UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

DISCIPLINA:

REȚELE DE CALCULATOARE ȘI INTERNET

TEMA:

HADOOP și MAPREDUCE – SOLUȚIE PENTRU PROCESAREA DISTRIBUITĂ A VOLUMELOR MARI DE DATE ÎN REȚELE DE CALCULATOARE

STUDENT: CLAUDIA FLORINELA CHIȚU

MASTER IISC, AN 2

Cuprins

1. Noțiuni introductive

Hadoop este un framework open source foarte răspândit pentru Cloud Computing și are ca scop procesarea distribuită a volumelor mari de date (petabytes de date). Hadoop este un framework de dezvoltare de software realizat în limbajul de programare Java.

O caracteristică importantă a frameworkului Hadoop este reprezentată de partiționarea datelor și procesarea pe mii de host-uri și execuția în paralel pe calculatoare diferite. Hadoop este caracterizat și de o scalabilitate foarte bună pentru capacitatea de calcul, de stocare prin folositea de servere commodity. Yahoo folosește Hadoop pentru stocarea a 40 petabytes de date cu un cluster de 4000 de servere. Yahoo este doar un exemplu, însă multe alte sute de organizații[1] folosesc Hadoop pentru dezvoltarea afacerilor. Componența Hadoop este următoarea:

-Pachetul Common
-HDFS
-MapReduce

Prima componentă cuprinde biblioteci și utilitare folosite mai departe de modulele Hadoop.

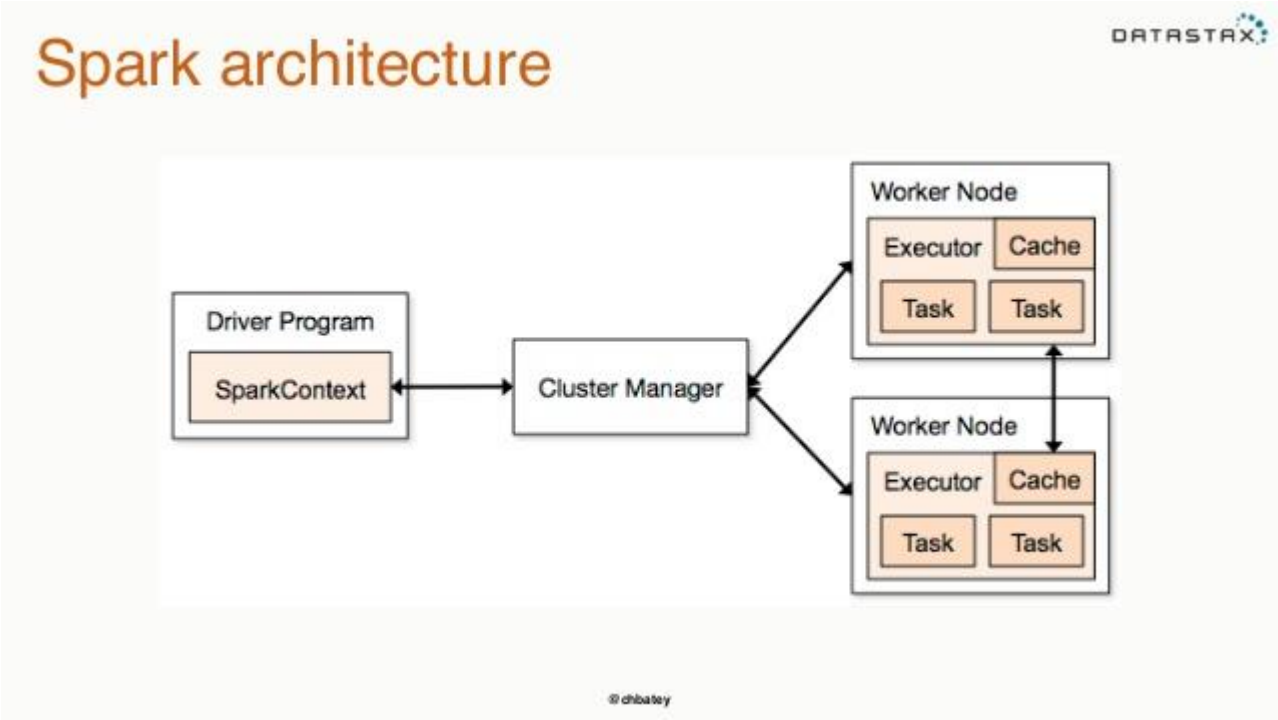
HDFS (Hadoop Distributed File System) este un sistem de fișiere scrise în Java pentru frameworkul Hadoop. Acest sistem este scalabil, distribuit și portabil. Un cluster Hadoop are un nod nume (namenode) și un cluster de noduri de date (datanodes). Fiecare nod de date își aduce aportul în punerea datelor pe rețea folosind un protocol specific sistemului HDFS. Sistemul folosește socheții TCP/IP pentru comunicație, în timp ce clienții folosesc o procedură remote (RPC) pentru a comunica între ei. Conform [2], scopul principal al HDFS este stocarea datelor într-un mod fezabil/fiabil inclusiv în momente critice cum ar fi eșuarea Namenode-urilor, DataNode-urilor sau a partițiilor de rețea.

Conform [3], **MapReduce** este un subproiect al proiectului Hadoop de la Apache. Acest framework, se ocupă de **programarea task-urilor**, **monitorizarea lor** și reexecutarea în cazul eșuarii. Scopul principal al acestui framework este împărțirea seturilor de date de intrare în seturi independente pentru procesare paralelă. Practic, se sortează seturile de date de ieșire, care devin apoi date de intrare pentru reducerea taskurilor. Atât datele de intrare, cât și cele de ieșire sunt stocate în fișiere ale sistemului.

Hadoop nu este singura coluție pentru cazuri de big data în sisteme distribuite. Două alte exemple sde soluții unt Storm (http://storm.apache.org/) și Spark (http://spark.apache.org/) .

