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

Algoritmul MapReduce

Proiect la disciplina
Algoritmi paraleli și distribuiți

Profesor îndrumător
Ș. l. Adrian Alexandrescu

Student
Scînteie Gabriel, 1409B

2022 - 2023

Cuprins

1. Enunțarea temei

În cadrul oricărui sistem de regăsire a informațiilor, colecția de date țintă este reorganizată pentru a optimiza funcția de căutare. Un exemplu în acest sens este dat chiar de către 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 proiectului 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. Noțiuni teoretice

2.1. Definiție

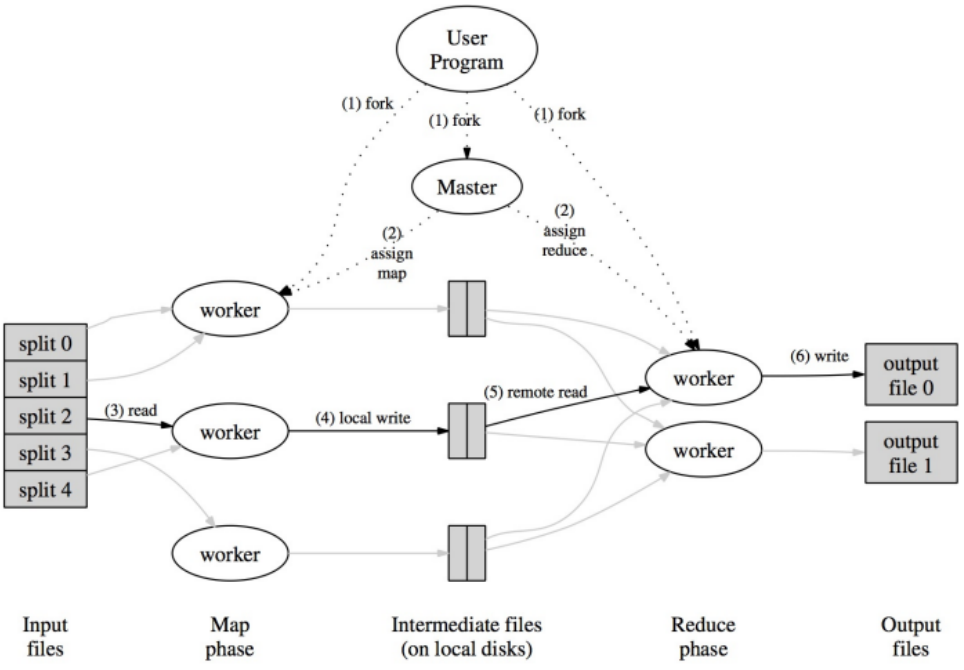
MapReduce este un tipar de dezvoltare a aplicațiilor paralele/distribuite folosit pentru procesarea unor cantități mari de date. În general, se consideră că acest model implică existența unui nod de procesare cu rol de coordonator(master) și a mai multor noduri de procesare cu rol de worker.

2.1. Etapele MapReduce

Cele două etape ale algoritmului MapReduce sunt *maparea* și *reducerea*, ele întâmplându-se în ordinea aceasta.

- Etapa de **mapare**: preia un set de date de intrare și-l transformă într-un nou set de date de forma (cheie, valoare)
    1. nodul cu rol de *coordonator* împarte problema „originala” în subprobleme și le distribuie către workeri pentru procesare
    2. această divizare a problemei se realizează într-o manieră similară metodei de programare *divide et impera*. În anumite cazuri, nodurile *worker* pot divide la rândul lor subproblema primită și pot trimite aceste subdiviziuni către alți *worker-i*, rezultând astfel o arhitectură arborescentă
    3. divizarea caracteristică acestei etape nu trebuie să coreleze efectiv dimensiunea datelor de intrare cu numărul de *worker*-i existenți, astfel un *worker* poate primi mai multe subprobleme de rezolvat

- Etapa de **reducere**: preia seturile de date rezultate în urma etapei de mapare și agregă tuplele respective, rezultând un nou set de perechi (cheie, valoare) de o dimensiune mai mică
    1. nodurile cu rol de *worker* determină soluțiile subproblemelor identificate în faza de mapare
    2. nodul cu rol de *coordonator*(sau un set de noduri cu rol de *worker* desemnate de coordonator) colectează soluțiile subproblemelor și le combină pentru a obține rezultatul final al procesării dorite



3. Prezentarea soluției

Reprezentarea datelor este următoarea: seturile de date de intrare sunt stocate în fișiere aflate într-un folder dat de către utilizator din linia de comandă, iar indexul invers al tuturor fișierelor se află într-un singur fișier. Soluția finală va fi un fișier a cărui linii vor fi de forma:

<cuvânt: (fișier 1, frecvență 1) (fișier 2, frecvență 2) … (fișier N, frecvență N)>

Inițial se verifică că directorul dat de utilizator, unde ar trebui să se afle fișierele de intrare, există. Dacă acesta există se continuă algoritmul cu următorii pași.

