

Algoritmi Paraleli si Distribuiti

Tema de casă

Algoritmul MapReduce

Proiect realizat de
Chelarașu Elena-Denisa

Profesor îndrumător
Archip Alexandru

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

Enunț problemă

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 parametri 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.

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 lista de forma $\langle \text{docID}_x, \{\text{term}_1 : \text{count}_1, \text{term}_2 : \text{count}_2, \dots, \text{term}_n : \text{count}_n\} \rangle$ (**index direct** – count_k înseamnă numărul de apariții ale termenului k)
2. fiecare lista obținută în pasul anterior este spartă în perechi de forma: $\langle \text{docID}_x, \{\text{term}_k : \text{count}_k\} \rangle$; pentru fiecare astfel de pereche, se realizează o inversare de valori astfel încât să obținem: $\langle \text{term}_k, \{\text{docID}_x : \text{count}_k\} \rangle$
3. perechile obținute în pasul anterior sunt sortate după term_k (cheie primară), docID_x (cheie secundară)

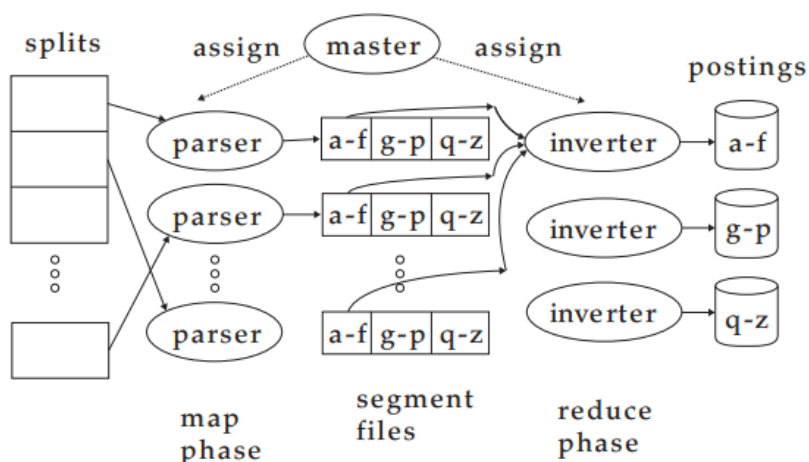


Fig. 1: Preluat din 1, capitolul 4.4: Distributed Indexing

Teorie

MapReduce este o paradigmă de programare folosită pentru procesarea unor cantități mari de date în mod paralel și distribuit pe un cluster. Un program MapReduce este compus dintr-o procedură Map() care selectează și sortează datele și o procedură Reduce() care îndeplinește operația de însumare a rezultatelor. Ca și mod de lucru, se considera existența unui nod master (root-ul programului) care orchestrează împărțirea sarcinilor unui număr de lucrători(workeri), precum este prezentat în Fig.2.

După cum spune și denumirea, avem două etape primare [4, slide 6]:

Mapare: $(k, v) \mapsto \langle (k_1, v_1), (k_2, v_2), (k_3, v_3), \dots, (k_n, v_n) \rangle$

Reducere: $(k',) \mapsto \langle (k', v''_1), (k', v''_2), \dots, (k', v''_{n''}) \rangle$

Dar cele două etape pot fi rafinate în cinci subetape:

1. Preprocesare – pregătirea datelor pentru etapa de mapare
2. Mapare– stabilirea datelor de interes
3. Amestecare și sortare – organizarea datelor a.i. să fie optimizată etapa de reducere
4. Reducere – determinarea rezultatului
5. Stocare rezultat

