Universitatea Tehnică „Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare

Domeniul: Tehnologia Informației

Algoritmul Map-Reduce

Proiect la disciplina

Algoritmi paraleli și distribuiți

Student: Enachi Vasile

Coordonator: Alexandru Arhip

Anul: 4

Grupa: 1409B

CUPRINS

Capitolul 1. Enunțarea problemei

În cadrul oricărui sistem de regăsire a informaţiilor, colecţia de date ţintă este reorganizată (sau re-modelată) pentru a optimiza funcţia de căutare. Un exemplu în acest sens este dat chiar de motoarele de căutare a informaţiilor pe Web: colecţia de documente este stocată sub forma unui ***index invers***.

Paşii implicaţi în construirea unui astfel de ***index invers*** sunt următorii:

1. fiecare document din cadrul colecţiei ţintă (identificat printr-un docID) va fi parsat şi spart în cuvinte unice (sau termeni unici); se obţine în finalul acestui pas o listă de forma $< docID_x , \{term_1 : count_1, term_2 : count_2 , \ldots , term_n : count_n\} >$ (***index direct*** – $count_k$ înseamnă numărul de apariţii al termenului k);
2. fiecare listă obţinută în pasul anterior este spartă în perechi de forma: $< docID_x , \{term_k : count_k\} >$; pentru fiecare astfel de pereche, se realizează o inversare de valori, astfel încât să obţinem: $< term_k , \{docID_x : count_k\} >$;
3. perechile obţinute în pasul anterior sunt sortate după $term_k$ (cheie primară), $docID_x$ (cheie secundară);
4. pentru fiecare $term_k$ se reunesc $< term_k , \{docID_x : count_k\} >$, astfel încât să obţinem: $< term_k , \{docID_{k1} : count_{k1}, docID_{k2} : count_{k2}, \ldots , docID_{km} : count_{km}\} >$ (***indexul invers***).

Tema de casă constă în implemenatrea unei soluţii MPI de tip MapReduce pentru problema construirii unui index invers pentru o colecţie de documente text.

Aplicaţia de test va primi ca parametrii de intrare numele unui director ce conţine fişiere text (cu extensia ”.txt”) şi un nume de director pentru stocarea datelor de ieşire şi va genera pe post de răspuns un set de fişiere text ce conţin indexul invers corespunzător colecţiei de documente de intrare.

Capitolul 2. Descrierea generală a problemei

MapReduce este un model de programare și o implementare asociată pentru procesarea și generarea seturilor de date mari cu un algoritm paralel, distribuit pe un cluster. Termenul „MapReduce” referă, în prezent, un tipar de dezvoltare a aplicaţiilor paralele / distribuite ce procesează volume mari de date. În general, se consideră că acest model implică existenţa unui nod de procesare cu rol de coordonator (sau master sau iniţiator) şi mai multe noduri de procesare cu rol de worker.

Modelul MapReduce conține 2 etape:

- Etapa de **mapare** – preia un set de date şi îl converteşte într-un alt set de date, unde elementele individuale sunt „sparte” în tuple (adică perechi cheie / valoare).
  - ➔ nodul cu rol de coordonator împarte problema „originală” în sub probleme şi le distribuie către *workeri* pentru procesare.
  - ➔ trebuie reţinut faptul că această divizare a problemei de lucru (a datelor de procesat) se realizează într-o manieră similară divide-et-impera – în unele cazuri nodurile *worker* pot divide la rândul lor sub-problema primită şi pot trimite aceste subdiviziuni către alţi ; rezultă, în acest caz o arhitectură arborescentă;
  - ➔ divizarea caracteristică acestei etape nu trebuie să coreleze efectiv dimensiunea datelor de intrare cu numărul de *workeri* din sistem; un *worker* poate primi mai multe sub-probleme de rezolvat;
- Etapa de **reducere** – preia ieşirea de la etapa de mapare ca fiind datele de intrare şi combină aceste tuple, rezultând un alt set de tuple, dar de dimensiuni mai mici.
  - ➔ nodul cu rol de coordonator (sau un set de noduri cu rol de *woker* „desemnat” de coordonator) colectează soluţiile sub-problemelor şi le combină pentru a obţine rezultatul final al procesării dorite.