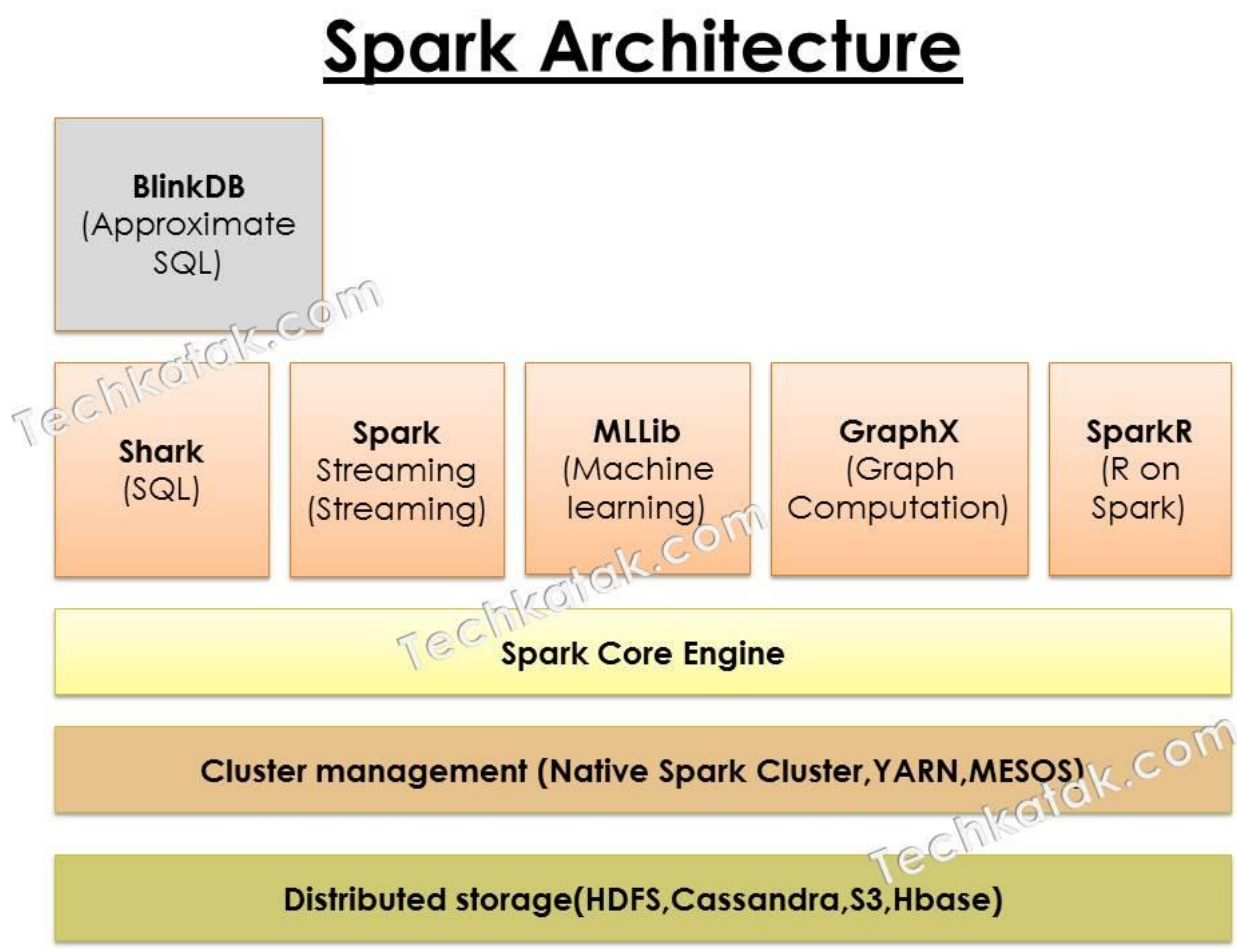
Spark este cunoscut ca un motor rapid și general pentru Big Data. **Generalizează modelul MapReduce** și poate chiar înlocui acest motor. Acest lucru se traduce prin faptul că Spark trebuie să lucreze bine cu date de mărime de ordin GBs până la PBs, să lucreze bine cu algorimti complecși, de la ETL (Extract Transform Load) până la SQL și machine learning, să gestioneze datele bine fără diferențe de mediu (disc, SSD-storage solid drive, etc.). În 2014, Spark este cel mai activ proiect Big Data. Spark poate depăși performanța MapReduce Hadoop atunci când data este în memorie (https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html).

Spark este competitiv cu Hadoop și *superior pentru sarcini de machine learning* pentru că acesta este scopul în care a fost proiectat. Dar Hadoop *MapReduce este superior lui Spark pentru sarcini de ETL.* Hadoop este superior și în cazul când este necesară o comunicare între entitățile de calcul.

