

Paketna obrada podataka

Hadoop, HDFS, MapReduce, Apache Spark

Arhitekture sistema velikih skupova podataka, dr Vladimir Dimitrieski

1

Sadržaj

- Osnovi distribuirane obrade podataka
- Distribuirani sistemi datoteka (HDFS)
- Distribuirana obrada podataka (MapReduce)
- Apache Spark (paketna obrada)
- Pomoćni alati

2

2

1

Osnovi distribuirane obrade podataka

3

Osnovi distribuirane obrade podataka

- Postoji potreba za skladištenjem i obradom velike količine podataka
 - kapacitet skladišta raste zadovoljavajućom brzinom
 - **ali brzina pristupa podacima ne prati trend** povećanja količine podataka
 - imamo velika ali spora skladišta podataka
 - potrebno je obraditi podatke u odgovarajućem vremenu
 - jedno rešenje je **paralelizacija čitanja i pisanja podataka** (horizontalno skaliranje)
 - kompleksno rešenje sa stanovišta implementacije i eksploracije
 - drugo rešenje je pronalaženje i korišćenje novih uređaja koji omogućavaju bolje performanse (vertikalno skaliranje)
 - skupo rešenje

4

4

Osnovi distribuirane obrade podataka

- Potreba za skladištenjem i obradom velike količine podataka
 - rešenje: paralelizacija čitanja i pisanja podataka
 - skladištenje podataka na više diskova istovremeno
 - u distribuiranom sistemu datoteka
 - veliko ubrzanje u pristupu podacima u odnosu na nedistribuirani sistem
 - izazovi
 - otkaz pojedinačnih diskova
 - uvođenje redundancije replikacijom podataka, npr. RAID
 - kompleksno upravljanje infrastrukturom (mrežom)
 - paralelna obrada podataka
 - zahtev da analiza podataka obuhvati podatke sa više diskova istovremeno
 - potreban mehanizam koji apstrahuje sinhronizaciju i čitanje podataka u distribuiranom okruženju, npr. MapReduce
 - izazov - što bolje iskorišćenje resursa distribuiranog sistema

5

5

Distribuirani sistemi datoteka

- Distribuirani sistem datoteka (engl. *distributed file system*)
 - **sistem datoteka** obuhvata strukture i metode za upravljanje datotekama i direktorijumima
 - implementiran od strane operativnog sistema
 - **distribuirani sistem datoteka** upravlja datotekama i direktorijumima smeštenim na više od jednog računara
 - koristi se kada količina podataka preraste mogućnosti skladištenja na jednom računaru
 - kompleksniji za implementaciju i održavanje

6

6

Distribuirani sistem datoteka

- Distribuirani sistem datoteka
 - **sistem datoteka** čiji su klijenti, serveri i skladišni uređaji raspoređeni na više računara u okviru distribuiranog sistema
 - **servis** - softver koji se izvršava na jednom ili više računara i pruža određeni tip funkcionalnosti klijentima koji nisu unapred poznati
 - **server** - računar na kojem se izvršava servis
 - **klijent** - proces koji komunicira sa servisom
 - koristeći skup operacija koje čine **interfejs klijenta**
 - klijent sistema datoteka ima proste operacije za rad sa datotekama u interfejsu
 - kreiranje, brisanje, čitanje i pisanje

7

7

Distribuirani sistem datoteka

- Distribuirani sistem datoteka - očekivane karakteristike
 - interfejs klijenta bi trebalo da bude transparentan
 - tj. da apstrahuje lokaciju datoteke i da se ne vidi eksplicitno razlika između lokalnih i udaljenih datoteka
 - performanse distribuiranog sistema datoteka bi trebalo da budu uporedive sa centralizovanim sistemom datoteka
 - čak i u prisustvu dodatnog mrežnog saobraćaja
 - distribuirani sistem datoteka treba da omogući linearno skaliranje performansi sa povećanjem broja čvorova
 - ili makar približno linearno skaliranje

8

8

Distribuirani sistem datoteka

- Distribuirani sistemi datoteka
 - Andrew File System (AFS)
 - distribuirani sistem datoteka kod kojeg je osnovni cilj da pruži što veću mogućnost skaliranja
 - male datoteke koje je moguće keširati i koje se ne dele među korisnicima
 - učestanost čitanja je mnogo veća od učestanosti pisanja
 - preovladava sekvencijalno čitanje podataka
 - Network File System (NFS)
 - razvijen od strane kompanije *Sun Microsystems*
 - omogućava da se direktorijumi logički jedinstvenog sistema datoteka, fizički nalaze na različitim mašinama
 - autonomnost klijenata i servera u arhitekturi
 - Hadoop File System (HDFS)
 - često korišćeni distribuirani sistem datoteka
 - deo alata Hadoop

9

9

Distribuirana obrada podataka

- Tipovi obrade i izvršenja programa
 - konkurentna obrada
 - obrada od strane više procesa nad istim vremenskim intervalom
 - svaki od procesa dobije deo procesorskog vremena
 - obrada nije paralelna
 - ne postoji koordinacija između procesa
 - ne postoji komunikacija između procesa
 - paralelna obrada
 - obrada od strane više procesa istovremeno
 - zahteva postojanje više jezgara procesora
 - ne postoji koordinacija niti komunikacija između procesa
 - distribuirana obrada
 - obrada od strane više procesa koji međusobno komuniciraju radi izvršavanja obrade
 - najčešće se procesi izvršavaju na različitim procesorima/računarima

10

10

Distribuirana obrada podataka

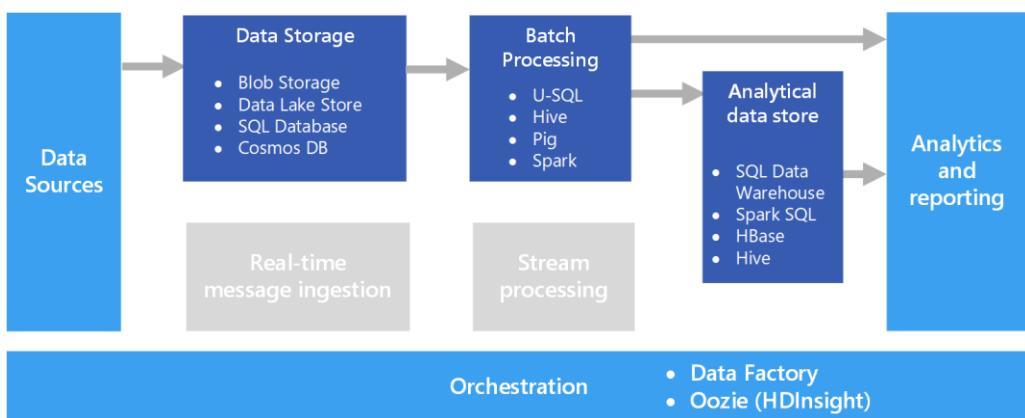
- Paketna obrada podataka (engl. *batch processing*)
 - **distribuirana obrada velike količine podataka smeštenih u distribuiranom sistemu datoteka**
 - prilikom obrade podataka čitaju se isključivo paketi podataka
 - paketi su iste, unapred podešene veličine
 - pisanje podataka u distribuirani sistem datoteka se najčešće obavlja za međurezultate koraka obrade
 - može se koristiti paradigma obrade podataka MapReduce
 - ili alternativni model distribuirane obrade podataka: Spark, Hama, Giraph, MPI, Tez
 - ili neki od alata koji povišuju stepen apstrakcije: Pig, Hive, Presto

11

11

Paketna obrada podataka

- Paketna obrada podataka



izvor: *Big data architectures*, Microsoft, <https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/batch-processing>

12

12

Hadoop

- Hadoop je distribuirani sistem za skladištenje i analizu podataka koji obuhvata
 - **HDFS** za skladištenje podataka
 - **MapReduce** za obradu podataka
- Hadoop - istorija
 - kreator Doug Cutting
 - kreator Apache Lucene biblioteke
 - bivši radnik kompanije Yahoo!
 - kreiran za potrebe web pretraživača
 - 2002, inicijalno kao biblioteka Apache Nutch koja je deo projekta Apache Lucene
 - NDFS i MapReduce
 - 2008. kao samostalan projekat Apache Hadoop
 - obuhvata MapReduce i HDFS
 - Hadoop v2 podržava i druge paradigme za distribuiranu obradu podataka
 - obuhvata i veliki broj drugih alata i biblioteka

13

13

Distribuirani sistem datoteka (HDFS)



14

Distribuirani sistem datoteka Hadoop (HDFS)

- Distribuirani sistem datoteka Hadoop (engl. *Hadoop Distributed File System*, HDFS)
 - sistem datoteka implementiran u okviru alata Hadoop
 - koristi se Hadoop I/O apstrakcija nad sistemom datoteka
 - tj. implementacioni detalji HDFS-a skriveni od krajnjeg korisnika
 - nije obavezno koristiti HDFS sa Hadoop-om
 - Hadoop podržava i druge distribuirane sisteme datoteka
 - ali se najčešće koristi HDFS

15

15

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osobine
 - **HDFS je distribuirani sistem datoteka projektovan da omogući skladištenje izuzetno velikih datoteka, pruži optimizovane metode za čitanje podataka, i da radi na klasteru napravljenom od regularnog hardvera**
 - **izuzetno velike datoteke**
 - u ovom kontekstu, datoteke veličine od nekoliko stotina megabajta do nekoliko terabajta
 - postoje Hadoop klasteri koji skladište i nekoliko petabajta podataka
 - npr. Yahoo! klaster
 - **optimizovane metode za čitanje podataka**
 - šablon upravljanja podacima: piši jednom, čitaj više puta
 - podaci su generisani ili kopirani iz nekog izvora
 - obrade nad podacima se ponavljaju više puta
 - svaka obrada obuhvata čitanje (skoro) celokupnog skupa podataka
 - **vreme čitanja celog skupa je bitnije od čitanja pojedinačnog podatka**

16

16

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osobine
 - regularni hardver (engl. *commodity hardware*)
 - hardver koji je napravljen za svakodnevnu upotrebu u različitim domenima primene
 - standardni delovi svakog personalnog računara
 - napravljen od strane različitih proizvođača

17

17

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osobine
 - HDFS nije pogodan za obradu u sledećim slučajevima
 - pristup podacima sa niskom latencijom
 - engl. *low-latency data access*
 - HDFS je projektovan za prenos i čitanje velike količine podataka i zbog toga nije pogodan za pristup sa niskom latencijom
 - bolje rešenje je korišćenje **HBase** baze podataka
 - postojanje velikog broja malih datoteka
 - usled potrebnog indeksiranja i održavanja liste datoteka u sistemu
 - postojanje višestrukih programa koji pišu podatke i izmena slučajnog sloga datoteke
 - HDFS podržava postojanje samo jednog programa koji piše u datoteku
 - podaci se uvek upisuju na kraj datoteke

18

18

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **blok** (engl. *block*)
 - osnovna jedinica manipulacije podacima u HDFS-u
 - ukoliko nije izmenjena, veličina bloka je 64MB
 - blok je namerno napravljen da bude veliki zbog prirode obrade koja se izvršava nad HDFS sistemom datoteka
 - potreba za čitanjem, traženjem, i razmenom velike količine podataka
 - **datoteke su podeljene u blokove fiksne veličine koji su smešteni nezavisno jedan od drugog**
 - moguće je da su smešteni i na različitim računarima u klasteru
 - fiksna veličina blokova omogućava procenu kapaciteta, lakšu replikaciju i jednostavnije upravljanje memorijom
 - ukoliko je datoteka manja od veličine bloka ona ne zauzima celi blok na disku
 - kao što je to slučaj sa uobičajenim sistemima datoteka