Precum indică şi numele algoritmului, etapa de reducere este mereu efectuată după etapa de mapare.

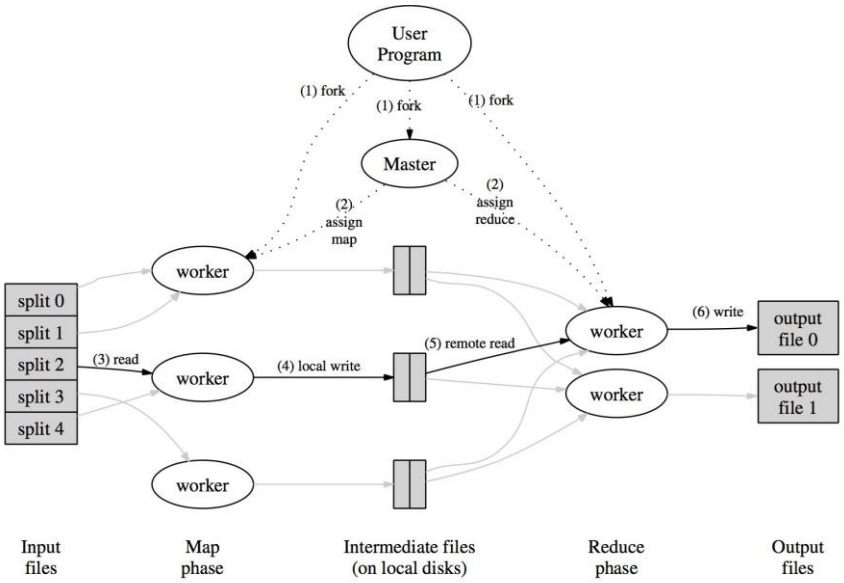


Figura 1: Paradigma MapReduce

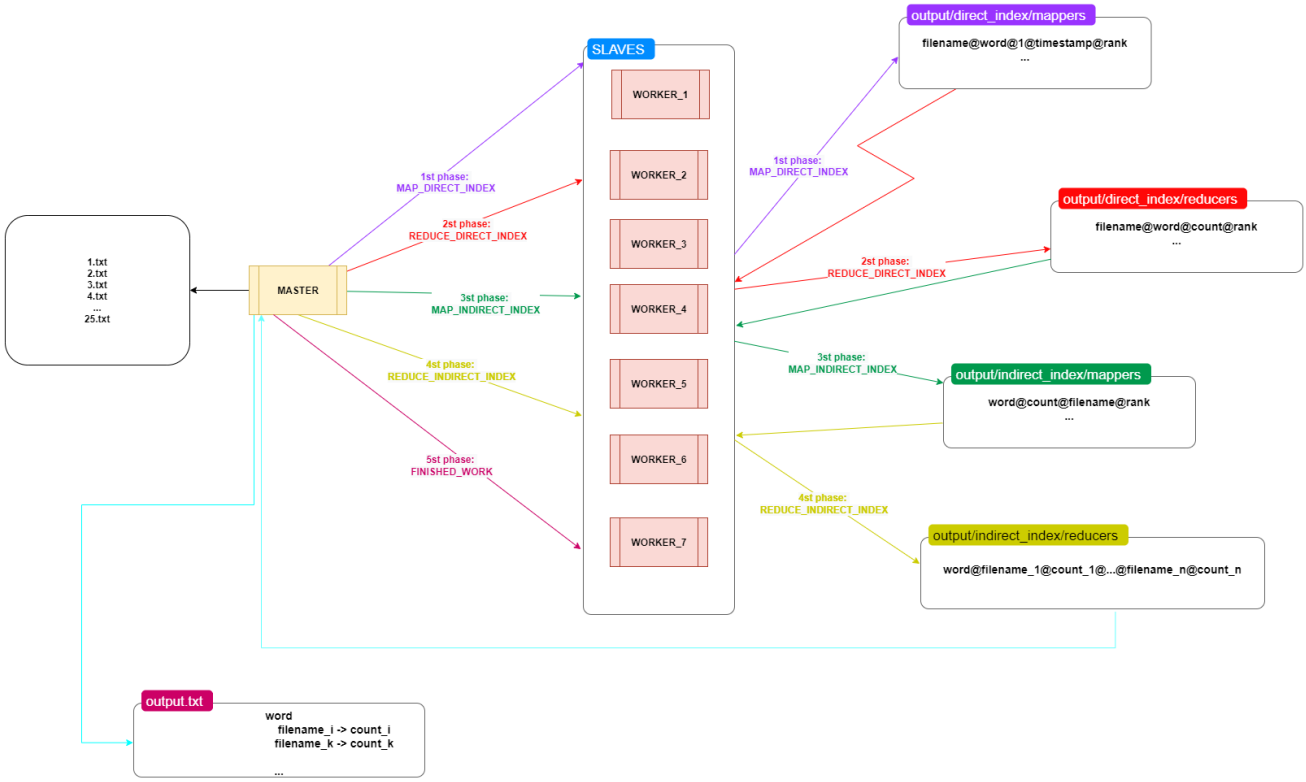
Capitolul 3. Modalitatea de rezolvare

Cerința problemei constă în implementarea unei aplicații MPI care, folosind paradigma MapReduce să genereze indexul invers al cuvintelor din documentele prezente într-un fișier de intrare.

Aplicația este rulată cu 8 procese. În rularea acesteia, un proces prestabilit este marcat ca fiind masterul sau coordonatorul. Acesta primește directorul de input și asignează celorlalte procese câte un task.

În cadrul acestei aplicații avem 2 faze:

- MapReduce pentru generarea indexului direct
- MapReduce pentru generarea indexului indirect



3.1 Procesul master

Inițial, masterul verifică că programul a fost lansat cu parametrii necesari. Ulterior, acesta primește directorul de input și își extrage toate denumirile de fișiere de acolo.

Ulterior, masterul emite către workeri un task prin intermediul unui tag și un fișier pentru prelucrat. Tag-urile folosite de master pentru a asigna un task workerilor sunt:

- **MAP_DIRECT_INDEX_PHASE**
- **REDUCE_DIRECT_INDEX_PHASE**
- **MAP_INDIRECT_INDEX_PHASE**
