Primjena MapReduce algoritma na analizu nizova tekstualnih podataka

Volarić, Karolina

Master's thesis / Diplomski rad

2014

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:795312

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Karolina Volarić

PRIMJENA MAPREDUCE ALGORITMA NA ANALIZU NIZOVA TEKSTUALNIH PODATAKA

Diplomski rad

Voditelj rada:
izv.prof.dr.sc.Luka Grubišić

Zagreb, srpanj 2014.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **3**

1 Hadoop **4**
1.1 Definicija i opis 4
1.2 Spremanje podataka i analiza 5
1.3 Usporedba s drugim sustavima 5

2 MapReduce **8**
2.1 Opis rada algoritma 8
2.2 Combiner funkcija 10
2.3 Razvoj MapReduce aplikacije 10
2.3.1 Word count 11

3 Hive **14**
3.1 Što je Hive? 14
3.2 Način rada 14
3.3 Metastore 16

4 HDFS **18**
4.1 Definicija i opis 18
4.2 Blokovi 19
4.3 HDFS ograničenja 20

5 Twitter **21**
5.1 Ukratko o Twitter-u 21
5.2 Zašto analiza Twitter-a 22
5.3 Prikupljanje i obrada podataka 22
5.3.1 Prikupljanje podataka pomoću Apache Flume 22
5.3.2 Upravljanje podacima s Oozie-om 24

5.3.3 Upiti nad složenim podacima s Hive-om 25
5.4 Rezultati 25
5.5 Zaključak 28

Bibliografija **29**

A Word Count **31**

B Tweet u JSON formatu **34**

Uvod

Živimo u vremenu podataka. Nije lako odrediti koliko danas postoji podataka na svijetu, ali se procjenjuje da je veličina digitalnog svijeta 2011. godine bila oko 1.8 zetabajta [1] što je ekvivalentno milijardi terabajta. Navodimo par primjera, odnosno izvora podataka, kako bismo stvorili dojam o količini podataka koji nas okružuju:

- Facebook sadrži preko 10 milijardi fotografija za čije je spremanje potrebno više od jednog petabajta prostora.
- Burza u New Yorku dnevno generira oko terabajt podataka o prodaji i kupnji dionica.
- The Internet Archive sadrži oko 2 petabajta podataka i u prosjeku se povećava za 20 terabajta svaki mjesec.

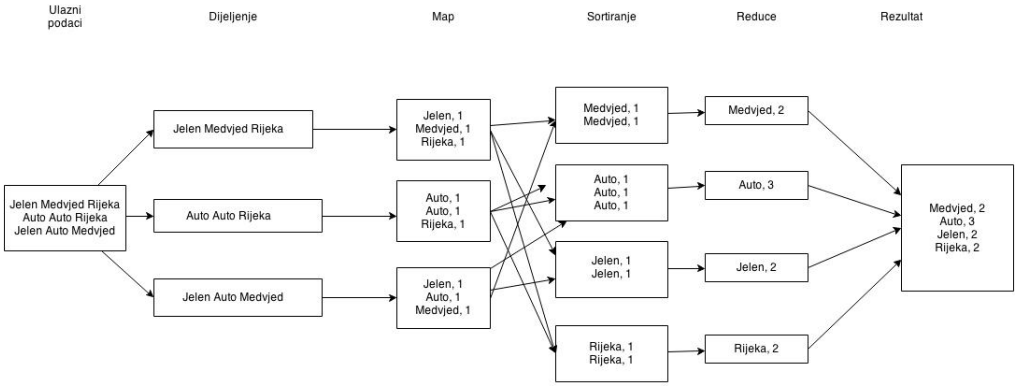
Iz ovoga zaključujemo da se radi o ogromnoj količini podataka. Pitanje je koliko to utječe na pojedinca, znajući da se većina podataka odnosi na velike kompanije poput financijskih i zdravstenih institucija.

Postojeći algoritmi za obradu podataka nisu dovoljno skalabilni za toliku količinu podataka. Zbog toga se javila potreba za algoritmom koji će zadovoljiti specifikaciju i dati rezultate u relativno brzom vremenu. Oni koji prate događanja u svijetu poslovne inteligencije, zasigurno su čuli za pojam 'Big Data'. Wikipedija taj pojam opisuje kao skup podataka koji su toliko veliki i kompleksni da se ne mogu obraditi putem tradicionalnih baza podataka ili tradicionalnih aplikacija za obradu. Big Data služi za prikupljanje, obradu i analizu velikog broja podataka, koji su opsegom, kompleksnosti i brzini dolaska veliki.

Cilj ovog rada je opisati jedan od pristupa obradi podataka na skupu veličine Interneta, a to je MapReduce algoritam. MapReduce je programski model za distibuiranu obradu podataka koji radi na način da ulazne podatke dijeli na više dijelova koji se mogu istovremeno obrađivati i time se postiže skalabilnost algoritma.

MapReduce algoritam se sastoji od dvije funkcije, map() i reduce(). Funkcija map() sortira podatke i prosljeđuje ih reduce() funkciji koja ih obrađuje. MapReduce dijeli zadani posao na više dijelova koji se paralelno izvršavaju i time se postiže skalabilnost. Primjer rada MapReduce algoritma je opisan na primjeru brojanja pojavljivanja riječi kao na slici:

[1] *Zetabajt je* 10^{21} *bajtova*



Slika 0.1: Brojanje pojavljivanja riječi pomoću MapReduce algoritma[2]

Map() funkcija dijeli primljene podatke i dodijeljuje im broj 1 kao oznaku pojavljivanja. Reduce() funkcija prima riječi s oznakama pojavljivanja i na temelju njih daje konačni rezultat. Osim Map() i reduce() funkcije, često se koristi *combiner* funkcija za smanjenje količine podataka koji se izmjenjuju između map() i reduce() funkcija. *Combiner* funkcija prima izlazne podatke Map() funkcije, te ih optimizira tako da se smanji količina podataka koja se prosljeđuje reduce() funkciji.

Važno je istaknuti da MapReduce nije baza podataka niti konkurencija relacijskim bazama podataka, međutim susrećemo mišljenja da će MapReduce u potpunosti zamijeniti baze podataka. Postoji mnoštvo zadataka koje možemo napraviti pomoću MapReduce-a. MapReduce je većinom oblikovan u Google-u i Yahoo!-u. Danas je općeprihvaćen Hadoop koji je popularna open source verzija MapReduce-a koji je napravljen od strane poznatog Apache-a. Hadoop, Hive, Oozie su neki od pojmova koje danas srećemo u Big Data svijetu.

Praktična primjena MapReduce algoritma je pokazana na primjeru analize društvene mreže Twitter. Twitter sadrži ogromnu količinu podataka o, ukratko, svemu. Korisnik Twitter-a može biti bilo tko, i podaci koje ljudi međusobno dijele su dostupni svima. Proučavanjem statusa korisnika, tzv. tweet-ova, možemo doći do brojnih korisnih rezultata i informacija. Na jednostavan način možemo doznati koja tema je trenutno najpopularnija u svijetu informatike, tko najviše piše o pojedinim stvarima, tko ima najviše sljedbenika, jesu li diskusije burzovnih brokera na Twitter-u povezane s kretanjem dionica na burzi i slično.

Prikupljeni podaci s Twittera nisu sasvim jasno strukturirani. Hadoop omogućuje programeru da sam definira kako će se ti podaci prikazivati, te dopušta korištenje formata kao što je JSON. Zbog toga je Hadoop prikladan za analiziranje podataka s Twitter-a. U radu je opisan način prikupljanja podataka pomoću Apache Flume, servisa za prikupljanje i po-

[2] http://www.alex-hanna.com/tworkshops/lesson-5-hadoop-and-mapreduce/

mjeranje velikih količina podataka, i Apache Oozie-a, koordinacijskog sustava tijeka rada, te analiza tih podataka pomoću Hive-a.

U prvom poglavlju ovog rada je opisan Apache Hadoop, njegove komponente, način spremanja, te usporedba s drugim sustavima. Najvažnija komponenta Hadoop-a je Mapreduce koji je detaljno opisan u drugom poglavlju. Osim same definicije, opisan je način rada algoritma. Nadalje, pokazana je primjena *Combiner* funkcije na jednostavnom primjeru. U nastavku poglavlja o MapReduce-u opisana je 'Word Count' aplikacija kao primjer jednostavnog MapReduce algoritma. Kod aplikacije 'Word Count' se nalazi u dodatku A.

Još jedan važan dio Hadoop-a je Apache Hive opisan u trećem poglavlju. Hive služi za proučavanje podataka, upite i analizu. Način rada je pokazan na par jednostavnih primjera, te je opisa metastore, središnje skladište metapodataka.

Glavna komponenta distribuiranog sustava za pohranu podataka koja se koristi u Hadoop-u je Hadoop-ov distribuirani datotečni sustav HDFS opisan u četvrtom poglavlju. Također su opisani i blokovi datotečnog sustava, te ograničenja HDFS-a.

U zadnjem, petom poglavlju je opis analize Twittera. Navedeni su razlozi analiziranja Twittera, te je detaljno opisano prikupljanje podataka pomoću Apache Flume-a, upravljanje tim podacima s Oozie-om, upiti nad podacima pomoću Hive-a, te u konačnici, dobiveni rezultati. Prikupljeni podaci s Twitter-a se spremaju na HDFS u JSON formatu, te je izgled tweet-a u tom formatu prikazan u dodatku B.

