Responsabili temă: Gabriel Guțu, Dragoș Comăneci

Data publicării: 06.11.2013
Termenul de predare: 23.11.2013
Update 12.11.2013, ora 1:30 AM: eliminare limitare la "cele mai frecvente cuvinte"
Update 12.11.2013, ora 9:00 PM: adăugare recomandare privitoare la șirul de delimitatori
Update 16.11.2013, ora 10:50 PM:

- •modificare numar zecimale pentru trunchiere la 3 in loc de 2
- •eliminare recomandare privind sirul de tokenizare; folositi orice tokeni considerati a fi "natural" eliminati

Tip 17.11.2013, ora 07:05 PM: valorile numerice din output nu reprezintă un sistem de referință pentru notarea temei; ele sunt aproximative, diferind de șirul de delimitatori folosit și de modul în care se face prelucrarea cuvintelor "de margine". Vă sugerăm, totuși, să returnați niște valori corecte relativ la implementarea voastră.

1. Introducere. Cerinţele temei

Pentru a facilita obţinerea rapidă si precisă a informaţiilor existente in paginile Web, motoarele de căutare indexează (colectează, parsează si stochează) datele în avans, pentru a facilita procesele de căutare. Procesul se numeşte Web indexing atunci cand se referă la indexarea paginilor existente în Internet.
Căutarea unui cuvânt se poate face fie într-un document (sau o mulțime de documente) neindexat (dar, în general, într-un timp de execuție mult mai lung), ori într-un document (ori mulțime de documente) într-un timp mai rapid. Indexarea unui document se poate face în timp ce se execută o căutare în el pentru prima dată, ori se pot indexa în prealabil toate documentele. Pentru optimizarea timpului de căutare, motoarele de căutare cele mai populare fac indexarea totală a textului existent online (practic, indexarea și căutarea sunt pași complet separați).

Cerinţele temei

În aceasta temă se cere scrierea unui program paralel in Java care să realizeze indexarea unui set de documente primit ca input și apoi să verifice dacă un anumit document este plagiat, prin compararea similarității semantice a documentului curent vs. o serie de documente indexate.
În urma procesului de indexare se determină numărul de apariții al fiecărui cuvânt existent într-un document, obținându-se o listă de perechi (cuvânt, număr de apariţii). Programul trebuie să permită calcularea similarității semantice (sub formă de procent) între documentul primit ca parametru și toate documente indexate și să afișeze documentele cu grad maxim de similaritate.
Pentru paralelizarea indexării sa va folosi paradigma Replicated Workers (vezi Laborator 5) și modelul MapReduce. Fiecare document se va fragmenta în părţi de dimensiune fixă ce vor fi procesate în paralel (operațiunea de Map), pentru fiecare parte rezultând câte un vector parțial ce conține termenii și numărul de apariții ale acestora. Pasul următor îl reprezintă combinarea vectorilor (operațiunea de Reduce) în urma căruia se obține un vector ce caracterizează întregul document. Pentru fiecare document se vor calcula frecvențele de apariție ale cuvintelor dintr-un document, care vor reprezenta indexul documentului.~~Pentru ca un cuvânt din documentul verificat împotriva plagiatului să fie considerat relevant într-un calcul de similaritate, cuvântul respectiv trebuie să se afle în vectorul care conține termenii cu frecvențele cele mai mari de apariție al documetului comparat.~~

2. Implementare

2.1 Paradigma Map-Reduce - Prezentare generala

Pentru rezolvarea acestei probleme se va folosi un model Replicated Workers, asemănător cu modelul MapReduce folosit de inginerii de la Google pentru procesarea unor seturi mari de documente în sisteme paralele și distribuite. Acest articol prezintă modelul MapReduce folosit de Google și o parte dintre aplicațiile lui (mai importante pentru înțelegerea modelului sunt primele 4 pagini).
MapReduce este un model de programare paralelă (și implementarea asociată) pentru procesarea unor seturi imense de date folosind sute sau mii de procesoare. Modelul permite paralelizarea și distribuția

automată a taskurilor. Paradigma MapReduce se bazează pe existența a două funcții care îi dau și numele: map și reduce. Funcția map primește ca input o functie f și o listă de elemente și returnează o noua listă de elemente, rezultată în urma aplicării funcției f fiecărui element din lista inițială. Funcțiareduce combină rezultatele obtinuțe anterior.
Mecanismul MapReduce funcționează în modul următor:

- •utilizatorul cere procesarea unui set de documente; această cerere este adresată unui proces (fir de execuție) master;
- •master-ul împarte documentele în fragmente de dimensiuni fixe, care vor fi asignate unor procese (fire de execuție)worker; un worker va executa pentru un fragment de fisier o operație numită “map”, care va genera niște rezultate parțiale având forma unor perechi de tip (cheie, valoare);
- •Dupa ce operațiile “map” au fost executate, master-ul asignează worker-ilor task-uri de tip “reduce”, prin care se combină rezultatele parțiale.

2.2 Cerințe pentru problema propusă

Dându-se un set de ND documente și un document DOC de verificat împotriva plagiatului (prin calcularea gradului de similaitate cu celelalte documente), să se determine documentele cu gradul de similaritate mai mare decât un prag X.
Gradul de similariate între două documente se va calcula cu ajutorul formulei din acest articol:
sim(di, dj) = sum(f(t,di) * f(t,dj)) [%], unde t aparține lui V,
în care:

- •di și dj sunt documentele ale căror grad de similaritate se dorește calculat
- •f(t, di) reprezintă frecvența de apariție a termenului t în documentul di
- •V reprezintă vocabularul de termeni (se poate considera ca reuniunea termenilor din ele două documente)

Frecvența de apariție a unui termen într-un document se calculează cu formula:
f(t, d) = (nr_apariții_cuvânt(t, d) / nr_total_cuvinte(d)) * 100 [%],
în care:

- •nr_apariții_cuvânt(t, d) este numărul de apariții ale termenului t în documentul d
- •nr_total_cuvinte(d) este numărul total de cuvinte din documentul d ~~(în cazul problemei de față se poate considera ca fiind numărul de cuvinte cu cele mai mari frecvențe)~~

Pentru cerința descrisă mai sus se consideră:

- •[MAP]: Împărțirea fișierelor se va face în fragmente de câte D octeți (cu excepția ultimului fragment, care poate fi mai scurt)

Observație: Poate apărea problema ca fragmentul prelucrat de un worker să se termine sau să înceapă în mijlocul unui cuvânt. 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 cuvant. În acest mod, cuvintele care sunt "la graniță" dintre două fragmente vor fi prelucrate tot timpul de worker-ul ce se ocupă de fragmentul care se află în fișier înaintea cuvântului respectiv.
Astfel un task de tip “map”:

- •primește ca input: numele fișierului, poziția de unde începe să citească din fișier, și pozitia de sfârșit;
- •întoarce ca output: perechi de tip (cheie, valoare), în cazul problemei de față: (nume_document, vector_parțial). Vectorul parțial conține setul de cuvinte împreună cu numărul de apariții ale acestora.
- •[REDUCE]: Se calculează similaritatea semantică între documentul primit ca parametru și toate documentele indexate.

2.3 Observații generale

- •rezultatele operațiilor "map" vor fi ținute în memorie; în mod normal ele s-ar fi scris si pe disc;
- •ca mod de execuție, se pot folosi (deși nu este obligatoriu) obiecte de tip "thread pool" care

sunt deja implementate în Java (vezi interfața ExecutorService); astfel, un thread worker poate prelucra mai multe task-uri;
•pentru simplificare se pot utiliza mai multe thread pool-uri – de ex. unul pentru operațiile de tip "map", și unul pentru operațiile de tip "reduce";
~~•cuvintele pot fi de orice dimensiune si conțin doar litere;~~
~~•pentru tokenizare se recomandă folosirea șirului de delimitatori " \t\n\r\f"~~

•compararea între cuvinte nu va fi case sensitive (veți transforma toate cuvintele în cuvinte cu litere mici înainte de a le prelucra);
•frecvențele se vor considera cu 3 zecimale, obținute prin trunchiere (nu prin rotunjire).

3. Formatul datelor de intrare/ieşire.

Programul va primi ca argumente în linia de comandă: NT (numărul de thread-uri worker), numele unui fișier de intrare și numele unui fișier de ieșire (în această ordine).
Observație: Se vor porni NT thread-uri de tip Map, respectiv NT thread-uri de tip Reduce.
Fișierul ce conține datele de intrare are următorul format:

•pe linia I: numele documentului DOC pentru care se dorește determinarea gradului de plagiere
•pe linia II: dimensiunea D (în octeți) a fragmentelor în care se vor împărți fișierele
•pe linia III: numărul X reprezentând “pragul de similaritate” (ex.: vreau să mi se returneze documentele cu gradul de similaritate mai mare de X față de documentul D)
•pe linia IV: numarul ND de documente de tip text de indexat și în care se va face căutarea
•pe urmatoarele ND linii: numele celor ND documente (câte unul pe linie)

Observatie: Toate fisierele de intrare vor contine doar caractere ASCII.
În fișierul de ieșire, pentru documentul verificat se vor afișa numele documentelor cu gradul de similaritate mai mare ca X - fiecare nume pe câte un rand, în ordine descrescătoare a gradului de similaritate - împreună cu gradul de similaritate.
Formatul fisierului de iesire este urmatorul:
Rezultate pentru: nume_document_D
DOCUMENT_1 (sim(D, DOCUMENT_1))
…
DOCUMENT_P (sim(D, DOCUMENT_P)),
unde sim(D,DOCUMENT_i) > X, i = [1, … P]

4. Teste.

Pentru a verifica functionalitatea temei puteti folosi aceste teste.

5. Resurse (opţional)

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