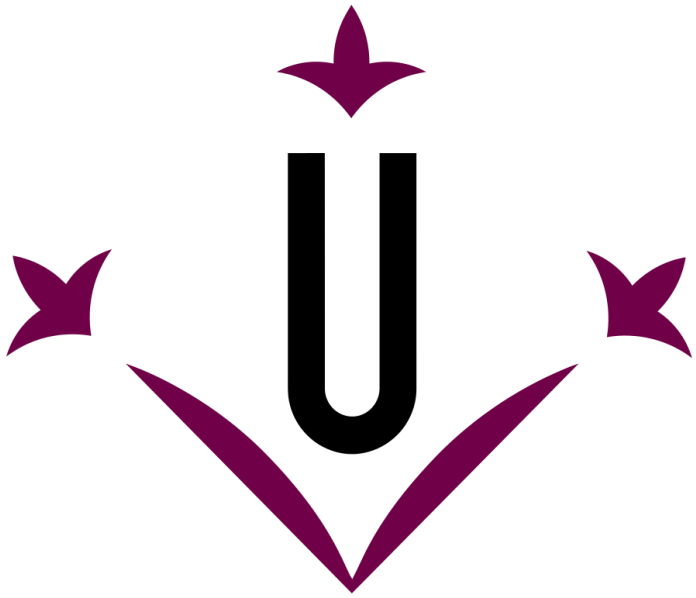
PRÀCTICA 1

SISTEMES CONCURRENTS I PARAL·LELS



Universitat de Lleida

Alumnes: Jordi Arenas Romero i Sergi Baron Pascual

Professor: Fernando Cores Prado

Grau en Enginyeria Informàtica

Curs 2021-2022

ÍNDEX

I. INTRODUCCIÓ

En l'àmbit de la informàtica, la concurrència i la paral·lelització són conceptes fonamentals per millorar el rendiment i l'eficiència de les aplicacions. En aquest treball, abordarem la implementació d'una aplicació concurrent multi-fil en el llenguatge de programació Java. L'objectiu principal d'aquesta pràctica és consolidar els coneixements adquirits a l'assignatura de Sistemes Concurrents i Paral·lels en un projecte concret.

La pràctica es centra en el disseny i la creació d'una aplicació que fa servir la indexació i la cerca d'informació mitjançant un Índex Invertit. Un Índex Invertit és una estructura de dades que permet mapejar paraules o números a les seves ubicacions dins d'un conjunt de documents o en una base de dades. Això possibilita la realització de cerques de text de manera eficient, encara que el cost de processament augmenta cada vegada que s'afegeix un nou document a la base de dades.

El propòsit d'aquesta pràctica és desenvolupar una versió concurrent d'una aplicació que realitzi consultes a un Índex Invertit. Per aconseguir-ho, s'utilitzaran dos programes principals, "Indexing" i "Query", encarregats de construir l'índex invertit i realitzar consultes, respectivament.

Al final de la pràctica, s'espera que la versió concurrent ofereixi un rendiment superior a la versió seqüencial en arxius suficientment grans, demostrant així la importància de la concurrència en aplicacions informàtiques. Al llarg d'aquest treball, es presentaran els detalls tècnics de la implementació, es discutiran els resultats i s'avaluarà el grau d'eficiència i concurrència aconseguit en ambdues versions de l'aplicació.

II. IMPLEMENTACIÓ

I. Indexing

La classe *Indexing* és la encarregada de crear l'índex invertit. Per a fer-ho, utilitzarà la classe *InvertedIndex*, tal i com passa en la versió seqüencial.

Inicialment, es crida a la funció *buildIndex*, que busca els fitxers dins del directori indicat i els guarda en un *ConcurrentLinkedDeque*. Menters tant, un fil virtual creat pel fil principal s'encarrega d'assignar un fil per a cada fitxer que entra a la cua, que el processarà i crearà el índex parcial del fitxer. D'aquesta forma, el fil principal només ha de crear aquest únic fil i els fitxers es processen mentre es van buscant, per tant, es perd menys temps que si hagués de buscar primer tots els fitxers i crear després un fil per cada.

Finalment, el fil que ha anat creant els fils per cada fitxer integra els resultats parcials i quan acaba, el fil principal recupera el resultat final.

Per tal de salvar el índex creat es crida a la funció *saveInvertedIndex*, es crea un fil per cada fitxer, el de ids de fitxers, el del contingut dels fitxers i els fitxers que contenen l'índex. Els dos primers fils guarden directament el contingut de les estructures de dades en els fitxers. El tercer fil s'encarregarà de calcular el nombre de fitxers que composaran l'índex i crearà un fil per cada un, passant-li el subSet de keys que s'ha d'encarregar de guardar.