Poglavlje 1

Hadoop

1.1 Definicija i opis

Hadoop je nastao 2005. godine, a prvu verziju su razvili Doug Cutting i Mike Cafarella. Hadoop je stvoren kao otvorena realizacija MapReduce radnog okvira razvijenog u tvrtki Google za potrebe ostvarenja Nutch-a, open source web-pretraživača temeljenog na Javi u svrhu pretrage i indeksiranja podataka. Nutch je 'stvorio' MapReduce i distribuirani datotečni sustav kako bi zadovoljio potrebu za obradom podataka na više strojeva. Istraživanje IBM-a je pokazalo da skalabilni sustav kao što je Nutch može postići razinu performansi na grozdu veću od bilo kojeg scale-out računala. 2006.godine, odvajanjem MapReduce-a i HDFS-a od projekta Nutch, nastaje Hadoop. Hadoop je 2008.godine postao glavni projekt u sklopu Apache grupe. Danas, Hadoop ima široku primjenu, te ga koriste Twitter, Facebook, Yahoo! te mnogi drugi.

Radni okvir Hadoop je programski sustav za analiziranje velikih količina podataka koji se nalaze u pouzdanom i raspodijeljenom spremištu. 'Velike količine' podataka se odnose na baze podataka programskih sustava koji učinkovito upravljaju volumenom podataka na razini interneta kao što je rudarenje podataka (eng. Data mining).

Radni okvir Hadoop je ostvaren programskim jezikom Java, ali u isto vrijeme omogućuje korisniku programsko sučelje koje ne zahtjeva pisanje isključivo u jednom određenom jeziku, kao što je Java, već je omogućeno korištenje bilo kojeg programskog jezika koji može čitati ulazne podatke i zapisivati izlazne vrijednosti.

Apache Hadoop se sastoji od sljedećih modela:

- Hadoop Common - sadrži biblioteke i alate potrebne ostalim Hadoop modelima
- Hadoop Distributed File System (HDFS) - distribuirani sustav datoteka koji pohranjuje podatke, te pruža visoku propusnost preko grozda računala

- Hadoop YARN - platforma odgovorna za upravljanje računalnim resursima u grozdovima, te korištenje istih za raspoređivanje aplikacija korisnika
- Hadoop MapReduce - progamski model za obradu ogromnih količina podataka

1.2 Spremanje podataka i analiza

Kapacitet prostora za spremanje, odnosno kapacitet tvrdih diskova se tijekom zadnjih par godina jako povećao. Problem je što se brzina pristupa podacima, u odnosu na kapacitet diska, nije dovoljno povećala. Čitanje podataka je sporo, a zapis je još i sporiji. Opisani problem se može riješiti tako da se podaci čitaju s više diskova u isto vrijeme. Da bi to ostvarili potrebno je podijeliti podatke na više diskova i time olakšati rad s istim.

Prvi problem koji trebamo riješit je hardverski. Kada upotrebljavamo veliki broj manjih komada diska, veća je vjerojatnost da će se dogoditi greška. Uobičajeni način izbjegavanja gubitka podataka je repliciranje: dodatne kopije podataka se čuvaju u sustavu i dostupne su u slučaju neuspjeha. Takav način rada koristi RAID (Redudant Array of Independent Disks), doh HDFS dijeli datoteku na blokove koji se kopiraju na više čvorova.

Drugi problem je taj što većina analiza podataka zahtjeva mogućnost kombiniranja podataka, podaci koji se čitaju s jednog diska možda trebaju biti u kombinaciji s podacima s bilo kojeg od ostalih diskova. Razni distibuirani sustavi omogućuju da se podaci kombiniraju s više izvora, ali pravilno izvođenje je dosta složeno. MapReduce omogućuje model programiranja koji rješava problem koristeći setove ključeva i vrijednosti, a najvažnija stvar je da postoje dva načina računanja, map i reduce, i upravo se na taj način ostvaruje kombiniranje podataka.

To je, ukratko, ono što Hadoop pruža: pouzdan zajednički prostor za pohranu i analizu sustava. Pohranu osigurava HDFS, a analizuMapReduce iz čega proizlazi zaključak da su to dva najvažnija dijela Hadoop-a.

1.3 Usporedba s drugim sustavima

Hadoop, točnije MapReduce, se temelji na pretpostavci da se cijeli skup podataka, ili barem veći dio, procesira za svaki poslani upit. Upravo to je njegova snaga. MapReduce je zapravo hrpa upita koji se izvršavaju na čitavom skupu podataka i rezultati se dobiju u razumnom vremenskom razdoblju. Zahvaljujući velikoj skalabilnosti možemo izračunati već poznate statistike na ogromnim količinama podataka. Hadoop i MapReduce su dali odgovore na mnoga pitanja, ali su u isto vrijeme postavili nova. Na primjer, Mailtrust, sustav za raspodijelu elektroničke pošte, koristi Hadoop. Jedan od upita koji su napisali je bio pronalazak geografske distribucije svojih korisnika. Koristeći MapReduce nad stotinama

gigabajta podataka uspjeli su dobiti tražene podatke, što je prije Hadoop-a bilo gotovo nemoguće zbog količine podataka.

Razlog zašto je MapReduce toliko potreban je sljedeći: vrijeme traženja se poboljšava mnogo sporije nego vrijeme prijenosa podataka. Traženje je proces pomicanja glave diska do određenog dijela diska s kojeg čitamo podatke ili ih zapisujemo. To traženje karakterizira latenciju operacije na disku, dok brzina prijenosa podataka određuje propusnost.

Usporedimo MapReduce s RDBMS-om (Rational Database Management System). RDBMS je sustav upravljanja bazom podataka koji se temelji na relacijskom modelu kojeg danas koriste mnoge popularne baze podataka. RDBMS zahtjeva strukturirane podatke, podatke koji su organizirani u određenom formatu, kao što je XML dokument ili tablica baze podataka, dok MapReduce radi dobro i na nestruktuiranim ili polu-struktuiranim podacima. Drugim riječima, ulazne podatke za MapReduce određuje osoba koja izvršava analizu podataka. Zbog toga MapReduce ima bolju skalabilnost od RDBMS-a. MapReduce je pogodan za probleme koji zahtjevaju analizu čitavog skupa podataka, dok je RDBMS dobar za ažuriranje koje zahtjeva nisku latenciju i ažuriraju se relativno male količine podataka. MapReduce odgovara aplikaciji koja jednom zapiše podatke i čita ih više puta, dok RDBMS pogoduje podacima koji se neprestano ažuriraju. Navedene i ostale razlike su prikazane u tablici 1.1.

Tablica 1.1: Usporedba MapReduce i RDBMS

	MapReduce	RDBMS
Veličina podataka	Petabajti	Gigabajti
Pristup podacima	Upit	Upiti i interakcija
Rad s podacima	Piše jednom, čita više puta	Čita i piše više puta
Struktura	Dinamička	Statička
Integritet	Nizak	Visok
Skaliranje	Linearno	Nelinearno

Razvojni timovi u području HPC-a (The High Performance Computing) i Grid Computing-a su radili obradu velikih količina podataka godinama, koristeći API-je kao je MPI (Messade Passing Interface). Općenito, način rada HPC-a je distibucija posla skupu strojeva koji imaju pristup zajedničkom datotečnom sustavu. Takav način rada funkcionira za zadatke računanja, ali je problematičan kad je potreban pristup većim količinama podataka (stotinama gigabajta) budući da je mrežna propusnost usko grlo i čvorovi ne mogu obavljati svoj posao. MapReduce pokušava rasporediti podatke na čvorove koji vrše računanje i time postiže brži pristup podacima. Ova značajka je bit samog MapReduce-a, poznata kao podatkovna lokalnost (eng. data locality). MPI daje veliku kontrolu programeru, ali zahtjeva da programer svlada u potpunosti način rada i protok podataka, kao

i složene algoritme za analizu tih podataka. MapReduce zahtjeva jedino da programer razmišlja u smislu parova ključa i vrijednosti kao izlaznih vrijednosti map() funkcije.

Poglavlje 2

MapReduce

MapReduce je programski model za obradu velikih količina podataka pomoću paralelnog, distibuiranog algoritma na grozdu računala. Kao što je već spomenuto, Hadoop nije ograničen na MapReduce programe pisane u samo jednom programskom jeziku, već podržava više njih uključujući Javu, Python i C++. MapReduce algoritam se sastoji od dvije procedure, odnosno funkcije. Map() funkcija je zadužena za filtriranje i sortiranje podataka. Reduce() obrađuje podatke koji su prethodno sortirani map() funkcijom. Čitav MapReduce funkcionira na način da se posao dijeli na različite podzadatke koji se paralelno izvršavaju omogućavajući redudanciju i toleranciju grešaka.

2.1 Opis rada algoritma

Postoje dvije vrste čvorova koji kontroliraju proces izvedbe zadatka: jobtracker i tasktracker. Jobtracker koordinira svim poslovima koji se izvode na sustavu raspoređujući zadatke na tasktrackere i vodi evidenciju o cjelokupnom napretku posla. Tasktracker pokreće zadatak i šalje jobtrackeru izvješća o napretku. Ukoliko se neki zadatak nije uspio izvršiti, jobtracker ga može dodijeliti drugom tasktrackeru. Hadoop uvijek pokuša pokrenuti map zadatak na čvoru gdje je ulazni podatak, što se naziva optimizacija lokalnosti podataka (eng. data locality optimization) budući da ne koristi propusnost grozda.

