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

Presentado por:

Albert Ripoll y Jose David Angulo

Índice

[1. Introducción 3](#_Toc506891313)

[2. Entorno de desarrollo 3](#_Toc506891314)

[3. IpCount 5](#_Toc506891315)

[4. Avarage Length 8](#_Toc506891316)

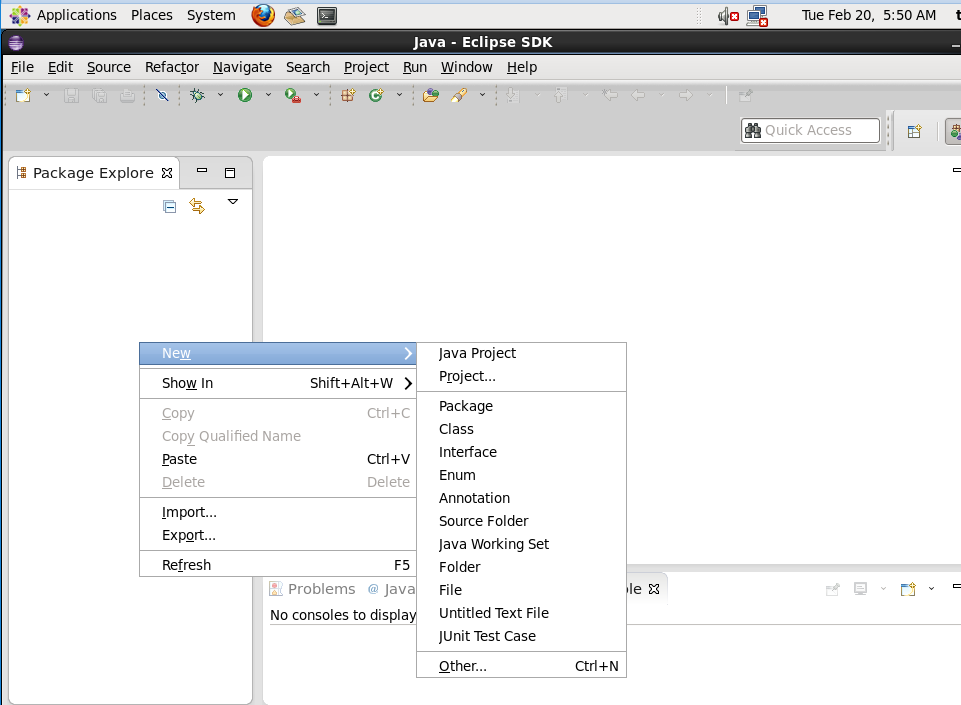
# Introducción

El objetivo de este Hands-On es realizar el desarrollo, en Java, de dos Jobs sencillos de MapReduce y ejecutarlos.

