Anàlisi del programa

Octubre 2021

Índex

1 Introducció

L'objectiu d'aquesta pràctica se centra en analitzar un programa que s'executa de forma seqüencial. Es tracta d'utilitzar eines que permetin veure quines parts del programa són les que més triguen en executar-se. Aquesta informació serà utilitzada en aquesta pràctica per tal d'optimitzar algunes funcions per fer l'aplicació més eficient, mantenint l'execució seqüencial de l'aplicació. A les següents dues pràctiques es re-escriurà part del codi per tal de fer l'aplicació concurrent.

A la secció 2 es descriu el programa a analitzar i a la secció 3 es proposen experiments per analitzar l'eficiència de les funcions que s'executen. L'aplicació correspon al codi de la pràctica 4 de Sistemes Operatius 1 del curs 2020-2021.

2 Programa a analitzar: recomanador de pel·lícules

Els sistemes de recomanació són sistemes computacionals que permeten, d'una forma general, realitzar recomanacions de contingut a diferents usuaris.

Els sistemes de recomanació, normalment, estan dividits en sistemes de col·laboració i en sistemes basats en continguts. Els sistemes de col·laboració normalment utilitzen similituds entre usuaris i ítems simultàniament per realitzar les recomanacions. D'altra banda, els sistemes basats

en continguts són sistemes que basen les seves recomanacions en les accions prèvies realitzades per l'usuari.

Per a aquest i següents laboratoris s'utilitzarà un sistema de col·laboració en què usuaris han valorat aquelles pel·lícules que han vist prèviament. La llista de pel·lícules, usuaris i puntuacions serà un subconjunt del conjunt de dades de Kaggle "*Netflix Prize data*", que es pot trobar a `https://www.kaggle.com/netflix-inc/netflix-prize-data`. Aquest conjunt de dades conté uns fitxers CSV que contenen, per cada identificador únic de pel·lícula (MovieID, amb rang de l'1 al 17770), totes les valoracions fetes per a aquesta pel·lícula, on cada valoració es compon de l'identificador de l'usuari que ha valorat la pel·lícula (CustomerID, amb rang de l'1 al 2649429), la puntuació (enter entre l'1 i el 5, ambdós inclosos) i la data de valoració. En aquesta pràctica s'inclou només un subconjunt de les dades que vindrà donat per el arxiu `reduced_data.txt` inclòs als arxius de la pràctica.

La informació de les valoracions s'estructurarà, per motius de facilitat a l'hora de programar, en una matriu que estructuri les valoracions proporcionades pel fitxer CSV:

$$R = \begin{pmatrix} r(1,1) & r(1,2) & \cdots & r(1,n) \\ r(2,1) & r(2,1) & \cdots & r(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ r(m,1) & r(m,2) & \cdots & r(m,n) \end{pmatrix}$$

on n és el número de pel·lícules analitzades, m el nombre d'usuaris analitzats i $r(i,j)$ és la valoració, de l'1 al 5, de l'usuari de la fila i a la pel·lícula de la columna j. Un valor de $r(i,j) = 0$ indica que l'usuari no ha visualitzat la pel·lícula.

Imaginem ara que volem predir quina puntuació donaria l'usuari de la fila i a la pel·lícula de la columna j que encara no ha vist. És a dir, $r(i,j) = 0$. Quina és la valoració $p(i,j)$ que assignaria l'usuari a la pel·lícula? Una cosa que podem fer per tractar de aproximar-la és fer una mitjana ponderada del les valoracions de tots els usuaris que han vist la pel·lícula, és a dir, de totes les valoracions de la columna de la pel·lícula. En fer la ponderació, buscarem una mètrica adequada que avaluï la importància d'una valoració a partir de la similitud entre l'usuari que l'ha fet i l'usuari al qual li volem recomanar. En el nostre cas, tindrem en compte quins usuaris han vist les mateixes pel·lícules que l'usuari de la fila i i les valoracions que han fet. Amb això obtenim la fórmula:

$$p(i,j) = \frac{1}{\sum_{t=1,t\neq i}^{m} \text{sim}(i,t)} \sum_{t=1,t\neq i}^{m} \text{sim}(i,t) \cdot r(t,j) \tag{1}$$

on $1 \geq \text{sim}(i,t) \geq 0$ és la funció de ponderació que mesura la similitud entre l'usuari de la fila i i l'usuari de la fila t.

