|  |
| --- |
| Máster en Big Data |
| Tecnologías de Almacenamiento |
| 1. Hands-On: Desarrollo MapReduce Avanzado |

Presentado por:

Albert Ripoll y Jose David Angulo

Índice

[1. Introducción 3](#_Toc507233944)

[2. Entorno de desarrollo 3](#_Toc507233945)

[3. Tool Runner y parámetros 5](#_Toc507233946)

[4. Combiner 9](#_Toc507233947)

[5. Partitioner 14](#_Toc507233948)

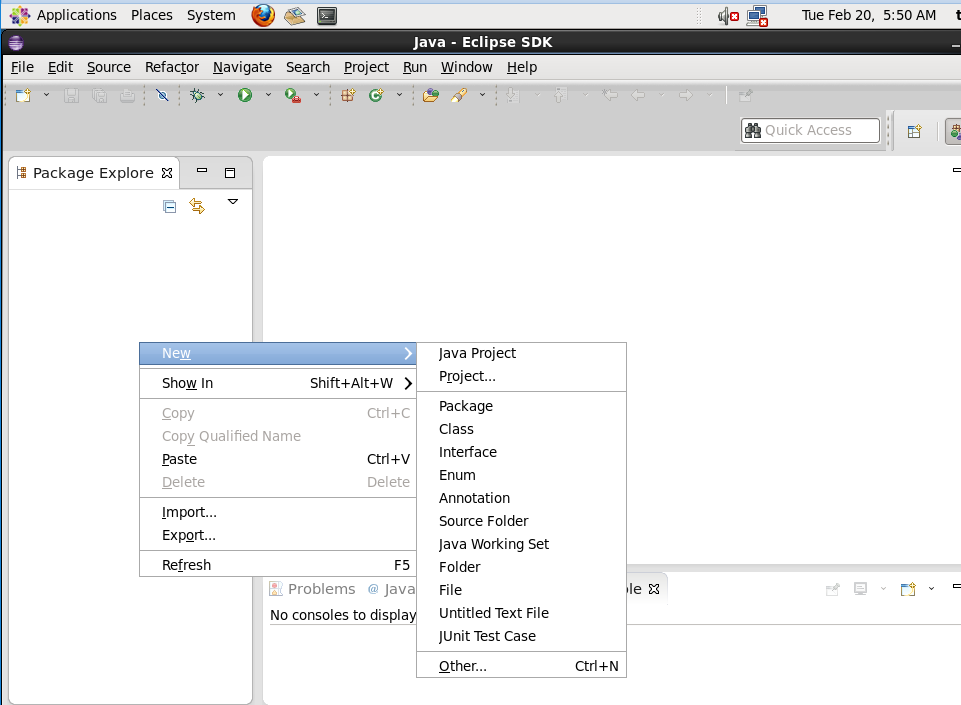
# Introducción

El objetivo de este Hands-On es poner en práctica conceptos avanzados en el desarrollo de Jobs de MapReduce

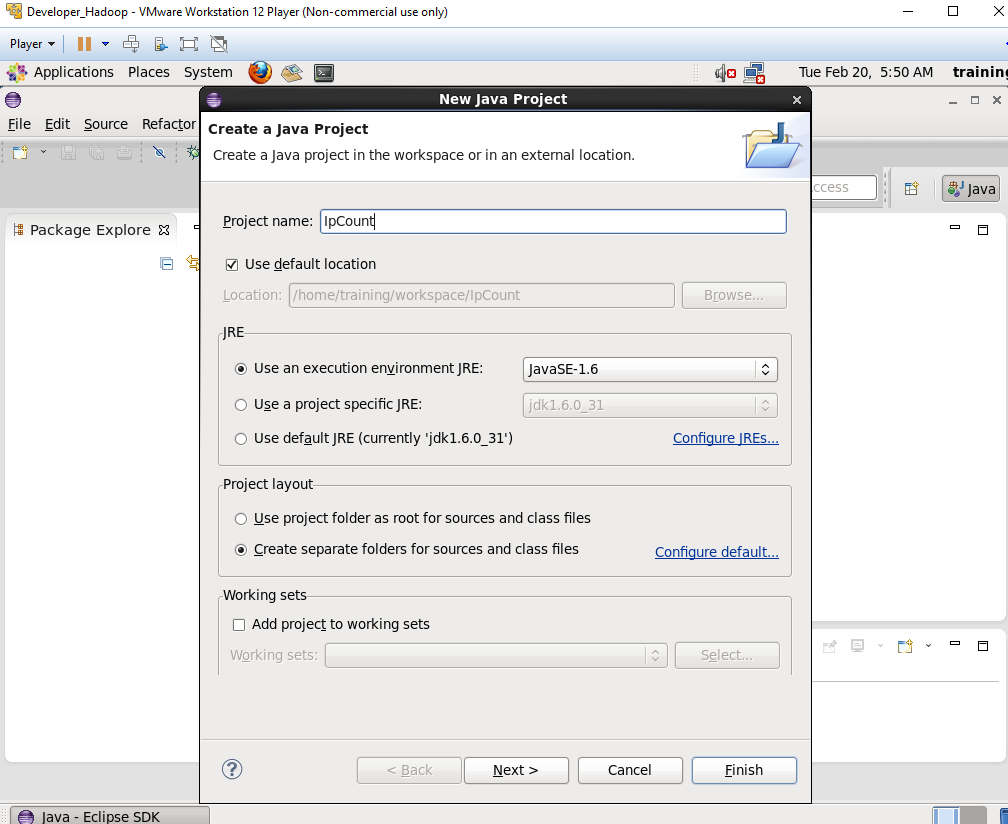
# Entorno de desarrollo

Para realizar el desarrollo lo haremos mediante el IDE Eclipse de la máquina virtual importada en ejercicios anteriores.

