ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #2 Procesarea de documente folosind paradigma Map-Reduce

Responsabili: Radu-Ioan Ciobanu, Silviu-George Pantelimon, Lucian Grigore, Bogdan Oprea, Diana Megelea

Termen de predare: 15-12-2021 23:59
Ultima modificare: 23-11-2021 18:50

Cuprins

Cerință

Să se implementeze un program paralel în Java pentru procesarea unui set de documente text primit ca input, evaluarea lungimilor cuvintelor procesate, precum și ordonarea documentelor în funcție de lungimea cuvintelor și frecventa cu care acestea apar. Fiecare cuvânt va avea asociată câte o valoare, în funcție de numărul de litere. Valoarea unui cuvânt este determinată de o formulă bazată pe șirul lui Fibonacci, așa cum se va explica mai târziu. Rangul unui document se calculează însumând valorile tuturor cuvintelor din acesta. În plus, pentru fiecare document se va stabili cuvântul de lungime maximă (sau cuvintele, dacă sunt mai multe cu aceeași lungime maximă).

În urma procesului de parsare, se va determina numărul de litere al fiecarui cuvânt existent într-un document, obținându-se o listă de perechi $\{lungime, numar\_aparitii\}$, unde $numar\_aparitii$ reprezintă numărul de apariții ale tuturor cuvintelor din document care au lungimea egală cu $lungime$. Programul trebuie să permită calcularea unei metrici pentru toate documentele procesate și să afișeze documentele în ordinea acestei metrici.

Pentru paralelizarea procesării documentelor, se va folosi modelul Map-Reduce. Fiecare document se va fragmenta în părți de dimensiune fixă ce vor fi procesate în paralel (operațiunea de **Map**), pentru fiecare parte rezultând câte un dicționar parțial (care conține lungimea cuvintelor și numărul de apariții ale acestora) și o listă conținând cuvintele de dimensiune maximă din fragmentul procesat. Pasul următor îl reprezintă combinarea dicționarelor (operațiunea de **Reduce**) în urma căreia se obține un dicționar ce caracterizează întregul document, iar la fel se va face și în cazul listelor de cuvinte maximale. Pentru fiecare document, se vor calcula rangul (în funcție de numărul de apariții ale cuvintelor de o anumită lungime) și numărul de cuvinte maximale.

Paradigma Map-Reduce

Pentru rezolvarea temei, se va folosi un model Map-Reduce similar cu cel folosit la Google pentru procesarea unor seturi mari de documente în sisteme distribuite. Acest articol prezintă modelul Map-Reduce folosit de Google și o parte dintre aplicațiile lui (mai importante pentru înțelegerea modelului sunt primele 4 pagini).

Map-Reduce este un model (și o implementare asociată) de programare paralelă pentru procesarea unor seturi imense de date, folosind sute sau mii de procesoare. În majoritatea cazurilor, Map-Reduce este folosit într-un context distribuit, fiind, de fapt, un model de programare care poate fi adaptat pentru ambele situații. Cea mai cunoscută implementare este Apache Hadoop, dezvoltat inițial de către Doug Cutting și Mike Cafarella. Modelul permite paralelizarea și distribuirea automată a task-urilor. Paradigma Map-Reduce se bazează pe existența a două funcții care îi dau și numele: Map și Reduce. Funcția Map primește ca input o funcție f și o listă de elemente, și returnează o nouă listă de elemente rezultată în urma aplicării funcției f asupra fiecărui element din lista inițială. Funcția Reduce combină rezultatele obținute anterior.

Mecanismul Map-Reduce funcționează în modul următor:

- utilizatorul cere procesarea unui set de documente
- această cerere este adresată unui proces (sau fir de execuție) coordonator
- coordonatorul împarte documentele în fragmente de dimensiuni fixe, care vor fi asignate unor procese (fire de execuție) worker
- un worker va executa o operație numită Map pentru un fragment de fișier, care va genera niște rezultate parțiale având forma unor perechi de tip $\{cheie, valoare\}$
- dupa ce operațiile Map au fost executate, coordonatorul asignează worker-ilor sarcini de tip Reduce, prin care se combină rezultatele parțiale.

