Tehnologije za rad s velikim podacima

Grozdek, Andrea

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Andrea Grozdek

TEHNOLOGIJE ZA RAD S VELIKIM PODACIMA

Diplomski rad

Voditelj rada:
prof. dr. sc Robert Manger

Zagreb, veljača, 2017.

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 **1**

1 Veliki podaci **2**
1.1 Definicija velikih podataka 2
1.2 3V karakteristike 2
1.3 Usporedba rada s podacima nekad i danas 7

2 Izgradnja okoline za rad s velikim podacima **9**
2.1 Slojevi tehnologija za rad s velikim podacima 9
2.2 Virtualizacija 14
2.3 Virtualizacija po slojevima 15
2.4 Računalni oblak 15

3 Hadoop **17**
3.1 Osnovne informacije o Hadoopu 17
3.2 Hadoop distribuirani datotečni sustav 19
3.3 MapReduce 25
3.4 Hadoop Ekosustav 35
3.5 Obrada nestrukturiranih podataka 38

4 Studijski primjer **40**
4.1 Zadatak 40
4.2 Rješenje 41

Bibliografija **51**

Uvod

Veliki podaci (big data) postali su vrlo popularna tema u području brojnih znanosti. Razvojem tehnologije stvorili su se uvjeti u kojima se granica memorijskih ograničenja sustava polako gubi, vremenski intervali između generiranja podataka mogu biti gotovo pa proizvoljno mali, a podaci ne moraju poštovati neku strogu strukturu.

Sustavi za rad s velikim podacima mogu pohranjivati podatke koji pristižu u masivnoj količini, veoma brzo i u različitim oblicima. Ništa od navedenog više ne predstavlja problem. Ako postoji potreba za pohranom veće količine podataka, sustav se može lagano proširiti. To svojstvo naziva se **skalabilnost** i zajedničko je svim sustavima za rad s velikim podacima.

Pod pojmom skalabilnost, možemo promatrati vertikalnu i horizontalnu skalabilnost. Vertikalna skalabilnost označava nadogradnju sustava u smislu dodavanja ili zamjene hardverskih komponenti (jačeg procesora, više memorije i slično), dok horizontalna skalabilnost označava dodavanje jedne ili više osnovnih jedinica u sustav. Pod pojmom skalabilnost, kada je riječ o velikim podacima, uglavnom se misli na horizontalnu skalabilnost. Osnovna (hardverska) jedinica u građi takvog sustava je računalo generalne namjene (engl. *commodity hardware*). Dakle, nema potrebe za investicijom u veliki broj skupih računala. Takva računala grupiraju se u ormare (engl. *rack*), a ormari u klastere ili računalne grozdove (engl. *cluster*). Klaster se ponaša kao jedna cjelina - jedno veliko računalo.

Sofverski sustav za rad s velikim podacima prilagođen je takvoj fizičkoj arhitekturi, kako bi zajedno pružili optimalne performanse. Softverski sustav Hadoop, kojem je posvećeno treće poglavlje, predstavlja takav sustav, koji u svojoj osnovi koristi MapReduce algoritam za paralelno izvršavanje zadataka na čvorovima u klasteru. Očita prednost paralelnog nad sekvencijalnim izvršavanjem je ušteda vremena. Zadaci koji se mogu paralelizirati, a koji, kada se izvode sekvencijalno, troše puno vremena, sada se izvršavaju puno jednostavnije i daleko brže. To je otvorilo mogućnost za optimizaciju mnogih procesa, što je do sada bilo uvjetovano nedostupnom količinom resursa.

O velikim podacima često se priča s velikim očekivanjima. Očekuje se da će se primjenom odgovarajućih metoda i tehnika otkriti nove informacije koje mogu predstavljati veliku vrijednost u donošenju odluka i strategija u poslovanju, a u nekim slučajevima, mogu biti i revolucionarne. Primjerice, otkriće uzroka pojave neke bolesti.

Poglavlje 1

Veliki podaci

1.1 Definicija velikih podataka

Pojam veliki podaci (big data) odnosi se na podatake i tehnologije za njihovu obradu. Svojstva koja posjeduju veliki podaci objedinjena su pod nazivom *3V* (engl. *Volume, Velocity, Variety*).

Potreba za obradom sve veće količine podataka, koji su prelazili memorijske mogućnosti računala, dovela je do razvoja novih tehnologija za njihovu obradu. Tehnologije za obradu velikih podataka, poput Hadoopa, omogućuju obradu znatno većih količina podataka, koji više ne moraju biti strogo strukturirani.

1.2 3V karakteristike

Volumen

Velika količina podataka je najčešće prva asocijacija koju vežemo uz pojam big data. No, što znači "velika"? Da bismo neki skup podataka prozvali *velikim podacima*, nije dovoljno da oni budu određene veličine, već je potrebno da posjeduju i ostala svojstva opisana u ovom poglavlju. Dakle, donja granica za volumen nije strogo definirana. Zbog prirode nastajanja velikih podataka, njihova količina obično se kreće od nekoliko gigabajta pa do zetabajta.

Procijenjuje se da Facebook trenutno skladišti oko 300 PB podataka. Broj novih komentara i lajkova na Facebooku dnevno iznosi oko 3 milijarde. Broj pregleda videa na YouTubeu dnevno premašuje milijardu. Twitterovi korisnici dnevno objave preko 500 milijuna tvitova. Sve te podatke potrebno je negdje pohraniti te omogućiti da njihova obrada bude brza i jednostavna.

Instagram	754	novih fotografija
Skype	2367	poziva
Twitter	7428	poslanih tvitova
Google	57511	pretraženih upita
YouTube	136677	pregleda videa
Email	2544176	poslanih emailova

Slika 1.1: Podaci o novim akcijama na internetu u razdoblju od jedne sekunde

Za razliku od statističkih metoda, pomoću kojih bi odgovor na pitanje tražili na uzorku cijele populacije, big data tehnologije omogućavaju nam da obrađujemo cijelu populaciju. Taj pomak u količini obrađenih podataka, otvorio je vrata novim perspektivama iz kojih možemo promatrati veze među podacima.

Brzina

Razvoj tehnologije donosi nove izvore podataka. Procjenjuje se da će broj povezanih *internet stvari* u svijetu ove godine doseći 6.4 milijarde, a njihov broj dnevno raste za 5.5 milijuna. Novi izvori donose i novu brzinu kojom se podaci prikupljaju. Senzori koji se koriste u prometu prikupljaju podatke u realnom vremenu. Web stranice prikupljaju podatke o svakom kliku svojih posjetitelja. Takvi podaci mogu se koristiti u analizi korisnikovog ponašanja kako bi se unaprijedila prodaja, odnosno bilokakav ciljni događaj.

Tehnologije za rad s velikim podacima moraju podržavati pohranu podataka koji pristižu velikom brzinom. Akcije poput objavljivanje posta ili komentara na socijalnim mrežama, trebaju biti vidljive velikom broju korisnika u realnom vremenu.

Aktivnosti na društvenim mrežama, poput Facebooka, Twittera ili Instagrama, idealan su primjer za promatranje izazova u radu s velikim podacima. Da bismo bolje shvatili zašto je brzina jedna od glavnih karakteristika velikih podataka, pogledajmo sliku 1.1 u kojoj su navedeni okvirni podaci[1] za vremensko razdoblje od jedne sekunde.

[1] preuzeto s http://www.internetlivestats.com/one-second/

Raznolikost

Veliki podaci mogu biti strukturirani, polu-strukturirani i nestrukturirani te mogu nastati iz direktne komunikacije čovjeka sa strojem ili stroja sa strojem.
Podaci u relacijskim bazama podataka su strogo strukturirani. Smješteni su u tablice sa unaprijed definiranim stupcima. Poznata je maksimalna duljina i tip podatka u svakom stupcu. Svaki redak tablice predstavlja jedan zapis. Takvi podaci su primjerice podaci o transakcijama, podaci koje prikupljaju senzori, web logovi i slično.

Primjeri strukturiranih podataka generiranih od strane stroja:

- *Podaci dobiveni od senzora*: RFID tagovi, medicinski uređaji, GPS podaci
- *Web log datoteke*: prikupljene od strane servera, aplikacija i mreža
- *Podaci o kupnji*: prikupljeni na blagajni skeniranjem bar koda proizvoda
- *Financijski podaci*: podaci o dnevnom kretanju vrijednosti dionica

Primjeri strukturiranih podataka generiranih od strane čovjeka:

- *Korisnikov unos*: obuhvaća sav unos koji korisnik unese u sustav, poput imena, dobi, odgovora u anketama
- *Podaci klika mišem*: bilježe se podaci o svakom kliku mišem na neku poveznicu na web stranici
- *Podaci igara*: svaki potez igrača tokom igranja igre se bilježi

Kada skup podataka djelomično poštuje neku strukturu, ali ne u potpunosti, kažemo da je to skup **polu-strukturiranih podataka**. Polu-strukturirani podaci mogu sadržavati parove oznaka i vrijednosti, poput XML datoteka.

Nestrukturirani podaci su primjerice blog postovi, novinski članci, slike. Takvi podaci ne poštuju određeni format pa ih je do razvoja tehnologija za obradu velikih podataka bilo moguće samo prikupljati te naknadno ručno analizirati. Nestrukturirani podaci čine većinu od ukupne količine podataka. Procjenjuje se da je taj udio oko 80%. Nestrukturirane podatke također možemo podijeliti na podatke generirane od strane čovjeka te od strane računala.

Primjeri nestrukturiranih podataka generiranih od strane strojeva (*machine data*):

- *Satelitske snimke*: podaci o vremenu (meteorološke snimke)

- *Znanstveni podaci*: atmosferski podaci, seizmički podaci
- *Fotografije i video snimke*: podaci prikupljeni nadzornim kamerama, podaci o stanju u prometu
- *Podaci prikupljeni sonarom ili radarom*: uključuju automobilske, meteorološke i oceanografske seizmičke profile

Primjeri nestrukturiranih podataka generiranih od strane čovjeka:

- *Tekstualni podaci unutar neke kompanije*: uključuje dokumente, elektroničku poštu, rezultate anketa i istraživanja, log datoteke
- *Podaci društenih mreža*: podaci generirani na društvenim mrežama poput Facebooka, Twittera, LinkedIna, Flickera
- *Podaci mobilnih telefona*: tekstualne poruke, informacije o lokaciji
- *Podaci web stranica*: odnosi se na sve stranice koje nude nestrukturirane podatke, poput YouTubea, Instagrama, Flickera

Navedena svojstva čine velike podatke neprikladnim za obradu i pohranu u standardnim relacijskim bazama podataka te su se stoga pojavila potreba za razvojem novih tehnologija i alata za njihovu obradu.

Mnogi autori vole nadodati još nekoliko V-ova pri definiranju velikih podataka. Neki od uobičajenih dodataka su vjerodostojnost (engl. *Veracity*), vrijednost (engl. *Value*), promjenjivost (engl. *Variability*), nestalnost (engl. *Volatility*), vizualizacija (engl. *Visualisation*). To su također svojstva velikih podataka, no ona ne čine razliku između velikih podataka i ostalih podataka. U nastavku će biti pobliže objašnjeni.

Vjerodostojnost (engl. Veracity)

Analize podataka rade se kako bi ponudile odgovor na neko pitanje. Procjena vjerodostojnosti tog odgovora ovisi o samim podacima. Prije početka analize, podaci se obično čiste od neispravnih zapisa, jer oni mogu nepovoljno utjecati na rezultate analize. Detekcija neispravnih zapisa, odnosno, općenito, zapisa za koje ne želimo da ulaze u analizu, predstavlja jedan od izazova u razvoju tehnologija za rad s velikim podacima.

Kao primjer podataka upitne vjerodostojnosti, možemo promatrati nestrukturirane podatke s društvenih mreža. Korištenjem *hashtaga* moguće je na proizvoljan način označiti različite sadržaje poput fotografija i tekstualnih objava. Primarna uloga *hashtaga* je grupirati srodne sadržaje te time olakšati pretraživanje po nekom pojmu. No, zbog slobode

korisnika da označavaju sadržaje kako god žele, dio velikih podataka koji ulazi u sustav činit će podaci koje bismo u većini slučajeva mogli smatrati ”smećem”. Prepoznavanje i tretiranje takvih zapisa u analizi jedan je od zadataka sustava za rad s velikim podacima.

Vrijednost (engl. Value)

Vrijednost podataka mjeri se vrijednošću informacija koje se iz njih mogu izvući. Prije početka prikupljanja podataka, teško je predvidjeti sve scenarije u kojima bi se ti podaci mogli koristiti. Ponekad do zanimljivih spoznaja dolazimo naknadno, u kombiniranju podataka iz različitih izvora. Osim vrijednosti koje podaci nude u poslovnom smislu, važno je spomenuti primjere vezane uz znanost.
U kombinaciji sa prediktivnom analitikom, veliki podaci mogu biti temelj za stvaranje prediktivnih modela, koje je moguće primjeniti u ranoj dijagnostici. Na primjer, moguće je izgraditi model koji će predvidjeti pojavu infarkta (detektirati visoki rizik pojave infarkta) dovoljno rano da se poduzmu potrebne akcije, kojima se na kraju može spasiti ljudski život. Za izradu takvog modela, potreban je veliki skup mjerenja prikupljenih kroz određeni vremenski period od skupine ljudi koja je doživjela infarkt i skupine ljudi koja je zdrava. Ako je skup mjerenja premalen, odnosno zastupljenost neke skupine nije dovoljno velika, postoji mogućnost da model neće davati točne rezultate.

