Arquitectura i tecnologies del software

Pràctica (I) - BigData

**Enginyeria Informàtica**

**Escola d’Enginyeria (EE)**

**Universitat Autònoma de Barcelona (UAB)**

**Adonis González Godoy**

**Alex García Carballo**

**Contenido**

# 

# 1 Introducción

El objetivo principal que se propone es desarrollar una pequeña aplicación que implemente el algoritmo Map-Reduce desde 0, sin usar ninguna librería externa que implemente este proceso, esta aplicación tiene que contar las palabras de uno o varios ficheros que se le puedan pasar como parámetro.

También se tiene que programar un contador de palabras sin la implementación de Map Reduce para poder comparar los resultados de tiempo y precisión con respecto al contador de palabras con Map-Reduce.

# 2 Entorno de trabajo

El lenguaje de programación que utilizaremos para desarrollar la aplicación es **PYTHON 2.7**, hemos decidido usar este lenguaje ya que tiene una sintaxis muy legible e intuitiva que nos permite manejar listas y diccionarios de una manera más fácil, de tal manera que nos podemos centrar en el algoritmo y no tanto en la sintaxis.

# 3 WordCount

Es una pequeña y sencilla aplicación que cuenta el número de palabras dado uno o varios ficheros, adjuntamos el fichero que implementa esta funcionalidad.

# 4 Map-Reduce

Es un algoritmo que propone resolver problemas cuando se tiene un volumen de datos grandes, con una técnica de manipulación de datos que consiste en dividir las tareas en diferentes algoritmos procesándolos de forma paralela, está basado en dos fases el **map** y el **reduce**.

A continuación, mostramos un diagrama donde podemos observar la implementación grafica del Wordcount implementando el Map-Reduce.