- **REDUCE_INDIRECT_INDEX_PHASE**

```
function master_emit_task(filenames, phase_tag)
    active_workers(i) = false, for i=1..workers_size
    for i = 1 to filenames.count
        worker_index = get_inactive_worker(active_workers)
        if worker_index = -1
            worker_index = await a worker_to_finish_a_job()
        Emit(filenames(i), worker_index, phase_tag)
        active_workers(worker_index) = true
    while any(active_workers(i) = true, i = 1 .. workers_size)
        worker_index = await a worker_to_finish_a_job()
        active_workers(worker_index) = false
```

În cadrul unei etape, masterul își construiește un vector *active_workers* în care salvează starea unui worker (ocupat sau nu). Fiecare fișier sau cuvânt de prelucrat este emis unui worker liber. Dacă toți workerii sunt ocupați, atunci masterul așteaptă ca unul să se elibereze pentru a-i asigna acestuia task-ul, după ideea „primul venit, primul servit”.

La final, masterul așteaptă ca toți workerii să emită tag-ul de **FINISHED_WORK**, pentru a începe etapa următoare.

3.2 Procesele workeri

Fiecare din workerii disponibili intră într-o buclă în care așteaptă comenzi de la procesul Master. În momentul în care acesta a terminat taskul, notifică Masterul trimițând un mesaj cu tagul **FINISHED_WORK**.