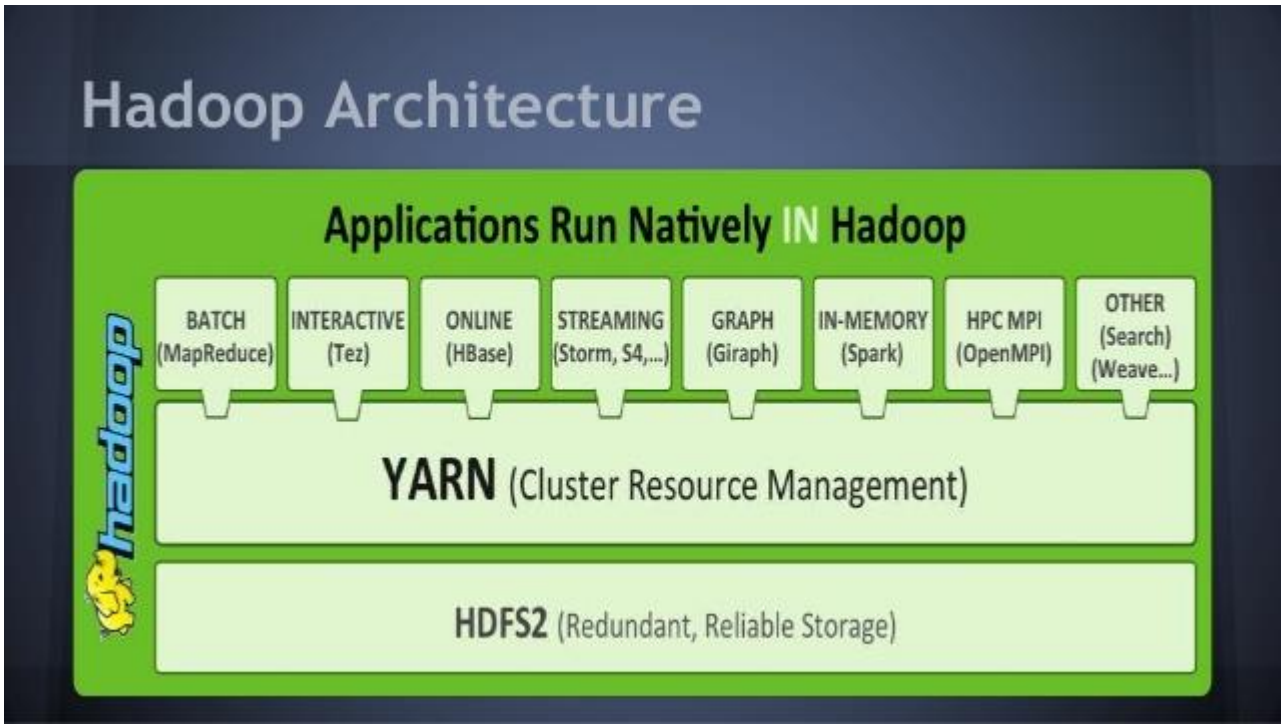


Cum se poate observa și în imaginea de deasupra, de pe site-ul http://www.slideshare.net/chbatey/reading-cassandra-meetup-feb-2015-apache-spark , Spark se folosește de cache pentru manipularea datelor. Spark este mai rapid de 10 ori pe disc și de 100 de ori în memorie decât este Hadoop. Volume mari de date sunt divizate în părți mici și procesate local înainte de a fi combinate cu celelalte părți. Spark are un **modul**

de machine learning dedicat ceea ce îi oferă avantajul în fața lui Hadoop în ceea ce privește viteza de lucru. Acest aspect este redat în următoarea comparație grafica:



(sursa: http://techkatak.com/spark-for-fast-batch-processing/)



(sursa: http://image.slidesharecdn.com/dinatechtalk-150201035119-conversion-gate01/95/josa-techtalks-big-data-on-hadoop-7-638.jpg?cb=1440319902)

Hadoop prezintă avantajele că este mai popular și are o comunitate mai mare, având nume mari în spate (Cloudera etc.); ca atare sunt mai multe persoane –ingineri – care au lucrat și dezvoltă clustere pentru Hadoop. Poate că și integrarea în arhitectură este mai ușoară la Hadoop decât la Spark, dar aceasta este o discuție separată care poate fi larg dezbătută și nu face obiectul acestui studiu.

2.HDFS

Un aspect care nu a fost menționat în primul capitol, este că în sistemele *HDFS, replicile sunt stocate separat* pentru nodurile de date pentru a mări disponibilitatea datelor[4]. Astfel, atunci când se detectează că un nod de date a eșuat, se apelează la reconstrucția replicii, moment în care performanța procesării pentru sistemele distribuite scade. Autorii articolului [4] propun reconstruirea pe baza replicii pentru clustere HDFS multi-rack și evaluarea eficienței, cu ajutorului mediului de simulare. În schemele propuse de ei, transferul de date în rack este realizat cu o structură unidirecțională ring, iar transferul datelor între rack-uri este realizat cu ajutorul metodei robin. Astfel, ei obțin o îmbunătățire

a toleranței la erori și un timp de execuție mult mai bun (16% reducere comparativ cu al schemelor standard).

Hadoop lucrează direct cu sisteme de fisiere distribuite, iar pentru reducerea traficului pe rețea, Hadoop trebuie sa cunoască informații despre servere astfel încât să știe care servere sunt cele mai aproape de date.

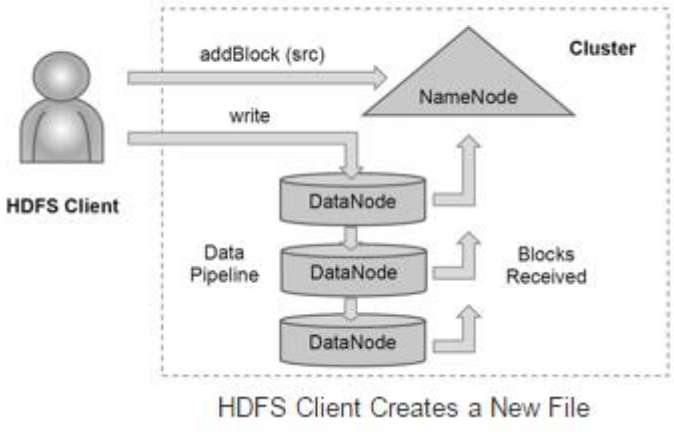
Deasupra sistemelor de fișiere vine motorul MapReduce, ce constă într-un JobTracker căruia aplicațiile client îi trimit job-urile MapReduce. Cu fișierele de sistem rack, JobTracer-ul cunoaște care nod conține date și care mașini sunt cele mai aproape de aceste date. Dacă aplicația nu poate fi găzduită de actualul nod unde sunt datele, atunci prioritatea este pasată către nodurile din același rack. Astfel **se reduce traficul de rețea**. Dacă taskul tracker-ului eșuează sau dă timeout ca răspuns, atunci job-ul este regestionat. Un semnal de keep alive este trimis de către task tracker către job tracker la fiecare câteva minute pentru a verifica statusul. Statusul lor este expus de Jetty[5] și poate fi vizualizat în web browser.

Pentru gestionare*, Hadoop folosește gestionarea de tip FIFO*, dar are și opțiunea de a folosi un mecanism de gestionare alternativ; de exemplu Fair scheduler sau Capacity scheduler.

Primul tip de mecanism de gestionare a fost dezvoltat de Facebook pentru a obține un timp de răspuns mai mic pentru toate job-urile mici. Astfel, job-urile sunt grupate în pool-uri, niște seturi de resurse pregătite pentru utilizare. Deci, pentru sarcini mici de execuție, se dorește un timp de răspuns mic, iar pentru sarcini de producție se dorește calitate cât mai bună pentru servicii.

Al doilea tip de mecanism, capacity scheduler, a fost dezvoltat de Yahoo. Acesta suportă mai multe caracteristici, asemănătoare cu cele ale fair scheduler-ului. Cozilor le sunt alocate o parte din capacitatea toatală a resurselor. Resursele rămase disponibile sunt alocate cozilor care nu au fost incluse înainte. Iar într-o coadă, un job cu o prioritate mare are acces la resursele cozii.