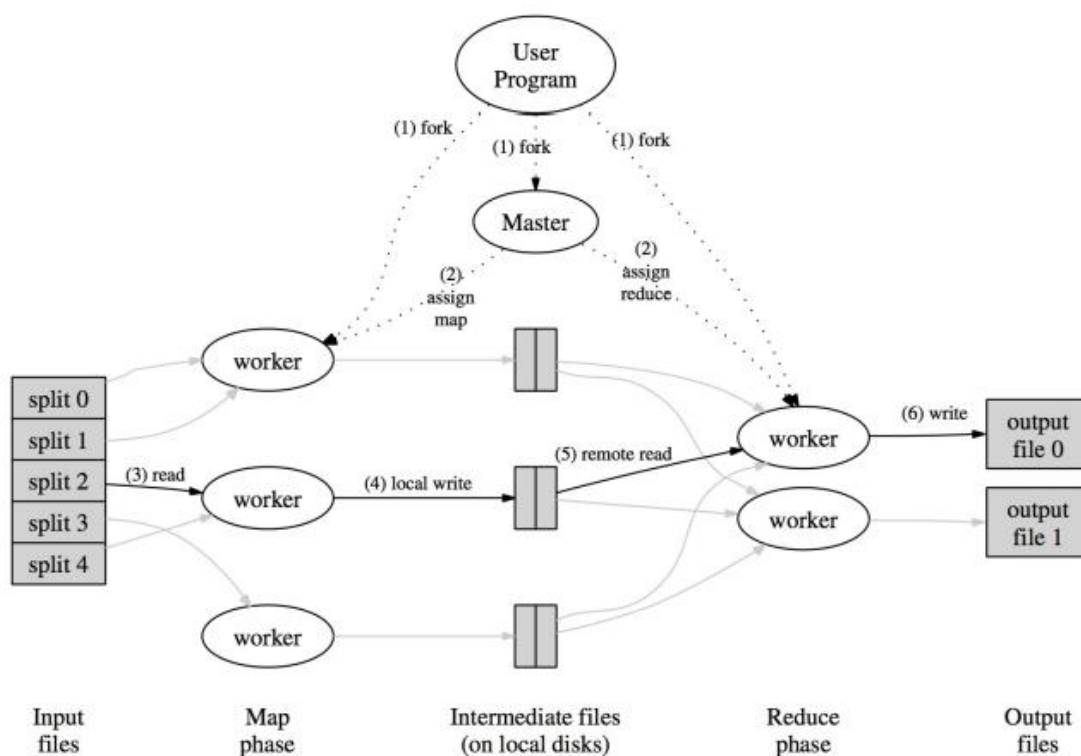


Fig. 2: Paradigma MapReduce

Implementare

Indexul invers: Funcția de mapare parsează fiecare document și emite o secvență de perechi $\langle \text{docID}, \text{termk} \rangle$. Funcția de reducere acceptă toate perechile pentru un cuvânt, sortează id-urile documentului corespunzător și întoarce o pereche. Setul tuturor perechilor de la ieșire formează un index invers de tipul $\langle \text{termk}, \{ \text{docIDk1: countk1}, \text{docIDk2: countk}, \dots, \text{docIDkn: countkn} \} \rangle$.

Prima data, masterul (procesul cu rank-iul 0) va face impartirea joburilor în mod egal tuturor workerilor. Acesta transmite fiecare fisier catre workerul corespunzator.

```
zona trimite_joburi:
    muncitor_actual = 1
    contor = 1
    for file in files_names:
        if contor == no_of_jobs and muncitor_actual != no_of_workers:
            muncitor_actual += 1
            contor = 1
        trimit fisierul de prelucrat catre muncitor_actual
        contor += 1
```

Workerul primește numele fisierului de parsat si incepe partea de mapare. Workerul deschide fișierul și prei fiecare cuvânt si il salveaza intr-un vector. Creeaza un cod unic (in cazul meu, un timestamp) si salveaza fiecare cuvânt ca fisier separat de forma docID_termk_timestamp.txt ca in Fig. 3. La final, workerul trimite un semnal de terminare a jobului către master.

```
zona mapare:
    primesc numele fisierului de parsat
    words -> cuvintele din fisier
    i -> 1
    for i in words:
        for p in semne_de_punctuatie:
            sterg semnul de punctuatie din cuvânt
    for cuvânt in words:
        nume -> denumire docID_termk_timestamp.txt
        creare fisier nume
    trimit la master semnalul de finalizare
```

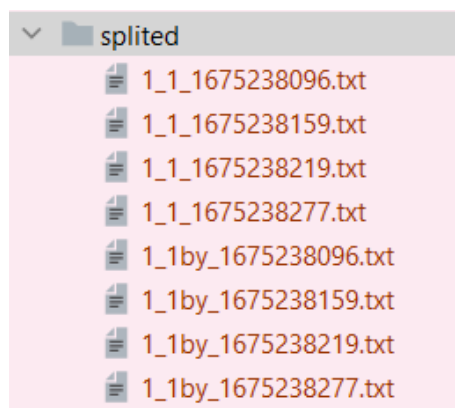


Fig. 3: Directorul splited ce contine perechi de forma docID_termen_timestamp.txt

