UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Sistemes Concurrents i Paral·lels

Práctica 2

Joaquim Picó Mora, Ian Palacín Aliana
PraLab1

Professorat : F. Cores
Data : 3 de Desembre 2019

Índex

1 Introducción

En este documento se compara la eficiencia en tiempo que supone ejecutar Indexing y Query de forma concurrente respecto de forma secuencial. También se muestra la diferencia en tiempo que supone la ejecución del programa con un número distinto de hilo.

Los datos que se han utilizado para realizar el estudio salen de la ejecución de forma secuencial y concurrente de múltiples ejemplos, calculando así el tiempo que tarda en realizarse cada una de las tareas. Este tiempo será el que se usará para sacar conclusiones.
Del mismo modo, se ejecutará varias veces el programa con distinto número de hilos, todos estos datos quedan recogidos en las gráficas.

2 Concurrente vs Secuencial

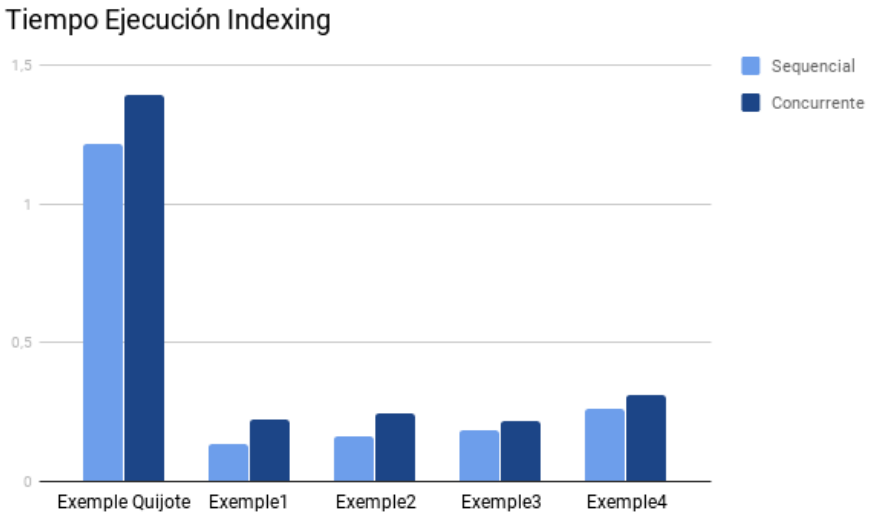

Figura 1: Indexing

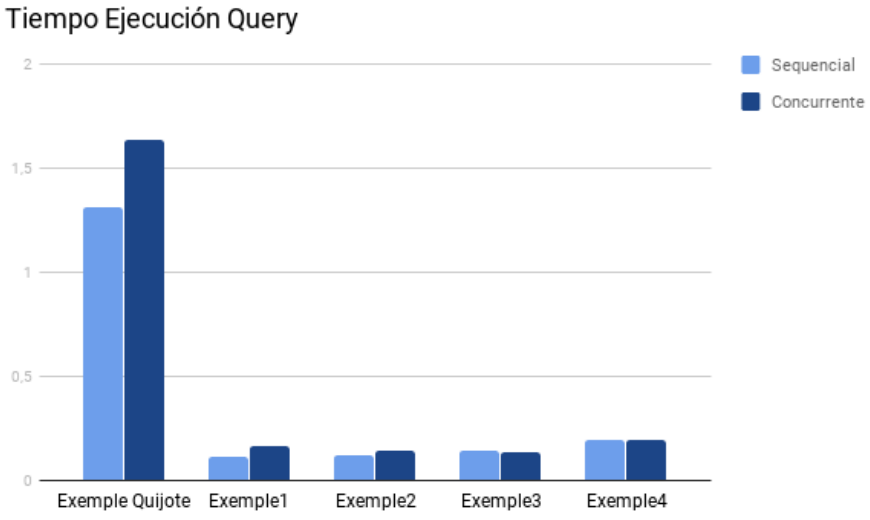

Figura 2: Query

El objetivo de la práctica ha sido implementar la versión concurrente de Query y Indexing, así como estudiar su comportamiento y tiempos de ejecución.

En las gráficas se pueden discernir los tiempos de ejecución de cada uno de los ejemplos en su versión secuencial y concurrente, sin embargo, cabe destacar algunas peculiaridades sobre los resultados obtenidos.