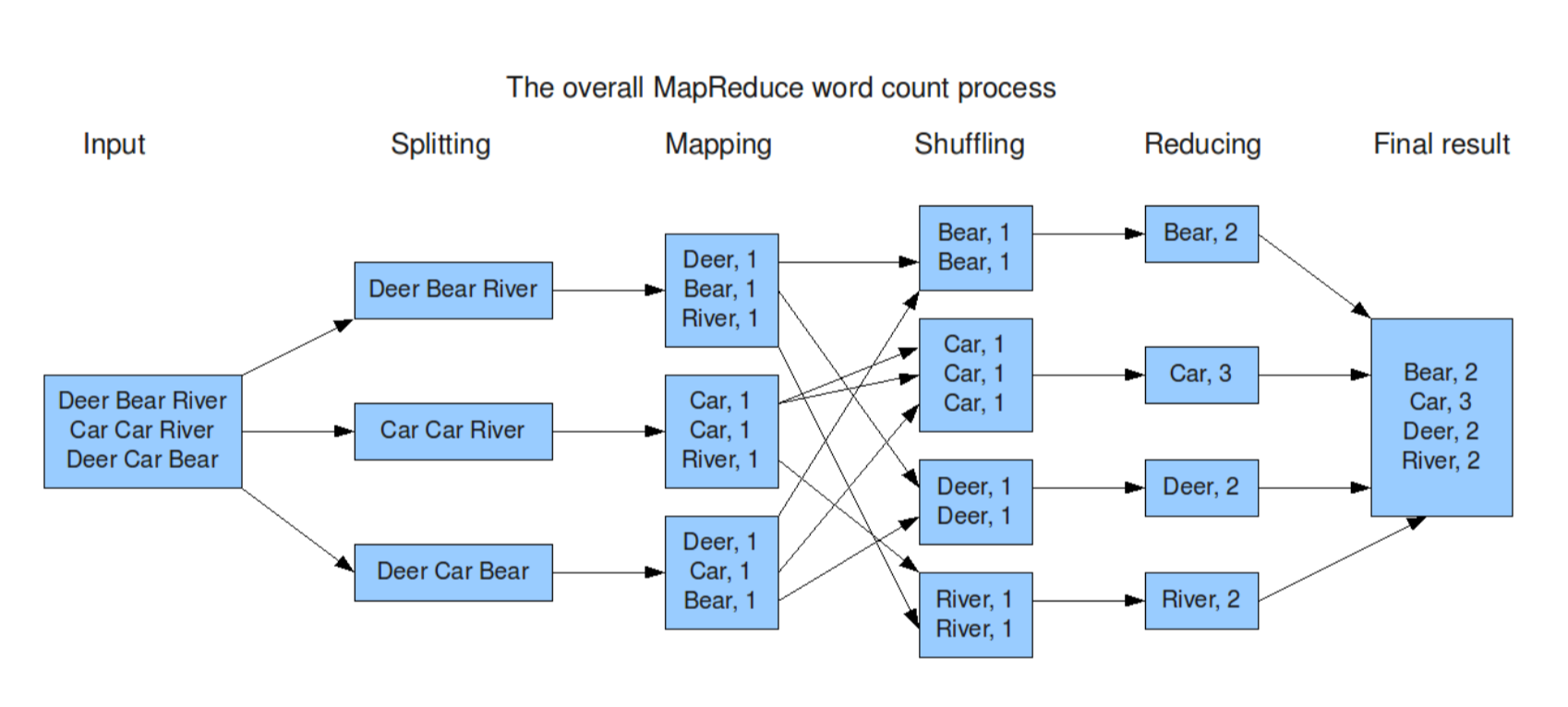
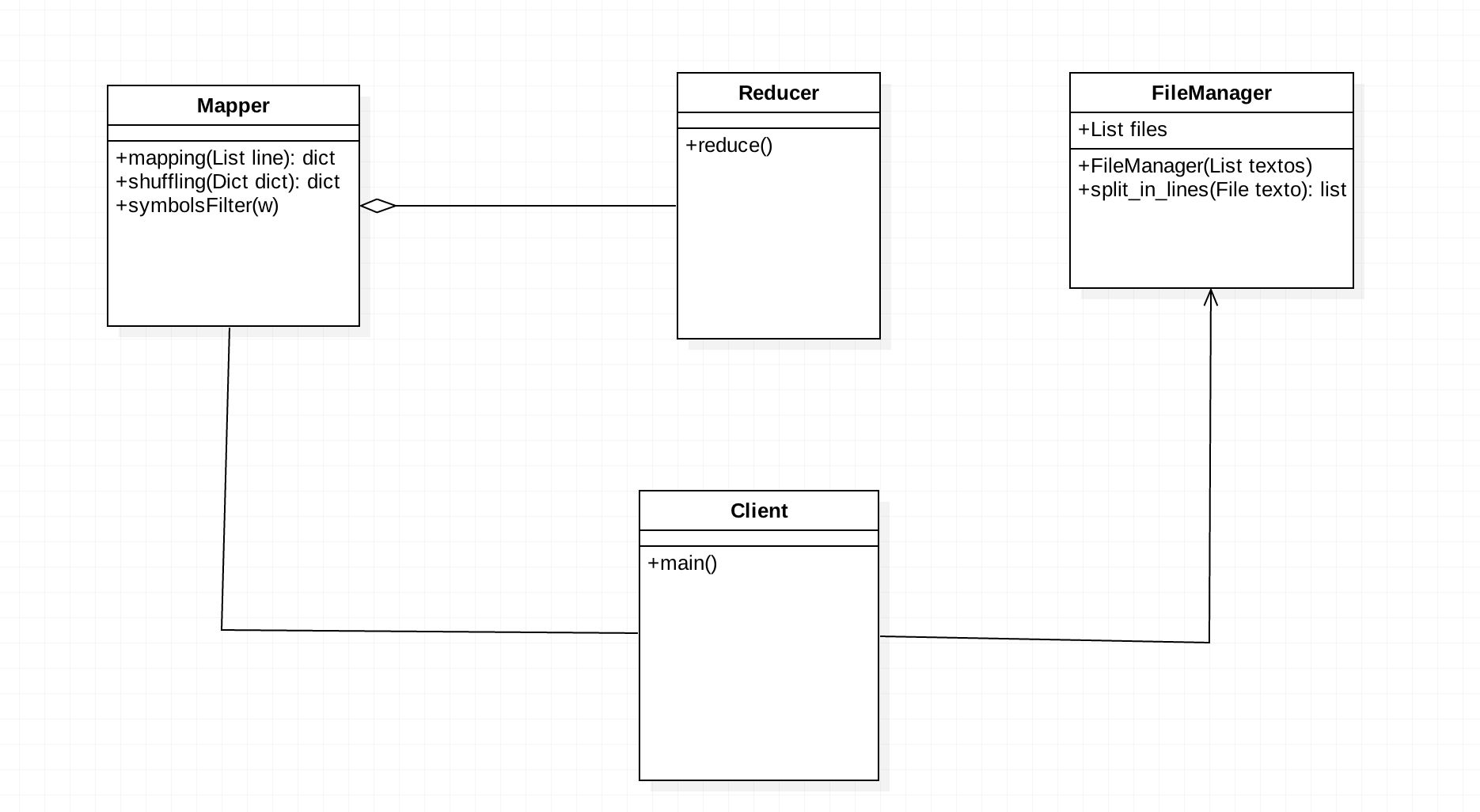
****

Diagrama 1 - Map-Reduce

# 5 Implementación

Para definir la estructura de la aplicación tenemos el siguiente diagrama UML.

## UML



EL UML falta añadir cosas que no lo tenia tan claro.

