Prirodno-matematički fakultet Kragujevac
Institut za matematiku i informatiku

Seminarski rad iz predmeta Operativni sistemi 2

tema: **Apache Hadoop**

Student:
Srđan Nikolić 1041/2011

Profesor:
dr Miloš Ivanović

Kragujevac 2012

Sadržaj

1 Uvod u Apache Hadoop

Kada pravimo aplikacije koje obrađuju skupove podataka, možemo se osloniti na tradicionalan način upravljanja i skladištenja istih. Možemo koristiti neki od trenutno popularnih sistema za upravljanje bazama podataka kao sto su Oracle, MySql, MS Sql i ostale pouzdane sisteme za upravljanje bazama podataka koje nam nude zaista velike mogućnosti i visoke performanse. Međutim, šta ako se radi o podacima koji su toliko masivni (reda veličine petabajta) da se ne mogu smestiti na jednom računaru ili ukoliko je obrada podataka, koja se radi serijski, veoma spora. Ako znamo da količina podataka, koje aplikacija obrađuje, može da raste i eksponencijalno, potrebno je pronaći rešenje koje bi rešilo pomenute nedostatke tradicionalnog načina skladištenja i obrade podataka. Rešenje u kome bi naš softver bio skalabilan u pogledu novih količina podataka i koji bi podatke obrađivao zadovoljavajućom brzinom. Prilikom skladištenja podataka moramo voditi računa i o bezbednosti. Ukoliko koristimo jedan računar za skladištenje, znamo da je on podložan kvarovima. Ukoliko ne napravimo rezervnu kopiju naših podataka, usled kvara hard diska, možemo i trajno izgubiti podatke, što nikako ne želimo. Ideja je da se koristi Distribuirano računarstvo koje poručava distribuirane sisteme. Distribuirani sistemi se sastoje od više računara koji komuniciraju putem računarske mreže. Dakle, želimo da upravljamo podacima korišćenjem masovne paralelizacije na desetine, stotine pa čak i hiljade servera. Podaci će se skladištiti na više računara, i obrađivati se paralelno. Time dobijamo performanse u pogledu brzine obrade podataka, a ako na to dodamo da podatke možemo čuvati u više kopija na vise različitih računara, onda dobijamo i pouzdan sistem. Sistem će bez problem nastaviti sa radom ako, na primer, otkaže jedan hard disk (ili više njih). Sve ovo želimo da postignemo korišćenjem jednostavnog programskog modela, jer je paralelne programe teže pisati od serijskih. Pa u tu svrhu želimo framework koji će, između ostalih, imati sledeće osobine:

- Ugrađeno čuvanje kopija podataka,
- Lak oporavak od pada sistema i kvarova komponenti,
- Jednostavan način pisanja koda,
- Jednostavno ispravlajnje gresaka,
- Jednostavna administracija sistema.

Apache Hadoop projekat razvija softver otvorenog koda, pisan u JAVA programskom jeziku, za pouzdane, skalabilne i distribuirane sisteme. Apache Hadoop framework omogućava distribuiranu obradu velikih skupova podataka kroz klastere računara koristeći jednostavan programski model. Dizajniran je da koristi kako samo jedan server tako i hiljade servera, gde svaki od njih nudi lokalnu obradu i skladištenje podataka. On se ne oslanja na visoke kvalitet hardvera i dizajniran je da otkrije kvar, kao i da ga ispravi, na nivou aplikacije.

Skalabilnost je jedna od primarnih snaga Apache Hadoop projekta jer u početku možete koristiti samo nekoliko čvorova klastera a zatim bez problema je moguće koristiti na stotine i hiljade čvorova, po potrebi. Dakle, Apache Hadoop će nam omogućiti da jednostavnim programskim modelom upravljamo velikom količinom podataka u distribuiranom sistemu.

1.1 Zadatak seminarskog rada

Apache Hadoop projekat uključuje nekoliko podprojekata:

- **Hadoop Common** – zajednički alati koje podržavaju drugi projekti.
- **Hadoop Distributed File System (HDFS)**: Distribuirani sistem datoteka koji obezbeđuje visok pristup podacima.
- **Hadoop MapReduce**: Biblioteka za distribuirane obrade velikih setova podataka na klasterima.

Ova tri podprojekta će biti opisana u ovom seminarskom radu, takođe ćemo rešiti problem skladištenja velike količine podataka na HDFS fajl sistem. Predložiti najbolji način skladištenja i obrade podataka u smislu što boljeg iskorišćenja prostora. Opisaćemo Map-Reduce model i napisati kod u JAVA programskom jeziku, koji demonstrira rad Map/Reduce modela.

Pored ovih, postoji jos projekata koji su bazirani na Hadoop projektu u okviru Apache porjekta (oni neće biti deo ovog seminarskog rada):

- **Avro™**: Sistem za serilizaciju podataka.
- **Cassandra™**: Skalabilna multi-master baza podataka bez pojedinačnih tačaka neuspeha.
- **Chukwa™**: Sistem za prikupljanje podataka za upravljanje velikim distribuiranim sistemima.
- **HBase™**: Skalabilna, distribuirana baza podataka za podršku kreiranja struktuiranog načina skladistenja velikih tabela podataka.
- **Hive™**: Infrastruktura magacina za skladištenje podataka koja omogućava generalizovanje i ad hok pravljenje upita nad podacima.
- **Mahout™**: Skalabilna mašina za učenje i "data mining" biblioteka.
- **Pig™**: Visoki "data-flow jezik i biblioteka za paralelno programiranje.
- **ZooKeeper™**: Harmonični servis sa visokim performansama za distribuirane aplikacije.

2 Hadoop Distributed File System (HDFS)

U ovom delu ćemo se upoznati sa HDFS fajl sistemom. Idejama i ciljevima samog fajl sistema i načinom na koji on funkcioniše. HDFS je distribuirani fajl sistem napravljen za korišćenje na hardveru koji je lako dostupan u smislu da HDFS nema neke posebne hardverske zahteve. Ovaj fajl sistem ima puno sličnosti sa postojecim distribuiranim fajl sistemima. Međutim, razlike od drugih fajl sistema su značajne. HDFS je veoma otporan na greške i dizajniran da bude raspoređen na hardveru koji radi paralelno. Obezbeđuje visok nivo pristupa podacima, i pogodan je za aplikacije koje upravljaju velikim skopom podataka.

2.1 Rešavanje hardverskih grešaka

HDFS instanca može da se sastoji iz stotina ili hiljada serverskih mašina. Sama činjenica da postoji toliko veliki broj komponenti navodi na to da je verovatnoća otkaza velika i da skoro uvek neka komponenta HDFS fajl sistema ne funkcioniše. Zato se uzima da je hardverski kvar pravilo, a ne izuzutak. Otkrivanje grešaka i automatski oporavak od njih je osnovni cilj HDFS arhitekture. Aplikacijama koje su pokrenute na HDFS fajl sistemu nisu opšte namene kao one koje obično rade na opštim fajl sistemima. HDFS nije dizajniran za interaktivan pristup korisnika već za serijsku obradu podataka. Naglasak je na visoko produktivnom pristupu podacima, i zato se zahteva "streaming data access". To znači da je potrebno čitanje podataka što većom konstantnom brzinom.

