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

Algoritmi Paraleli și Distribuiți

Implementarea unei soluții de tip MapReduce

Bărbuță Delia Elena, 1409B

Cuprins

1. Enunțarea temei

În cadrul oricărui sistem de regăsire a informațiilor, colecția de date țintă este reorganizată (sau re-modelată) pentru a optimiza funcția de căutare. Un exemplu în acest sens este dat chiar de motoarele de căutare a informațiilor pe Web: colecția de documente este stocată sub forma unui index invers. Pașii implicați în construirea unui
astfel de **index invers** sunt următorii:

1. fiecare document din cadrul colecției țintă (identificat printr-un docID) va fi parsat și spart în cuvinte unice (sau termeni unici); se obține în finalul acestui pas o listă de forma <*docIDx*,{*termk* : *count*1, *term*2: *count*2, ...,*termn*:*countn*}> (**index direct** – *countk* înseamnă numărul de apariții al termenului k);

2. fiecare listă obținută în pasul anterior este spartă în perechi de forma: <*docIDx*,{ *termk*:*countk* }>; pentru fiecare astfel de pereche, se realizează o inversare de valori astfel încât să obținem: <*termk* ,{*docIDx* : *countk* }>;

3. perechile obținute în pasul anterior sunt sortate după *termk* (cheie primă), *docIDx*(cheie secundară);

4. pentru fiecare *termk* se reunesc <*termk* ,{*docIDx* : *countk* }> astfel încât să obținem: <*termk*,{*docIDk*1 : *countk*1 ,*docIDk*2 :*countk*2 ,...,*docIDkm countkm* }> (**indexul invers**)

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 **parametrii 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.

2. Aspecte teoretice

MapReduce este o paradigma de programare, utilă în procesarea seturilor mari de date. Modelul constă în două funcții principale: Map și Reduce.

Funcția **Map** preia un set de date de intrare și aplică o transformare specifică fiecărui element din setul de date. Funcția **Reduce** preia rezultatul din funcția Map și îl agregă într-un set de date cu dimensiuni mai reduse. Avantajul principal al metodei MapReduce este că permite procesarea paralelă a seturilor de date mari, crescand eficienta comparativ cu procesarea

secvențială. În plus, paradigma oferă scalabilitate şi avantajul adaptarii la eşecuri, într-un mediu de calcul distribuit.

3. Etapele MapReduce

Etapa de mapare: aceasta etapa transforma setul de date în perechi (cheie, valoare)

1. Nodul cu rol de coordonator împarte problema „originala" în subprobleme şi le distribuie către workeri pentru procesare;
2. 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 subproblema primită şi pot trimite aceste subdiviziuni către alţi workeri, rezultând o arhitectură arborescentă;
3. 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 subprobleme de rezolvat;

Etapa de reducere: preia setul de date rezultat în urma etapei de mapare și manipulează datele respective rezultând un nou set de date tot sub forma unor perechi de tipul (cheie, valoare), doar că sub formă mai compactă și de dimensiuni mai mici.

1. Nodurile cu rol de worker determina soluțiile subproblemelor identificate în faza de mapare/selecție;
2. Nodul cu rol de coordonator (sau un set de noduri cu rol de woker „desemnat" de coordonator) colectează soluțiile subproblemelor și le combina pentru a obține rezultatul final al procesării dorite.