Promjenjivost (engl. Variability)

U analizi teksta, problem predstavljaju riječi koje nemaju uvijek jednako značenje, već ono ovisi o kontekstu. Tako, na primjer, riječ *super*, ne možemo uvijek promatrati kao pozitivnu, jer se ona može naći u sarkastičnom komentaru poput *Kasniš pola sata. Baš super.*.

Vizualizacija (engl. Visualisation)

Jasan i razumljiv prikaz rezultata analize bitan je za razumijevanje važnosti informacija dobivenih analizom. Analize mogu obuhvaćati mnogo parametara i pomoću novih alata možemo ih prikazivati na kreativnije i razumljivije načine. Vizualizacija je posebno važna u radu sa velikim podacima jer se lako možemo pogubiti u interpretaciji rezultata, čime svrha analize postaje upitna. Zamislimo, na primjer, da želimo vizualizirati skup od 10 milijuna zapisa po nekom kriteriju. Kada bi svaki zapis bio predstavljen jednom točkom, graf bi sadržavao 10 milijuna točaka. Takav prikaz nije čitljiv i ne prenosi relevantne informacije na dovoljno jasan način. Elegantniji prikaz moguće je dobiti klasteriranjem podataka, pri čemu manje grupe podataka postaju vidljive.
Primjeri alata za vizualizaciju su Tableau Software, Infogram, Chartblocks, Plotly.

Nestalnost (engl. Volatility)

Kada je riječ o velikim podacima, obično se naglašava mogućnost spremanja ogromnih količina podataka i njihovo korištenje u nove nepredvidive svrhe. No, za neke podatke moguće je odmah procijeniti da će nakon nekog trenutka postati bezvrijedni. Pohranjivanje takvih podataka bilo bi besmisleno zauzimanje resursa. Srećom, postoje tehnologije za rad s velikim podacima, pomoću kojih možemo podatke obrađivati dok pristižu u sustav i pohranjivati samo dio obrađenih informacija koje bi mogle poslužiti.

1.3 Usporedba rada s podacima nekad i danas

Termin "big data", odnosno veliki podaci, postoji tek nekoliko godina. No, problemi koje bismo mogli smjestiti u područje velikih podataka postoje već duži niz godina. Razmislimo malo o činjenici da su Google, Amazon i Ebay osnovani prije dvadesetak godina. Amazon je u svojim počecima, 1995. godine, nudio oko milijun knjiga. Jesu li podaci o prodajama, korisnicima i proizvodima, koje je Amazon prikupio u prvih par godina, bili veliki podaci? Koja je razlika u analizama koje su se provodile nad podacima tada, u odnosu na mogućnosti koje imamo danas?

Razvoj tehnologije jedan je od glavnih faktora koji nam je omogućio da prikupljamo i analiziramo veće količine podataka. Prosječan hard disk početkom devedesetih mogao je pohraniti 1,370 MB podataka. Pri brzini prijenosa podataka od 4.4 MB/s, vrijeme potrebno za čitanje cijelog diska iznosilo je oko 5 minuta. Danas, za standardnu veličinu diska od jednog terabajta, uz brzinu prijenosa od 100 MB/s, čitanje cijelog diska traje oko dva i pol sata.

Memorijski kapaciteti su se povećali, imamo uvjete za bržu pohranu veće količine podataka. No, želimo li iz tih podataka brže dobiti odgovor na neka pitanja, trebat ćemo istražiti ostale pristupe.

Jedna od učinkovitih mogućnosti nalazi se u konceptu paralelnog rada. Podijelimo li skup podataka veličine 1 TB na sto dijelova jednakih veličina, te svaki dio spremimo na jedan od sto slobodnih diskova, čitanje cijelog skupa podataka trajat će oko dvije minute. Prednost tog pristupa je očito ušteda vremena. Mana je veća vjerojatnost pojave hardverske greške. Pri gašenju dijela hardvera, dolazi do gubitka podataka koji su pohranjeni na njemu. Zato je bitno da sustav osigura kopije podataka. Sljedeći problem je povezivanje podataka. Većina analiza zahtjeva povezivanje rezultata dobivenih iz različitih izvora, što nije trivijalan zadatak. MapReduce model apstrahira problem čitanja i pisanja sa diska, pretvarajući ga u problem izračunavanja nad skupom parova oblika (ključ, vrijednost).

U mnogo slučajeva, MapReduce može se promatrati kao komplement RDBMS-a. Usporedba je dana tablicom 1.1.

	Tradicionalni RDBMS	**MapReduce**
Veličina podataka	Gigabajti	Petabajti
Pristup	Interaktivan i batch	Batch
Promjene	Čitaj/piši više puta	Piši jednom, čitaj više puta
Struktura	Statična shema	Dinamična shema
Integritet	Visok	Nizak
Skaliranje	Nelinearno	Linearno

Tablica 1.1: Usporedba RDBMS i MapReduce

MapReduce je prikladniji izbor za probleme u kojima želimo analizirati cijeli skup podataka, dok je RDBMS bolji za probleme koji zahtjevaju česte promjene (update) nad postojećim podacima. RDBMS je predviđen za rad sa strukturiranim podacima, dok kod MapReducea nije bitno jesu li podaci strukturirani, polustrukturirani ili nestrukturirani, jer ulazni parovi (ključ, vrijednost) u MapReduce ne ovise nužno o samim podacima, već ih programer sam bira. Jedna od osobina MapReduce modela je linearna skalabilnost. Za duplo veći skup ulaznih podataka, moguće je zadržati jednaku brzinu obrade podataka povećanjem veličine klastera. To svojstvo ne vrijedi općenito kod SQL upita.

Prije razvoja tehnologija za rad s velikim podacima, analize su se uglavnom radile na uzorku, a ne na skupu svih prikupljenih podataka. Taj pristup ima brojnih prednosti, poput brzine i točnosti rješenja nekih problema. No, uzorak ima svojih ograničenja. Ukoliko je zastupljenost neke podgrupe u uzorku veoma mala, moguće je da nećemo dobiti relevantne rezultate. Na primjer, tražimo li odgovor na pitanje ”za kojeg političkog kandidata će glasati neudana, fakultetski obrazovana žena, mlađa od 30 godina, azijskog porijekla”, moguće je da se rezultati dobiveni na uzorku neće poklapati s rezultatima dobivenim obradom cijele populacije.

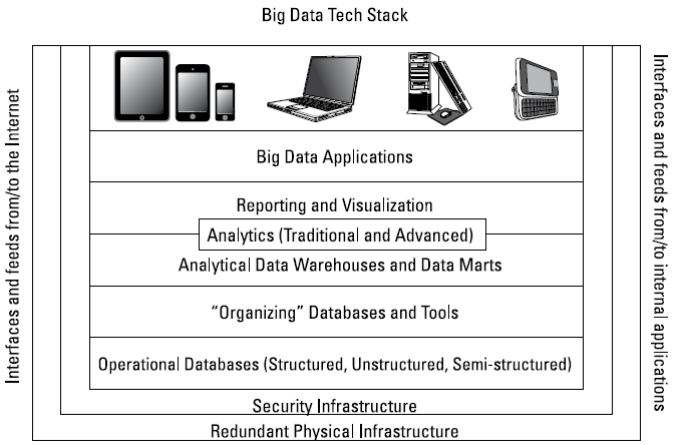
Veliki podaci otvaraju vrata kreativnosti i pronalaženju novih veza među podacima.

Poglavlje 2

Izgradnja okoline za rad s velikim podacima

2.1 Slojevi tehnologija za rad s velikim podacima

Na slici 2.1 nalazi se slojeviti prikaz okoline za rad s velikim podacima. U nastavku će biti više riječi o pojedinim slojevima.



Slika 2.1: Tehnološki stog

Fizički sloj

Na najnižoj razini stoga nalazi se fizički sloj. On obuhvaća hardware, mreže i slično. Pri odabiru fizičke infrastrukture za rad s velikim podacima, potrebno je uzeti u obzir sljedeće

kriterije:

- Performanse
- Dostupnost
- Skalabilnost
- Fleksibilnost
- Trošak

Većina big data rješenja zahtjeva visoku dostupnost, stoga mreže, serveri i fizička memorija trebaju biti otporni na pojavu greške. Pojava greške ne bi smjela uzrokovati prekid rada sustava.

Sigurnosni sloj

Zahtjevi za sigurnošću i zaštitom velikih podataka u velikom se dijelu preklapaju sa zahtjevima za sigurnošću ostalih podataka.

- Pristup podacima: Pristup podacima treba biti dozvoljen samo onim korisnicima koji imaju valjanu poslovnu potrebu za interakcijom s podacima. Svim ostalim korisnicima pristup treba biti onemogućen.
- Pristup podacima iz aplikacije: Većina sučelja za programiranje aplikacija nudi zaštitu od neautoriziranog korištenja. Taj stupanj zaštite prikladan je za većinu big data aplikacija.
- Šifriranje podataka: Šifriranje i dešifriranje podataka i ranije su predstavljali proces koji je zahtjevan za resurse.
- Otkrivanje sigurnosne prijetnje: Uključivanje sve većeg broja internet stvari, posebice pametnih telefona, dovodi do razmjene sve veće količine podataka, a time i do povećanja sigurnosnih prijetnji. Stoga je razvoj sigurnosti u radu s velikim podacima još bitniji nego u radu s ostalim podacima.

Baze podataka

Baze podataka predstavljaju centralni dio svakog sustava za rad s velikim podacima. Baza treba biti brza, skalabilna i pouzdana. Sustav za upravljanje relacijskom bazom podataka

(RDBMS) moguće je primjeniti u rješavanju problema vezanih uz velike podatke, no taj izbor neće uvijek biti idealno rješenje u pogledu performansi, troška i skalabilnosti. NoSQL baze podataka u većini slučajeva nude bolje performanse. Usporedba je dana tablicom 2.2.

Ukoliko sustav treba podržavati rad sa transakcijama, važno je da zadovoljava *ACID svojstva*:

- Atomarnost(*engl. Atomicity*): Transakcija je "sve ili ništa". Ako minimalno jedan dio transakcije nije proveden do kraja, transakcija se smatra nevažećom.
- Konzistentnost (*engl. Consistency*): Transakcija se može provesti samo na ispravnim podacima. Ukoliko su podaci nepotpuni ili koruptirani, transakcija se neće provesti do kraja i promjene neće biti zapisane u bazu.
- Izolacija (*engl. Isolation*): Transakcije koje se odvijaju usporedno, ne bi smjele utjecati jedna na drugu i trebaju se moći prikazati kao da se izvršavaju sekvencijalno.
- Trajnost (*engl. Durability*): Nakon što je transakcija uspješno završila, podaci ostaju trajno pohranjeni u bazi.

Table 4-1 Important Characteristics of SQL and NoSQL Databases

Engine	*Query Language*	*MapReduce*	*Data Types*	*Transactions*	*Examples*
Relational	SQL, Python, C	No	Typed	ACID	PostgreSQL, Oracle, DB/2
Columnar	Ruby	Hadoop	Predefined and typed	Yes, if enabled	HBase
Graph	Walking, Search, Cypher	No	Untyped	ACID	Neo4J
Document	Commands	JavaScript	Typed	No	MongoDB, CouchDB
Key-value	Lucene, Commands	JavaScript	BLOB, semityped	No	Riak, Redis

Slika 2.2: Usporedba SQL i NoSQL baza podataka

NoSQL baze podataka

NoSQL ("*Not only SQL*") baze podataka počele su se intenzivno razvijati kako bi pružile podršku za rad s velikim podacima.

Glavne karakteristike NoSQL baza podataka:

- Nema zahtjeva za striktnom shemom. Shema relacijske baze podataka je opis tipova koji se pohranjuju u bazi i njihove strukture. U NoSQL bazama podataka nije potrebno unaprijed navoditi tipove podataka.
- Mogućnost horizontalnog skaliranja na klasteru računala generalne namjene.
- Ne koristi se relacijski model. U relacijskim bazama podataka, relacije podataka uspostavljaju veze među tablicama. U NoSQL bazama podataka, svi logički povezani podaci spremaju se unutar jednog zapisa.

Postoje četiri vrste NoSQL baza podataka:

- Ključ-vrijednost baze podataka
- Dokument baze podataka
- Stupčane baze podataka
- Graf baze podataka

Ključ-vrijednost baze podataka

U ključ-vrijednost bazama podataka podaci se pohranjuju u asocijativnom nizu, odnosno mapi ili rječniku. Svaki zapis sprema se u bazu pod jedinstvenim ključem. Zapisi mogu sadržavati različite tipove podataka i nije ih potrebno prethodno definirati. Vrijednosti mogu biti bilo kojeg BLOB (Binary Large OBject) tipa: slika, audio zapis, dokument, video zapis, internetska stranica.

Jedna od prednosti je svakako mogućnost promjene tipa zapisa. Moguće je dodati ili izbaciti atribute iz zapisa. Pritom, postoji ograničenje na veličinu zapisa.

Za rad s podacima koriste se naredbe put, get i delete.

Glavna obilježja ključ-vrijednost baza podataka su brzina, skalabilnost i jednostavnost.

Brzina se postiže korištenjem radne memorije. Algoritmima se određuje koji podaci će dobiti prioritet da budu pohranjeni u radnoj memoriji.