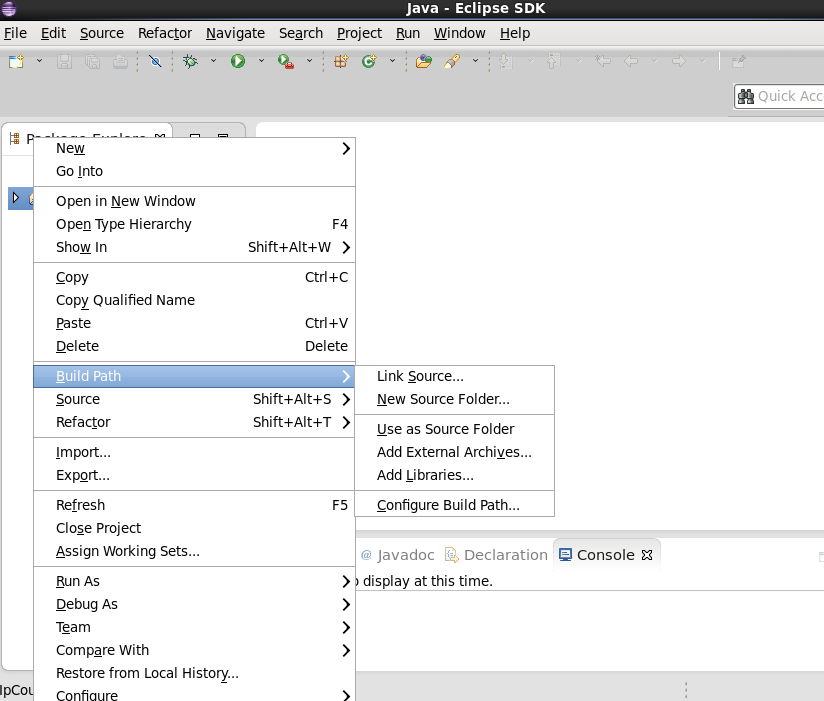
Para crear un nuevo proyecto, haremos click derecho sobre el package explorer New  Java Project



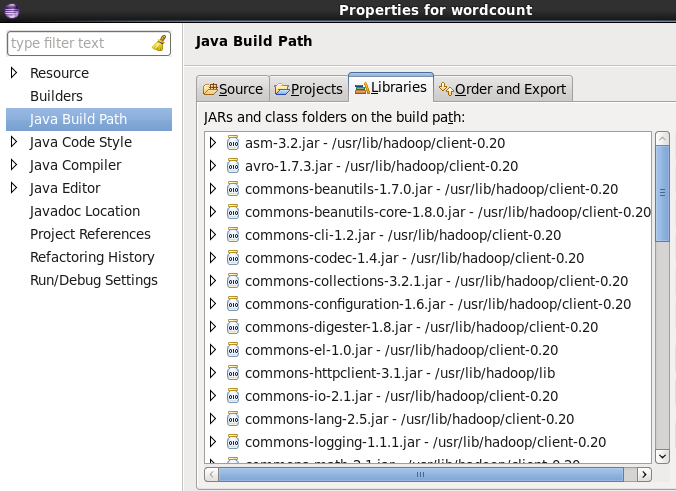
Introducimos el nombre del proyecto y click en Finish



Importamos manualmente las lilbrerías necesarias haciendo click derecho sobre el proyecto que acabamos de crear y seleccionamos Build Path  Configure Build Path



En la pestaña de libraries, seleccionamos Add Extertnal Jars e importamos todo el contenido de la carpeta /usr/lib/hadoop/client-0.20/



# Tool Runner y parámetros

Desarrollar y ejecutar el siguiente MapReduce:

Aprovechando el ejercicio del Hands-On anterior (**AvarageWordLength)** realizar las siguientes modificaciones:

* La clase driver use ToolRunner
* Modificar el Mapper para referenciar una variable booleana llamada caseSensitive. Si esta variable es true, el mapper no diferenciara entre mayúsculas ni minúsculas, si es false, hará una conversión de todas las letras a minúscula.

Creación del IPCOUNT5:

**package** LAB5;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.FloatWritable;

**import** org.apache.hadoop.io.Text;

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

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

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

//user/training/weblog/access\_log

**public** **class** IPCOUNT5 **extends** Configured **implements** Tool {

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

//int exitCode = ToolRunner.run(new Configuration(), new WordCount(), args);

**int** exitCode = ToolRunner.*run*(**new** Configuration(), **new** IPCOUNT5(), args);

System.*exit*(exitCode);

}

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

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

System.*out*.printf (

"Usage: %s [generic options] <input dir> <output dir>\n", getClass().getSimpleName());

**return** -1;

}

Job job = **new** Job (getConf());

job.setJarByClass (IPCOUNT5.**class**);

//job.setJobName ("Word Count");

job.setJobName ("Ip Driver");

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

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

job.setMapperClass (IPMAPER5.**class**) ;

job.setReducerClass (IPREDUCER5.**class**) ;

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

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

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

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

**boolean** success = job.waitForCompletion(**true**);

**return** success ? 0 : 1;

}

}

Creación del IPMAPER5:

**package** LAB5;

**import** java.io.IOException;

**import** org.apache.hadoop.io.FloatWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Mapper;

**public** **class** IPMAPER5 **extends** Mapper<LongWritable, Text, Text,

FloatWritable> {

**private** **boolean** caseSensitive;// estamos obteniendo el valor de la variable que es caseSensitive

**protected** **void** setup(Context context) **throws** IOException, InterruptedException {

caseSensitive = context.getConfiguration().getBoolean("caseSensitive",**true**);