La funció de similitud sim pot ser definida de moltes maneres, nosaltres utilitzarem la fórmula:

$$\text{sim}(i,t) = \left(1 + \sqrt{\sum_{z=1,\text{ s.t. } r(i,z)\neq 0\neq r(t,z)}^{n} (r(i,z) - r(t,z))^2}\right)^{-1}. \tag{2}$$

Observar que a l'equació anterior es calcula una distància de les valoracions de les pel·lícules que han vist en comú l'usuari de la fila i i l'usuari de la fila t. Si els usuaris han donat valoracions similars a les pel·lícules que han vist en comú tindran associada una ponderació alta, i.e. $\text{sim}(i,t) \approx 1$. En canvi, si han donat valoracions diferents tindran associada una ponderació baixa, i.e. $\text{sim}(i,t) \approx 0$.

Amb la fórmula (1) podem doncs predir quina serà la puntuació que un usuari donarà a una determinada pel·lícula. Quina pel·lícula seria la que un sistema automàtic recomanaria de veure a l'usuari de fila i ? Aquella que tingui el valor de $p(i, j)$ més alt! La recomanació de pel·lícula que farem a l'usuari de fila i serà doncs:

$$\text{rec}(i) = \text{argmax}_{j \in j: r(i,j)=0}(p(i, j)) \tag{3}$$

El programa s'ha dividit en dues parts de codi ben diferenciades. A continuació es detalla cadascuna de les parts. Teniu disponible el codi complet de cada part.

2.1 Codi part 1: Generació de la matriu R

El primer que s'ha de fer per generar la matriu R és llegir el fitxer CSV, que es troba a ./data/reduced_data.txt., i estructurar les dades en un format que es pugui manipular més fàcilment. Aquesta tasca la realitza el mètode *getMoviesFromCSV* que, donada la localització del fitxer CSV, genera una estructura de tipus *FileMovies*, que conté una llista de les pel·lícules del CSV i el número d'elements de la llista. Cada pel·lícula està modelada mitjançant una estructura *Movie*, que conté l'ID de la pel·lícula, el nombre de valoracions rebudes a aquesta pel·lícula i una llista d'aquestes valoracions. Per últim, aquestes valoracions estan definides com a estructures *Rating*, que contenen els identificadors dels usuaris que han valorat les pel·lícules i les valoracions.

Un cop s'ha obtingut aquesta llista cal construir un llistat d'usuaris i de pel·lícules úniques per construir la matriu de recomanacions. En aquesta matriu cada usuari tindrà assignada una única fila mentre que cada pel·lícula tindrà assignada una única columna. La fila (resp. columna) assignada a un usuari (resp. pel·lícula) es mapen de forma que no necessàriament hagi de ser igual al ID que li correspon al fitxer CSV. El mapatge, però, és únic. És a dir, a cada usuari (resp. pel·lícula) se li assignarà el valor ID corresponent del fitxer CSV així com un número únic a la matriu.

Per aconseguir aquest mapat únic d'usuaris i pel·lícules s'utilitza *getUniqueUsers* i *getUniqueMovies* que retornen el nombre d'elements únics i un vector d'aquests elements, tant com per a les pel·lícules com per als usuaris. Un cop es tenen aquestes llistes, el següent pas és assignar un valor de columna j per a cada pel·lícula i un valor de fila i per a cada usuari. Per fer aquesta tasca s'utilitzen les funcions *createHashMoviesAndUsers*, *addUsersToTable* i *addMoviesToTable*. Aquestes funcions fan ús de la taula de hash global nativa de C per a Linux; veure *hsearch* (feu "man hsearch" per a més informació respecte aquesta funció)

Un cop s'ha fet açò, es crea una matriu a memòria de disc fent ús de la funció *mmap*, a la funció *createEmptyRecommendationMatrix*, que retorna aquesta matriu a l'estructura *RecommendationMatrix*, que s'omple utilitzant la funció *fillRecommendationMatrix*. Un cop creada la matriu, s'emmagatzema a un fitxer utilitzant la funció *saveMatrixIntoDisk*, que guardarà la matriu de recomanació R en el fitxer `matrix.bin`. Les seves metadades (la mida de la matriu) es guarden en un altre fitxer diferent, `meta.bin`. També s'emmagatzemarà a disc la taula de hash de forma que es pugui carregar més endavant i així poder fer l'associació entre el identificador d'usuari (resp. de pel·lícula) del fitxer CSV i la fila (resp. columna) que li correspon a la matriu de recomanació R.