Primjeri: Amazon DynamoDB, Riak, Voldemort, Aerospike, Apache Cassandra, Oracle NoSQL Database

Dokument baze podataka

Dokument baze podataka nude pohranu podataka u obliku samo-opisujuće stablaste strukture poput JSON-a, XML-a ili BSON-a(Binary JSON). Svaki zapis u bazi može se promatrati kao par (ključ, vrijednost), gdje je ključ jednistveni identifikator zapisa. Razlika u odnosu na ključ-vrijednost bazu podataka je u tome što se u dokument bazi podataka ključ ne treba koristiti u operacijama nad bazom. Upiti se mogu izvršavati nad vrijednostima, odnosno pohranjenim dokumentima.

Primjeri: MongoDB, CouchDB, Couchbase Server, Azure DocumentDB, MarkLogic

Stupčane baze podataka

U stupčanim bazama podataka, podaci se spremaju po stupcima, a ne po recima, kao u RDBMS. Takav način pohrane podataka omogućuje veću brzinu pristupa podacima te brze izračune nad stupcem podataka. Stupci se grupiraju u familije stupaca. Stupčane baze pružaju mogućnost naknadnog dodavanja stupaca. Od svih NoSQL baza podataka, stupčane baze pružaju najveću skalabilnost. Već smo istaknuli da je skalabilnost veoma važna osobina sustava za rad s velikim podacima.

Primjeri: Oracle RDBMS Columnar Expression, Apache Cassandra, HBase, Google BigTable

O HBase bazi podataka biti će više riječi u trećem poglavlju.

Graf baze podataka

Graf baza podataka koristi strukturu grafa za pohranu podataka. Graf se sastoji od čvorova i bridova. Bridovi opisuju vezu između čvorova. Stoga su podaci u kojima postoji puno veza, dobar kandidat za pohranu u graf bazi podataka. Reprezentacija podataka u obliku grafa, pruža mogućnost primjene algoritama iz teorije grafova. Graf baze podataka podržavaju ACID svojstva za rad s transakcijama.
Primjeri: Neo4J, InfiniteGraph, FlockDB, ArangoDB

Servisi i alati za organizaciju podataka

Ovaj sloj zadužen je za prikupljanje, validaciju i grupiranje podataka u smislene cjeline. Riječ je o cijelom ekosustavu koji se sastoji od alata i tehnologija koji skupljaju i grupiraju podatke kako bi bili spremni za daljnje korištenje. Kao takvi, alati trebaju podržavati integraciju, normalizaciju i skaliranje podataka. MapReduce jedna je od često korištenih tehnika, koja daje dobre rezultate u radu s velikim podacima.

Tehnologije u ovom sloju uključuju slijedeće:

- Distribuirani datotečni sustav: nužan je za pohranu podataka

- Servis za serializaciju: nužan je za trajnu pohranu podataka i višejezičnu podršku poziva udaljenih procedura (RPC)
- Servis za koordinaciju: nužan je za izradu distribuiranih aplikacija
- ETL (*engl. Extract, transform and load*) alati: nužni su za unos strukturiranih i nestrukturiranih podataka u Hadoop te njihovu pravilnu konverziju
- Workflow servisi: nužni su za raspoređivanje poslova i sinhronizaciju između slojeva

Analitička skladišta podataka

Skladišta podataka uglavnom se sastoje od normaliziranih podataka prikupljenih iz različitih izvora, s namjenom da budu korišteni u (poslovnim) analizama. Pojednostavljuju stvaranje izvještaja i vizualizaciju neusporedivih podataka.

2.2 Virtualizacija

Virtualizacija je proces simuliranja više (različitih) okolina unutar jednog fizičkog resursa. Virtualizacija optimizira iskoristivost fizičkih resursa te omogućuje bolji nadzor nad njihovim korištenjem.

Makar virtualizacija tehnički nije nužna u izgradnji okruženja za rad s velikim podacima, zbog brojnih prednosti koje pruža, postala je nezaobilazan dio.
Navodimo nekoliko karakteristika virtualizacije koje omogućuju *skalabilnost* i *radnu efikasnost* u okruženju za rad s velikim podacima:

- **Particioniranje**: Particioniranjem dostupnih resursa fizičkog stroja, možemo omogućiti rad više različitih operacijskih sustava i aplikacija na istom stroju.
- **Izolacija**: Svaki virtualni stroj predstavlja okolinu za sebe, odvojenu od ostalih virtualnih strojeva te od samog fizičkog sloja. Time je omogućeno da kvar jednog stroja ne utječe na rad ostalih strojeva koje dijele zajednički fizički resurs.
- **Enkapsulacija**: Virtualni stroj moguće je pohraniti u obliku datoteke, koju aplikacija vidi kao jednu zatvorenu cjelinu.

Virtualizacija poboljšava efikasnost svakog sloja IT infrastrukture, stoga omogućuje bolje performanse cijelog sustava za rad s velikim podacima. Skalabilnost sustava je nužna

iz više razloga. Sustav treba moći primati nove količine različitih tipova podataka i pohranjivati ih u distribuiranoj okolini. Također, treba moći provoditi analizu nad svim podacima. Za sve to, nužno je da sustav možemo lagano proširiti. MapReduce i Hadoop, o kojima će biti kasnije riječi u radu, također podrazumijevaju skalabilnost.

2.3 Virtualizacija po slojevima

Virtualizacija servera

Virtualizacija servera je particioniranje jednog fizičkog servera na više virtualnih servera.

Virtualni stroj je softverska reprezentacija fizičkog stroja koji može izvršavati iste funkcije kao fizički stroj.

Hardverski dijelovi servera, poput procesora, RAM memorije, hard diska te mrežnih kontrolera, također se mogu logički podijeliti u niz virtualnih strojeva, od kojih svaki može raditi u drugačijem operacijskom sustavu te podržavati različiti skup aplikacija.

Monitor virtualnog stroja (engl. *hypervisor*) je softver koji se nalazi između virtualnih strojeva i fizičkog stroja. On se brine za efikasno korištenje fizičkih resursa. Postoje dva tipa monitora. Monitor koji direktno komunicira sa hardverom te monitor koji komunicira sa operacijskim sustavom servera.

Virtualizacija mreže

Različite faze rada s velikim podacima ne zahtjevaju nužno jednake mrežne performanse. Stvaranje više virtualnih mreža na istoj fizičkoj mreži omogućuje stvaranje optimizirane mreže za specifičnu namjenu. Na primjer, možemo imati virtualnu mrežu za prikupljanje podataka i drugačiju virtualnu mrežu za potrebe neke aplikacije. Virtualizacija smanjuje mogućnost pojave uskog grla i optimizira korištenje mreže.

Virtualizacija procesora i memorije

Virtualizacija procesora i memorije provodi se također kako bi se optimiziralo njihovo korištenje. Algoritmi koji se koriste u analizi velikih podataka mogu zahtjevati dosta procesorske snage i RAM memorije. Čekanje da se resursi osobode, uzrokuje nepoželjno kašnjenje rezultata.

2.4 Računalni oblak

Tehnologije za rad s velikim podacima dostupne su i u oblaku.
Usluge (servisi) u oblaku mogu se grupirati u slijedeće skupine:

- Infrastruktura kao usluga (IaaS)
- Platforma kao usluga (PaaS)
- Softver kao usluga (SaaS)

Infrastruktura kao usluga

Infrastruktura kao usluga (IaaS - *Infrastructure as a Service*) obuhvaća hardver, mreže, operacijski sustav, memoriju. Pružatelj usluge brine se o ispravnom radu navedenih dijelova. Korisniku je dostupna obično u obliku virtualnog stroja u oblaku.

Primjeri: Amazon EC2, Microsoft Azure, Rackspace.

Platforma kao usluga

Platforma kao usluga (PaaS - *Platform as a Service*) pruža okruženje za razvoj i izvršavanje aplikacija. Pružatelj usluge omogućava potrebni hardver, operacijski sustav, bazu podataka, *middleware* i slično. U uslugama namjenjenim za rad s velikim podacima, obično se nalazi i Hadoop, odnosno neka kombinacija elemenata Hadoop ekosustava. Platforma pruža okruženje za razvoj aplikacija s velikim podacima.

Naplata usluga u oblaku uglavnom se temelji na stvarnoj potrošnji. Dakle, ovisi o količini pohranjenih podataka i broju dohvaćanja podataka u analizama (get i put zahtjevi).

Platforma kao usluga za rad s velikim podacima nudi usluge za prikupljanje, pohranu, obradu, analizu i vizualizaciju velikih podataka.

Primjeri platformi kao usluga su AWS (Amazon Web Services), Microsoft Azure, Google Cloud Platform.

Softver kao usluga

Softver kao usluga (SaaS - *Software as a Service*) namjenjen je za krajnjeg korisnika. Pružatelj usluge nudi kompletnu podršku za softver u oblaku. Korisnik pristupa softveru (aplikaciji) putem interneta te nema potrebe za instalacijom na korisnikovo računalo.

Primjeri korištenja SaaS modela: email, igre, CRM (*Customer relationship management*).

Svašta se može nuditi i promatrati kao usluga (*as a service*). Vezano uz velike podatke, noviji pojam BDaaS (*Big Data as a Service*) predstavlja skup alata i tehnologija za rad s velikim podacima koji su dostupni kao usluga.

Poglavlje 3

Hadoop

3.1 Osnovne informacije o Hadoopu

Apache Hadoop je softverski okvir otvorenog koda namijenjen za distribuiranu pohranu i obradu velikih podataka.

Razvoj Hadoopa započeo je Doug Cutting u sklopu Apache Nutch[1] projekta, na kojem je radio od 2003. godine. Objava Googlovih publikacija "The Google File System" (2004. godine) i "MapReduce: Simplified Data Processing on Large Clusters" (2006. godine), utjecala je na razvoj Nutch distribuiranog datotečnog sustava otvorenog koda te implementaciju MapReduce modela unutar Nutcha. Godine 2006. iz Nutcha se izdvaja Hadoop kao samostalni projekt te započinje intenzivniji razvoj. Već dvije godine kasnije, Yahoo je objavio da svoj indeks pretraživanja generira pomoću Hadoopovog klastera koji se sastoji od 10000 jezgri. Iste godine, Hadoop pobjeđuje na natjecanju u sortiranju terabajta podataka, uz vrijeme od 209 sekundi, na klasteru od 910 čvorova.

Hadoop se udomaćio u brojnim kompanijama, koje ga koriste u edukacijske i produkcijske svrhe. Neke od njih su Facebook, LinkedIn, Amazon, Ebay, Spotify. Detaljnija lista s navedenom veličinom klastera i namjenom dostupna je na web stranicama `https://wiki.apache.org/hadoop/PoweredBy`.

Doug Cutting dodijelio je ime Hadoopu prema istoimenoj plišanoj igrački svoga sina. Žuti slon Hadoop ujedno je postao maskota projekta. Riječ Hadoop nema skriveno značenje, lako se pamti i izgovara, a taj princip korišten je u dodjeljivanju imena ostalim većim komponentama u Hadoop ekosustavu (Pig, Spark, Oozie, ZooKeeper, ...). Manje komponente imaju sugestivne nazive (namenode, datanode).

Okosnicu Hadoopa čine Hadoop distribuirani datotečni sustav (HDFS), zadužen za pohranu podataka, te MapReduce, zadužen za obradu. O njima će biti više riječi u nastavku

[1] web tražilica otvorenog koda

ovog rada.

Oko osnovnog softvera, Hadoopa, razvili su se srodni kompatibilni projekti koji zajedno čine Hadoopov ekosustav.

Distribucije Hadoopa

Hadoop se može besplatno preuzeti sa Apacheovih web stranica `http://hadoop.apache.org/`, gdje se nalaze i upute za instalaciju na Linux i Windows operacijskim sustavima.

Hadoop je moguće instalirati na tri načina:

- Local (Standalone) Mode: Instalacija Hadoopa na jednom računalu. Klaster se sastoji od samo jednog čvora. Korisno za učenje i otkrivanje grešaka u kodu.
- Pseudo-Distributed Mode: Simulacija Hadoopovog klastera od nekoliko čvorova na jednom računalu. Također je korisno za učenje.
- Fully-Distributed Mode: Hadoopov klaster se sastoji od većeg broja čvorova. Ovaj način je prikladan za produkcijsku upotrebu.

Postoje brojne distribucije Hadoopa. Neke od njih su:

- Cloudera's Distribution including Apache Hadoop (CDH)
- Hortonworks Data Platform (HDP)
- Amazon Web Services: Amazon Elastic MapReduce (Amazon EMR)
- MapR
- IBM BigInsights
- Microsoft Azure HDInsight: Hadoop in the Azure cloud

Navedene platforme nude široku paletu usluga, od kojih su neke besplatne. Cloudera i Hortonworks nude jednostavnu instalaciju Hadoopa u obliku virtualnog stroja ili Docker containera. Cloudera Quickstart VM je besplatni virtualizirani klaster (single-node) koji je dostupan u formatu virtualnog stroja za VirtualBox, VMware ili KVM okolinu. Postoji i verzija (multi-node cluster) za Docker, u obliku Docker containera. Nije predviđen za produkcijsku uporabu, nego primarno za učenje i testiranje.

3.2 Hadoop distribuirani datotečni sustav

Glavna obilježja Hadoop distribuiranog datotečnog sustava

Hadoop distribuirani datotečni sustav (HDFS[2]) dizajniran je za pohranu velikih datoteka na klasteru računala generalne namjene.