19

19

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **imenski čvorovi** (engl. *namenodes*) i **čvorovi sa podacima** (engl. *datanodes*)
 - dve vrste čvorova u HDFS klasteru
 - **imenski čvorovi**
 - sadrži registre svih datoteka koji se nalaze u sistemu datoteka
 - meta-podaci o svim datotekama i direktorijumima
 - trajno uskladišteni na imenskom čvoru
 - sastoji se od **slike imenskog prostora** (engl. *namespace image*) i **logova izmena** (engl. *edit logs*)
 - sadrže lokacije blokova svake datoteke
 - ne skladište se trajno na disku, već svi čvorovi sa podacima prijavljuju blokove prilikom svakog pokretanja sistema
 - **čvorovi sa podacima**
 - čvorovi na kojima se skladište blokovi sa podacima
 - po zahtevu imenskog čvora, čitaju i pišu blokove u trajnu memoriju

20

20

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **imenski čvorovi i čvorovi sa podacima**
 - bez imenskog čvora HDFS ne bi mogao da funkcioniše
 - gubitak računara sa imenskim čvorom bi značio gubitak svih datoteka
 - ne postoji mogućnost rekonstruisanja registra pomoću čvorova sa podacima
 - neophodno je obezbediti da persistenti deo imenskog čvora bude što je moguće otporniji na otkaze
 - način 1: paralelno zapisivanje meta-podataka na lokalni i udaljeni sistem datoteka (upis podataka je atomička i sinhrona operacija)
 - način 2: postojanje sekundarnog imenskog čvora
 - nema ulogu imenskog čvora u arhitekturi već isključivo radi periodično prebacivanje zapisa iz loga izmena u sliku imenskog prostora
 - posebna mašina jer prebacivanje zapisa zahteva dosta vremena CPU-a
 - u slučaju otkaza gubitak podataka je siguran jer sekundarni čvor ima starije podatke od imenskog čvora

21

21

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **klijenti** (engl. *clients*)
 - softverske komponente koje pristupaju podacima ili koriste usluge HDFS-a
 - mogu biti klijentski programi izvan Hadoop ekosistema
 - mogu biti MapReduce programi ili druge komponente Hadoop ekosistema

22

22

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **HDFS federacija** (engl. *HDFS federation*)
 - dozvoljava skaliranje klastera dodavanjem novih imenskih čvorova
 - svaki imenski čvor se brine o posebnom delu sistema datoteka
 - imenski čvorovi ne komuniciraju međusobno
 - otkaz jednog čvora ne dovodi do otkaza drugih imenskih čvorova
 - skladište blokova **nije particionisano** i deljeno je od strane svih imenskih čvorova
 - npr. /user i /share mogu biti u nadležnosti različitih imenskih čvorova
 - porast broja datoteka dovodi do porasta broja zapisa u perzistentnom delu imenskog čvora
 - memorija imenskog čvora postaje ograničavajući faktor

23

23

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **visok nivo dostupnosti HDFS-a**
 - čak i u slučaju paralelnog pisanja meta-podataka i postojanja sekundarnog imenskog čvora, imenski čvor predstavlja usko grlo sistema
 - otkazom imenskog čvora ceo sistem otkazuje dok administrator ne pokrene novi imenski čvor
 - što može i da potraje do 30 minuta na većim klasterima
 - dugotrajno pokretanje imenskog čvora ima posledice na **trajanje održavanja sistema**
 - u praksi bitniji slučaj usled male verovatnoće otkaza imenskog čvora
 - potrebno omogućiti postojanje paralelnih imenskih čvorova
 - jedan u aktivnom stanju a drugi u stanju pripravnosti

24

24

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **visok nivo dostupnosti HDFS-a**
 - neophodno je da imenski čvorovi dele log izmena
 - kako bi mogli uvek rekonstruisati sliku imenskog prostora
 - svi čvorovi sa podacima šalju odgovore i izveštaje svim imenskim čvorovima
 - HDFS klijenti moraju biti konfigurisani sa parametrima koji im omogućavaju da vide oba imenska čvora i u slučaju okaza izvršće preključivanje
 - u slučaju otkaza aktivnog čvora, čvor u pripravnosti postaje aktivni imenski čvor
 - nakon nekoliko desetina sekundi
 - usled posedovanja poslednje slike imenskog prostora
 - u slučaju otkaza oba čvora
 - potrebno je podići novi imenski čvor od nule
 - svodi se na slučaj bez paralelnih imenskih čvorova

25

25

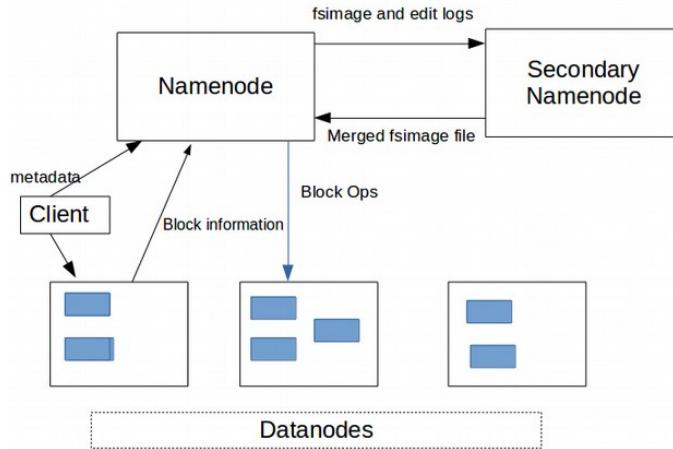
Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **visok nivo dostupnosti HDFS-a**
 - **preuzimanje posla** (engl. *failover*)
 - može biti inicirano ručno, od strane administratora, ili automatski u slučaju otkaza aktivnog imenskog čvora
 - prebacivanje sa aktivnog na čvor u pripravnosti je zaduženje **kontrolera za preuzimanje posla** (engl. *failover controller*)
 - **ogradijanje** (engl. *fencing*)
 - u slučaju automatskog preuzimanja posla potrebno je osigurati se da je aktivni imenski čvor otkazao
 - a ne samo da nije privremeno dostupan zbog mrežnih problema
 - jer vraćanje aktivnog imenskog čvora u sistem koji nije svestan izmene statusa može imati velike posledice po konzistentnost sistema
 - čvoru u otkazu su uklonjena sva prava pristupa čvorovima sa podacima a procesu je poslat signal za zaustavljanje i prestanak rada sa mrežom

26

26

Distribuirani sistem datoteka Hadoop (HDFS)



27

27

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovne naredbe
 - hadoop fs <args>
 - hadoop fs -help
 - za potpun spisak naredbi pogledati <https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html>

```
% hadoop fs -copyFromLocal input/docs/test.txt hdfs://localhost/user/vdimitrieski/test.txt
% hadoop fs -copyToLocal hdfs://localhost/user/vdimitrieski/test.txt test.copy.txt
% md5 input/docs/test.txt test.copy.txt
MD5 (input/docs/test.txt) = a16f231da6b05e2ba7a339320e7dacd9
MD5 (test.copy.txt) = a16f231da6b05e2ba7a339320e7dacd9

% hadoop fs -mkdir books
% hadoop fs -ls .
Found 2 items
drwxr-xr-x - tom supergroup 0 2009-04-02 22:41 /user/vdimitrieski/books
-rw-r--r-- 1 tom supergroup 118 2009-04-02 22:29 /user/vdimitrieski/test.txt
```

28

28

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **Hadoop apstrakcija sistema datoteka**
 - HDFS je samo jedna implementacija tzv. Hadoop sistema datoteka
 - moguće vršiti obradu nad bilo kojim sistemom datoteka
 - preporuka da se to radi samo nad sistemima koji podržavaju lokalizaciju obrade podataka
 - kao što je to slučaj sa HDFS-om

29

29

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **Hadoop apstrakcija sistema datoteka**

Sistem datoteka	URI schema	Java Implementacija	Opis
Local	<i>file</i>	fs.LocalFileSystem	lokalni sistem datoteka
HDFS	<i>hdfs</i>	hdfs.DistributedFileSystem	HDFS
HFTP	<i>hftp</i>	hdfs.HftpFileSystem	omogućava samo čitanje iz HDFS sistema datoteka preko HTTP protokola
HSFTP	<i>hsftp</i>	hdfs.HsftpFileSystem	omogućava samo čitanje iz HDFS sistema datoteka preko HTTPS protokola
WebHDFS	<i>webhdfs</i>	hdfs.web.WebHdfsFileSystem	čitanje i pisanje u HDFS-u preko HTTP protokola. Zamena za HFTP i HSFTP

30

30

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **Hadoop apstrakcija sistema datoteka**

Sistem datoteka	URI schema	Java Implementacija	Opis
HAR	<i>har</i>	fs.HarFileSystem	sistem za arhiviranje datoteka koji se implementira kao poseban sloj nad HDFS-om
KFS (CloudStore)	<i>kfs</i>	fs.kfs.KosmosFileSystem	CloudStore sistem datoteka (Google, C++)
FTP	<i>ftp</i>	fs.ftp.FTPFileSystem	sistem datoteka koji obuhvata FTP server
S3 (native)	<i>s3n</i>	fs.s3native.NativeS3FileSystem	sistem datoteka koji obuhvata Amazon S3 servis
S3 (block based)	<i>s3</i>	fs.s3.S3FileSystem	sistem datoteka koji obuhvata Amazon S3 servis ali smešta datoteke u blokovima kako bi se prevazišlo ograničenje od 5GB po datoteci

31

31

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - osnovni koncepti
 - **Hadoop apstrakcija sistema datoteka**

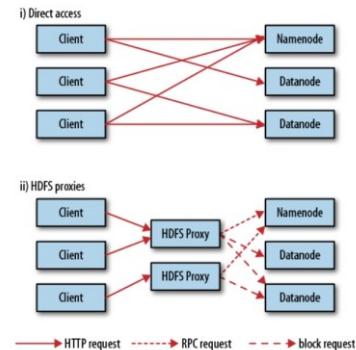
Sistem datoteka	URI schema	Java Implementacija	Opis
Distributed RAID	<i>hdfs</i>	hdfs.DistributedRaidFileSystem	sistem za arhiviranje datoteka koji se implementira kao HDFS sa podrškom za RAID
View	<i>viewfs</i>	viewfs.ViewFileSystem	Obuhvata tabelu povezanih skladišta koja je implementirana na klijentskoj strani. Uobičajno korišćena za kreiranje HDFS federacije

32

32

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - pristup čvorovima
 - pristup preko HTTP protokola
 - direktan pristup
 - ugrađeni web server u imenskom čvoru
 - indirektni pristup
 - preko HTTP Proxy-ja
 - pristup iz programskih jezika
 - Java, C
 - pristup iz bilo kog Unix OS-a
 - HDFS može biti zakačen kao bilo koji disk u UNIX sistemu

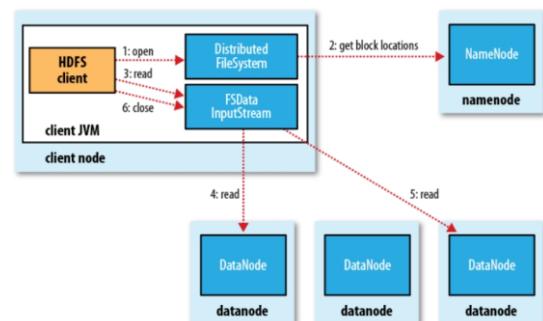
izvor: *Hadoop - The Definitive Guide, Tom White*

