.
• Use the Parallel Java 2 Library classes edu.rit.numeric.plot.Plot, edu.rit.numeric.ListXYSeries, and other classes in package edu.rit.numeric to create plots right in your program.
• Store a plot object in a file for later viewing. Use the View program to view the plot.
• A task that needs to read or write files in the user’s account should be specified to run in the job process on the cluster’s frontend node.