Nastao je prema modelu Googleovog datotečnog sustava[3]. Implementiran je u Javi.

Otpornost na pojavu greške i kvar hardvera

HDFS dizajniran je s pretpostavkom da pojava greške i kvara hardvera nije iznimka, već uobičajeni događaj. Pojava greške ne izaziva prekid rada niti dugo čekanje. Razlog tome je u načinu na koji HDFS sprema podatke, replicirajući ih na više računala u klasteru. Jedna od prednosti HDFS-a je što nema visokih zahtjeva za hardverom. Dovoljno je koristiti računala generalne namjene, odnosno računala nižeg cjenovnog ranga.

Protočni pristup podacima

HDFS je namijenjem za protočni pristup podacima, za *batch* obrade nad cijelim skupom podataka. Vrijeme potrebno za dohvaćanje cijelog skupa podataka je bitnije od vremena potrebnog za dohvaćanje nekog konkretnog zapisa. Interakcija s korisnikom i brzi pristup nekom konkretnom zapisu nisu primarna namjena korištenja HDFS-a.

Rad s velikim podacima

Uobičajena datoteka koje se pohranjuju u HDFS je veličine gigabajta ili više. HDFS je namijenjen za rad s velikim podacima i nije prikladan u slučaju velikog broja malih podataka (nekoliko megabajta ili manje).

Podaci se zapisuju jednom i čitaju više puta

Podaci koji se pohranjuju obično se jednom generiraju ili kopiraju iz nekog izvora te se kasnije više puta čitaju i koriste u raznim analizama.

Izračunavanje se odvija blizu podataka

Najveća efikasnost postiže se kada se izračunavanje nad nekim skupom podataka odvija u neposrednoj blizini tih podataka. To je posebno važno kada je riječ o velikim poda-

[2] engl. *Hadoop Distributed File System*
[3] GFS - Google File System

cima. Premještanje podataka je skupo, stoga je HDFS dizajniran na način da lokaciju izračunavanja izabire ovisno o lokaciji podataka.

Portabilnost

HDFS je dizajniran s namjerom da bude prenosiv te da se može jednostavno preseliti s jedne platforme na drugu.

HDFS nije prikladan izbor u sljedećim situacijama:

- Aplikacija zahtjeva brzi pristup podacima, uz dozvoljeno kašnjenje veličine desetak milisekundi (engl. *low-latency data access*)
- Podaci se sastoje od velikog broja malih datoteka

Pohrana datoteka

Hadoop distribuirani datotečni sustav podržava tradicionalni hijerarhijski sustav organizacije datoteka. Korisnik ili aplikacija mogu stvarati nove direktorije te u njima pohranjivati datoteke.

Poput ostalih datotečnih sustava, HDFS sprema datoteke u memorijskim blokovima.

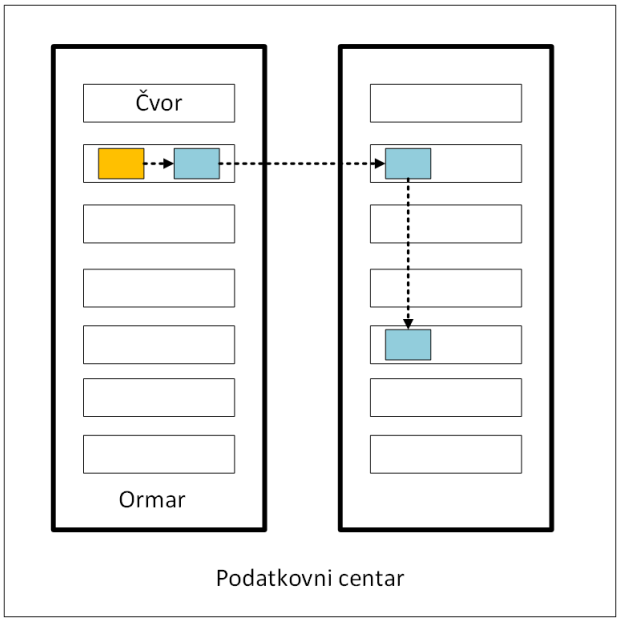
Defaultna veličina bloka iznosi 128 MB, a moguće ju je i povećati.

Veličina bloka u HDFS-u znatno je veća u usporedbi sa standardnim datotečnim sustavom, gdje obično iznosi nekoliko kilobajta.

Razlog tome je minimizacija vremena pretraživanja. Ako sa t označimo vrijeme potrebno za pronalaženje početka bloka, a sa n broj blokova koje zauzima datoteka, tada će $t * n$ biti vrijeme koje datotečni sustav troši na pronalaženje svih blokova neke datoteke. Što je manji broj blokova na koje dijelimo datoteku, to će vrijeme potrebno za prijenos datoteke, koja se sastoji od više blokova, više ovisiti o brzini prijenosa podataka na disku. Na primjer, ako za datoteku od 100 MB vrijeme pretraživanja iznosi 10 ms, a brzina prijenosa podataka je 100 MB/s, tada vrijeme pretraživanja čini 1% brzine prijenosa podataka. Kada bi veličina bloka bila manja od 100 MB, taj omjer bi bio nepovoljniji.

Prednosti distribuiranog datotečnog sustava koji koristi blokovsku podjelu datoteka su slijedeće:

- Datoteka može biti veća od veličine diska. Datoteke se dijele i spremaju u blokovima pa je nužno da veličina diska bude veća od veličine bloka te da u klasteru postoji dovoljno slobodne memorije za pohranu svih blokova i njihovih kopija. Teoretski, jedna datoteka sa svojim kopijama može zauzimati cijeli memorijski kapacitet klastera.



Slika 3.1: Razmještaj replika blokova u klasteru

- Odabir bloka kao osnovne jedinice, umjesto datoteke, pojednostavljuje rad datotečnog podsustava.
- Blokovska podjela omogućuje jednostavnu implementaciju oporavka u slučaju greške (engl. *fault tolerance*). Kopije blokova se pohranjuju na fizički odvojenim računalima, stoga u slučaju fizičkog kvara, kopija bloka može se dohvatiti sa drugog računala.

Razmještaj blokova

Datoteka se u HDFS pohranjuje u blokovima. Kako bi se spriječio gubitak podataka u slučaju pojave greške, HDFS pohranjuje tri kopije svakog bloka na različitim čvorovima. Zahtjev za zapisivanjem datoteke u HDFS dolazi od strane klijenta. Klijent može biti čvor koji pripada klasteru, a može se nalaziti i izvan klastera. Ukoliko je klijent čvor unutar klastera, prva replika pohranjuje se na taj čvor. U suprotnom, čvor, na kojem će biti pohranjena prva replika, bira se nasumično, pazeći pritom da odabrani čvor nije preopterećen. Čvor za drugu repliku bira se nasumično, uz uvjet da se fizički ne nalazi u istom ormaru u kojem je smještena prva replika. Treća replika smješta se unutar istog ormara u kojem se nalazi druga replika, ali na različitom čvoru, koji se također bira nasumično. Na slici 3.1 prikazan je navedeni način replikacije blokova.

Arhitektura rob-gospodar

Arhitektura Hadoop distribuiranog datotečnog sustava oblikovana je prema modelu rob-gospodar. HDFS klaster sadrži dvije vrste čvorova: *glavni čvor* (engl. *namenode*) i *podatkovni čvor* (engl. *datanode*). Glavni čvor ima ulogu gospodara, a podatkovni čvorovi imaju ulogu robova. U klasteru može postojati samo jedan aktivan glavni čvor, a podatkovnih čvorova može biti više.

Glavni čvor

Glavni čvor zadužen je za imenički prostor datotečnog sustava. Održava datotečno stablo i meta podatke za sve datoteke i direktorije u stablu. Postoje dvije vrste datoteka u koje se pohranjuju navedene informacije: slika imeničkog prostora (*namespace image*) i podaci o promjenama (*edit log*). Oba tipa datoteka pohranjuju se lokalno na disk. Za svaku datoteku u sustavu, glavni čvor sprema informacije o svim podatkovnim čvorovima na kojima se nalaze blokovi te datoteke.

Podatkovni čvor

Podatkovni čvorovi pohranjuju i dohvaćaju blokove podataka. Periodički se javljaju glavnom čvoru pomoću *heartbeat* mehanizma. Na taj način glavni čvor dobiva potvrdu o aktivnim čvorovima. Ukoliko se neki podatkovni čvor ugasi, glavni čvor započinje postupak replikacije blokova koji su bili pohranjeni na tom čvoru.

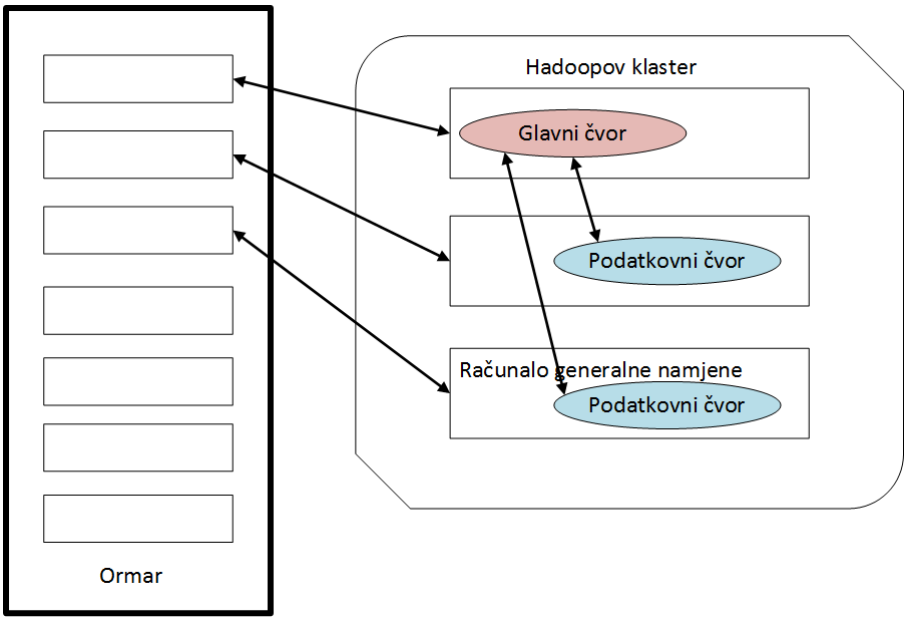
Glavni čvor šalje podatkovnom čvoru naredbe za:

- Replikacijom bloka na neki drugi čvor
- Brisanjem lokalne replike bloka
- Ponovnom registracijom čvora i slanjem informacija o svim blokovima pohranjenim na tom čvoru
- Gašenjem čvora

Sekundarni glavni čvor

Glavni čvor sadrži informacije o lokaciji svih blokova svih datoteka u datotečnom sustavu. U slučaju pojave greške i gašenja glavnog čvora, pristup datotekama bio bi onemogućen. Sekundarni glavni čvor (engl. *secondary namenode*) je pomoćni čvor, čija je uloga stvaranje kopije podataka iz glavnog čvora, pomoću kojih se može rekonstruirati glavni čvor.

Uloga sekundarnog glavnog čvora je i povremeno spajanje slike imeničkog prostora i podataka o promjenama, kako podaci o promjenama ne bi postali preveliki. Sekundarni glavni čvor i glavni čvor izvršavaju se na različitim računalima unutar klastera.



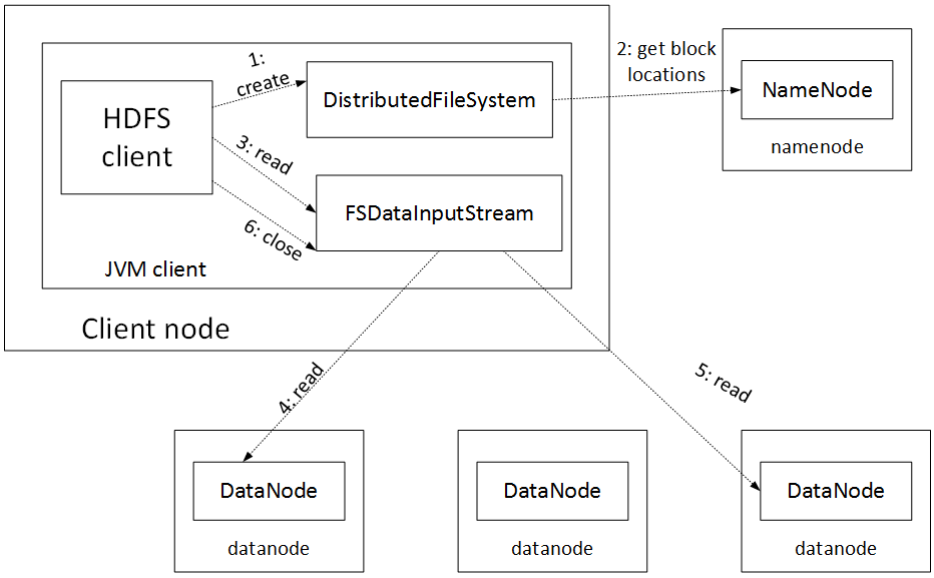
Slika 3.2: Veza Hadoopovog klastera i hardvera

Na slici 3.2 prikazana je veza Hadoopovog klastera i hardvera. Glavni čvor i podatkovni čvorovi izvršavaju se na odvojenim računalima generalne namjene.

