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

Algoritmi paraleli și distribuiți
- Map Reduce -

STUDENT,
Ursachi Gabriela - 1410B

An universitar 2022 - 2023

CUPRINS

1. Prezentarea soluției

1.1. Prezentare generala

MapReduce este o paradigmă de programare folosită pentru procesarea unor cantități mari de date în mod paralel și distribuit pe un cluster.[1] Printre domeniile de aplicabilitate ale acestui tipar de dezvoltare se numără și regăsirea informațiilor pe Web, acolo unde colecția de date țintă este reorganizata si stocată sub forma unui index invers, în vederea optimizării funcției de căutare.

Abordarea menționată este adoptată și în cadrul acestui proiect: se are în vedere implementarea unei soluții MPI de tip MapReduce pentru problema construirii unui index invers pentru o colecție de documente text. Fiecare astfel de document are propriul identificator (docID) si va trebui parsat, apoi spart în cuvinte unice și procesat, astfel incat, in final, sa se obțină un rezultat de forma:

$$\langle termen_k, \{docId_{k1} : aparitii_{k1}, docId_{k2} : aparitii_{k2}, \ldots, docId_{kn} : aparitii_{kn}\}\rangle$$

Soluția propusă constă în existența unui nod cu rol de coordonator, care distribuie task-uri și noduri cu rol de workeri, care procesează aceste task-uri în mod paralel. Procesul este divizat 2 etape: etapa de **mapare**, respectiv **reducere.**

1.2. Etapa de mapare

Fiecare worker primește, la un moment dat, câte 2 documente spre procesare și extrage termenii distincti, precum și numărul de apariții al acestora, separat, pentru fiecare document. Rezultatele vor fi scrise într-un director intermediar, în subdirectoare proprii fiecărui worker:

- **input:** identificatorii celor 2 documente, sub forma unei liste:

$$[doc_1, doc_2]$$

- **output:** perechi de tip <cheie, valoare> pentru fiecare termen unic

$$\langle termen_k, \{docId_x : count_k\}\rangle$$

1.3. Etapa de reducere

Fiecare worker primește de la coordonator câte un set de cuvinte (chei), pe care le cauta in directoarele intermediare rezultate din etapa de mapare, urmand ca apoi sa grupeze și să numere cate perechi sunt cu aceeași cheie:

- **input:** un set de termeni/chei

$$(termen_1, termen_2, ..., termen_n)$$

- **output:** indexul invers -> pentru fiecare termen va rezulta cate o pereche de forma:

$$\langle termen_k, \{docId_{k1}:apariții_{k1}, docId_{k2}:apariții_{k2}, \ldots, docId_{kn}:apariții_{kn}\}\rangle$$

2. Pseudocodul algoritmilor implementați

2.1. Nod cu rol de coordonator

```
input inputPath, nrOfProcesses, outputPath

currentPhase = PREPROCESSING
filesToProcess = []
isFinished = False

While Not isFinished Do
   When(currentPhase):
         Case PREPROCESSING:
            read filePaths from input path
            filesToProcess = list of read filePaths
            groupsOfFiles = group filesToProcess in groups of two

            create intermediate directory

            currentPhase = MAP
            break;

         Case MAP:
            processed_groups = 0
            sent_groups_count = 0

            // trimite catre fiecare mapper primul set de date
            For workerRank in 0,nrOfProcesses-2 Do
                  send(groupsOfFiles[sent_groups_count], intermediatePath) to workerRank
            Endfor


            // asteapta confirmarea procesarii de la workeri pentru a le trimite
            // urmatorul set de date
            While processed_groups < groups_number Do
                  recv(FreeWorkerConfirmation) from worker W
```

```
                processed_groups ++

                // daca mai sunt seturi de date de procesat, trimit
                // unul worker-ului care tocmai a devenit liber
                If sent_groups_count < groups_number Then
                  send(groupsOfFiles[sent_groups_count], intermediatePath) to W
                  sent_groups_count ++
                Endif

            Endwhile

            currentPhase = INTERMEDIATE

        Case INTERMEDIATE:
            wordsToProcess = extract distinct words from mappers output directory
            sort wordsToProcess ASC
            groupsOfWords = group wordsToProcess in groups of 50

            currentPhase = REDUCE
            break;

        Case REDUCE:
            processed_groups = 0
            sent_groups_count = 0

            // trimite catre fiecare reducer primul set de cuvinte
            For workerRank in 0,nrOfProcesses-2 Do
                  send(groupsOfWords[sent_groups_count], outputPath) to workerRank
            Endfor


            // asteapta confirmarea procesarii de la workeri pentru a le trimite
            // urmatorul set de cuvinte
            While processed_groups < groups_number Do
                  recv(FreeWorkerConfirmation) from worker W
                  processed_groups ++

                  // daca mai sunt seturi de date de procesat, trimit
                  // unul worker-ului care tocmai a devenit liber
                  If sent_groups_count < groups_number Then
                    send(groupsOfWords[sent_groups_count], outputPath) to W
                    sent_groups_count ++
                  Endif

            Endwhile

            currentPhase = FINISH
            break;

        Case FINISH:
            // trimite fiecarui worker un mesaj de oprire executie
            For workerRank in 0,nrOfProcesses-2 Do
                  send(finishExecution) to workerRank
            Endfor
            isFinished = True
            break;
Endwhile
```