După ce a primit semnalul ca fișierele au fost parsate, împart aproximativ munca egal la toți workerii. Pentru a crea intervalele de [start, finish] ce trebuie prelucrate de către worker din cadrul directorului splited, trebuie realizat un offset: în cazul în care următorul fișier va conține tot un cuvânt din același document cu cel actual, voi prelunge intervalul pentru a prelucra toate cuvintele din același fișier cu același worker. Intervalul de baza este considerat {inceput, inceput + pas + adaos}. Ultimul worker va avea ce a mai rămas de prelucrat.

```
zona trimitere_intervale:
  primesc semnalul de ok
  l <- numarul de fisiere
  pas <- numarul de fisiere impartit la workeri
  i <- 1
  muncitor_curent <- 1
  while i < l:
    i <- i + pas
    if sunt la ultimul muncitor:
      trimit ultimul interval
    next_i <- urmatorul docID
    while i egal cu next_i si i nu este la final:
      iterez prin indecsi pana nu mai sunt egali
      i <- i + 1
    trimit intervalul [inceput, sfarsit]
    muncitor_curent <- muncitor_curent + 1
    i <- i + 1
```

Master trimite acum intervalele de lucru. Funcția de prelucrare realizează un dicționar de forma {"cuvant" : count} pentru a realiza perechi de forma cuvânt_docID_count.txt. Creează și fișierele cu ajutorul dicționarului (Fig. 4).

```
zona inversare_valori:
  primesc intervalul de lucru
  fisiere <- toate fisierele
  nume_fisier <- docID de la indexul start
  i <- start
  while i < final:
    stochez cuvintele intr-un dictionar si le aflu numarul de aparitii
  j <- 0
  for j in dictionar_contoare:
    creez fisierul cu nume in ./counters
  trimit semnal de finalizare catre master
```

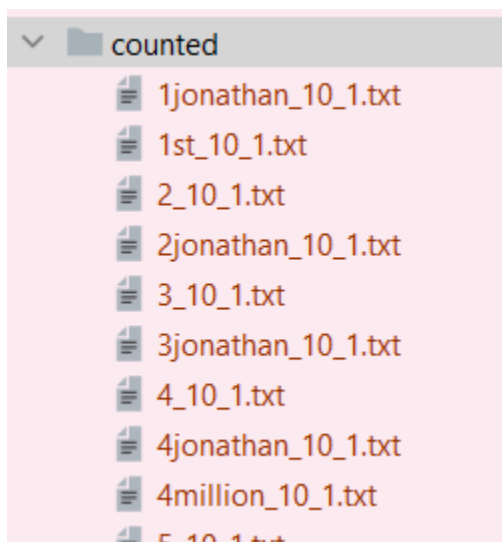



Fig. 4: Directorul counted ce conține perechi de forma termen_docID_count.txt

Repetăm partea de creare de intervale pentru prelucrarea lor, dar de data aceasta pentru directorul ./counted precum în Fig.5. Workerul poate să înceapă crearea indexului invers prin algoritmul prezentat mai jos. De menționat faptul că partea de sortare a cuvintelor a fost realizată automat de către sistemul de operare prin aranjarea alfabetică a fișierelor.

```
zona index_invers:  
primesc intervalul de lucru  
creez nume de fisiere de forma termk_docID1_count1_docID2_count2_ .. _docIDn_countn.txt  
creez fisierele cu numele facute  
trimit semnal de finalizare
```



```
Eu sunt 0 si am trimis catre 1 intervalul [0, 2542] de prelucrat  
Eu sunt 0 si am trimis catre 2 intervalul [2543, 5085] de prelucrat  
Eu sunt 0 si am trimis catre 3 intervalul [5086, 7628] de prelucrat  
Eu sunt 0 si am trimis catre 4 intervalul [7629, 10171] de prelucrat  
Eu sunt 0 si am trimis catre 5 intervalul [10172, 12714] de prelucrat  
Eu sunt 0 si am trimis catre 6 intervalul [12715, 15257] de prelucrat  
Eu sunt 0 si am trimis catre 7 intervalul [15258, 17800] de prelucrat
```

Exemplu împărțirea fișierelor de prelucrat în etapa de indexare inversa

Bibliografie

1. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. An Introduction to Information Retrieval. Cambridge University Press, Cambridge, England, Online 2009
Cambridge UP edition, 2009
2. “Tema de casa MapReduce”, Laborator Algoritmi Paraleli si Distribuiti
3. Lisandro Dolcin. “MPI for Python.”
<https://mpi4py.readthedocs.io/en/stable/index.html#mpi-for-python>
4. Michael Kleber. The MapReduce paradigm.
<https://sites.google.com/site/mriap2008/lectures>, January 2008.