API za HDFS

Čitanje podataka

Klijent započinje akciju čitanja datoteke kreiranjem objekta klase DistributedFileSystem te pozivanjem njegove metode open(). U metodi open(), kontaktira se glavni čvor, pomoću RPC poziva. Za svaki blok datoteke, glavni čvor vraća adrese svih podatkovnih čvorova koji sadrže kopiju tog bloka. Metoda open() vraća objekt klase FSDataInputStream, koji se odmah pretvara u objekt klase DFSInputStream. Na njemu se više puta poziva metoda read(), dok cijela datoteka nije pročitana. Klijent čita blok podataka od najbližeg čvora koji ga sadrži. Kada je pročitan cijeli blok, DFSInputStream objekt zatvara konekciju s podatkovnim čvorom te započinje tražiti najbliži čvor sa slijedećim blokom podataka. Kada je pročitana cijela datoteka, poziva se metoda close() na FSDataInputStream. Ukoliko se neki

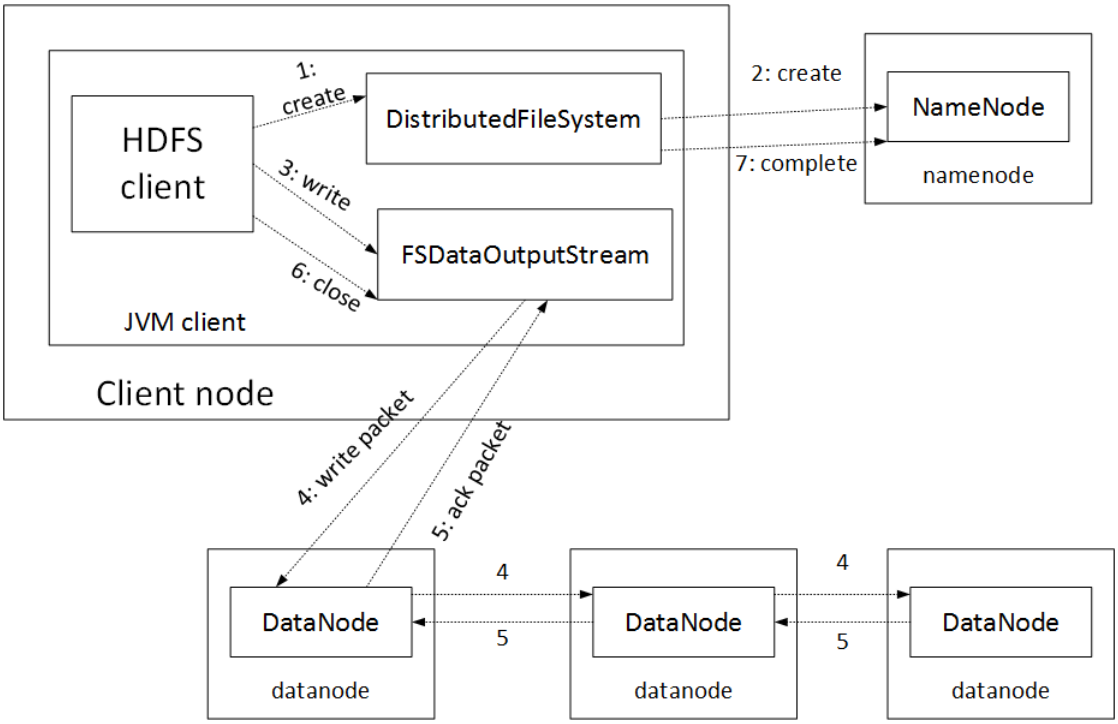


Slika 3.3: Čitanje datoteke iz HDFS-a

čvor tokom čitanja ugasi, akcija čitanja tog bloka će se ponoviti na slijedećem najbližem čvoru koji ga sadrži. Opisani tijek čitanja podataka prikazan je na slici 3.3.

Pisanje podataka

Klijent stvara novu datoteku pozivanjem metode create() na objektu klase DistributedFileSystem. U metodi create(), kontaktira se glavni čvor RPC pozivom, kako bi se stvorila nova datoteka u imeničkom prostoru datotečnog sustava. Glavni čvor prvo provjerava postoji li već takva datoteka u sustavu te ima li klijent dozvolu stvaranja nove datoteke. Ukoliko neki uvjet nije zadovoljen, akcija kreiranja datoteke će biti obustavljena i klijent će dobiti IOException. Ako su zadovoljeni traženi uvjeti, glavni čvor stvara novu datoteku. Metoda create() vraća FSDataOutputStream objekt pomoću kojeg se izvršava pisanje podataka. Podaci se dijele u pakete, koji se prvo spremaju u red. Ponovo se kontaktira glavni čvor te on njega stižu informacije o tri podatkovna čvora na kojima se trebaju spremiti replike bloka. Ta tri čvora tvore cjevovod. Paketi prvo stižu do prvog čvora, a zatim putuju dalje do drugog pa do trećeg čvora. Svaki čvor pristigle podatke pohranjuje te proslijeđuje slijedećem čvoru. Svaki čvor, nakon što primi paket, šalje poruku o primitku kroz cjevovod u suprotnom smjeru. Tek kada su pristigle potvrde od sva tri čvora za dani paket, on se može obrisati iz reda. Ukoliko dođe do gašenje nekog čvora u cjevovodu, stvara se novi cjevovod od preostalih aktivnih čvorova te se zatim poduzimaju daljnje akcije za stvaranje treće kopije bloka. Kada su podaci uspješno zapisani u sustav, poziva se metoda close() na



Slika 3.4: Unos podataka u HDFS

FSDataOutputStream. Opisani tijek zapisivanja podataka prikazan je na slici 3.4.

3.3 MapReduce

MapReduce model

Pojam *MapReduce* odnosi se na model programiranja i pripadnu implementaciju za obradu i generiranje velikih podataka na klasteru računala.

MapReduce model razvijen je u Googleu, u svrhu obrade velikih podataka. Temelji se na principu paralelnog računanja nad nekim skupom podataka.

MapReduce posao (engl. *MapReduce job*) sastoji se od ulaznih podataka, MapReduce programa i konfiguracijskih informacija.
Hadoop izvršava posao dijeleći ga u manje zadatke. Postoje dvije vrste takvih zadataka: *map zadatak* (engl. *map task*) i *reduce zadatak* (engl. *reduce task*).

Posao se sastoji od dvije faze - map i reduce, koje se odvijaju slijedno. Pritom, map zadaci pripadaju map fazi, a reduce zadaci reduce fazi. Izlaz iz map faze je ulaz u reduce

fazu.
Ulaz i izlaz u svakoj fazi su podaci oblika (ključ, vrijednost).

Zadaci se odvijaju na čvorovima unutar klastera. Za raspoređivanje i usklađivanje zadataka zadužen je poseban dio Hadoop ekosustava zvan YARN. YARN (engl. Yet Another Resource Negotiator) je sustav za upravljanje resursima unutar Hadoop klastera.

Ulazni podaci dijele se u manje dijelove fiksne veličine. Ti dijelovi nazivaju se *input splits*. Ukupan broj map zadataka ovisi upravo o broju input splitsa. Hadoop stvara po jedan map zadatak za svaki split, koji zatim poziva map funkciju za svaki zapis unutar splita.

Za većinu poslova, prikladna veličina splita je veličina HDFS bloka, koja je po defaultu jednaka 128 MB.

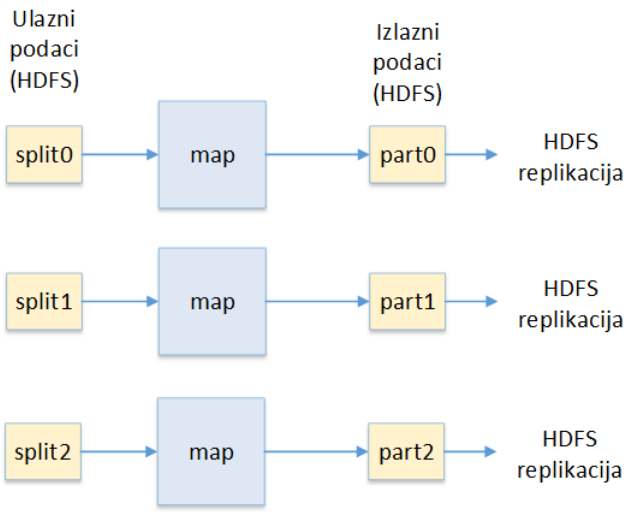
Najveća efikasnost postiže se kada se map zadatak izvršava na istom čvoru na kojem se nalaze i ulazni podaci dodijeljeni tom zadatku.

Izlazni podaci map zadatka zapisuju se lokalno na disk, a ne u Hadoop distribuirani datotečni sustav. Razlog tome je način na koji HDFS sprema podatke, koji je ranije opisan u radu. Izlazni podaci map zadataka su među podaci cijelog MapReduce posla pa nema potrebe opterećivati HDFS sa dodatnom replikacijom podataka koji, nakon završetka posla, više neće biti potrebni. Ukoliko se čvor, na kojem se odvija map zadatak, ugasi, prije nego što je reduce zadatak iskoristio među podatke, Hadoop će automatski ponoviti taj map zadatak na nekom drugom čvoru.

Ulazni podaci za reduce zadatak sastoje se od skupa izlaznih podataka svih map zadataka. Izlazni podaci svih map zadataka šalju se mrežom do čvora na kojem se odvija reduce zadatak. MapReduce posao može se sastojati od 0, 1 ili više reduce zadataka.

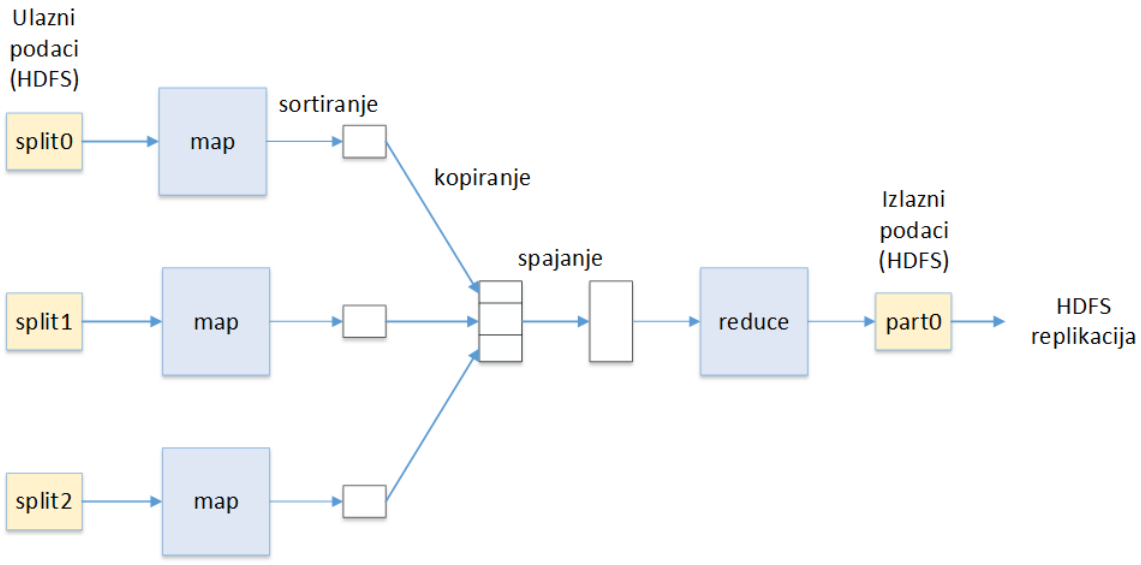
Označimo s N broj reduce zadataka. Promotrimo tok podataka u sljedeća tri slučaja (slike 3.5, 3.6, 3.7):

1. Prvi slučaj: $N = 0$



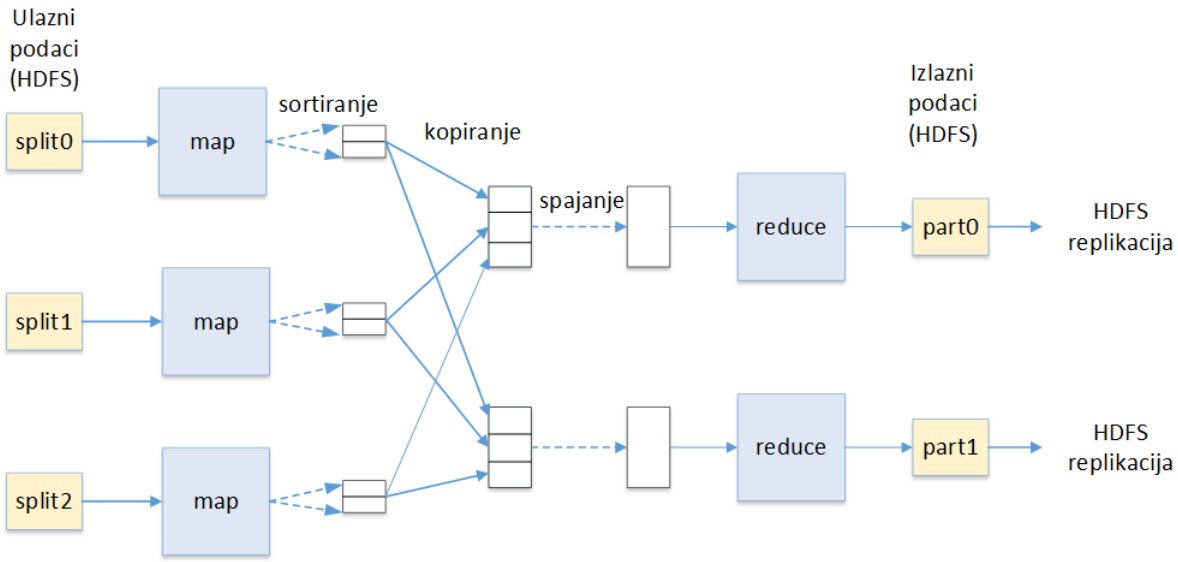
Slika 3.5: Tok podataka bez reduce zadatka