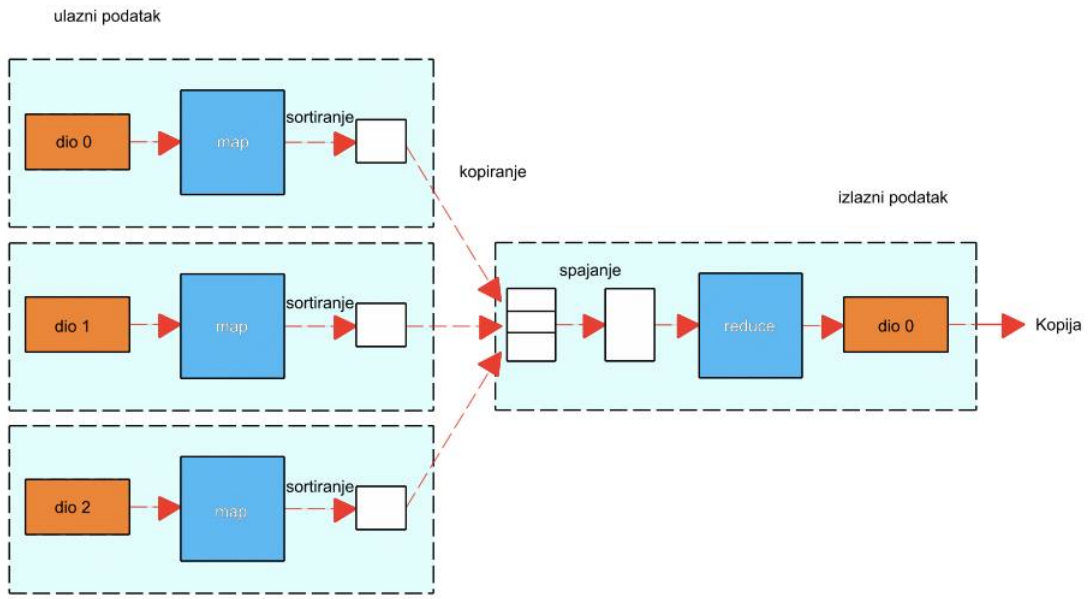
Hadoop dijeli ulazne podatke MapReduce-a na dijelove fiksne veličine koje nazivamo ulazni dijelovi (eng. input splits) ili jednostavno dijelovi (eng.splits). Za svaki dio, Hadoop kreira zadatak kojeg prvo obrađuje unaprijed definirana Map funkcija. Stvaranjem više manjih dijelova smanjuje se ukupno vrijeme obrade u odnosu na vrijeme koje bi bilo potrebno za obradu cjeline. Ako obradu dijelova izvodimo paralelno, obrada je bolje raspodijeljena ako su dijelovi manji budući da brži strojevi mogu obraditi više dijelova tijekom izvršavanja. S druge strane, ako su dijelovi premali, gubi se vrijeme na upravljanje dijelovima i kreiranje map zadataka i to uvelike utječe na ukupno vrijeme izvedbe zadatka.

Za većinu poslova, optimalna veličina dijela je veličina HDFS bloka, zadane vrijednosti 64MB, iako se može mijenjati za grozdove ili naknadno nakon stvaranje datoteke.

Razlog zašto je optimalna veličina jednaka veličini bloka je to što je to najveća veličina ulaznog podatka koja će sigurno biti pohranjena u jednom čvoru. Ako bi dio bio veličine dva bloka, HDFS čvor bi pohranio oba bloka, pa bi neki dio morao biti prebačen mrežom do čvora koji pokreće map zadatak, što je manje učinkovito nego pokretanje cijelog map zadatka lokalno.

Map zadatak zapisuje svoje izlazne vrijednosti na lokalni disk, ne na HDFS. Izlazna vrijednost map zadatka nije konačna izlazna vrijednost, već se u nastavku obrađuje reduce zadatkom da bi se dobio konačan rezultat i izvršio posao, tako da nema potrebne za spremanjem izlaznih podataka map zadatka na HDFS. Ako se čvor koji izvodi map zadatak sruši prije nego izlaznu vrijednost preuzme reduce zadatak, tada Hadoop ponovno pokrene map zadatak na drugom čvoru da bi dobio izlaznu vrijednost map-a.

Izlazne vrijednosti svih map zadataka čine ulaznu vrijednost reduce zadatka. Sortirani izlazni podaci map-a se prenose mrežom do čvora u kojem počinje reduce zadatak gdje se spoje u jednu cjelinu i proslijede reduce funkciji. Izlazna vrijednost reduce zadatka se obično pohranjuje u HDFS. Za svaki HDSF blok izlazne vrijednosti reduce-a, prva kopija se pohranjue na lokanom čvoru, dok se ostale pohranjuju na off-rack čvorovima, tj.slučajno odabranim čvorovima koji se razlikuju od lokalnog čvora.



Slika 2.1: Tok podataka s jednim reduce zadatkom[1]

2.2 Combiner funkcija

MapReduce može biti ograničen propušnošću grozda, pa se zbog toga smanjuje količina podataka koji se izmjenjuju između map i reduce zadatka. Hadoop dopušta korisniku da koristi combiner funkciju koja se izvršava na izlaznim vrijednostima zadatka i formira ulazne vrijednosti reduce zadatka. Combiner funkcija služi isključivo za optimizaciju, pa Hadoop ne jamči koliko puta će se ta funkcija pozvati. Neovisno o tome koliko puta se combiner funkcija pozove i izvrši, izlazni podaci reduce zadatka moraju ostati isti. Način rada combiner funkcije je najlakše opisati pomoću primjera.

Primjer 2.2.1. *Potrebno je odrediti maksimalnu temperaturu tijekom godine, kao primjer ćemo uzeti 2013.godinu. Pretpostavimo da su očitanja temperature obrađena s dvije map funkcije. Neka je izlazna vrijednost prve funkcije:*

(2013,0) (2013,10) (2013,20)

a izlazna vrijednost druge funkcije:

(2013,25) (2013,15)

Reduce funkcija se poziva na listi vrijednosti

(2000, [0,10,20,25,20])

i kao izlaznu vrijednost daje

(2013, 25)

budući da je 25 maksimalna vrijednost u listi. Ako definiramo combiner funkciju koja će naći maksimalnu temperaturu za svaki izlaz map funkcije, tada se reduce funkcija, u ovom slučaju, poziva za listu

(2013, [20,25])

i daje isti rezultat kao prije.

Važno je naglasiti da combiner funkcija ne zamjenjuje reduce, već se koristi samo za smanjivanje količine podataka koja se prenosi između map i reduce funkcija. Upravo iz tog razloga, uvijek je poželjno razmotriti ideju uvođenja combiner funkcije.

2.3 Razvoj MapReduce aplikacije

Pisanje MapReduce programa, kao i većine ostalih, slijedi neki standardni način i redoslijed pisanja. Na samom početku, potrebno je kreirati map i reduce funkcije, te ih testirati na

[1] White, Tom: Hadoop: The definitive Guide

malim količinama podatak da se uvjerimo da rade ono što želimo. Nakon toga, povećamo količinu podataka kako bi provjerili moguće greške u kodu, te poboljšali map i reduce funkciju. Ako program radi prema očekivanju, možemo ga pustiti na grozd. Vrlo vjerojatno ćemo nakon toga otkriti još nepravilnosti. Ispravljanje pogrešaka na grozdu je vrlo zahtjevno, pa slijedeći određene metode pisanje nastojimo to izbjeći. Na samom kraju, program je spreman za moguću optimizaciju i poboljšanje brzine rada programa.

2.3.1 Word count

Prvi program koji naučimo pisati u nekom programskom jeziku je obično 'Hello, world!'. Kada su u pitanju Hadoop i MapReduce, onda se radi o programu *Word count*, odnosno 'Brojaču riječi'. Kao što i sam naziv kaže, radi se o broju pojavljivanja riječi u tekstu. 'Word count' je odličan primjer za razvijanje razmišljanja na način MapReduce algoritma, map i reduce funkcije su trivijalne, pa ga je zbog toga jednostavno razumjeti.

U nastavku su opisane komponente 'Word Count' aplikacije i njihove funkcije:

- *FileInputFormat*: čita sve datoteke iz zadanog direktorija, te ih prosljeđuje TextInputFormat-u koji ih raspodjeljuje maperima
- *TextInputFormat*: zadani format ulaznih podataka za Hadoop je tekst, odnosno TextInputFormat koji čita liniju po liniju...
- *Word Count Mapper*: klasa koja radi sljedeće: svaku liniju teksta koju dobije od InputFormat-a oblikuje u žeton koji sadrži riječ i broj 1 koji označuje da se ta riječ pojavila u tekstu.
- *Word Count Reducer*: prima mapu svih riječi i broja njihovih pojavljivanja. Ukoliko se ne koristi combiner funkcija, Reducer će primiti riječi i kolekciju brojeva 1. U ovom primjeru, Reducer ćemo korisiti i kao combiner funkciju, pa ćemo imati kolekciju brojeva 1 koje treba zbrojiti.
- *TextOutputFormat*: klasa koju koristimo u ovom primjeru i u kojoj definiramo da će ključevi biti tipa Text, a vrijednosti tipa InWritable.
- *FileOutputFormat*: zapisuje rezultate primljene od TextOutputFormata u izlazni direktorij.

```
1
2       public void map( LongWritable key,
3                        Text value,
4                        OutputCollector<Text, IntWritable> output,
5                        Reporter reporter)
```

```
6         {
7             String text = value.toString().toLowerCase();
8             text = text.replaceAll( "'", "" );
9             text = text.replaceAll( "[^a-zA-Z]", " " );
10            StringTokenizer st = new StringTokenizer( text );
11            while( st.hasMoreTokens() )
12            {
13                word.set( st.nextToken() );
14                output.collect( word, one );
15            }
16        }
```

Listing 2.1: Map funkcija

```
1
2         public void reduce( Text key, Iterator<IntWritable> values,
3                             OutputCollector<Text, IntWritable> output,
4                             Reporter reporter)
5         {
6             int count = 0;
7             while( values.hasNext() )
8             {
9                 count += values.next().get();
10            }
11            output.collect( key, new IntWritable( count ) );
12        }
```

Listing 2.2: Reduce funkcija

Možemo primjetiti da su map i reduce funkcije pisane u istoj datoteci. Ne postoji strogo pravilo koje definira gdje pišemo funkcije, ali obično se map i reduce funkcije pišu u istoj datoteci, osim ako su funkcije složene, pa ih zbog lakšeg razumijevanja pišemo odvojeno.

Budući da je kod sam po sebi dosta jednostavan, nije potrebno detaljno objašnjenje. MapReduce nam pruža dosta elegantno rješenje problema i često će se dogoditi da ćemo potrošiti više vremena na razmišljanje o rješenju, nego na samo pisanje rješenja.