33

33

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - čitanje podataka
 - (1) otvaranje datoteke
 - (2) dobijanje lokacija blokova
 - imenski čvor vraća **najблиžу** lokaciju sa blokom
 - (3-5) čitanje blok po blok
 - u redosledu u kojem blokovi sačinjavaju datoteku
 - u slučaju nemogućnosti komunikacije sa čvorom sa podacima
 - bira se **najблиžи** alternativni čvor koji sadrži željeni blok

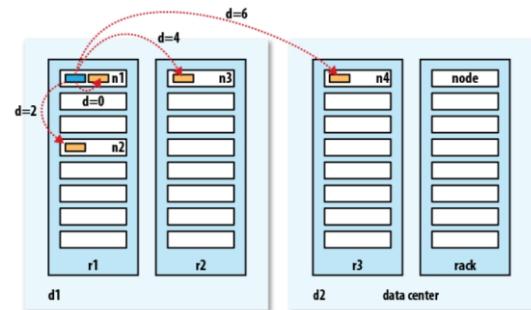
izvor: *Hadoop - The Definitive Guide, Tom White*

34

34

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - mrežna topologija
 - blizina čvorova
 - zapravo predstavlja brzinu i količinu podataka koja može biti razmenjena
 - u Hadoopu postoji stablo čvorova
 - distanca predstavlja sumu udaljenosti čvorova do zajedničkog pretka
 - $\text{distance}(/d1/r1/n1, /d1/r1/n1) = 0$
 - $\text{distance}(/d1/r1/n1, /d1/r1/n2) = 2$
 - $\text{distance}(/d1/r1/n1, /d1/r2/n3) = 4$
 - $\text{distance}(/d1/r1/n1, /d2/r3/n4) = 6$



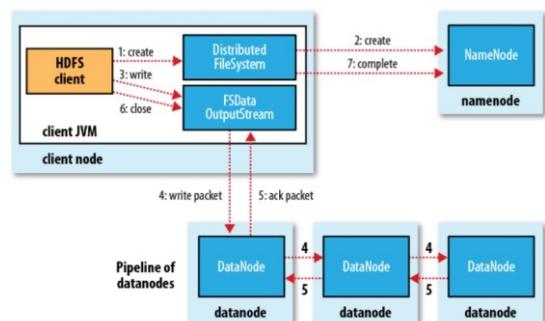
izvor: Hadoop - The Definitive Guide, Tom White

35

35

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - pisanje podataka
 - (1) iniciranje kreiranja datoteke
 - (2) kreiranje datoteke u imenskom čvoru
 - (3-5) slanje podataka za upis
 - podaci se dele na pakete i pored upisa repliciraju se na dodatne čvorove sa podacima
 - čvorovi obrazuju **lanac replikacije**
 - engl. *replication pipeline*
 - (6) nakon upisa zatvara se datoteka
 - (7) imenskom čvoru se šalje signal za završetak upisa



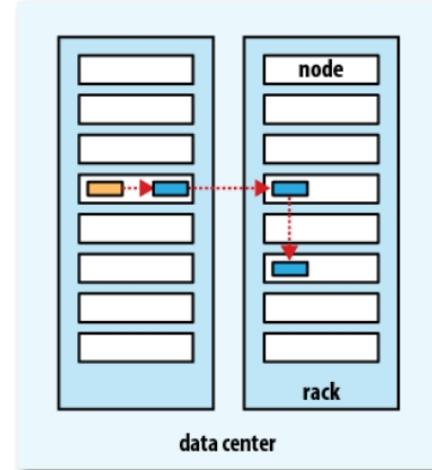
izvor: Hadoop - The Definitive Guide, Tom White

36

36

Distribuirani sistem datoteka Hadoop (HDFS)

- HDFS - pisanje podataka
 - odabir čvora za skladištenje replike zavisi od nekoliko faktora
 - brzina upisa i količina prenetih podataka
 - replike je potrebno postaviti što bliže kako bi se povećala brzina upisa podataka
 - otpornost na greške
 - replike je potrebno postaviti što dalje kako bi se smanjila mogućnost potpunog otkaza
 - Hadoop strategija
 - prva replika u isti čvor kao i original
 - druga replika se postavlja na drugi rack
 - treća replika na isti rack kao i druga ali u drugi slučajno odabrani čvor



izvor: *Hadoop - The Definitive Guide*, Tom White

37

37

Hadoop I/O

- Hadoop I/O koncepti i primitive
 - predstavljaju softverske elemente koji koriste usluge HDFS-a
 - i nalaze se na višem nivou apstrakcije
 - koncepti koji su od posebnog značaja pri radu sa velikim količinama podataka i velikim datotekama od više terabajta
 - koncepti nezavisni od Hadoop-a
 - **integritet podataka** (engl. *data integrity*)
 - **kompresija** (engl. *compression*)
 - koncepti sa specifičnom Hadoop implementacijom
 - **radni okviri za serijalizaciju** (engl. *serialization frameworks*)
 - **strukture podataka na disku** (engl. *on-disk data structures*)

38

38

Hadoop I/O

- **Integritet podataka**

- pisanje podataka u sistem datoteka nosi sa sobom rizik od pojave grešaka i oštećenih podataka
 - što je veća količina upisanih podataka i rizik je veći
- računanje kontrolne sume (engl. *checksum*)
 - uobičajeni način za identifikaciju oštećenja u snimljenim podacima
 - detekcija ali ne i oporavak od grešaka
 - često korišćen algoritam CRC-32 (engl. *Cyclic Redundancy Check*)
 - sračunava 32 bitni integer na osnovu ulaznih podataka bilo koje veličine
 - računa se prvi put kada se podaci snime u sistem datoteka
 - računa se svaki naredni put kada se podaci čitaju

39

39

Hadoop I/O

- Integritet podataka - HDFS

- u HDFS-u se kontrolna suma izračunava u pozadini, bez eksplicitnog zahtevanja od strane korisnika
 - prilikom svakog upisa podataka
 - zbog male veličine podataka koji predstavljaju kontrolnu sumu, **manje od 1%** podataka obuhvata kontrolne sume
 - čvorovi sa podacima su zaduženi za proveru podataka koje skladište
 - bilo da su podaci direktno upisani ili primljeni usled replikacije
 - čvor koji je poslednji u replikacionom lancu proverava kontrolnu sumu
 - čvorovi čuvaju istoriju provere kontrolne sume za svaki podatak
 - izuzetno bitno za detekciju kvarova na disku
- kontrolna suma se proverava **prilikom svakog čitanja podataka**
 - klijenti koji čitaju podatke proveravaju kontrolnu sumu
 - poredeći je sa kontrolnom sumom skladištenom na čvoru sa podacima

40

40

Hadoop I/O

- Integritet podataka - HDFS
 - u HDFS-u moguća je ispravka grešaka u podacima
 - povlačenjem jedne od replikacionih kopija sa drugih čvorova sa podacima
 - ne prevlači se na čvor na kojem je otkrivena greška već se podaci repliciraju na novi čvor (ili čvorove) da bi predefinisani broj validnih replika bio ponovo tačan
 - originalni oštećeni blok sa podacima se briše iz čvora
 - ažuriraju se zapisi u imenskom čvoru
 - moguće je isključiti proveru kontrolne sume prilikom čitanja podataka
 - korisno ukoliko imamo oštećene blokove i želimo da vidimo šta se može spasiti od podataka pre nego što se blok obriše

41

41

Hadoop I/O

- Kompresija
 - dve glavne prednosti korišćenja kompresije
 - omogućava da se više datoteka smesti na manjem skladišnom prostoru
 - ubrzava razmenu podataka preko mreže kao i razmenu podataka sa diskom
 - što je veća količina podataka koja se skladišti izraženiji su benefiti koje kompresija donosi
 - stoga izuzetno bitan element svakog (distribuiranog) sistema datoteka
 - postoji veliki broj formata, alata i algoritama za kompresiju
 - veliki broj može biti korišćen sa Hadoop-om
 - različiti kompresioni formati imaju različit odnos vremena potrebnog za kompresiju/dekompresiju i uštede prostora

42

42

Hadoop I/O

- Kompresija - formati

Compression format	Tool	Algorithm	Filename extension	Splittable?
DEFLATE ^a	N/A	DEFLATE	.deflate	No
gzip	gzip	DEFLATE	.gz	No
bzip2	bzip2	bzip2	.bz2	Yes
LZO	lzop	LZO	.lzo	No ^b
LZ4	N/A	LZ4	.lz4	No
Snappy	N/A	Snappy	.snappy	No

43

43

Hadoop I/O

- Kompresija - Hadoop kodeci
 - Kodek (engl. codec)
 - skraćeno od kompresija-dekompresija
 - predstavljaju implementaciju algoritama za kompresiju i dekompresiju

Compression format	Hadoop CompressionCodec
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
LZ4	org.apache.hadoop.io.compress.Lz4Codec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

44

44

Hadoop I/O

- Kompresija i MapReduce paketi
 - podrška kompresionog formata za MapReduce se ogleda u podršci za razdvajanje kompresovane datoteke
 - razdvajanje podrazumeva podelu kompresovane datoteke na delove kod kojih je moguće dekompresovati svaki deo pojedinačno
 - razdvajanje omogućava da MapReduce čita pojedine kompresovane blokove
 - bez ponovnog kreiranja cele datoteke
 - u slučaju nedostatka podrške razdvajajuju
 - MapReduce zadatak mora dobiti sve blokove iz svih čvorova sa podacima
 - koji ne moraju biti smešteni isti čvor na kojem se izvršava zadatak niti moraju biti u istom rack-u

45

45

Hadoop I/O

- Kompresija - uputstva (sortirana po efikasnosti)
 - koristiti strukture podataka na disku koje omogućavaju kompresiju i razdvajanje i zatim iskoristiti neki od bržih kompresionih algoritama
 - npr. LZO, LZ4, Snappy
 - koristiti kompresioni format koji podržava razdvajanje
 - npr. bzip2 - inače spor algoritam
 - neophodno za velike fajlove
 - jer su inače MapReduce zadaci neefikasni
 - aplikativno podeliti datoteke na delove koji se mogu nezavisno kompresovati
 - kompresovani delovi bi trebalo da su veličine HDFS bloka
 - ne koristiti kompresiju

46

46

Hadoop I/O

- **Serijalizacija**

- serijalizacija je proces pretvaranja strukturiranih objekata u nizove bajtova spremnih za slanje preko mreže ili skladištenje na disku
 - obrnut proces, pretvarjanje nizova bajtova u strukturirane objekte naziva se deserijalizacija
- serijalizacija se koristi u dva dela procesa obrade podataka
 - međuprocesna komunikacija
 - Hadoop koristi pristup pozivanja udaljenih metoda
 - engl. *Remote Procedure Calls, RPC*
 - skladištenje podataka

47

47

Hadoop I/O

- Serijalizacija

- poželjne karakteristike serijalizacije
 - kompaktnost serijalizovanih podataka
 - korišćenje kompaktnog formata kako bi se najefikasnije iskoristio disk ili mrežni saobraćaj
 - brza serijalizacija/deserijalizacija
 - poželjno da sam proces serijalizacije/deserijalizacije traje što kraće
 - proširivost procesa serijalizacije
 - novim koracima i novim algoritmima
 - interoperabilnost
 - između različitih programskih jezika i hardverskih sistema