2. Drugi slučaj: $N = 1$



Slika 3.6: Tok podataka sa jednim reduce zadatkom

3. Treći slučaj: $N = 2$



Slika 3.7: Tok podataka sa dva reduce zadatka

Ukoliko postoji barem jedan reduce zadatak ($N \geq 1$), događa se sljedeće:

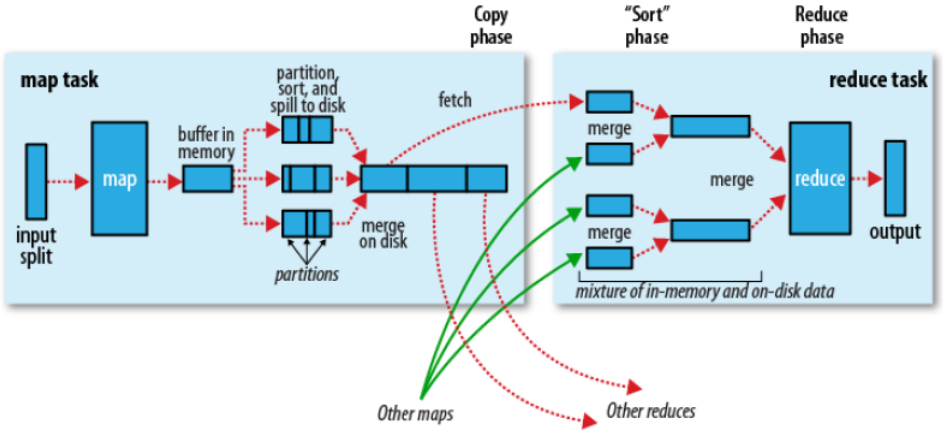
1. izlazni podaci iz map zadatka sortiraju se lokalno na čvoru zaduženom za taj map zadatak te se grupiraju u N particija
2. sortirani podaci zatim se kopiraju
3. kopije podataka šalju se mrežom do čvora na kojem se izvršava reduce zadatak
4. podaci se tamo spajaju (merge) te tada postaju ulaznim podacima za reduce zadatak

Korisnično sučelje

Implementacija modela MapReduce u Javi dana je klasama Mapper i Reducer. U klasi Mapper nalazi se apstraktna metoda *map()* u kojoj definiramo map fazu. Slično, u metodi *reduce()* klase Reducer definiramo reduce fazu.

Osnovne zadaće MapReduce okvira:

- **Raspoređivanje (scheduling)** : MapReduce posao dijeli se na map zadatke i na reduce zadatke. MapReduce okvir dodijeljuje map zadacima veći prioritet nego reduce



Slika 3.8: Shuffle and Sort

zadacima, budući da svi map zadaci trebaju završiti da bi se reduce zadaci krenuli izvršavati.

- **Sinhronizacija** : kada se više procesa odvija u isto vrijeme unutar jednog klastera, potrebno je uvesti sinhronizacijski mehanizam koji će se brinuti o pravilnom redoslijedu izvršavanja zadataka. Kada je map faza (djelomično) završena, mehanizam "shuffle and sort" kopira i distribuira kroz mrežu među-podatke koji će postati ulazni podaci za reduce fazu.

- **Razmještaj koda i podataka**: najveća efikasnost postiže se kada su kod i podaci, koje taj kod treba obraditi, smješteni unutar istog stroja.

- **Fault/error handling**: unutar MapReduce okvira, predviđene su pojave različitih grešaka te su definirane odgovarajuće akcije. Na primjer, ako neki čvor, kojem je dodijeljen map zadatak, ne završi s radom na očekivani način, okvir će taj zadatak automatski dodijeliti nekom drugom čvoru.

Shuffle and sort

Već je istaknuto da izlazni podaci iz map faze čine ulazne podatke u reduce fazu. Sada će biti malo detaljnije objašnjen proces koji obuhvaća pohranu, pripremu i prijenos tih podataka iz map faze u reduce fazu. Proces je skiciran na slici (3.8).

MapReduce posao započinje tako da se ulazni podaci particioniraju na manje dijelove (input splits) te se za svaki manji dio kreira map zadatak kojem su to ulazni podaci. Kada map zadatak krene stvarati izlazne podatke, oni se najprije zapisuju u međuspremnik (buffer). Veličina međuspremnika definirana je svojstvom `mapreduce.task.io.sort.mb` i

iznosi $100MB$ po defaultu, a moguće ju je promjeniti. Kada podaci popune 80% veličine međuspremnika[4], pozadinska dretva počinje zapisivati podatke iz međuspremnika na disk.

Prije nego zapiše podatke na disk, pozadinska dretva dane podatke particionira, ovisno o reduce zadatku kojem su namjenjeni. Podaci se unutar svake particije sortiraju po ključu. Ukoliko je u kodu definirana combiner funkcija, ona se tada poziva nad sortiranim podacima. Svrha korištenja combiner funkcije je optimizacija resursa i ona ovisi o samom zadatku.

Svaki put kada podaci u međuspremniku dosegnu definiranu razinu popunjenosti spremnika, pozadinska dretva, osim što počinje proces premještanja sadržaja na disk, kreira novu datoteku za tu konkretnu akciju pražnjenja međuspremnika (engl. *disk spill*). U svakom map zadatku može se nalaziti više takvih datoteka o pražnjenju međuspremnika. Neposredno prije završetka map zadataka, sve datoteke se spajaju u jednu datoteku (particiju) koja predstavlja izlazni podatak map zadatka.

Ukoliko u zadatku postoje barem tri datoteke o pražnjenju spremnika, prije kreiranja izlazne datoteke, ponovo će se pozvati combiner funkcija.

Opcionalno, moguće je definirati da se izlazna datoteka komprimira prije nego što se zapiše na disk.

Faza kopiranja

Reduce zadatak treba prvo prikupiti sve svoje ulazne podatke. Taj proces započinje čim prvi map zadatak završi s radom. U reduce zadatku nalaze se dretve namjenjene za kopiranje tih podataka. Čim neki map zadatak završi s radom, dretva započinje kopiranje particije koja je potrebna za dani reduce zadatak. Maksimalan broj dretvi, koje paralelno kopiraju podatke, definira se svojstvom `mapreduce.reduce.shuffle.parallelcopies`[5].

Ukoliko su kopirani podaci dovoljno mali[6], pohranjuju se u memoriju JVM-a, inače se pohranjuju na disk. Sličan princip, koji je već opisan u map fazi, koristi se i u reduce fazi. Kada količina podataka u međuspremniku dosegne definiranu razinu, pozadinska dretva započinje spajanje i zapisivanje tih podataka na disk. Pozadinska dretva povremeno spaja nakupljene datoteke na disku u novu sortiranu datoteku. Ukoliko se koristila kompresija podataka u map fazi, prije sortiranja u reduce fazi, svakako se treba provesti dekompresija podataka.

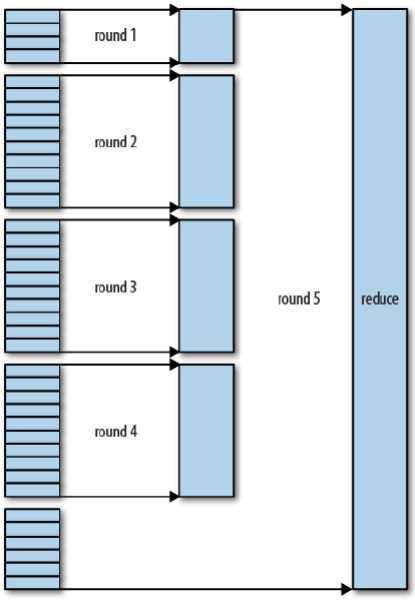
Faza sortiranja

Nakon što su prikupljeni svi podaci iz map faze, započinje faza sortiranja. Prikupljeni podaci tada se spajaju s ciljem stvaranja sortiranog niza ulaznih podataka za reduce zadatak.

[4] defaultna vrijednost svojstva mapreduce.map.sort.spill.percent iznosi 0.80, a moguće ju je promjeniti

[5] Defaultan broj dretvi je pet.

[6] Definirano svojstvom `mapreduce.reduce.shuffle.input.buffer.percent`



Slika 3.9: Spajanje 40 datoteka uz faktor k = 10

Kopirane datoteke iz map zadataka već sadrže sortirane podatake pa ih je potrebno samo ujediniti, pazeći pritom na pravilan redoslijed. Broj datoteka koje ulaze u jedan merge poziv, definiran je svojstvom `mapreduce.task.io.sort.factor`.

Označimo taj faktor sa k. Broj datoteka koje sudjeluju u jednom koraku spajanja može biti manji ili jednak k. Pri jednom koraku spajanja, nastaje nova datoteka sa sortiranim podacima, koja se zapisuje na disk. Na primjer, za 40 datoteka, uz $k = 10$, biti će potrebno pet koraka kako bi se stvorio konačni rezultat. Konačni rezultat se proslijeđuje direktno reduce funkciji bez zapisivanja cijelog niza na disk, za razliku od među-rezultata, koji se zapisuju na disk. Kako bi se minimizirao broj zapisivanja na disk, u jednom koraku spajanja može sudjelovati manje od k datoteka, kako bi u zadnjem koraku, zbroj svih među-rezultata i preostalih neobrađenih datoteka bio jednak k. Primjer je vidljiv na slici 3.9.

Implementacija MapReduce modela u Javi

U paketu `org.apache.hadoop.mapreduce` nalaze se klase `Mapper` i `Reducer`, pomoću kojih je moguće implementirati MapReduce model u Javi.

Potrebno je implementirati dvije nove klase: jednu koja naslijeđuje klasu `Mapper` te drugu koja naslijeđuje klasu `Reducer`.

Klasa `Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>` sadrži metodu `map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)`, koja

se poziva za svaki par (key, value) ulaznih podataka. Metoda map od ulaznih podataka kreira izlazne podatke map faze (među-podatke MapReduce algoritma). Potrebno je implementirati metodu map. Po defaultu, ona je jednaka identiteti.

Klasa `Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>` sadrži metodu `reduce(KEYIN key, Iterable<VALUEIN> values, org.apache.hadoop.mapreduce.Reducer.Context context)` koja se poziva jednom za svaki (key, collection of values). Metodu reduce također je potrebno implementirati. U suprotnom, biti će jednaka identiteti.

Klasa `Job` omogućuje korisniku unos konfiguracijskih podataka za MapReduce posao.

Primjer 1 - WordCount

Ovo je jednostavni primjer koji ilustrira princip rada MapReduce algoritma. Često se naziva *HelloWorld* primjerom za MapReduce. Program za ulazni tekstualni skup podataka ispisuje koliko puta se pojedina riječ pojavljuje u tekstu. U ovom primjeru, koristit ćemo tri tekstualne datoteke različitog sadržaja.

```
file0.txt:  Hadoop is an elephant

file1.txt:  Hadoop is as yellow as can be

file2.txt:  Oh what a yellow fellow is Hadoop
```

Slika 3.10: Ulazni podaci

Zadaća map faze je pronaći sve riječi u tekstu te formirati izlaz kao multiskup parova $< w, 1 >$, gdje je w riječ. Zatim slijedi reduce faza, čiji zadatak je prebrojati koliko ima parova sa istim ključem te formirati novi niz parova $< w, n >$, gdje w označava riječ, a n broj pojavljivanja riječi w.

```
package org.myorg;

import java.io.IOException;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
```

```
Hadoop  3
Oh      1
a       1
an      1
as      2
be      1
can     1
elephant        1
fellow  1
is      3
what    1
yellow  2
```

Slika 3.11: Rezultat

```
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(WordCount.class);

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new WordCount(), args);
    System.exit(res);
  }

  public int run(String[] args) throws Exception {
    Job job = Job.getInstance(getConf(), "wordcount");
    job.setJarByClass(this.getClass());
    // Use TextInputFormat, the default unless job.setInputFormatClass is
        used
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(Map.class);
```

```
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static class Map extends Mapper<LongWritable, Text, Text,
      IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private long numRecords = 0;
    private static final Pattern WORD_BOUNDARY =
        Pattern.compile("\\s*\\b\\s*");

    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      Text currentWord = new Text();
      for (String word : WORD_BOUNDARY.split(line)) {
        if (word.isEmpty()) {
            continue;
        }
            currentWord = new Text(word);
            context.write(currentWord,one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text,
      IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context
        context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
        sum += count.get();
      }
      context.write(word, new IntWritable(sum));
    }
  }
}
```

3.4 Hadoop Ekosustav

Naziv Hadoop koristi se i za cijelu familiju srodnih projekata koji podržavaju distribuiranu obradu velikih skupova podataka. Dvije komponente ekosustava već su opisane (HDFS i MapReduce), a u nastavku će biti navedene i ukratko opisane još neke. Hadoopov ekosustav je bogat i njegove komponente razvijene su kako bi pružile različite funkcionalnosti potrebne u radu s velikim podacima.

