ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1a Calculul paralel al unui index inversat folosind paradigma Map-Reduce

Responsabili: Radu-Ioan Ciobanu, Alexandra Ana-Maria Ion

Termen de predare: 06-12-2024 23:59 (soft), 13-12-2024 23:59 (hard)
Ultima modificare: 15-11-2024 17:15

Cuprins

Cerință **2**

Paradigma Map-Reduce **2**

Index inversat **2**

Detalii tehnice **3**
Operațiile de tip Map . . . 3
Operațiile de tip Reduce . . . 3

Execuție **4**
Exemplu . . . 4

Notare **6**

Testare **7**
Docker . . . 8

Recomandări de implementare și testare **8**

Link-uri utile **8**

Cerință

Să se implementeze un program paralel în Pthreads pentru găsirea unui index inversat pentru o mulțime de fișiere de intrare. Indexul va conține lista tuturor cuvintelor diferite din fișierele de intrare, împreună cu fișierele în care acestea se găsesc.

Pentru paralelizarea procesării documentelor de intrare, se va folosi modelul Map-Reduce. Fișierele de intrare vor fi împărțite (în mod dinamic) cât mai echilibrat la niște thread-uri care le vor parsa și vor extrage cuvintele unice (operațiunea de **Map**), rezultând astfel liste parțiale cu intrări de tipul {*cuvânt, ID_fișier*} pentru fiecare fișier de intrare. Pasul următor îl reprezintă combinarea listelor parțiale (operațiunea de **Reduce**) în urma căreia se vor obține liste agregate de forma {*cuvânt,* {*ID_fișier1, ID_fișier2, ...*}} pentru fiecare cuvânt unic în parte. Pe baza listelor agregate, se va crea câte un fișier de ieșire pentru fiecare literă a alfabetului, care va conține în final toate cuvintele care încep cu acea literă și fișierele de intrare în care acestea apar. Cuvintele din fișierele de ieșire vor fi sortate descrescător după numărul de fișiere de intrare în care se regăsesc.

Paradigma Map-Reduce

Pentru rezolvarea temei, se va folosi un model Map-Reduce similar cu cel folosit la Google pentru procesarea unor seturi mari de documente în sisteme distribuite. Acest articol prezintă modelul Map-Reduce folosit de Google și o parte dintre aplicațiile lui (mai importante pentru înțelegerea modelului sunt primele 4 pagini).

Map-Reduce este un model (și o implementare asociată) de programare paralelă pentru procesarea unor seturi imense de date, folosind sute sau mii de procesoare. În majoritatea cazurilor, Map-Reduce este folosit într-un context distribuit, fiind, de fapt, un model de programare care poate fi adaptat pentru ambele situații. Cea mai cunoscută implementare este Apache Hadoop, dezvoltat inițial de către Doug Cutting și Mike Cafarella. Modelul permite paralelizarea și distribuirea automată a task-urilor. Paradigma Map-Reduce se bazează pe existența a două funcții care îi dau și numele: Map și Reduce. Funcția Map primește ca input o funcție f și o listă de elemente, și returnează o nouă listă de elemente rezultată în urma aplicării funcției f asupra fiecărui element din lista inițială. Funcția Reduce combină rezultatele obținute anterior.