Detalii tehnice

Dându-se un set de N documente, să se determine rangul fiecărui document conform unei metrici prezentate în această secțiune, precum și cuvântul/cuvintele de lungime maximă din fiecare document.

Operațiile de tip Map

Pornind de la lista de documente de procesat ce va fi disponibilă în fișierul de intrare, se determină dimensiunea fiecărui document și se creează câte un task de tip Map pentru fiecare fragment de câte D octeți dintr-un document (cu excepția ultimului fragment, care poate fi mai scurt). Un task de tip Map va avea următoarele informații:

- numele documentului
- offset-ul de început al fragmentului din document
- dimensiunea fragmentului.

Atenție! În pasul de creare a task-urilor de tip Map, nu se va încărca în memorie conținutul documentului de indexat.

După ce au fost create task-urile de tip Map, acestea se vor împărți unui grup de thread-uri pentru executare. Pentru fiecare task, se va citi fragmentul de dimensiune D din document și se va construi un dicționar având ca și chei lungimile cuvintelor găsite în fragment, iar ca valori numărul de apariții ale unor cuvinte de această lungime. Tot în etapa de Map, se va construi o listă de cuvinte de lungime maximă pentru fragmentul procesat. Rezultatul unui task de tip Map va conține numele documentului, dicționarul cu numărul de apariții pentru lungimile cuvintelor găsite, precum și lista de cuvinte de lungime maximă.

Pentru a delimita cuvintele din documente, se vor folosi ca separatori următoarele caractere:

$$;:/?\tilde{}\backslash.,><`[]\{\}()!@\#\$\%\hat{}\&-\_+'=*"|spatiu/tab/endline$$

Atenție! Prin spațiu, înțelegem atât \n, cât și \r.

Atenție! Poate apărea problema ca fragmentul prelucrat de un worker să se termine sau să înceapă în mijlocul unui cuvânt. În acest caz, se va adopta următoarea convenție: dacă fragmentul începe în mijlocul unui cuvânt, worker-ul va sări peste acel cuvânt; dacă fragmentul se termină în mijlocul unui cuvânt, worker-ul va prelucra și acel cuvânt. În acest mod, cuvintele care sunt la granița dintre două fragmente vor fi prelucrate tot timpul de worker-ul ce se ocupă de fragmentul care se află în fișier înaintea cuvântului respectiv.

Rezultatele operațiilor Map vor fi ținute în memorie. În mod normal, ele s-ar fi scris și pe disc.

Operațiile de tip Reduce

Având rezultatele din cadrul operațiunii de Map, un task de tip Reduce va avea următoarele informații:

- numele documentului
- lista de rezultate din cadrul operațiunii de Map pentru acel document.

Înainte să începeți operațiile de tip Reduce, trebuie să vă asigurați că toate operațiile de tip Map s-au finalizat. Așadar, task-urile de tip Reduce pot fi alocate unui grup nou de thread-uri. Execuția unui task Reduce va consta din două etape: etapa de combinare și etapa de procesare.

Etapa de combinare constă în combinarea listei de rezultate parțiale din etapa de Map. Astfel, se vor combina dicționarele cu lungimile cuvintelor astfel încât să avem la final un singur dicționar ce reprezintă lungimea cuvintelor împreună cu numărul lor de apariții pentru un întreg document. În cazul listelor de

cuvinte maximale, se vor combina cele care au cuvinte de dimensiunea cea mai mare per întreg documentul, iar restul cuvintelor (cele local maximale) vor fi decartate.

Etapa de procesare constă în calcularea rangului unui document folosind dicționarul din etapa de combinare și șirul lui Fibonacci (0, 1, **1, 2, 3, 5, 8, 13, 21, 34**, etc.). Tot în cadrul etapei de combinare, se va determina numărul de cuvinte de lungime maximă din document.