2.2 Skupovi podataka na HDFS fajl sistemu

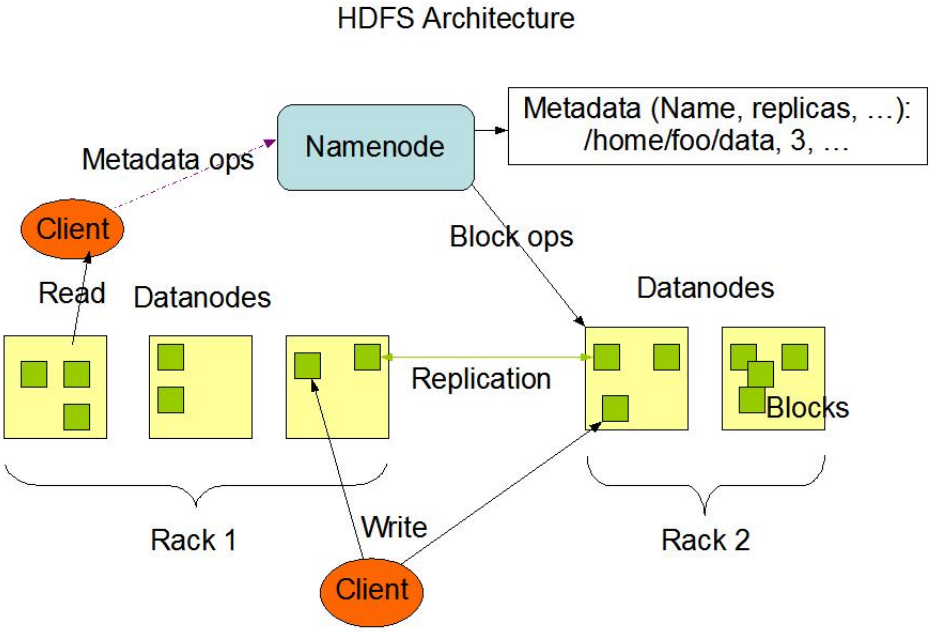
Aplikacije koje rade na HDFS fajl sistemu imaju velike skupove podataka koje obrađuju. Tipičan fajl u HDFS fajl sistemu je reda veličine od gigabajta do terabajta. Tako da je on podešen da podrži velike fajlove. To bi trebalo da obezbedi visok propusni opseg podataka stotinama čvorova u jednom klasteru. Takođe, to obezbeđuje podršku za desetine miliona fajlova u jednoj instanci.

2.3 Opis modela podataka

Aplikacijama na HDFS fajl sisstemu je potreban model piši-jednom-čitaj-više-puta. Kada se fajl jednom kreira, i u njega se upišu podaci ne postoji potreba za njegovom izmenom. Ova predpostavka pojednostavljuje povezanost podataka što omogućuje visoki propusni opseg protoka. Map/Reduce aplikacija se savršeno uklapa u ovaj model. Postoji plan da se podrži i izmena fajlova u budućnosti.
Obrada podataka od strane aplikacije je mnogo efikasnija ako se ona izvršava na računaru na kome se nalaze podaci. Ovo je posebno tačno kada je veličina podataka velika. To smanjuje opterećenje mreže i povećava propusni opseg sistema. Predpostavka je da je bolje da se aplikacija koja obrađuje podatke prebaci do mesta gde se podaci nalaze nego da se podaci prebacuju do mesta gde se nalazi aplikacija. HDFS obezbeđuje kretanje same aplikacije do mesta gde su oni skladišteni.
HDFS fajl sistem je dizajniran da bude lako prenosiv sa jedne platforme na drugu. Ovo omogućava da se HDFS izabere kao platforma za veliki skup aplikacija.

2.4 Namenode čvor i Datanode čvorovi

HDFS ima master/slave arhitekturu. HDFS klaster se sastoji od jednog Namenode čvora, master servera, koji upravlja fajl sistemom i reguliše pristup klijenata. Pored njega, postoji niz Datanode čvorova, obično jedan po čvoru na klasteru, koji upravljaju skladištem podataka na čvoru gde su pokrenuti. HDFS omogućava korisnicima da podatke čuvaju u fajlovima. Interno, fajl je podeljen na jedan ili više blokova i ovi blokovi se čuvaju na Datanode čvorovima. Nodename čvor izvršava komande fajl sistema kao sto su otvaranje, zatvaranje i preimenovanje datoteka i direktorijuma. On takođe izvršava mapiranje blokova za DataNode čvorove. Datanode čvorovi su odgovorni za opsluživanje zahteva za čitanje i pisanja od strane korisnika fajl sistema. U Datanode čvoru se obavlja kreiranje bloka, brisanje, i kreiranje njegovih kopija po nalogu iz Datanode čvora.



Slika 1: HDFS arhitektura

Namenode i Datanode su delovi softvera koji se mogu pokrenuti na lako dostupnim računarima. Radi se o računarima koji mogu pokrenuti GNU-Linux operativni sistem, pa možemo doći do zaključka da nemaju posebne hardverske zahteve. HDFS je izgrađen korišćenjem JAVA programskog jezika, pa tako svaki računar koji podržava JAVA programski jezik može da pokrene Namenode ili Datanode softver. JAVA je široko rasprostranjena, pa samim tim HDFS se moze pokrenuti na širokom spektru računara. Jedina razlika prilikom podešavanja računara je da jedan računar mora biti Namenode, on se podesava drugačije od ostalih računara. Svaki drugi računar u klasteru, koji nije Namenode, pokreće jednu istancu Datanode softvera. Arhitektura dozvoljava da se više Datanode aplikacija pokrene na jednom računaru, dok je u praksi to redak slučaj.

Postojanje jednog Namenode čvora u velikoj meri pojednostavljuje arhitekturu sistema. Namenode vrši upravljanje i skladištenje svih medapodataka na HDFS fajl sistemu.

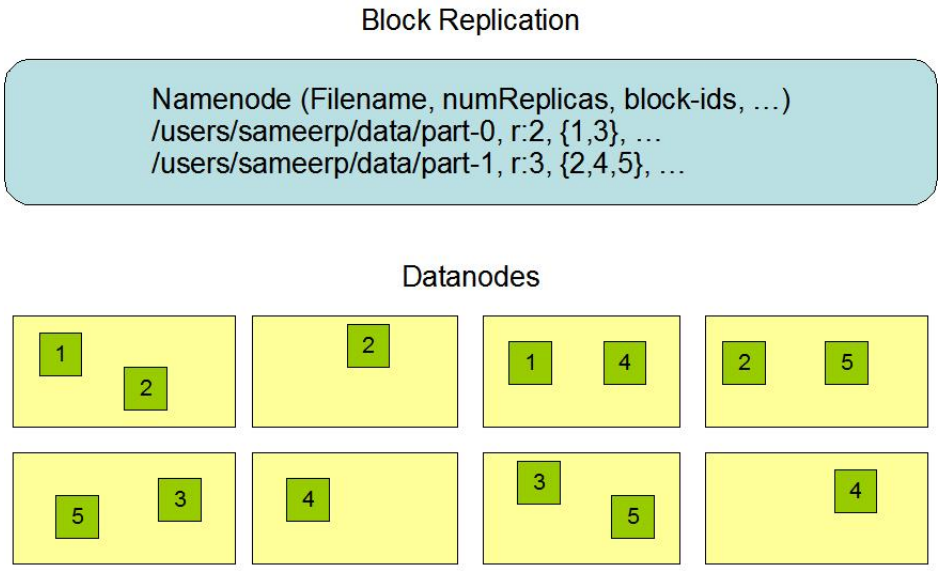
2.5 Organizacija fajl sistema

HDFS podržava tradicionalan hijerarhijsku organizaciju fajlova. Korisnik ili aplikacija mogu da kreiraju direktorijume i memorišu fajlove unutar njih. Hijerarhija fajlova i direktorijuma je ista kao na drugim postojećim fajl sistemima. HDFS jos uvek ne sprovodi dozvole za korisnički pristup kao ni hard linkove ili soft linkove. Međutim, HDFS arhitektura ne isključuje primenu ovih funkcija. Namenode održava fajl sistem. Sve promene unutar fajl sistema ili promena svojstva fajlova i direktorijuma se čuvaju na Namenode čvoru. Aplikacija može odrediti broj kopija fajlova kojima HDFS treba da upravlja. Broj primeraka jednog fajla sa naziva njegovom replikom. Ova informacija se takođe čuva na Namenode čvoru.

