O imagine care conține text, exterior, semn

Descriere generată automat

**UNIVERSITATEA TEHNICĂ “GHEORGHE ASACHI” DIN IAȘI**

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI**

**DISCIPLINA ALGORITMI PARALELI SI DISTRIBUIȚI**

**APLICAȚIE MAP REDUCE**

**Student,**

**Vlad Batalan, grupa 1410A**

**Iași, 2022**

1. **Prezentarea scurtă a soluției**

Problema avută în vedere este stocarea sub forma unui index invers a unei colecții de documente. Un exemplu în acest sens este dat de motoarele de căutare a informațiilor pe Web. Fiecare fișier din folderul de date de intrare are propriul id (docID) și acesta trebuie parsat și spart în cuvinte unice, astfel încât, în fisierul rezultat să se regăsească înregistrări sub forma:

<

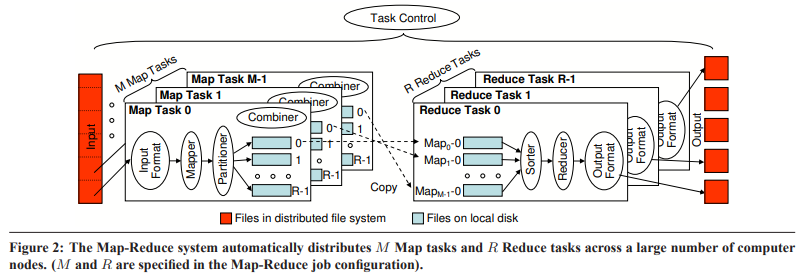
Soluția abordată de aplicația realizată constă în impărțirea task-urilor la mai mulți workeri care execută operații în mod paralel. Această abordare este preferată, deoarece a trata problema într-un mod clasic, secvențial, necesită foarte mult timp de procesare. Modelul din figura 1 preluat de la referința [1] reprezintă soluționarea unei probleme de numărare a apariției fiecărui cuvânt din documentele de intrare.

În cazul prezentat, datele se află în două fișiere: File1 și File2. După realizarea pasului de Mapare, acestea vor fi aranjate în perechi de tip cheie valoare de forma <mid\_key, mid\_value>. Aceste perechi sunt grupate printr-un procedeu cât se poate de optim după mid\_key. Procedeul Reduce grupează și numără câte perechi sunt cu aceași cheie.

O imagine care conține masă

Descriere generată automat

Pentru eficiență sporită, procedeul de mapare poate fi împărțit în M task-uri diferite care lucrează îndependent, fiecare având asignată o porțiune de date de parsat. După gruparea datelor, procedeul de reducere este realizat de R task-uri care lucrează de asemenea în mod independent și scriu rezultatele în diferite fișiere de output (figura 2 preluată din referința [1]). De asemenea, există două tipuri de worker: worker-ul simplu care se ocupa cu realizarea operațiilor de Map și de reduce pentru datele asignate pentru procesare, și coordonatorul care se ocupă cu orchestrarea si comandarea workerilor (inițiază procesul, oferă task-uri, așteaptă workerii să finalizeze și colectează datele).



1. **Pseudocodul algoritmilor implementați**

Worker: MapFunction

* un worker primește toate căile către fisierele pe care trebuie să le proceseze și pentru fiecare fișier, el parsează cuvintele și le salvează în directorul propriu în fișiere text sub forma <litera\_de\_inceput>.txt

Algoritm:

Functie MapLoop(){

Crează directorul de lucru (worker\_dir = „worker\_<id>”);

// Primeste numarul de fisier de procesat

Recv(num\_files);

Pentru i de la 0 la num\_files{

// Primeste calea catre fisierul de procesat

Recv(file\_path);

FileMapper(file\_path, worker\_dir);

}

// Comunica faptul ca task-ul a fost terminat catre coordinator

SendFinishedTaskMessage();

}

Functie FileMapper(file\_path, worker\_dir){

cuvinte = ParseazaFisier(file\_path);

pentru cuvant in cuvinte{

fisier\_de\_scris = worker\_dir + “\\” + cuvant[0] + “.txt”;

Scrie in fisier\_de\_scris perechea <cuvant, file\_name>;  
 }

}

Worker: ReduceFunction

* worker-ul primește de la coordonator numărul de workeri exstenți (existing\_workers), folderul intermediar unde toți workerii scriu informațiile intermediare de la procesul de mapping (intermediar\_file) și folderul de output (output\_file).
* acesta va primi mai multe litere de la coordonator și va vizita si procesa fișierele procesate de fiecare worker care are ca denumire <litera>.txt, scriind rezultatele în fișierul <litera>.txt din folderul de output (output\_file).