Nakon pisanja koda, potreban nam je samo veći tekst za analizu. 'Project Gutenberg' je internet stranica na kojoj možemo pronaći preko 100,000 besplatnih e-knjiga. Jedna od njih je 'Moby Dick' i upravu nju smo koristili za brojanje riječi. Kod izvršavamo pomoću naredbe *hadoop jar* kojoj proslijeđujemo direktorij u kojem se nalazi tekst koji analiziramo i direktorij za spremanje rezultata.

Manji dio sadržaja direktorija u kojem se nalaze rezultati izgleda ovako:

Tablica 2.1: Rezultati

riječ	broj pojavljivanja
a	4687
aback	2
abaft	2
...	
i	1320
is	920
..	
king	40
...	
man	731
..	
sea	250
...	
whale	912
...	
your	251
youre	6
youve	1

Zbog jednostavnosti, način rada 'Word Count'-a je pokazan na primjeru jedne datoteke. Tako malu količinu podataka možemo analizirati i drugim algoritmima. Skalabilnost Hadoop-a i MapReduce algoritma dolazi do izražaja kada analiziramo veće količine podataka (stotine gigabajta), vrijeme obrade je znatno kraće u odnosu na ostale algoritme.

Poglavlje 3

Hive

3.1 Što je Hive?

Apache Hive je infrastruktura za skladištenje podataka izgrađena na Hadoop-u, te služi za proučavanje podataka, upite i analizu. Iako je razvijena od strane Facebooka, danas ju koriste i razvijaju i druge tvrtke kao što je Netflix.

Apache Hive podržava analizu velikih količina podataka spremljenih na Hadoop-ovu HDFS-u, te pruža SQL jezik HiveQL i podržava MapReduce algoritam. Hive sprema metapodatke u bazi podataka Apache Derby i drugim klijent/server bazama kao što je MySQL. Zadaća Hive-a je prebacivanje SQL upita u niz MapReduce poslova koji se mogu izvršavati na Hadoop grozdu. Hive organizira podatke u tablice koje osiguravaju sredstva za priključivanje strukture podacima pohranjenim u HDFS-u. Metapodaci, kao što su sheme tablica, su pohranjenu u bazi koja se naziva metastore.

3.2 Način rada

Kao što je već spomenuto, Hive organizira podatke u tablice. Tablica se kreira na sljedeći način:

```
1 CREATE TABLE podaci (godina STRING, temperatura INT, kvaliteta INT)
2 ROW FORMAT DELIMITED
3 FIELDS TERMINATED BY '\t';
```

Prva linija je deklaracija tablice *podaci* koja ima tri kolone: godina, temperatura i kvaliteta. Tip svake kolone mora, također, biti naveden. U ovom slučaju, godina je string, temperatura i kvaliteta su integer-i. ROW FORMAT naredba se odnosi na HiveQL, dok je prvi dio bio standardni SQL. ROW FORMAT naredba određuje da je svaka kolona odvojena tab-om, dok su reci odvojeni znakom za novi redak. Nakon kreiranja tablica, potrebno je napuniti Hive podacima. Način unosa podataka je sljedeći:

```
LOAD DATA LOCAL INPATH 'primjer.txt'
OVERWRITE INTO TABLE podaci;
```

Navedenom naredbom podaci se učitavaju u prethodno kreiranu tablicu *podaci* iz datoteke 'primjer.txt', a po potrebi je moguće učitavanje iz više datoteka. Riječ OVERWRITE u naredbi kaže da se obrišu svi postojeći podaci i zatim upišu novi, tj.stari podaci se prebrišu novima. Nakon učitavanja podataka, možemo pokrenuti željeni upit:

```
hive>SELECT godina, MAX(temperatura)
    >FROM record
    >WHERE temperatura != 9999
    >AND kvaliteta = 0 OR kvaliteta = 1
    >GROUP BY godina;
```

Ovo je naizgled obični SQL upit. Razlika je jedino u tome što Hive pretvara dani upit u MapReduce zadatak. Moć Hive-a je upravo njegova mogućnost obrade 'sirovih' podataka.

```
hive> CREATE EXTERNAL TABLE tweets (
    >   id BIGINT,
    >   created_at STRING,
    >   source STRING,
    >   favorited BOOLEAN,
    >   retweet_count INT,
    >   retweeted_status STRUCT<
    >     text:STRING,
    >     user:STRUCT<screen_name:STRING,name:STRING>>,
    >   entities STRUCT<
    >     urls:ARRAY<STRUCT<expanded_url:STRING>>,
    >     user_mentions:ARRAY<STRUCT<screen_name:STRING,name:STRING>>,
    >     hashtags:ARRAY<STRUCT<text:STRING>>>,
    >   text STRING,
    >   user STRUCT<
    >     screen_name:STRING,
    >     name:STRING,
    >     friends_count:INT,
    >     followers_count:INT,
    >     statuses_count:INT,
    >     verified:BOOLEAN,
    >     utc_offset:INT,
    >     time_zone:STRING>,
    >   in_reply_to_screen_name STRING
    > )
    > PARTITIONED BY (datehour INT)
    > ROW FORMAT SERDE 'com.cloudera.hive.serde.JSONSerDe'
    > LOCATION '/Users/hadoop/tweets';
OK
```