Tot aquest flux d'execució té com a punt d'entrada el *main* de l'arxiu `mainSave.c`, l'objectiu del qual serà crear totes les estructures necessàries per implementar el recomanador tal i com hem explicat a la introducció, guardant-les a disc per poder fer un ús ràpid de les mateixes.

2.2 Codi part 2: Recomanador de pel·lícules

Aquesta segona part implementa el recomanador donada la matriu R i les taules de hash corresponents. Aquesta part del treball té com a punt d'entrada la funció *main* del fitxer `mainRecc.c`. La implementació dels següents punts s'ha fet fent servir aritmètica de punters.

Recordar que a la matriu un valor de 0 a la $r(i, j)$ implica que l'usuari que té associada la fila i no ha valorat la pel·lícula que té associada la columna j. El mapat entre l'identificador d'usuari CustomerID (resp. pel·lícula MovieID) i la fila (resp. columna) corresponent es fa

1. El mètode *getNumberOfMoviesSeenByUser* de l'arxiu `RecommendationMatrix.c` retorna la quantitat de pel·lícules vistes per un usuari CustomerID.

2. El mètode *getNumberOfUsersThatHaveSeenMovie* retorna la quantitat d'usuaris que han vist una determinada pel·lícula MovieID.

3. Els mètodes *forecastRating* i *_getAccSum* implementen la fórmula (1).

4. El mètode *getRecommendedMovieForUser* retorna el MovieID (no la columna j de la matriu) de la pel·lícula recomanada per a un determinat usuari, veure la fórmula (3).

3 Anàlisi de l'eficiència d'execució de `mainSave`

A continuació s'analitzarà l'eficiència del codi implementat per veure quines parts són les que més triguen en executar-se. A partir d'aquí es proposen una sèrie d'experiments per modificar el codi i augmentar-ne l'eficiència. S'insisteix que en aquesta pràctica no es fa servir concurrència. L'objectiu és analitzar quines funcions poden optimitzar-se mantenint l'execució seqüencial de l'aplicació. En aquesta secció ens centrarem en el codi de la part 1: la generació de la matriu R, el codi `mainSave`. El codi de la part 2 s'analitzarà a les següents pràctiques.

Abans de començar amb les proves es demana compilar i assegurar el bon funcionament del codi amb les proves descrites al fitxer `README.txt`. Per compilar el codi disposeu del fitxer `Makefile` corresponent.

3.1 Operacions sobre la CPU

Atès que el codi fa computacions numèriques així com ús intensiu d'operacions d'entrada-sortida (per desar les dades a disc, per exemple) es dividirà l'anàlisi del codi en l'ús de la CPU i en l'ús de les operacions d'entrada-sortida.

Començarem amb l'anàlisi de l'ús de la CPU de l'aplicació `mainSave`. Per facilitar els experiments es demana modificar el codi i comentar totes les crides a *fwrite* que es realitzen a les funcions `MatrixDiskService.c` i `HashTableDiskService.c`. Aquestes dues funcions són funcions en què es fa un ús intensiu d'operacions d'entrada-sortida.

3.1.1 Valgrind/Callgrind

Callgrind és una eina que registra el nombre d'instruccions CPU que fan falta per executar cada funció així com l'historial de crides entre les funcions d'un programa. Callgrind no mesura el temps real d'execució, sinó que només mesura el temps de CPU. Aquesta eina és doncs útil per mesurar el temps de CPU de cada funció. Callgrind no inclou, però, el temps d'entrada-sortida associat al

procés i, per tant, no és útil per analitzar un procés que tingui moltes operacions d'entrada-sortida. En aquesta pràctica es realitzaran experiments de les operacions d'entrada-sortida més endavant.

Per defecte, les dades recollides consisteixen en el temps d'execució CPU a cada funció així com el mapat amb el codi font i un graf que relaciona les crides de les funcions entre sí incloent el nombre de vegades que cada funció ha sigut cridada. Un dels beneficis que té callgrind respecte altres eines com gprof és el fet que el primer analitza també les crides realitzades a funcions de llibreries compartides (com, per exemple, la llibreria d'usuari que inclou funcions com *printf*, ...).