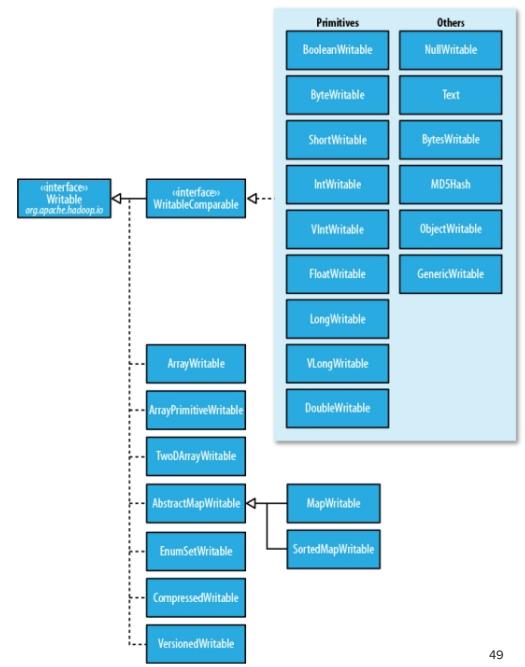
48

48

Hadoop I/O

- Serijalizacija - Hadoop
 - Hadoop serijalizacioni format - **Writable** interfejs
 - kojeg nasleđuju sve konkretne klase čije objekte je moguće serijalizovati
 - moguće kreirati svoju klasu koja implementira interfejs

```
import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;
public interface Writable {
    void write(DataOutput out) throws IOException;
    void readFields(DataInput in) throws IOException;
}
```



izvor: *Hadoop - The Definitive Guide*, Tom White

49

Hadoop I/O

- Radni okviri za serijalizaciju
 - iako MapReduce podržava Writable implementaciju sa radnim okvirom **WritableSerialization**
 - nije neophodno koristiti Hadoop-ov sistem za serijalizaciju
 - moguće je koristiti bilo koji drugi radni okvir
 - **JavaSerialization** - podržava standardnu Java serijalizaciju objekata što omogućava korišćenje standardnih Java tipova podataka
 - ne zadovoljava poželjne karakteristike serijalizacionih formata
 - stoga nije često korišćena
 - **Apache Avro** - serijalizacioni format koji je nezavisan od programskog jezika
 - omogućava portabilnost između programskih jezika
 - moguće je serijalizovati podatke u binarni ili JSON oblik

50

50

Hadoop I/O

- **Strukture podataka na disku**

- za neke primene potrebno je podatke organizovati u strukture podataka na disku
 - umesto čuvanja u nestrukturiranim datotekama
 - omogućava bolje skaliranje podataka
- Hadoop podržava nekoliko tipova struktura podataka
 - **SequenceFile** - sekvencijalna datoteka
 - **MapFile** - indeks-sekvencijalna datoteka

51

51

Hadoop I/O

- Strukture podataka na disku - SequenceFile

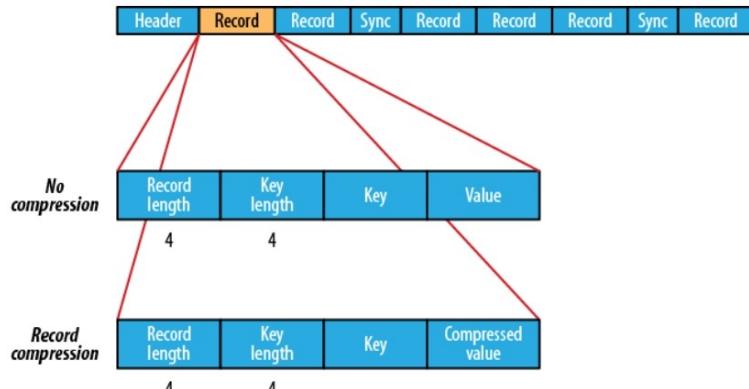
- svaki red u datoteci predstavlja jedan zapis
 - par ključ/vrednost
- jedna od upotreba jeste smeštanje malih datoteka u veće sekvencijalno organizovane datoteke
 - na taj način se bolje koriste prednosti sistema Hadoop
- spajanje i sortiranje sekvencijalno organizovanih datoteka
 - se najčešće radi pomoću MapReduce programa

52

52

Hadoop I/O

- Strukture podataka na disku - SequenceFile



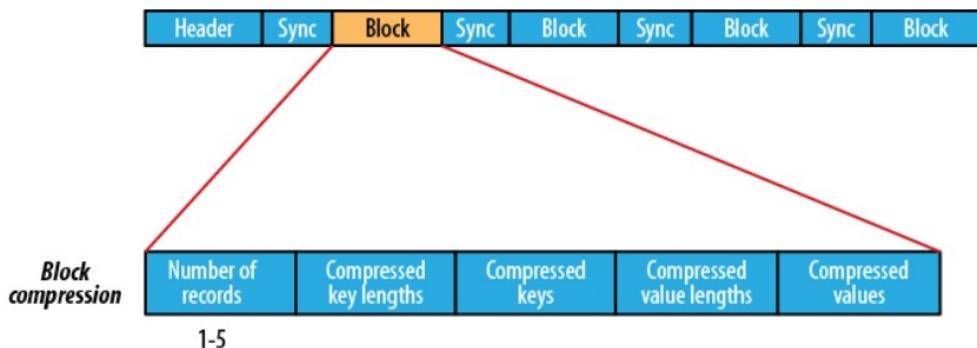
izvor: Hadoop - The Definitive Guide, Tom White

53

53

Hadoop I/O

- Strukture podataka na disku - SequenceFile sa kompresijom blokova



izvor: Hadoop - The Definitive Guide, Tom White

54

54

Hadoop I/O

- Strukture podataka na disku - MapFile
 - indeks-sekvencijalna datoteka
 - obuhvata po ključu sortirane slogove
 - sadrži indeks nad ključem da bi omogućila direktni pristup slogovima
 - postoje različite varijante
 - **SetFile** - sadrži skup ključeva koji implementiraju Writeable interfejs
 - ključevi moraju biti dodati u sortiranom redosledu
 - **ArrayFile** - ključ je integer a vrednost implementira Writeable interfejs
 - **BloomMapFile** - poseduje veoma brzu get() metodu
 - koristi *bloom* filter za utvrđivanje da li je ključ u datoteci ili ne
 - probabilistički filter
 - https://en.wikipedia.org/wiki/Bloom_filter

55

55

Distribuirana obrada podataka (MapReduce)

56

Obrada podataka - primer

- Analiza meteoroloških podataka
 - National Climatic Data Center (NCDC, <http://www.ncdc.noaa.gov/>)*
 - svako merenje je tekstualna datoteka u kojoj je svaki red novi zapis o meteorološkim parametrima
 - organizovane po datumu i stanicu na kojoj je merenje izvršeno
 - podaci su grupisani po godini zbog lakše obrade
 - postoji struktura koji podaci prate
 - sa mnogo opcionih delova

```

0057
332130      # USAF weather station identifier
99999       # WBAN weather station identifier
19500101    # observation date
0300        # observation time
4
+51317      # latitude (degrees x 1000)
+028783     # longitude (degrees x 1000)
FM-12
+0171        # elevation (meters)
99999
V020
320          # wind direction (degrees)
1            # quality code
N
0072
1
00450        # sky ceiling height (meters)
1            # quality code
CN
010000      # visibility distance (meters)
1            # quality code
N9
-0128        # air temperature (degrees Celsius
x 10)
1            # quality code
-0139        # dew point temperature (degrees
Celsius x 10)
1            # quality code
10268       # atmospheric pressure
(hectopascals x 10)
1            # quality code

```

57

Obrada podataka - primer

- Analiza meteoroloških podataka - Unix alati
 - upit za pronalaženje najviše godišnje temperature za svaku godinu u skupu podataka
 - moguće izvršiti pomoću standardnih Unix programa i alata
 - na nedistribuiranom sistemu
 - sa porastom količine podataka raste i vreme izvršavanja skripte

```

#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ &&
               temp > max) max =
               temp }
         END { print max }'
done

## test run ##
% ./max_temperature.sh
1901 317
1902 244
1903 289
1904 256
1905 283
...

```

58

58

Obrada podataka - primer

- Analiza meteoroloških podataka - Unix alati
 - potreban mehanizam za paralelno izvršavanje skripte
 - idealno, podelili bismo skup podataka po godini i pokrenuli skriptu više puta
 - svaki od procesa bi se izvršavao paralelno (paralelna obrada)
 - postoje tri problema sa ovim pristupom
 - ujednačena podela inicijalnog skupa je teška
 - različite godine imaju različit broj merenja i samim tim različite količine podataka
 - vreme obrade podataka i opterećenje resursa zavisi od najveće datoteke
 - moguće deliti i po veličini podataka a ne po godini
 - objedinjavanje rezultata obrade može zahtevati dodatnu obradu
 - potrebna koordinacije procesa ukoliko ne delimo po godini već po veličini
 - još uvek smo ograničeni kapacitetima jednog računara
 - na kojem se izvršavaju procesi
 - korišćenje više računara otvara nove probleme
 - mahom u domenu koordinacije procesa i obezbeđivanja pouzdanosti

59

59

MapReduce

- MapReduce
 - predstavlja pristup pisanju programa za obradu podataka
 - jednostavan a ekspresivan pristup
 - ne zavisi od konkretnog programskog jezika
 - inherentno podržava paralelizam u izvršavanju napisanih programa
 - obrada podataka je implementirana kroz dva različita tipa operatora
 - **operatori map i reduce**
 - ulazni i izlazni podaci su predstavljeni kroz parove ključ-vrednost
 - čiju semantiku i značenje bira sam programer
 - potpis funkcija map i reduce
 - map: $(K1, V1) \rightarrow list(K2, V2)$
 - reduce: $(K2, list(V2)) \rightarrow list(K3, V3)$

60

60

MapReduce - primer

- Analize meteoroloških podataka - MapReduce
 - ulaz predstavljaju redovi iz datoteka za koje je ključ redni broj reda
 - ključ nije relevantan za ovaj slučaj
 - operator *map* služi za izvlačenje relevantnih podataka
 - godina i temperatura
 - operator *map* služi i za uklanjanje nepotpunih podataka

```
#input data
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)

#output data
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

61

61

MapReduce - primer

- Analize meteoroloških podataka - MapReduce
 - MapReduce okruženje grupiše rezultate izvršenja operatora *map* po ključu
 - pre slanja operatoru *reduce*
 - operator *reduce* kao ulaz dobija listu temperaturna za neku godinu
 - prolazeći kroz listu pronađi najvišu temperaturu

```
#grouped data / reducer input
(1949, [111, 78])
(1950, [0, 22, -11])