Rangul unui document se va calcula folosind formula:

$$Rang = \frac{\sum_{k=1}^{n} F(lungime_k + 1) \times numar\_aparitii_k}{numar\_cuv\_total}$$

În formula de mai sus:

- $F(lungime_k + 1)$ reprezintă valoarea din șirul lui Fibonacci a respectivei lungimi. De exemplu, un cuvânt de o literă are valoarea 1, unul de două litere are valoarea 2, unul de trei litere are valoarea 3, unul de patru litere are valoarea 5, unul de cinci litere are valoarea 8, etc.
- $numar\_aparitii_k$ reprezintă numărul de cuvinte din document care au lungimea egală cu $lungime_k$
- *numar_cuv_total* reprezintă numărul total de cuvinte din documentul respectiv.

Atenție! Afisați rangurile cu două zecimale, obținute prin rotunjire. În Java, puteți face acest lucru astfel: `String.format("%.2f", rang)`.

Execuție

Programul se va rula în felul următor:

```
java Tema2 <numar_workeri> <fisier_intrare> <fisier_iesire>
```

Atenție! Se vor porni câte *numar_workeri* thread-uri pentru fiecare tip de operație (Map și Reduce).

Fișierul de intrare are următorul format:

```
dimensiune_fragment
numar_documente
nume_doc1
...
nume_docN
```

Așadar, prima linie specifică dimensiunea în octeți a fragmentelor în care se vor împărți fișierele. A doua linie conține numărul de documente text de procesat, iar următoarele linii conțin numele documentelor, câte unul pe linie. Toate fișierele de intrare vor conține **doar** caractere ASCII și se pot considera valide.

În fișierul de ieșire, se vor afișa documentele în ordinea descrescătoare a rangului, câte unul pe rând, conținând informații despre numele documentului, rangul său, precum și dimensiunea cuvântului sau cuvintelor maximale și numărul de cuvinte de această dimensiune găsite în text. Astfel, formatul fișierului de ieșire este următorul:

```
nume_doc1,rang_doc1,dimensiune_maximala_doc1,numar_cuvinte_maximale_doc1
...
nume_docN,rang_docN,dimensiune_maximala_docN,numar_cuvinte_maximale_docN
```

Atenție! Dacă două (sau mai multe) documente au același rang, se vor adăuga în fișierul de ieșire unul după altul în ordinea în care ele apar în fișierul de intrare.

Exemplu

Să presupunem că avem următorul fișier de intrare:

```
10
3
in1.txt
in2.txt
in3.txt
```

Avem deci trei documente de analizat, iar dimensiunea unui fragment este de 10 octeți. Să prespunem că avem de analizat următoarele documente:

```
$ cat in1.txt
Algoritmi Paraleli si Distribuiti

$ cat in2.txt
Ana are mere

$ cat in3.txt
mutex, lock, zavor, mutex
```

La primul pas, se creează task-urile de tip Map, pe baza împărțirii fișierelor de intrare în fragmente de câte maxim 10 octeți. Task-urile rezultate pentru cele trei fișiere de intrare de mai sus vor fi următoarele:

- T1 → *in1.txt*, offset 0, dimensiune 10 (“Algoritmi ”)
- T2 → *in1.txt*, offset 10, dimensiune 10 (“Paraleli s”)
- T3 → *in1.txt*, offset 20, dimensiune 10 (“i Distribu”)
- T4 → *in1.txt*, offset 30, dimensiune 3 (“iti”)
- T5 → *in2.txt*, offset 0, dimensiune 10 (“Ana are me”)
- T6 → *in2.txt*, offset 10, dimensiune 2 (“re”)
- T7 → *in3.txt*, offset 0, dimensiune 10 (“mutex, loc”)
- T8 → *in3.txt*, offset 10, dimensiune 10 (“k, zavor, ”)
- T9 → *in3.txt*, offset 20, dimensiune 5 (“mutex”).