}

@Override

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

**for** (String word: line.split("\\W+")){

**if** (word.length() > 0){

String letra;

**if** (caseSensitive){

letra = word.substring(0,1);

} **else** {

letra = word.substring(0,1).toLowerCase();

}

**float** longitud = word.length();

context.write(**new** Text(letra), **new** FloatWritable(longitud));

}

}

}

}

Creación del IPREDUCER5:

**package** LAB5;

**import** java.io.IOException;

**import** org.apache.hadoop.io.FloatWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Reducer;

**public** **class** IPREDUCER5 **extends** Reducer<Text, FloatWritable, Text, FloatWritable>

{

@Override

**public** **void** reduce (Text key, Iterable<FloatWritable> values, Context context)

**throws** IOException, InterruptedException {

**float** wordCount = 0;

**float** i=0;

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

wordCount += value.get();

i+=1;

}

**float** promedio = wordCount / i;

context.write (key, **new** FloatWritable (promedio));

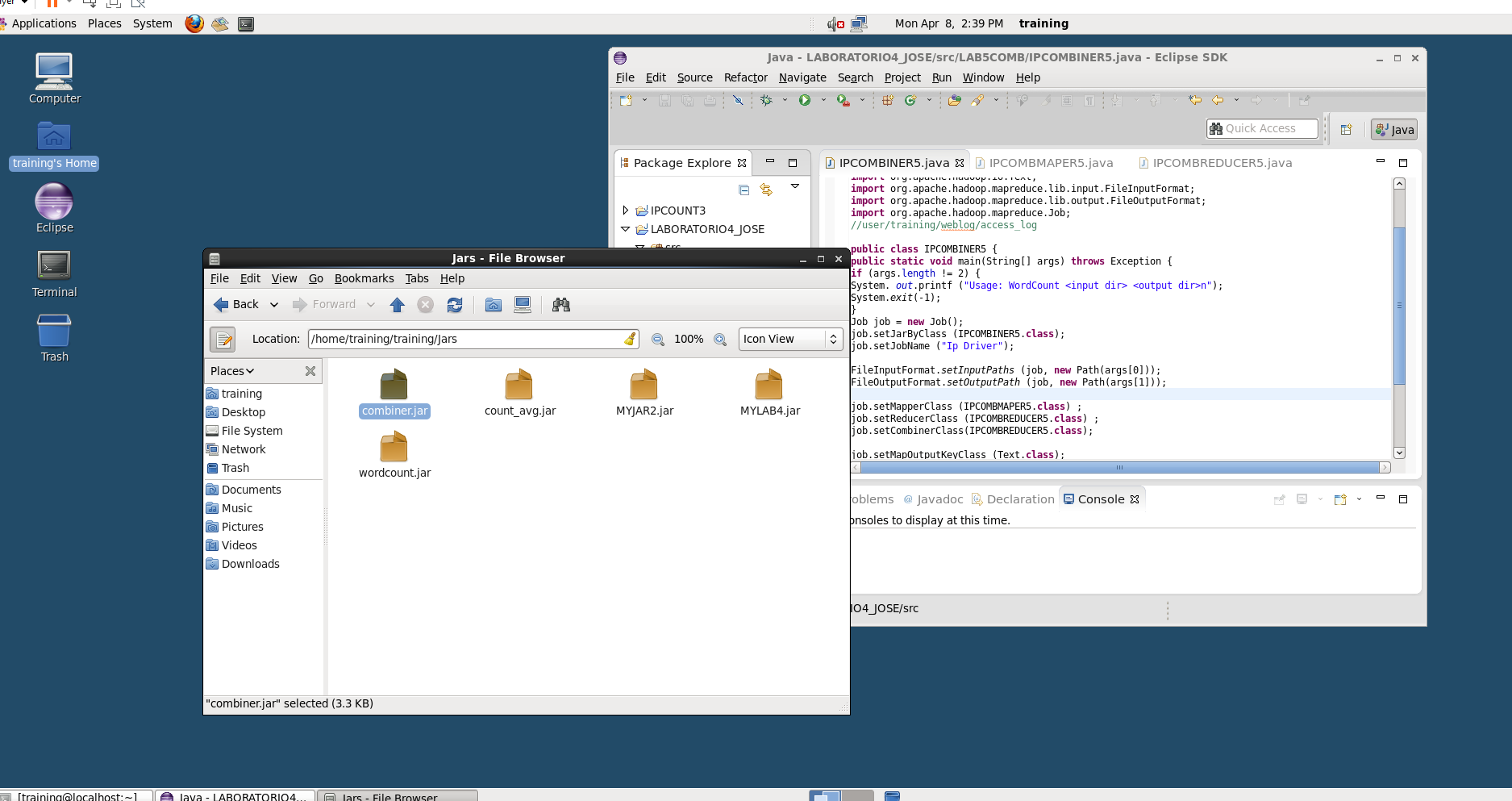
}

}

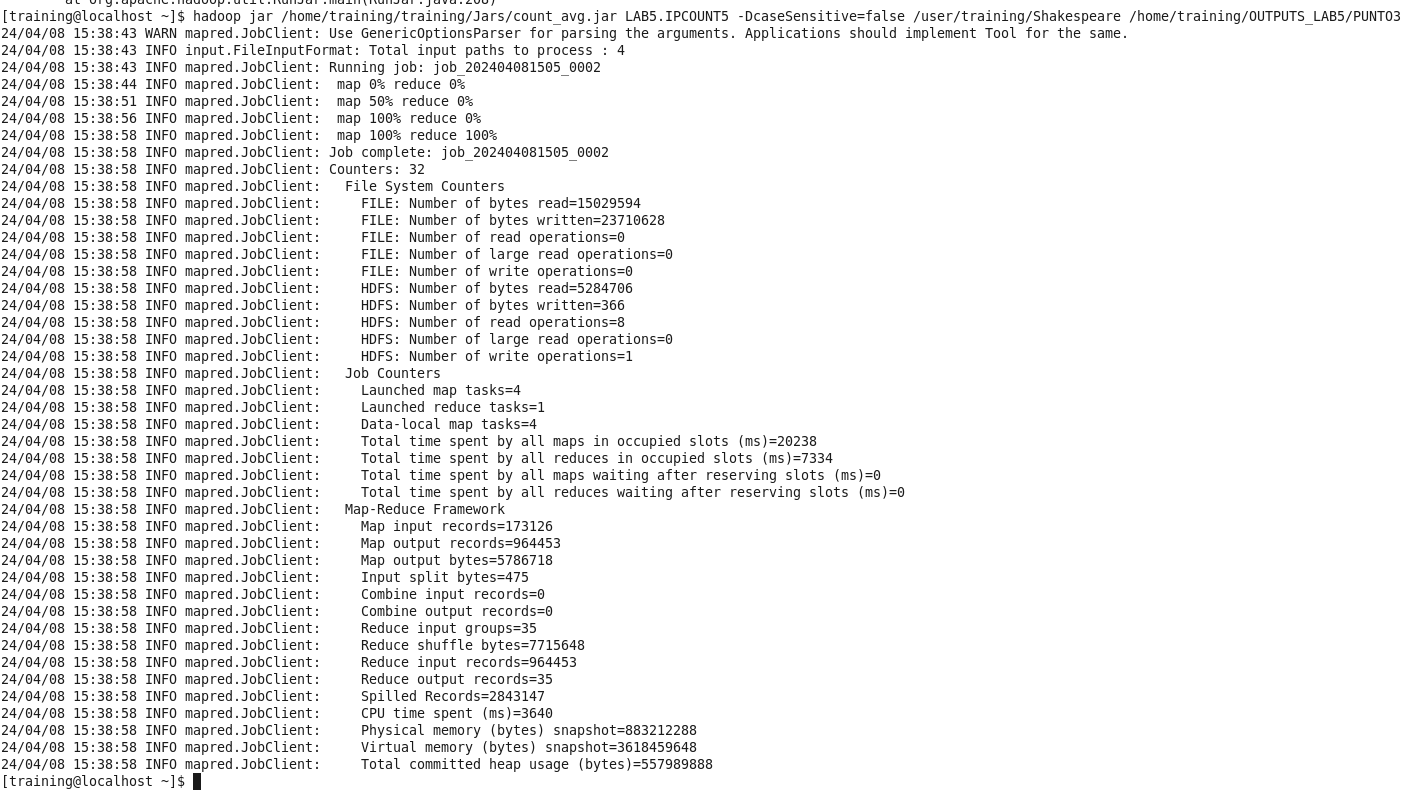
1. Para ejecutar primero creamos los JAR.export /training/jars/count\_avg.jar

Para hacerlo: Click derecho en el IPCOUNT5 >Export>Jar>Buscamos directorio donde está > y ponemos el nombre del jar como JAR.export /training/jars/count\_avg.jar

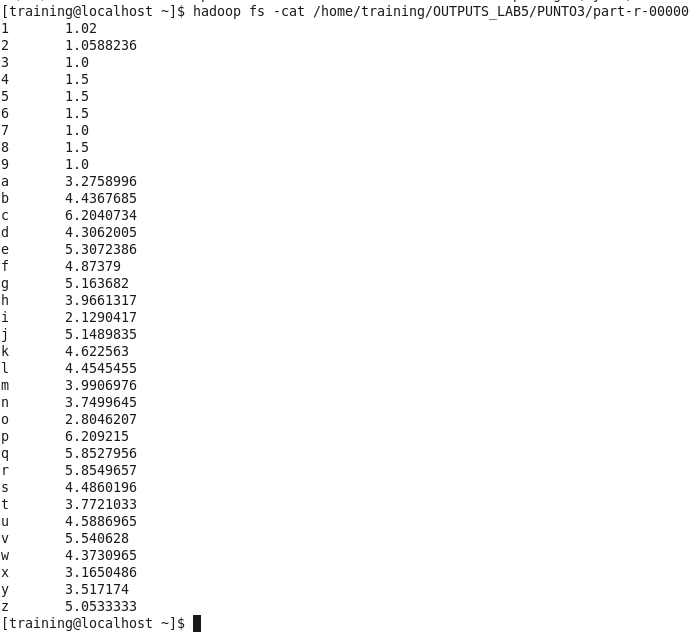
Esto nos quita tres pasos de ejecución respecto a los Hands On anteriores.