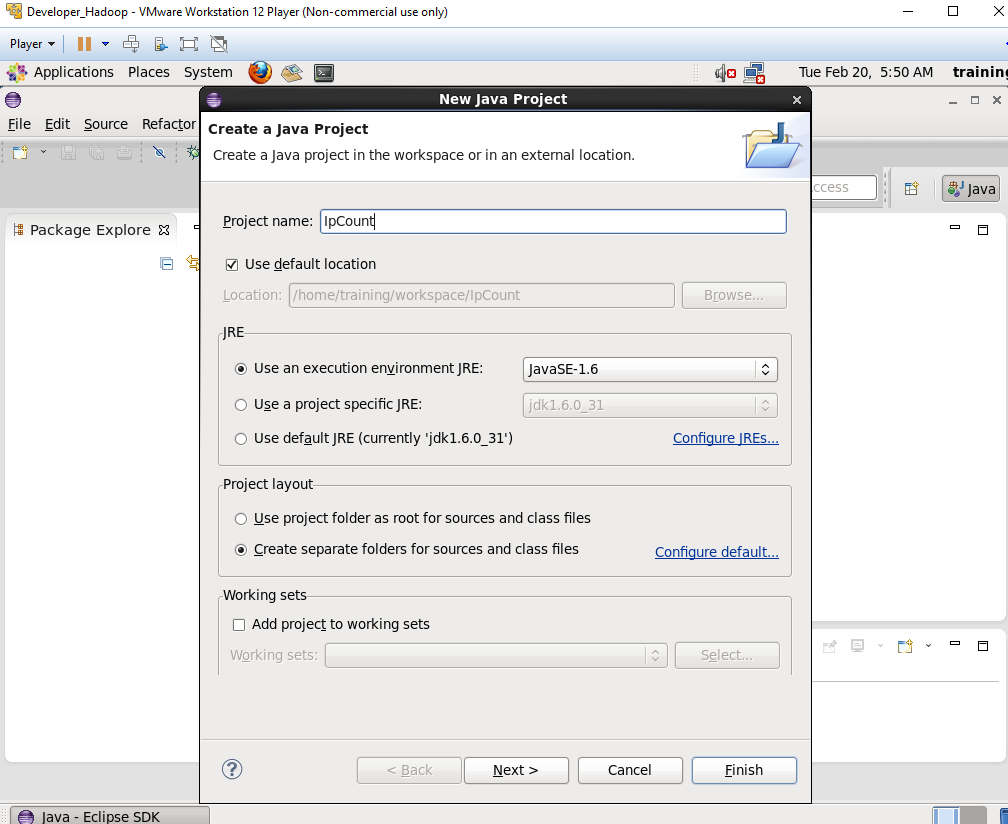
# Entorno de desarrollo

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

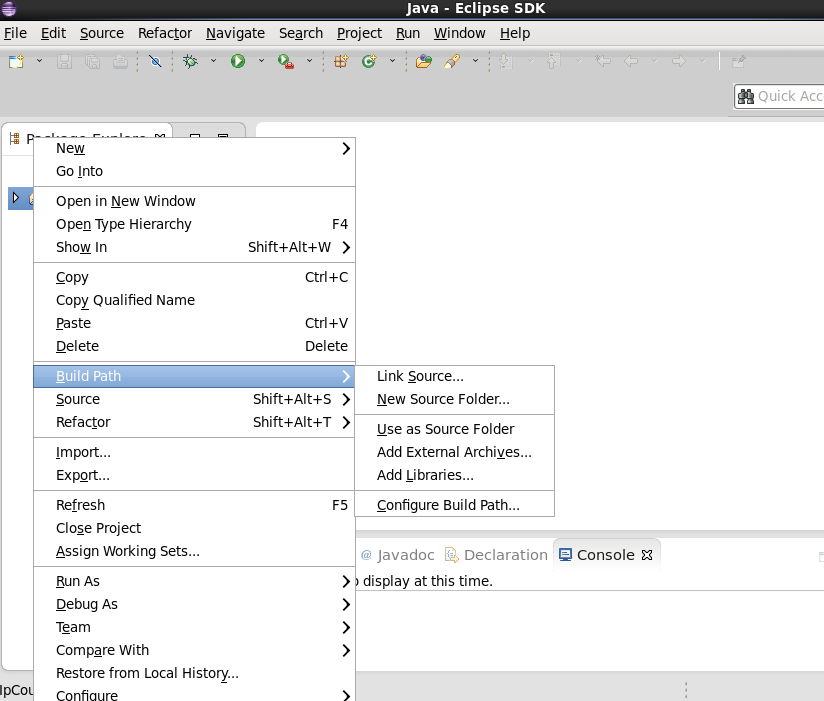
Para crear un nuevo proyecto, haremos click derecho sobre el package explorer New 🡪 Java Project



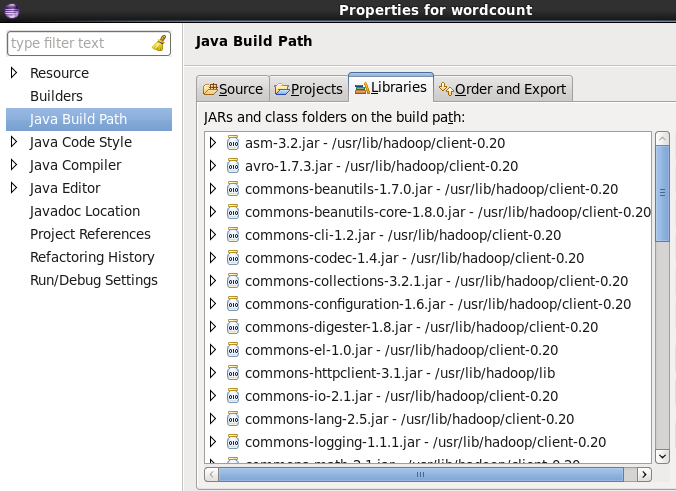
Introducimos el nombre del proyecto y click en Finish