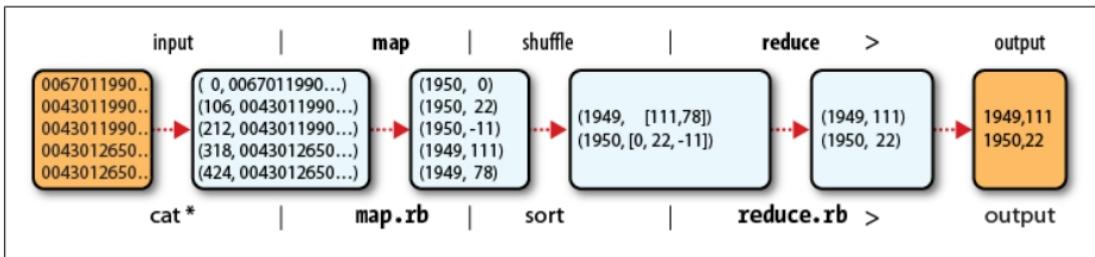
#final output data / reducer output
(1949, 111)
(1950, 22)
```

62

62

MapReduce - primer

- Analize meteoroloških podataka - MapReduce
 - dijagram aktivnosti prethodno opisanog programa MapReduce



izvor: *Hadoop - The Definitive Guide, Tom White*

63

63

MapReduce

- MapReduce i programski jezici
 - paradigma je nezavisna od programskog jezika
 - podržana u više programskih jezika
 - Java, Python, Scala, itd.
 - Hadoop je implementiran u programskom jeziku Java
 - moguće koristiti bez značajnog povišenja kompleksnosti i u drugim programskim jezicima
 - postoji API nazvan Hadoop Streaming
 - koji omogućava pristup iz drugih programskih jezika
 - stoga preporuka da se i MapReduce piše u istom jeziku
 - smanjuje kompleksnost (jako neznatnu) izazvanu razlikama u programskim jezicima
 - obično je preporuka da se programi pišu u jezicima u kojima je napisan i radni okvir
 - npr. Hadoop - Java ili Spark - Scala

64

64

MapReduce - primer

- Analize meteoroloških podataka - MapReduce i Java (funkcije *map* i *reduce*)

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {
    private static final int MISSING = 9999;
    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String line = value.toString();
        String year = line.substring(15, 19);
        int airTemperature;
        if (line.charAt(87) == '+') { // parseInt doesn't like leading plus
            airTemperature = Integer.parseInt(line.substring(88, 92));
        } else {
            airTemperature = Integer.parseInt(line.substring(87, 92));
        }
        String quality = line.substring(92, 93);
        if (airTemperature != MISSING && quality.matches("[01459]")) {
            context.write(new Text(year), new IntWritable(airTemperature));
        }
    }
}
```

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        int maxValue = Integer.MIN_VALUE;
        for (IntWritable value : values) {
            maxValue = Math.max(maxValue, value.get());
        }
        context.write(key, new IntWritable(maxValue));
    }
}
```

65

65

MapReduce - primer

- Analize meteoroloških podataka - MapReduce i Java (funkcija za pokretanje)

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class MaxTemperature {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MaxTemperature <input path> <output
                path>");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(MaxTemperature.class);
        job.setJobName("Max temperature");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

66

66

Hadoop I/O

- Kompresija

- primer pokretanja MapReduce programa čiji je rezultat kompresovan

```

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.compress.GzipCodec
public class MaxTemperatureWithCompression {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MaxTemperatureWithCompression " +
                "<input path> <output path>");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(MaxTemperature.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileOutputFormat.setCompressOutput(job, true);
        FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setCombinerClass(MaxTemperatureReducer.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

67

67

MapReduce

- MapReduce i RDBMS
 - paketna / pojedinačna obrada
 - nestrukturirani / strukturirani podaci
 - denormalizovani / normalizovani podaci

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

izvor: *Hadoop - The Definitive Guide, Tom White*

68

68

MapReduce

- MapReduce - osnovni pojmovi
 - **posao** (engl. *job*)
 - osnovna jedinica obrade
 - sastoji se od ulaznih podataka, MapReduce programa i konfiguracije
 - **zadatak** (engl. *task*)
 - Hadoop deli svaki posao na zadatke
 - dva tipa zadatka: **zadatak map i zadatak reduce**
 - implementiraju operacije *map* i *reduce*
 - **izvršni čvorovi** (engl. *execution nodes*)
 - čvorovi u distribuiranom sistemu koji kontrolišu izvršenje poslova
 - jedan čvor **upravljač poslovima** (engl. *jobtracker*)
 - upravlja izvršenjem svih poslova i raspoređuje zadatke na čvorove za upravljanje zadacima
 - više čvorova **upravljača zadacija** (engl. *tasktracker*)
 - izvršavaju prosledene zadatke i šalju informacije o statusu izvršavanju (napretku)

69

69

MapReduce

- MapReduce - osnovni pojmovi
 - **paket** (engl. *input split / split*)
 - deo ulaznih podataka koji predstavlja osnovnu jedinicu obrade u zadacima
 - Hadoop kreira jedan zadatak za svaki paket koji je potrebno obraditi
 - radi paralelizacije obrade podataka
 - uvek je tačno definisane veličine (podrazumevano 64MB)
 - obično je jednaka veličini bloka u kojem su podaci uskladišteni (HDFS)
 - moguće je ručno podesiti veličinu paketa
 - premali paket - potrebno previše vremena za logistiku
 - preveliki paket - najsporiji računar prouzrokuje veliki zastoj u celokupnoj obradi
 - **slog** (engl. *record*)
 - deo paketa, pojedinačni zapis u paketu
 - nad svakim slogom u paketu se primenjuje *map* operacija

70

70

MapReduce

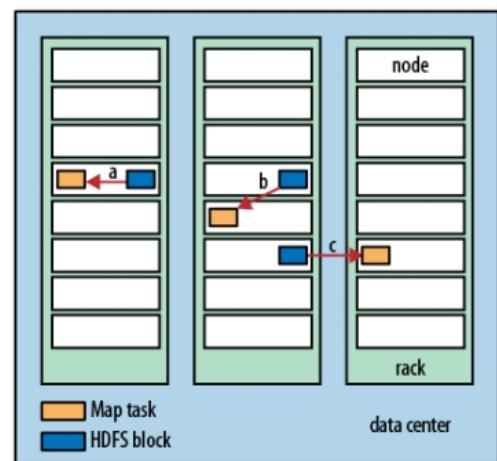
- MapReduce - osnovni pojmovi
 - **lokalizacija obrade podataka** (engl. *data locality optimization*)
 - Hadoop pokušava obraditi podatke u onim čvorovima u kojim su podaci skladišteni
 - ukoliko to nije moguće pribegava se izvršenju na drugim, dostupnim čvorovima
 - odnosi se na obradu u operaciji *map*
 - koja preuzima podatke iz distribuiranog sistema datoteka
 - tri mogućnosti izvršavanja zadataka
 - **lokalno izvršavanje zadataka** (engl. *data-local task execution*)
 - izvršavanje na čvoru na kojem se nalaze podaci
 - **izvršavanje unutar rack-a** (engl. *rack-local task execution*)
 - izvršavanje na čvoru koji se fizički nalazi unutar smeštajne jedinice za računare (*rack-a*)
 - **globalno izvršavanje** (engl. *off-rack task execution*)
 - izvršavanje na čvoru koji se nalazi unutar klastera ali nije u okviru istog *rack-a*

71

71

MapReduce

- MapReduce - osnovni pojmovi
 - a) lokalno izvršavanje zadataka
 - b) izvršavanje unutar racka
 - c) globalno izvršavanje
 - veličina paketa je obično jednaka veličini bloka
 - radi smanjenja saobraćaja i površenja performansi obrade



izvor: Hadoop - The Definitive Guide, Tom White

72

72

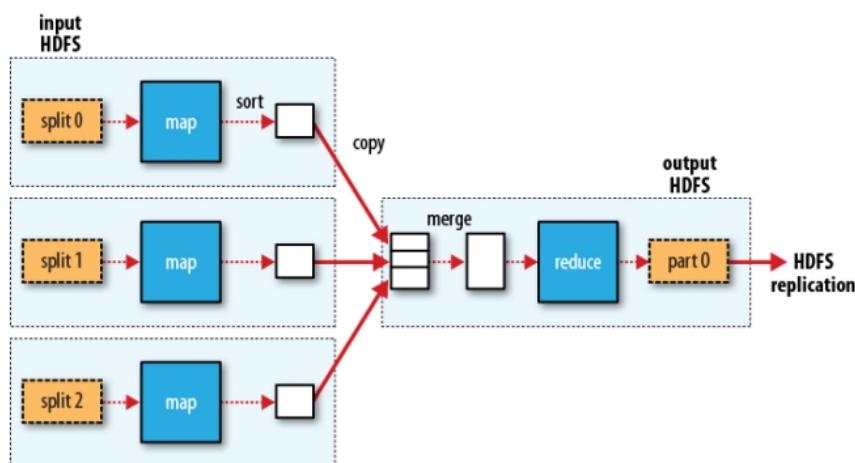
MapReduce

- MapReduce - osnovni pojmovi
 - lokalizacija obrade podataka
 - ne odnosi se na obradu u zadatku *reduce*
 - pošto preuzima podatke koji su rezultat zadatka *map* a koji su smešteni lokalno u čvorovima u kojima se taj zadatak i izvršava
 - ne smeštaju se u distribuirani sistem datoteka
 - pošto se podaci preuzimaju od više zadataka *map*
 - koji se ne moraju izvršavati na istom čvoru
 - rezultati obrade podataka u okviru zadatka *reduce*
 - obično se smeštaju u distribuirani sistem datoteka
 - radi obezbeđenja pouzdanosti i redundantne podataka

73

73

MapReduce



izvor: Hadoop - The Definitive Guide, Tom White

74

74

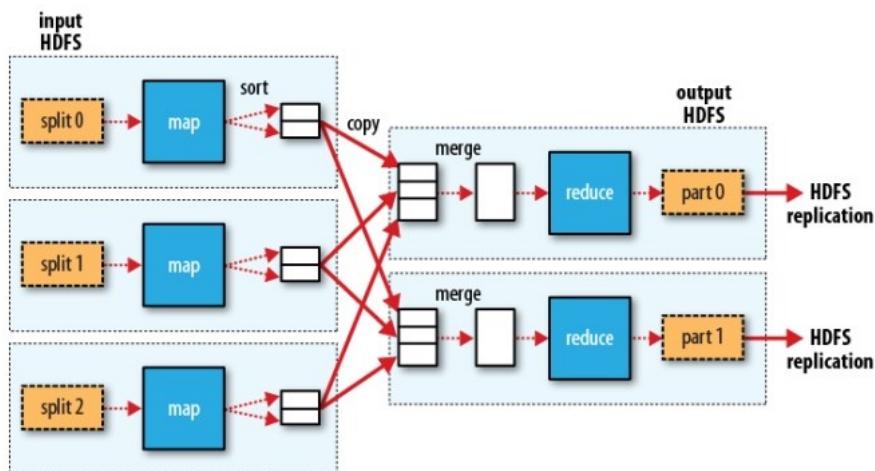
MapReduce

- MapReduce - osnovni pojmovi
 - broj zadataka *reduce* nije uslovljen veličinom ulaznog skupa podataka
 - već predstavlja parametar koji se posebno može podešavati
 - **particije** (engl. *partitions*)
 - skupovi podataka koji predstavljaju ulazne podatke za zadatak *reduce*
 - sačinjeni od izlaznih podataka iz zadatka *map*
 - može sadržavati više ključeva
 - ali svi slogovi koji pripadaju istom ključu moraju se naći u jednoj particiji
 - uobičajeno se koristi ugrađena funkcija za partionisanje
 - zasnovana na izračunavanju *hash* funkcije nad ključem
 - moguće implementirati i koristiti korisnički definisani funkciju za partionisanje
 - **raspodela podataka** (engl. *shuffle*)
 - obuhvata razmenu podataka između čvorova sa zadatkom *map* i zadatkom *reduce*
 - može biti izuzetno kompleksna i imati veliki uticaj na performanse celokupnog sistema

