ALGORITMI PARALELI SI DISTRIBUITI:
Tema #2 Procesarea documentelor folosind paradigma Map-Reduce

Termen de predare: 25 Noiembrie 2015, ora 23:55

Titulari curs: *Valentin Cristea, Ciprian Dobre, Elena Apostol*
Responsabili Tema: *Elena Apostol, Vlad Dragoi, Valeriu Stanciu*

Cuprins

1.Cerinţele temei

In aceasta tema veti implementa un program paralel in Java pentru procesarea unui set de documente text primit ca input si apoi evaluarea lungimilor cuvintelor procesate si ordonarea documentelor in functie de lungimea cuvintelor si frecventa cu care acestea apar. Fiecare cuvant va avea asociata o valoare, in functie de numarul de litere. Valoarea unui cuvant este determinata de indexul in *sirul lui Fibonacci* corespunzator lungimii cuvantului. Rangul unui document se calculeaza insumand valorile tuturor cuvintelor din acesta. In plus, pentru fiecare document, se va stabili cuvantul/cuvintele de lungime maxima.

In urma procesului de parsare se determina numarul de litere al fiecarui cuvânt existent intr-un document, obtinându-se o lista de perechi **(*lungime, numar_aparitii*)**, unde ***numar_aparitii*** reprezinta numarul de aparitii a tuturor cuvintelor din document care au lungimea egala cu ***lungime***. Programul trebuie sa permita calcularea unei metrici pentru toate documentele procesate si sa afiseze documentele in ordinea acestei metrici.

Pentru paralelizarea procesarii documentelor se va folosi paradigma Replicated Workers (*vezi Laborator 5*) si modelul MapReduce. Fiecare document se va fragmenta in parti de dimensiune fixa ce vor fi procesate in paralel (operatiunea de MAP), pentru fiecare parte rezultând câte un hash partial ce contine lungimea cuvintelor si numarul de aparitii ale acestora si o lista continand cuvintele de dimensiune maxima din acel fragment. Pasul urmator il reprezinta combinarea hash-urilor (operatiunea de REDUCE) in urma caruia se obtine un hash ce caracterizeaza intregul document; la fel se va face si in cazul listelor de cuvinte maximale. Pentru fiecare document se vor calcula rangul in functie de numarul de aparitie ale cuvintelor de o anumita lungime, cat si numarul de cuvinte maximale.

2. Implementare

2.1 Paradigma Map-Reduce - Prezentare generala

Pentru rezolvarea acestei probleme se va folosi un model Replicated Workers, asemanator cu modelul MapReduce folosit de inginerii de la Google pentru procesarea unor seturi mari de documente in sisteme paralele si distribuite. Acest articol prezinta modelul MapReduce folosit de Google si o parte dintre aplicatiile lui (mai importante pentru intelegerea modelului sunt primele 4 pagini).

MapReduce este un model de programare paralela (si implementarea asociata) pentru procesarea unor seturi imense de date folosind sute sau mii de procesoare. Modelul permite paralelizarea si distributia automata a taskurilor. Paradigma MapReduce se bazeaza pe existenta a doua functii care ii dau si numele: map si reduce. Functia map primeste ca input o functie f si o lista de elemente si returneaza o noua lista de elemente, rezultata in urma aplicarii functiei f fiecarui element din lista initiala. Functia reduce combina rezultatele obtinute anterior.

Mecanismul MapReduce functioneaza in modul urmator:

- utilizatorul cere procesarea unui set de documente; aceasta cerere este adresata unui proces (fir de executie) master;
- master-ul imparte documentele in fragmente de dimensiuni fixe, care vor fi asignate unor procese (fire de executie) worker; un worker va executa pentru un fragment de fisier o operatie numita “map”, care va genera niste rezultate partiale având forma unor perechi de tip *(cheie, valoare)*
- Dupa ce operatiile “map” au fost executate, master-ul asigneaza worker-ilor task-uri de tip “reduce”, prin care se combina rezultatele partiale.

2.2 Detalii tehnice privind cerintele temei

Dându-se un set de ND documente, sa se determine rangul fiecarui document conform metricii prezentate anterior si cuvantul/cuvintele de lungime maxima din fiecare document.

Operatiile de tip MAP

Pornind de la lista de documente de procesat ce va fi disponibila in fisierul de intrare, se determina dimensiunea fiecarui document si se creaza cate un task de tip **MAP** pentru fiecare fragment de cate *D* octeti dintr-un document (cu exceptia ultimului fragment, care poate fi mai scurt). Un task de tip **MAP** va avea urmatoarele informatii:

- Numele documentului
- Offset-ul de inceput al fragmentului din document
- Dimensiunea fragmentului

Observatie: In pasul de creare a task-urilor de tip **MAP** nu se va incarca in memorie continutul documentului de indexat.

Dupa ce au fost create task-urile de tip **MAP**, acestea se vor aloca unui thread pool pentru executare. Pentru fiecare task de tip **MAP** se va citi fragmentul de dimensiune *D* din document si se va construi un hash avand ca si chei lungimile cuvintelor gasite in fragment iar ca valori numarul de aparitii ale acestor cuvinte. Tot in etapa de **MAP** se va construi o lista de cuvinte maximale pentru acel fragment. Rezultatul unui task de tip **MAP** va fi

numele documentului impreuna cu hash-ul cu numarul de aparitii pentru lungimea cuvintelor gasite si lista de cuvinte maximale.

