

Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Domeniul: Calculatoare și Tehnologia Informației
Specializarea: Tehnologia Informației

Algoritmul MapReduce

Temă de casă

Student: Ignătescu Ștefan
Grupa: 1409A

Iași, 2020

Cuprins

Enunțul problemei.....	1
Noțiuni teoretice.....	1
Prezentarea soluției.....	2
Implementare.....	2
Bibliografie.....	13

Enunțul problemei

Tema de casă constă în implementarea unei soluții MPI de tip MapReduce pentru problema construirii unui index invers pentru o colecție de documente text.

Aplicația de test va primi ca parametri de intrare numele unui director ce conține fișiere text (cu extensia „.txt”) și un nume de director pentru stocarea datelor de ieșire și va genera pe post de răspuns un set de fișiere text ce conțin indexul invers corespunzător colecției de documente de intrare.

Noțiuni teoretice

Termenul MapReduce referă, în prezent, un tipar de dezvoltare a aplicațiilor paralele/distribuite ce procesează volume mari de date. În general, se considera că acest model implică existența unui nod de procesare cu rol de coordonator (sau master) și mai multe noduri de procesare cu rol de worker.

Algoritmul MapReduce conține 2 sarcini de lucru importante, anume maparea și reducerea. Etapa de mapare preia un set de date și îl convertește într-un alt set de date, unde elementele individuale sunt împărțite în perechi cheie / valoare. Etapa de reducere preia ieșirea de la etapa de mapare ca fiind datele de intrare și combină aceste tuple, rezultând un alt set de tuple, dar de dimensiuni mai mici.