Pentru o înțelegere mai ușoară a conceptului, apelez la următoarea imagine:



Sursa: http://www.aosabook.org/en/hdfs.html

HDFS este structurat într-*o ierarhie de fișiere și foldere*, reprezentate de *inodes*. Acestea conțin atribute precum permisiuni, modificări, timp de access, spațiul de pe disc etc. Conținutul fișierului este împărțit în blocuri (de obicei 128 megabytes) și fiecare bloc din fișier este replicat independednt la nivelul datanodes (într-un arbore de obicei). Acest design are un singur namenode pentru fiecare cluster. Clusterul poate avea până la mii de datanodes și zeci de mii de clienți HDFS pentru un cluster pentru că fiecare datanode poate executa multiple sarcini ale aplicației în mod concurent.

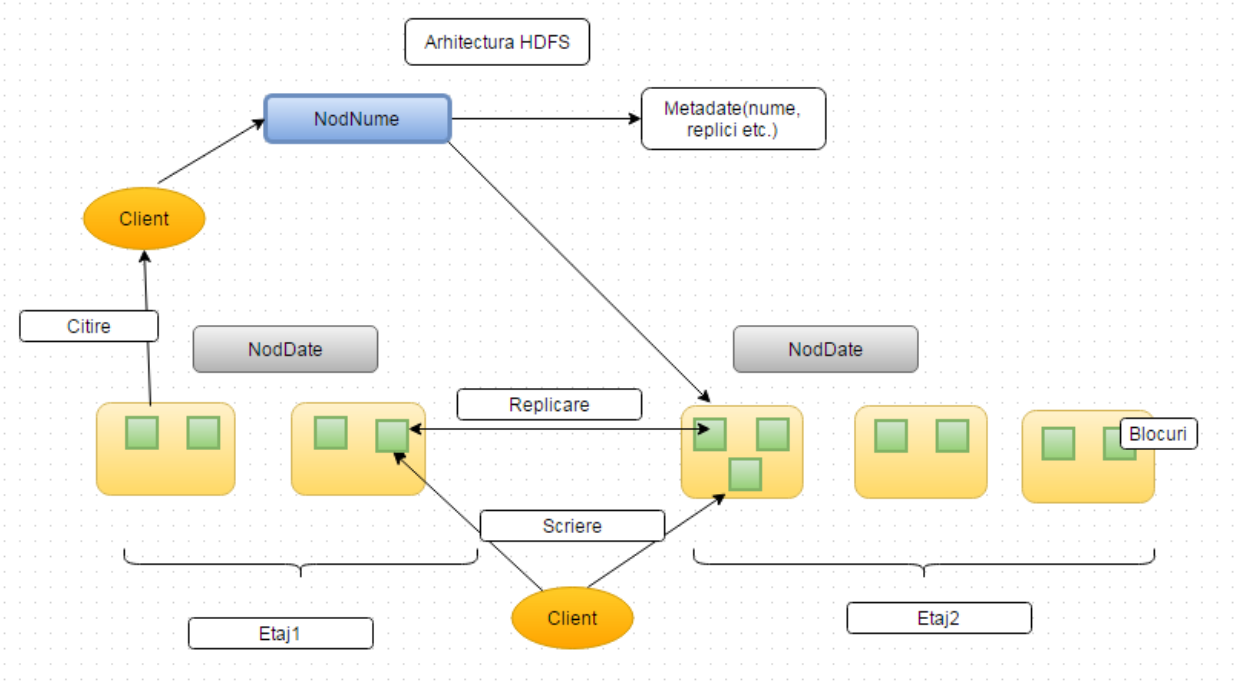
Aplicațiile user accesează fișierele de sistem prin intermediul clientului HDFS, a bibliotecii care exportă fișiere HDFS de interfață. HDFS suportă **operații de citire, scriere și ștergere** de fițiere și scriere și ștergere de foldere.

Imaginea poate fi explicată astfel: atunci când o aplicație citește fișiere, clientul HDFS întreabă namenodul de lista de datanodes care găzduiește replicile blocurilor de fișiere. Lista este sortată în funcție de distanța topologiei de rețea. Clientul trimite datele după ce face un pipeline de la nod la nod. Când primul bloc este umplut, clientul cere noului datanode să aleagă o altă gazdă pentru replica următorului bloc. Se face iar pipeline, și clientul trimite mai departe datele. Imaginea ilustrează de fapt interacțiunea dintre client, namenode și datanode. HDFS folosește API pentru a arăta locațiile blocurilor din fișiere. Acest lucru permite aplicațiilor precum MapReduce să gestioneze task-uri și astfel să se îmbunătățească performanța de citire.

O trăsătură importantă a HDFS-ului este aceea că prezintă checkpoint-uri. Crearea periodică a checkpointurilor este un mod de a proteja metadatele fișierelor de sistem.

2.1. Arhitectura HDFS

Arhitectura HDFS este una de tip master/sclav. Un cluster HDFS constă într-un singur NodNume, un server master care gestionează spațiul fișierelor de sistem și accesul clienților la fișiere. Mai există și NodDate pentru gestiunea stocării aferente nodului. HDFS utilizează un spațiu de fișiere de sistem pentru a stoca datele utilizatorului. Intern, un fișier este împărțit în unul sau mai multe blocuri care sunt stocate în NodDate. NumeNod execută operații de fișiere de sistem precum deschidere, închidere, redenumire fișiere și foldere. NodDate este responsabil de cererile de scriere și citire ale fișierelor de sistem ale clienților. NodDate se ocupă și de crearea blocurilor, ștergerea și replicarea instrucțiunilor de la NodNume. Acest flux este prezentat în imaginea următoare:



NodNume și NodDate sunt părți de software destinate să ruleze pe *mașini commodity*. Acest lucru înseamnă că ele folosesc un număr mare de componente deja disponibile pentru calcul paralel, pentru a obține cea mai mare putere de calcul cu cel mai mic cost. Existența unui singur NodNume simplifică mult arhitectura sistemului. NodNume este locul de stocare pentru toate metadatele HDFS.

HDFS este proiectat astfel încât să stocheze în mod fezabil pe mașini fișiere foarte mari într-un cluster mare. Stochează fiecare fișier ca o secvență de blocuri. Toate blocurile dintr-un fișier exceptând ultimul, au aceeași mărime. *Blocurile unui fișier sunt replicate pentru toleranță la erori*. Mărimea unui bloc și factorul de replicare sunt configurabile pentru fiercare fișier. O aplicație permite specificarea numărului de replici ale fiecărui fișier.

Factorul de replicare poate fi specificat la crearea fișierului dar poate fi schimbat ulterior. Fișierele în HDFS au restricția că pot fi scrise doar de un autor la un moment de timp dat.