Se vor folosi ca separatori: ;:/?~\.,><~`[]{}()!@#$%^&-_+'=*"| spatiu/tab/endline

Observatie: Poate aparea problema ca fragmentul prelucrat de un worker sa se termine sau sa inceapa in mijlocul unui cuvânt. In cazul acesta, se va adopta urmatoarea conventie: daca fragmentul incepe in mijlocul unui cuvânt, worker-ul va "sari peste" acel cuvânt; daca fragmentul se termina in mijlocul unui cuvânt, worker-ul va prelucra si acel cuvant. in acest mod, cuvintele care sunt "la granita" dintre doua fragmente vor fi prelucrate tot timpul de worker-ul ce se ocupa de fragmentul care se afla in fisier inaintea cuvântului respectiv.

Operatiile de tip REDUCE

Avand rezultatele din cadrul operatiunii de MAP, un task de tip **REDUCE** va avea urmatoarele informatii:

- Numele documentului
- Lista de rezultate din cadrul operatiunii de MAP pentru acel document

Task-urile de tip **REDUCE** vor fi alocate pe un alt thread pool. Executia unui task **REDUCE** va consta din doua etape: etapa de combinare si etapa de procesare.

Etapa de combinare: consta in combinarea listei de rezultate partiale din etapa de MAP. Astfel, se vor combina hash-urilor cu lungimea cuvintelor astfel incat sa avem la final un singur hash ce reprezinta lungimea cuvintele impreuna cu numarul lor de aparitii pentru intreg documentul. In cazul listelor de cuvinte maximale, se vor combina cele care au cuvinte de dimensiunea cea mai mare per intreg documentul, restul cuvintelor vor fi decartate *(cuvintele local maximale)*.

Etapa de procesare: consta in calcularea rangului unui document folosind hash-ul din etapa de combinare si sirul lui Fibonacci {0, 1, **1, 2, 3, 5, 8, 13, 21, 34, ...**}

$$F_n = \begin{cases} 0 & ,n = 0 \\ 1 & ,n = 1 \\ F_{n-1} + F_{n-2} & ,n > 1 \end{cases}$$

Rangul documentului se va calcula folosind formula:

$$\text{Rang} = (\sum_{k=0}^{n} F(lungime_k + 1) * numar\_aparitii_k) / numar\_cuv\_total$$

, unde

- $F(lungime_k + 1)$ reprezinta valoarea din sirul lui Fibonacci a respectivei lungimi. *De exemplu, un cuvant de lungime de o litera are valoarea 1, unul de doua 2, unul de trei 3, unul de patru 5, unul de cinci 8, etc.*
- $numar\_aparitii_k$ reprezinta numarul de cuvinte din document care au lungimea egala cu $lungime_k$
- $numar\_cuv\_total$ reprezinta numarul total de cuvinte din documentul respectiv

Observatie: Afisati rangurile cu 2 zecimale, obtinute prin trunchiere (nu prin rotunjire).

Tot in cadrul acestei etape se va determina numarul de cuvinte diferite maximale din document.

2.3 Observatii generale

- rezultatele operatiilor MAP vor fi tinute in memorie; in mod normal ele s-ar fi scris si pe disc;
- ca mod de executie, se pot folosi (desi nu este obligatoriu) obiecte de tip "thread pool" care sunt deja implementate in Java (vezi interfata ExecutorService);
- pentru simplificare se pot utiliza mai multe thread pool-uri – de ex. unul pentru operatiile de tip MAP, unul pentru operatiile de tip REDUCE;
- dupa fiecare pas ce se executa in paralel pe un thread pool va trebui sa asteptati finalizarea tuturor task-urilor din thread pool inainte de trecerea la pasul urmator

3. Formatul datelor de intrare/ieşire.

Programul va primi ca argumente in linia de comanda: NT (numarul de thread-uri worker), numele unui fisier de intrare si numele unui fisier de iesire (in aceasta ordine).

Observatie: Se vor porni NT thread-uri in fiecare thread pool.

Fisierul ce contine datele de intrare are urmatorul format:

- pe linia I: dimensiunea D (in octeti) a fragmentelor in care se vor imparti fisierele
- pe linia II: numarul ND de documente de tip text de procesat
- pe urmatoarele ND linii: numele celor ND documente (câte unul pe linie)

Observatie: Toate fisierele de intrare vor contine doar caractere ASCII.

In fisierul de iesire, se vor afisa documentele in ordinea descrescatoare a metricii calculate, metrica fiecarui document, dimensiunea in cazul cuvintelor maximale, si numarul de cuvinte maximale diferite gasite in text.

Formatul fisierului de iesire este urmatorul:

DOCUMENT_1;rang_doc_1;[dimensiune_maximala_1,numar_cuvinte_maximale_1]

…

DOCUMENT_i;rang_doc_i;[dimensiune_maximala_i,numar_cuvinte_maximale_i]

Observatie: Daca doua (sau mai multe) documente au acelasi rang, se vor adauga in fisierul de iesire unul dupa altul in ordinea in care ele apar in fisierul de intrare.

4. Teste si punctare.

Pentru a verifica functionalitatea temei puteti folosi aceste teste.

Punctajul temei va fi distribuit astfel:

- Implementarea pasilor conform specificatiilor temei (60 puncte)
- Consistenta rezultate (la rulari multiple pe aceleasi date de intrare trebuie obtinute aceleasi rezultate) (40 puncte)

5. Continutul arhivei temei.

Arhiva temei va contine codul sursa pentru tema impreuna cu un fisier build.xml reprezentand pasii de compilare si impachetare pentru aplicatie pentru sistemul de build Ant (tutorial pentru Ant gasiti aici).

In urma rularii:

```
ant compile jar
```

va trebui sa se construiasca un fisier numit mapreduce.jar ce va putea fi rulat cu comanda:

```
java -jar mapreduce.jar $NT $INPUTFILE $OUTPUTFILE
```

6. Resurse (optional)

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Alt articol introductiv despre MapReduce

MapReduce on Wikipedia