Procesul cu rank-ul 0 va fi procesul Master care va coordona fiecare worker. Pentru început, procesul Master va crea folderul “tempFiles” unde se vor stoca fișierele intermediare. După ce acest folder a fost creat, va trimite câte un mesaj individual fiecărui worker pentru ca aceștia să înceapă să aștepte următoarele mesaje de la Master.

În continuare, va începe etapa de mapare, procesul master va trimite fiecărui worker numele de fișiere pe care fiecare va trebui să le parcurgă. Fiecare proces worker va parcurge fiecare fișier primit, îl va transforma într-o listă de cuvinte, va itera prin lista de cuvinte, apoi, într-un director corespunzător worker-ului respectiv(de exemplu worker-ul 2 va avea directorul p2), va adăuga în fișierul corespunzător cuvântului respectiv, numele fișierelor de intrare primite de către worker în care s-a regăsit acel cuvânt. Dacă un cuvânt s-a regăsit de mai multe ori într-un fișier de intrare, atunci numele acelui fișier va apărea de mai multe ori în fișierul corespunzător cuvântului. După ce fiecare worker a terminat de procesat fișierele sale și a scris rezultatele intermediare în folderul “tempFiles”, aceștia îl vor anunța pe procesul Master.

După ce procesul Master a fost anunțat de către toți worker-ii că etapa de mapare s-a încheiat, va începe etapa de reducere, trimițându-i fiecărui worker în parte o partiție din literele alfabetului englez. După ce un worker și-a primit lista de litere, va parcurge toate folderele din “tempFiles” și va deschide fiecare fișier care începe cu una din literele asignate. După ce a deschis un fișier intermediar, va calcula frecvența de apariție a acelui cuvânt în fiecare fișier găsit în fișierul intermediar, rezultând tuple de forma (fișier i, frecvență i). Aceste tuple le va adăuga într-un fișier aflat în directorul “output” care va avea același nume precum fișierul intermediar. După ce procesul worker a terminat etapa de reducere, îl va anunța pe Master. După această etapă în folderul “output” se vor afla câte un fișier cu numele fiecărui cuvânt distinct, iar în interiorul fișierelor se vor afla frecvențele de apariție a cuvântului respectiv în fiecare set de date de intrare în care acesta există.

Într-un final, procesul Master îl deleagă pe procesul cu rank-ul 1 să parcurgă toate fișierele din folderul “output” și să agrege rezultatele într-un fișier numit “finalOutput”.

4. Implementare

Implementarea a fost făcută în python cu ajutorul modulului mpi4py care implementează MPI(Message Passing Interface). Acest modul permite paralelizarea prin lucrul cu mai multe procese.

- **Master**

```
Creare folder intermediar
Creare folder output

foreach worker do
    Anunta inceperea etapei de mapare

while ExistaFisiereNeasignate do
    Asigneaza fisierul unui worker

foreach worker do
    Anunta terminarea transmisiei de fisiere

foreach worker do
    Asteapta confirmarea terminarii fazei de mapare

foreach worker do
    Anunta inceperea etapei de reducere
    Asigneaza o partitie a alfabetului

foreach worker do
    Asteapta confirmarea terminarii fazei de reducere

Anunta worker-ul 1 ca poate incepe etapa finala de agregare
```

- **Worker**

```
Astept confirmare creare folder intermediar si folder final
Creez folderul propriu din folderul intermediar

while EtapaDeReducereFinalizata is False do
    Primeste mesaj de la Master

    if EtapaDeMapare do
        foreach cuvant in fisier do
            if NuExistaFisierIntermediarCuvant do
                Creeaza fisier intermediar si adauga
            Adauga numele fisierului in fisier intermediar

    else if MesajFinalizareTransmisieFisiere do
        Anunta Master de finalizarea prelucrarii fisierelor

    else if EtapaDeReducere do
        foreach folder in folderIntermediar do
            foreach cuvant in folder do
                if cuvant incepe cu o litera asignata do
                    Extrage numele fisierelor unde apare cuvantul
                    Creeaza un dictionar de forma <fisier:frecventa> pentru cuvant
                    Adaugam toate perechiile <fisier:frecventa> din dictionar
                        in folderul de output, in fisierul cuvant

        FinalizareEtapaReducere

if suntProcesul1 do
    Primeste mesaj de la Master
    foreach fisierCuvant in folderOutput do
        Extrage din fisierCuvant indexul invers al cuvantului respectiv
        Adauga indexul invers al cuvantului extras in fisierul output
```