2.6 Replike podataka

HDFS je dizajniran da pouzdano čuva veliku količinu podataka rasprostranjenu na veliki broj računara u klasteru. Svaki fajl se čuva kao niz blokova i svi blokovi osim poslednjeg su iste veličine. Veličina bloka kao i broj replika se podešavaju u konfiguracionim fajlovima. Aplikacija može da navede broj replika fajla. Takođe, broj replika se može navesti u trenutku stvaranja datoteke i može se menjati kasnije. Fajlovi u HDFS fajl sistemu su tipa piši-jednom i samo jedan proces može da vrši upisivanja u jednom trenutku.

Namenode donosi sve odluke u vezi replika blokova. On povremeno dobija informacije od Datanode čvorova o njihovom stanju i informacije o blokovima. Prijem informacija o stanju Datanode čvora podrazumeva da on ispravno funkcioniše a informacije o blokovima sadrže spisak svih blokova na Datanode čvoru.



Slika 2: Čuvanje fajla

2.7 Izbor replike

Da bi poboljšao brzinu čitanja i što manje opteretio sistem, HDFS pokušava da zahtev čitanja opsluži replikom koja je najbliža onom ko je zahtev poslao. Ako postoji replika na čvoru odakle je

stigao zahtev, onda se ta replika koristi da zadovolji njegov zahtev. Prilikom pokretanja Namenode čvora on ulazi u specijalno stanje koje se naziva Safemode. On tada ne vrši replikacije blokova. U ovom stanju on prima informacije da od Datanode čvorova da rade bez poteškoća, kao i podatke o blokovima koji se na njima nalaze. Svaki blok ima određeni minimalni broj replika. Blok je bezbedan, ukoliko ima minimalan broj svojih replika u proveri NameNode čvora. Posle provere bezbednosti blokova namenode izlazi iz stanja Safemode. Zatim utvrđuje listu blokova koji nisu bezbedni, ako ih ima, i posle toga replicira te blokove drugim Datanode čvorovima.

2.8 Robusnost

Primarni cilj HDFS fajl sistema je pouzdano čuvanje podataka čak i kada se dešavaju greške. Tri osnova tipa grešaka su greške Namenode čvora, greške Datanode čvora i greške mrežnih particija. Svaki Datanode čvor, šalje povremeno *heartbeat* poruku Namenode čvoru. Ukoliko se desi da Datanode izgubi vezu sa Namenode čvorom, on će to primetiti ukoliko više ne dobija *heartbeat* poruke od tog Datanode čvora. U tom slučaju, Namenode markira Datanode kao da ne postoji i ne šalje mu više IO zahteve. Svi podaci koji su bili na njemu više nisu dostupni HDFS fajl sistemu. U ovom slučaju može doći do replikacije nekih blokova ukoliko je njihov broj ispod minimalnog posle markiranja Datanode čvora kao neupotrebljivog. Ponovna replikacija se može raditi iz više razloga: Datanode može postati nedostupan, greške prilikom čitanja sa hard diska, ili se povećao broj replika koji treba da postoje na sistemu.

2.9 Integritet podataka

Postoji mogućnost da blok podataka koji je preuzet od Datanode čvora bude korumpiran. Ovo se može desiti zbog nedostatka prostora za skladištenje, mrežnih grešaka ili grešaka u softveru. HDFS softver za klijente implementira *checksum* proveru sadržaja HDFS fajlova. On računa *checksum* i dobivenu vrednost čuva u skrivenom fajlu. Kada klijent zatraži blok podataka, prvo se proverava *checksum*, u ukoliko se vrednosti ne podudaraju blok se čita sa drugog Datanode čvora. Tipična veličina bloka HDFS sistema iznosi 64MB. Tako da je HDFS datoteka razdeljena na delove od po 64 MB, a ako je moguće, svaki deo je smećten na različitom Datanode čvoru.

2.10 DFSShell

HDFS omogućava DFSShell interfejs koji omogućava korisniku da upravlja podacima koji se nalaze na HDFS fajl sistemu. Sintaksa ovih komandi je identična onim shell komandama s kojima su korisnici već upoznati, dok su razlike opisane. Komande se pozivaju na sledeći način *bin/hadoop dfs <args>*. Evo i nekoliko primera:

cat
hadoop dfs -cat URI [URI . . .]
Kopira sadržaje navedenih fajlova na standardni izlaz.

Primer:
hadoop dfs -cat hdfs://host1:port1/file1 hdfs://host2:port2/file2

hadoop dfs -cat file:///file3 /user/hadoop/file4

chmod
hadoop dfs -chgrp [-R] GROUP URI [URI . . .]
Promena ovlašćenja nad fajlovima. Korisik mora biti vlasnik fajlova ili super-user

chown
hadoop dfs -chown [-R] [OWNER][:[GROUP]] URI [URI]
Menja vlasnika fajlova. Može je pokrenuti samo super-user.

copyFromLocal
Usage: hadoop dfs -copyFromLocal <localsrc> URI
Kopira fajl(fajlove) sa lokalnog fajl sistema na HDFS.

chown
hadoop dfs -chown [-R] [OWNER][:[GROUP]] URI [URI]
Kopira sadržaje navedenih fajlova sa HDFS fajl sismema na lokalni fajl sistem.

mkdir
hadoop dfs -mkdir <paths>
Ponaša se nalik UNIX komande *mkdir -p*.

rm
hadoop dfs -rm URI [URI . . .]
Briše fajlove navedene koji su dati kao argumenti komande.

put
hadoop dfs -put <localsrc> ... <dst>
Kopira fajl(fajlove) sa lokalnog fajl sistema na HDFS. Takođe čita ulaz sa stdin i upisuje ga na HDFS fajl sistem.

- hadoop dfs -put localfile /user/hadoop/hadoopfile
- hadoop dfs -put localfile1 localfile2 /user/hadoop/hadoopdir
- hadoop dfs -put localfile hdfs://host:port/hadoop/hadoopfile
- hadoop dfs -put - hdfs://host:port/hadoop/hadoopfile - **Čita podataka sa standardnog ulaza.**

2.11 HDFS Java

U ovom delu ćemo dati primere kako se vrši kreiranje, brisanje i izmena imena fajlova na HDFS fajl sistemu uz pomoć JAVA programskog jezika. Potrebno je uključiti odgovarajuće klase iz biblioteka *org.apache.hadoop.conf*, *org.apache.hadoop.fs* i *org.apache.hadoop.io* Hadoop frameworka.