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/



# IpCount

Desarrollar y ejecutar el siguiente MapReduce:

Contar el número de veces que una misma Ip hace una petición en los logs contenidos en /weblog/ importado anteriormente

## Código del Driver (diapositiva 23 + cambios resaltados)

/home/training/workspace/IpCount/src/lab\_4/IpMap.java

**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 WordCount { → Cambiado por IpDriver

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 (WordCount.class); → Cambiado por IpDriver

job.setJobName ("Word Count"); → Cambiado por IpDriver

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

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

job.setMapperClass (WordMapper.class) ; → Cambiado por IpMap

job.setReducerClass (SumReducer.class) ; → Cambiado por IpReducer

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);

}

}

## Código del Maper (diapositiva 35 + cambios resaltados)

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 WordMapper extends Mapper<LongWritable, Text, Text, ; → Cambiado por IpMap

IntWritable> {

@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) {; → Cambiado por String[] parts = line.split(" - - "); String word = parts[0];

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

}

}

## Código del Reducer (diapositiva 38 + cambios resaltados)

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce .Reducer;

public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{ → Cambiado por IpReducer

@Override

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

throws IOException, InterruptedException {

int wordCount = 0;

for (IntWritable value : values) {

wordCount += value.get();

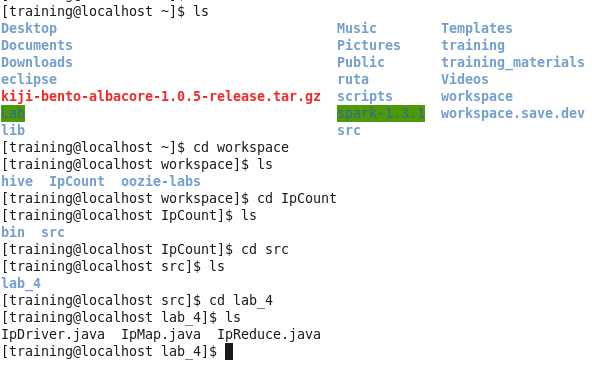
}

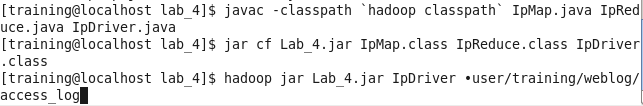
context.write (key, new IntWritable (wordCount));

}

}

## Ejecución del Map-Reduce por consola

• Después de que con Eclipse creásemos las clases .java de IpDriver, IpMap y IpReduce, accedemos en local para comprovar su ruta:

• Con el primer commando, creamos las distintas clases dentro de la carpeta “lab\_4”. Y con el Segundo commando creamos el document comprimido Lab\_4.jar 