- **Common**: Skup komponenti i sučelja za distribuirane datotečne sustave te opću I/O namjenu (serijalizacija, Java RPC, strukture podataka).
- **Pig**: Jezik i okolina za skriptiranu obradu i analizu podataka, s fokusom na tok podataka (*dataflow*).
- **Oozie**: Sustav zadužen za raspoređivanje Hadoopovih poslova (*workflow*). Hadoopovi poslovi mogu se predočiti kao usmjereni aciklički graf akcija.
- **ZooKeeper**: Distribuirani servis za koordinaciju distribuiranih sustava. Nudi podršku za sinhronizaciju procesa, izbor vođe i slično.
- **Mahout**: Podrška za strojno učenje. Sadrži algoritme za klasifikaciju, klasteriranje, filtriranje.

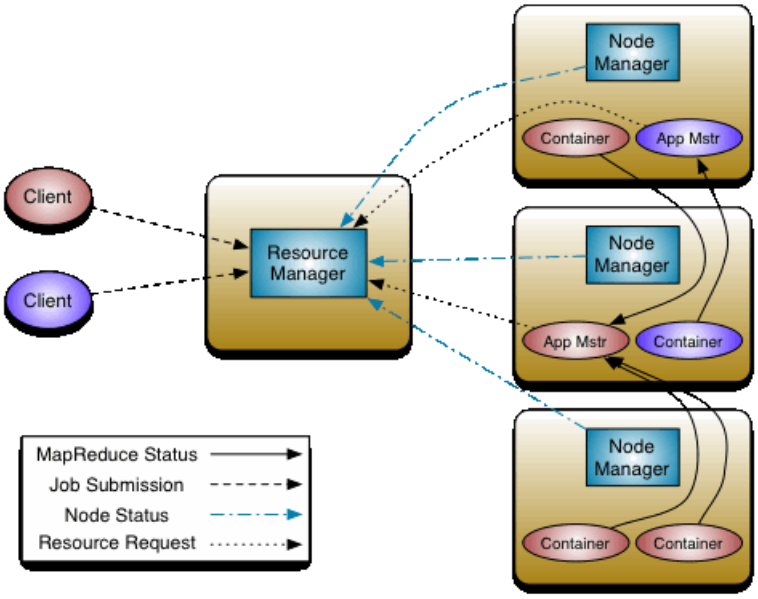
U nastavku će malo detaljnije biti opisani **YARN**, **HBase**, **Spark** i **Hive**.

YARN

YARN (Yet Another Resource Negotiator) je sustav za upravljanje resursima i raspoređivanje poslova u Hadoopovom klasteru.

Uveden je u drugom izdanju Hadoopa (Hadoop 2), kako bi se zadaci upravljanja resursima i raspoređivanja poslova odvojili od MapReduce okvira. YARN okvir kompatibilan je sa ostalim komponentama u Hadoopovom sustavu. Sadrži aplikacijska programska sučelja (API), preko kojih se upravlja resursima u klasteru. Korisnici (programeri) ne trebaju direktno pristupati YARN-ovim sučeljima u kodu, već se za to brine okvir distribuirane okoline poput MapReducea, Sparka, Teza i slično.

Arhitektura YARN-a prikazana je na slici 3.12. Sastoji se od dvije vrste pozadinskih procesa (*deamona*): ResourceManager i NodeManager.



Slika 3.12: Arhitektura YARN-a

ResourceManager ima ulogu upravitelja resursima. U klasteru se nalazi po jedan ResourceManager, koji je odgovoran samo za taj klaster. Zadužen je za dodijeljivanje resursa svim aplikacijama u sustavu.

NodeManager zadužen je za komunikaciju sa ResourceManagerom. NodeManager nalazi se na svakom čvoru klastera. Nadgleda korištenje resursa (procesor, memorija, disk, mreža) spremnika (Containera) te o tome obavještava ResourceManager-a.

Za svaku aplikaciju u sustavu postoji po jedan ApplicationMaster. Aplikacija se može sastojati od jednog posla ili usmjerenog acikličkog grafa poslova.

ResourceManager sastoji se od dvije komponente: Scheduler i ApplicationsManager.

Scheduler je zadužen za alokaciju resursa aplikacija. Ne nadgleda daljnje ponašanje aplikacije pa u slučaju pojave greške (aplikacijska ili hardverska greška), ne garantira ponavljanje neuspjelog zadatka.

ApplicationsManager je zadužen za prihvaćanje posla, pokretanje ApplicationMaster i njegovo ponovno pokretanje u slučaju greške.

HBase

HBase je distribuirana stupčana baza podataka. Stupčane baze podataka podvrsta su NoSQL baza podataka.

HBase je baza otvorenog koda, implementirana u programskom jeziku Java. Razvijena je po uzoru na Google Big Table. Koristi Hadoop distribuirani datotečni sustav za pohranu podataka.

Omogućava pohranu vrlo velikih tablica, primjerice od milijardu redaka i milijun stupaca.

HBase posjeduje osnovne karakteristike Hadoopovog sustava: (linearna) skalabilnost, mogućnost pohrane velikih podataka, izvršavanje na klasteru računala generalne namjene.

HBase je distribuirana baza podataka, izgrađena po modelu rob-gospodar. Prisutna su dva tipa čvorova: HBase Master i RegionServer. HBase Master ima ulogu gospodara, a čvorovi tipa RegionServer imaju ulogu robova.

Operacije poput čitanja i pisanja podataka odvijaju se u realnom vremenu. U oba slučaja, klijent prvo kontaktira HBase Master, kako bi saznao u kojoj regiji se nalazi podatak sa danim ključem. HBase Master odgovara sa informacijama o regiji i čvoru u kojima je podatak pohranjen. Klijent zatim direktno kontaktira čvor RegionServer na kojem je pohranjen podatak.

Direktnom komunikacijom klijenta i čvora na kojem su pohranjeni podaci (čvora poslužitelja), izbjegava se pojava greške (*single point of failure*) i uskog grla u distribuiranom sustavu.

Informacije o RegionServer čvoru klijent pohranjuje u radnu memoriju, što omogućava brži ponovni pristup podacima.

HBase Manager i NameNode u pravilu se izvršavaju na istom računalu, kao i parovi DataNode i RegionServer čvorova.

Spark

Apache Spark je okvir otvorenog koda namijenjen za izračunavanja na klasteru računala.

Razvijen je u AMPLab[7], u sklopu kalifornijskog sveučilišta Berkeley. Kasnije je priključen Apache Software Foundation, gdje se i danas razvija.

Glavna karakteristika Sparka je velika brzina izračunavanja.

Za razliku od većine komponenti u Hadoopovom ekosustavu, Spark ne koristi MapReduce okvir za izračunavanje unutar klastera. Umjesto toga, koristi vlastitu okolinu za distribuirano izvršavanje u klasteru, koja ima dosta sličnosti sa MapReduce okolinom.

Prednost Sparkove okoline je postizanje puno veće brzine, zahvaljujući pohranjivanju među-rezultata u memoriji, a ne na disku. Sparkov model izračunavanja posebice je prikladan za korištenje u iterativnim algoritmima i interaktivnoj analizi.

Spark se može izvršavati kao zasebna cjelina, u kombinaciji sa Hadoop-om, Apache Mesosom ili u oblaku.

Kompatibilan je s komponentama Hadoop ekosustava (YARN, HDFS, HBase).

Pruža podršku za razvoj aplikacija koristeći jezika Java, Scala, Python i R.

[7] Algorithms, Machines and People Lab

Integrirane biblioteke pružaju široki spektar mogućnosti primjene Sparka.
Biblioteke unutar Sparka:

- SQL and DataFrames
- MLLib
- GraphX
- Spark Streaming

Spark je dobar izbor za probleme koji se trebaju riješiti u realnom vremenu, na primjer:

- Interaktivni upiti, gdje se odgovor očekuje unutar najviše nekoliko sekundi
- Obrada toka podataka (stream data) u realnom vremenu: obrada log datoteka; detekcija pokušaja prijevare i kreiranje *alerta*
- Obrada podataka koji pristižu od senzora

MapReduce pruža manju brzinu, no zadovoljavajuću za npr. batch obrade.

Hive

Apache Hive je softver za skladišta podataka koji omogućava rad s velikim podacima (čitanje, pisanje, upiti, analize) u distribuiranoj okolini.

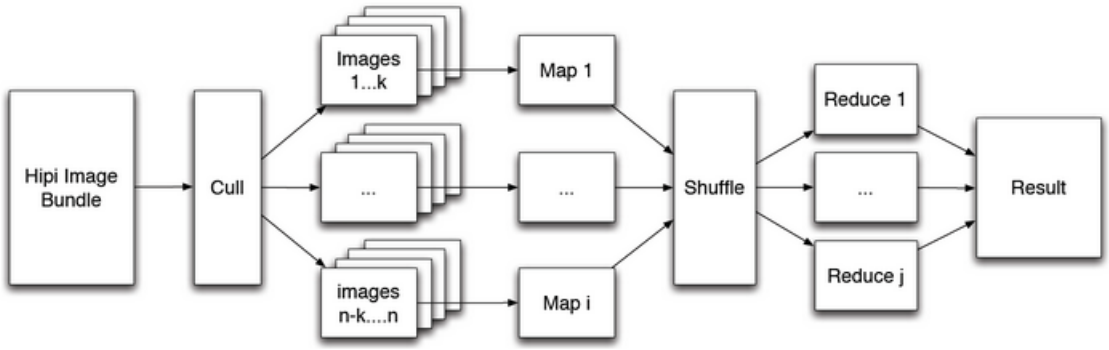
Hive je započeo svoj razvoj u Facebooku, iz potrebe da se stvori okvir koji podržava upite nad velikim podacima te će biti jednostavan za korištenje analitičarima koji već jako dobro poznaju SQL, ali nisu vješti u programiranju u Javi.

3.5 Obrada nestrukturiranih podataka

Obrada slika

Veliku količinu nestrukturiranih podataka čine upravo slike. Danas, u vrijeme pametnih telefona, fotografije preplavljuju društvene mreže i medije. Satelitske snimke, karte, medicinska dokumentacija (CT snimke) samo su neki od primjera podataka koje možemo promatrati kao slike u sustavu za rad s velikim podacima.

HIPI (Hadoop Image Processing Interface) je sučelje za obradu slika u Hadoop okruženju. Slike se pohranjuju u Hadoop distribuirani datotečni sustav te se zatim paralelno obrađuju, na klasteru računala generalne namjene, pomoću MapReduce okvira.



Slika 3.13: HIPI - faze obrade slika

Budući da je veličina prosječne slike znatno manja od veličine bloka u HDFS-u, a HDFS nije namjenjen za pohranu malih datoteka, slike se grupiraju u posebne vrstu datoteka.

HipiImageBundle (HIB) je kolekcija slika koja u HDFS-u predstavlja jednu datoteku sa slikama te ujedno i osnovni ulazni tip podatka u HIPI program.

U HIPI distribuciji dostupni su alati pomoću kojih je moguće jednostavno kreirati HIB datoteku od kolekcije slika.

Tipične faze obrade slika pomoću HIPI programa vidljive su na slici 3.13.

Ulazni podataci sastoje se od HIB datoteka. Prva faza obrade je faza raspodjele, odnosno odabira podataka (engl. *cull*). Odabir podataka moguće je definirati u kodu, pomoću funkcionalnosti dostupnih u Culler klasi (npr. rezolucija slike). Slike koje ne zadovoljavaju zadane kriterije, dalje se ne obrađuju. Slike koje zadovolje kriterije, postaju ulazni podaci map zadataka, kao objekti klase ByteImage ili FloatImage, koji nude mnogobrojne funkcionalnosti karakteristične za obradu slika: konverzija boja, izrezivanje dijela fotografije, skaliranje.

HIPI nudi podršku za OpenCV, biblioteku otvorenog koda za računalni vid.

Moguća je konverzija objekata klase ByteImage i FloatImage u objekte klase OpenCV Java Mat. OpenCVMatWritable klasa koristi se za definiranje ključeva i vrijednosti u MapReduce programu.

Poglavlje 4

Studijski primjer

4.1 Zadatak

Zadatak je napisati MapReduce program koji će, za zadani niz riječi, izračunati broj pojavljivanja svake od tih riječi u skupu knjiga. Skup knjiga sastoji se od .txt datoteka.

Napomene

Pretpostavit ćemo da se, prije izvršavanja programa, zadani skup riječi nalazi u .txt datoteci u lokalnom datotečnom sustavu, a knjige u .txt formatu u Hadoop distribuiranom datotečnom sustavu.

U izradi rješenja koristila sam Cloudera CDH 5.8.0. za VirtualBox, single-node setup Hadoopa (pseudo-cluster setup zahtjeva više radne memorije, nego što ima moje računalo).