La primirea unui mesaj de la master cu tag-ul **FINISHED_WORK**, workerii ies din buclă.

```
function worker_receive_task()
    buffer, tag_phase = await_command_from_master()
    while tag_phase !=  FINISHED_WORK
        switch(tag_phase)
            case MAP_DIRECT_INDEX_PHASE:
                mapper_direct_index_phase(initial_directory, buffer, rank);
            case REDUCE_DIRECT_INDEX_PHASE:
                reduce_direct_index_phase(buffer, rank);
            case MAP_INDIRECT_INDEX_PHASE:
                mapper_indirect_index_phase(buffer, rank);
            case REDUCE_INDIRECT_INDEX_PHASE:
                reduce_indirect_index_phase(buffer);
        Emit_To_Master_That_Task_Is_Done();
        buffer, tag_phase = await_command_from_master()
```

3.2.1 Map pentru generarea indexului direct

În cadrul acestei faze mapperul împarte textul în linii și fiecare linie împarte la rândul său în cuvinte pe care le emite, creând fișiere pe disk de forma filename@word@count@timestamp@rank. Ca separatori de creare a unui cuvânt s-a orice caracter care nu este literă.

Cuvintele sunt convertite la low-case. Ulterior este creat un folder cu denumirea fișierului curent procesat, apoi sunt emise. Folderul nou creat înlocuiește sortarea de la faza următoare.

```
function mapper_direct_index_phase(directory_name, filename, rank)
    lines = split_into_lines(directory_name, filename)
    for i = 1 to lines.count
        words = split_into_words(lines(i))
        for j = 1 to words.count
            Save_To_Storage_Direct_Index(filename, words(j), rank);
// filename@word@1@timestamp@rank
```

3.2.2 Reduce pentru generarea indexului direct

În cadrul acestei faze, reducerul primește un fișier de la master și acesta preia datele din folderul cu denumirea fișierului, generat de la mapperi la etapa precedentă.

Este creat un folder cu denumirea fișierului care este procesat și folosind un dicționar are loc reducerea cuvintelor repetitive (numărarea apariției acestora).

```
function reducer_direct_index_phase(filename, rank)
        files = get_files(filename)
        for i = 1 to files.count
                if files(i) in dict
                        dict(files(i)) += 1
                else
                        dict(files(i)) = 1
        for i = 1 to dict.count
                Save_To_Storage_Indirect_Index(filename, dict(i), rank);
// filename@word@count@rank
```

3.2.3 Map pentru generarea indexului indirect

În cadrul acestei faze, mapperul primește un fișier de la master și acesta preia datele din folderul cu denumirea fișierului, generat de reduceri la etapa precedentă.

Pentru fiecare cuvânt găsit, este creat un folder cu denumirea acestui cuvânt, pentru a preveni sortarea de la faza următoare. Aici are loc inversarea indexului, iar fișierul emis are forma word@filename@count@rank.

```
function mapper_indirect_index_phase(filename, rank)
        filenames = get_filenames(filename)
        for i = 1 to filenames.count
                new_filename = indirect_index(filenames(i))
                create_folder_if_not_exist_with_word(filenames(i).word)
                Save_To_Storage_Indirect_Index(new_filename);
// word@filename@count@rank
```

3.2.4 Map pentru generarea indexului indirect

În cadrul acestei faze, reducerul primește un cuvânt de la master și acesta preia datele din folderul cu denumirea cuvântului, generat de mapperi la etapa precedentă.

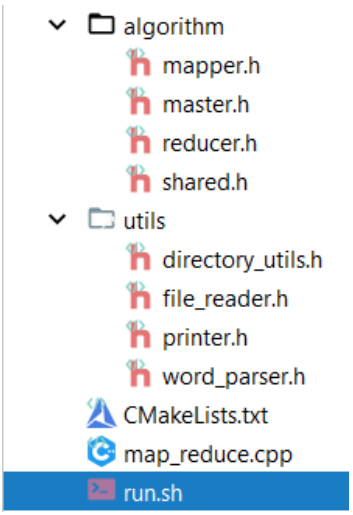
Preia toate fișierele ce conțin acel cuvânt și crează un nou fișier

```
function reducer_indirect_index_phase(word, rank)
        words = get_files(word)
        for i = 1 to files.count
                if files(i) not in dict
                        dict(files(i)) = init list()
                dict(files(i)).push(filename, count);
        for i = 1 to files.count
                Save_To_Storage_Indirect_Index(words(i), rank);
        // word@file_1@count_1@...@file_n@count_n@ @rank
```

Capitolul 4. Analiza soluției

4.1 Structura soluției

Structura soluției este formată din:



- **map_reduce.cpp** - fișierul cu funcția main
  folderul algorithm – conține fișierele header specifice cu funcționalitatea algoritmului (pentru procesul master, fazele de map și reduce) și un fișier header cu constante (shared.h)
- **folderul utils** – conține funcțiile necesare parsării fișierelor, scrierii în fișiere, creării fișierelor și folderelor.