Slika 3.1: Kreiranje tablice *tweets* pomoću Hive-a

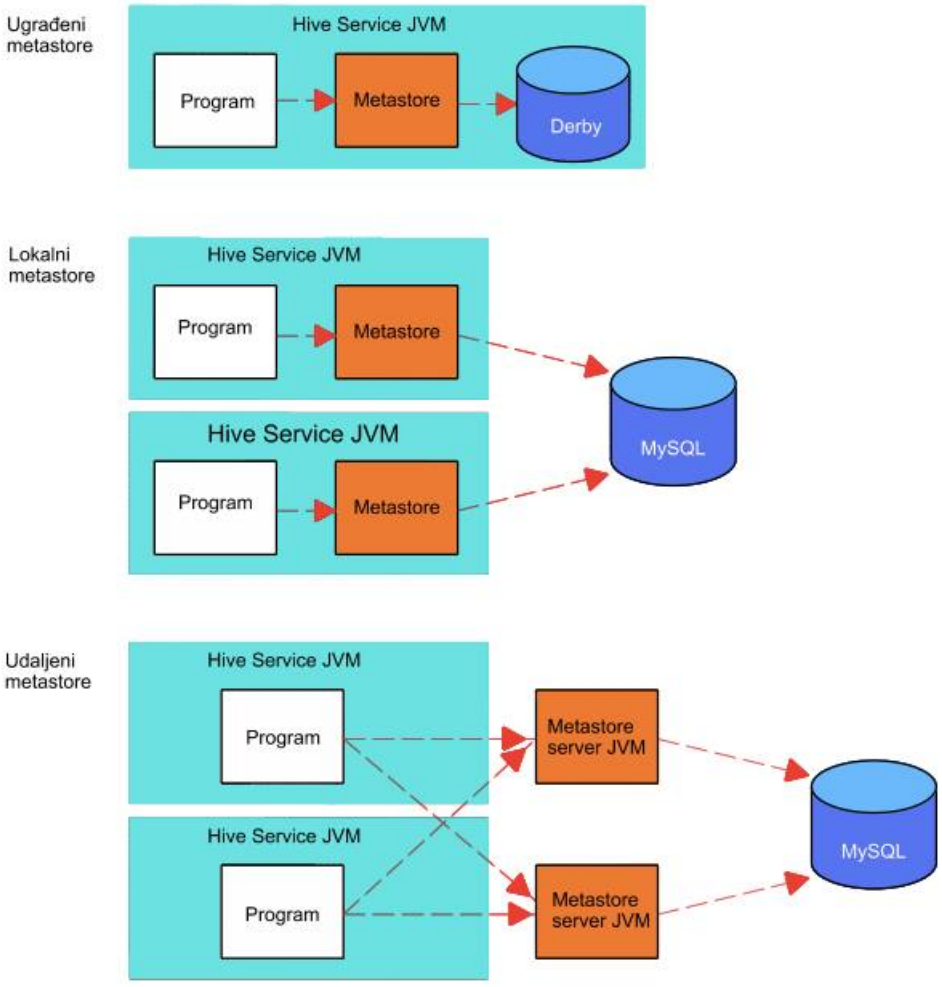
3.3 Metastore

Metastore je središnje skladište metapodataka. Podijeljeno je u dva dijela: servis i skladište za podatke. U pravilu, metastore servis radi na istom JVM, odnosno virtualnom stroju, kao i Hiveservis i sadrži ugrađenu Derby bazu podataka uz potporu lokalnog diska. Takva konfiguracija se naziva ugrađena metastore konfiguracija. Korištenje ugrađenog metastore-a je jednostavan način za pokretanje Hive-a, ali samo jedna ugrađena Derby baza može pristupati podacima s diska u isto vrijeme, što znači da možemo imati pokrenutu samo jednu sesiju nad istim metastore-om. Rješenje za korištenje više sesija u isto vrijeme je korištenje samostalne baze podataka. Takva konfiguracija se naziva lokalni megastore, budući da metastore servis koristi isti proces kao i Hive servis, ali se spaja na bazu u zasebnom procesu, bilo na istom stroju, bilo na udaljenom.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| metastore          |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.03 sec)
```

Slika 3.2: metastore kreiran pomoću MySql-a

[1]White, Tom: Hadoop: The definitive Guide



Slika 3.3: Konfiguracije Metastore-a[1]

Poglavlje 4

HDFS

4.1 Definicija i opis

Hadoop-ov distribuirani datotečni sustav, HDFS (Hadoop Distributed File System), je glavna komponenta distribuiranog sustava za pohranu podataka koja se koristi u Hadoop-u. HDFS, također, može poslužiti i kao samostalni distribuirani datotečni sustav. Apache Hadoop okvir uključuje brojne kompenente koje podržavaju distribuirano računanje za rješavanje problema s velikih količina podataka. HDFS omogućuje korisnicima API za implementaciju drugih distribuiranih datotečnih sustava ili sustava za pohranu podataka kao što su *Amazon Simple Storage Servise*, *Parallel Virtual File System* i drugi.

HDFS je datotečni sustav s hijerarhijom datoteka i direktorija. Implementiran je s dva servisa na skupu poslužitelja:

- *NameNode* sadrži stablo direktorija i upravlja prostorom i pristupom datotekama od strane klijenta
- *DataNode* pohranjuje i upravlja blokovima podataka kao lokalnim datotekama na servera preko ostatka grozda.

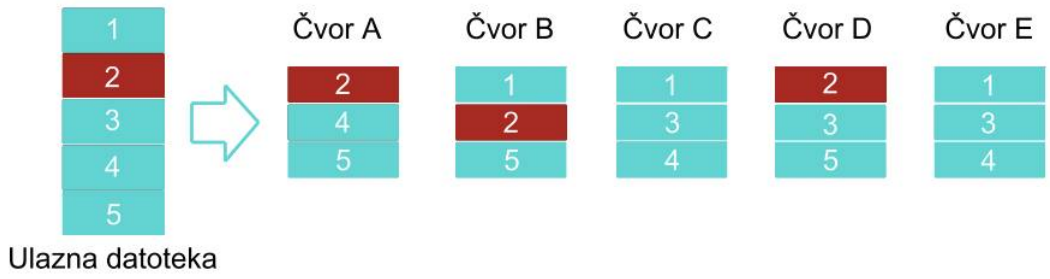
HDFS dijeli datoteke u velike blokove, koji se kopiranju na više DataNode čvorova. Dijeljenje se vrši zbog redudancije i dostupnosti podatkovnih datoteka. Metadata se održava pomoću NameNode-a, koji izvršava operacije otvaranja, zatvaranja i preimenovanja datoteka, te mapira blokove na DataNode-ove. U postojećoj verziji HDFS-a, jedan NameNode čuva cijeli prostor imena (eng. namespace) u RAM memoriji dok se podaci repliciraju i pohranjuju u DataNode-u. DataNode čvorovi poslužuju upite klijentana i upravljaju operacijama nad blokovima podataka koje suodređene NameNode čvorom, kao što su brisanje i kopiranje.

4.2 Blokovi

Disk ima blokovnu veličinu koja označava najmanju količinu podataka koja može biti čitana ili pisana. Datotečni sustav za jedan disk dijeli podatke na blokove. Blokovi datotečnog sustava su obično veliki nekoliko kilobajta, dok se blokovi diska obično veliki 512 bajtova.

HDFS, također, ima koncept bloka, ali se radi o većoj veličini - podrazumijevanja vrijednost je 64MB. Kao kod datotečnog sustava za jedan disk, datoteke u HDFS su podijeljenje u komade veličine bloka i pohranjene su kao neovisne cjeline. Za razliku od datotečnog sustava za jedan disk, datoteka u HDFS-u koja je manja od veličine bloka ne zauzima čitavu vrijednost bloka.

Zašto su blokovi u HDFS tako veliki? U usporedbi s diskovnim blokovima, blokovi HDFS-a su jako veliki, kako bi se smanjili troškovi traženja. Čineći blok dovoljno velikim, vrijeme za prijenos podataka s diska može biti znatno veće od vremena traženja početka bloka. Vrijeme prijenosa velikih datoteka raspoređenih u blokove djeluje na brzinu prijenosa diska. Brzi izračun pokazuje da ako je vrijeme traženja oko 10ms, a brzina prijenosa je 100MB/s, da bi vrijeme traženja ubrzali za 1% brzine prijenosa, trebamo blok veličine 100MB. Zadana vrijednost bloka je 64MB, iako mnoge HDFS instalacije koriste 128MB. Ova brojka će se mijenjati naviše kako bude rasla brzina prijenosa na novim generacijama diskova.



Slika 4.1: Raspoređivanje podataka pomoću HDFS-a[1]

[1]White, Tom: Hadoop: The definitive Guide

4.3 HDFS ograničenja

HDFS može pohraniti vrlo veliki skup podataka pouzdano i usmjeriti te pkupove podataka korisničkoj aplikaciji. Međutim, kako količina podataka raste, HDFS i druge komponente Hadoop-a će postat ograničene svojim dizajnom. Budući da su metapodaci spremljeni u RAM memoriji na jednom NameNode čvoru, postoji ograničenje koliko objekata možemo pohraniti u tu memoriju. Osim toga, velika opterećenja u sustavu imaju potencijal za stvaranje uskog grla poražavajući moć NameNode čvora. Apache Hadoop zajednica radi na tome da prevladaju ove granice skalabilnosti. Jedan od mogućih načina rješavanja problema je pomoću 'federacije' Konfiguracija 'federacija' je povratno kompatibilna i omogućuje da postojeći NameNode čvor radi bez ikakvih promjena. Nova konfiguracija je dizajnirana tako da svi čvorovi u grozdu imaju istu konfiguraciju, bez potrebe za implementacijom različitih konfiguracija, ovisno o vrsti čvora u grozdu.

```
karolinas-air:hadoop-1.2.1 hadoop$ jps
26705 DataNode
26792 SecondaryNameNode
26968 RunJar
27014 Jps
26939 TaskTracker
26618 NameNode
26854 JobTracker
```

Slika 4.2: Pokrenuti čvorovi na Hadoop-u

Poglavlje 5

Twitter: analiza podataka

Najčešće analizirana područja su financije, politika, organizacije, energija. Mreže se, općenito, modeliraju tako da ih se prezentira grafom i analiziraju se korištenjem teorije grafova. Prebacivanje mreže u graf nam omogućava uočavanje veza koje prije nisu mogle biti uočene, uvid u strukturu mreže, ispitivanje povezanosti i optimizaciju.

Zahvaljujući MapReduce algoritmu možemo u kraćem vremenu obraditi i analizirati jako veliku količinu podataka. U nastavku rada opisana je analiza podataka s društvene mreže Twitter pomoću Hadoop-a i MapReduce-a, od načina prikupljanja i spremanja podataka, do rezultata za traženi upit.

5.1 Ukratko o Twitter-u

Twitter je društvena mreža za tzv.mikro-blogging, odnosno za slanje i čitanje kratkih poruka koje se nazivaju tweet-ovi i imaju ograničenje na najviše 140 znakova. Twitter su u ožujku 2006.godine osnovali Jack Dorsey, Evan Williams, Biz Stone i Noah Glass. Velikom brzinom je postao popularan i već 2012. godine je bilo 500 milijuna registriranih korisnika, koji su objavljivali oko 340 milijuna tweet-ova dnevno. 2013.godine, Twitter je bio među 10 najposjećenijih internet stranica. Tweet-ovi su po definiciji vidljivi svima, ali pošiljatelj može ograničiti vidljivost svojih tweet-ova na samo određene osobe. Karakteristika tweet-ova su i ljestve (eng.hashtags) # pomoću kojih je moguće grupirati tweet-ove prema naslovu, sadržaju, osobi koja se spominje . . . Ukratko, Twitter možemo gledati kao izvor ogromnih količina podataka čijom analizom možemo doći do određenih rezultata i pokazatelja.

5.2 Zašto analiza Twitter-a

Društvene mreže su stekle veliku popularnost u svijetu zahvaljujući marketingu, a Twitter je koristan alat pomoću kojeg tvrtke promoviraju svoje proizvode. Twitter omogućava da tvrtke komuniciraju izravno s korisnicima, a korisnici daju javno svoju kritiku proizvoda. Ako želimo analizirat određene podatke radi dobivanja rezultata, rezultat će biti točniji što je količina podataka veća. Nije lako doći do ogromnih količina određenih podataka, i upravo zbog toga je Twitter idealan.

Korisnik Twittera, nazovimo ga Luka, slijedi određeni broj ljudi. Isto tako, drugi ljudi slijede njega, tzv.sljedbenici. Kad Luka objavi nešto, njegovi sljedbenici to vide. Luka također može retweet-ati ono što su drugi objavili, to jest može podijeliti njihove tweet-ove tako da njegovi sljedbenici to vide, iako oni ne slijede osobu koja je tweet prvotno objavila. Tako se poruke mogu proširiti na veliku skupinu ljudi. Budući da Twitter prati broj retweet-ova, pomoću toga možemo procijeniti koliko je neka osoba popularna na Twitteru, koji korisnik objavljuje najviše retweet-ova i slično.

5.3 Prikupljanje i obrada podataka

SQL upiti se mogu koristiti da bi odgovorili na određena pitanja: Koji korisnik ima najviše retweet-ova? Kojeg korisnika prati najviše ljudi? . Postavljanje upita nad Twitter podacima u tradicionalnom RDBMS je nezgodno, jer Twitter Streaming API emitira tweet-ove u JSON formatu što može biti dosta složeno. U Hadoop-ovu sustavu, projekt Hive pruža sučelje za upite koje se može koristiti za upite nad podacima koji se nalaze u HDFS-u. Jezik u kojem se pišu upiti je jako sličan SQL jeziku, ali nam omogućuje lakše modeliranje složenih upita, tako da se lako može postaviti upit za vrstu podataka koju imamo.

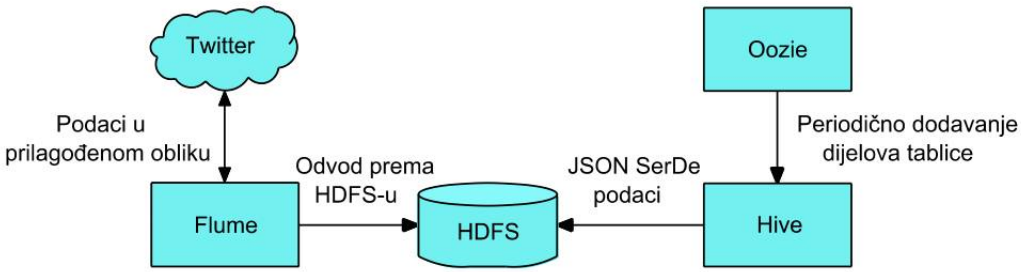
Na početku, potrebno je spremiti podatke s Twitter-a u HDSF, te nakon toga možemo reći Hive-u gdje se podaci nalaze i na koji način ih je potrebno čitati.

Gornji dijagram pokazuje kako neke od CDH (Cloudera's Distribution Including Apache Hadoop) komponenti mogu biti spojene skupa kako bi izgradile cjevovod podataka (eng. data pipeline) koji je potreban za željenu analizu podataka.

5.3.1 Prikupljanje podataka pomoću Apache Flume

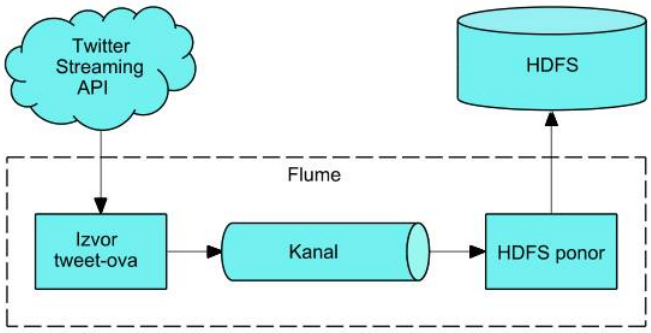
The Twitter Streaming API omogućuje stalno dotok tweet-ova. Najjednostavniji način spremanja tih podataka je korištenje komponente unutar CDH koja automatski sprema datoteke iz API-a u HDFS. Apache Flume je sustav za 'ubrizgavanje' podataka koji je konfiguriran definiranjem krajnjih točaka u protoku podataka pod nazivom izvori i ponori (eng.

[1] http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/



Slika 5.1: Kontrolirani tok podataka[1]

sources and sinks). Internet stranica Apache Flume-a opisuje Flume kao 'distibuirani, pouzdani i dostupni servis za učinkovito prikupljanje i pomjeranje velikih količina podataka'. U Flume-u, svaki pojedini dio podataka (u našem slučaju, tweet-ovi) se naziva događaj. Izvori proizvode događaje i šalju ih kroz kanal koji vodi sve do ponora. Nakon toga, ponor zapisuje događaje na unaprijed određeno mjesto. Flume podržava neke standardne izvore podataka, kao što je syslog ili netcat.



Slika 5.2: Flume cjevovod[2]

Za naš slučaj, potrebno je izgraditi prilagođeni izvor koji pristupa Twitter Streaming API-u i šalje tweet-ove kroz kanal do ponora koji ih zapisuje na HDFS. Osim toga, možemo koristiti prilagođene izvore za filtriranje tweet-ova na temelju ključnih riječi kako bi pronašli bitne tweet-ove, što je bolje nego da koristimo čitav skup tweet-ova.

[2]http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/

Za početak, potrebno je instalirati Flume, uz uvjet da je Hadoop već instaliran. Unutar datoteke *flume.conf* definiramo lokaciju, odnosno HDFS, na koju će se prikupljeni tweet-ovi spremati. Vrijednosti *consumerKey*, *consumerSecret*, *acessToken* i *acessTokenSecret* su dostupne na *dev.twitter.com*, te su jedinstevene za svakog korisnika. Direktorij *flume-sources* sadrži prilagođeni Flume kod koji povezuje Twitter Streaming API i sprema podatke na HDFS u JSON formatu.