75

75

MapReduce

izvor: *Hadoop - The Definitive Guide*, Tom White

76

76

MapReduce

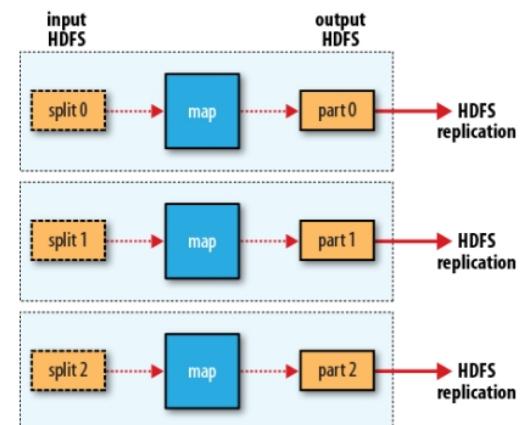
- MapReduce - osnovni pojmovi
 - postoje MapReduce programi bez zadatka *reduce*
 - ukoliko celokupna obrada podataka može biti izvršena paralelno
 - u okviru zadatka *map* izlazni podaci se snimaju u distribuirani sistem datoteka
 - npr. simulacije nad parametrima
 - ulaz: parametri za simulaciju
 - zadatak *map* obuhvata implementaciju simulacije
 - koja se izvršava veliki broj puta nad ulaznim parametrima
 - izlaz: rezultat simulacije
 - npr. učitavanje podataka iz različitih izvora podataka
 - ulaz: lista izvora podataka
 - po jedan zadatak *map* se može izvršiti za svaki izvor podataka
 - izlaz: dobavljeni podaci iz izvora podataka

77

77

MapReduce

- MapReduce programi bez zadatka *reduce*

izvor: *Hadoop - The Definitive Guide*, Tom White

78

78

MapReduce

- MapReduce - osnovni pojmovi
 - unapređenje performansi uvođenjem paralelizacije ograničeno je propusnošću mrežne infrastrukture kojom su čvorovi povezani
 - teži se smanjenju obima podataka koji se razmenjuju između čvorova
 - npr. između čvorova sa zadatkom *map* i zadatkom *reduce*
 - **funkcija za kompakciju podataka** (engl. *combiner function*)
 - funkcija koja vrši optimizaciju količine podataka koja se razmenjuje preko mreže
 - predstavlja optimizaciju, pa Hadoop ne garantuje koliko će puta biti izvršena
 - neophodno je da bude idempotentna
 - da za isti ulaz uvek daje isti izlaz
 - ograničava broj funkcija koje je moguće iskoristiti
 - **komutativne i asocijativne funkcije** (često nazvane i **distributivne**)
 - npr. `max()` je moguće dok `mean()` nije
 - izvršava se nad podacima koji predstavljaju izlaz iz zadatka *map*
 - rezultat izvršenja predstavlja ulaz za zadatak *reduce*

79

79

MapReduce - primer

- Analize meteoroloških podataka - MapReduce
 - funkcija za kompakciju podataka
 - obuhvata kompakciju izlaza iz svakog zadatka *map*
 - ne menja funkciju *reduce* već samo smanjuje količinu podataka koja se šalje preko mreže
 - $\max(0, 20, 10, 25, 15)$
 $= \max(\max(0, 20, 10), \max(25, 15))$
 $= \max(20, 25)$
 $= 25$

```
#split 1 / map 1 output
(1950, 0)
(1950, 20)
(1950, 10)

#split 2 / map 2 output
(1950, 25)
(1950, 15)

-----
#reducer without combiner function
(1950, [0, 20, 10, 25, 15]) #input
(1950, 25) #output

-----
#combiner outputs
(1950, 20) #for map1
(1950, 25) #for map2

#reducer with combiner function
(1950, [20, 25]) #input
(1950, 25) #output
```

