Responsabili tema: Dragos Comaneci, Gabriel Gutu

Data publicarii: 03.11.2014

Termenul de predare: 21.11.2014

1. Introducere.

Pentru a facilita obtinerea rapida si precisa a informatiilor existente in paginile Web, motoarele de cautare indexeaza (colecteaza, parseaza si stocheaza) datele in avans. Procesul se numeşte Web indexing atunci cand se refera la indexarea paginilor existente in Internet.

Cerințele temei

In aceasta tema veti implementa un program paralel in Java pentru indexarea unui set de documente text primit ca input si apoi verificarea daca exista documente in set cu un grad de similaritate intre ele mai mare decat o valoare limita.

In urma procesului de indexare se determina numarul de aparitii al fiecarui cuvânt existent intr-un document, obtinându-se o lista de perechi (cuvânt, numar de apariții). Programul trebuie sa permita calcularea similaritatii semantice (sub forma de procent) intre toate documentele indexate si sa afiseze documentele cu grad maxim de similaritate.

Pentru paralelizarea indexarii sa 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 termenii si numarul de aparitii ale acestora. Pasul urmator il reprezinta combinarea hash-urilor (operatiunea de REDUCE) in urma caruia se obtine un hash ce caracterizeaza intregul document. Pentru fiecare document se vor calcula frecventele de aparitie ale cuvintelor dintr-un document, care vor reprezenta indexul documentului.

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.

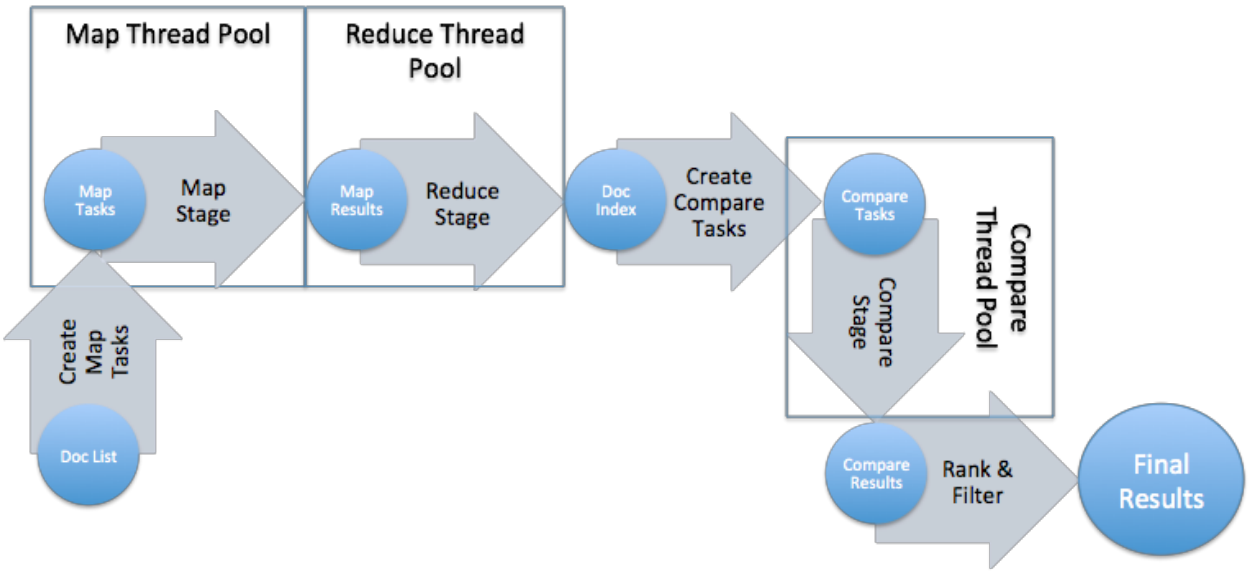
Mecanismul MapReduce functioneaza in modul urmator:

- 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 perechile de documente cu gradul de similaritate mai mare decât un prag X.

Pentru tema se va folosi ca referinta urmatoarea reprezentare schematica a pasilor si a rezultatelor dupa fiecare pas:



Pornind de la lista de documente de indexat 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 cuvintele gasite in fragment iar ca valori numarul de aparitii ale acestor cuvinte. Rezultatul unui task de tip MAP va fi acel hash cu numarul de aparitii pentru cuvintele gasite impreuna cu numele documentului.

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.

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 iar executia fiecarui task va consta in combinarea hash-urilor cu numarul de aparitii ale cuvintelor astfel incat sa avem la final un singur hash ce reprezinta cuvintele impreuna cu numarul lor de aparitii pentru intreg documentul.

Dupa executia task-urilor de tip REDUCE se vor crea task-uri de tip COMPARE (pentru fiecare pereche de documente) ce vor avea asociate urmatoarele informatii:

- nume document 1 si hash-ul cu numarul de aparitii al cuvintelor pentru documentul 1
- nume document 2 si hash-ul cu numarul de aparitii al cuvintelor pentru documentul 2

Observatie: Nu se vor crea task-uri de tip COMPARE in care se compara un document cu el insusi.

Task-urile de tip COMPARE se vor aloca pe un alt threadpool iar executia task-ului va consta in determinarea gradului de similaritate dintre cele doua documente.

Dupa executia task-urilor de tip COMPARE se vor sorta rezultatele in ordine descrescatoare in functie de gradul de similaritate si se vor pastra doar acele perechi de documente ce au gradul de similaritate peste pragul X.

Gradul de similariate intre doua documente se va calcula cu ajutorul formulei din acest articol:

sim(di, dj) = sum(f(t,di) * f(t,dj)) [%], unde t apartine lui V,

in care:

- di si dj sunt documentele ale caror grad de similaritate se doreste calculat
- f(t, di) reprezinta frecventa de aparitie a termenului t in documentul di
- V reprezinta vocabularul de termeni (se poate considera ca reuniunea termenilor din ele doua documente)

Frecventa de aparitie a unui termen intr-un document se calculeaza cu formula:

f(t, d) = (nr_aparitii_cuvânt(t, d) / nr_total_cuvinte(d)) * 100 [%],

in care:

- nr_aparitii_cuvânt(t, d) este numarul de aparitii ale termenului t in documentul d
- nr_total_cuvinte(d) este numarul total de cuvinte din documentul d

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 si unul pentru COMPARE;
- compararea intre cuvinte nu va fi case sensitive (veti transforma toate cuvintele in cuvinte cu litere mici inainte de a le prelucra);
- frecventele se vor considera cu 3 zecimale, obtinute prin trunchiere (nu prin rotunjire).
- 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 X reprezentând “pragul de similaritate” (ex.: vreau sa mi se returneze perechile de documente cu gradul de similaritate mai mare de X)
- pe linia III: numarul ND de documente de tip text de indexat si comparat
- 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 perechile de documente cu gradul de similaritate mai mare ca X - fiecare pereche pe câte un rand, in ordine descrescatoare a gradului de similaritate - impreuna cu gradul de similaritate.

Formatul fisierului de iesire este urmatorul:

DOCUMENT_1;DOCUMENT_2;sim(DOCUMENT_1, DOCUMENT_2)

…

DOCUMENT_i;DOCUMENT_j;sim(DOCUMENT_i, DOCUMENT_j),

4. Teste si punctare

Pentru a verifica functionalitatea temei puteti folosi aceste teste [TODO: add results for the tests]. Punctajul temei va fi distribuit astfel:

- Implementarea pasilor conform specificatiilor temei (40 puncte)
- Consistenta rezultate (la rulari multiple pe aceleasi date de intrare trebuie obtinute aceleasi rezultate) (40 puncte)
- Scalabilitate (20 puncte)
    - micsorarea timpului de rulare odata cu cresterea numarului de thread-uri din thread pool (se va valida ca t(NT=4) < t (NT=2) < t(NT=1) pentru 5 rulari consecutive pe acelasi set de date)
    - consumul de memorie nu va varia semnificativ in functie de dimensiunea D a fragmentului

Tema se va trimite prin intermediul interfetei vmchecker (disponibila in curand).

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

Algoritm pentru compararea textelor:

http://www.umiacs.umd.edu/~jimmylin/publications/Elsayed_etal_ACL2008_short.pdf