El resultado más chocante es que en algunos casos la ejecución concurrente da un tiempo superior a la implementación secuencial, esto se debe lo siguiente: La implementación tanto en Query como en Indexing se basa en una estrategia Master/Worker que consiste en dividir la construcción del HashMap de Inverted Index y repartirlo en diferentes hilos, haciendo que cada hilo construya su propio HashMap parcial. Posteriormente se unen los HashMaps de cada uno de los hilos para generar el acumulado. Esta última operación (putAll) genera un cuello de botella que se suma al de I/O haciéndola bastante costosa en tiempo, motivo por el que se explica que a veces la aplicación concurrente sea menos eficiente que la secuencial. Estamos convencidos que hay implementaciones que pueden rodear o prescindir de putAll, en ese caso la implementación concurrente no solo no sería más lenta que la secuencial, sinó que sería más rápida. Una posible solucion sería sustituir HashMultiMap por algun HashMap thread safe e ir acumulando las claves sobre el mismo objeto.

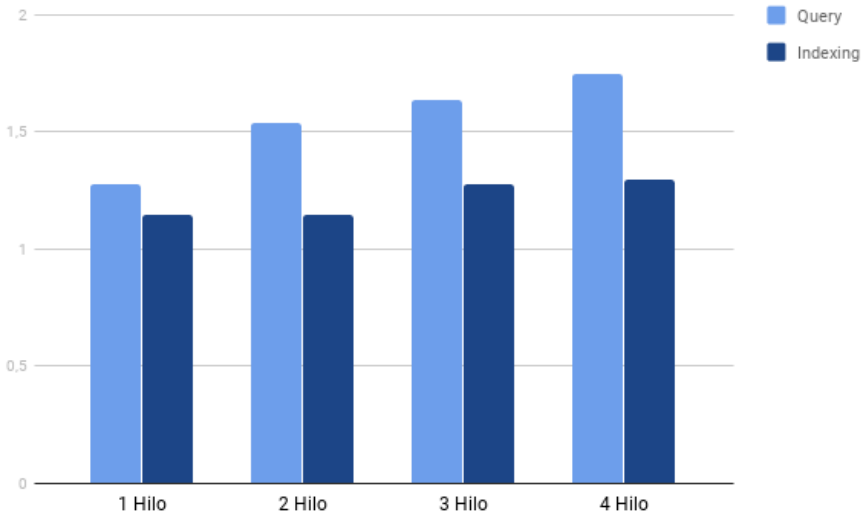

Figura 3: Ejecucion con múltiples hilos

3 Diferencias entre número de hilos

Como se puede ver en la Figura 3, en el caso tanto de Query como de Indexing conforme mas hilos ejecutándose de forma paralela, más coste en el tiempo. Esto es debido a lo ya mencionado en el apartado anterior, a más hilos más veces se tiene que realizar la operación putAll, alargando más el cuello de botella. Se le tiene que añadir, también, el coste de creación y mantenimiento de más hilos.

4 Diseño de la Solución

La solución presentada en esta práctica consiste en lo siguiente.

4.1 Indexing

Se recibe por parámetro el número de hilos con los que se desea ejecutar la aplicación. Se realiza un balanceo de carga asignando a cada hilo un número determinado de caracteres a procesar del fichero. Una vez asignado a cada hilo la carga de trabajo a realizar, estos empiezan a procesar carácter a carácter las claves y a construir sus HashMap parciales. Una vez realizado el join de todos los hilos, se juntan los HashMaps parciales en uno y este se guarda o se imprime según los argumentos especificados.

4.2 Query

Utiliza la misma estructura que Indexing, pero esta vez el balanceo de carga se realiza respecto a los ficheros que deben leer cada uno de los hilos. Cada thread leerá un número equitativo de ficheros y creará su propio hashmap así continuando con la misma arquitectura que la concurrencia en Indexing.

5 Apuntes finales

La parte donde se guardan las claves de forma concurrente está comentada debido a unos problemas, no obstante nos gustaría acentuar algunas curiosidades.

De forma muy parecida a putAll, la mayoría de tiempo de esta función la acarriaba una llamada a un método, en este caso era remove() de ArrayUtils. Este método lo utilizábamos para borrar las llaves leídas y preparar el set de llaves para el siguiente thread. Como curiosidad, en textos muy extensos este método ocupaba hasta un 97% del tiempo de ejecución del guardado de los índices.