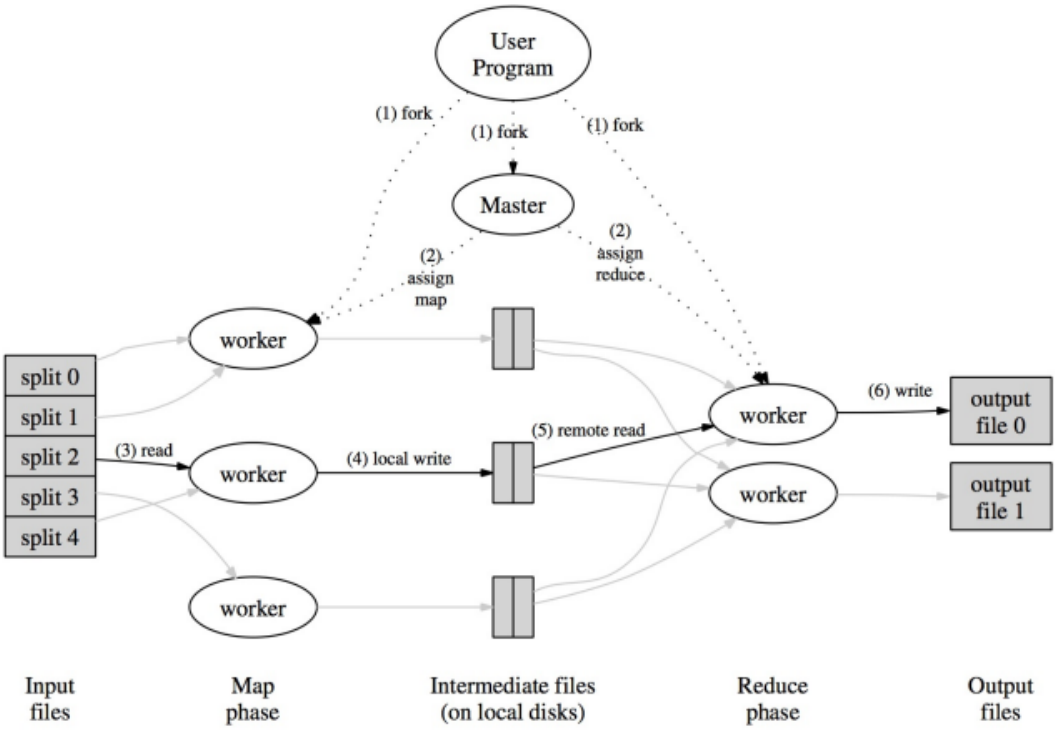


Figura 1: Paradigma MapReduce

4. Prezentarea soluției

Aplicația este dezvoltata folosind limbajul python si functionalitatile bibliotecii MPI4PY. Atat datele de intrare, cât și datele de ieșire sunt reprezentate de seturi de fișiere text. Fișierele de intrare conțin cuvinte, iar datele de ieșire contin indexul invers corespunzator colecției de documente de intrare.

Procesul principal, ce are rangul 0, are rolul de a distribui fișierele către celelalte procese worker. Fiecare proces worker prelucrează fișierul primit, eliminand semnele de punctuație, cifrele și spațiile multiple și creeaza, daca nu exista, un director cu rangul sau în interiorul directorului intermediary. Pentru fiecare cuvant regasit in fișierele pe care le-a procesat, este creat un fisier, in care se adauga, pentru fiecare aparitie, numele fișierului în care a fost regasit.

Următorul pas este cel de reducere. Procesul master asigneaza pentru fiecare worker o lista de litere, iar workerul respectiv procesează toate fișierele din directorul intermediary, care încep cu una din literele primite. Se construiește un dicționar de tipul (nume_fisier, nr_aparitii), iar în directorul output se creeaza cate un fișier pentru fiecare cuvant, în care vor fi scrise valorile din dicționar.

În ultima etapa, procesul cu rangul 1 se ocupă de crearea fișierului care conține lista cu toate cuvintele și aparițiile lor. Acest fișier este rezultatul final al procesării și poate fi utilizat, de exemplu, pentru a realiza o analiză a textului.

Implementarea a fost realizata folosind MPI (Message Passing Interface), un sistem de comunicare între procese care rulează pe același sistem, ce permite paralelizarea procesării și reducerea timpului de procesare, îmbunătățind astfel eficiența și performanța sistemului.

5. Implementare

- Distribuirea fișierelor de intrare către workeri:

```
while cnt != len(fileNames):
   for i in range(1, nrOfProcesses):
       comm.send(fileNames[cnt], dest=i, tag=FILE_NAME)
       cnt += 1
       if cnt == len(fileNames):
           break
```