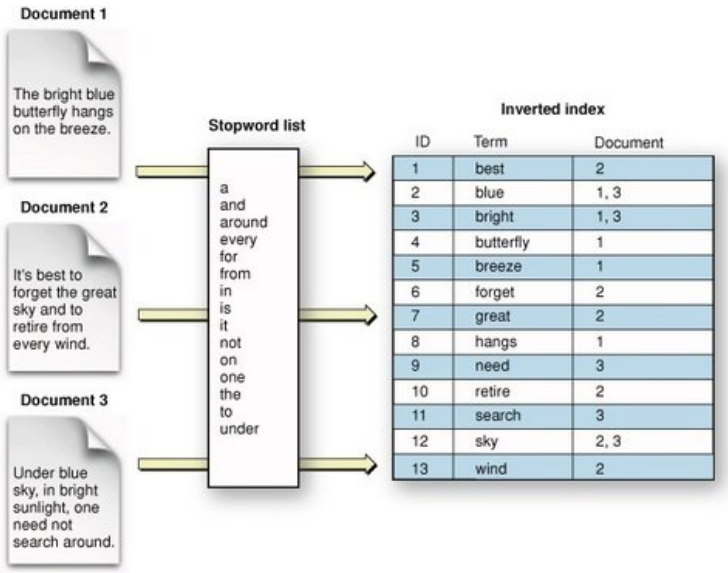
Mecanismul Map-Reduce funcționează în modul următor:

- utilizatorul cere procesarea unui set de documente
- această cerere este adresată unui proces (sau fir de execuție) coordonator
- coordonatorul asignează documentele unor procese (sau fire de execuție) de tip **Mapper**[1]
- un Mapper va analiza fișierele de care este responsabil și va genera niște rezultate parțiale, având în general forma unor perechi de tip $\{cheie, valoare\}$
- după ce operațiile Map au fost executate, alte procese (sau fire de execuție) de tip **Reducer** combină rezultatele parțiale și generează soluția finală.

Index inversat

Indexul inversat este o tehnică de indexare utilizată frecvent în domeniul regăsirii informațiilor, care împarte textul dintr-o mulțime de fișiere de intrare în cuvinte individuale și construiește un index de tipul {*cuvânt,* {*ID_fișier1, ID_fișier2, ...*}}, permițând căutări rapide pentru a determina ce documente conțin un anumit cuvânt. Imaginea de mai jos (preluată de aici) prezintă un exemplu de index inversat pe 3 documente.

[1] Acest lucru nu trebuie neapărat realizat la început, el se poate face și în mod dinamic pe măsură ce fișierele sunt procesate.



Detalii tehnice

Dându-se un set de N documente de intrare, să se creeze un index inversat organizat alfabetic folosind Map-Reduce. În implementarea temei, vor exista thread-uri care pot fi de două tipuri, Mapper sau Reducer, toate fiind pornite împreună la începutul rulării.

Operațiile de tip Map

Pornind de la lista de documente de procesat ce va fi disponibilă în fișierul de intrare, fiecare Mapper va ajunge să proceseze niște documente. Alocarea de documente thread-urilor Mapper poate fi realizată static înainte de pornirea acestora (deși nu este recomandat acest lucru), sau poate fi realizată în mod dinamic pe măsură ce documentele sunt procesate (într-o astfel de situație, un Mapper care se mișcă mai repede decât ceilalți poate să "fure" muncă de la alt Mapper, lucru care va duce la o procesare mai eficientă). Fiecare Mapper va executa următoarele acțiuni, **pentru fiecare fișier de care este reponsabil**:

- deschide fișierul și îl parcurge cuvânt cu cuvânt
- pentru fiecare cuvânt nou, se creează o pereche nouă de tipul {*cuvânt, ID_fișier*}, care este plasată într-o listă parțială
- când s-a terminat de procesat un fișier, Mapper-ul îl închide.

Operațiile de tip Reduce

Un fir de execuție de tip Reducer va fi responsabil pentru agregarea rezultatelor de la pasul de Map, sortarea lor după numărul de fișiere de intrare în care se află, și apoi scrierea lor în fișierele de ieșire. Astfel, având rezultatele din cadrul operațiunii de Map, un Reducer va realiza următoarele acțiuni:

- combină listele parțiale în liste agregate, care enumeră toate fișierele de intrare în care se găsește fiecare cuvânt
- grupează cuvintele după litera cu care încep
- sortează cuvintele care încep cu o anumită literă descrescător după numărul de fișiere de intrare în care apar.