NodNume ia toate deciziile cu privire la replicarea blocurilor.

Periodic primește un puls care este numir ”heartbit” și oferă informații despre starea DataNodului, dar și un raport cu informații de la blocuri. Daca DataNodul trimite acel puls, înseamnă că funcționează. Replicarea nu are loc atunci când NodNume este în starea ”safemode”. Un bloc este considerat replicat în siguranță atuni când numărul minim de replici sunt verificate cu NodNume (plus alte 30 secunde de siguranță). Nodul Nume folosește un fișier de log numit EditLog pentru a păstra evidența înregistrărilor produse în sistem. Toate fișierele sunt stocate într-un fișier mai amplu numit FsImage. Acesta este stocat în sistemul de fișiere local al lui NodNume.

Când un NodNume pornește, el citește din memorie FsImage și EditLog de pe disc aplică tranzacțiile din Editlog în memoria reprezentată în FsImage și noua versiune este scrisă în FsImage pe disc. Acest proces este numit checkpoint.

Când un NodDate pornește, el caută prin fișierele de sistem locale, generând o listă cu toate blocurile de date HDFS corespunzătoare cu fișierele și trimite raportul la NodNume. Acesta este raportul de bloc.

Toate protocoalele de comunicație sunt deasupra protocolului TCP/IP. Un client stabilește conexiunea cu un port TCP configurabil pe mașina unde este NodNume. NodDate comunică cu NodNume folosind NodDate protocol.

Scopul Hadoop este procesarea paralelă a datelor. Pentru a face acest lucru, sunt necesare mașini care să lucreze cu datele în același timp. Deci, clientul va împărți fișierele de date în părți mai mici numite blocuri și va distribui pe mașini aceste blocuri. Pentru a evita pierderea datelor, în cazul eșecului unei mașini, trebuie să existe blocurile pe mai multe mașini. Deci, atunci când este încărcat, un bloc este și replicat. Standardul Hadoop este să existe 3 copii ale fiecărui bloc din cluster. Acest lucru se configurează modificând parametrii din fișierul hdfs-site.xml. Clientul împarte fișierul în blocuri . Pentru fiecare bloc consultă NodNume și primește o listă de NodDate care fiecare ar trebui să conțină o copie a blocului. Apoi clientul scrie blocul direct în NodDate. Nodul care primește datele replică blocul în alte NodDate și se repetă procedeul pentru toate celelalte blocuri rămase. NodNume doar furnizează harta unde se găsesc datele care ar trebui să meargă în cluster.

Regula pentru HDFS este ca pentru fiecare bloc de date, 2 copii să existe pe același etaj și alta într-un etaj diferit.

Paralelizarea este posibilă pentru că fiecare nod are indexul lui în fișier și fiecare client care scrie are fișierul său de log.

2.2. HDFS vs. NFS

Un sistem de fișiere distribuite este proiectat să gestioneze un număr mare de date și să furnizeze acces la date unor clienți distribuiți într-o rețea. Această situație poate fi implementată în mai multe moduri.

NFS, Newtwork File System este cel mai întâlnit sistem de fișiere distribuite și printre cel mai vechi folosit. NFS furnizează acces la un singur volum logic stocat pe o singură mașină. Un server NFS face o parte din sistemul său de fișiere să fie vizibil clienților externi. Apoi clienul poate monta fișierul de la distanță și poate interacționa cu acesta ca și cum ar fi parte din sistemul local.

Un avantaj al NFS este transparența: clienții nu sunt constrânși de faptul ca fișierele sunt stocate la distanță.

Dar ca un sistem de fișiere distribuite, are o putere limitata: toate fișierele în NFS sunt stocate pe o singură mașină. Acest lucru înseamnă că pot fi stocate atât de multe fișiere cât încap pe o mașină și nu este garantat faptul că totul va fi ok dacă mașina se închide sau se întâmplă vreun dezatru. Dacă există mai mulți clienți, atunci serverul poate fi supraîncărcat pentru că toți clienții vor accesa singura mașină. De altfel, clienții trebuie să își copieze datele mereu pe mașina locală pentru a opera cu ele.

Comparativ cu NFS, **HDFS,** Hadoop Distributed File System, este proiectat să fie robust la un număr de probleme la care NFS sau alte DFS (distributed file sistem) sunt vulnerabile. În particular, HDFS este conceput să:

- stocheze un număr mare de informații (terabytes sau petabytes). Acest lucru presupune distribuirea datelor de-a lungul unui număr mare de mașini. De asemenea, suportă fișiere mult mai mari decât NFS.

-stocheze datele în mod fezabil și eficient. Dacă mașinile individuale funcționează într-un mod care nu este propice, atunci datele trebuie să fie totuși disponibile.

-furnizeze acces scalabil și rapid la date. Trebuie să fie posibilă servirea mai multor clienți doar prin simpla adăugare la cluster a mai multor mașini.

-se integreze bine cu Hadoop MapReduce permițând datelor să fie citite și folosite local atunci când este nevoie.

Deși HDFS este foarte scalabil, performanța design-ului este restricționată la niște clase particulare de aplicații. Nu este atât de general precum NFS. *HDFS este proiectat să optimizeze performanța fluxurilor de citire*. Datele care sunt scrise în HDFS și apoi citite de mai multe ori și închise nu pot fi

actualizate (se aplează în acest caz la un suport pentru a adăuga noi date la sfârșitul fișierelor). Sistemul HDFS nu permite un mecanism de caching local al datelor din cauza fișierelor mari și a naturii secvențiale de citire a fluxurilor de date. Pe de altă parte, replicarea datelor combate problema dezatrelor pe mașini chiar în cazul simultan al mai multor mașini.

3.MapReduce

3.1. Descriere

MapReduce este un model de programare și este implementat pentru procesarea datelor. Hadoop are capabilitatea de a executa programe Map Reduce care sunt scrise și în limbaje de programare diferite.

MapReduce este un nume sugestiv care provine de la modul de funcționare al motorului. Astfel, primul pas este cel de mapare – o procedură de mapare care realizează filtrarea și sortarea, iar al doilea, cel de reducere –calculul frecvențelor etc.