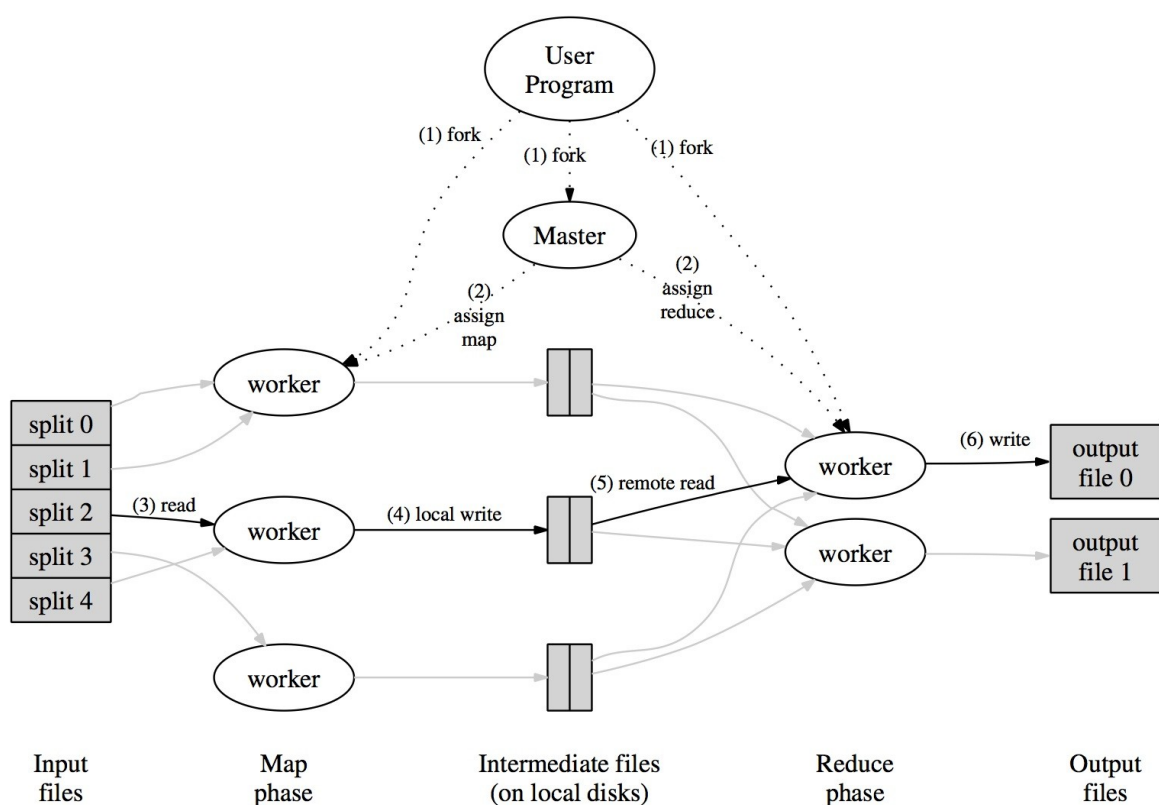


Figura 1: Paradigma MapReduce

Prezentarea soluției

Tema de casă constă în implementarea unei soluții MPI de tip MapReduce pentru problema construirii unui index invers pentru o colecție de documente text. Aplicația primește ca parametru de intrare numele unui director ce conține fișiere text (cu extensia „.txt”) și un nume de director pentru stocarea datelor de ieșire și va genera pe post de răspuns un set de fișiere text ce conține indexul invers corespunzător colecției de documente de intrare.

La rularea aplicației, un proces prestabilit, creat de MPI, își asumă rol de coordonator (master), iar ceilalți sunt workeri. Coordonatorul va deschide folderul de intrare și va salva numele fișierelor într-un vector.

În etapa de mapare procesul master va asocia fiecărui worker câte un fișier din vectorul de fișiere de intrare. Worker-ul va deschide fișierul primit, va împărți textul în cuvinte distincte și va reține numărul de apariții al fiecărui cuvânt. Datele vor fi stocate în formatul <cheie, valoare> într-un dicționar, iar apoi vor fi scrise într-un fișier cu numele “nume_fisier.txt” dintr-un director temporar. Etapa de mapare se termină atunci când toate fișierele de intrare sunt preluate și se creează fișierele corespunzătoare cu numărul de apariții a cuvintelor distincte. După ce va fi finalizată aceasta etapă, în folderul temporar vom avea indexul direct pentru fiecare fișier.

În etapa de reducere procesul master va asocia următorului worker sarcina de mapare. Worker-ul va crea un fișier nou în care vor fi preluate date din etapa de mapare. Procesul desemnat de master va parcurge fișierele și va pune datele într-un dicționar de tipul: <termk, {docIDX : countk}> formndu-se astfel indexul invers. Etapa de reducere se consideră finalizată atunci când worker-ul a terminat de procesat fișierele din etapa de mapare.

Implementare

Soluția descrisa mai sus a fost implementată în c++ și codul sursa este prezentat în continuare:

main.cpp

```
#include <string>
#include <stdio.h>
#include <mpi/mpi.h>
#include <fstream>
#include <iostream>
#include <dirent.h>
#include <list>
#include <vector>
#include <algorithm>

#include "utils.h"

using namespace std;

void Test()
{
    Reducer(DIRECTORY_OUT);
}
```

```

int main(int argc, char **argv)
{
    int rank;
    int np;
    int st=1;
    char message[]="Gata maparea";
    char buff[MAX_VAL];

    MPI_Request req;
    MPI_Status status;

    bool isMapped=false;
    bool isReduce=false;

    int *slaves;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&np);

    if(rank==ROOT)
    {
        //iau numele fisierelor din folderul din linia de comanda
        list<string> list=getFileNameFromDirectory(argv[1]);
        int noFiles=list.size();
        int noMappeFile;
        slaves=initSlaves(np,ROOT);
        //showlist(list);
        //cout<<list.size();
        int i=0;
        //faza de mapare
        //golesc folderul de mapare
        deleteFilesFromDirectory(DIRECTORY_MAPP);

        while (!isMapped)
        {
            //trebuie sa trimit taskuri de mapare
            while(i<np )
            {
                string docName=list.front();
                if(i!=ROOT)
                {
                    char buff[MAX_VAL];
                    char *temp=stringToChar(docName);
                    strcpy(buff,temp);
                    list.pop_front();
                    //send
                    MPI_Isend(buff,docName.length()
                    +1,MPI_CHAR,i,MAPPER_TAG,MPI_COMM_WORLD,&req);
                    cout<<"Radacina "<<ROOT<<" a trimis mesajul: "<<buff<<" procesului
                    "<<i<<"\n";
                }
                i++;
            }
            if(i>=np && !list.empty())