# 6 Diagrama de secuencias

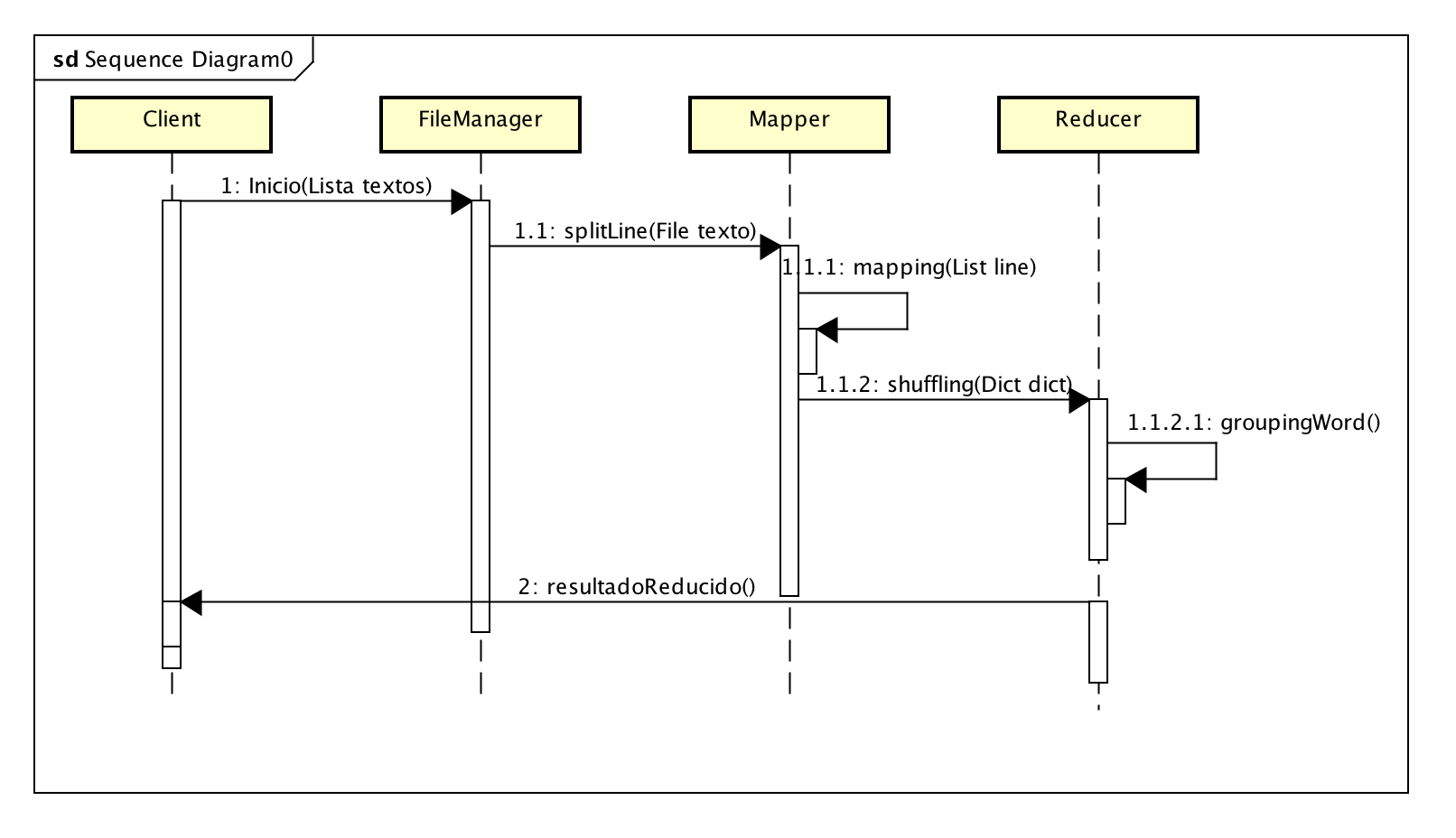
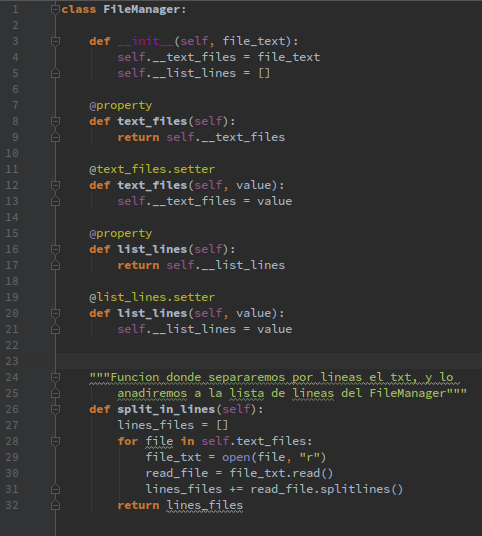


