Tema de casa APD – map reduce

Iorga Elena, grupa 1411B

Cuprins

1. Prezentarea problemei

Colecțiile sunt uneori atât de mari încât nu putem face construirea indexilor eficient pe o singură mașină, acesta fiind cazul și pentru World Wide Web care folosește indexarea distribuită pentru a face asta. Rezultatul procesului este distribuit pe mai multe mașini, fie după termen fie după document.

Astfel, trebuie găsită o metodă care să proceseze datele de intrare (care pot fi foarte mari) și a cărei performanță în timp să fie cât mai bună. O astfel de soluție este reprezentată de folosirea procesării distribuite care, deși este complexă, este necesară. Dacă unul dintre modulele folosite eșuează, atunci programul în sine va continua să lucreze, reprezentând încă un mare avantaj al acestei soluții. În această lucrare, vom rezolva problema indexării după termen.

2. Modul de rezolvare

Pentru a rezolva problema descrisă mai sus este o aplicație a algoritmul MapReduce. Acesta se bazează pe faptul că distribuim munca în bucăți mai micuțe pentru a fi mai ușor de rezolvat într-un timp mai scurt.

Vom avea o etapă de mapare unde un nod cu rolul de coordonator împarte problema „originală” în sub-probleme și le distribuie către workeri pentru procesare. Apoi, una de reducere în care rolul de coordonator colectează soluțiile sub-problemelor (ce au facut workerii) și le combină pentru a obține rezultatul final al procesării dorite.

Astfel, o figură care ar putea reprezenta vizual modul de rezolvare al problemei este:

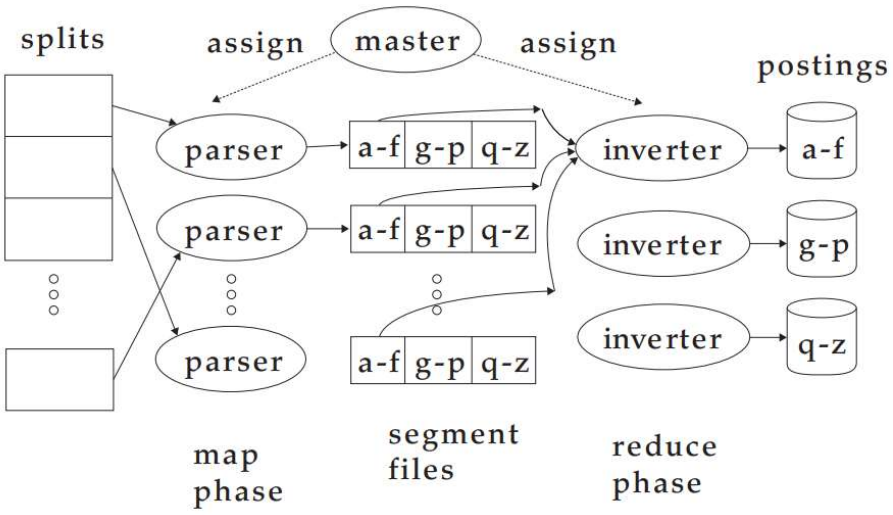

Fig. 2.1. Reprezentarea grafică a algoritmului

Pentru implementare, s-a folosit limbajul C++ cu MPI pentru a face procesarea paralelă. Programul a fost rulat pe un sistem de operare Windows, folosind Visual Studio Code cu extensia WSL pentru Debian.

3. Prezentarea implementării

În primul rând, s-a adăugat setul de date într-un director numit „date” în care sunt toate fișierele de intrare, 25 la număr.

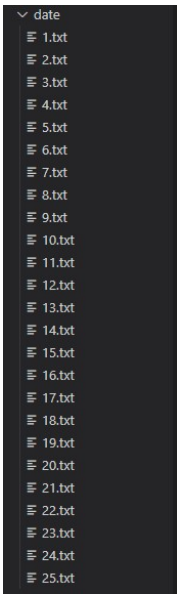

Fig. 3.1 Fișierul cu date de intrare

Apoi, din acest director au fost extrase care sunt numele tuturor fișierelor și adăugate într-un vector folosind această funcție:

```
std::vector<std::string> getFiles()
{
    std::vector<std::string> files;
    struct dirent *entry;
    DIR *dir = opendir("./date");

    if (dir == NULL)
    {
        return files;
    }
    string str1 = ".";
    string str2 = "..";
    while ((entry = readdir(dir)) != NULL)
    {
        cout << entry->d_name << endl;
        if (entry->d_name != str1 && entry->d_name != str2)
        {
            files.push_back(entry->d_name);
        }
    }
    closedir(dir);

    return files;
}
```