Fluxul de date de intrare format dintr-o cheie și o valoare asociată este transformat de **Map** într-o altă pereche de cheie și valoare, dar de ieșire. Nodul master preia input-ul, îl segmentează în subtask-uri pentru a le distribui celorlalte noduri. Celelalte noduri pot replica comportamentul într-o strctură sub formă de arbore, pe mai multe niveluri. După ce este procesat taskul, se trimite un răspuns către nodul master. *Operațiile de mapare sunt independente, aspect care explică paralelizarea*, dar aceatsa este dependentă de numărul de procesoare din vecintatea fiecărei surse.

Tranformarea **Reduce** are rolul de a prelua toate valorile unei chei specificate și de a genera rezultatul redus, dar cu o nouă listă. Reducerea se poate realiza într-o manieră similară fazei de mapare. Avantajul folosirii acestui tip de algoritm este ilustrat de faptul că un set de servere este capabil să sorteze un petabyte de date doar în câteva ore, spre deosebire de algoritmii secvențiali.

MapReduce poate fi privit și dintr-un alt unghi, și anume:

-pregătirea intrării de Map, când se asignează procesoare de mapare, se asignează valoarea cheii cu index1 pe care va lucra primul procesor, și furnizarea de date necesare pentru cheia cu index1 pentru procesorul asociat.

-rularea codului Map pentru generarea ieșirii care va avea cheia cu index2

-împărțirea mapărilor pentru procesoarele de Reduce pentru asigurarea faptului că fiecare procesor are toate datele necesare asociate cheii cu index2

-rularea codului de Reduce pentru a se îndeplini condiția ca reducerea să fie executată exact o singură dată pentru fiecare cheie cu index2

-producerea rezultatului final când sistemul MapReduce colectează ieșirile produse la pasul de reducere.

MapReduce este foarte util pentru aplicații precum sortare distribuită, web link-graph reversal, descompunere în valori singulare, indexare inversă, clasificare de documente, machine learning și statistical machine translation [6].

3.2. Aplicație

3.2.1. Descrierea algoritmului

Pentru implementarea interfeței MapReduce sunt posibile mai multe opțiuni, depinzând de mediul de lucru. De exemplu, nu se poate folosi aceeași soluție pentru o mașină cu o memorie distribuită mică și pentru o colecție mare de mașini în rețea.

În continuare este prezentată soluția folosită de Google: clustere mari de calculatoare conectate împreună printr-un Ethernet Gigabit. Utilizatorii trimit job-uri într-un sistem de gestionare. Fiecare job constă într-un set de task-uri și este mapat de mecanismul de gestionare la un set de mașini disponibile în cadrul clusterului.

Maparea este distribuită pe mai multe mașini prin partiționarea automată a setului de intrare în M seturi. Acest set de intrare poate fi procesat în paralel pe diferite mașini. Reducerea este distribuită și se face prin partiționarea intermediară a spațiului cheilor în R părți folosind funcții de partiționare (de exemplu, key hash mod R etc.). Aceste aspecte precum numărul de funcții sau de partiții sunt specificate de utilizator.

Pentru implementarea efectivă, utilizatorul apelează funcția MapReduce, apel la care se produce următoarea secvență de acțiuni:

1. Biblioteca MapReduce împarte fișierele de intrare în M părți de obicei în 16-64MB pentru fiecare parte, după care începe crearea unor copii ale programului pe un cluster de mașini.

2. Una dintre copiile programului se numește master și are niște proprietăți speciale. Restul copiilor primesc sarcini de la master. Pe lângă aceste două clase de entități mai sunt și M task-uri de mapare și R task-uri de reducere.

3. Una din copiile care primesc sarcini de la master, numită și sclav, citește conținutul corespunzător fișierului de intrare asignat. Acesta parsează perechea cheie-valoare a intrării, iar valoarea intermediară a funcției de mapare este stocată în memorie.

4. Periodic, perechile stocate sunt scrise local pe disc și partiționate în R regiuni de către funcția de partiționare. Aceste locații de pe disc sunt trasnmise masterului responsabil de trimiterea acestor locații la sclavii dedicați operației de reducere.

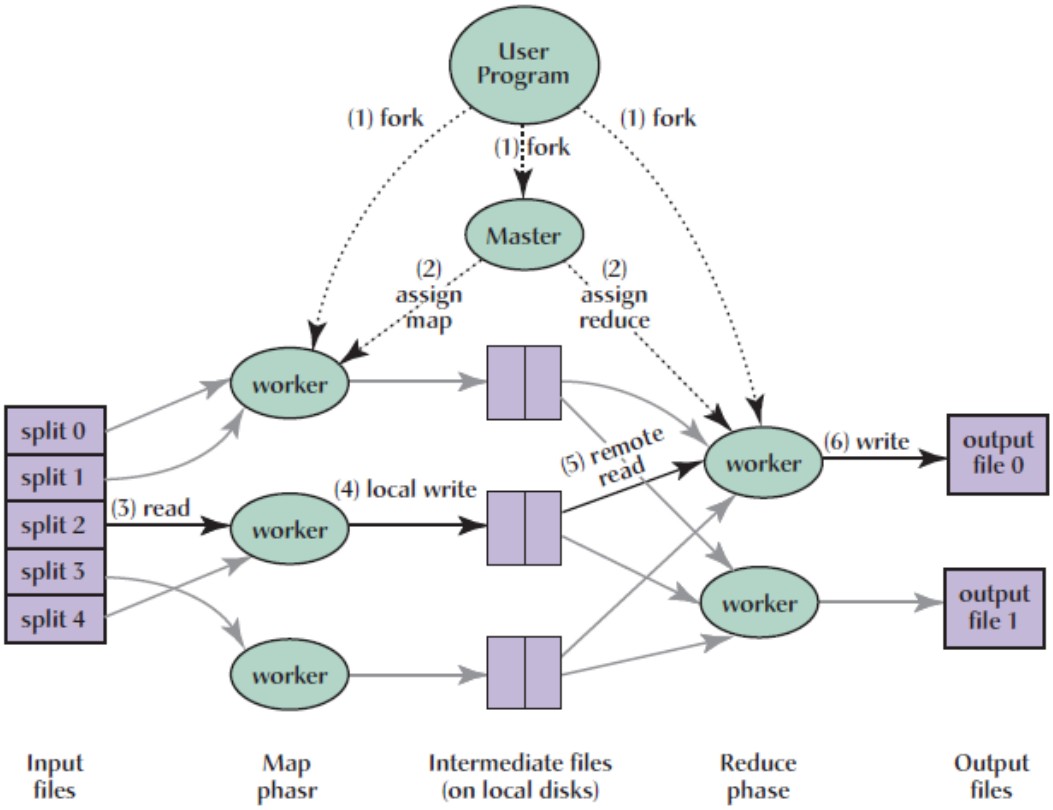
5. Când un sclav care se ocupă de realizarea operației de reducere este anunțat de către master despre aceste locații, acesta folosește proceduri remote să citească datele din memorie de pe discuri ale sclavilor de mapare. Când un sclav a citit datele intermediare pentru partiționarea sa, sortează cheile intermediare astfel încât toate instanțele legate de aceeași cheie să fie grupate împreună. Sortarea este necesară deoarece multe chei diferite de mapare sunt dedicate unui singur task de reducere. Iar dacă datele intermediare sunt prea mari pentru a fi stocate doar în memorie, atunci se apelează la o sortare externă.