Kopiranje fajla sa lokalnog fajla na HDFS fajl sistem:

```
Configuration config = new Configuration();
FileSystem hdfs = FileSystem.get(config);
Path srcPath = new Path(srcFile);
Path dstPath = new Path(dstFile);
hdfs.copyFromLocalFile(srcPath, dstPath);
```

Kreiranje fajla na HDFS fajl sistemu:

```
//byte[] buff - Sadržaj fajla
Configuration config = new Configuration();
FileSystem hdfs = FileSystem.get(config);
Path path = new Path(fileName);
FSDataOutputStream outputStream = hdfs.create(path);
outputStream.write(buff, 0, buff.length);
```

Preimenovanje fajla na HDFS fajl sistemu:

```
Configuration config = new Configuration();
FileSystem hdfs = FileSystem.get(config);
Path fromPath = new Path(fromFileName);
Path toPath = new Path(toFileName);
boolean isRenamed = hdfs.rename(fromPath, toPath);
```

Brisanje fajla sa HDFS fajl sistema:

```
Configuration config = new Configuration();
FileSystem hdfs = FileSystem.get(config);
Path path = new Path(fileName);
boolean isDeleted = hdfs.delete(path, false);
```

2.12 Rad sa kompresovanim fajlovima

Kao što je više puta pomenuto Hadoop radi sa veoma velikim fajlovima pa bi i mala kompresija tih fajlova uštedela značajan prostor. Posebno ako su zapisi u fajlu slični pa je moguće njihovu veličinu smanjiti nekoliko puta kompresovanjem. Nije svejedno da li treba skladištiti fajl od 5 terabajta ili njegovu kompresovanu verziju od 2 terabajta. Pre svega treba napomenuti da se jedan kompresovan fajl obrađuje od strane jednog mapiranog zadatka. Dakle ne može se deliti po blokovima i tako obrađivati paralelno. Jasno je da je to ograničenje uslovljeno načinom kompresije fajla, jer Hadoop u ovom slučaju ne zna kako da podeli fajl. Prlikom pisanja JAVA koda za Map/Reduce posao dovoljno je navesti da je ulazni format podataka *TextImportFormat* u konfiguraciji posla kao *jobConf.setInputFormat(TextInputFormat.class);*. Ovo je inače i podrazemevani ulazni format, tako da dolazimo do zaključka da Hadoop sam zaključuje da li je fajl kompresovan ili ne, i kako će sa njim da postupi. Ovo je, naravno, jos jedna dobra strana Hadoop-a.

Zbog ograničenja da samo jedna mapa obrađuje jedan kompresovan fajl, u nastanku su data dva programa napisana u JAVA programskom jeziku koja čitaju podatke sa standardnog ulaza i kreiraju kompresovane fajlove na HDFS sistemu. *CompressToHdfsMB.java* kreira niz kompresovanih fajlova na HDFS sistemu i u argumentu očekuje maksimalnu veličinu kompresovanih fajlova, dok *CompressToHdfsLine.java* u argumentu očekuje maksimalan broj linija ulaznog fajla koje treba kompresovati u kompresovani fajl. Detaljnija objašnjenja su data u komentarima koda pomenutih programa u nastavku.

```
1  import java.io.*;
2  import java.util.zip.*;
3
4  import org.apache.hadoop.conf.Configuration;
5  import org.apache.hadoop.fs.FileSystem;
6  import org.apache.hadoop.fs.FSDataInputStream;
7  import org.apache.hadoop.fs.FSDataOutputStream;
8  import org.apache.hadoop.fs.Path;
9
10 public class CompressToHdfsMB
11 {
12
13 /*
14   Klasa koja cita podatke, liniju po liniju, sa standardnog ulaza i prazi GZ fajlove na lokalnom
       fajl sistemu,
15   zatim ih kopira na HDFS fajl sistem i brise sa lokalnog. U komandnoj liniji se zadaju sledeci
       argumenti:
16   argument_1 - putanja na HDFS fajl sistemu gde se cuvaju kompresovani fajlovi
17   argument_2 - prefiks naziva fajlova koji se kreiraju
18   argument_3 - maksimalna velicina GZ fajlova
19 */
20   public static void main(String[] args) throws IOException
21   {
22      try
23      {
24     int brojFajla = 0; //redni broj gzip fajla koji se kreira
25
26     String imeFajla; //naziv gzip fajla
27     String prefiks = args[1]; //prefix naziva fajla
28     String putanja = args[0]; //putanja na HDFS fajl sistemu
29     int velicinaFajla = Integer.parseInt(args[2])*1024*1024; //velicina gzip fajla u MB
30
31     InputStreamReader inp = new InputStreamReader(System.in);
32     BufferedReader br = new BufferedReader(inp);
33     File gzipFile;
34
35     Path path;
36     FileOutputStream fos;
37     GZIPOutputStream gzos;
38
39     //napravi instancu HDFS fajl sistema
40     Configuration config = new Configuration();
41       FileSystem hdfs = FileSystem.get(config);
42
43       //kreiraj prvi fajl
44     imeFajla = prefiks.concat(brojFajla + ".gz");
45     path = new Path(putanja + "/" + imeFajla);
46     fos = new FileOutputStream(imeFajla);
47     gzos = new GZIPOutputStream(fos);
48     gzipFile = new File(imeFajla);
49
50     System.out.println("Kreiram fajl: " + imeFajla);
51
52     String linija;
53     //citaj fajl sve dok ne naidjes na znak EOF
54     while ((linija = br.readLine()) != null)
```

```
55      {
56        if(gzipFile.length() >= velicinaFajla)
57        {
58          //zatvori fajl
59          gzos.close();
60
61          //ako fajl postoji na HDFS fajl sistemu izbrisi ga
62          if (hdfs.exists(path))
63                hdfs.delete(path);
64          //kopiraj kreirani fajl sa lokalnog diska na HDFS
65          hdfs.copyFromLocalFile(new Path(imeFajla), path);
66          //izbrisi fajl sa lokalnog diska
67          gzipFile.delete();
68
69          //kreiraj novi fajl
70          brojFajla++;
71          imeFajla = prefiks.concat(brojFajla + ".gz");
72          path = new Path(putanja + "/" + imeFajla);
73          fos = new FileOutputStream(imeFajla);
74          gzos = new GZIPOutputStream(fos);
75              gzipFile = new File(imeFajla);
76
77          System.out.println("Kreiram fajl: " + imeFajla);
78        }
79
80        //dodaj oznaku za novi red
81        linija = linija.concat("\n");
82        //upisi liniju u kompresovan fajl
83        gzos.write(linija.getBytes(),0,linija.length());
84      }
85
86
87      gzos.close();
88
89      //ako fajl nije prazan kopiraj ga na HDFS
90      if(gzipFile.length()>0)
91      {
92        if (hdfs.exists(path))
93            hdfs.delete(path);
94
95        hdfs.copyFromLocalFile(new Path(imeFajla), path);
96        gzipFile.delete();
97      }
98
99
100     System.out.println("Zavrseno kreiranje fajlova!");
101
102   }catch (Exception e)
103   {
104     System.out.println(e.getMessage());
105   }
106
107   }
108
109 }
```