2.2. Nod cu rol de worker

```
isFinished = False

While Not isFinished Do
   recv(data,tag) from coordinator

   When(tag):
     Case FINISH:
          isFinished = True
          break;

     Case MAP_OPERATION:
          files_to_process,output_path = data
          words = {}

          For file in files_to_process Do
                   lines = file.readlines()
                   For Each line in lines Do
                       // se sterg cuvintele nesemnificative (cifre/semne de punctuatie/prepozitii/cuvinte
cu mai putin de 2 caractere)
                       filtered_line = remove_unnecessary_words(line)

                       For Each word in filtered_line Do
                            // daca a mai fost gasit cuvantul in acest fisier, doar incrementez counter-ul
                            If words[word].keys().contains(filename) Then
                              words[word][filename] += 1
                            Else: // daca e prima aparitie a cuvantului in fisier, initializez counter-ul cu 1
                              words[word][filename] = 1
                            Endif
                      Endfor
                   Endfor
          Endfor

          For word in words Do
                   create file with name of the word in the intermediate_directory
                   write to file all the docIds where the word appears and number of occurrences
          Endfor

          send to coord FreeWorkerConfirmation
          break

   Case REDUCE_OPERATION:
          words_to_process,input_path,output_path, max_index_of_intermediate_directory = data

          For word in words_to_process Do
                   word_dict = {}

                   For Each intermediate_directory Do
                   If exists file with name of the word Then
                            read all docIds where word appears
                            word_dict[doc_id] = word occurrence
                   Endif
                   Endfor
                   lines = file.readlines()

                   open output file
                   write to file all occurrences of current word from all intermediate directories
          Endfor

          send to coord FreeWorkerConfirmation
          break

Endwhile
```

3. Explicații asupra implementării și funcționării soluției

3.1. Implementare

În implementarea soluției se urmăresc, in mare masura, etapele prezentate în [3], referitoare la paradigma MapReduce, astfel:

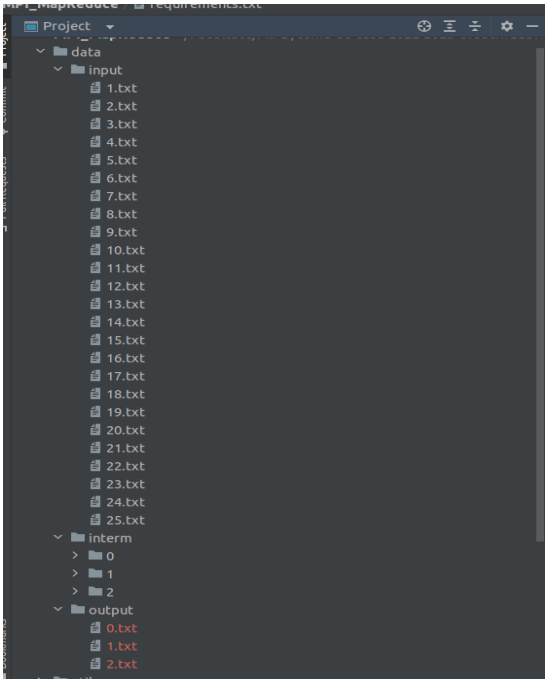
1. Initial, coordonatorul se afla în faza de **preprocesare**: pregateste si grupează fișierele de intrare în partiții de cate doua, în vederea optimizării pasului de mapare.
2. Urmeaza faza de **mapare**, în care se trimite fiecărui worker liber câte o partitie anterior creata. Atunci cand nu mai sunt workeri liberi, următoarea partiție se va trimite primului worker care va trimite un mesaj prin care confirma finalizarea task-ului precedent.
3. Cand s-au procesat toate fișierele de intrare de catre mapperi, coordonatorul intra intr-o faza **intermediara**, în care initializeaza o lista cu toate cuvintele distincte rezultate în faza de mapare. Sortează alfabetic aceste cuvinte și le grupează în partiții de cate 50, pentru a optimiza procesul de reducere.
4. Urmează faza de **reducere**, in care, similar pasului de mapare, se trimite fiecărui worker liber câte o partitie anterior creata si se așteaptă finalizarea procesării unei partiții de termeni de către un reducer pentru a-i putea trimite alta.
5. Atunci cand nu mai sunt termeni pentru reducere, se ajunge in ultima faza, cea de **finish.** In acest moment, coordonatorul va trimite fiecărui worker cate un mesaj prin care îi înștiințează că trebuie să-și încheie execuția. Coordonatorul, la randul sau, isi va incheia executia.