Atenție! Înainte să începeți operațiile de tip Reduce, trebuie să vă asigurați că toate operațiile de tip Map s-au finalizat.

Execuție

Programul se va rula în felul următor:

```
./tema1 <numar_mapperi> <numar_reduceri> <fisier_intrare>
```

Atenție! Trebuie să porniți toate thread-urile (atât cele Mapper, cât și cele Reducer) într-o singură iterație a thread-ului principal. Nu se acceptă mai multe perechi de *pthread_create*/*pthread_join* în cod, acest lucru ducând la un punctaj de **0** pe întreaga temă.

Fișierul de intrare are următorul format:

```
numar_fisiere_de_procesat
fisier1
...
fisierN
```

Așadar, prima linie conține numărul de documente text de procesat, iar următoarele linii conțin numele documentelor, câte unul pe linie. Toate fișierele de intrare vor conține **doar** caractere ASCII și se pot considera valide. Aceste fișiere trebuie împărțite cât mai echilibrat thread-urilor Mapper, fără să existe fișiere neprocesate de niciun Mapper sau fișiere procesate de mai multe thread-uri Mapper.

Atenție! O împărțire echilibrată a fișierelor de procesat nu înseamnă neapărat că fiecare thread Mapper va avea câte un număr aproximativ egal de fișiere, deoarece acestea pot avea dimensiuni foarte diferite. De exemplu, dacă există trei thread-uri Mapper care au de procesat 6 fișiere de dimensiuni 5 MB, 5 MB, 1 MB, 1 MB, 1 MB, 1 MB, este mai eficient ca primele două thread-uri Mapper să proceseze câte un fișier de 5 MB și al treilea Mapper să proceseze cele patru fișiere de câte 1 MB, decât să se aloce câte două fișiere fiecărui Mapper. Acest lucru se poate face static (înaintea de pornirea thread-urilor Mapper) sau dinamic (fiecare Mapper care termină de procesat un fișier vede dacă mai sunt fișiere disponibile). O împărțire ineficientă poate afecta scalabilitatea temei și implicit punctajul obținut.

Atenție! Se va considera cuvânt orice șir de minim un caracter aflat între două spații[2], în format lowercase, fără alte caractere decât cele alfanumerice. Astfel, de exemplu, dacă în textul dintr-un fișier de intrare întâlnim *That's*, vom considera cuvântul *thats* (deci cu litere mici și fără apostrof).

Programul va avea câte un fișier de ieșire pentru fiecare literă a alfabetului, fără literele cu diacritice (*a.txt*, *b.txt*, etc.), care va conține toate cuvintele unice din fișierele de intrare care încep cu acea literă, sortate descrescător după numărul de fișiere de intrare în care apar. Dacă sunt mai multe cuvinte cu același număr de fișiere de intrare în care apar, sortarea se va face alfabetic. Fiecare cuvânt unic trebuie scris pe un rând separat, cu litere mici, în formatul *cuvânt:[ID_fișier1 ID_fișier2 ...]*.

Atenție! ID-urile fișierelor de procesat sunt date de ordinea lor în fișierul dat ca parametru la rularea temei, pornind de la 1.

Exemplu

Se dă următorul fișier de intrare:

```
$ cat test.txt
3
file1.txt
file2.txt
```

[2]Prin spațiu, înțelegem un caracter de tip whitespace (spațiu, tab, newline, etc.).

```
file3.txt
```

Cele trei documente de procesat arată în felul următor:

```
$ cat file1.txt
The bright sun shines in the blue sky as birds are singing today.

 cat file2.txt
This calm morning, the sky is blue, and gentle clouds float by.

$$ cat file3.txt
In the peaceful evening, the stars shine brightly in the quiet sky.
```

Tema se rulează cu următoarea comandă:

```
./tema1 2 3 test.txt
```

Așadar, există două thread-uri Mapper (la care se împart cele trei fișiere) și trei thread-uri Reducer (care vor agrega rezultatele finale, le vor sorta după numărul de fișiere de intrare în care apar, și vor crea fișierele de ieșire).