5. Coduri sursa

- **Extragere cuvinte din fișier**

```
def readFile(fileName):
    file = open(f"input/{fileName}", "r", encoding='unicode_escape')
    input = file.read()
    file.close()

    splittedInput = re.findall("[a-zA-Z]+", input)

    if len(splittedInput[len(splittedInput) - 1]) == 0:
        splittedInput = splittedInput[:len(splittedInput) - 1]

    return splittedInput
```

- **Partiționare litere alfabet**

```
def getSliceOfAlphabet(rank, noProcesses):
    # mapping the processes ranks (0, n - 1) to (1, n)
    rank = rank - 1
    noProcesses = noProcesses - 1
    letters = string.ascii_lowercase
    slice = []
    for i in range(len(letters)):
        if i % noProcesses == rank:
            slice.append(letters[i])
    return slice
```

- **Master - creare folder intermediar și folder output**

```
if os.path.isdir(directoryPath):
      try:
          shutil.rmtree(directoryPath)
      except OSError as e:
          print("Error: %s - %s." % (e.filename, e.strerror))
  os.mkdir(directoryPath)

  if os.path.isdir(outputPath):
      try:
          shutil.rmtree(outputPath)
      except OSError as e:
          print("Error: %s - %s." % (e.filename, e.strerror))
  os.mkdir(outputPath)
```

- **Master - mapare**

```
for i in range(1, size):
      comm.send(1, dest=i, tag=TAG.START)

index = 0
destination = 1

while index != len(fileNames):
      comm.send(fileNames[index], dest=destination, tag=TAG.FILE)
      destination = (destination + 1)
      if destination == size:
          destination = 1
      index += 1

  for i in range(1, size):
      comm.send(True, dest=i, tag=TAG.DONE_FILES)

  for i in range(1, size):
      data = int(comm.recv(source=MPI.ANY_SOURCE, tag=TAG.DONE_FILES))
```

- **Master - reducere**

```
for i in range(1, size):
      comm.send(getSliceOfAlphabet(i, size), dest=i, tag=TAG.START_REDUCE)

for i in range(1, size):
      comm.recv(source=MPI.ANY_SOURCE, tag=TAG.DONE)

comm.send(1, dest=1, tag=TAG.FINAL_REDUCE)
```

- **Worker - mapare**

```
status = MPI.Status()
data = comm.recv(source=MPI.ANY_SOURCE, tag=MPI.ANY_TAG, status=status)
tag = status.Get_tag()

if tag == TAG.FILE:
    words = readFile(data)

    for word in words:
        if word != '':
            word = word.lower()
            outFilePath = os.path.join(tempProcessPath, word)
            outFile = open(outFilePath, 'a')
            outFile.write(f'{data} ')
            outFile.close()
```

- **Worker - reducere**

```
for root, dirs, files in os.walk(directoryPath):
    for filename in files:
        if filename[0] in data:
            f = open(os.path.join(root, filename), 'r')

            words = f.read().split()
            words.sort()

            result = {}

            for word in words:
                if word in result:
                    result[word] += 1
                else:
                    result[word] = 1

            for key in result:
                outFilePath = os.path.join(outputPath, filename)
                outFile = open(outFilePath, 'a')
                outFile.write(f'({key},{result[key]}) ')
                outFile.close()
```

```
            f.close()
comm.send(rank, dest=0, tag=TAG.DONE)
```

- **Worker - agregare rezultate finale**

```
comm.recv(source=MPI.ANY_SOURCE, tag=TAG.FINAL_REDUCE)
outputFilePath = os.path.join(path, 'finalOutput')

if os.path.isfile(outputFilePath):
    os.remove(outputFilePath)
output = open(outputFilePath, 'a')

for root, dirs, files in os.walk(outputPath):
    files.sort()
    for filename in files:
        f = open(os.path.join(root, filename), 'r')
        content = f.read()
        f.close()

        output.write(f'{filename}:{content}\n')
output.close()
```