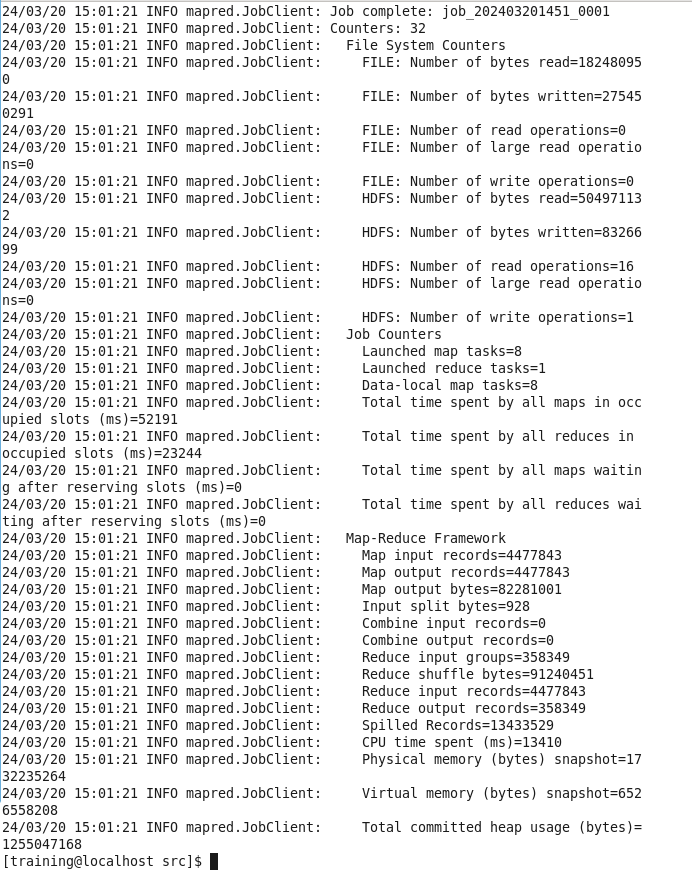
# 

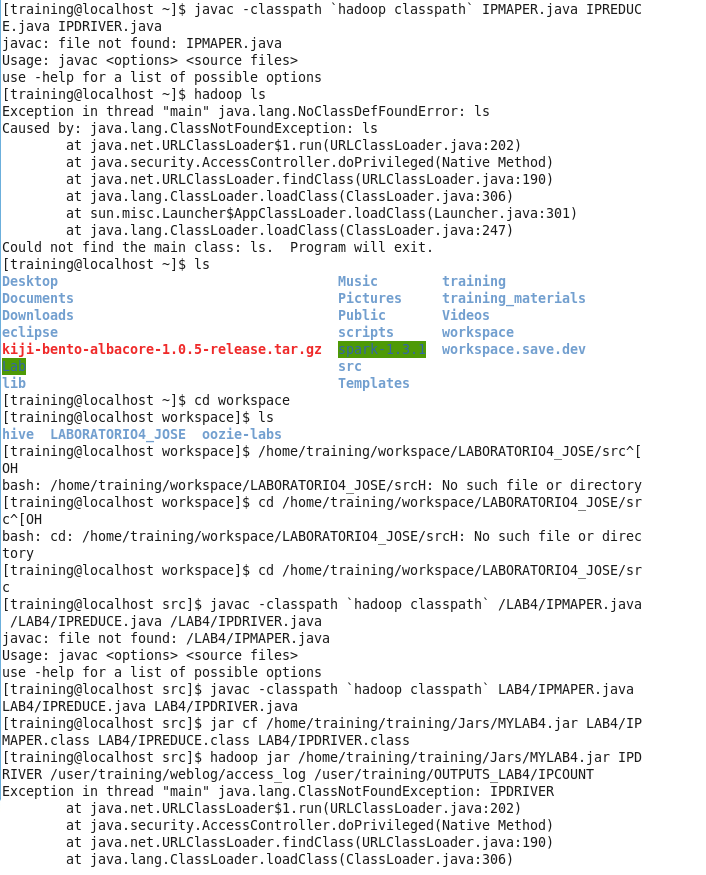
• Entonces vamos al directorio de src (de fuera del lab) para ejecutar el siguiente comando. En este caso los nombres del comando están cambiados de IpDriver a IPDRIVER, IpMap a IPMAPER, IpReduce a IPREDUCER y la carpeta Lab\_4 a LABORATORIO4\_JOSE/MYLAB4 ya que la ejecución se realizó en otra computadora con archivos con otros nombres debido a que la anterior se colgaba todo el rato.

javac -classpath `hadoop classpath` IPMAPER.java IPREDUCE.java IPDRIVER.java

jar cf MYJAR.jar IPMAPER.class IPREDUCE.class IPDRIVER.class







• Si observamos el resultado de la salida vemos que ha contado correctamente las Ip. El comando abre el archivo “part-r-00000” que está dentro de la carpeta “IPCOUNT” dentro de “OUTPUTS\_LAB4” que son las carpetas que se han generado. Al abrir, se muestra las últimas (“Tail”) “20” líneas.



