Experiment No.2

MapReduce Program on a Real Time Dataset

**Saumitra Chaskar**

**BE COMPS  
2017130011**

**Aim:** To implement a MapReduce program on a real time dataset.

**Theory:**

**MapReduce:**

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

* The map task is done by means of Mapper Class
* The reduce task is done by means of Reducer Class.

Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.

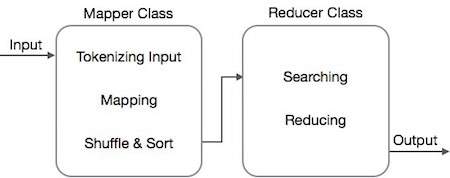


Fig.1 General MapReduce Approach

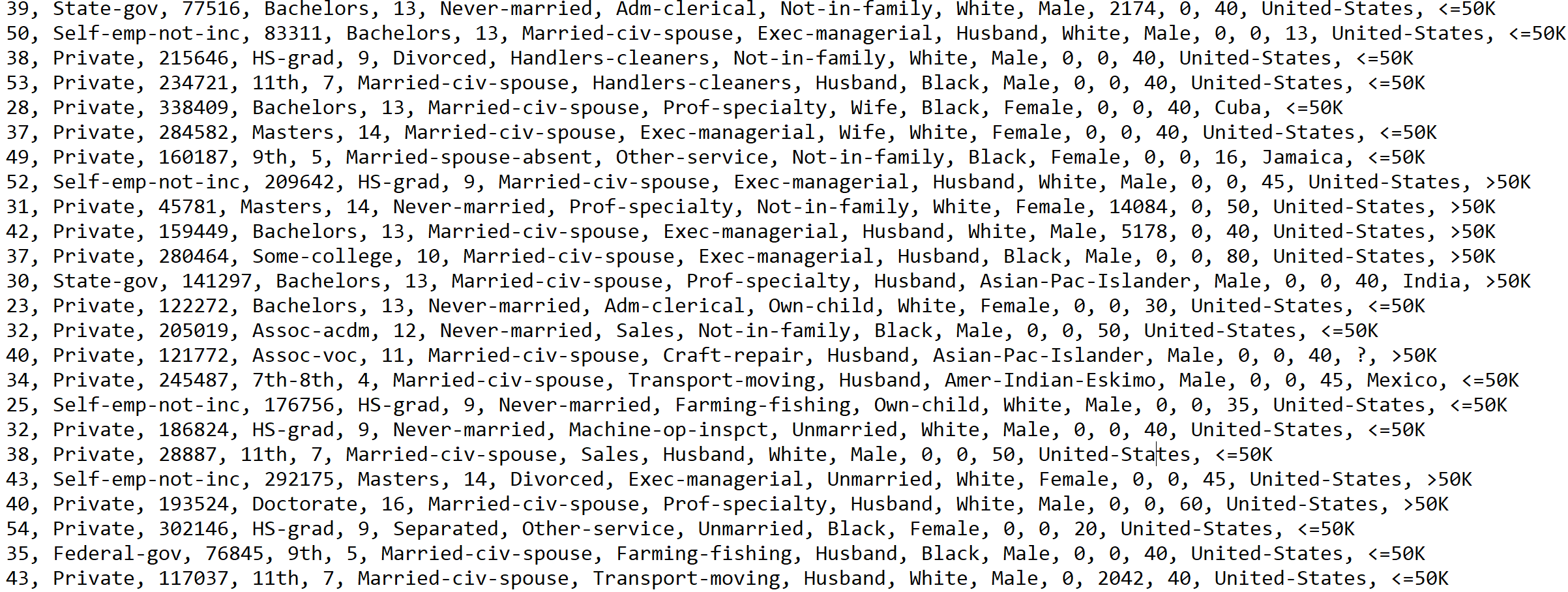
MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems.

These mathematical algorithms may include the following −

* Sorting
* Searching
* Indexing

**Dataset:**

The 1994 US Census database (subset).



**Problem Definition:**

To calculate average working hours per week for different marital statuses using MapReduce Algorithm.