```
# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel
##agent.channels.memoryChannel.capacity = 100
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = [redacted]
TwitterAgent.sources.Twitter.consumerSecret = [redacted]
TwitterAgent.sources.Twitter.accessToken = [redacted]
TwitterAgent.sources.Twitter.accessTokenSecret = [redacted]

TwitterAgent.sources.Twitter.keywords = hadoop, big data, analytics, bigdata, cloudera, data science, da
ta scientiest, business intelligence, mapreduce, data warehouse, data warehousing, mahout, hbase, nosql,
 newsql, businessintelligence, cloudcomputing

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs: //localhost:8888/tweet/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000
```

Slika 5.3: Datoteka *flume.conf* unutar koje upisujemo podatke s Twitter aplikacije

5.3.2 Upravljanje podacima s Oozie-om

Nakon što učitamo podatke s Twitter-a i spremimo ih na HDFS, keiramo vanjsku tablicu u Hive-u pomoću koje možemo koristiti potrebne upite. Korištenje vanjske tablice nam omogućava da koristimo podatke bez da ih premještamo s lokacije na HDFS-u. Da bi omogućili skalabilnost, moramo dijeliti tablicu kako se povećava količina podataka. Podijeljena tablica nam omogućava da 'odrežemo' podatke koje smo pročitali nekim upitom, što rezultira boljom performansom kada se radi o velikim količinama podataka. Međutim, Twitter API će nastaviti učitavati tweet-ove i Flume će neprestano stvarati nove datoteke. Moguće je automatizirati periodični proces koji će dodavati dijelove tablice kako podaci dolaze.

Apache Oozie je koordinacijski sustav tijeka rada koji se može koristiti za rješavanje ovog problema. Oozie je iznimno fleksibilan sustav za izradu tijeka rada, koji se izvršava na temelju niza kriterija. Na primjer, možemo postaviti da se naredba ALTER TABLE

izvršava svakih sat vremena, tj. da se kreira dio tablice koji će sadržavat podatke koji su pristigli u zadnjih sat vremena. To nam osigurava da uvijek imamo svježe (eng. up-to-date) podatke.

5.3.3 Upiti nad složenim podacima s Hive-om

Prije nego krenemo s postavljanjem upita za podatke, moramo osigurat da Hive tablica ispravno interpretira JSON (JavaScript Object Notation) podatke. Po defniciji, Hive očekuje da ulazne datoteke koriste određeni format, ali naši podaci s Twittera su u JSON formatu, pa neće raditi za spomenutu definiciju. Hive nam omogućuje da sami definiramo ili dodefiniramo kako će se podaci prikazivati. To je ujedno i najveća prednost Hive-a. Dana shema se provodi samo kad čitamo podatke i možemo koristiti Hive SerDe sučelje za određivanje interpretacije onoga što smo učitali. SerDe je skraćenica od 'Serializer and Deserializer' i označava sučelja koja govore Hive-u kako treba prevesti podatke u nešto što može razumijeti. Konkretno, sučelje Deserializer se koristi kada čitamo podatke s diska i pretvara ih u objekte s kojima Hive zna raditi. Možemo napisati prilagođeni SerDe koji će čitati JSON podatke i prevodoti iz u objekte za Hive. Nakon toga smo spremni za korištenje upita nad podacima.

SerDe će uzeti tweet u JSON formatu, na način opisan u dodatku 'Tweet u JSON formatu', i prevesti ga u format pogodan za upite:

```
1 SELECT created_at, entities, text, user
2 FROM tweets
3 WHERE user.screen_name='ParvezJugon'
4   AND retweeted_status.user.screen_name='ScottOstby';
```

Listing 5.1: Upit

Direktorij *hive-serdes* koji se nalazi unutar Hive-a sadrži SerDe koji omogućuje da izvršavamo upite nad 'sirovim' JSON podacima. Metastore Hive-a je konfiguriran za upotrebu MySQL upita. Nakon pokretanja Hive-a, kreiramo potrebnu tablicu za spremanje željenih podataka.

Nakon opisanih koraka, dobili smo end-to-end sustav koji prikuplja podatke s Twitter Streaming API-a, šalje ih datotekama na HDFS-u kroz Hive i koristi Oozie za periodično učitavanje datoteka u Hive gdje, pomoću SerDe, možemo izvršavati upite nad podacima.

5.4 Rezultati

Nakon što je Flume skupljao podatke s Twittera tijekom tri dana na osnovnu sljedećih ključnih riječi:

[3] podaci skinuti s demo.gethue.com dana 27.7.2014.

created_at	user_followers_count	user_screen_name	user_location	retweet_count	user_name	user_friends_count
2014-02-25T16:05:33Z	103	ahyu_savitri	Indonesia	0	Savitri	347
2014-02-25T16:08:13Z	137	eghavia	Indonesia	0	mhegaa'via	139
2014-02-25T16:08:24Z	176	zipopucuviz	London	0	Cristen Carlozzi	221
2014-02-25T16:09:09Z	125	SecuteBiebsxo	New York	0	WISH ME LUCK_éÇ	687
2014-02-25T16:09:29Z	126	Darkgame_Cp	argentina	0	Darkgame Cp	85
2014-02-25T16:09:56Z	109	shaialej29	philippines	0	Shaira Alejado	714
2014-02-25T16:10:06Z	116	mhamidah22	Indonesia	0	Hamidah Muthmainah	160
2014-02-25T16:13:51Z	174	BABY_Bianx	Philippines	0	B I A N C A _Ñ_ _É_ _Ç	233
2014-02-25T16:15:43Z	108	Di_re_cti_0ner	Malaysia	0	free follow	249
2014-02-25T16:17:11Z	150	profebarral441	Venezuela	0	malita	865
2014-02-25T16:17:43Z	158	_ASWE	EspaÕ±a	0	ASWE	110
2014-02-25T16:20:16Z	108	ErdasErdal	stanbul	0	Erdal Erdaª	100
2014-02-25T16:21:48Z	110	ryosukeSoccer4	_±__Î	0	£ ó_@ µ óó ó_	160
2014-02-25T16:23:34Z	134	Yhoenanda	INDONESIA	0	Yunanda Dwi Pramono	227
2014-02-25T16:27:36Z	173	jamieshannon19	UK	0	Jamie Shannon	921

Slika 5.4: dio tablice u kojoj su pohranjenji tweet-ovi[3]

hadoop, big data, analytics, bigdata, cloudera, data science, data scientist, business intelligence, mapreduce, data warehouse, data warehousing, mahout, hbase, nosql, newsql, businessintelligence, cloudcomputing

količina podataka je bila oko pola gigabajta JSON podataka, koji se već opisani, a dateljan kod se nalazi u dodatku 'Tweet u Json formatu'. Prikupljeni podaci imaju određenu strukturu, ali pojedina polja nedostaju. Polje retweeted_status je popunjeno jedino ukoliko je dani tweet bio retweet-an, odnosno ponovno objavljen. Ovakva polovična struktura podataka je dosta složena za obradu pomoću RDBMS, dok Hive može obraditi ovakve podatke jednostavnije.

Sljedeći upit će kao rezultat dati korisničko ime i broj retweet-ova za 10 korisnika s najviše retweet-ova:

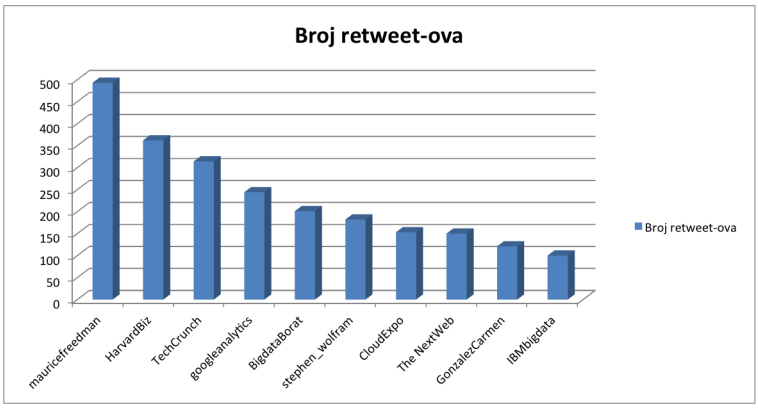
```
SELECT
  t.retweeted_screen_name,
  sum(retweets) AS total_retweets,
  count(*) AS tweet_count
FROM (SELECT
        retweeted_status.user.screen_name as retweeted_screen_name,
      retweeted_status.text,
      max(retweet_count) as retweets
      FROM tweets
      GROUP BY retweeted_status.user.screen_name,
               retweeted_status.text) t
GROUP BY t.retweeted_screen_name
ORDER BY total_retweets DESC
LIMIT 10;
```

Rezultat je sljedeći:

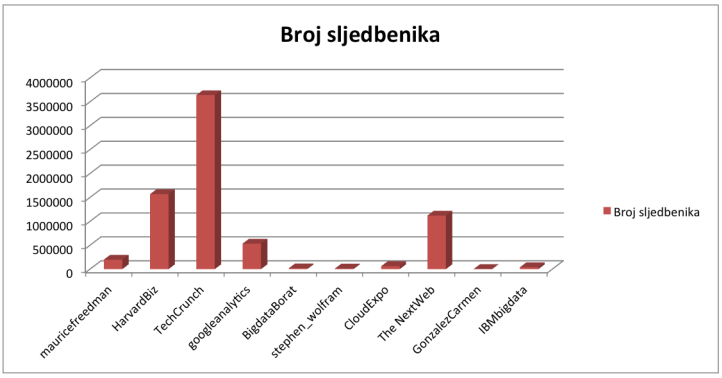
Tablica 5.1: Rezultati

korisnik	broj retweetova	broj sljedbenika
mauricefreedman	493	199000
HarvardBiz	362	1570000
TechCrunch	314	3640000
googleanalytics	244	533000
BigDataBorat	201	18300
stephen_wolfram	182	14000
CloudExpo	153	62200
TheNextWeb	150	1120000
GonzalezCarmen	121	3021
IBMbigdata	100	39100

Iz dobivenih rezultata, osim korisnika koji imaju najviše retweet-ova, možemo vidjeti da broj retweet-ova nije povezan s brojem sljedbenika. Odnosno, korisnik s najviše retweet-ova ima relativno malo sljedbenika. Iz toga zaključujemo da su neki korisnici poprilično popularni na Twitteru, ali njihovi tweet-ovi nisu toliko rašireni.

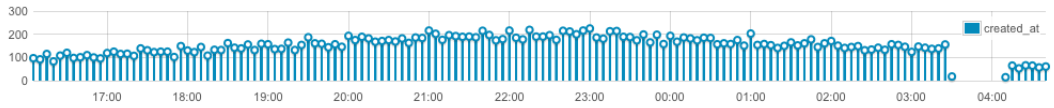


Slika 5.5: Korisnici s najviše retweetova



Slika 5.6: Broj sljedbenika korisnika s najviše retweetova

Zahvaljujući popularnosti Twitter-a, podaci prikupljeni s te društvene mreže i analiza istih mogu biti jako korisni za marketing, promociju, ali i dobivanje korisnih informacija. Pretpostavimo da želimo znati kad je najviše korisnika aktivno. Do tog rezultata dolazimo jednostavnim brojanjem tweet-ova obzirom na vrijeme obrade. Uzmimo manju količinu podataka prikupljenih s Twittera, bez filtriranja ključnih riječi. Rezultat je sljedeći:



Slika 5.7: Broj objavljenih tweet-ova prema vremenu objave

Do ovih rezultata je moguće doći i upotrebom drugih aplikacija i algoritama, ali ono po čemu se Hadoop i Mapreduce razlikuju od ostalih je nskalabilnost ostvarena podjelom poslova koja omogućuje dobivanje rezultata u relativno kratkom vremenu.

5.5 Zaključak

Pokazali smo da Hadoop, odnosno MapReduce, može biti vrlo dobar alat za obradu velikih količina podataka. Na primjeru Twittera vidjeli smo kako se tok podataka može prikupljati i spremati za daljnju obradu. Uz razumijevanje načina rada Hadoop-a i njegovih komponenti, možemo na dosta jednostavan način, programiranjem na višem nivou softvera, doći do traženih informacija iz distribuiranog skladišta podataka. Zbog distribuirane arhitekture softvera i izmjerene skalabilnosti, možemo pretpostaviti da će implementirano rješenje biti funkcionalno i u slučaju bitnog povećanja volumena podataka.

Bibliografija

[1] *Apache Hadoop*. http://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_distributed_file_system.

[2] *Apache Hive*. http://en.wikipedia.org/wiki/Apache_Hive.

[3] *Blogclub tworkshops, Lesson 5: Hadoop and mapreduce*. http://www.alex-hanna.com/tworkshops/lesson-5-hadoop-and-mapreduce/.

[4] *Nutch*. http://en.wikipedia.org/wiki/Nutch.

[5] *The Streaming APIs*, srpanj 2014. https://dev.twitter.com/docs/streaming-apis.

[6] Center, Intel IT: *Apache HDFS*. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/hadoop-spotlight-apache-hdfs-paper.pdf.

[7] Haines, Steven: *Building a MapReduce Application with Hadoop*, Siječanj 2013. http://www.informit.com/articles/article.aspx?p=2017060.

[8] Miner, Donald i Adam Shook: *MapReduce Design Patterns*. O'Reilly Media, 2012.

[9] Natkins, Jon: *Analyzing Twitter Data with Apache Hadoop, Part2*, 2012. blog.cloudera.com/blog/2012/10/analyzing-twitter-data-with-haadoop-part-2-gathering-data-with-flume/.

[10] Natkins, Jon: *How to: Analyze Twitter Data with Apache Hadoop*, Studeni 2012. http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/.

[11] Phillips, John: *Why your kids want to be data scientists*. http://www.cnbc.com/id/101618128.

[12] Ranasinghe, Dhara: *Big data, the cloud, they all mean bigger IT budgets*. http://www.cnbc.com/id/101678413.

[13] Šitum, Mirjam: *Analiza društvenih mreža*, svibanj 2013. http://www.fer.unizg.hr/_download/repository/Seminar_-_Mirjam_Situm.pdf.

[14] White, Tom: *Hadoop: The definitive Guide, Second Edition*. O'Reilly Media, 2009.

Dodatak A

Word Count

Prilikom izvršavanja koda programa 'Word Count', gotovo sav posao se odvija u funkcijama Map() i Reduce().

Klasa *MapClass* prima sljedeće podatke: ključ - pomak bajtova (byte offset), vrijednost - liniju teksta i output - mehanizam preko kojeg kreiramo izlazne podatke u obliku para ključa i vrijednosti. Primljenu vrijednost u obliku stringa MapClass pretvara u mala slova tako da riječi 'Algoritam' i 'algoritam' budu jednake, briše nepotrebne praznine, a sve posebne znakove pretvara u praznine jer ih ne želimo brojati. Svakoj riječi se dodijeli broj 1 koji označavanje pojavljivanje te riječi.

Klasa *ReduceClass* prima slične parametre kao i MapClass, s razlikom da je ključ riječ danog teksta, te klasa ne prima samo jednu vrijednost već listu vrijednosti (točnije, iterator na listu vrijednosti). U ovom primjeru, ulazni parametri će biti u obliku riječi 'Algoritam' i iterator na listu vrijednosti 1, 1, 1, 1. Reduce funkciju možemo koristiti i Combiner funkciju, pa ćemo, umjesto brojanja broja 'ulaza' pojedine riječi, već ćemo dane brojeve 1 dodavati zbroju koji je zapravo rezultat.