Mai departe, se alocă cele nouă task-uri worker-ilor, într-un mod cât mai echilibrat. Pentru un task, un worker deschide fișierul, se duce la offset-ul specificat în task, verifică dacă fragmentul începe sau se termină în mijlocul unui cuvânt (caz în care ajustează fragmentul pe care îl analizează), iar apoi execută operațiile cerute.

Să presupunem că avem trei thread-uri worker (W1, W2, W3), deci fiecăruia îi vor fi alocate câte trei task-uri (în exemplul nostru, worker-ul W1 va procesa task-urile T1, T4, T7, worker-ul W2 va procesa task-urile T2, T5, T8, iar worker-ul W3 va procesa task-urile T3, T6, T9). Ajustarea fragmentelor pentru a nu procesa părți incomplete de cuvinte (conform descrierii de mai sus) ne duce la următoarea împărțire a fragmentelor de documente între thread-urile worker:

- W1 → "Algoritmi " (T1, *in1.txt*), "" (T4, *in1.txt*), "mutex, lock" (T7, *in3.txt*)
- W2 → "Paraleli si" (T2, *in1.txt*), "Ana are mere" (T5, *in2.txt*), ", zavor, " (T8, *in3.txt*)
- W3 → "Distribuiti" (T3, *in1.txt*), "" (T6, *in2.txt*), "mutex" (T9, *in3.txt*).

Fiecare task va avea ca rezultat câte un dicționar cu dimensiunile de cuvinte dintr-un fragment și numărul lor de apariții, precum și cu lista de cuvinte de dimensiune maximă. În cazul exemplului nostru, rezultatele vor fi următoarele:

- RT1 → *in1.txt*; {9: 1}; ("Algoritmi")
- RT2 → *in1.txt*; {8: 1, 2: 1}; ("Paraleli")
- RT3 → *in1.txt*; {11: 1}; ("Distribuiti")
- RT4 → *in1.txt*; {}; ()
- RT5 → *in2.txt*; {3: 2, 4: 1}; ("mere")
- RT6 → *in2.txt*; {}; ()
- RT7 → *in3.txt*; {5: 1, 4: 1}; ("mutex")
- RT8 → *in3.txt*; {5: 1}; ("zavor")
- RT9 → *in3.txt*; {5: 1}; ("mutex").

În continuare, coordonatorul preia rezultatele etapei de Map și creează noile task-uri pentru thread-urile worker care se ocupă cu etapa de Reduce. Aceste task-uri, pe baza datelor prezentate anterior, arată în felul următor:

- T1 → *in1.txt*; {9: 1}, {8: 1, 2: 1}, {11: 1}, {}; ("Algoritmi") ("Paraleli") ("Distribuiti") ()
- T2 → *in2.txt*; {3: 2, 4: 1}, {}; ("mere") ()
- T3 → *in3.txt*; {5: 1, 4: 1}, {5: 1}, {5: 1}; ("mutex") ("zavor") ("mutex")

Fiecare worker primește câte un astfel de task și calculează rezultatele finale, adică rangul fiecărui fișier, precum și lungimea maximă de cuvânt dintr-un fișier și numărul de apariții ale cuvintelor cu acea lungime. În exemplu prezentat mai sus, rezultatele celor trei task-uri sunt următoarele:

- T1 → *in1.txt*, 58.75 (rang), 11 (lungime maximă), 1 (cuvânt cu acea lungime)
- T2 → *in2.txt*, 3.67 (rang), 4 (lungime maximă), 1 (cuvânt cu acea lungime)
- T3 → *in3.txt*, 7.25 (rang), 5 (lungime maximă), 3 (cuvinte cu acea lungime)

Pentru a clarifica, fișierul *in1.txt* are rangul 58.75 pentru că are un cuvânt de lungime 9 (cu valoarea 55), un cuvânt de lungime 8 (cu valoarea 34), un cuvânt de lungime 2 (cu valoarea 2) și un cuvânt de lungime 11 (cu valoarea 144), iar rangul va fi $235/4 = 58.75$. Fișierul de ieșire va arăta deci astfel:

```
in1.txt,58.75,11,1
in3.txt,7.25,5,3
in2.txt,3.67,4,1
```

Notare

Tema se va trimite și testa automat la această adresă și se va încărca de asemenea și pe Moodle. Se va încărca o arhivă Zip care va conține, **în rădăcină**, fișierele sursă pornind de la scheletul oferit în repository-ul temei, precum și un *README* în format text, în care să se descrie pe scurt implementarea temei.