Algoritm:

Functie ReduceLoop(){

// Primeste numarul de fisier de procesat

Recv(num\_files);

Pentru i de la 0 la num\_files{

// Primeste numele fisierului de procesat (doar o litera aici)

Recv(file\_name);

FileReducer(existing\_workers, file\_name, intermediar\_file, output\_file);

}

// Comunica faptul ca task-ul a fost terminat catre coordonator

SendFinishedTaskMessage();

}

Functie FileReducer(existing\_workers, file\_name, intermediar\_file, output\_file){

// Map-ul care retine structura de forma

// { <word> : { <filename> : <count> }

reduce\_map = map(string, map(string, int));

Pentru worker\_id de la 1 la existing\_workers + 1{

// Folder-ul creat de worker-ul cu worker\_id dat

// cu numele fisierului de procesat ptimit de la coordonator

fisier\_curent = worker\_<worker\_id>/<file\_name>;

Daca nu exista (fisier\_curent){

Continua;

}

Cat timp (citeste cuvant si fisier\_provenienta din fisier\_curent){

Adauga in reduce\_map(cuvant, fisier\_provenienta);

}

}

// Creaza fisierul de output

fisier\_output = output\_file/file\_name;

Pentru fiecare element item din reduce\_map{

// Elementul este de forma <cuvant, lista\_fisiere>

// Fiecare element din lista\_fisiere are forma <nume, count>

Scrie in fisier\_output ( “<item.cuvant>” )

Pentru fiecare element fisiere din item.lista\_fisiere{

Scrie in fisier\_output( “(<fisiere.nume>, <fisiere.count>)” );

}

Scrie in fisier\_output (“\n”);

}

}

Coordonator: MapReduceFunction

* știind căile către fișierele de procesat (input\_file\_paths), folder-ul intermediar și cel de output, coordonează toți workerii (workers) pentru a realiza etapele de Map și Reduce

Algoritm:

Functie MapReduce(){

// Realizeaza etapa de Map

MapStep(input\_file\_paths);

// Realizeaza etapa de Reduce

ReduceStep();

// Termina activitatea tuturor workerilor

TrimiteTuturor( “exit” );

}

Functie MapStep(input\_file\_paths){

// Comunica tuturor workerilor sa se pregateasca pentru faza de map

TrimiteTuturor( “map” );

// Comunica tuturor workerilor care este folderul intermediar in care

// isi vor scrie rezultatele etapei

TrimiteTuturor( intermediar\_path );

// Apeleaza functia care imparte workerilor fisierele de procesat prin

// metoda RoundRobin

RoundRobinTasks( input\_files\_paths );

// Asteapta sa termine task-ul

WaitForWorkers();

}

Functie ReduceStep(input\_file\_paths){

// Comunica tuturor workerilor sa se pregateasca pentru faza de reduce

TrimiteTuturor( “reduce” );

// Comunica tuturor workerilor numarul total de workeri existenti

TrimiteTuturor( workers );

// Comunica tuturor workerilor care este folderul de output in care

// isi vor scrie rezultatele etapei

TrimiteTuturor( output\_path );

// Creaza vectorul cu task-uri de impartit la workeri

to\_process = new vector();

// Creaza fisierele de output

Pentru caracter in [a-z0-9] {

Creaza fisier( “<output\_file>/<caracter>.txt”);

to\_process.push\_back( “<caracter>.txt” );

}

// Round robin task-uri la workeri

RoundRobinTasks( to\_process );

// Asteapta sa termine task-ul

WaitForWorkers();

// Colecteaza rezultatele pe care le scrie in result.txt din

// folderul de output

CollectAnswers(to\_process);

}

1. **Explicații scurte asupra implementării și funcționării soluției**

Implementarea algoritmului MapReduce a fost realizată în limbajul C++. Gestionarea lucrului cu fișiere a fost realizată cu bibliotecile “filesystem” [5] și “fstream” [6]. Pentru lizibilitate, dar și reutilziarea porțiunilor de cod, a fost utilizată paradigma orientată obiect. În cadrul proiectului se evidențiază trei entități de sine stătătoare: CoordonatorUnit, Worker și Logger (figura 3). Acestea utilizează comunicarea oferită de MPI pentru a se coordona și pentru a realiza pașii de Map Reduce [4].