```
15
16 public class WordCount extends Configured implements Tool {
17
18     public static class MapClass extends MapReduceBase
19             implements Mapper<LongWritable, Text, Text, IntWritable>
20     {
21         private Text word = new Text();
22         private final static IntWritable one = new IntWritable( 1 );
23
24         public void map( LongWritable key, // Offset into the file
25                          Text value,
26                          OutputCollector<Text, IntWritable> output,
27                          Reporter reporter) throws IOException
28         {
29             // Get the value as a String
```

```
30              String text = value.toString().toLowerCase();
31
32              // Replace all non-characters
33              text = text.replaceAll( "'", "" );
34              text = text.replaceAll( "[^a-zA-Z]", " " );
35
36              // Iterate over all of the words in the string
37              StringTokenizer st = new StringTokenizer( text );
38              while( st.hasMoreTokens() )
39              {
40                  // Get the next token and set it as the text for
41                  //our "word" variable
42                  word.set( st.nextToken() );
43
44                  // Output this word as the key and 1 as the value
45                  output.collect( word, one );
46              }
47          }
48      }
49
50      public static class Reduce extends MapReduceBase
51              implements Reducer<Text, IntWritable, Text, IntWritable>
52      {
53          public void reduce( Text key, Iterator<IntWritable> values,
54                              OutputCollector<Text, IntWritable> output,
55                              Reporter reporter) throws IOException
56          {
57              // Iterate over all of the values (counts of occurrences
58              //of this word)
59              int count = 0;
60              while( values.hasNext() )
61              {
62                  // Add the value to our count
63                  count += values.next().get();
64              }
65
66              // Output the word with its count (wrapped in an IntWritable
    )
67              output.collect( key, new IntWritable( count ) );
68          }
69      }
70
71
72      public int run(String[] args) throws Exception
73      {
74          // Create a configuration
75          Configuration conf = getConf();
```

```
76
77          // Create a job from the default configuration that will
78          //use the WordCount class
79          JobConf job = new JobConf( conf, WordCount.class );
80
81          // Define our input path as the first command line argument
82          //and our output path as the second
83          Path in = new Path( args[0] );
84          Path out = new Path( args[1] );
85
86          // Create File Input/Output formats for these paths (in the job)
87          FileInputFormat.setInputPaths( job, in );
88          FileOutputFormat.setOutputPath( job, out );
89
90          // Configure the job: name, mapper, reducer, and combiner
91          job.setJobName( "WordCount" );
92          job.setMapperClass( MapClass.class );
93          job.setReducerClass( Reduce.class );
94          job.setCombinerClass( Reduce.class );
95
96          // Configure the output
97          job.setOutputFormat( TextOutputFormat.class );
98          job.setOutputKeyClass( Text.class );
99          job.setOutputValueClass( IntWritable.class );
100
101         // Run the job
102         JobClient.runJob(job);
103         return 0;
104     }
105
106     public static void main(String[] args) throws Exception
107     {
108         // Start the WordCount MapReduce application
109         int res = ToolRunner.run( new Configuration(),
110                 new WordCount(),
111                 args );
112         System.exit( res );
113     }
114 }
```

Dodatak B

Tweet u JSON formatu


```
{
   "retweeted_status": {
      "contributors": null,
      "text": "#Crowdsourcing { drivers already generate traffic
               data for your smartphone to suggest alternative
               routes when a road is clogged. #bigdata",
      "geo": null,
      "retweeted": false,
      "in_reply_to_screen_name": null,
      "truncated": false,
      "entities": {
         "urls": [],
         "hashtags": [
            {
               "text": "Crowdsourcing",
               "indices": [
                  0,
                  14
               ]
            },
            {
               "text": "bigdata",
               "indices": [
                  129,
                  137
               ]
```

```
      }
    ],
    "user_mentions": []
},
"in_reply_to_status_id_str": null,
"id": 245255511388336128,
"in_reply_to_user_id_str": null,
"source": "<a href=\"http://www.socialoomph.com\"
     rel=\"nofollow\">SocialOomph<\/a>",
"favorited": false,
"in_reply_to_status_id": null,
"in_reply_to_user_id": null,
"retweet_count": 0,
"created_at": "Mon Sep 10 20:20:45 +0000 2012",
"id_str": "245255511388336128",
"place": null,
"user": {
    "location": "Oregon, ",
    "default_profile": false,
    "statuses_count": 5289,
    "profile_background_tile": false,
    "lang": "en",
    "profile_link_color": "627E91",
    "id": 347471575,
    "following": null,
    "protected": false,
    "favourites_count": 17,
    "profile_text_color": "D4B020",
    "verified": false,
    "description": "Dad, Innovator, Sales Professional.
    Project Management Professional (PMP).  Soccer Coach,
    Little League Coach  #Agile #PMOT - views are my own -",
    "contributors_enabled": false,
    "name": "Scott Ostby",
    "profile_sidebar_border_color": "404040",
    "profile_background_color": "0F0F0F",
    "created_at": "Tue Aug 02 21:10:39 +0000 2011",
    "default_profile_image": false,
    "followers_count": 19005,
```

```
        "profile_image_url_https": "https://si0.twimg.com/
               profile_images/1928022765/scott_normal.jpg",
        "geo_enabled": true,
        "profile_background_image_url": "http://a0.twimg.com/
         profile_background_images/327807929/xce5b8c5dfff3d
         c3bbfbdef5ca2a62b4.jpg",
        "profile_background_image_url_https": "https://si0.twimg
        .com/profile_background_images/327807929/xce5b8c5dfff3dc
        3bbfbdef5ca2a62b4.jpg",
        "follow_request_sent": null,
        "url": "http://facebook.com/ostby",
        "utc_offset": -28800,
        "time_zone": "Pacific Time (US & Canada)",
        "notifications": null,
        "friends_count": 13172,
        "profile_use_background_image": true,
        "profile_sidebar_fill_color": "1C1C1C",
        "screen_name": "ScottOstby",
        "id_str": "347471575",
        "profile_image_url": "http://a0.twimg.com/profile_images/
             1928022765/scott_normal.jpg",
        "show_all_inline_media": true,
        "is_translator": false,
        "listed_count": 45
     },
     "coordinates": null
},
"contributors": null,
"text": "RT @ScottOstby: #Crowdsourcing { drivers already generate
    traffic data for your smartphone to suggest alternative routes
    when a road is  ...",
"geo": null,
"retweeted": false,
"in_reply_to_screen_name": null,
"truncated": false,
"entities": {
   "urls": [],
   "hashtags": [
      {
```

```
            "text": "Crowdsourcing",
            "indices": [
               16,
               30
            ]
         }
      ],
      "user_mentions": [
         {
            "id": 347471575,
            "name": "Scott Ostby",
            "indices": [
               3,
               14
            ],
            "screen_name": "ScottOstby",
            "id_str": "347471575"
         }
      ]
   },
   "in_reply_to_status_id_str": null,
   "id": 245270269525123072,
   "in_reply_to_user_id_str": null,
   "source": "web",
   "favorited": false,
   "in_reply_to_status_id": null,
   "in_reply_to_user_id": null,
   "retweet_count": 0,
   "created_at": "Mon Sep 10 21:19:23 +0000 2012",
   "id_str": "245270269525123072",
   "place": null,
   "user": {
      "location": "",
      "default_profile": true,
      "statuses_count": 1294,
      "profile_background_tile": false,
      "lang": "en",
      "profile_link_color": "0084B4",
      "id": 21804678,
```

```
      "following": null,
      "protected": false,
      "favourites_count": 11,
      "profile_text_color": "333333",
      "verified": false,
      "description": "",
      "contributors_enabled": false,
      "name": "Parvez Jugon",
      "profile_sidebar_border_color": "C0DEED",
      "profile_background_color": "C0DEED",
      "created_at": "Tue Feb 24 22:10:43 +0000 2009",
      "default_profile_image": false,
      "followers_count": 70,
      "profile_image_url_https": "https://si0.twimg.com/
      profile_images/2280737846/ni91dkogtgwp1or5rwp4_normal.gif",
      "geo_enabled": false,
      "profile_background_image_url": "http://a0.twimg.com/images/
          themes/theme1/bg.png",
      "profile_background_image_url_https": "https://si0.twimg.com/
         images/themes/theme1/bg.png",
      "follow_request_sent": null,
      "url": null,
      "utc_offset": null,
      "time_zone": null,
      "notifications": null,
      "friends_count": 299,
      "profile_use_background_image": true,
      "profile_sidebar_fill_color": "DDEEF6",
      "screen_name": "ParvezJugon",
      "id_str": "21804678",
      "profile_image_url": "http://a0.twimg.com/profile_images/
      2280737846/ni91dkogtgwp1or5rwp4_normal.gif",
      "show_all_inline_media": false,
      "is_translator": false,
      "listed_count": 7
   },
   "coordinates": null
}
```

Sažetak

U ovom radu je opisan način rada i primjena Apache Hadoop-a i njegovih komponenti. Najvažnija komponenta je MapReduce koja ima sve veću primjenu. Da bismo mogli koristiti MapReduce algoritam potrebno je razumjeti njegov način rada, te naučiti neka pravila za pisanje samog algoritma kao što je korištenje combiner funkcije. Da bi u potpunosti razumijeli koncept Hadoop-a, objašnjeni su pojmovi Flume, Hive, HDFS, te Oozie. Primjena Hadoop-a i MapReduce-a je pokazana na analizi društeve mreže Twitter. Podaci su prikupljani prema određenim uvjetima pomoću Apache Flume-a, zatim su obrađeni s Oozie-em, a upiti nad njima su izvršeni pomoću Hive-a.

Summary

This thesis describes the operation and use of Apache Hadoop and its components. The most important component is MapReduce. To use MapReduce algorithm it is necessary to understand its mode of operation, and learn some of the rules for writing the algorithm as well as the use of the combiner function. In order to fully understand the concept of Hadoop, the following concepts are explained: Flume, Hive, HDFS and Oozie. Use of Hadoop and MapReduce is shown in the analysis of social network Twitter. Data were collected according to certain conditions using Apache Flume, then they were processed with Oozie-operation and queried using Hive.

Životopis

Rođena sam 23. listopada 1987. godine u Mostaru. 2002. godine sam završila osnovnu školu, te upisala opću gimnaziju fra Slavka Barbarića u Čitluku. Tijekom sljedeće četiri godine sam sudjelovala na županijskim natjecanjima iz matematike. 2006. godine upisujem Prirodoslovno-matematiči fakultet u Zagrebu, Matematički odsjek Sveučilišta u Zagrebu. Nakon završenog preddiplomskog studija matematike upisujem Diplomski sveučilišni studij Računarstva i matematike na istom odsjeku.