3.2. Structura de fisiere

data/input/ -> documentele initiale

data/interm/ -> fiecare mapper are cate un subdirector propriu în care își va crea fișiere specifice fiecărui cuvânt identificat din documentele procesate

data/output -> fiecare reducer are cate un fișier propriu în care va scrie toate cuvintele analizate, împreuna cu numărul de apariții al acestora în cadrul tuturor documentelor

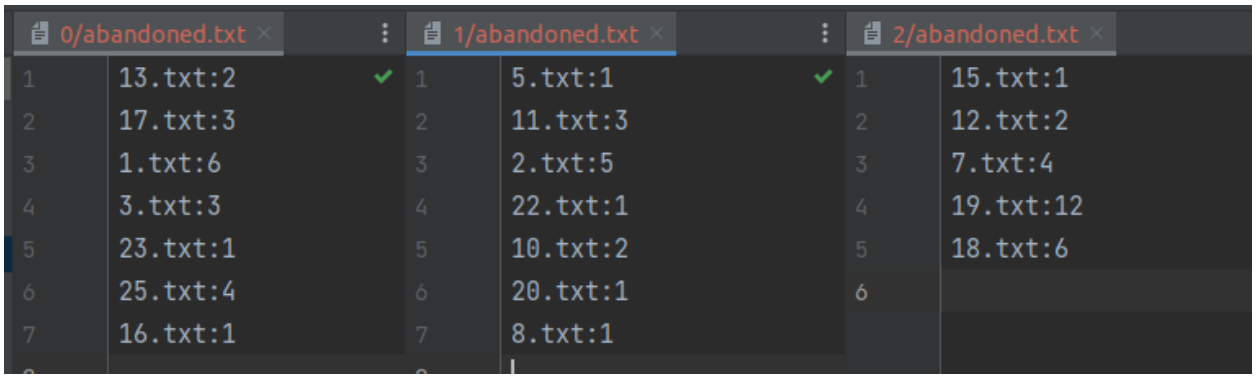


3.3. Rezultate execuție program

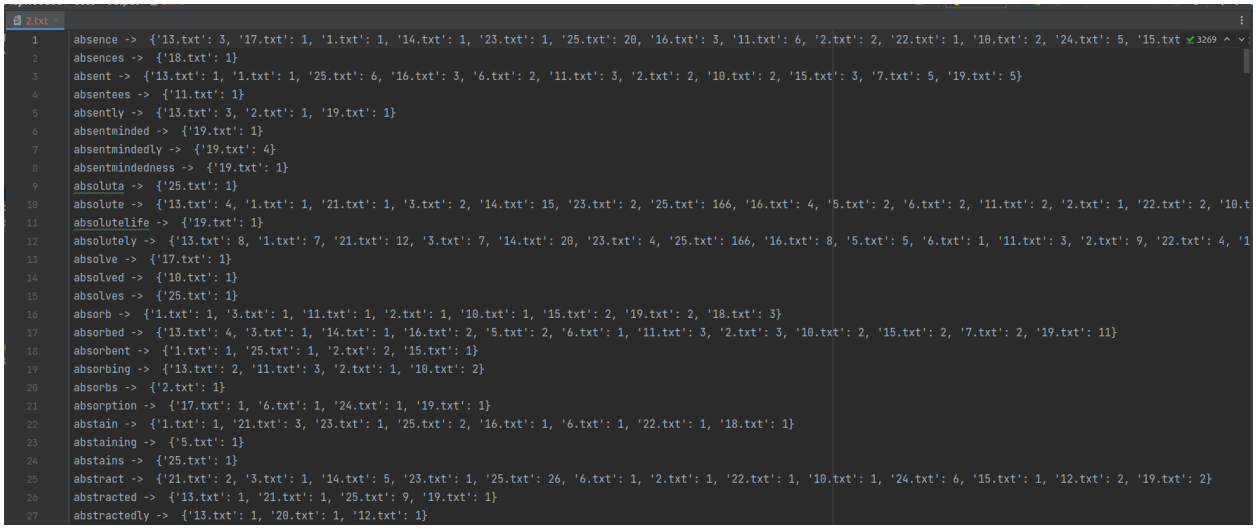
Pentru rezultatele prezentate, programul a fost rulat cu un număr de 4 procese:

- Un coordonator
- Trei workeri

3.3.1. Rezultat produs în urma fazei de mapare



3.3.2. Rezultat produs în urma fazei de reducere



3.3.3. Timpi de execuție pentru diferite valori ale numărului de procese

	4 processes	3 processes
Total execution time	14.791887760162354	16.02229928970337
Execution time for MAP	6.61927604675293	8.234440326690674
Execution time for REDUCE	7.727142810821533	7.537741184234619

4. Bibliografie

[1] Ralf L̈ammel, *Google's MapReduce programming model*, July 2007

[2] 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

[3] Michael Kleber. The MapReduce paradigm, January 2008

[4] https://mpi4py.readthedocs.io/en/stable/tutorial.html

[5] https://www.spiceworks.com/tech/big-data/articles/what-is-map-reduce/