- Procesarea textului și crearea fișierelor intermediare:

```
if tag == FILE_NAME:

   # citirea cuvintelor din fișier
   path = os.path.join(inputDirectory, fileToHandle)
   f = open(path, "r", errors="ignore")
   text = f.read()
   text = text.lower()
   f.close()

   # inlocuirea semnelor de punctuație și a cifrelor
   text = re.sub('\W+\s*', ' ', text).replace("\n", " ")
   text = text.translate(str.maketrans('', '', digits))
   text = re.sub(' +', ' ', text)

   processDirectory = str(myRank)

   # crearea unei liste de cuvinte
   text = text.split(" ")
   for word in text:

       # crearea unui director pentru fiecare worker
       path = os.path.join(intermediaryDirectory, processDirectory)
       exists = os.path.exists(path)
       if not exists:
           os.mkdir(path)
```

```
        if word != "":
            # crearea unui fișier pentru fiecare cuvant,
            # în care vor fi scrise numele fișierelor în care a apărut
            f = open(os.path.join(path, word), "a")
            f.write(msgFromMaster + " ")
            f.close()
```

- Distribuirea listei de litere către fiecare worker:

```
alphabetLetters = list(string.ascii_lowercase)

# literele sunt amestecate, pentru o impartire uniforma a task-urilor
random.shuffle(alphabetLetters)

chunkSize = int(len(alphabetLetters) / (nrOfProcesses - 1))
chunks = [alphabetLetters[x:x + chunkSize] for x in range(0, (nrOfProcesses -
1) * chunkSize, chunkSize)]
for i in range((nrOfProcesses - 1) * chunkSize, len(alphabetLetters)):
   chunks[len(chunks) - 1].append(alphabetLetters[i])

for i in range(1, nrOfProcesses):
   comm.send(chunks[i - 1], dest=i, tag=START_REDUCING)
```

- Etapa de reducere:

```
# fiecare fișier e procesat de workerul corespunzător
for path, subdirs, files in os.walk(intermediaryDirectory):
   for name in files:
       if name[0] in listOfLetters:

     # pentru fiecare cuvant, este construit dicționarul cu numărul de apariții
     # astfel incat, pentru fiecare fișier în care apare, exista o pereche
     # (nume_fisier, nr_de_aparitii)
     counts = dict()
     file = open(os.path.join(path, name), "r")
     text = file.read().split(" ")[:-1]
     file.close()

     for word in text:
         if word in counts:
             counts[word] += 1
         else:
             counts[word] = 1

     f = open(os.path.join(outputDirectory, name), "a")
     for pair in counts:
```

```
            f.write("(" + pair + ", " + str(counts[pair]) + ") ")
        f.close()
```

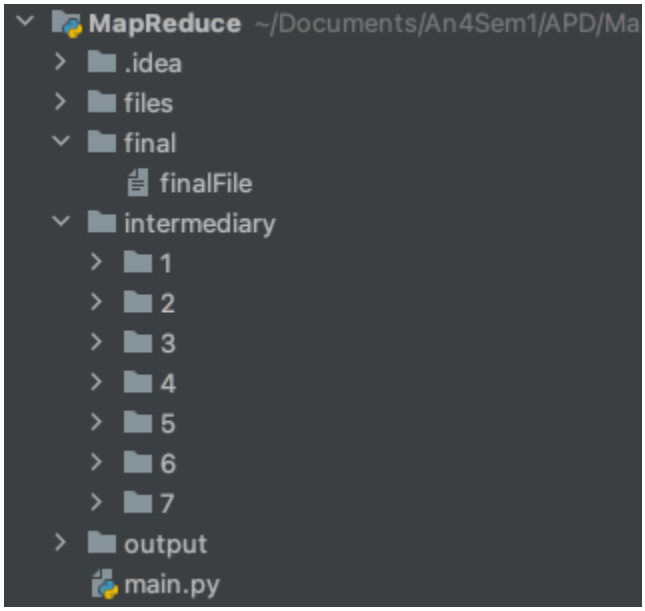
- Crearea fișierului final:

```
elif tag == CREATE_FINAL_FILE and myRank == 1:

    # toate fișierele din directorul output sunt preluate și sortate alfabetic
    outputFiles = os.listdir(outputDirectory)
    outputFiles.sort()

    for outputFile in outputFiles:
        if not outputFile.startswith('.'):
            path = os.path.join(outputDirectory, outputFile)
            f = open(path, "r")
            text = f.read()
            f.close()
            path = os.path.join(finalFileDirectory, "finalFile")
            finalFile = open(path, "a")
            finalFile.write(outputFile + " --- " + text + "\n")
            finalFile.close()
```