Per analitzar el codi fent servir el callgrind cal compilar el codi amb l'opció de debugger. Per això cal incloure aquesta opció al fitxer `Makefile`.

```
CFLAGS=-O0 -g -Wall
```

Un cop compilat el codi executeu la següent instrucció

```
valgrind --tool=callgrind ./mainSave
```

El codi pot trigar una estona en executar-se (alguns minuts). Aquesta instrucció genera un fitxer anomenat `callgrind.out.<pid>`, on `<pid>` és l'identificador del procés executat, i que inclou informació de l'execució.

Un cop ha acabat l'execució podeu fer l'anàlisi dels resultats de l'historial de crides executant

```
kcachegrind
```

Aquesta aplicació mostrarà per pantalla un munt d'informació respecte el temps d'execució de cada funció així com un graf associat a les crides realitzades.

Es recomana esborrar el fitxer `callgrind.out.<pid>` cada cop que vulgueu tornar a executar el callgrind atès que sembla que la comanda kcachegrind llegeix tots els fitxers en cas que n'hi hagi múltiples d'aquest tipus.

3.1.2 Anàlisi general del codi

A partir dels resultats que us mostra l'aplicació per pantalla

- Quines són les funcions que triguen més en executar-se? Quin és el percentatge de temps que "ocupa" l'execució de cadascuna de les funcions?
- Analitzem entre altres la funció *createEmptyRecommendationMatrix*. Quina creieu que és la raó per la qual triga tant en executar-se?
- Observeu la funció *fillRecommendationMatrix*. Quines son les línies/crides que més temps "ocupen" en l'execució del codi?
- La funció anterior crida a les funcions *lookRowForUser* i *lookColForMovie*. Quina és la funció que fa "alentir" aquestes dues funcions?
- Què opineu de la funció *getMoviesFromFile*? Quines són les línies/crides que més temps "ocupen" en l'execució del codi?

A les següent secció es proposen un conjunt de modificacions a realitzar al codi i mesurar la millora en l'eficiència a nivell de CPU. Es recomana realitzar els experiments proposats i al final treure'n les conclusions globals respecte les modificacions realitzades.

3.1.3 Funció *createEmptyRecommendationMatrix*

L'objectiu és millorar el temps d'execució CPU associat a la funció *createEmptyRecommendationMatrix*. Per això mesurarem el temps d'execució del codi fent modificacions a la implementació d'aquesta funció.

Per mesurar el temps d'execució executeu aquesta instrucció

```
time ./mainSave
```

El fet d'haver comentat les funcions *fwrite* permetrà mesurar amb més fiabilitat les operacions de CPU associades a la funció *createEmptyRecommendationMatrix*.

És molt important mesurar el temps d'execució en una màquina Linux (o Mac) nativa. No ho feu dintre de la màquina virtual atès que la comanda `time` fa servir el rellotge de la màquina host per mesurar el temps execució. Si ho feu a la màquina virtual us sortiran resultats esbiaixats.

Per començar mesureu, fent múltiples execucions amb la comanda `time`, el temps d'execució del codi original. Apunteu-vos els valors de temps d'execució real, d'usuari i de sistema que surten.

- Quins valors (mitjans) d'execució real, d'usuari i de sistema us surten? Per què la suma del temps d'usuari i de sistema dóna, aproximadament, el temps real d'execució?

Anem modificar el codi de la funció *createEmptyRecommendationMatrix* i anem a veure com influeix en l'eficiència d'execució del codi.

Primer de tot, es demana modificar el codi de forma que s'inicialitzin els valors dels elements de la matriu fent servir punters i un únic bucle. Perquè la implementació sigui eficient assegureu-vos que la condició de finalització al bucle sigui un valor que s'ha calculat prèviament.

- Mesureu l'eficiència de la implementació del nou codi fent servir el callgrind/kcachegrind i compareu els resultats obtingut amb el codi original a la secció 3.1.2. Hi ha hagut millora? Segons callgrind, és útil fer servir punters per recòrrer els elements de la matriu de recomanació?

- Mesureu el temps d'execució del codi amb la instrucció `time`. Com han canviat els valors del temps d'execució respecte els valors obtinguts amb el codi original? Raoneu quins han canviat i per què d'altres no ho fan.

Es proposa fer una segona modificació. En comptes de fer servir punters es farà servir la funció *memset* de la llibreria d'usuari per inicialitzar tots els valors de la matriu a zero (amb una única crida a *memset*). De nou, es demana fer l'anàlisi fet fa un moment:

- Mesureu l'eficiència de la implementació del nou codi fent servir el callgrind/kcachegrind i compareu els resultats amb els obtinguts amb el codi original a la secció 3.1.2 així com el codi que fa servir punters. Hi ha hagut millora?