Diagrama 2 - Secuencias

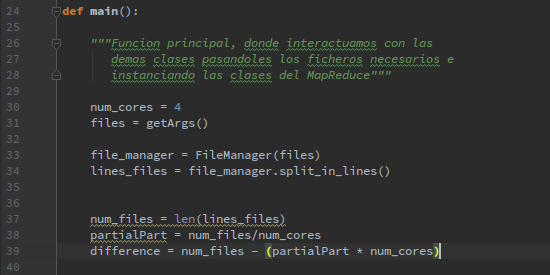
# 7 Funcionamiento del programa

La clase **FileManager** gestiona los ficheros que se pasarán por parámetro, también se encarga de dividir los ficheros en líneas, esta clase equivale a la fase de separar o splitting. Todas estas líneas de uno o varios ficheros se guardarán en una lista.



En esta clase, nos encargamos de que dada una lista de ficheros que debemos contar, nos devolvera una lista con todas las líneas de todos los ficheros:

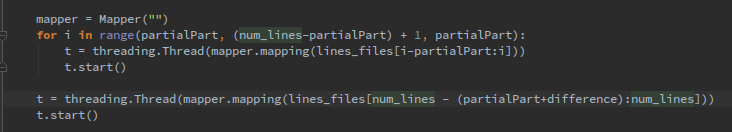
[“linea1”, “linea2” , “linea3”...]



En el main, recogemos esa lista de líneas que nos devuelve el FileManager, y calculamos diferentes valores como el número de líneas, el partial part, que es el número de partes en las que dividiremos todo el contenido, para intentar que cada thread tenga una carga equivalente y adecuada al número de cores del ordenador, que en este caso hemos puesto 4 cores de forma predeterminada. También calculamos la diferencia, que la necesitamos en los casos donde la división no es exacta, para poder recoger todas las líneas. Ejemplo:

partial part = 46/4 = 11 (en entero)

En total tendriamos 4 partes de 11 líneas, 11 \* 4 = 44 , nos faltarian dos líneas por contar, así que le sumamos la diferencia al último thread, para que recoja todas las líneas.



Dividimos el contenido en los bloques comentados anteriormente(por ejemplo si son 46 líneas sería, thread1 11 líneas,thread2 11 líneas, thread3 11, thread4 13) y ejecutamos los threads.

En cada uno de los threads se ejecuta la clase **Mapper**.

La clase **Mapper** contiene tres funciones:

symbolsFilter(w) 🡪 se encarga de quitar los símbolos a las palabras

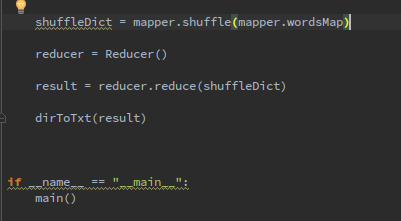
mapping() 🡪 creamos una lista donde tendremos tuplas con dos valores, una palabra y un valor clave para cada una de estas, en concreto el valor 1. Esta función acaba retornando una lista con tuplas que pasaremos a la función shuffle().

[(palabra1, 1), (palabra2, 1), (palabra1, 1), (palabra3, 1)]

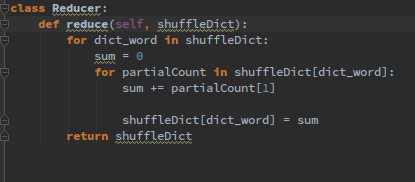
shuffle() 🡪 La función shuffle agrupa las mismas palabras en un diccionario, de forma que a la hora de hacer el reducer será sencillo contarlas. Un ejemplo de salida de esta función seria el siguiente diccionario:

{ ‘palabra1’ : [(‘palabra1’ , 1), (‘palabra1’ , 1)] , ‘palabra2’ : [‘palabra2’ , 1] }

Por último debemos pasar el diccionario agrupado al reducer, para contar todas las veces que se repite una palabra dentro del diccionario.



La clase **Reducer** contiene el método reduce, que se encarga de ir por cada una de las agrupaciones del diccionario que generó el shuffle y contarlas, guardándolas en un diccionario con la palabra como clave, y el número de veces que sale como valor.



Un ejemplo de salida que tendría esta función seria:

{‘palabra1’ : 1 , ‘palabra2’ : 3 , ‘palabra3’ : 2 }

# Resultados

Graficas con resultados de distinas ejecuciones.

# Compilación

Para ejecutar el programa debemos tener instalado la versión de Python 2.7.

Usaremos el comando siguiente:

python TextCounter.py File1.txt

Se puede pasar uno o varios ficheros por parámetros.

python TextCounter.py File1.txt File2.txt

Si el fichero de texto no esta en la misma carpeta que el script de TextCounter.py, se debe incluir la ruta como en el ejemplo:

