UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare şi Tehnologia Informaţiei
SPECIALIZAREA: Calculatoare

Algoritmul MapReduce

TEMĂ DE CASĂ LA DISCIPLINA
ALGORITMI PARALELI ȘI DISTRIBUIȚI

Profesor îndrumător :

Ș.L. dr. ing. Adrian Alexandrescu

Student :

Lupascu Andrei, 1406B

Conținut

Enunțul temei de casă :

În cadrul oricărui sistem de regăsire a informaţiilor, colecţia de date ţintă este re-organizată (sau re-modelată) pentru a optimiza funcţia de căutare. Un exemplu în acest sens este dat chiar de motoarele de căutare a informaţiilor pe Web: colecţia de documente este stocată sub forma unui index invers.

Paşii implicaţi în construirea unui astfel de index invers sunt următorii:

- fiecare document din cadrul colecţiei ţintă (identificat printr-un docID) va fi parsat şi spart în cuvinte unice (sau termeni unici); se obţine în finalul acestui pas o listă de forma < $docID_x$, {$term_1$: $count_1$, $term_2$: $count_2$, . . . , $term_n$: $count_n$} > (index direct – $count_k$ înseamnă numărul de apariţii al termenului k);
- fiecare listă obţinută în pasul anterior este spartă în perechi de forma: < $docID_x$, {$term_k$: $count_k$} >; pentru fiecare astfel de pereche, se realizează o inversare de valori, astfel încât să obţinem: < $term_k$, {$docID_x$: $count_k$} >;
- perechile obţinute în pasul anterior sunt sortate după $term_k$ (cheie primară), $docID_x$ (cheie secundară);
- pentru fiecare termk se reunesc < $term_k$, {$docID_x$: $count_k$} >, astfel încât să obţinem: < $term_k$, {$docID_{k1}$: $count_{k1}$, $docID_{k2}$: $count_{k2}$, . . . , $docID_{km}$: $count_{km}$} > (**indexul invers**).

Tema de casă constă în implemenatrea unei soluţii MPI de tip MapReduce pentru problema construirii unui index invers pentru o colecţie de documente text.

Aplicaţia de test va primi ca parametrii de intrare numele unui director ce conţine fişiere text (cu extensia ”.txt”) şi un nume de director pentru stocarea datelor de ieşire şi va genera pe post de răspuns un set de fişiere text ce conţin indexul invers corespunzător colecţiei de documente de intrare.[1]

MapReduce. Noțiuni teoretice

MapReduce este o tehnică de procesare şi, în acelaşi timp, un model de programare pentru calculul distribuit. Termenul „MapReduce” referă, în prezent, un tipar de dezvoltare a aplicaţiilor paralele / distribuite ce procesează volume mari de date. În general, se consideră că acest model implică existenţa unui nod de procesare cu rol de coordonator (sau master sau iniţiator) şi mai multe noduri de procesare cu rol de worker.[1]

Etapele MapReduce [1]

Algoritmul MapReduce conţine 2 sarcini de lucru importante, şi anume „Map” (Maparea) şi „Reduce” (Reducerea):

- Etapa de mapare – preia un set de date şi îl converteşte într-un alt set de date, unde elementele individuale sunt „sparte” în tuple (adică perechi cheie / valoare)

    - nodul cu rol de coordonator împarte problema “originală” în sub probleme și le distribuie către workeri pentru procesare
    - trebuie reţinut faptul că această divizare a problemei de lucru (a datelor de procesat) se realizează într-o manieră similară divide-et-impera – în unele cazuri nodurile worker pot divide la rândul lor sub-problema primită şi pot trimite aceste subdiviziuni către alţi ; rezultă, în acest caz o arhitectură arborescentă;
    - divizarea caracteristică acestei etape nu trebuie să coreleze efectiv cu dimensiunea datelor de intrare cu numărul de worker-i din sistem; un worker poate primi mai multe sub-probleme de rezolvat;

- Etapa de reducere – preia ieşirea de la etapa de mapare ca fiind datele de intrare şi combină aceste tuple, rezultând un alt set de tuple, dar de dimensiuni mai mici.

    - Nodul cu rol de coordonator (sau set de noduri cu rol de worker “desemnat” de coordonator) colectează soluțiile sub-problemelor si le combină pentru a obține rezultatul final al procesării dorite.

Etapele algoritmului, conform Michael Kleber: [1]

Michael Kleber (Google Inc.) rafinează etapele implicate de paradigma MapReduce după cum urmează:

- Pre-procesare – datele sunt pregătite pentru funcția de mapare;
- Mapare – stabilirea datelor de interes;
- Amestec și sortare – datele pot fi organizate astfel încât să fie optimizată etapa de reducere;
- Reducere – determinarea rezultatului;
- Stocare rezultat.

Avantajele algoritmului

Avantajul major al algoritmlui MapReduce este faptul că este mai ușor de scalată procesarea de date peste mai multe noduri computaționale. Sub modelul MapReduce, primitevele de procesare de date sunt denumite mapatori și reductori. Descompunerea unei aplicaţii pentru procesare de date în mapatori şi reductori este, uneori, non-trivială. Dar, odată ce scriem o aplicaţie în forma MapReduce scalarea acelei aplicaţii pentru a se putea executa peste sute, mii, sau chiar zeci de mii de maşini într-un cluster reprezintă doar o chestiune de modificarea unei configurări. Această scalabilitate simplă a atras atenţia multor programatori în a folosi modelul MapReduce.[2]

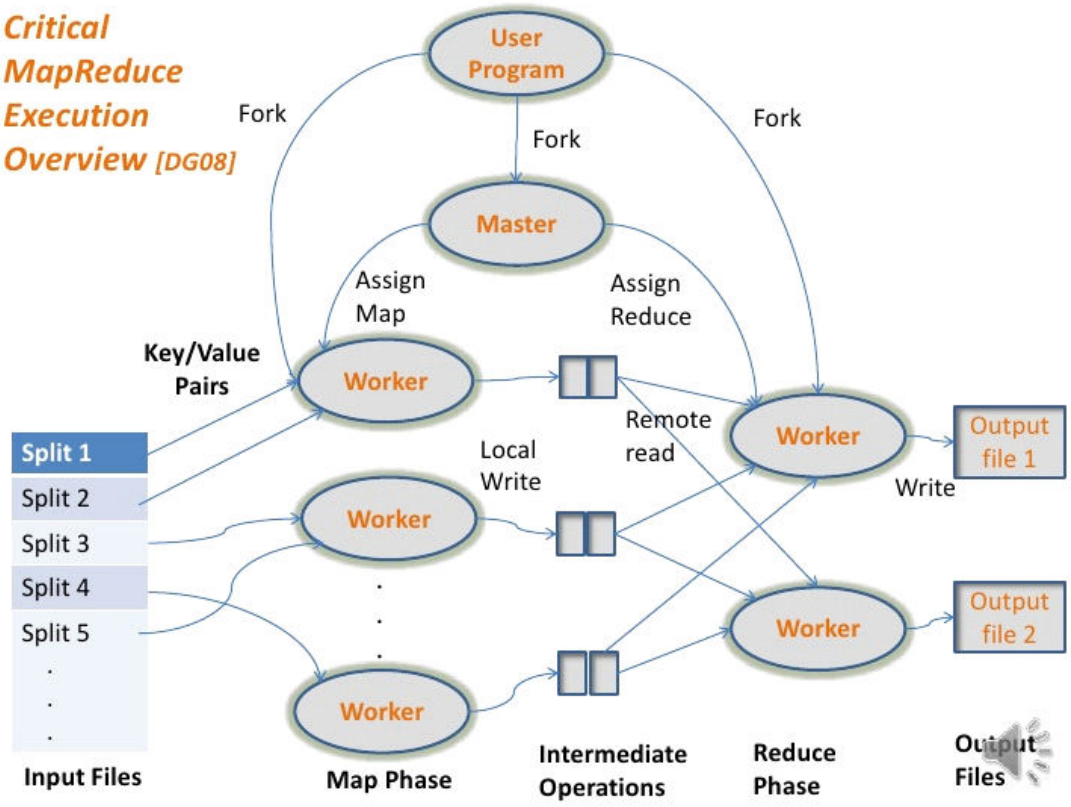

Figura 1. Paradigma MapReduce ([3])

MapReduce Pseudocod in cazul general [4]

Mapper-ul :

```
map(String key, String value)
// key: document name
// value: document contents
for each word w in value
        EmitIntermediate(w, "1")
```

Reducer-ul :

```
reduce(String key, Iterator values):
// key: word
// values: a list of counts
 for each v in values:
        result += ParseInt(v);
        Emit(AsString(result));
```

Exemplu:

În următorul exemplu, vom afla frecvența cu care apar cuvintele in diferite fișiere, folosind modelul MapReduce.