80

80

MapReduce - primer

- Analize meteoroloških podataka - MapReduce i Java (funkcija za pokretanje)
 - implementacija funkcije za kompakciju podataka je identična implementaciji funkcije *reduce*

```

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class MaxTemperature {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MaxTemperature <input path> <output
path>");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(MaxTemperature.class);
        job.setJobName("Max temperature");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(MaxTemperatureMapper.class);
        job.setCombinerClass(MaxTemperatureReducer.class);
        job.setReducerClass(MaxTemperatureReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

81

81

MapReduce - verzije

- MapReduce 1
 - koristi servise *JobTracker* i *TaskTracker*
 - za upravljanje izvršavanjem poslova
 - nisu pogodni za skaliranje
 - jedan *JobTracker* je zadužen za pokretanje svih MapReduce poslova
 - zauzima resurse fiksne veličine za izvršenje poslova
 - resursi se dele na *map* resurse i *reduce* resurse
 - dodeljene samo odgovarajućim zadacima bez mogućnosti dinamičke izmene
- MapReduce 2
 - koristi YARN (engl. *Yet Another Resource Manager*) za upravljanje resursima
 - dinamička alokacija i upravljanje resursima (ne oslanja se na *JobTracker* i *TaskTracker*)
 - omogućava pokretanje i drugih programa osim MapReduce na Hadoop klantru

82

82

MapReduce - izvršavanje

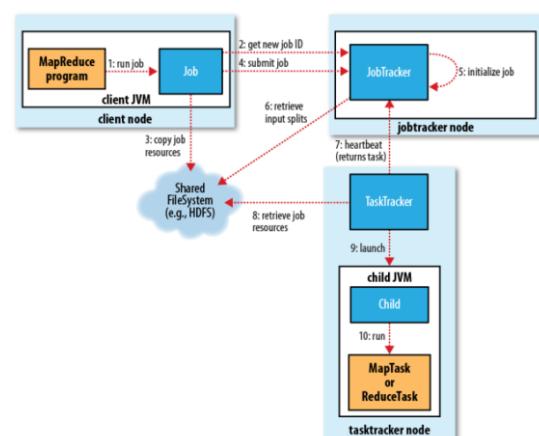
- Pokretanje MapReduce posla
 - pozivom metoda nad objektom job
 - submit() ili waitForCompletion()
 - potrebno je prethodno izvršiti konfiguraciju Hadoop klastera
 - nekolicina parametara iz konfiguracione datoteke utiče direktno i na izvršavanje MapReduce programa
 - npr. mapreduce.framework.name property
 - local - za poslove koji se izvršavaju u lokalnu
 - classic - za poslove koji se izvršavaju u skladu sa verzijom MapReduce 1
 - yarn - za poslove koji se izvršavaju u skladu sa verzijom MapReduce 2
 - na YARN upravljaču resursima

83

83

MapReduce 1

- Istorijski prvi MapReduce radni okvir
 - složenije i manje dinamički kreirano okruženje
 - dovodi do pojave uskog grla izvršavanja zadatka



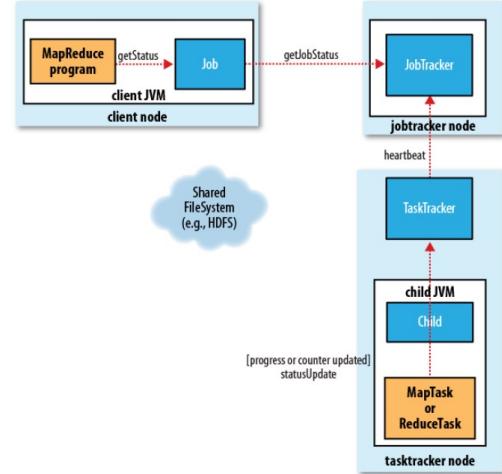
izvor: Hadoop - The Definitive Guide, Tom White

84

84

MapReduce 1

- MapReduce periodično vraćaju obaveštenje o trenutnom statusu obrade
 - interno se prati u kojoj se fazi trenutno obrada nalazi (*sort, shuffle, reduce...*)

izvor: *Hadoop - The Definitive Guide, Tom White*

85

85

MapReduce 2

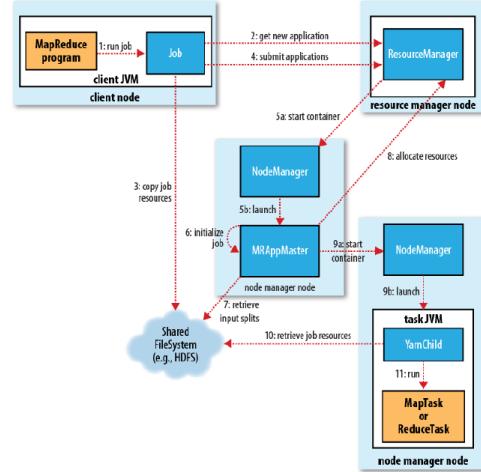
- YARN
 - napravljen u Yahoo! kao odgovor na mane i nemogućnost skaliranja zadataka u slučajevima velikog broja čvorova
 - deli zaduženja *JobTracker* procesa
 - umesto da jedan proces vodi računa o dodeli resursa i o praćenju napretka izvršavanja zadataka
 - postoje dva procesa
 - upravljač resursima (engl. *resource manager*)
 - upravljač aplikacija (engl. *application manager*)
 - omogućava izvršavanje i drugih aplikacija na klasteru a ne samo MapReduce
 - npr. distribuirana konzola za pokretanje skriptova na više čvorova u klasteru, MPI aplikacija, itd.

86

86

MapReduce 2

- MapReduce 2 i YARN upravljač resursima



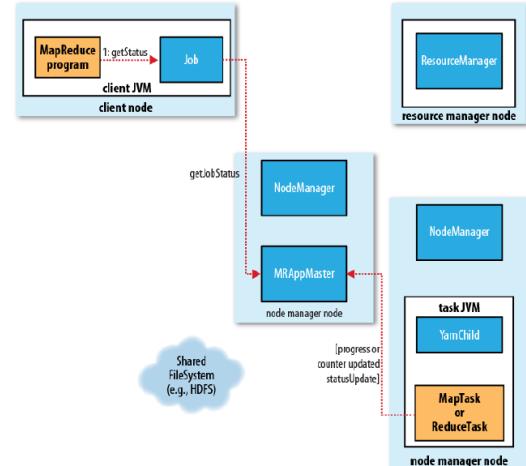
izvor: Hadoop - The Definitive Guide, Tom White

87

87

MapReduce 2

- MapReduce periodično vraćaju obaveštenje o trenutnom statususu obrade



izvor: Hadoop - The Definitive Guide, Tom White

88

88

Raspoređivanje poslova

- Raspoređivanje poslova (engl. *job scheduling*)
 - u početku, raspoređivanje je bilo predodređeno vremenom kreiranja posla
 - FIFO raspoređivač
 - čak i sa kasnijim dodatkom prioriteta poslovima dugotrajni spori proces je blokirao prioritetsnije poslove koji su nastali nakon što je on bio pokrenut
 - danas postoje još dva tipa raspoređivača
 - ravnopravni raspoređivač (engl. *fair scheduler*)
 - raspoređivač zasnovan na kapacitetu (engl. *capacity scheduler*)

89

89

Raspoređivanje poslova

- Ravnopravni raspoređivač
 - svaki korisnik koji kreira poslove u sistemu bi trebalo da dobije ravnomerni deo infrastrukture
 - ako se izvršava jedan posao od jednog korisnika, dobiće ceo klanster
 - ukoliko postoji više korisnika koji su pokrenuli poslove
 - svi će biti izvršeni sa odgovarajućim delom resursa
 - svi poslovi jednog korisnika smešteni su u kolekcije poslova
 - tako da svaka kolekcija poslova dobija odgovarajući deo resursa
 - korisnik sa više poslova neće dobiti više resursa
 - podržava dodelu resursa prioritetsnjim resursima čak i u slučaju da je sporiji proces zauzeo dobar deo resursa klastera
 - delovi tog procesa biće zaustavljeni kako bi se oslobođili resursi

90

90

Raspoređivanje poslova

- Raspoređivač zasnovan na kapacitetu
 - umesto kolekcija poslova kreiranih za svakog korisnika postoji hijerarhija redova čekanja
 - svaki red čekanja ima dodeljeni kapacitet tj. deo resursa klastera
 - unutar svakog reda čekanja, poslovi su raspoređeni u FIFO strukturu

91

91

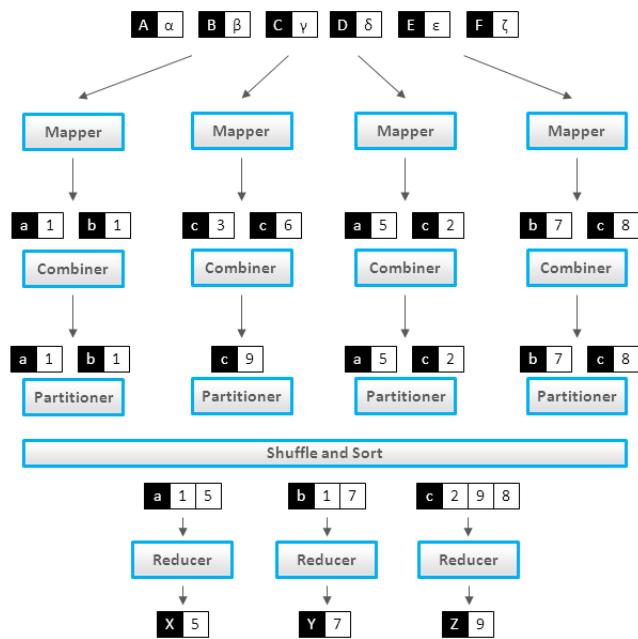
Raspodela i sortiranje podataka

- Raspodela (engl. *shuffle*) i sortiranje (engl. *sort*) podataka
 - MapReduce okvir garantuje da će ulaz u svaki *reducer* biti sortiran po ključu
 - **proces u okviru kojeg je implementirano sortiranje vrednosti koje predstavljaju izlaz iz koraka *map* i raspoređivanje tih vrednosti odgovarajućim koracima *reduce*, naziva se raspodela podataka**
 - definisan radnim okvirom a ne od strane korisnika
 - optimizacija MapReduce programa
 - da se što bolje iskoristi algoritam raspodele podataka

92

92

Raspodela i sortiranje podataka

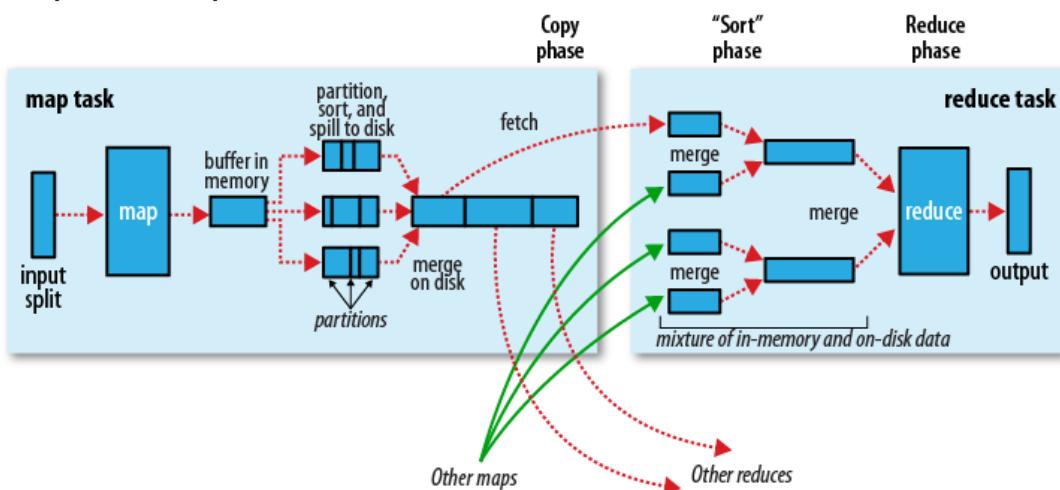


izvor: Mapreduce patterns, algorithms, and use cases, Ilya Katsov, <https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/>

93

93

Raspodela podataka



izvor: Hadoop - The Definitive Guide, Tom White

94

94

Raspodela podataka

- Raspodela podataka (u koraku *map*)
 - svaki *map* zadatak rezultate piše u cirkularni *buffer*
 - 100 MB prepodešena vrednost
 - kada se *buffer* napuni do podešene granice pozadinska nit će početi da snima istisnute podatke (engl. *spills*) na disk
 - ukoliko se *buffer* popuni pre nego što si svi istisnuti podaci snimljeni a novi podaci pristižu, zadatak *map* koji generiše podatke biće blokiran
 - pre svakog pisanja na disk
 - istisnuti podaci se partitionišu u particije koje odgovaraju ciljnim *reducer-ima*
 - svaka particija se sortira u pozadinskom procesu
 - ukoliko postoji, primenjuje se *combiner*
 - za kompakciju torki i smanjenje saobraćaja preko mreže
 - rezultat partitionisanja i sortiranja su datoteke na disku koje je potrebno proslediti *reducer-ima*

95

95

Sortiranje podataka

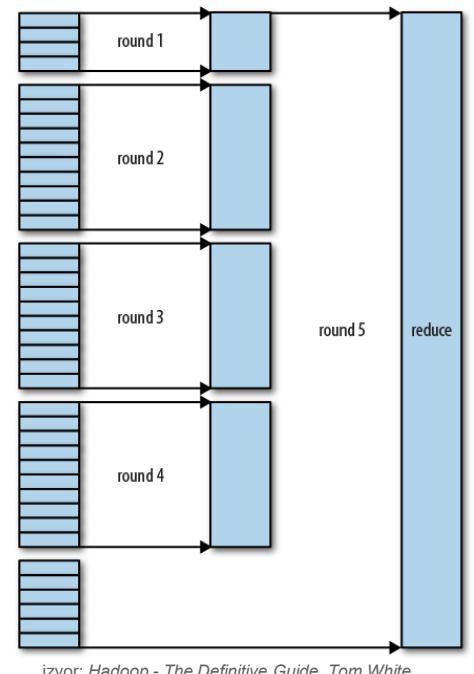
- Sortiranje podataka (u koraku *reduce*)
 - *reducer* "dovlači" partitionisane podatke kada koraci *map* završe sa njihovim kreiranjem
 - koraci *map* obaveštavaju *JobTracker-a* kada su završili sa svojom obradom
 - moguće da različiti koraci *map* završe u različitim vremenskim trenucima
 - **faza kopiranja** (engl. *copy phase*) u *buffer reducer-a*
 - u radnu memoriju *reducer-a* za male datoteke
 - nakon što je odgovarajući broj datoteka kopiran u *buffer*, vrši se njihovo spajanje u veće, sortirane datoteke na disku *reducer-a*
 - ili direktno na disk *reducer-a* ako su datoteke velike
 - nakon što je završeno kopiranje svih datoteka iz koraka *map* vrši se spajanje
 - **faza sortiranja** (engl. *sort phase*)
 - bolji naziv je faza spajanja (engl. *merge phase*)
 - datoteke na disku se spajaju u iteracijama
 - u svakoj iteraciji se spaja predefinisan broj datoteka
 - održava se sortiranost paketa unutar datoteka

96

96

Sortiranje podataka

- Sortiranje podataka (u koraku *reduce*)
 - nakon spajanja datoteka sa paketima izvršava se funkcija *reduce*
 - faza reduce** (engl. *reduce phase*)
 - umesto poslednje iteracije spajanja
 - da bi se minimizovao broj pisanja na disk



izvor: Hadoop - The Definitive Guide, Tom White

37

97

MapReduce - tipovi

- Tipovi MapReduce programa
 - postoje četiri osnovna, najčešće korišćena, tipa MapReduce programa
 - ulaz-map-reduce-izlaz
 - ulaz-map-izlaz
 - ulaz-višestruki map-reduce-izlaz
 - ulaz-map-combiner-reduce-izlaz

98

98

MapReduce - tipovi

- Tip *ulaz-map-reduce-izlaz*
 - najčešće korišćen za agregacije

<i>Scenario</i>	Računanje odnosa između pola radnika i plate
<i>Map</i>	ključ: pol, vrednost: plata
<i>Reduce</i>	grupiše po polu, izračunava sumu plata za svaki pol



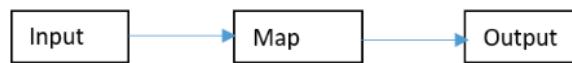
99

99

MapReduce - tipovi

- Tip *ulaz-map-izlaz*
 - najčešće korišćen za promenu formata podataka

<i>Scenario</i>	Neki zaposleni imaju isti pol označen sa <i>F</i> , <i>Female</i> , <i>f</i> , <i>0</i>
<i>Map</i>	ključ: ID zaposlenog vrednost: Pol koji predstavlja izlaz iz funkcije if Gender is Female/ F/ f/ 0 then converted to F else if Gender is Male/M/m/1 then convert to M

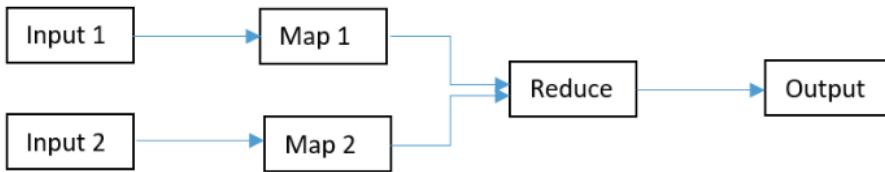


100

100

MapReduce - tipovi

- Tip *ulaz-višestruki map-reduce-izlaz*
 - najčešće korišćen za situacije kada imamo različite šeme ulaznih podataka



101

101

MapReduce - tipovi

- Tip *ulaz-višestruki map-reduce-izlaz*

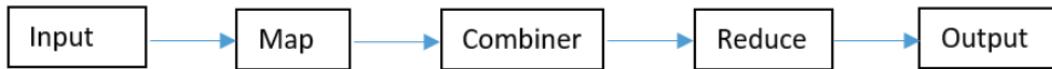
<i>Scenario</i>	Ukupna plata za pol, fajlovi u različitim formatima datoteka 1: pol dat kao prefiks imenu: <i>Mr. Vladimir</i> datoteka 2: pol dat u posebnoj koloni: <i>Vladimir, Male</i>
<i>Map</i>	Map 1: parsira ime i izvlači prefiks, rezultat par (pol, plata) Map 2: direktno preuzimanje vrednosti, rezultat par (pol, plata)
<i>Reduce</i>	grupisanje po polu, sumiranje plata

102

102

MapReduce - tipovi

- Tip *ulaz-map-combiner-reduce-izlaz*
 - najčešće korišćen kada *reduce* funkciju nije moguće paralelizovati
 - smanjuje se opterećenje na toj funkciji



103

103

MapReduce - tipovi

- Tip *ulaz-map-combiner-reduce-izlaz*
 - Scenario: Ukupna plata za pol, radnici u različitim departmanima. Departmana ima 5. Ako je plata u okviru departmana preko 200k dodaj 20k, ako je preko 100k dodaj 10k

ulaz	map (paralelno)	combiner (paralelno)	reduce (nije paralelno)	izlaz
datoteka_dep1	Male<10,20,25,45,15,45,25,20> Female <10,30,20,25,35>	Male <250,20> Female <120,10>		
datoteka_dep2	Male<15,30,40,25,45> Female <20,35,25,35,40>	Male <155,10> Female <175,10>	Male <250,20,155,10,90,90,30>	Male <645>
datoteka_dep3	Male<10,20,20,40> Female <10,30,25,70>	Male <90,00> Female <135,10>	Female <120,10,175,10,135,10,110,10,130,10>	Female <720>
datoteka_dep4	Male<45,25,20> Female <30,20,25,35>	Male <90,00> Female <110,10>		
datoteka_dep5	Male<10,20> Female <10,30,20,25,35>	Male <30,00> Female <130,10>		

104

104

MapReduce - osnovni šabloni i primeri

- Prebrojavanje i sumiranje
 - **problem**
 - snimljeni dokumenti sadrže skupove termina
 - potrebno je prebrojati broj pojava svakog termina u svim dokumentima
 - ili izvršiti bilo koju drugu funkciju koja zavisi od termina
 - **primena**
 - analiza logova, upiti nad podacima

105

105

MapReduce - osnovni šabloni i primeri

- Prebrojavanje i sumiranje - rešenje 1
 - najprostije rešenje
 - funkcija map šalje termin i broj 1 kao izlaz za svaki termin koji parsira
 - mana
 - preveliki broj brojača koji se prenosi preko mreže
 - rešenje
 - potrebno nekako agregirati podatke koji predstavljaju izlaz iz funkcije map