6. Rezultate

Fișierul rezultat în urma rulării pe setul de date primit arată astfel:

```
a:(1.txt,2152) (12.txt,1083) (13.txt,1434) (17.txt,655) (19.txt,4413) (24.txt,1478) (3.txt,2097) (6.txt,339) (7.txt,1999) (14.txt,774) (15.txt,1933) (18.txt,1583) (21.txt,420) (23.txt,242) (25.txt,5109) (5.txt,695) (8.txt,248) (10.txt,2973) (11.txt,1110) (16.txt,1128) (2.txt,2069) (20.txt,914) (22.txt,241) (4.txt,348) (9.txt,309)
aa:(2.txt,1)
aback:(1.txt,2) (10.txt,1) (2.txt,2)
abaft:(6.txt,1) (10.txt,2)
abandon:(1.txt,1) (13.txt,1) (17.txt,1) (19.txt,1) (3.txt,4) (7.txt,4) (23.txt,1) (25.txt,8) (10.txt,1) (2.txt,3) (20.txt,1) (22.txt,1)
abandoned:(1.txt,6) (12.txt,2) (13.txt,2) (17.txt,3) (19.txt,12) (3.txt,3) (7.txt,4) (15.txt,1) (18.txt,6) (23.txt,1) (25.txt,4) (5.txt,1) (8.txt,1) (10.txt,2) (11.txt,3) (16.txt,1) (2.txt,5) (20.txt,1) (22.txt,1)
abandoning:(1.txt,1) (18.txt,1) (25.txt,2)
abandonment:(1.txt,1) (17.txt,1) (18.txt,1) (2.txt,1)
abandons:(25.txt,4) (2.txt,1)
abasement:(10.txt,1) (9.txt,1)
abate:(1.txt,1) (19.txt,1) (7.txt,1)
abated:(19.txt,1) (10.txt,1)
abating:(7.txt,1) (10.txt,2)
abba:(19.txt,1)
abbe:(25.txt,1)
abbey:(10.txt,9)
abbie:(3.txt,2)
abbott:(24.txt,1) (14.txt,1) (15.txt,2)
abbottsville:(15.txt,5)
abbreviation:(16.txt,1)
abbreviations:(19.txt,2)
abc:(3.txt,2)
abcd:(21.txt,1)
abdalla:(7.txt,6)
abdera:(16.txt,1)
abdicating:(7.txt,1)
abdomen:(3.txt,1) (2.txt,1)
abdominal:(3.txt,1)
abduct:(18.txt,1)
abduction:(19.txt,1) (7.txt,1)
```

```
36308 zhuk:(19.txt,1)
36309 zigzag:(1.txt,1) (3.txt,1) (18.txt,1)
36310 zigzagged:(19.txt,1)
36311 zinc:(12.txt,1) (19.txt,1)
36312 zinebi:(7.txt,2)
36313 zinnias:(6.txt,2)
36314 zionism:(5.txt,4)
36315 zionist:(5.txt,7)
36316 zip:(1.txt,3) (24.txt,1) (14.txt,1) (25.txt,1) (10.txt,2) (16.txt,2) (20.txt,1)
36317 zipper:(12.txt,1) (20.txt,3)
36318 zippered:(9.txt,1)
36319 zippers:(2.txt,1)
36320 zipping:(20.txt,1)
36321 zips:(1.txt,1) (2.txt,1)
36322 zlydarikha:(19.txt,3)
36323 znamensky:(19.txt,1)
36324 zobeida:(7.txt,31)
36325 zoic:(20.txt,1)
36326 zola:(5.txt,1)
36327 zone:(17.txt,1) (19.txt,6) (3.txt,3) (18.txt,2) (11.txt,1) (2.txt,4)
36328 zones:(18.txt,2) (2.txt,1)
36329 zonked:(9.txt,1)
36330 zoo:(19.txt,1) (3.txt,11) (10.txt,1) (9.txt,1)
36331 zoological:(10.txt,3)
36332 zoom:(1.txt,1) (18.txt,1)
36333 zooms:(3.txt,1)
36334 zoophagous:(10.txt,5)
36335 zoophagy:(10.txt,1)
36336 zoos:(3.txt,1)
36337 zop:(16.txt,1)
36338 zouman:(7.txt,1)
36339 zulus:(7.txt,1)
36340 zurich:(3.txt,2) (5.txt,4)
36341 zusha:(19.txt,3)
36342 zvonarskaia:(19.txt,1)
36343 zybushi:(19.txt,1)
36344 zybushino:(19.txt,15)
```

Bibliografie

1. Wikipedia
2. Laborator Algoritmi Paraleli și Distribuiți
3. IBM Corp. What is MapReduce?
4. Michael Kleber. The MapReduce paradigm, January 2008