CompressToHdfsMB.java

Skripta za kompajliranje:

```
1 #!/bin/bash
2 #kompajliraj
3 rm -r klase_zip
4 mkdir klase_zip
5 javac -classpath hadoop-core-1.0.3.jar -d klase_zip CompresToHdfsMB.java
6 #napravi .jar
7 rm gzip.jar
8 jar -cvf gzip_MB.jar -C klase_zip/ .
```

kompajliraj_CompressToHdfsMB.sh

Skripta za pokretanje:

```
#!/bin/bash
cat proizvoljan_fajl | hadoop jar gzip_MB.jar CompresToHdfsMB probaGZ fajl 2
```

pokreni_CompressToHdfsMB.sh

```
import java.io.*;
import java.util.zip.*;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.compress.*;

/*
  Klasa koja cita podatke, liniju po liniju, sa standardnog ulaza i prazi GZ fajlove na HDFS fajl sistemu.
  U komandnoj liniji se zadaju sledeci argumenti:
  argument_1 - putanja na HDFS fajl sistemu gde se cuvaju kompresovani fajlovi
  argument_2 - prefiks naziva fajlova koji se kreiraju
  argument_3 - broj linija ulaznog fajla koji se arhiviraju po jednom izlaznom GZ fajlu
*/

public class CompressToHdfsLine
{

  public static void main(String[] args) throws IOException
  {


    String imeFajla; //ime zpit fajla
    String prefiks = args[1]; //prefiks gzip fajla
    String putanja = args[0]; //putanja na HDFS fajl sistemu
    int brojLinijaPoFajlu = Integer.parseInt(args[2]); // broj linija ulaznog fajla koje se arhiviraju po jednom zip fajlu

    int len;
    int brojFajla = 0; //redni broj fajla koji se kreira
    int brojLinije = 0; //redni broj linije koja se kompresuje
    String linija;

    InputStreamReader inp = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(inp);

    //Hadoop promenljive
    FileOutputStream fos;
    GZIPOutputStream gzos;
    Path filenamePath;
    OutputStream outputStream;
    CompressionCodecFactory codecFactory;
    CompressionCodec codec;
    CompressionOutputStream compressedOutput = null;

    //Napravi instancu HDFS fajl sistema
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);

        //citaj linije sa stdandardnog ulaza sve dok ne naidjes na EOF
    while ((linija = br.readLine()) != null)
    {
      if(brojLinije%brojLinijaPoFajlu  == 0)
      {
        //zatvori fajl
        if(compressedOutput != null)
          compressedOutput.close();

        //kreiraj putanju do novog fajla
        imeFajla = prefiks.concat(brojFajla + ".gz");
```

```
63          filenamePath = new Path(putanja.concat("/"+imeFajla));
64
65          //ako fajl vec postoji izbrisi ga
66          if (fs.exists(filenamePath))
67                fs.delete(filenamePath);
68
69          //kreiraj novi fajl
70          outputStream = fs.create(filenamePath);
71          codecFactory = new CompressionCodecFactory(conf);
72          codec = codecFactory.getCodec(filenamePath);
73          compressedOutput = codec.createOutputStream(outputStream);
74
75          System.out.println("Kreiram fajl: " + imeFajla);
76          brojFajla++;
77        }
78
79        //upisi liniju u fajl
80              linija = linija.concat("\n");
81        compressedOutput.write(linija.getBytes(),0,linija.length());
82        brojLinije++;
83
84
85      }
86
87      //zatvori fajl ako je kreiran
88      if(compressedOutput != null)
89        compressedOutput.close();
90
91      System.out.println("Zavrseno kreiranje fajlova!");
92    }
93  }
```

CompressToHdfsLine.java

Skripta za kompajliranje:

```
1  #!/bin/bash
2
3  #kompajliraj
4  rm -r klase_zip
5  mkdir klase_zip
6  javac -classpath hadoop-core-1.0.3.jar -d klase_zip CompressToHdfsLine.java
7
8  #napravi .jar
9  rm gzip.jar
10 jar -cvf gzip_Line.jar -C klase_zip/ .
```

kompajliraj_CompressToHdfsLine.sh

Skripta za pokretanje:

```
1  #!/bin/bash
2  cat proizvoljan_fajl | hadoop jar gzip_Line.jar CompressToHdfsLine probaGZ fajl 20000000
```

pokreni_CompressToHdfsLine.sh

3 Map/Reduce model

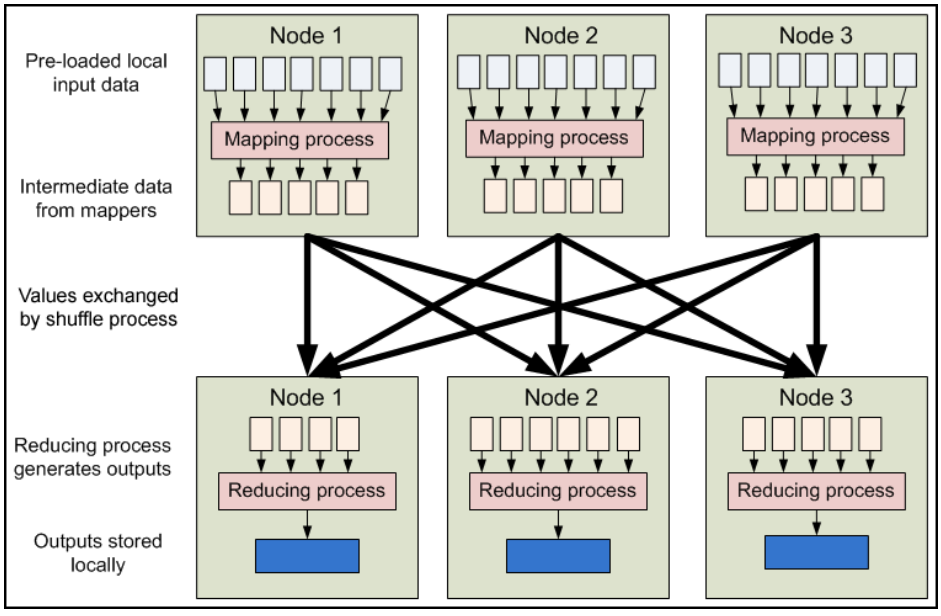
3.1 Uvod u Map/Reduce model

Hadoop podržava Map/Reduce model koji služi za pisanje programa koji obrađuju ogromne skupove podataka (više terabajta setova podataka) paralelno na velikim klasterima koji mogu sadržati stotine, pa i hiljade čvorova. Pisanje ovakvih programa je omogućeno u sledećim programski jezicima : Java, Ruby, Python, i C++. Mi ćemo se fokusirati na JAVA programski jezik. Klasteri na kojima se pokreću MapReduce aplikacije su sastavljeni od lako dostupnih komponenti koje su pouzdane.

Model je baziran na dve posebna koraka:

- *Mapiranje*: Početni korak obrade i transformacije ulaznih podataka, u kome se zapisi ulaznih podataka obrađuju paralelno,
- *Redukcija*: Ovaj korak se radi posle mapiranja i predstavlja agregaciju, gde se svi povezani zapisi obrađuju kao jedan entitet(detaljnije objašnjenje u nastavku).

Glavni koncept Map/Reduce modela u Hadoop-u je da se ulaz može podeliti u više logičkih komada, i da se svaki od tih komada može obrađivati paralelno, od strane zadataka mapiranja. Rezultati ovih pojedinačnih obrada mogu biti podeljeni na različite skupove podataka koji se sortiraju. Svaki sortirani komad se šalje redukcionom zadatku. Na *slici 3* prikazan je Map/Reduce model, i tok podataka na najvišem nivou.