Cele trei fișiere de intrare se pot împărți la cele două thread-uri Mapper astfel:

- M0 → *file1.txt*
- M1 → *file2.txt*, *file3.txt*

Conform operației de Map, fiecare Mapper verifică toate numerele din fișierele asignate lui și salvează liste parțiale care conțin cuvintele unice, cu litere mici, din fiecare fișier, împreună cu ID-ul fișierului. Pentru exemplul de mai sus, operațiile de Map pot duce la următoarele rezultate (care vor fi oferite mai departe către thread-urile Reducer):

- M0 → {*are,1*}, {*as,1*}, {*birds,1*}, {*blue,1*}, {*bright,1*}, {*in,1*}, {*shines,1*}, {*singing,1*}, {*sky,1*}, {*sun,1*}, {*the,1*}, {*today,1*}
- M1 → {*and,2*}, {*blue,2*}, {*by,2*}, {*calm,2*}, {*clouds,2*}, {*float,2*}, {*gentle,2*}, {*is,2*}, {*morning,2*}, {*sky,2*}, {*the,2*}, {*this,2*}, {*brightly,3*}, {*evening,3*}, {*in,3*}, {*peaceful,3*}, {*quiet,3*}, {*shine,3*}, {*sky,3*}, {*stars,3*}, {*the,3*}

Se poate observa faptul că, dacă unele cuvinte apar de mai multe ori în același fișier (eventual și cu literă mare la început de cuvânt), ele se vor regăsi o singură dată per fișier în lista finală.

Odată ce toate operațiile de Map s-au terminat, thread-urile Reducer se apucă de agregare, sortare, și scriere în fișierele de ieșire. Conform rulării, în acest exemplu sunt trei astfel de thread-uri. Pentru eficiență, fiecare din ele va putea lua o treime din perechile rezultate după etapa de Map, deci câte 11 astfel de perechi.

Atenție! Modul în care împărțiți sarcinile către thread-urile Reducer este la latitudinea voastră. Important este ca, după etapa de agregare, sa ajungeți să aveți o listă cu toate cuvintele și fișierele în care acestea apar. O astfel de listă poate arăta în felul următor: {*and,{2}*}, {*are,{1}*}, {*as,{1}*}, {*brightly,{3}*}, {*blue,{1,2}*}, {*by,{2}*}, {*birds,{1}*}, {*bright,{1}*}, {*clouds,{2}*}, {*calm,{2}*}, {*evening,{3}*}, {*float,{2}*}, {*gentle,{2}*}, {*is,{2}*}, {*in,{1,3}*}, {*morning,{2}*}, {*peaceful,{3}*}, {*quiet,{3}*}, {*shine,{3}*}, {*stars,{3}*}, {*singing,{1}*}, {*sky,{1,2,3}*}, {*shines,{1}*}, {*sun,{1}*}, {*today,{1}*}, {*this,{2}*}, {*the,{1,2,3}*}.

În final, odată ce toate listele parțiale au fost agregate, tot thread-urile Reducer trebuie să genereze fișierele

de ieșire. Așa cum s-a menționat anterior, va exista câte un fișier de ieșire pentru fiecare din cele 26 de litere fără diacritice ale alfabetului, și fiecare din ele va conține toate cuvintele care încep cu acea literă, sortate descrescător după numărul de fișiere de intrare în care apar (și apoi alfabetic).