II. Query

Inicialment, es crida a la funció *loadIndex*, que crea un fil virtual per cada fitxer que s'encarregaran de crear els fitxers *IndexFile*, i dos fils més que s'encarregaran de crear els fitxers *FilesIds* i *FilesLinesContent*.

Finalment, es crida a la funció query, que funciona de la mateixa forma que en la versió inicial. Inicialment, vam realitzar el problema de forma concurrent utilitzant 1 fil per cada N paraules de la query. Tanmateix, tal i com s'explica en el punt 3 d'aquest document, això no resultava òptim i es va decidir acatar la implementació seqüencial per a aquesta part del problema.

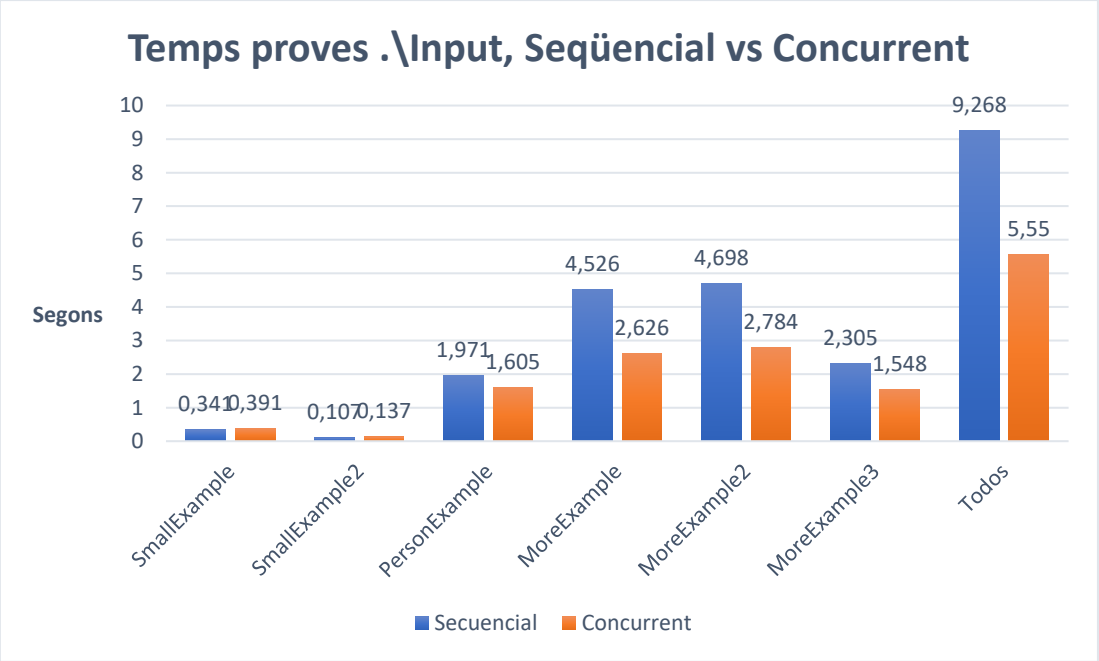
III. EFICIÈNCIA

I. Processador

Les proves d'eficiència realitzades en els següents punts han sigut realitzades amb un processador i7-11700K, que consta de 8 nuclis i 16 subprocessos. Aquest nombre de nuclis/subprocessos permet aprofitar al màxim el potencial de l'aplicació concurrent en els diferents directors de prova utilitzats, excepte en el cas en el que s'utilitzen com a input tots els exemples, ja que en aquest cas hi ha més arxius dels que pot aprofitar el processador.

II. Indexing

Per tal d'explorar l'eficiència en temps dels dos programes, seqüencial i concurrent, hem comparat el temps que tarden els programes en indexar, guardar i carregar novament el índex. Els resultats es poden veure en la *gràfica* 1.



Gràfic 1: Temps d'execució de l'indexació dels fitxers en .\Input

Interpretem els resultat: quan hi ha molt pocs fitxers (1 o 2) i, a més, aquests tenen poques línies, el seqüencial pot arribar a executar-se més ràpid, ja que crear els fils també suposa un cost de temps, però la diferència és pràcticament inexistent. Això ho podem veure en SmallExample i SmallExample2, carpetes amb 3 i 1 fitxer respectivament d'unes 10 línies com a màxim.