Slika 3: Najviši nivo Map/Reduce toka podataka

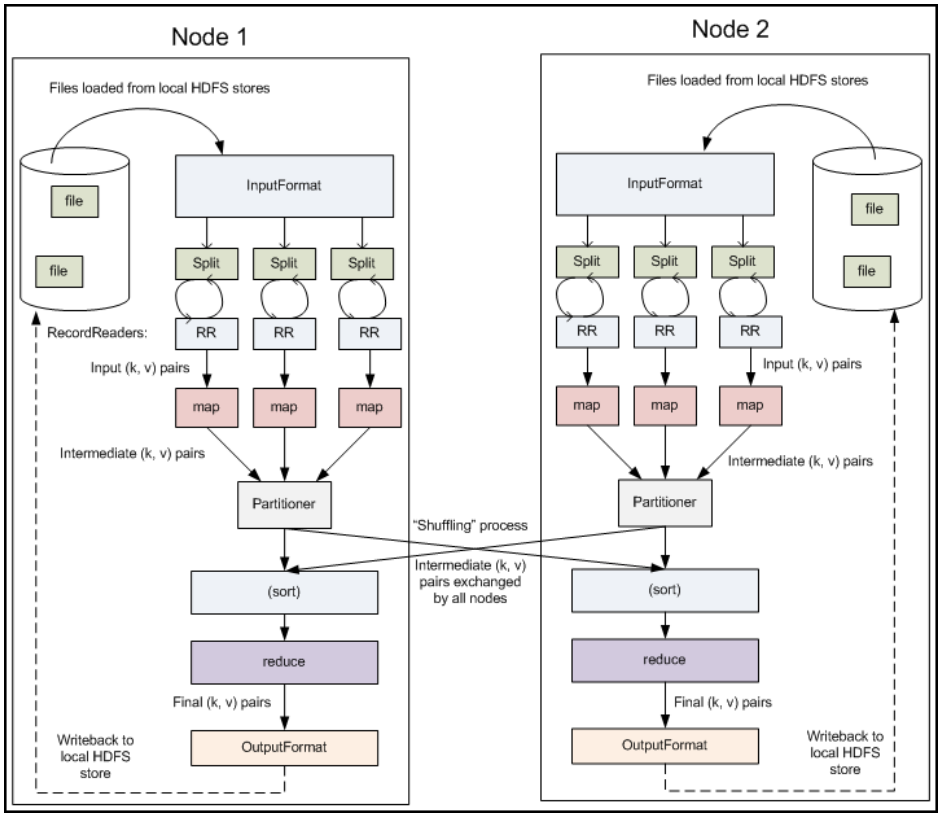
Map/Reduce ulazni podaci su smešteni na HDFS fajl sistemu. Oni su raspoređeni po svim čvorovima. Startovanje Map/Reduce programa uključuje pokretanje mapiranja na većini, ili svim čvorovima u klasteru. Svaki zadatak mapiranja je ekvivalentan sa ostalim. Dakle, ne postoje ograničenja po pitanju prihvatanja delova fajla, svaki zadatak može obrađivati bilo koji deo fajla.

Map/Reduce framework radi isključivo sa **(key, value)** parovima, odnosno Map/Reduce je framework koji vidi ulaz posla kao skup parova **(key, value)** i izlaz je takođe skup parova **(key, value)**. Faza mapiranja služi da se od ulaznog fajla naprave odgovarajući parovi **(key, value)**.

Zadaci mapiranja (*map tasks*) ne razmenjuju među sobom informacije. Takođe, redukcioni zadaci (*reduce tasks*) ne komuniciraju međusobno. Korisnik nema uticaj na tok podataka, i kako se informacije razmenjuju između zadataka mapiranja i zadataka redukcije, za sve transfere podataka je zadužen Hadoop. Ova karakteristika oslikava sposobnost Hadoop Map/Reduce modela.

Map task se može pokrenuti na bilo kom čvoru koji obrađuje podatke (Datanode čvoru). On je zadužen da od ulaznih podataka napravi parove ključ/vrednost. Svi izlazi posle mapiranja će biti podeljenji i sortirani. I svaki novi sortirani komad će se obrađivati od strane redukcionih zadataka, koji rede potpuno paralelno.

Na *slici 4* dat je još detaljniji opis toka podataka gde se može videti tok podataka i komunikacija između objekata.



Slika 4: Detaljan prikaz toka podataka Map/Reduce posla

Ulazni fajlovi su inicijalno skladišteni na HDFS sistemu i njihov format je proizvoljan. Možemo koristiti fajlove bazirane na linijama, binarne fajlove i ostale. Za ove ulazne fajlove je tipično da su veoma veliki - desetine gigabajta i više.

InputFormats: Definiše kako su podeljeni fajlovi na manje komade. Postoji nekoliko tipova:

- *TextInputFormat* - Ovo je osnovni format. Njegov ključ je bajt na početku linije, a vrednost sadržaj linije,
- *KeyValueInputFormat* - Sastoji se iz (key, value) parova. Ključ je od početka linije do prvog tab-a, dok je vrednost ostatak linije,
- *SequenceFileInputFormat* - Specijalni Hadoop binarni format. Ključ i vrednost se definišu od strane korisnika.

InputSplits: Opisuje jedinicu posla koji svaki map zadatak izvršava. Map zadatak može čitati i ceo fajl, ali ovde je najčešći slučaj da se čitaju samo delovi fajlova. Veličina InputSplit-a se može podesiti u *hadoop-site.xml*, podešavanjem parametra *mapred.min.split.size*.

Treba naglasiti da se yipovani fajlovi ne mogu deliti, već će jedan se jedan zipovan fajl obrađivati od strane jednog zadatka mapiranja. **RecordReader**: InputSplit definiše jedinicu rada, ali ne definiše i način na koji mu se pristupa. RecordReader klasa učitava podatke iz svog izvora i pretvara ih u parove (key, value) koji su pogodni za rad od strane klase Mapper. InputFormat klase definiše na koji način će se kreirati parovi (key, value), i RecordReader prema tome učitava zapise.

Mapper: Mapper obavlja korisnički definisan posao prve faze Map/Reduce programa. Klasa dobija ključ i vrednost a opciono može da prosledi par (ključ, vrednost) reduktorima. Nova instanca Mapper-a je nezavisan JAVA proces, tako da se svaki zadatak Mapiranja obavlja potpuno paralelno, samim tim se i fajl obrađuje paralalno.

Partition and Shuffle: Kada maperi završe obradu oni moraju razmeniti svoj izlaz sa reduktorima. Pre nego što reduktori dobiju ove podatke, oni se mešaju (*shuffleing*). Svaki redikcioni posao ima svoje posebno parče koje obrađuje, dok svaki maper može proslediti svoje parove podataka u bilo koje parče koje pripada nekom reduktoru i predstavlja njegov ulaz. Svi parovi sa istim ključem se prosleđuju istom reduktoru.

Sort: Pre nego što se parovi (key, value) posalju reduktoru, vrši se njihovo sortiranje.