- Mesureu el temps d'execució del codi amb la instrucció `time`. Com han canviat els valors del temps d'execució respecte els valors obtinguts abans? Raoneu quins han canviat i per què d'altres no ho fan.

3.2 Operacions d'entrada-sortida

En aquesta secció es demana activar les crides a les funcions *fwrite* que s'han comentat a la secció anterior.

A la implementació que hi ha disponible actualment la matriu de recomanació és una matriu de valors flotants. És a dir, cada valor de la matriu és un element que ocupa 4 bytes (per defecte) a la memòria.

- Mesureu la mida de la matriu (nombre de columnes, nombre de files) i calculeu el que ocupa la matriu a memòria. Coincideix amb la mida del fitxer `matrix.bin` que es genera?
- Quin és el temps d'execució del codi original (sense les optimitzacions de CPU)? Què observeu al codi? Ha canviat el temps d'usuari? I el de sistema? I el temps real? Raoneu per què Per què el temps d'execució real és més gran que la suma del temps d'usuari i de sistema? Recordar que cal executar la instrucció `time` en un Linux natiu

Observeu que la matriu de recomanació és una matriu que només emmagatzemarà els valors 0, 1, 2, 3, 4 i 5. En aquest cas particular fer servir valors flotants fa que la matriu ocupi un espai de memòria gran respecte del que realment li fa falta.

Es proposa modificar el codi perquè els elements de la matriu siguin *chars* en comptes de *floats*. Tingueu en compte que s'haurà de modificar el codi a diversos punts (no hi ha gaires llocs en què s'ha de modificar el codi). Per fer les modificacions es proposa que partiu del codi que utilitza el *memset* a la funció *createEmptyRecommendationMatrix*. Assegureu-vos que el codi modificat funciona correctament fent els experiments que es proposen al `Readme.txt`.

- Quant ocupa (teòricament) la matriu si es fan servir *chars* en comptes de *floats* per emmagatzemar cadascun dels elements de la matriu?
- Mesureu el temps d'execució del codi. Com han canviat els valors del temps d'execució respecte els valors obtinguts amb la matriu de *floats*? Raoneu la resposta
- Quines conclusions traieu dels experiments realitzats?

4 Entrega de la pràctica

Es demana entregar un fitxer ZIP que inclogui les respostes raonades a les preguntes que es fan en aquesta pràctica. El nom del fitxer ha d'indicar el número del grup, seguit del nom i cognom dels integrants del grup (e.g. **P3_XX_nom1_cognom1_nom2_cognom2.zip** on XX és l'identificador de grup de parella) . L'informe a entregar ha d'estar en format `PDF` o equivalent (no s'admeten formats com `odt`, `docx`, ...).

Un informe està estructurat generalment en una introducció, treball realitzat i conclusions. A la part d'introducció es descriu breument el problema solucionat (i.e. resum del que proposa en aquesta pràctica). A la part de del treball realitzat es responen les preguntes realitzades al la pràctica i es descriuen les proves que s'han realitzat. Per respondre les preguntes podeu seguir un fil conductor que us permeti descriure el treball realitzat. També podeu respondre de forma explícita a cadascuna de les preguntes (en aquest cas indiqueu clarament a quina pregunta esteu responent en cada moment). Sigueu breus i clars en els comentaris i experiments realitzats, no cal que us

esteneu en el text. Finalment, a la part de conclusions es descriuen unes conclusions tècniques de les proves realitzades. Per acabar, es poden incloure conclusions personals.

Inclogueu, preferentment en format text, les comandes que heu executat i algun comentari breu descrivint el resultat si ho creieu necessari. En cas que preferiu incloure captures de pantalla en comptes d'incloure el resultat en format text, assegureu-vos que el text de la captura es pot llegir bé (és a dir, que tingui una mida similar a la resta del text del document) i que totes les captures siguin uniformes (és a dir, que totes les captures tinguin la mateixa mida de text).

El document ha de tenir una llargada màxima de 4 pàgines (sense incloure la portada). El document s'avaluarà amb els següents pesos: proves realitzades i comentaris associats, un 60%; escriptura sense faltes d'ortografia i/o expressió, un 20%; paginació del document feta de forma neta i uniforme, un 20%.