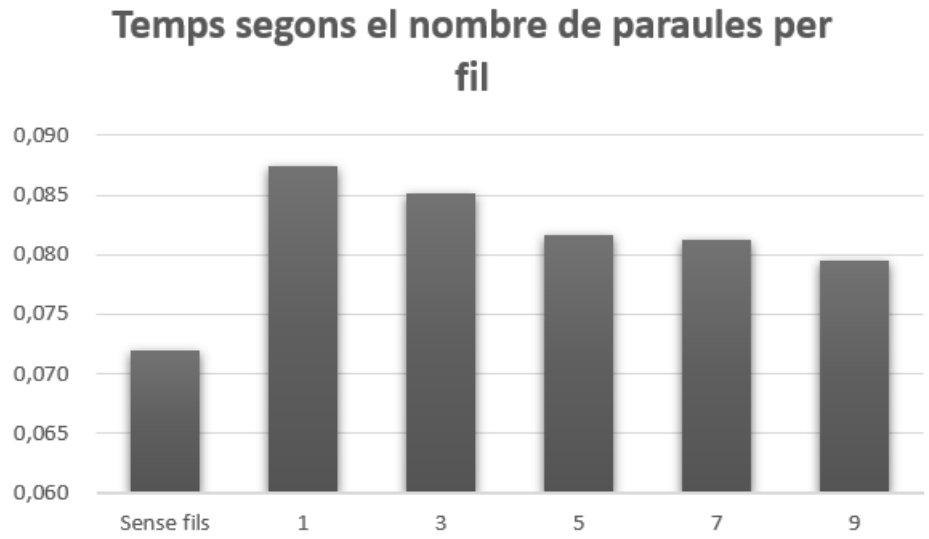
En canvi, quan hi ha molts fitxers i, a sobre, aquests són bastant extensos, és notòria la millora en la eficiència del programa concurrent, ja que obrir tants fitxers o processar tantes línies té un cost elevat que resulta més convenient repartir en varis fils. Observem això en tots els altres casos, on el nombre de fitxers és al voltant de 7 i els fitxers arriben a tenir milers de línies, fins a unes 60.000 i una mitja com de 30.000.

En un cas afegit per nosaltres, PersonExample, hem posat només dos fitxers però ambdós amb unes 300 línies, fragments de pag2600.txt i quijote.txt presents en un altre exemple. En aquest cas es veu que tot i ser pocs fitxers, a poc que hi hagi unes poques línies més de una desena, ja es veu la diferència d'eficiència.

III. Query

En quant a les consultes, inicialment vam plantejar un processament multi-fil segons el nombre de paraules. Aquest, creava 1 fil cada N paraules de la consulta de forma equitativa i, finalment, ajuntava els resultats. Tanmateix, fent una comparació del rendiment per descobrir el valor òptim de N, hem observar que no valia la pena assumir el cost de creació d'un fil virtual.

Com es pot veure en el *gràfic 2*, si N és 1 (es crea un fil virtual per a cada paraula de la query), el rendiment és el pitjor, mentre que a major sigui la N millor és el rendiment, ja que es creen menys fils. Tanmateix, si utilitzem el fil principal per a realitzar aquesta tasca en comptes de crear un fil addicional, el rendiment del programa millora considerablement.



Gràfic 2. Temps d'execució de Query segons el nombre de paraules per fil

IV. CONCLUSIONS

Una vegada hem acabat l'aplicació concurrent i hem avaluat els resultats, podem concloure que s'ha aconseguit millorar l'eficiència de l'aplicació significativament, arribant a quasi la mitat del temps d'execució respecte a la versió seqüencial en els arxius més pesats.

També hem pogut observar com millora el rendiment proporcionalment a la mida dels arxius processats, sent pitjor en els exemples més petits (ja que el temps dedicat a crear els fils no compensa els beneficis de la paralelització del treball) i millorant considerablement el temps de processament dels directoris més pesats (Com mostrem en PersonExample, es comença a veure la millora relativament ràpid).

Per altra banda, creiem important remarcar el cost addicional per als programadors en fer l'aplicació concurrent, ja que cal tindre en compte la sincronització dels fils i l'accés a les dades compartides. També és important considerar quines parts del codi cal paralelitzar i quines no. Tanmateix, el cost addicional és compensat pel notable increment de rendiment aconseguit, sempre que els fils realitzin una quantitat significativa del treball.

En resum, aquest treball ens ha proporcionat una experiència valuosa per aplicar la concurrència en un context pràctic. Hem aconseguit millorar el rendiment de l'aplicació i abordar els reptes associats i abordar els reptes associats a la concurrència.