6. Sclavul care se ocupă de realizarea operației de reducere pasează cheia corespunzătoare setului intermediar de valori funcției de reducere. Rezultatul de ieșire al funcției de reducere este anexat unui fișier final de ieșire asociat reducerii.

7. Când toate taskurile de mapare și de reducere au fost finalziate, masterul apelează programul utilizatorului. În acest punct, procedura MapReduce se întoarce la codul utilizatorului.

   Dacă totul s-a terminat cu succes, rezultatul de ieșire al execuției de mapreduce este disponibil în R fișiere de ieșire(câte unul pentru fiecare task de reducere cu numele fișierului specificat de utilizator). De obicei, nici nu este nevoie sa fie combinate aceste R fișiere deoarece ele sunt folosite ca intrare pentru o altă procedură MapReduce sau folosite într-o altă aplicație distribuită.

   Fluxul de mai sus este reprezentat în schema [7] care urmează:



Flux de executie

Modelul MapReduce a avut succes la Google din multe motive, printre care pot enumera faptul că este ușor de folosit chiar și pentru programatori fără experiență în sisteme paralele și distribuite, încât sunt destul de puțin exteriorizate aspectele de paralelizare, toleranță la erori, optimizarea localității; alt motiv ar fi varietatea problemelor pentru care poate fi folosit acest MapReduce: sortare, data mining, machien learning, alte sistem. Deoarece acest algoritm utilizează resursele mașinilor foarte eficient, atunci așa se explică de ce Google l-a considerat potrivit pentru utilizarea în probleme computaționale mari.

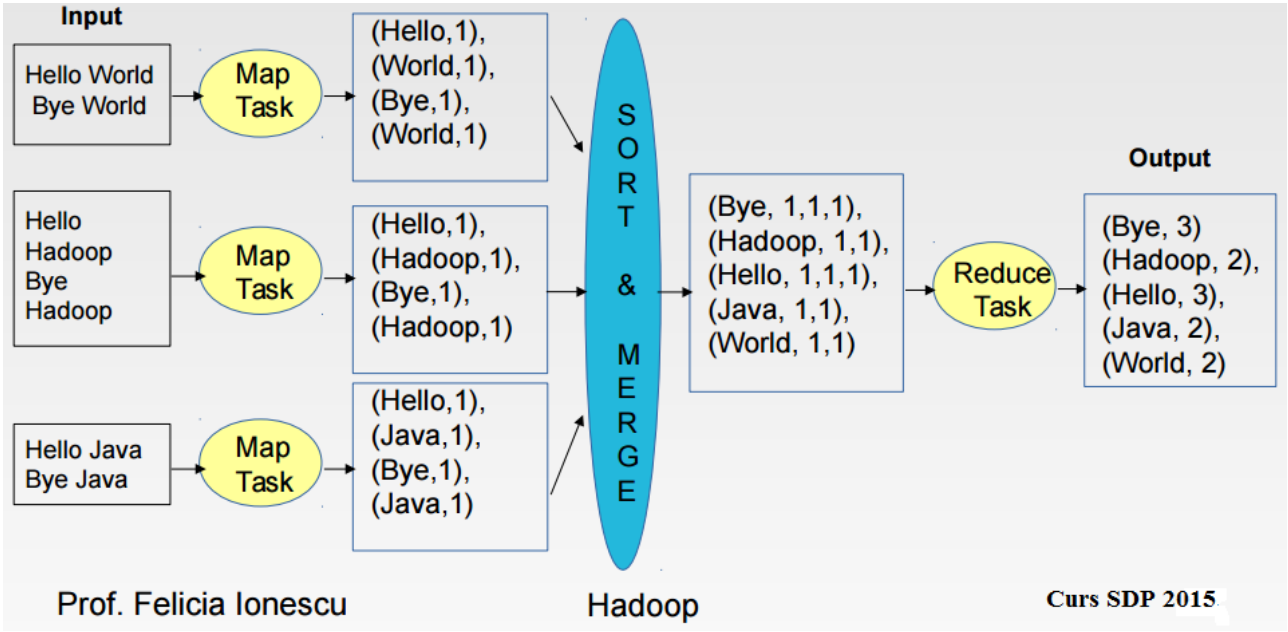
3.2.2. Exemplu de problemă pentru MapReduce

Pentru a explica în detaliu pe un exemplu concret cum funcționează MapReduce, voi folosi exemplul de numărare a cuvintelor, frecvența cu care apar cuvintele în diferite fișiere.

Acesta este cel mai întâlnit exemplu, și îl preiau de la cursul de sisteme paralele și distribuite, al doamnei profesoare Felicia Ionescu.

Motorul MapReduce este folosit în acest caz pentru a număra aparițiile fiecărui cuvânt într-un text. La intrare sunt 3 fișiere, deci pentru fiecare fișier se va crea un bloc.

Dacă numărul task-urilor de reducere este 1(implicit), datele intermediare generate de sarcinile de mapare sunt ordonate (sort) și toate elementele care au aceeași cheie sunt grupate (merge) și transmise mai departe către sarcina care se ocupă cu reducerea.