# Average Length

Desarrollar y ejecutar el siguiente MapReduce:

Calcular el promedio de la longitud de las palabras que empiecen por una letra determinada en todas las obras de Shakespeare importadas anteriormente

Ejemplo de salida :

*N 2.0*

*n 3.0*

*d 10.0*

*i 2.0*

*t 3.5*

## Código del Driver

**package** LAB4;

**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;

//user/training/weblog/access\_log

**public** **class** IPDRIVER2 {

**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 (IPDRIVER2.**class**);

job.setJobName ("Ip Driver");

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

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

job.setMapperClass (IPMAPER2.**class**) ;

job.setReducerClass (IPREDUCE2.**class**) ;

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

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

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

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

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

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

}

}

## Código del Maper

**package** LAB4;

**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** IPMAPER2 **extends** Mapper<LongWritable, Text, Text,

FloatWritable> {

@Override

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

**throws** IOException, InterruptedException {

String line = value.toString();

**for** (String word: line.split("\\W+")){ //se separa la linia por palabras “word”

**if** (word.length() > 0){ //Si la palabra tiene longitude mayor que 0

String letra = word.substring(0,1); //letra = string que va del elemento 0 al 1 de la palabra

**float** longitud = word.length(); //longitud = longitude total de la palabra

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

//Devolvemos la letra como key y la longitude de la palabra como valor float. De hecho de todo el código cambiamos los valores a float.

}

}

}

}

## Código del Reducer

**package** LAB4;

**import** java.io.IOException;

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

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

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

**public** **class** IPREDUCE2 **extends** Reducer<Text, FloatWritable, Text, FloatWritable>

{

@Override

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

**throws** IOException, InterruptedException {

**float** wordCount = 0; //Contador de valores (longitudes de palabras)

**float** i=0; //Contador de letras (palabras con misma key)

**for** (FloatWritable value : values) { //Para cada valor que tenemos

wordCount += value.get(); //Sumamos acumulativamente el valor

i+=1; //Sumamos cuantas veces hemos tenido que sumar (cuantas key así había)

}

**float** promedio = wordCount / i; //Se calcula el promedio como la suma del número de letras en una palabra / cuantas palabras había con esa key

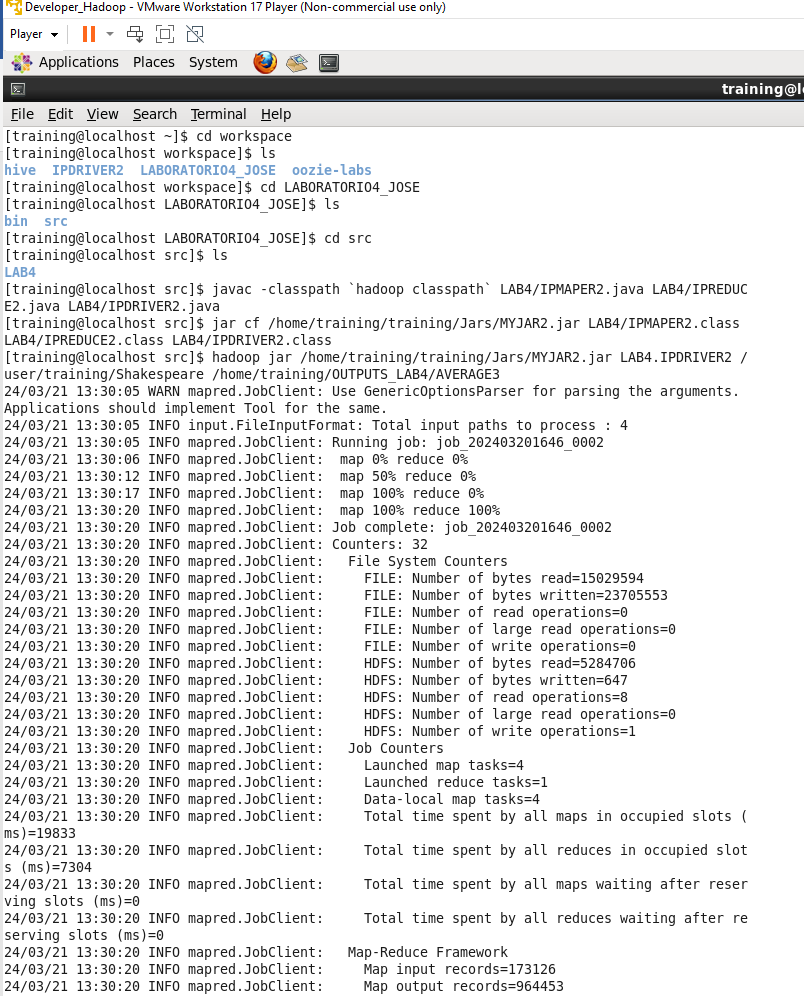
context.write (key, **new** FloatWritable (promedio)); //Se usa Float en todo el código.