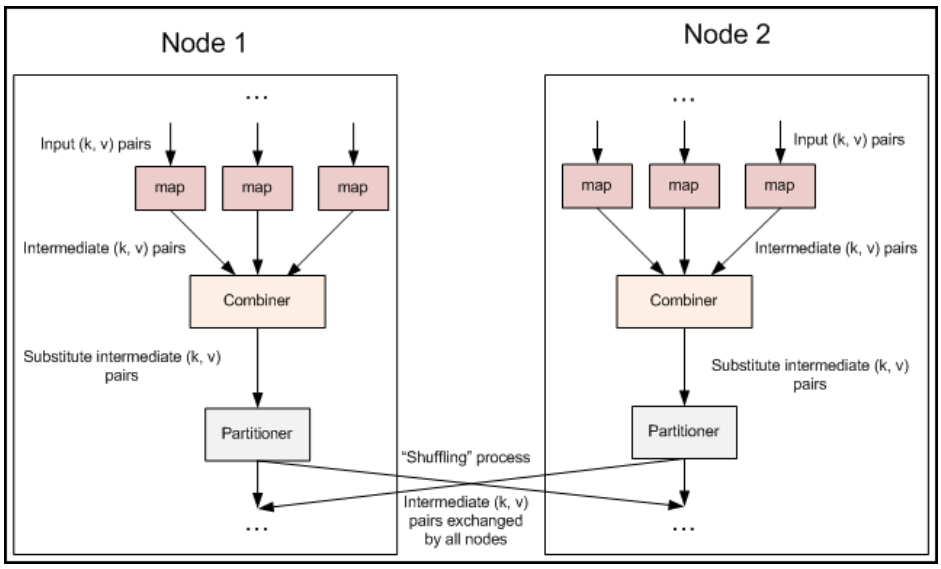
Reduce: Reduce instanca se kreira za svaki redukcioni zadatak. Ova instanca pokreće kod koji je definisan od strane korisnika. Za svaku vrednost ključa reduce() funkcija se poziva se jednom poziva. Izlaz iz Reducer-a su parovi (key, value).

OutputFormat: Način na koji se pišu izlazni (key, value) parovi je definisan od strane OutputFormat klase. Instanca ove klase upisuje izlazne podatke na lokalni disk ili HDFS fajl sistem. Svaki reduktor piše zaseban fajl u zajedničkom direktorijumu za sve reduktore. Ovi fajlovi su najčešće numerisani kao *part-nnnnn*.

RecordWriter: Kao što InputFormat čita pojedinačne zapise pomoću klase RecordReader, Out-

putFormat klasa se služi klasom RecordWriter da bi zapisala pojedinačne zapise na HDFS fajl sistem ili na lokalni disk.

3.2 Dodatna funkcionalnost Map/Reduce model



Slika 5: Kombinator

Combiner: Upotreba Combinier klase je opciona. Ona predstavlja *"mini-reduce"* koji radi samo sa podacima koji su generisani na jednom računaru. Koristi se za optimizaciju. On prima podatke iz mepera i optimizuje ih prema potrebi. U primeru koji sledi u nastavku biće opisana jedna od mogućnosti upotrebe. Za Combiner klasu ne moramo pisati nov kod u većini slučajeva, već možemo iskoristiti postojeću Reducer klasu.

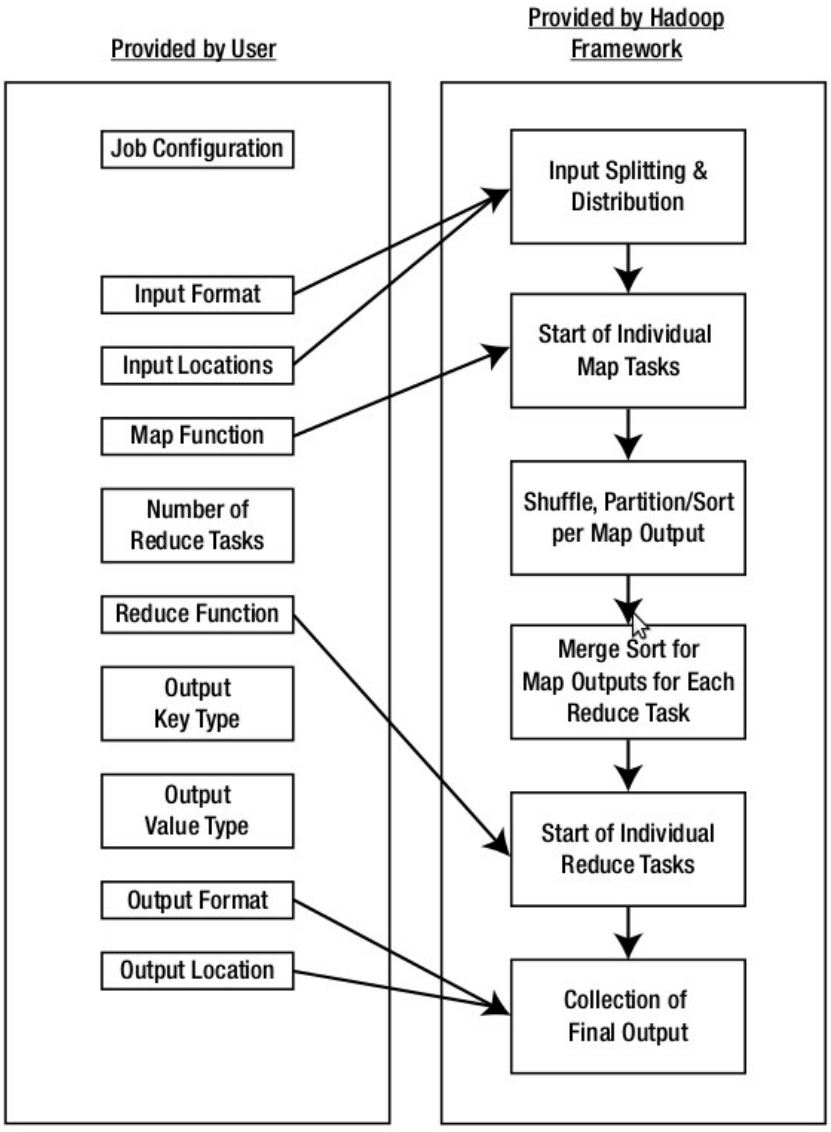
3.3 Osnove Map/Reduce posla

Mapreduce job je jedinica posla koji klijent želi da izvrši. Sastoji se iz ulaznih podataka, Map/Reduce programa i informacija o konfiguraciji. U ovom delu ćemo proći kroz delove iz kojih se sastoji Map/Reduce posao i sve to ilustrovati na primeru. Korisnik konfiguriše i podnosi *Map/Reduce job*(ili samo *job*) *Map/Reduce framework-u*, koji deli posao na manje zadatke, zatim vrši mešanje podataka, sortiranje, i na kraju izvršava zadatake redukovanja.

Sada ćemo navesti delove Map/Reduce posla koje on mora da sadrži kao i ko vrši podešavanje tih delova:

- *Konfiguracija posla* - **korisnik**
- *Ulazna podela podataka (spliting) i njihova distribucija* - Hadoop framework
- *Startovanje pojedinačnih zadataka mapiranja za svaki deo fajla (split)* - Hadoop framework
- *Map funkcija, poziva se za svaki par (key, value)* - **korisnik**
- *Mešanje podataka po izlasku iz mapiranja* - Hadoop framework

- *Sortiranje izlaza posle mešanja podataka* - Hadoop framework
- *Startovanje pojedinačnih zadataka redukcije za svake sortirane delove fajla (split)* - Hadoop framework
- *Reduce funkcija, poziva se za sve parove (key, value) koji imaju istu vrednost key* - **korisnik**
- *Sakupljanje i skladištenje izlaznih podataka u direktorijom koji je naveden prilikom startovanja posla. Postojeća onoliko fajlova koliko je definisano zadataka redukcije* - Hadoop framework



Korisnik je odgovoran za definisanje posla, definisanje lokacije na kojoj se nalaze ulazni podaci, definisanje lokacije izlaznih podataka, i obezbeđivanje ulaza u određenom formatu. Hadoop je odgovoran za distribuiranje posla kroz TaskTracker čvorove klastera. Pokretanje mapiranja, mešanja, sortiranja, i faze redukcije. Takođe, Hadoop je odgovoran za smeštanje izlaza na definisanu lokaciju kao i informisanje korisnika o uspešnosti posla.