Fig. 3.2 Funcția folosită pentru a extrage fișierele

Folosind aceste fișiere, procesul cu rank-ul 0 care în cazul nostru va prelua rolul de master, va delega câte un fișier celoralte procese (workerilor) pentru a putea face maparea.

```
if (rank == 0) // master
{

    int next_task = 0;
    while (next_task < 25)
    {
        int j = next_task + 1;
        const char *message = fls[next_task].c_str();
        MPI_Send(message, 9, MPI_CHAR, j, 1, MPI_COMM_WORLD);
        next_task++;
    }
    MPI_Recv(&mesaj, 1, MPI_INT, 25, 0, MPI_COMM_WORLD, &status);
    if (mesaj == 1)
    {
        map<std::string, vector<pair<string, int>>, less<string>> finalList;
        finalList = read_word_by_word_final("beforeReduce.txt", finalList);
        string stx = "\x02";
        string etx = "\x03";
        string eoh = "\x01";
        for (auto it = finalList.begin(); it != finalList.end(); it++)
        {
            ofstream outfile;
            outfile.open("final.txt", ios_base::app);
            outfile << "<" << it->first << ",{";
            for (auto pair : it->second)
            {
                if (pair.first != stx && pair.first != etx && pair.first != eoh)
                    outfile << pair.first << ":" << pair.second << ",";
            }
            outfile << "}" << std::endl;
        }
    }
}
```

Fig. 3.3 Procesul master

Masterul ia fișierul din vector unde este sub forma de string și îl parsează în char* pentru a îl putea trimite la alte procese. După, așteaptă mesajul cum că un proces a terminat de scris în fișier (sunt șanse ca și celelalte să fi terminat), pentru a aștepta să nu existe posibilitatea mare de a parsa fișierul înainte ca ceilalți să termine de scris în el și după se apucă de etapa de reducere.

Aceasta constă în a lua datele procesate de workeri care au fost scrise în fișierul *beforeReduce.txt* și a le adăuga într-un map format dintr-un string (cuvântul din fișier) și un vector de perechi care conține fișierul în care se află cuvântul și numărul de apariții. Pentru a prelua datele se folosește funcția read_word_by_word_final care procesează fișierul și returnează map-ul.

```
map<std::string, vector<pair<string, int>>> read_word_by_word_final(string filename, map<std::string, vector<pair<string, int>>, less<string>> finalList)
{
    fstream file;
    string word;
    file.open(filename.c_str());
    while (file >> word)
    {
        string firstWord = word;
        string space = "  ";
        file >> word;
        pair<string, int> pair;
        pair.first = word;
        file >> word;
        // cout << "incercare " << word << endl;
        pair.second = stoi(word);

        auto it = finalList.find(firstWord);
        if (it != finalList.end())
        {
            auto list = it->second;
            auto it_pair = std::find_if(list.begin(), list.end(), [&](const std::pair<std::string, int> &p)
                                        { return p.first != pair.first; });
            if (it_pair != list.end())
            {
                (*it_pair).second += 1;
            }
            else
            {
                vector<std::pair<string, int>> newValue;
                std::pair<string, int> par;
                par.first = pair.second;
                par.second = stoi(pair.first);
                newValue.push_back(par);
                finalList[firstWord] = newValue;
            }
        }
        else
        {
            vector<std::pair<string, int>> newValue;
            std::pair<string, int> par;
            par.first = pair.second;
            par.second = stoi(pair.first);
            newValue.push_back(par);
            finalList[firstWord] = newValue;
        }
        finalList[firstWord].push_back(pair);
    }
    file.close();
    return finalList;
}
```

Fig. 3.4 Funcția de citire din fișier a rezultatelor de la worker

Se extrag datele din fișier care știm că sunt pune în ordinea de care avem nevoie de workeri, iar pe măsura ce se iau elementele de tip <cuvant, {fisier, numarAparitii}> se verifică dacă un cuvânt a fost adăugat deja în noua listă. Dacă da, dacă deja a fost adăugat și fișierul respective, se incrementează numărul de apariții. Dacă a fost adăugat cuvântul dar nu din fișierul curent, se adaugă și fișierul curent cu numărul de apariții la termen. Dacă nu a fost adăugat deloc până atunci, se adaugă simplu în listă și cuvântul cu fișierul și numărul de apariții.