MapReduce va fi folosit pentru a număra aparițiile unui cuvânt dintr-un text. La intrare avem 3 fișiere cu câte 3 cuvinte fiecare, deci pentru fiecare fișier se va crea un bloc separat, în etapa de pre-procesare. În etapa de mapare, pentru fiecare element se creeaza o tuplă având cheia și valoarea acesteia, adică numărul de apariții. Apoi se sorteaza în funcție de cheie, pentru ca etapa de reducere să fie optimizată. La reducere, urmează doar să adunăm pentru fiecare cheie numărul de apariții și astfel obținem rezultatul ca în figura de mai jos.[5]

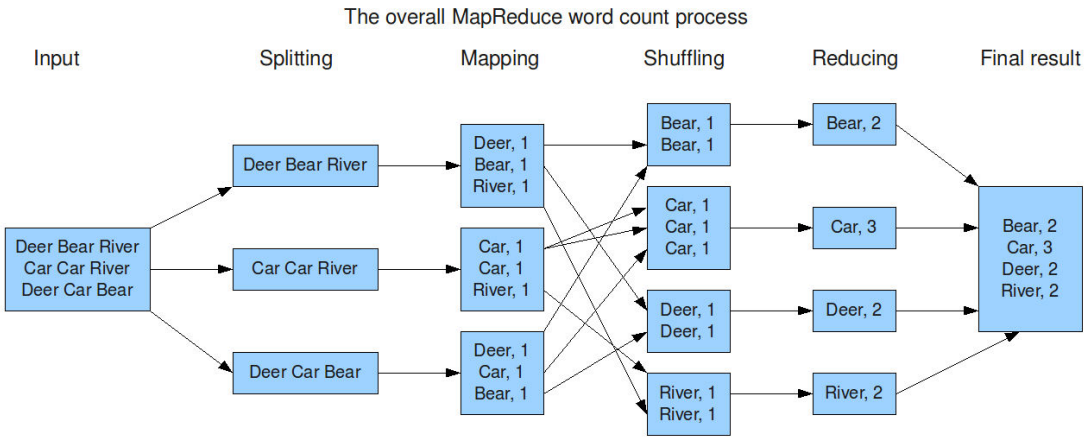

Figura 2. Exemplu [5]

Cod sursa:

https://github.com/andreiLupascu/alpd/tree/master/ALPD

Versiunea Python folosita: 3.7.4

Diferente intre algoritmul conform Kleber si cel prezentat / caracteristici cod:

-codul prezentat are nevoie de minim 2 procese pentru a lucra, procesul 0 fiind "masterul", impartind treaba intre procesele "slave".

-cuvintele din indexul direct se scriu intr-un al 3lea director de unde apoi sunt citite pentru crearea indexului invers, nu sunt tinute in memorie si folosite ca mesaje intre procese.

-procesul 0 foloseste un array de elemente cu nr total de procese active - 1, daca array[i] = 0 inseamna ca procesul i+1 inca lucreaza, altfel ca a terminat. Cand acest array e format doar din 1-uri stie ca poate trece la stagiul urmator.

-procesul 0 functioneaza in 3 stagii:

-0: procesul 0 trimite catre toate celelalte procese flagul "stage" si nr. de fisiere pe care trebuie sa le proceseze fiecare, apoi acestea fac indexul direct pe acele fisiere

-1: procesul 0 a primit de la toate procesele flagul ca toate fisierele de test initiale au fost creeate si indexul direct exista pentru fiecare cuvant per fisier, trimite flagul de start catre toate procesele sa treaca la stage=1, unde acestea primesc numarul total de fisiere din indexul direct respectiv cate trebuie fiecare sa proceseze, si apoi scriu indexul invers in directorul ales

-2: stage = 2 procesul 0 trimite flag-ul de finalizare catre toate celelalte procese apoi finalizeaza si el lucrul

Bibliografie

[1] Explicații laborator ALPD

[2] https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

[3] https://www.slideshare.net/diliprk/mapreduce-paradigm

[4] https://cs.stackexchange.com/questions/14470/mapreduce-pseudocode

[5] https://www.todaysoftmag.ro/article/1309/introducerea-si-tuning-ul-hadoop-mapreduce

[6] https://mpi4py.readthedocs.io/en/stable/

[7] Documentatii la diverse librarii python built-in pentru I/O