**Implementation:**

**1. Setup Environment.**

Setup Hadoop MapReduce environment for java by adding Hadoop JAR files in your java project.

Common JARs

MapReduce JARs

Yarn JARs

HDFS JARs

**2. Creating a Mapper.**

The Map class will implement a **map function** that will take as input sequentially, lines from the input dataset line by line.

**Mapper Input:**

LongWritable -> (Line Number)

Text -> (Line Content)

A) The marital status column is the 6th Column (index 5).

B) The working hours column is the 13th Column (index 12)

A and B are extracted for each of the input record (line) and then written to the context.

**Mapper Output:**

Text -> (Marital Status)

DoubleWritable -> (Working Hours)

These output pairs are used by the reducer to reduce the data obtained as per their keys.

**A) Map.java**

package summaries;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class Map extends Mapper<LongWritable,Text,Text,DoubleWritable>

{

@Override

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

{

String line = value.toString();

String[] data = line.split(",");

try

{

String maritalStatus = data[5];

Double hrs = Double.parseDouble(data[12]);

context.write(new Text(maritalStatus), new DoubleWritable(hrs));

}

catch(Exception e)

{

System.out.println(e);

}

}

}

**3. Creating a Reducer**

The Reducer class will implement a **reduce function** that will take as input, the output obtained from the mapper function. The input will consist of a key and all the values for that particular key obtained by combining the results from the mapper function (Iterable).

**Reducer Input:**

Text -> (Marital Status)

Iterable<DoubleWritable> -> (List of Working hours)

The reduce function will iterate through all the key value pairs and compile the result. The key value will be the Marital Status and the average for that key will be ontained by calculating the average of the Working Hours in the value field.

**Reducer Output:**

**B) Reduce.java**

Text -> (Marital Status)

DoubleWritable -> (Working Hours)

These output pairs are then written to the context and will form the output for the entire MapReduce algorithm.

package summaries;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class Reduce extends Reducer<Text,DoubleWritable,Text,DoubleWritable>

{

@Override

public void reduce(final Text key,

final Iterable<DoubleWritable> values,

final Context context) throws IOException, InterruptedException

{

Double sum = 0.0;

Integer count = 0;

for(DoubleWritable value : values)

{

sum += value.get();

count += 1;

}

Double ratio = sum / count;

context.write(key, new DoubleWritable(ratio));

}

}

**3. Runner Program.**

The Main program will create a MapReduce Job called **average**.

The final **Output Key Class** and the **Output Value Class** are specified in the program.

The **Mapper Class** is set.

The **Reducer Class** is set.

The **Input** and **Output path** for the job is set.

Finally, the Main program is started using the ToolRunner.run function.

The output is written as a text file present in the output path.

**C) Main.java**

package summaries;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

public class Main extends Configured implements Tool

{

@Override

public int run(String[] args) throws Exception

{

Job job = Job.getInstance(getConf());

job.setJobName("average");

job.setJarByClass(Main.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

Path inputFilePath = new Path("D:/SometimeOf20/Coursera/MapReduce/PluralSight/data/input/census.txt");

Path outputFilePath = new Path("D:/SometimeOf20/Coursera/MapReduce/PluralSight/data/output");

FileInputFormat.addInputPath(job, inputFilePath);

FileOutputFormat.setOutputPath(job, outputFilePath);

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

}

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

{

int exitCode = ToolRunner.run(new Main(),args);

System.exit(exitCode);

}

}

**Output:**

Divorced 41.246680171055594

Married-AF-spouse 41.130434782608695

Married-civ-spouse 43.28485576923077

Married-spouse-absent 39.66985645933014

Never-married 36.9399981278667

Separated 39.30146341463415

Widowed 32.97985901309164

Marital Status and their corresponding Average Working Hours per Week.

**Conclusion:**

The MapReduce approach is a highly scalable processing technique for distributed computing. Instead of moving the data towards the computation, the computations are brought to the data nodes which improves the overall performance.

In the above implementation the map function extracts Marital Status and Working hours from the input records and then “average” as a aggregation function is implemented in the reduce phase which gives us the mean working hours per week for different Marital Statuses.