Astfel, de obține un fișier final denumit *final.txt* unde vor fi datele finale sub forma <term, {docId1:count1, docId2:count2, …}>.

Pentru procesele care sunt diferite de master, acestea este asignat rolul de mapare.

```
else
{
    char file_name[9];
    MPI_Recv(&file_name, 9, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &status);
    cout << "Process " << rank << " received file name " << file_name << endl;
    map<std::string, int> numbers;

    numbers = read_word_by_word(file_name, numbers);

    pair<string, map<std::string, int, less<string>>> pair1;
    pair1.first = file_name;
    pair1.second = numbers;

    list<pair<string, pair<string, int>>> firstList;
    for (auto it = numbers.begin(); it != numbers.end(); it++)
    {
        pair<string, pair<string, int>> newElement;
        newElement.first = pair1.first;
        newElement.second.first = it->first;
        newElement.second.second = it->second;
        firstList.push_back(newElement);
    }
    list<pair<string, pair<string, int>>> secondList;
    for (auto it = numbers.begin(); it != numbers.end(); it++)
    {
        pair<string, pair<string, int>> newElement;
        newElement.first = it->first;
        newElement.second.first = pair1.first;
        newElement.second.second = it->second;
        secondList.push_back(newElement);
        ofstream outfile;
        outfile.open("beforeReduce.txt", ios_base::app);
        string space = " ";
        string nothing = "";
        if (it->first != space && it->first != nothing && pair1.first != space)
            outfile << it->first << " " << pair1.first << " " << it->second << " ";
    }
    int mesaj = 1;
    MPI_Send(&mesaj, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
```

Fig. 3.5 Codul pentru workeri

Fiecare worker practic citește din fișierul de intrare care i-a fost asignat cu funcția read_word_by_word :

```
map<std::string, int, less<string>> read_word_by_word(string filename, map<std::string, int, less<string>> numbers)
{
    fstream file;
    string word;
    string path = "./date/";
    cout << path + filename.c_str();
    file.open(path + filename.c_str());
    while (file >> word)
    {
        word.erase(remove(word.begin(), word.end(), '.'), word.end());
        word.erase(remove(word.begin(), word.end(), ','), word.end());
        word.erase(remove(word.begin(), word.end(), ')'), word.end());
        word.erase(remove(word.begin(), word.end(), '\''), word.end());
        word.erase(remove(word.begin(), word.end(), '"'), word.end());
        word.erase(remove(word.begin(), word.end(), ';'), word.end());
        transform(word.begin(), word.end(), word.begin(), ::tolower);
        if (numbers[word] == 0)
            numbers[word] = 1;
        else
        {
            numbers[word] = numbers[word] + 1;
        }
    }
    file.close();
    return numbers;
}
```

Fig. 3.6 Funcția de citire din fișierele inițiale

Astfel, se iau toate cuvintele din fișier și se numără. Pentru a avea un rezultat cât mai bun, cuvintele sunt luate fără anumite elemente de punctuație pentru a nu parsa de exemplu cuvânt apple ca fiind diferit de apple, . Se returnează un map de string și int care este ordinat după string.

Apoi, workerii iau și mapează cuvintele acestea specificând și fișierul din care au parsat datele, rezultând o pereche dintre fișier și map-ul de cuvinte cu aparițiile lor sub forma de <docId, {term1:count}>. Pe acesta trebuie să îl inverseze sub forma <term, {docId:count}> și face acestea reordonând datele într-o listă de perechi de forma cerută. Acestea sunt adăugate într-un fișier numit *beforeReduce.txt* pentru a putea fi folosite după. În acest fel, fișierul va avea date sub această formă:

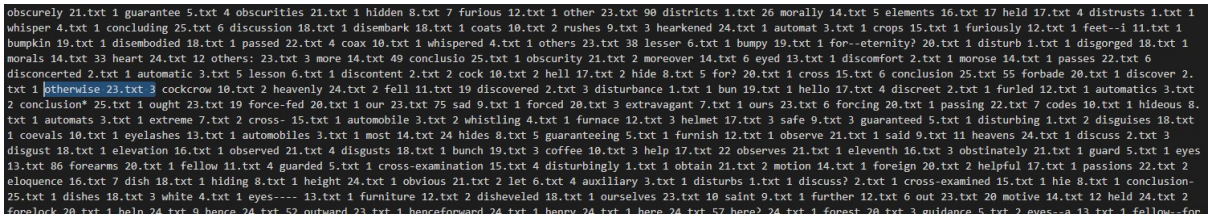

Fig. 3.7 Captură din fișierul beforeReduce.txt

În ordinea pe care o așteaptă coordonatorul : cuvânt, fișier, numărul de aparții.