Codul care poate fi utilizat este:

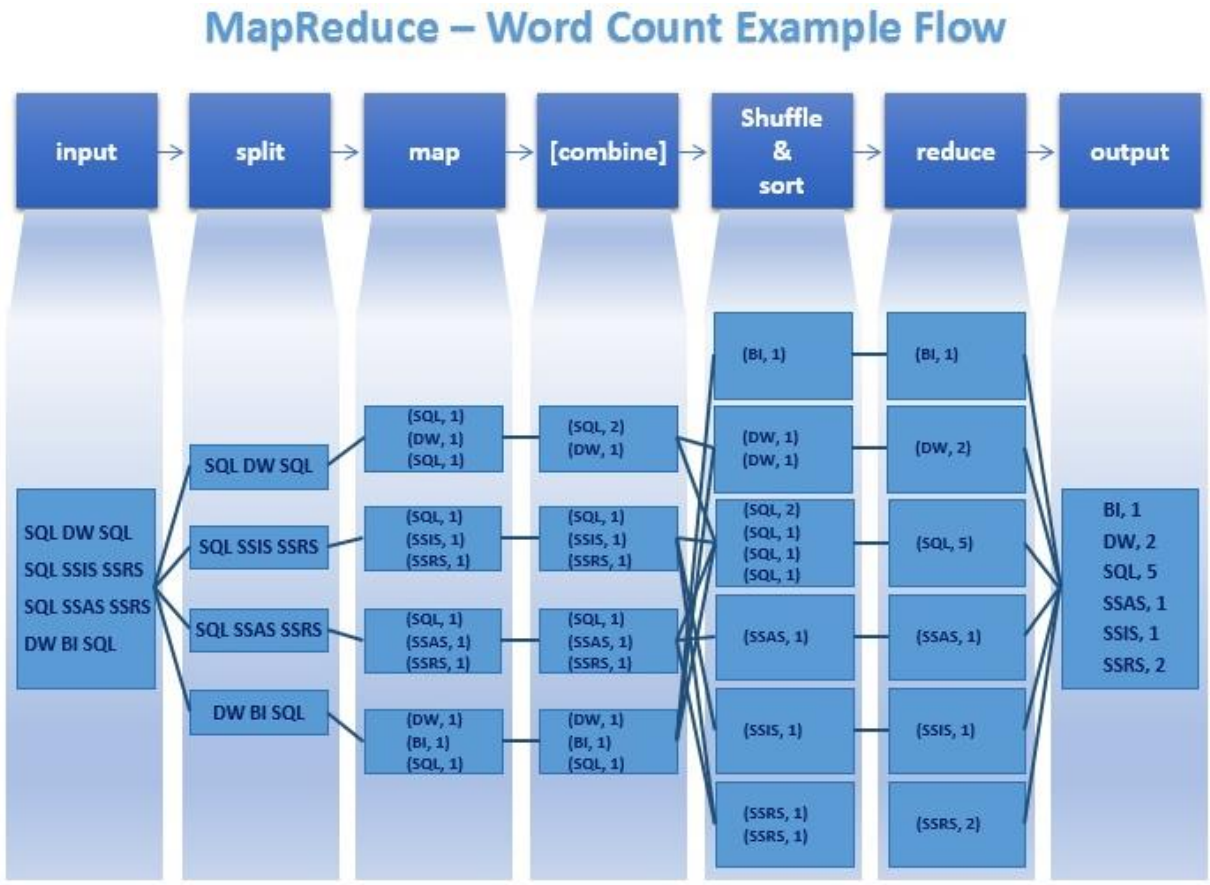
```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
 import org.apache.hadoop.mapreduce.Reducer;
 ... public class WordCount { public static class TokenizerMapper extends Mapper { private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();
public void map(Object key, Text value, Context context) throws IOException, InterruptedException { StringTokenizer itr = new StringTokenizer(value.toString());
 while (itr.hasMoreTokens()) { word.set(itr.nextToken());
context.write(word, one); } } // end map() function } // end class TokenizerMapper public static class IntSumReducer extends Reducer {...}
public static void main(String[] args) throws Exception {... } }
```

Acesta nu este singurul cod disponibil care se găsește pentru a construe MapReduce. De exemplu alt cod disponibil este cel al metodei descrisă în [8] :

```
public int run(String[] args) {
Job job = new Job(getConf());
job.setJarByClass(WordCount.class);
job.setJobName("wordcount");
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(Map.class);
job.setCombinerClass(Reduce.class);
job.setReducerClass(Reduce.class);
```

```
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPaths(job, new Path(args[1]));
boolean success = job.waitForCompletion(true);
return success ? 0 : 1;
}
```

Acest exemplu poate fi extins și aplicat pentru multe alte domenii cum a fost menționat pe parcursul lucrării. Mai jos se află un alt exemplu pentru același tip de aplicație, pentru o mai bună înțelegere a funcționării.



Sursa: https://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/

MapReduce poate fi folosit în multe alte aplicații de exemplu, în calcularea lungimii medii a cuvintelor dintr-un text dat, etc.

4. Concluzii

Hadoop este un framework open source foarte răspândit pentru Cloud Computing și are ca scop procesarea distribuită a volumelor mari de date (petabytes de date). O caracteristică importantă a frameworkului Hadoop este reprezentată de partiționarea datelor și procesarea pe mii de host-uri și execuția în paralel pe calculatoare diferite. Hadoop este caracterizat și de o scalabilitate foarte bună pentru capacitatea de calcul, de stocare prin folositea de servere commodity. Hadoop rezolvă problemele de toleranță la erori, paralelizare și distribuire, care de obicei sunt foarte dificile pentru sistemele distribuite. Atunci când se alege această soluție trebuie considerate tipurile de sisteme pentru că nu toate sistemele sunt concepute să lucreze la scara Hadoop (volume mari de date sau noduri commodity) sau nu toate sistemele se potrivesc cu tiparul MapReduce (de exemplu multe baze de date nu sunt optimizate să lucreze cu motorul MapReduce). Totuși, Hadoop este interesant deoarece este folosind pentru producție la o scara extremă în multe cazuri de big data din lume. Ca atare mii de probleme au fost idnetificate și rezolvate.

Bibliografie

[1] http://wiki.apache.org/hadoop/PoweredBy

[2] http://www.webopedia.com/TERM/H/hadoop_distributed_file_system_hdfs.html

[3] http://www.webopedia.com/TERM/H/hadoop_mapreduce.html

[4] Higai, Asami, et al. "A Study of Replica Reconstruction Schemes for Multi-rack HDFS Clusters." *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014.

[5] https://en.wikipedia.org/wiki/Jetty_(web_server)

[6] https://en.wikipedia.org/wiki/MapReduce

[7] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

[8] https://www.cs.colorado.edu/~kena/classes/5448/s11/presentations/hadoop.pdf

[9] https://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/