}

}

## Ejecución del Map-Reduce por consola

• Ir al directorio de LAB donde están los .java



• dentro ejecutar los comandos:

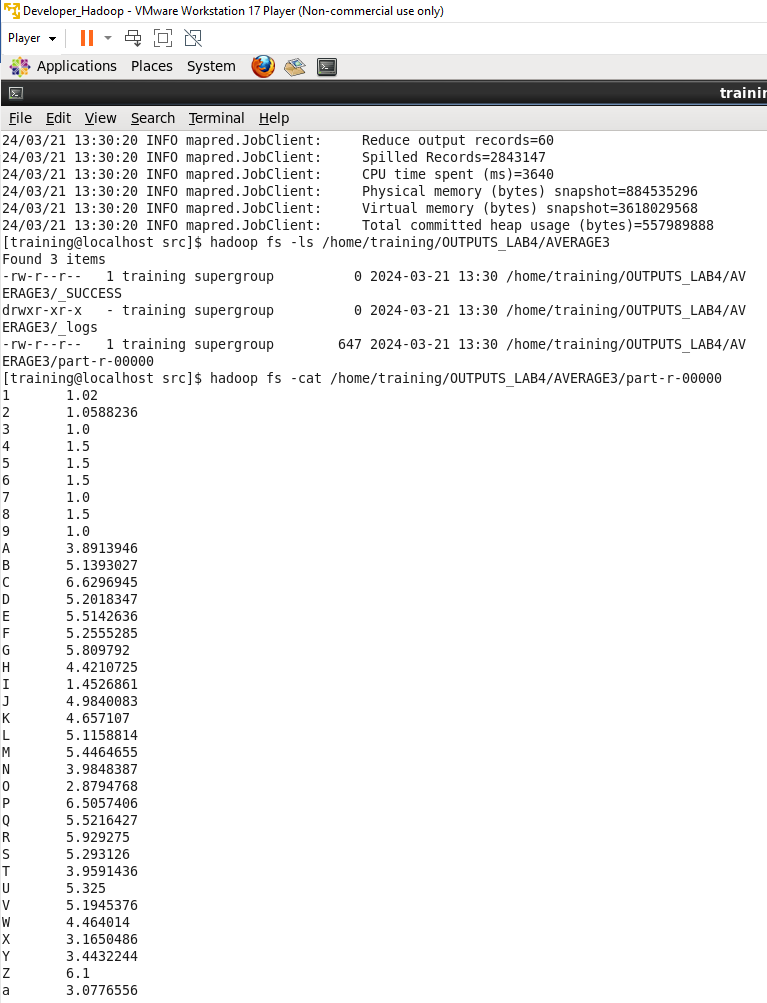
javac -classpath `hadoop classpath` LAB4/IPMAPER2.java LAB4/IPREDUCE2.java LAB4/IPDRIVER2.java

jar cf /home/training/training/Jars/MYJAR2.jar LAB4/IPMAPER2.class LAB4/IPREDUCE2.class LAB4/IPDRIVER2.class

hadoop jar /home/training/training/Jars/MYJAR2.jar LAB4.IPDRIVER2 /user/training/Shakespeare /home/training/OUTPUTS\_LAB4/AVERAGE3

• Aquí la ejecución por consola:





• Leemos el fichero donde se han guardado las palabras y su conteo:

hadoop fs -cat /home/training/OUTPUTS\_LAB4/AVERAGE3/part-r-00000