În final, după ce master-ul face etapa de reducere și adaugă cuvintele, rezultă fișierul final.txt care va avea rezultatul programului:

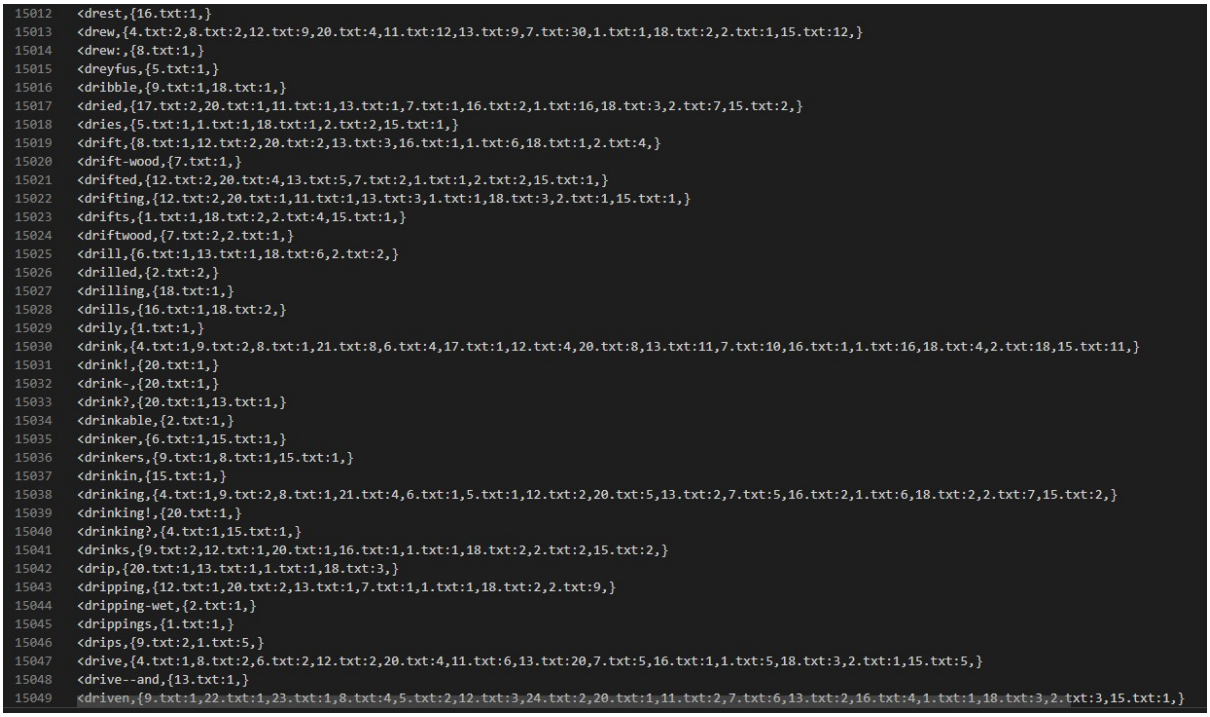

Fig. 3.8 Captură din fișierul final.txt

Se observă că datele sunt în ordine alfabetică întrucât s-au folosit map-uri ordonate după string-uri.

4. Concluzii

În concluzie, programul a reușit să facă ceea ce și-a propus folosind instrumentele precizate în modul de rezolvare. Acesta a rezultat un timp mult mai bun de rulare folosind procesele din MPI, un proces care deleagă acțiunile și returnează datele cerute și restul care le îndeplinesc, decât ar fi rezultat un program rulat secvențial

5. Bibliografie

[1] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England, Online c 2009 Cambridge UP edition, 2009.

[2] *Proiect – MapReduce,* disciplina Algoritmi paraleli și distribuiți

[3] ***, https://www.geeksforgeeks.org/map-vs-unordered_map-c/, ultima vizitare : 31/01/2023

[4] ***, https://linuxhint.com/map-sort-key-c/, ultima vizitare : 31/01/2023

[5] *Laborator nr. 2 – Introducere în MPI(1),* disciplina Algoritmi paraleli și distribuiți

[6] Michael Kleber. *The MapReduce paradigm.* https://sites.google.com/site/mriap2008/lectures, January 2008, ultima vizitare : 31/01/2023