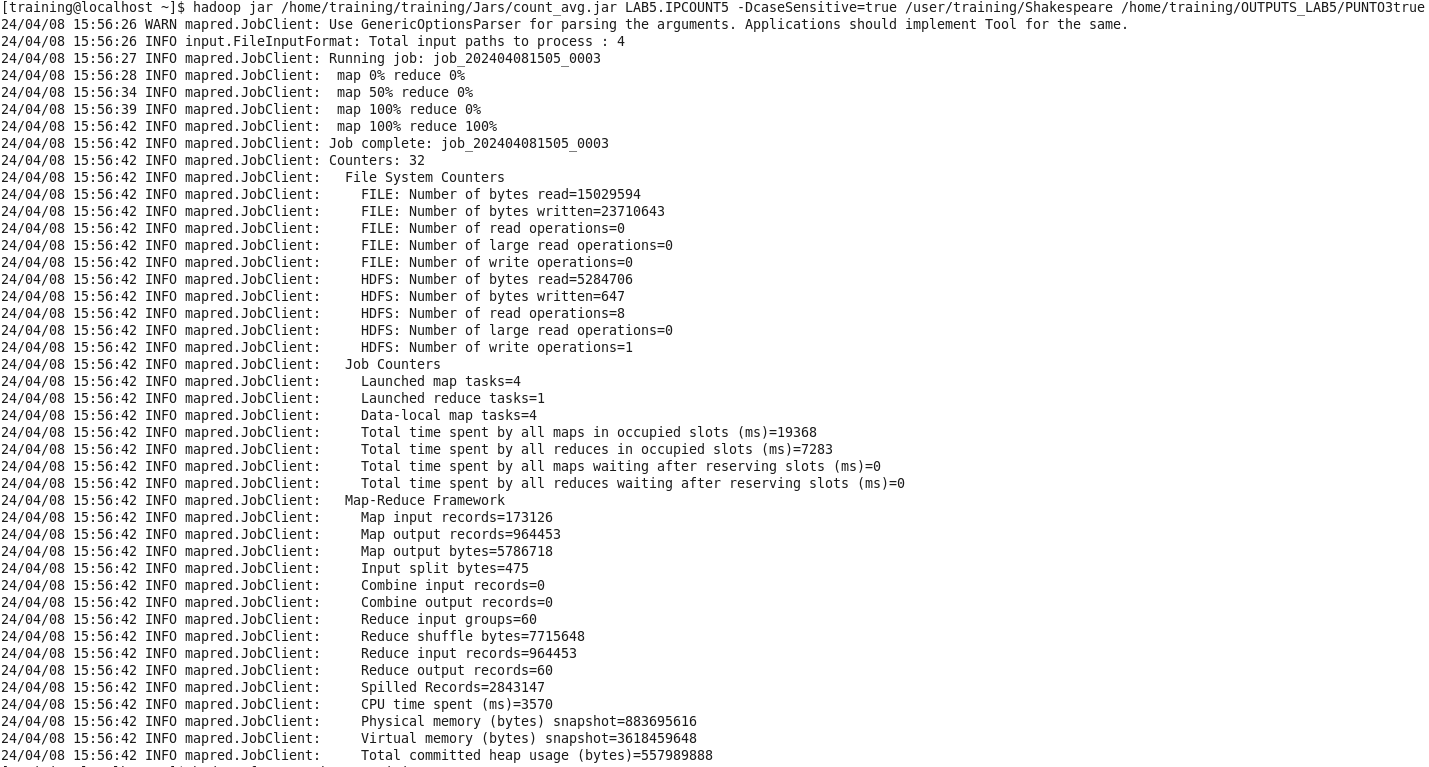
1. En la consola para ejecutar con el –DcaseSensitive=false:



1. En la consola para leer con el false:



1. En la consola para ejecutar con el –DcaseSensitive=true:



1. En la consola para leer con el true:

# 

# Combiner

Desarrollar y ejecutar el siguiente MapReduce:

Añadir un combiner al proyecto **IpCount** realizado en el Hands-On anterior

Copiamos el codigo del IPMAPER y IPREDUCER del Hands-On de la práctica 3 en un nuevo IPCOMBMAPER5 y IPCOMBREDUCER5. También copiamos el codigo del IPDRIVER de la práctica 3 en un nuveo IPCOMBINER5. A esa classe le añadimos la siguiente linea de código

*Job.setCombinerClass(IPCOMBREDUCER5.class);*

De esa forma tenemos:

• IPCOMBINER5:

**package** LAB5COMB;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

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

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

**import** org.apache.hadoop.mapreduce.Job;

//user/training/weblog/access\_log

**public** **class** IPCOMBINER5 {

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

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

System. *out*.printf ("Usage: WordCount <input dir> <output dir>n");

System.*exit*(-1);

}

Job job = **new** Job();

job.setJarByClass (IPCOMBINER5.**class**);

job.setJobName ("Ip Driver");

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

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

job.setMapperClass (IPCOMBMAPER5.**class**) ;

job.setReducerClass (IPCOMBREDUCER5.**class**) ;

job.setCombinerClass(IPCOMBREDUCER5.**class**);

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

job.setMapOutputValueClass (IntWritable.**class**);

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

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

Boolean success = job.waitForCompletion(**true**);

System.*exit*(success ? 0 : 1);

}

}

• IPCOMBMAPER5:

**package** LAB5COMB;

**import** java.io.IOException;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Mapper;

**public** **class** IPCOMBMAPER5 **extends** Mapper<LongWritable, Text, Text,

IntWritable> {

@Override

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

String[] parts = line.split(" - - ");

String word = parts[0];

context.write(**new** Text (word), **new** IntWritable(1));

}

}

• IPCOMBREDUCER5:

**package** LAB5COMB;

**import** java.io.IOException;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Reducer;

**public** **class** IPCOMBREDUCER5 **extends** Reducer<Text, IntWritable, Text, IntWritable>

{

@Override

**public** **void** reduce (Text key, Iterable<IntWritable> values, Context context)

**throws** IOException, InterruptedException {

**int** wordCount = 0;

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

wordCount += value.get();

}

context.write (key, **new** IntWritable (wordCount));