Dupa rulare, structura de directoare a proiectului arată în felul următor:



Fișierul rezultat în urma procesului de map-reduce:

```
1   a --- (10.txt, 2972) (7.txt, 1999) (22.txt, 241) (12.txt, 1083) (6.txt, 335) (
    23.txt, 242) (15.txt, 1933) (11.txt, 1110) (3.txt, 2096) (25.txt, 5108) (17.txt,
    655) (5.txt, 695) (20.txt, 907) (24.txt, 1477) (16.txt, 1127) (8.txt, 248) (
    1.txt, 2152) (18.txt, 1583) (14.txt, 773) (9.txt, 309) (2.txt, 2069) (19.txt,
    4392) (13.txt, 1434) (4.txt, 348) (21.txt, 418)
2   aa --- (2.txt, 1)
3   aback --- (10.txt, 1) (1.txt, 2) (2.txt, 2)
4   abaft --- (10.txt, 2) (6.txt, 1)
5   abandon --- (10.txt, 1) (7.txt, 4) (22.txt, 1) (23.txt, 1) (3.txt, 4) (25.txt, 8)
    (17.txt, 1) (20.txt, 1) (1.txt, 1) (2.txt, 3) (19.txt, 1) (13.txt, 1)
6   abandoned --- (10.txt, 2) (7.txt, 4) (22.txt, 1) (12.txt, 2) (23.txt, 1) (15.txt,
    1) (11.txt, 3) (3.txt, 3) (25.txt, 4) (17.txt, 3) (5.txt, 1) (20.txt, 1) (16.txt,
    1) (8.txt, 1) (1.txt, 6) (18.txt, 6) (2.txt, 5) (19.txt, 12) (13.txt, 2)
7   abandoning --- (25.txt, 2) (1.txt, 1) (18.txt, 1)
8   abandonment --- (17.txt, 1) (1.txt, 1) (18.txt, 1) (2.txt, 1)
9   abandons --- (25.txt, 4) (2.txt, 1)
10  abasement --- (10.txt, 1) (9.txt, 1)
11  abate --- (7.txt, 1) (1.txt, 1) (19.txt, 1)
12  abated --- (10.txt, 1) (19.txt, 1)
13  abating --- (10.txt, 2) (7.txt, 1)
14  abba --- (19.txt, 1)
15  abbe --- (25.txt, 1)
16  abbey --- (10.txt, 9)
17  abbie --- (3.txt, 2)
18  abbott --- (15.txt, 2) (24.txt, 1) (14.txt, 1)
19  abbottsville --- (15.txt, 5)
20  abbreviation --- (16.txt, 1)
21  abbreviations --- (19.txt, 2)
22  abc --- (3.txt, 2)
23  abcd --- (21.txt, 1)
24  abdalla --- (7.txt, 6)
25  abdera --- (16.txt, 1)
26  abdicating --- (7.txt, 1)
27  abdomen --- (3.txt, 1) (2.txt, 1)
28  abdominal --- (3.txt, 1)
29  abduct --- (18.txt, 1)
30  abduction --- (7.txt, 1) (19.txt, 1)
31  abed --- (10.txt, 1)
32  abel --- (8.txt, 1)
33  abernathy --- (17.txt, 2) (1.txt, 1) (2.txt, 2)
34  aberration --- (5.txt, 1) (16.txt, 1) (2.txt, 1)
35  aberrations --- (22.txt, 1) (23.txt, 1) (15.txt, 1)
36  abetting --- (3.txt, 2)
37  abhor --- (5.txt, 1)
38  abhorred --- (10.txt, 2)
39  abhorrence --- (14.txt, 1)
40  abhorrent --- (2.txt, 1)
```

6. Bibliografie

[1] NERSC Documentation What is MPI? https://docs.nersc.gov/development/programming-models/mpi/openmpi/

[2] IBM Corp. What is MapReduce? https://www.ibm.com/topics/mapreduce

[3] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation, pages 137–150, San Francisco, CA, 2004.
[4] Michael Kleber, The MapReduce paradigm https://sites.google.com/site/mriap2008/lectures
[5] Wikipedia. MapReduce. http://en.wikipedia.org/wiki/MapReduce