Cuvintele sunt considerate doar cele ce conțin litere din ASCII. La găsirea unui cuvânt, acesta este convertit în cuvânt cu litere minuscule.

Ca separator ales la construirea fișierelor a fost ales caracterul ‚@'.

4.2 Fișierul map_reduce.cpp

1. Inițial are loc verificarea că programul a fost rulat cu numărul prestabilit de procese și că a fost specificat un director de intrare. (liniile 23-32)
2. Procesul master (liniile 35-71):
   - crează directoarele pentru faza de mapReduce a indexului direct
   - apelează consecutiv cele 4 faze din cadrul algoritmului: mapare a indexului direct, reducere a indexului direct, mapare a indexului indirect, reducere a indexului indirect.
3. Procesele worker (liniile 72-105):
   - Așteaptă un task de la master și în funcție de tag-ul primit:
     ➔ execută maparea pentru indexul direct
     ➔ execută reducerea pentru indexul direct
     ➔ execută maparea pentru indexul indirect
     ➔ execută reducerea pentru indexul indirect
     ➔ iese din bucla de așteptare a taskurilor

```cpp
//
// Created by enaki on 05.01.2021.
//

#include <map>
#include "utils/directory_utils.h"
#include "utils/file_reader.h"
#include "mpi.h"
#include "algorithm/shared.h"
#include "algorithm/master.h"
#include "algorithm/mapper.h"
#include "algorithm/reducer.h"


int main(int argc, char *argv[]) {
    int rank, size;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    // ********** INITIAL CHECK **********
    if (size != SIZE || argc != 3 || strcmp(argv[1], "-p") != 0){
        if (rank == MASTER){
            std::cout << "[ MASTER ] - Programul nu a fost rulat corespunzator." << std::endl;
            std::cout << "[ MASTER ] - Compilati cu " << SIZE << " procese." << std::endl;
            std::cout << "[ MASTER ] - Rulati executabilul astfel: ./map_reduce -p <input_path>" << std::endl;
            std::cout << "[ MASTER ] - Unde <input_path> reprezinta denumirea directorului dvs" << std::endl;
        }
        MPI_Finalize();
        return(0);
    }

    // ********** MASTER BRANCH
    if (rank == MASTER){
        auto filenames_ptr = directory_utils::get_file_names(argv[2]);
        std::cout << "\n[ MASTER ] - DIRECTORY INTRODUCED: <" << argv[2] << ">\n";
        std::cout << "\n[ MASTER ] - ***READ FILENAMES***\n";
        if (filenames_ptr->empty()){
            std::cout << "[ MASTER ] - Nu a fost gasit nici un fisier in directorul specificat\n";
        }
        printer::cli_printer_vector(*filenames_ptr);

        create_directories();
        create_directories_in_another_directory(TEMP_FOLDER_DIRECT_IDX_PHASE_MAPPERS, *filenames_ptr);
        create_directories_in_another_directory(TEMP_FOLDER_DIRECT_IDX_PHASE_REDUCERS, *filenames_ptr);

        // TRIMITE COD DE START PENTRU WORKERI SI INCEPE TOTUL
        printf("\n[ MASTER ] - PRIMA ETAPA DE MAPARE: INDEX DIRECT\n");
        master_index_phase(filenames_ptr, MAP_DIRECT_INDEX_PHASE);

        printf("\n[ MASTER ] - PRIMA ETAPA DE REDUCERE: INDEX DIRECT\n");
        master_index_phase(filenames_ptr, REDUCE_DIRECT_INDEX_PHASE);

        printf("\n[ MASTER ] - A DOUA ETAPA DE MAPARE: INDEX INDIRECT\n");
        master_index_phase(filenames_ptr, MAP_INDIRECT_INDEX_PHASE);

        printf("\n[ MASTER ] - A DOUA ETAPA DE REDUCERE: INDEX INDIRECT\n");
        auto words_ptr = directory_utils::get_file_names(TEMP_FOLDER_INDIRECT_IDX_PHASE_MAPPERS);
        master_index_phase(words_ptr, REDUCE_INDIRECT_INDEX_PHASE);

        master_collect("output.txt", ":)");

        // TRIMITE COD DE STOP PENTRU WORKERI
        for (int worker_idx = 0; worker_idx < WORKER_SIZE; ++worker_idx){
            MPI_Send(nullptr, 0, MPI_BYTE, worker_idx, FINISHED_WORK, MPI_COMM_WORLD);
        }
        delete words_ptr;
        delete filenames_ptr;

    } else {
```

```
    } else {
        // ********** WORKER BRANCH
        const int buffer_size = 256;
        char buffer[buffer_size]{0};

        MPI_Recv(&buffer, buffer_size, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

        while (status.MPI_TAG != FINISHED_WORK) {
            log(rank, status.MPI_SOURCE, status.MPI_TAG, buffer);

            switch (status.MPI_TAG) {
                case MAP_DIRECT_INDEX_PHASE:
                    mapper_direct_index_phase(argv[2], buffer, rank);
                    break;
                case REDUCE_DIRECT_INDEX_PHASE:
                    reduce_direct_index_phase(buffer, rank);
                    break;
                case MAP_INDIRECT_INDEX_PHASE:
                    mapper_indirect_index_phase(buffer, rank);
                    break;
                case REDUCE_INDIRECT_INDEX_PHASE:
                    reduce_indirect_index_phase(buffer);
                    break;
                default:
                    break;
            }

            // NOTIFICA MASTER CA AM TERMINAT CE AM AVUT DE FACUT
            MPI_Send(nullptr, 0, MPI_BYTE, MASTER, FINISHED_WORK, MPI_COMM_WORLD);

            // WAITING FOR COMMANDS FROM MY MASTER <3
            MPI_Recv(&buffer, 100, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        }

        log(rank, status.MPI_SOURCE, status.MPI_TAG);
    }
    MPI_Finalize();
    return(0);
}
```