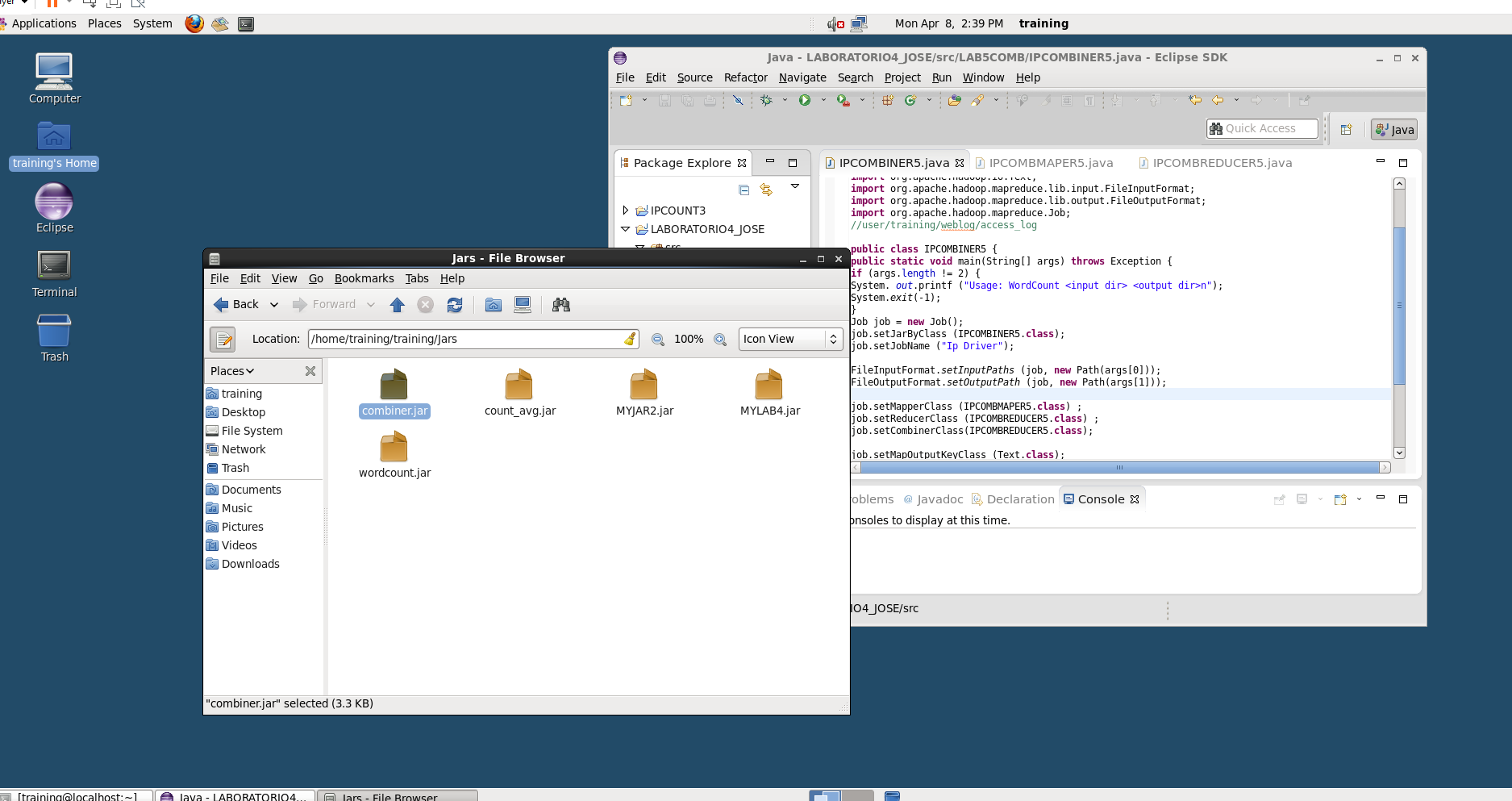
}

}

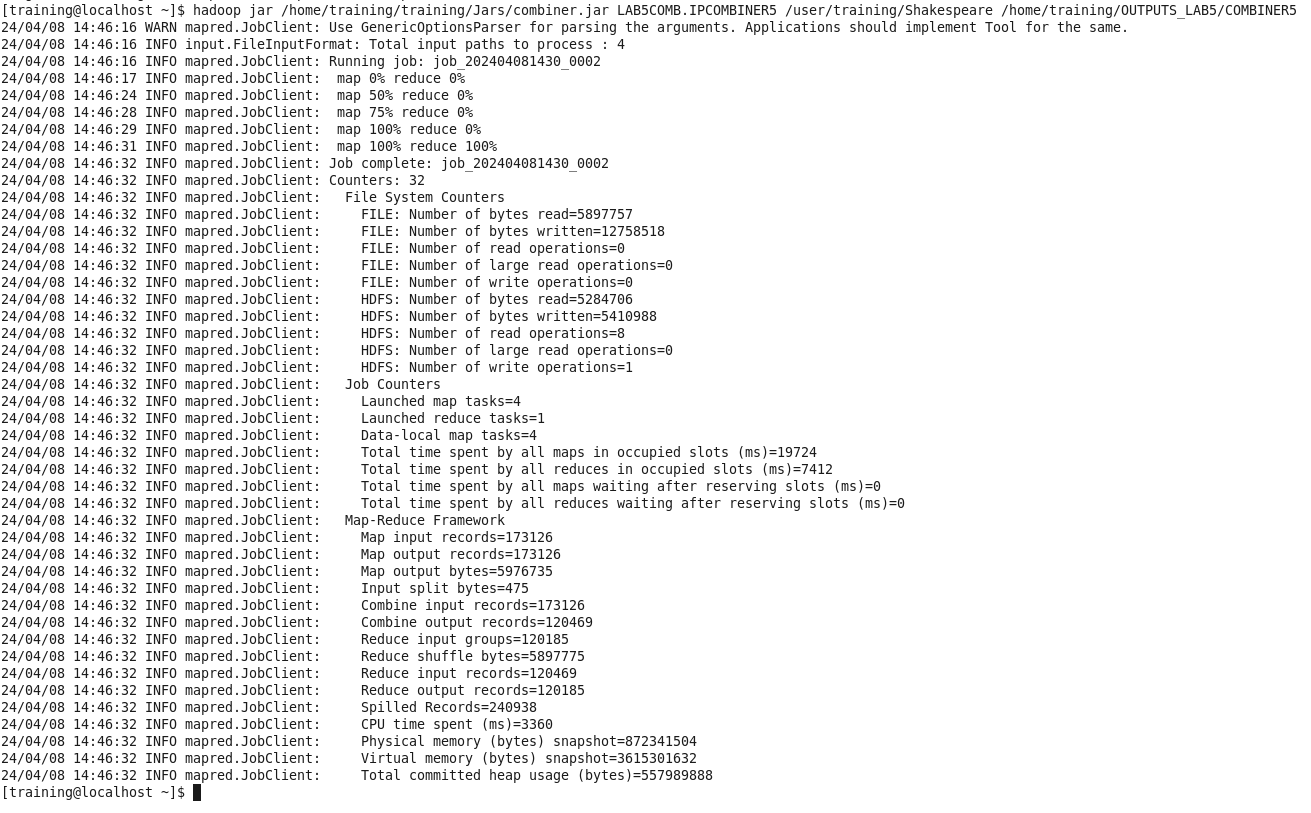
1. Crear los JAR.export /training/jars/count\_avg.jar