În exemplul de mai sus, nu există cuvinte care încep cu literele *d*, *h*, *j*, *k*, *l*, *n*, *o*, *r*, *u*, *v*, *w*, *x*, *y* sau *z*, deci fișierele aferente vor fi goale. Celelate fișiere de ieșire vor arăta în felul următor:

```
$ cat a.txt
and:[2]
are:[1]
as:[1]

$ cat b.txt
blue:[1 2]
birds:[1]
bright:[1]
brightly:[3]
by:[2]

$ cat c.txt
calm:[2]
clouds:[2]

$ cat e.txt
evening:[3]

$ cat f.txt
float:[2]

$ cat g.txt
gentle:[2]
```

```
$ cat i.txt
in:[1 3]
is:[2]

$ cat m.txt
morning:[2]

$ cat p.txt
peaceful:[3]

$ cat q.txt
quiet:[3]

$ cat s.txt
sky:[1 2 3]
shine:[3]
shines:[1]
singing:[1]
stars:[3]
sun:[1]

$ cat t.txt
the:[1 2 3]
this:[2]
today:[1]
```

Notare

Tema se poate testa local, direct pe Linux/WSL, sau folosind Docker, după cum se explică mai jos. Tema se va încărca pe Moodle, sub formă de arhivă Zip care, pe lângă fișierele sursă C/C++, va trebui să conțină următoarele două fișiere **în rădăcina arhivei**:

- *Makefile* - cu directiva *build* care compilează tema voastră (**fără flag-uri de optimizare**) și generează un executabil numit *tema1* aflat în rădăcina arhivei, și directiva *clean* care șterge executabilul
- *README* - fișier text în care să se descrie pe scurt implementarea temei.

Punctajul este divizat după cum urmează:

- **48p** - scalabilitatea soluției
- **36p** - corectitudinea rezultatelor (la rulări multiple pe aceleași date de intrare, trebuie obținute aceleași rezultate)[3]
- **16p** - claritatea codului și a explicațiilor din README.

Nerespectarea următoarelor cerințe va duce la depunctări:

[3]Acest punctaj este condiționat de scalabilitate. O soluție secvențială, deși funcționează corect și dă rezultate bune, nu se va puncta.

- **-100p** - ne-urmărirea paradigmei Map-Reduce conform descrierii din enunț
- **-100p** - pseudo-sincronizarea firelor de execuție prin funcții cum ar fi `sleep`
- **-100p** - utilizarea altor implementări de thread-uri în afară de Pthreads (cum ar fi *std::thread* din C++11)
- **-100p** - crearea și oprirea de thread-uri în mod repetat (pentru a nu lua această depunctare, trebuie să creați un singur set de cel mult $M + R$ thread-uri la început, unde M e numărul de thread-uri Mapper, iar R numărul de thread-uri Reducer, așa cum sunt date în argumentele programului)
- **-100p** - pornirea a mai mult de $M + R$ thread-uri
- **-100p** - utilizarea de flag-uri de optimizare în Makefile
- **-20p** - utilizarea variabilelor globale (soluția pentru a evita variabile globale este să trimiteți variabile și referințe la variabile prin argumentele funcției pe care o dați la crearea firelor de execuție)
- **-20p** - nerespectarea formatului arhivei încărcate.

Atenție! Checker-ul vă dă cele 84 de puncte pentru scalabilitate și corectitudinea rezultatelor, dar acela nu este punctajul final (dacă, de exemplu, temă vă scalează și dă rezultate corecte, dar nu ați urmat paradigma Map-Reduce, veți avea nota finală 0).

Testare

Pentru a vă putea testa tema, găsiți în repository-ul temei un set de fișiere de intrare de test, precum și un script Bash (numit *checker.sh*) pe care îl puteți rula pentru a vă verifica implementarea. Acest script va fi folosit și pentru testarea finală.