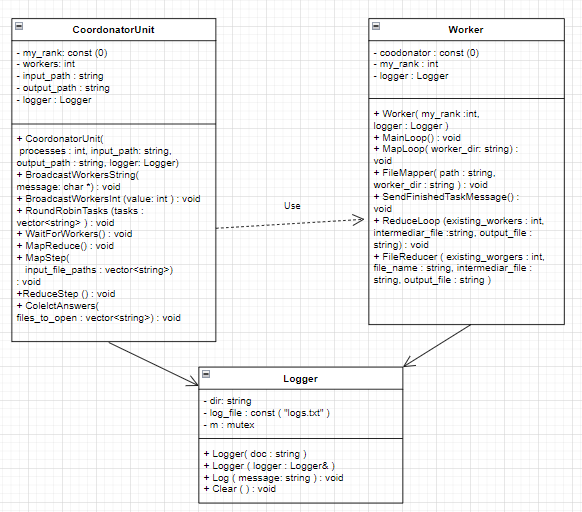
****

Figura 3

Componenta Logger este utilizată pentru a urmări exact procesul de comunicare dintre cele două entități existente în aplicatie. Fișierul de output redat de acesta este numit logs.txt și conține informatii organizate în felul urmator:

O imagine care conține text, masă

Descriere generată automat

Figura 4

Unitatea Worker are o buclă care așteaptă în permanență mesaje din partea Coordonatorului, iar în funcție de aceste mesaje, poate fi configurat să execute etapa de Map (la primirea mesajului “map”), etapa de Reduce (la primirea mesajului “reduce”) sau de a-și termina activitatea (la primirea mesajului “exit”). Odată inițializată o etapa, workerul inițiază o secvență specifică de comunicare cu coordonatorul prin care primește informațiile necesare pentru prelucrare (numărul de task-uri, fișierul de output, task-urile în sine), la finalul căruia îi trimite coordonatorului mesajul “end\_task” (figura 5).

O imagine care conține text, interior

Descriere generată automat

Figura 5

Structura de fișiere este constituită din 3 foldere: Input (folderul unde se află toate fișierele care sunt de procesat), Intermediar (utilizat de către workeri pentru a aranja datele intermediare și pentru a le grupa ușor) si Output (folderul de rezultate).

Folderul Intermediar are pentru fiecare worker cate un folder dedicat numit worker\_<id> în care acesta creează fișiere denumite <litera>.txt. (figura 6, 7)

Folderul de Output conține fișiere de tipul <litera>.txt cu rezultate scrise de workeri, dar și fișierul result.txt realizat de către coordonator care colectează și compactează rezultatele de la workeri(figura 8). Formatul fișierului result.txt este ilustrat în figura 9.

O imagine care conține masă

Descriere generată automat**O imagine care conține masă

Descriere generată automat**

Figura 6 Figura 7

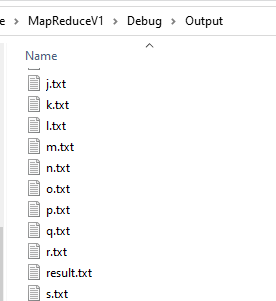
****

Figura 8

O imagine care conține text

Descriere generată automat

Figura 9

1. **Bibliografie**

[1] „Map-Reduce Meets Wider Varieties of Applications”, Shimin Chen, Steven W. Schlosser, IRP-TR-08-05

[2] IBM Corp. What is MapReduce? <https://www.ibm.com/analytics/hadoop/mapreduce>.

[3] Michael Kleber. The MapReduce paradigm. <https://sites.google.com/site/mriap2008/lectures>, January 2008. [4] Michael Kleber. What is MapReduce? https://sites.google.com/site/mriap2008/lectures, accesat pe data: January 2022.

[4] Wikipedia. MapReduce. <http://en.wikipedia.org/wiki/MapReduce>, accesat pe data: January 2022.

[5] Filesystem library. <https://en.cppreference.com/w/cpp/filesystem>.

[6] Fstream library. <https://www.cplusplus.com/reference/fstream/fstream/>.

[7] STL library. <https://www.geeksforgeeks.org/the-c-standard-template-library-stl/>.

[8] Mutex. <https://en.cppreference.com/w/cpp/thread/mutex>.

[9] OpenMpi documentation. <https://www.open-mpi.org/doc/>.

[10] V. Kumar, A. Grama A. Gupta & G Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Addison Wesley, 2003

[11] I H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, John Wiley & Sons, Inc., 2004

[12] I K. Berman, J. Paul, Algorithms: Sequential, Parallel, and Distributed, Thomson Learning, Inc., 2005

[13] I T. Mattson, B. Sanders, B. Massingill, Patterns for Parallel Programming, Addison-Weslwy, 2005