Para hacerlo: Click derecho el el IPCOMBINER>Export>Jar>Buscamos directorio donde está > y ponemos el nombre del jar como JAR.export /training/jars/combiner.jar

Esto nos quita tres pasos de ejecución respecto a los Hands On anteriores.



1. En la consola para ejecutar:



1. En la consola para leer/-cat:

# 

# Partitioner

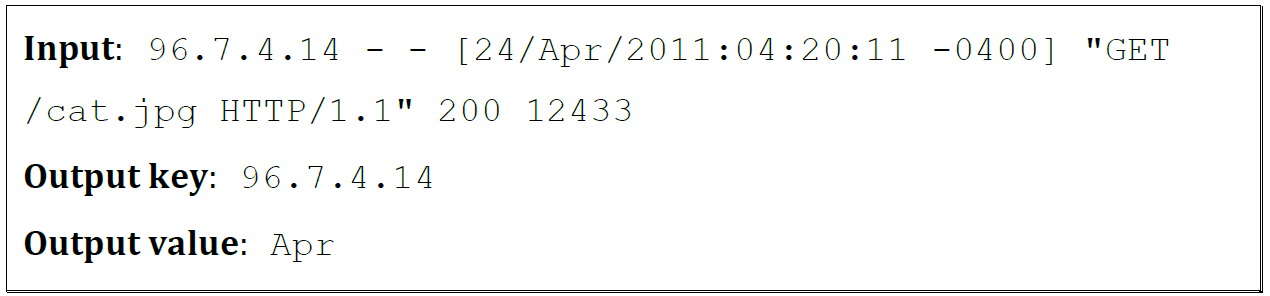
Desarrollar y ejecutar el siguiente MapReduce:

Aprovechando el proyecto original **IpCount** realizar los cambios pertinentes para escribir un Job con múltiples reducers e implementar un partitioner que redirija la salida según el mes del año hacia un reducer concreto.

Es decir, en total habrán 12 reducers (uno para cada mes del año) y el partitioner será el encargado de redirigir esa clave/valor hacia el reducer correcto.

La salida final consistirá en 12 ficheros, uno para cada mes del año, y contendrán el número de veces que se ha repetido la ip en ese mes del año.

Solución:



El código base original para el partitioner, el partitioner maper y el partitioner reducer es el siguiente. Hace falta modificarlo para hacer las 12 particiones que pide el enunciado.

• IPPARTITIONER5:

**package** LAB5PART;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

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

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

**import** org.apache.hadoop.mapreduce.Job;

//user/training/weblog/access\_log

**public** **class** IPPARTITIONER5 {

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

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

System. *out*.printf ("Usage: WordCount <input dir> <output dir>n");

System.*exit*(-1);

}

Job job = **new** Job();

job.setJarByClass (IPPARTITIONER5.**class**);

job.setJobName ("Ip Driver");

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

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

job.setMapperClass (IPPARTMAPER5.**class**) ;

job.setReducerClass (IPPARTREDUCER5.**class**) ;

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

job.setMapOutputValueClass (IntWritable.**class**);

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

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

Boolean success = job.waitForCompletion(**true**);

System.*exit*(success ? 0 : 1);

}

}

• IPPARTMAPER5:

**package** LAB5PART;

**import** java.io.IOException;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Mapper;

**public** **class** IPPARTMAPER5 **extends** Mapper<LongWritable, Text, Text,

IntWritable> {

@Override

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

String[] parts = line.split(" - - ");

String word = parts[0];

context.write(**new** Text (word), **new** IntWritable(1));

}

}

• IPPARTREDUCER5:

**package** LAB5PART;

**import** java.io.IOException;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce .Reducer;

**public** **class** IPPARTREDUCER5 **extends** Reducer<Text, IntWritable, Text, IntWritable>

{

@Override

**public** **void** reduce (Text key, Iterable<IntWritable> values, Context context)

**throws** IOException, InterruptedException {

**int** wordCount = 0;

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

wordCount += value.get();

}

context.write (key, **new** IntWritable (wordCount));

}

}

La modificación del código anterior para hacer las 12 particiones es:

• MONTHPARTITIONERTEST

**package solution;**

**import static org.junit.Assert.assertEquals;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.io.Text;**

**import org.junit.Test;**

**public class MonthPartitionTest {**

**static String[] months = {"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};**

**MonthPartitioner<Text,Text> mpart;**

**@Test**

**public void testMonthPartition() {**

**mpart=new MonthPartitioner<Text, Text>();**

**mpart.setConf(new Configuration());**

**int result;**

**for (int i = 0; i < months.length; i++) {**

**result = mpart.getPartition(new Text("foo"), new Text(months[i]), 12);**

**assertEquals(result,i);**

**}**

**}**

**}**

• MONTHPARTITIONER

**package solution;**

**import java.util.HashMap;**

**import org.apache.hadoop.io.Text;**

**import org.apache.hadoop.conf.Configurable;**

**import org.apache.hadoop.conf.Configuration;**

**import org.apache.hadoop.mapreduce.Partitioner;**

**public class MonthPartitioner<K2, V2> extends Partitioner<Text, Text> implements**

**Configurable {**

**private Configuration configuration;**

**HashMap<String, Integer> months = new HashMap<String, Integer>();**

**/\*\***

**\* Set up the months hash map in the setConf method.**

**\*/**

**@Override**

**public void setConf(Configuration configuration) {**

**this.configuration = configuration;**

**months.put("Jan", 0);**

**months.put("Feb", 1);**

**months.put("Mar", 2);**

**months.put("Apr", 3);**

**months.put("May", 4);**

**months.put("Jun", 5);**

**months.put("Jul", 6);**

**months.put("Aug", 7);**

**months.put("Sep", 8);**

**months.put("Oct", 9);**

**months.put("Nov", 10);**

**months.put("Dec", 11);}**

**/\*\***

**\* Implement the getConf method for the Configurable interface.**

**\*/**

**@Override**

**public Configuration getConf() {**

**return configuration; }**

**/\*\***

**\* You must implement the getPartition method for a partitioner class.**

**\* This method receives the three-letter abbreviation for the month**

**\* as its value. (It is the output value from the mapper.)**

**\* It should return an integer representation of the month.**

**\* Note that January is represented as 0 rather than 1.**

**\***

**\* For this partitioner to work, the job configuration must have been**

**\* set so that there are exactly 12 reducers.**

**\*/**

**public int getPartition(Text key, Text value, int numReduceTasks) {**

**return (int) (months.get(value.toString()));**

**}**

**}**

• LOGMONTHMAPPER

**package solution;**

**import java.io.IOException;**

**import java.util.Arrays;**

**import java.util.List;**

**import org.apache.hadoop.io.LongWritable;**

**import org.apache.hadoop.io.Text;**

**import org.apache.hadoop.mapreduce.Mapper;**

**public class LogMonthMapper extends Mapper<LongWritable, Text, Text, Text> {**

**public static List<String> months = Arrays.asList("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec");**

**/\*\***

**\* Example input line:**

**\* 96.7.4.14 - - [24/Apr/2011:04:20:11 -0400] "GET /cat.jpg HTTP/1.1" 200 12433**

**\***

**\*/**

**@Override**

**public void map(LongWritable key, Text value, Context context)**

**throws IOException, InterruptedException {**

**/\***

**\* Split the input line into space-delimited fields.**

**\*/**

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

**if (fields.length > 3) {**

**/\***

**\* Save the first field in the line as the IP address.**

**\*/**

**String ip = fields[0];**

**/\***

**\* The fourth field contains [dd/Mmm/yyyy:hh:mm:ss].**

**\* Split the fourth field into "/" delimited fields.**

**\* The second of these contains the month.**

**\*/**

**String[] dtFields = fields[3].split("/");**

**if (dtFields.length > 1) {**

**String theMonth = dtFields[1];**

**/\* check if it's a valid month, if so, write it out \*/**

**if (months.contains(theMonth))**

**context.write(new Text(ip), new Text(theMonth));**

**}**

**}**

**}**

**}**

• PROCESSLOG

**package solution;**

**import org.apache.hadoop.fs.Path;**

**import org.apache.hadoop.io.IntWritable;**

**import org.apache.hadoop.io.Text;**

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

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

**import org.apache.hadoop.mapreduce.Job;**

**public class ProcessLogs {**

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

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

**System.out.printf("Usage: ProcessLogs <input dir> <output dir>\n");**

**System.exit(-1);**

**}**

**Job job = new Job();**

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

**job.setJobName("Process Logs");**

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

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

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

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

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

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

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

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

**/\***

**\* Set up the partitioner. Specify 12 reducers - one for each**

**\* month of the year. The partitioner class must have a**

**\* getPartition method that returns a number between 0 and 11.**

**\* This number will be used to assign the intermediate output**

**\* to one of the reducers.**

**\*/**

**job.setNumReduceTasks(12);**

**/\***

**\* Specify the partitioner class.**

**\*/**

**job.setPartitionerClass(MonthPartitioner.class);**

**boolean success = job.waitForCompletion(true);**

**System.exit(success ? 0 : 1);**

**}**

**}**

• COUNTREDUCER

**package solution;**

**import java.io.IOException;**

**import org.apache.hadoop.io.IntWritable;**

**import org.apache.hadoop.io.Text;**

**import org.apache.hadoop.mapreduce.Reducer;**

**/\* Counts the number of values associated with a key \*/**

**public class CountReducer extends Reducer<Text, Text, Text, IntWritable> {**

**@Override**

**public void reduce(Text key, Iterable<Text> values, Context context)**

**throws IOException, InterruptedException {**

**/\***

**\* Iterate over the values iterable and count the number**

**\* of values in it. Emit the key (unchanged) and an IntWritable**

**\* containing the number of values.**

**\*/**

**int count = 0;**

**/\***

**\* Use for loop to count items in the iterator.**

**\*/**

**/\* Ignore warnings that we**

**\* don't use the value -- in this case, we only need to count the**

**\* values, not use them.**

**\*/**

**for (@SuppressWarnings("unused")**

**Text value : values) {**

**/\***

**\* for each item in the list, increment the count**

**\*/**

**count++;**

**}**

**context.write(key, new IntWritable(count));**

**}**

**}**