Pentru a putea rula scriptul așa cum este, trebuie să aveți următoatoarea structură de fișiere:

```
$ tree
.
+-- checker
|   +-- test_in
|       +-- [...] (fisierele de intare ale testului mare)
|   +-- test_in_small
|       +-- [...] (fisierele de intare ale testului mic)
|   +-- test_out
|       +-- [...] (fisierele de iesire ale testului mare)
|   +-- test_out_small
|       +-- [...] (fisierele de iesire ale testului mic)
|   +-- checker.sh
|   +-- test_small.txt
|   +-- test.txt
+-- src
|   +-- [...] (sursele voastre)
+-- docker-compose.yml
+-- run_with_docker.sh
+-- Dockerfile
```

La rulare, scriptul *checker.sh* din directorul *checker* execută următorii pași:

1. compilează programul
2. pentru testul mare, execută următoarele operații:

   (a) rulează varianta etalon cu 1 thread Mapper și 1 thread Reducer și verifică dacă rezultatele sunt corecte
   (b) rulează variante cu 1/2/4 thread-uri Mapper și 1/2/4 thread-uri Reducer și verifică dacă rezultatele sunt corecte
   (c) pentru testele cu minim 6 thread-uri, calculează accelerația și verifică dacă este peste niște limite așteptate
3. se calculează punctajul final din cele 84 de puncte alocate testelor automate (16 puncte fiind rezervate pentru claritatea codului și a explicațiilor, așa cum se specifică mai sus).

Scriptul necesită existența următoarelor utilitare: *awk*, *bc*, *diff*, *sed*, *time*, *timeout*, *xargs*. Totuși, pentru ușurință, vă recomandăm să nu rulați totuși script-ul direct, ci să folosiți varianta cu Docker prezentată mai jos.

Vă mai recomandăm să vă apucați de implementat tema și să încercați să o faceți să funcționeze (manual) pe fișierul de input *test_small.txt*, care este fix exemplul prezentat mai sus. Când ați obținut rezultate corecte pe acest test, puteți încerca să rulați și pe testul mare și să verificați scalabilitatea.

Docker

Pentru a avea un mediu uniform de testare, sau pentru a putea rula scriptul de test mai ușor de pe Windows sau MacOS (și cu mai puține resurse consumate decât dintr-o mașină virtuală), vă sugerăm să folosiți Docker. În repository-ul temei, găsiți un script numit *run_with_docker.sh*. Dacă aveți Docker instalat și rulați acest script, se va rula testul automat în interiorul unui container.

Recomandări de implementare și testare

Găsiți aici o serie de recomandări legate de implementarea și testarea temei:

1. la compilare, folosiți flag-urile `-Wall` și `-Werror`; `-Wall` vă afișează toate avertismentele la compilare, iar `-Werror` vă tratează avertismentele ca erori; aceste flag-uri vă ajută să preîntâmpinați erori cum ar fi variabile neinițializate, care se pot propaga ușor
2. folosiți un repository **privat** de Git și dați commit-uri frecvent; vă ajută să vedeți diferențe între iterații de scris cod și vă asigură că aveți undeva sigur codul ca să-l puteți recupera în cazul în care l-ați șters din greșeală sau mașina pe care lucrați nu mai funcționează
3. citiți paginile de manual pentru funcțiile din biblioteca Pthreads și verificați valorile returnate de acestea; dacă aceste valori corespund unor erori descrise acolo, vă puteți da seama ușor dacă folosiți greșit funcțiile sau structurile din bibliotecă
4. dacă lucrați pe Windows, vă recomandăm să folosiți WSL2 sau Docker (în loc de o mașină virtuală) pentru dezvoltare în Linux, deoarece sunt soluții care necesită mai puține resurse, iar performanța e mai aproape de cea a unui sistem de operare Linux care rulează direct pe mașinile voastre
5. folosiți un IDE precum CLion sau Visual Studio Code; pe lângă completarea automată a codului, aveți posibiliatea de a rula și a face debug pe cod.

Link-uri utile

1. The Anatomy of a Large-Scale Hypertextual Web Search Engine
2. Can Your Programming Language Do This?
3. MapReduce (Wikipedia)

4. MapReduce: Simplified Data Processing on Large Clusters
5. Apache Hadoop
6. Apache Spark.