```

```

{
i=0;
}
slaves[i]++;//
}
if(list.empty())
{
isMapped=true;
}
}
//print(slaves,np);

for(int j=1;j<np;j++)
{
MPI_Isend(message,strlen(message)
+1,MPI_CHAR,j,END_MAP_TAG,MPI_COMM_WORLD,&req);
//cout<<"Trimit la "<<j<<" "<<message<<" tag="<<END_MAP_TAG<<"\n";
}

for(int j=1;j<np;j++)
{
while(slaves[j]-->0)
{
MPI_Recv(buff,MAX_VAL,MPI_CHAR,j,MAPPER_TAG,MPI_COMM_WORLD,&status);
cout<<"Radacina am primit "<<buff<<" de la procesul
"<<status.MPI_SOURCE<<" cu tagul:"<<status.MPI_TAG<<"\n";
}
}
i++;
if(i>=np)
i=1;

for(int j=1;j<np;j++)
{
if(j==i)
{
MPI_Isend(argv[2],strlen(argv[2])
+1,MPI_CHAR,i,REDUCER_TAG,MPI_COMM_WORLD,&req);
cout<<"Trimit reducece procesului "<<i<<"\n";
}
else
{
char red[]="Sunt liber";
MPI_Isend(red,strlen(red)+1,MPI_CHAR,j,CHECK_MESSAGE,MPI_COMM_WORLD,&req);
cout<<"Trimit la "<<j<<" "<<red<<" libertate tag="<<CHECK_MESSAGE<<"\n";
}
}
MPI_Recv(buff,MAX_VAL,MPI_CHAR,i,REDUCER_TAG,MPI_COMM_WORLD,&status);
cout<<"Am finalizat reducearea in "<<buff<<" de la procesul "<<i<<"\n";
delete[] slaves;
}
else
{

```

```

while(true)
{
MPI_Recv(buff,MAX_VAL,MPI_CHAR,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&
status);
if(status.MPI_TAG==MAPPER_TAG)
{
char filename[MAX_VAL];
strcpy(filename,argv[1]);
strcat(filename,buff);
cout<<rank<<" "<<filename<<"\n";
string mapper=Mapper(filename);
char* response=stringToChar(mapper);

cout<<"Procesul "<<rank<<" a primit "<<buff<<" de la RADACINA, si mapeaza
in fisierul="<<filename<<" in "<<mapper<<"\n";
//trimit la master ca am terminat cu de procesat fisierul;
MPI_Send(response,strlen(response)
+1,MPI_CHAR,ROOT,MAPPER_TAG,MPI_COMM_WORLD);
}
else if(status.MPI_TAG==END_MAP_TAG)
{
//cout<<buff<<" proces "<<rank<<"\n";
break;
}
}
MPI_Recv(buff,MAX_VAL,MPI_CHAR,MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,&
status);
if(status.MPI_TAG==REDUCER_TAG)
{
cout<<"Sunt procesul: "<<rank<<" si fac reducerea"<<buff<<"\n";
string reduce=Reducer(buff);
//cout<<reduce<<"\n\n";
MPI_Send(stringToChar(reduce),reduce.size()
+1,MPI_CHAR,ROOT,REDUCER_TAG,MPI_COMM_WORLD);
}
else
{
cout<<"Sunt procesul: "<<rank<<" si sunt liber cu mesajul "<<buff<<"\n";
}
}
MPI_Finalize();
return 0;
}