Primer Map/Reduce posla

Rad Map/Reduce programa najbolje ćemo ilustrovati na primeru. Imamo fajl u kome se nalaze godine i izmerene temperature date kao:

(redni broj merenja, godina, temperatura)
(424,1950,1)
(318,1950,22)
(212,1950,15)
(123,1949,1)
(47,1949,8)
(47,1949,-9999)

Naš zadatak je da odredimo maksimalnu temperaturu za svaku godinu. Prva vrednost, redni broj merenja, će biti ignorisana od strane map funkcije koju ćemo napisati. Map funkcija parsira svaku liniju i formira izlazni fajl kao **(godina, izmerena temperatura)**:

(1950, 0)
(1950, 22)
(1950, -11)
1949, 111)
1949, 78)

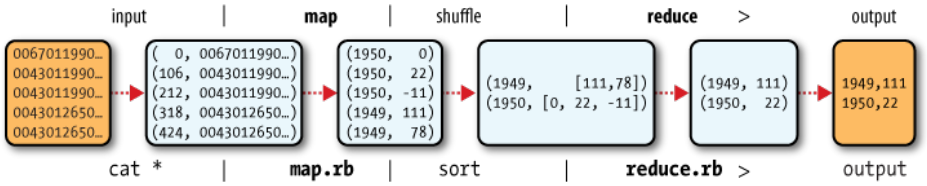
Primetimo da je map funkcija izbacila liniju koja ima vrednost za temperaturu -9999, što predstavlja indikator da ne postoji vrednost merenja. Dakle, mapiranje služi i za izbacivanje onih podataka koji nisu od važnosti za dalju obradu. Pre nego što se ovakav izlaz prosledi reduce funkciji, on se prvo obradi na sledeći način:

(1949, [111, 78])
(1950, [0, 22, 11])

Kao što vidimo, podaci se sortiraju i grupišu po ključu. Zatim svaka reduce funkcija dobija listu temperatura za svaku godinu i kroz njih vrsi iteracije da bi izračunala maksimalnu temperaturu za tu godinu i dobila izlazni fajl koji izgleda kao:

(1949, 111)
(1950, 22)

Ceo postupak je dat na sledećoj slici:



3.4 JAVA MapReduce

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

/*
   Klasu MaxTemperatureMapper cemo koristiti za mapiranje. Ona
   nasledjuje MapReduceBase klasu i implementira interfejs Mapper u kome je
   definisana funkcija map(). Interfejs Mapper je generickog tipa, sa cetiri
   deklarisana formalna tipa (input key, input value, output key, output value)
   definisanih kao <LongWritable, Text, Text, IntWritable>
*/
static class MaxTemperatureMapper extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable>
{
    private static final int MISSING = -9999;

    //LongWritable key - u nasem primeru je redni broj merenja i predstavlja input key
    //Text value - linija u fajlu
    //OutputCollector<Text, IntWritable> output - <godina,temperatura>
    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
        Reporter reporter)
    throws IOException
    {

      String tmp = value.decode(value.getBytes());
      String[] tmpValues = tmp.split(",");

      String year = tmpValues[1];
      int airTemperature;
      airTemperature = Integer.parseInt(tmpValues[2]);

      //ako ne postoji vrednost, ignorisi liniju
      if (airTemperature != MISSING)
        //kreiraj izlazni par <godina, temperatura>
        output.collect(new Text(year), new IntWritable(airTemperature));
    }

}


/*
   Klasa koju cemo koristiti za reduce podataka ,MaxTemperatureReducer,
   implementira interfejs Reducer u kome je definisana funkcija reduce().
   Interfejs Reducer je generickog tipa, sa cetiri deklarisana formalna tipa
   (input key, input value, output key, output value)
*/
static class MaxTemperatureReducer
 extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>
{

    //Text key - kljuc, koji je u nasem slucaju godina merenja
    //Iterator<IntWritable> values - ova promenljiva sadrzi sva merenja koja imaju istu vrednost
        kljuca
```

```
56      public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
            output, Reporter reporter)
57      throws IOException
58      {
59        int maxValue = Integer.MIN_VALUE;
60
61        //dok postoje vrednosti temperatura za trenutnu godinu
62        while (values.hasNext())
63        {
64          //trazi maksimum dva broja
65          maxValue = Math.max(maxValue, values.next().get());
66        }
67
68        //kreiraj izlazni par <godina, maksimalna_vrednost_temperatire>
69        output.collect(key, new IntWritable(maxValue));
70
71      }
72  }
73
74  public class MapREduce
75  {
76
77      public static void main(String[] args) throws Exception
78      {
79
80        if (args.length != 2)
81        {
82          System.err.println("<input path> <output path>");
83          System.exit(-1);
84        }
85
86        JobConf conf = new JobConf(MaxTemperature.class);
87        conf.setJobName("Max temperature");
88        FileInputFormat.addInputPath(conf, new Path(args[0]));
89        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
90        conf.setMapperClass(MaxTemperatureMapper.class);
91        conf.setReducerClass(MaxTemperatureReducer.class);
92        conf.setOutputKeyClass(Text.class);
93        conf.setOutputValueClass(IntWritable.class);
94
95        JobClient.runJob(conf);
96      }
97  }
```

MapReduce.java

4 Zaključak

Ukoliko bi se koristio standardan način paralelne obrade podataka, ili recimo ako koristimo MPI framework, odmah bi morali da razmisljamo kako da napišemo algoritam koji će raditi za proizvoljan proj procesora i veličinu fajla, zatim bi morali da razmisljamo o komunikaciji između procesa. Takođe treba razmišljati i o podeli fajla, tako da svaki procesor bude jednako opterećen, ili recimo, procesor koji ima bolje performanse bi trebao da odradi više posla od lošijeg i tako dalje. Moramo priznati da to nije bas lak zadatak. Pisanje paralelnih programa je svakako teže od pisanja serijskih programa. Međutim, kao što je navedeno u uvodnom delu, to se i pokazalo, Hadoop nam omogućava da jednostavnim programskim modelom obrađujemo veliku količinu podataka i da ne moramo da vodimo računa o tome kako on deli posao. Dovoljno je da navedemo broj zadataka za mapiranje i redukovanje a ostalo je briga Hadoop-a. Ako na to dodamo da samo treba da mu prosledimo putanju do foldera gde se nalazi fajl, ili više njih, i da on radi sa kompresovanim fajlovima isto kao i sa nekompresovanim, sa gledišta programera onda Hadoop deluje kao idealno rešenje za obradu velike količine podataka. Takođe je rešen i problem sa skladištenjem velikih fajlova, na više računara u distribuiranom sistemu. Hadoop će se svakako još razvijati i usavršavati. Da je to tako potvrđuju i velike IT kompanije koje ga koriste, počevši od Yahoo-a, preko Facebook-a do Microsofta.

Nastavak istraživanja bi išao u smeru ispitivanja performansi Map/Reduce poslova kao i istraživanje nekih od podprojekata Hadoop-a navedenih u uvodnom delu.

5 Literatura

- T. White, Hadoop, The Definitive Guide, O'Reilly Media, 2009.
- Apache Hadoop, Apache Map-Reduce Tutorial.
- Apache Hadoop, Apeche HDFS Tutorial.
- Apache Hadoop, Yahoo Map-Reduce Tutorial.
- Apache Hadoop, Yahoo HDFS Tutorial.