4.2 Rješenje

Kod programa

```
package org.myorg;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.net.URI;
import java.util.HashSet;
import java.util.Set;
import java.io.IOException;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.StringUtils;

import org.apache.log4j.Logger;


public class ProjektKnjige extends Configured implements Tool {

  private static final Logger LOG = Logger.getLogger(ProjektKnjige.class);

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new ProjektKnjige(), args);
    System.exit(res);
```

```
}

public int run(String[] args) throws Exception {
  Job job = Job.getInstance(getConf(), "projektknjige"); //
      getInstance(Configuration conf, String jobName)
  for (int i = 0; i < args.length; i += 1) {
    if ("-match".equals(args[i])) {
      job.getConfiguration().setBoolean("projektknjige.match.patterns",
          true);
      i += 1;
      job.addCacheFile(new Path(args[i]).toUri());
      // this demonstrates logging
      LOG.info("Added file to the distributed cache: " + args[i]);
    }
  }
  job.setJarByClass(this.getClass());
  // Use TextInputFormat, the default unless job.setInputFormatClass is
      used
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  job.setMapperClass(Map.class);
  job.setCombinerClass(Reduce.class);
  job.setReducerClass(Reduce.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  return job.waitForCompletion(true) ? 0 : 1;
}

public static class Map extends Mapper<LongWritable, Text, Text,
    IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  private boolean caseSensitive = false;
  private long numRecords = 0;
  private String input;
  private Set<String> patternsToMatch = new HashSet<String>();
  private static final Pattern WORD_BOUNDARY =
      Pattern.compile("\\s*\\b\\s*");

  protected void setup(Mapper.Context context)
      throws IOException,
      InterruptedException {
```

```
    if (context.getInputSplit() instanceof FileSplit) {
      this.input = ((FileSplit)
          context.getInputSplit()).getPath().toString();
    } else {
      this.input = context.getInputSplit().toString();
    }
    Configuration config = context.getConfiguration();
    this.caseSensitive =
        config.getBoolean("projektknjige.case.sensitive", false);
    if (config.getBoolean("projektknjige.match.patterns", false)) {
      URI[] localPaths = context.getCacheFiles();
      parseMatchFile(localPaths[0]);
    }
  }

  private void parseMatchFile(URI patternsURI) {
    LOG.info("Added file to the distributed cache: " + patternsURI);
    try {
      BufferedReader fis = new BufferedReader(new FileReader(new
          File(patternsURI.getPath()).getName()));
      String pattern;
      while ((pattern = fis.readLine()) != null) {
        patternsToMatch.add(pattern);
      }
    } catch (IOException ioe) {
      System.err.println("Caught exception while parsing the cached file
          '"
          + patternsURI + "' : " + StringUtils.stringifyException(ioe));
    }
  }

  public void map(LongWritable offset, Text lineText, Context context)
      throws IOException, InterruptedException {
    String line = lineText.toString();
    if (!caseSensitive) {
      line = line.toLowerCase();
    }
    Text currentWord = new Text();
    for (String word : WORD_BOUNDARY.split(line)) {
      if (word.isEmpty() || !(patternsToMatch.contains(word))) {
        continue;
      }
```

```
            currentWord = new Text(word);
            context.write(currentWord,one);
        }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text,
      IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context
        context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
        sum += count.get();
      }
      context.write(word, new IntWritable(sum));
    }
  }
}
```

Opis programa

Prevođenje i pokretanje programa

Za početak, kreiramo direktorij *build*, u kojem će se nalaziti .class datoteke. Putem komandne linije, možemo koristiti naredbu `mkdir -p build`.
Program prevodimo naredbom:
`javac -cp /usr/lib/hadoop/*:/usr/lib/hadoop-mapreduce/* ProjektKnjige.java -d build -Xlint`
Zatim kreiramo .jar datoteku:
`jar -cvf projektknjige.jar -C build/ .`
Ulazni niz riječi, pohranjen u lokalnom datotečnom sustavu u datoteci words.txt, ubacujemo u HDFS naredbom:
`hadoop fs -put words.txt /user/cloudera/projektknjige/`
Program pokrećemo putem komandne linije, naredbom oblika:
`hadoop jar <jar> [mainClass] args...`
Konkretnije, u ovom primjeru:
`hadoop jar projektknjige.jar org.myorg.ProjektKnjige /user/cloudera/projektknjige/input /user/cloudera/projektknjige/output`

```
-match /user/cloudera/projektknjige/words.txt
```

Argumenti koje unosimo pri pokretanju programa trebaju sadržavati putanju do ulaznih podataka (u HDFS), putanju u kojoj će biti pohranjeni izlazni podaci (u HDFS), te putanju .txt datoteke, u kojoj je pohranjen niz riječi. Po defaultu, program ne radi razliku između malih i velikih slova.
Ukoliko želimo da takve riječi prebrojava odvojeno, potrebno je u argumentima navesti `-Dprojektknjige.case.sensitive=true`, odnosno definirati sistem varijablu. U tom slučaju, pokrećemo putem naredbe:

```
hadoop jar projektknjige.jar org.myorg.ProjektKnjige
-Dprojektknjige.case.sensitive=true /user/cloudera/projektknjige/input
/user/cloudera/projektknjige/output
-match /user/cloudera/projektknjige/words.txt
```

Naredbe za pokretanje su dosta dugačke i nezgodne za upisivanje. Kao pomoćni alat možemo koristiti Makefile.

Opis programa

Argumente će preuzeti funkcija main, u kojoj se poziva metoda run. U metodi run, argumenti će se proslijediti odgovarajućim funkcijama. Za početak, instancira se objekt job klase Job. Klasa Job namjenjena je za pohranu konfiguracijskih podataka o mapreduce poslu.

U programu je definirano da se, neposredno prije navođenja datoteke u kojoj je pohranjen niz riječi, treba nalaziti string ”-match”. Prvi slijedeći argument poslije ”-match”, odnosi se na datoteku s riječima. Ta datoteka zatim se dodaje u distribuiranu cache memoriju, kako bi bila vidljiva map zadacima.

Dalje se na objektu job pozivaju metode koje postavljaju Map klasu (definiranu u nastavku) za MapperClass, odnosno klasu zaduženu za map fazu, te analogno Reduce klasu. Reduce klasa imat će i ulogu Combinera. Izlazni podaci su oblika (ključ, vrijednost). Tip ključa postavlja se na Text, a vrijednosti na IntWritable.

Za unos putanje ulaznih podataka, poziva se metoda addInputPath apstraktne klase FileInputFormat. Za unos putanje izlaznih podataka, poziva se metoda setOutputPath apstraktne klase FileOutputFormat.

Klasa Map

U klasi `Map` nalazi se metoda setup, koja služi za postavljanje početnih uvjeta za map zadatak. Metoda parseMatchFile dodaje zadane riječi iz distribuirane memorije u skup patternsToMatch. Metoda map(ključ, vrijednost, context), kao vrijednost prima jedan redak teksta ulaznog dokumenta. Iz retka se zatim izdvajaju riječi te se za svaku nepraznu riječ provjerava nalazi li se ona u zadanom uzorku riječi. Ako se nalazi, onda ju treba prebrojati, odnosno zapisati u context kao par (riječ, 1).

Klasa Reduce

Klasa `Reduce` sadrži metodu reduce, čija zadaća je sumirati sve vrijednosti po danom ključu. Primjetimo da se ta redukcija odvija i kod Combinera.

Testni primjer 1

Za ulazne podatke koristila sam besplatne e-knjige iz "Project Gutenberg" repozitorija, dostupne na `https://www.gutenberg.org/`. Zadani niz riječi prikazan je na slici 4.1.

```
hadoop
is
so
cool
books
are
an
exapmle
of
unstructured
data
```

Slika 4.1: Zadane riječi

U ovom primjeru, case sensitive opcija nije uključena. Rezultati izvođenja programa nad skupom knjiga prikazani su na 4.2, a rezultati izvođenja pojedinačno nad svakom knjigom vidljivi su na 4.3 i 4.4. Očekivano, rezultati nad skupom knjiga jednaki su sumi rezultata izvođenja programa nad svakom knjigom pojedinačno.

```
an      412
are     407
books   10
cool    9
data    2
is      982
of      4771
so      746
```

Slika 4.2: Rezultati

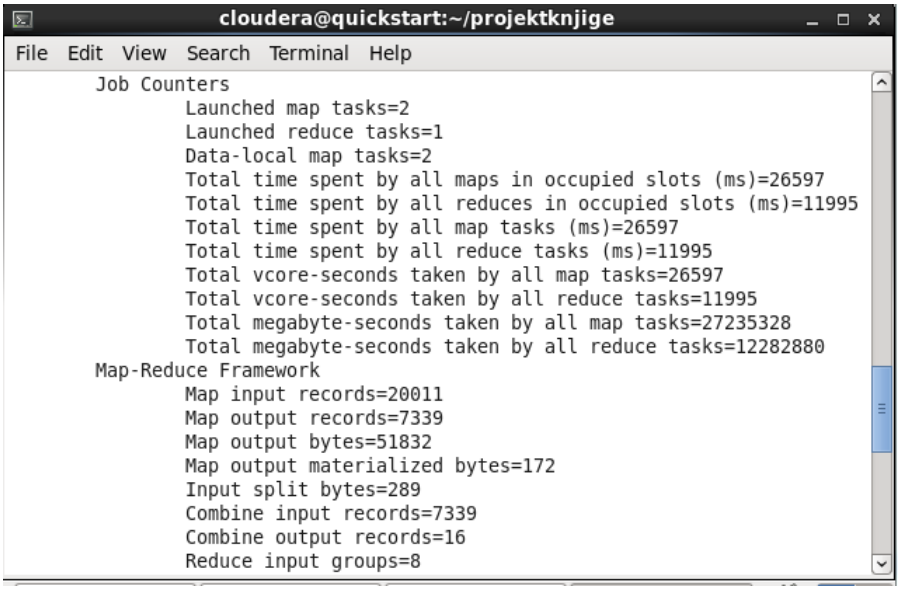
```
an      61
are     73
books   2
cool    2
data    1
is      135
of      631
so      152
```

Slika 4.3: fileAlice

```
an      351
are     334
books   8
cool    7
data    1
is      847
of      4140
so      594
```

Slika 4.4: fileCharles

Tijekom izvođenja programa, ispisuju se razne poruke vezane uz izvršavanje programa. Dio njih vidljiv je na slici 4.5, a odnosi se na izvođenje nad skupom knjiga. U ovom primjeru, podaci zauzimaju svaki po jedan blok. Vidi se da su u ovom primjeru bila potrebna samo dva map zadatka, jer pretražujemo samo dva bloka (dvije knjige).



Slika 4.5: Podaci o izvršavanju mapreduce posla

Testni primjer 2

Ulazni podaci su isti kao u prethodnom primjeru, uz uvjet da se tražene riječi moraju poklapati sa zadanim riječima po kriteriju velikih i malih slova (case sensitive). Rezultati izvođenja programa na skupu knjiga vidljivi su na slici 4.6, a pojedinačni rezultati po knjigama prikazani su na 4.7 i 4.8.

```
an      396
are     382
books   10
cool    9
data    2
is      916
of      4709
so      665
```

Slika 4.6: Rezultati (case sensitive)

```
an      56
are     63
books   2
cool    2
data    1
is      123
of      606
so      125
```

Slika 4.7: fileAlice (case sensitive)

```
an      340
are     319
books   8
cool    7
data    1
is      793
of      4103
so      540
```

Slika 4.8: fileCharles (case sensitive)

Bibliografija

[1] T. White, *Hadoop: The Definitive Guide*, 4. izdanje, O'Reilly Media, 2015.

[2] J. Hurwitz, A. Nugent, F. Halper, M. Kaufman, *Big Data For Dummies*, John Wiley & Sons, 2013.

[3] R. D. Schneider, *Hadoop For Dummies*, John Wiley & Sons, 2012.

[4] V. Mayer-Schönberger, K. Cukier, *Big Data: A Revolution That Will Transform How We Live, Work and Think*, John Murray, 2013.

[5] O'Reilly Team, *Big Data Now: 2015 Edition*, O'Reilly Media, 2015.

[6] J. Dean, S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, white paper, Google, 2004., https://research.google.com/archive/mapreduce.html

Sažetak

U radu je objašnjen pojam velikih podataka te su navedeni primjeri takvih podataka. Velika količina podataka, brzina generiranja podataka i raznolikost glavne su karakteristike velikih podataka. Zbog tih svojstava, veliki podaci zahtjevaju drugačiji pristup. Tehnologije za rad s velikim podacima obuhvaćaju koncepte koji se ranije nisu koristili u obradi podataka. Ti principi ilustrirani su u opisu Hadoop sustava, koji predstavlja polazišnu točku u razvoju softvera za rad s velikim podacima. Razumijevanje MapReducea i HDFS-a, kao osnovnih komponenti Hadoopa, važno je jer obuhvaća temeljne ideje koje se primjenjuju u cijelom Hadoopovom ekosustavu, ali i šire. U četvrtom poglavlju nalazi se studijski primjer u kojem je implementiran MapReduce model za obradu nestrukturiranih podataka (knjiga).

Summary

In this thesis we explain the term big data, and show the examples of such data. Volume, Velocity and Variety (3V) are the main characteristics of big data. Because of these properties, big data require a different approach. Big data technologies include concepts that haven't been used previously in data processing. These principles are illustrated in the description of Hadoop system, which is the starting point for development of big data software. Understanding of MapReduce and HDFS, as the main components of Hadoop, is important, because they include the basic ideas that apply to Hadoop ecosystem. In the fourth chapter is shown the case study on unstructured data (books).

Životopis

Rođena sam 29. rujna 1987. godine u Zagrebu. Nakon završetka osnovne škole, upisujem V. gimnaziju, prirodoslovno-matematičkog smjera, također u Zagrebu. Godine 2006. upisujem preddiplomski studij Matematika na Prirodoslovno-matematičkom fakultetu. Nakon četiri godine, prebacujem se na nastavnički smjer (Matematika; smjer: nastavnički) te 2013. godine završavam preddiplomski studij. Iste godine upisujem diplomski studij Računarstvo i matematika, također na Matematičkom odsjeku PMF-a.