Tema voastră trebuie să se poată compila și rula astfel (unde *Tema2.java* este fișierul care conține metoda `main`, așa cum este și în repository-ul temei):

```
$ javac *.java
$ java Tema2 <numar_workeri> <fisier_intrare> <fisier_iesire>
```

Punctajul este divizat după cum urmează:

- **50p** - implementarea pașilor conform specificațiilor temei
- **30p** - consistența rezultatelor (la rulări multiple pe aceleași date de intrare, trebuie obținute aceleași rezultate)
- **20p** - claritatea codului și a explicațiilor din README.

Atenție! Checker-ul vă dă doar cele 30 de puncte pentru consistența și corectitudinea rezultatelor rulării. Este sarcina voastră să respectați specificațiile temei, așa cum sunt ele prezentate în enunț.

Atenție! Checker-ul are instalat JDK 14.0.2.

Bonus

În cadrul acestei teme, puteți obține până la **20p** bonus la punctaj pentru stilul de programare. În general, programarea combină multe noțiuni, iar cunoștințele acumulate la alte materii pot fi folosite foarte bine în combinație cu programarea paralelă. Astfel, vom acorda acest punctaj bonus dacă:

- ați obținut punctajul maxim pentru rezolvarea problemei
- rezolvați tema în mod generic, adică dacă aveți o implementare de Map-Reduce pentru orice tip de date de intrare, orice tip de date de ieșire și orice tip de operații Map și Reduce, iar în final folosiți implementarea generică pe cazul particular al temei.

Adoptarea diverselor framework-uri de programare paralelă și distribuită cum sunt Apache Hadoop sau Apache Spark pe scară largă nu se datorează doar faptului că au o performanță bună, dar și pentru că interfețele expuse de acestea sunt generice și adesea funcționale din punct de vedere al paradigmei de programare.

Acest bonus este o provocare ca să vedem cum puteți folosi cunostințe de programare orientată pe obiecte, programare funcțională și design patterns.

Testare

Pentru a vă putea testa tema și locall, găsiți în repository-ul temei un set de fișiere de intrare de test, precum și un script Bash (numit *test.sh*) pe care îl puteți rula pentru a vă verifica corectitudinea rezultatelor. Acest script este folosit și pentru testarea automată.

Pentru a putea rula scriptul așa cum este, trebuie să aveți următoatoarea structură de fișiere:

```
$ tree
.
+-- skel
|   +-- Tema2.java
|   +-- [...] (celelalte fisiere sursa)
+-- test.sh
+-- tests
|   +-- files
|   |   +-- [...] (fisiere test)
|   +-- in
|   |   +-- [...] (fisiere input)
+---+-- out
        +-- [...] (fisiere output)
```

La rulare, scriptul execută următorii pași:

1. compilează sursele
2. rulează implementarea paralelă pe cinci fișiere de intrare pentru 1, 2, 3 și 4 thread-uri worker
3. se compară rezultatele obținute cu fișierele output etalon
4. se calculează punctajul final din cele 30 de puncte alocate corectitudinii rezultatelor (20 de puncte fiind rezervate pentru claritatea codului și a explicațiilor, iar 50 pentru respectarea specificațiilor Map-Reduce).

Scriptul necesită existența utilitarelor *diff* și *timeout*.

Link-uri utile

1. The Anatomy of a Large-Scale Hypertextual Web Search Engine
2. Can Your Programming Language Do This?
3. MapReduce (Wikipedia)
4. MapReduce: Simplified Data Processing on Large Clusters
5. Apache Hadoop
6. Apache Spark