Capitolul 5. Rezultatele obținute prin rularea programului

Programul a fost rulat pe 2 dispozitive diferite. Rezultatele rulării sunt reprezentate în tabelul ce urmează.

Număr de procese/Dispozitiv	I7-9750H (6 cores, 12 threads)	I7-7500U (2 cores, 4 threads)
4 procese (3 workeri)	16 s	37 s
8 procese (7 workeri)	**11 s**	40 s
16 procese (15 workeri)	15 s	55 s

Se observă că în unele cazuri, mărirea numărului de procese cu care este rulat programul, nu aduce întotdeauna performanță mai bună. De asemenea, codul poate fi optimizat dacă funcțiile din directorul „utils" ar fi integrate direct în codul workerilor și masterului (fără a crea structuri de date cu rezultatul și a le mai parcurge încă odată). Totuși, pentru a refolosi codul, s-a mers pe structura proiectului prezentată anterior.

Capitolul 6. Concluzii

Avantajele algoritmului MapReduce-ului constau în scalabilitatea acestuia și în toleranța la defecte. În cadrul proiectului am aplicat această paradigmă pe parsarea unui set de fișiere și ne-am convins de ușurința schimbării parametrilor algoritmilor (numărul de workeri, modalitatea de comunicare între procese, dimensiunea bufferelor), acest lucru reprezentând un factor major care determină alegerea acestuia în rezolvarea problemelor ce necesită prelucrări mari de date.

Capitolul 7 Bibliografie

- https://soft.vub.ac.be/~tvcutsem/talks/presentations/ErlangMapReduce.pdf
- http://www.ugr.es/~essir2013/slides/ESSIR_MapReduce_for_Indexing.pdf
- https://en.wikipedia.org/wiki/MapReduce
- https://timepasstechies.com/map-reduce-inverted-index-sample/
- Laborator ALPD
- https://www.talend.com/resources/what-is-mapreduce/