```

utils.h

```

#pragma once

#include <list>
#include <vector>
#include <string>

```

```

#include <map>

using namespace std;

#define ROOT 0
#define MAX_VAL 100
//debug -> deprecated
#define REDUCCER_MASTER 0

#define MAPPER_TAG 100
#define REDUCER_TAG 101
#define END_MAP_TAG 102
#define CHECK_MESSAGE 103

#define DIRECTORY_MAPP
"/home/stefan/Desktop/alpd-MapReduce/solution/dateMapper/"
#define DIRECTORY_IN
"/home/stefan/Desktop/alpd-MapReduce/solution/dataIn/"
#define DIRECTORY_OUT
"/home/stefan/Desktop/alpd-MapReduce/solution/dataOut/"

char *stringToChar(string test);
void showlist(list<string> g);
int *initSlaves(int noSlaves,int master);
list<string> getFileNameFromDirectory(const char *filePath);
void deleteFilesFromDirectory(const char * dirPath);

vector<string> readFile(const char *filename);
string Mapper(const char *filename);

string Reducer(const char *outputfolder);
void print(int *a,int n);

```

utils.cpp

```

#include "utils.h"

#include <iostream>
#include <fstream>
#include <algorithm>

#include <dirent.h>

#include <stdio.h>
#include <cstring>

#include <list>
#include <vector>
#include <string>
#include <map>

```



```

using namespace std;

void showlist(list<string> g)
{
    list<string> :: iterator it;
    for(it = g.begin(); it != g.end(); ++it)
        cout << *it<<"\n";
}

list<string> getFileNameFromDirectory(const char *filePath)
{
    list<string> fileName;

    DIR *dir;
    struct dirent *ent;
    if ((dir = opendir (filePath)) != NULL)
    {
        /* print all the files and directories within directory */
        while ((ent = readdir (dir)) != NULL)
        {
            // cout<<ent->d_name<<"\n";
            if(ent->d_type == DT_REG)
            {
                fileName.push_front(ent->d_name);
            }
        }
        closedir (dir);
        return fileName;
    }
    else
    {
        perror ("");
        return fileName;
    }
    return fileName;
}

string removeSpecialCharacter(string s)
{
    for (int i = 0; i < s.size(); i++)
    {
        // range
        if (s[i] < 'A' || s[i] > 'Z' && s[i] < 'a' || s[i] > 'z')
        {
            // cout<<s<<"s[]"<<s[i]<<" ";
            s.erase(i, 1);
            i--;
        }
    }
    return s;
}

//deprecated
string tokenizing(string s)
{

```

```

string chars = "~#*&%/[ ] !\"'\\.,-?!;:(){}$";
for (char c: chars) {
s.erase(std::remove(s.begin(), s.end(), c), s.end());
}
return s;
}
vector<string> readFile(const char *filename)
{
ifstream myReadFile;
//cout<<" sunt in read file cu filename="<<filename<<"\n";

myReadFile.open(filename);
//char output[100];
string output;
vector<string> text;
if(myReadFile.is_open())
{
while (!myReadFile.eof())
{
myReadFile>>output;
text.push_back(removeSpecialCharacter(output));
//cout<<output<<"\n";
}
}
myReadFile.close();
return text;
}
char *stringToChar(string test)
{
int n=test.length();
char *array=new char[n+1];
strcpy(array,test.c_str());
//cout<<array<<" aici \n";
return array;
}

string Mapper(const char *filename)
{
//cout<<"Mapper cu filename="<<filename<<"\n";
string mapfile(filename);
std::size_t found = mapfile.find_last_of("/\\");
mapfile=DIRECTORY_MAPP+mapfile.substr(found+1);
auto text = readFile(filename);
//cout<<" "<<filename<<" debug mapper\n";
if( remove(stringToChar(mapfile)) == 0 ); //sterg fisierul daca exista

ofstream file(mapfile); //open in constructor
map<string,int> dictionary;
for(int i=0;i<text.size();i++)
{
if(text.at(i)!="")
{
string word=text.at(i);

```

```

if(dictionary.find(word)!=dictionary.end())
{
//exista cuvantul in dictionar
dictionary[word]++;
}
else
{
dictionary.insert(pair<string,int>(word,1));
}
}
else
{
//cout<<"cuvand invalid\n";
}
}
map<string, int>::iterator itr;
for (itr = dictionary.begin(); itr != dictionary.end(); ++itr)
{
file<<itr->first<<" "<< itr->second <<"\n";
//cout<<itr->first<<" "<< itr->second <<"\n";
}
file.close();
return mapfile;

}

string Reducer(const char* outputfolder)
{
list<string> files=getFileNameFromDirectory(DIRECTORY_MAPP);
map<string,map<string,int>> reverseIndex;
map<string,int> directIndex;
ifstream myReadFile;

string outfile(outputfolder);
outfile+="output.txt";
if( remove(stringToChar(outfile)) == 0 ); //sterg fisierul daca exista

ofstream outStream(outfile); //open in constructor
//deschid fiecare fisier

for(string file:files)
{
string temp=DIRECTORY_MAPP;
temp+=file;
myReadFile.open(temp);
if(myReadFile.is_open())
{
//citesc datele si le pun in indexul direct
// cout<<"Am deschis fisierul: "<<temp<<"\n";
while (!myReadFile.eof())
{
string key;
int value;

```

```

myReadFile>>key>>value;
if(key!="" && value!=0)
{
//cout<<key<<" valoare "<<value<<"\n";
directIndex.insert(pair<string,int>(key,value));
}
}
//cout<<temp<<"\n";//numele fisierului procesat
//scot ultimul spatiu;
auto it = directIndex.begin();
directIndex.erase(it);
//parcure indexul direct
map<string, int>::iterator itr;
for (itr = directIndex.begin(); itr != directIndex.end(); ++itr)
{
string term=itr->first;
int value=itr->second;
string doc=file;
if(reverseIndex.find(term)!=reverseIndex.end())
{
//exista termenul in dictionar
map<string,int> docCounter;
docCounter[doc]=value;
auto element=pair<string,int>(doc,value);
reverseIndex[term].insert(element);
}
else
{
//nu exista
map<string,int> docCounter;
docCounter[doc]=value;
auto element=pair<string,int>(doc,value);
reverseIndex.insert(pair<string,map<string,int>>(term,docCounter));
}
}
myReadFile.close();
directIndex.clear();
}
else
{
cout<<"NU am deschis fisierul: "<<temp<<"\n";
}
}

//pun in fisier datele din map
auto it=reverseIndex.begin();
for (it = reverseIndex.begin(); it != reverseIndex.end(); ++it)
{
string term=it->first;
auto mapTerm=it->second;
//cout<<term<<": {\n";
ostream<<term<<": {\n";
auto itMap=mapTerm.begin();

```

```

for(itMap = mapTerm.begin(); itMap != mapTerm.end(); ++itMap)
{
//cout<<"\t"<<itMap->first<<" : "<<itMap->second<<"\n";
outStream<<"\t"<<itMap->first<<" : "<<itMap->second<<"\n";
}
//cout<<"}\n";
outStream<<"}\n";
}
outStream.close();
return outfile;
}

int *initSlaves(int noSlaves,int master)
{
int *slaves=new int[noSlaves];
for(int i=0;i<noSlaves;i++)
{
if(i==master)
{
slaves[i]=-1;
}
else
{
slaves[i]=0;
}
}
return slaves;
}

void deleteFilesFromDirectory(const char * dirPath)
{
list<string> files=getFileNameFromDirectory(dirPath);
string path(dirPath);
for(auto el:files)
{
el=dirPath+el;
if( remove(stringToChar(el)) == 0 );
}
}

void print(int *a,int n)
{
cout<<"Functia de print:\n";
for(int i=0;i<n;i++)
{
cout<<a[i]<<" ";
}
cout<<"\n";
}

```

Compilare:

```
mpic++ main.cpp utils.cpp -o main
```

Executare:

```
mpirun --hostfile hostfile -np 5 main directoryInput directoryOutput
```

Bibliografie

- [1] Suport laborator ALPD
- [2] <http://www.cplusplus.com/reference/map/map/>
- [3] Hadoop MapReduce Cookbook : Srinath Perera, Thilina Gunarathne