```
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts [c1, c2,...])
        sum = 0
        for all count c in [c1, c2,...] do
            sum = sum + c
        Emit(term t, count sum)
```

106

106

MapReduce - osnovni šabloni i primeri

- Prebrojavanje i sumiranje - rešenje 2
 - agregiraju se sve pojave istog termina u dokumentu
 - smanjuje se broj poslatih brojača preko mreže
- Prebrojavanje i sumiranje - rešenje 3
 - iskoristiti *combiner*
 - agregirati sve pojave istog termina u svim dokumentima procesiranim od strane istog *mapper-a*

```

class Mapper
    method Map(docid id, doc d)
        H = new AssociativeArray
        for all term t in doc d do
            H{t} = H{t} + 1
        for all term t in H do
            Emit(term t, count H{t})

-----
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do
            Emit(term t, count 1)

class Combiner
    method Combine(term t, [c1, c2,...])
        sum = 0
        for all count c in [c1, c2,...] do
            sum = sum + c
        Emit(term t, count sum)

class Reducer
    method Reduce(term t, counts [c1, c2,...])
        sum = 0
        for all count c in [c1, c2,...] do
            sum = sum + c
        Emit(term t, count sum)

```

107

107

MapReduce - osnovni šabloni i primeri

- Sakupljanje (engl. *collating*)
 - **problem**
 - snimljeni dokumenti sadrže skup torki
 - postoji funkcija nad pojedinačnim torkama
 - potrebno je snimiti sve torke koje daju istu vrednost funkcije u jednu datoteku
 - ili izvršiti neku analizu nad tako konstruisanom grupom torki
 - **primena**
 - **invertovani indeksi**, ETL

108

108

MapReduce - osnovni šabloni i primeri

- Sakupljanje (engl. *collating*)
 - **rešenje**
 - *mapper* izračunava funkciju za svaku torku koja se obraduje
 - vrednost funkcije je ključ
 - torka je vrednost
 - *reducer* procesira grupisane torke i snima ih u dokument/analizira
 - TF/IDF
 - funkcija vraća ID dokumenta u kojem se torka nalazi
 - $\{(DOC\ ID, Term), TF\}$
 - $N / \{Term, DF\}$

```
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do
            Emit(function(t), term t)

class Reducer
    method Reduce(funct t, set [t1, t2,...])
        analize = analize([t1, t2,...])
        Emit(funct t, analize)
```

109

109

MapReduce - osnovni šabloni i primeri

- Filtriranje, parsiranje i validacija
 - **problem**
 - postoji potreba za
 - preuzimanjem torki koje zadovoljavaju neki uslov
 - transformacijom svake torke u novi oblik
 - **primena**
 - ETL, validacija podataka, upiti, analiza logova

110

110

MapReduce - osnovni šabloni i primeri

- Filtriranje, parsiranje i validacija
 - **rešenje**
 - koristiti samo funkciju *map* koja vraća torke (njihov transformisani oblik) koje zadovoljavaju uslov

```
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do
            if condition t is true then
                Emit(function(t), null)
```

111

111

MapReduce šabloni i primeri

- Izvršavanje zadataka u distribuiranom okruženju
 - **problem**
 - postoji zadatak, zahtevan sa stanovišta potrebnog procesorskog vremena, koji je moguće dekomponovati na niz manjih zadataka i na kraju kombinovati rezultat
 - **primena**
 - simulacije u fizici i inženjerskim disciplinama, numeričke analize, analiza performansi
 - **rešenje**
 - svaki od podzadataka predstavljen je posebnom funkcijom *map*
 - ulazi su specifične, tražene vrednosti za svaki od zadataka
 - funkcija *map* preuzme ulaz, izvrši izračunavanje, vrati rezultat izvršenja
 - svi rezultati su objedinjeni u funkciji *reduce*

112

112

MapReduce - osnovni šabloni i primeri

- Izvršavanje zadataka u distribuiranom okruženju
 - **primer upotrebe**
 - simulator komunikacije preko računarske mreže
 - slučajno generisane podatke propušta preko modela mreže
 - potrebno je izračunati verovatnoću pojave greške u komunikaciji
 - svaka funkcija *map* (od ukupno N instanci) izvrši simulaciju nad 1/N ulaznih podataka
 - vraća broj uočenih grešaka u komunikaciji
 - funkcija *reduce* spaja sve izlaze funkcije *map*
 - npr. izračunava srednju vrednost broja grešaka

113

113

MapReduce - osnovni šabloni i primeri

- Sortiranje
 - **problem**
 - skup torki je potrebno sortirati ili procesirati u odgovarajućem redosledu
 - **primena**
 - ETL, analiza podataka
 - **rešenje**
 - jednostavno sortiranje je moguće uraditi tako što se kao izlaz iz mapera generiše
 - *ključ*: vrednost obeležja nad kojom se sortira
 - *vrednost*: torka
 - postoje tehnike koje dozvoljavaju sortiranje po vrednosti a ne samo po ključu
 - često je pogodnije održavati snimljene vrednosti u HDFS-u u sortiranom poretku

114

114

MapReduce - napredni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **problem**
 - potrebno je izračunati stanje čvora u grafu
 - na stanje čvora utiču osobine povezanih čvorova, tj. čvorova u okolini
 - stanje može predstavljati udaljenost između čvorova, indikator da postoji neki čvor u okolini koji zadovoljava određeni uslov itd.
 - **primena**
 - analiza grafova, indeksiranje web sadržaja

115

115

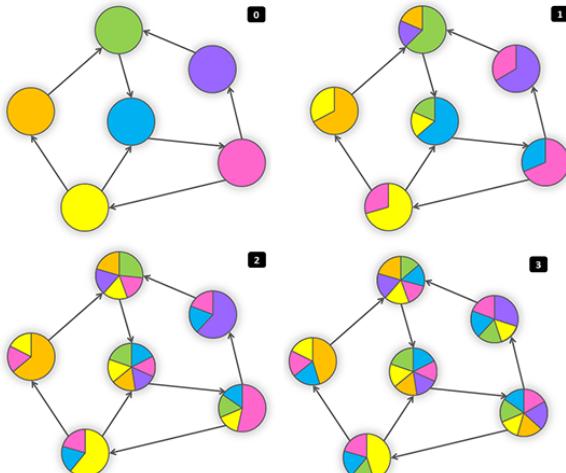
MapReduce - napredni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **rešenje**
 - konceptualno
 - svaki čvor poseduje listu ID-ova povezanih čvorova
 - MapReduce zadaci se izvršavaju iterativno
 - u svakoj iteraciji se šalje poruka susednom čvoru nakon koje on ažurira stanje
 - iteracije se završavaju nakon fiksнog broja koraka ili kada izmene stanja postanu zanemarljive
 - stanje čvora se brzo propagira kroz graf
 - tehnički
 - funkcija *map* šalje poruke (ID susednog čvora, poruka)
 - funkcija *reduce* prima grupisane poruke po ID-u čvora i izračunava novo stanje

116

116

MapReduce - osnovni šabloni i primeri



```

class Mapper
    method Map(id n, object N)
        Emit(id n, object N)
        for all id m in N.OutgoingRelations do
            Emit(id m, message getMessage(N))

class Reducer
    method Reduce(id m, [s1, s2,...])
        M = null
        messages = []
        for all s in [s1, s2,...] do
            if IsObject(s) then
                M = s
            else
                messages.add(s)
        M.State = calculateState(messages)
        Emit(id m, item M)
    
```

117

117

MapReduce - osnovni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **primer upotrebe 1 - propagacija dostupnosti kroz stablo kategorija**
 - kategorije proizvoda neke *online* prodavnice su organizovane u obliku stabla
 - od kategorija: muškarci, žene, deca
 - do kategorija: muške pantalone ...
 - kategorije na najnižim nivoima su dostupne ukoliko postoje artikli te kategorije na stanju
 - sve kategorije na višim nivoima su dostupne ako je dostupna makar jedna kategorija u njenom podstablu
 - potrebno je izračunati dostupnost kategorija višeg nivoa ako su poznate dostupnosti svih kategorija nižeg nivoa

118

118

MapReduce - osnovni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **primer upotrebe 1 - rešenje**

```
class N
    State in {True = 2, False = 1, null = 0},
    initialized 1 or 2 for end-of-line categories, 0 otherwise

method getMessage(object N)
    return N.State

method calculateState(state s, data [d1, d2,...])
    return max( [d1, d2,...] )
```

119

119

MapReduce - osnovni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **primer upotrebe 2 - pretraga prvi u širinu (BFS)**
 - potrebno je izračunati udaljenost od nekog čvora do svih drugih čvorova u grafu
 - **rešenje**
 - izvorni čvor šalje svim susedima 0, svaki od suseda preuzima poruku, uvećava brojač za jedan i šalje dalje susednim čvorovima

```
class N
    State is distance, initialized 0 for source node, INFINITY for all other nodes

method getMessage(N)
    return N.State + 1

method calculateState(state s, data [d1, d2,...])
    min( [d1, d2,...] )
```

120

120

MapReduce - osnovni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - **primer upotrebe 3 - PageRank**
 - potrebno je izračunati relevantnost stranica koja je funkcija autoritativnosti drugih stranica koje imaju vezu ka njoj
 - **rešenje**
 - svodi se na propagaciju težina
 - težina čvora je prosečna težina povezanih čvorova

```
class N
    State is PageRank

method getMessage(object N)
    return N.State / N.OutgoingRelations.size()

method calculateState(state s, data [d1, d2,...])
    return ( sum([d1, d2,...]) )
```

121

121

MapReduce - osnovni šabloni i primeri

- Obrada grafova/iterativna obrada poruka
 - optimizacija u slučaju razmene samo celobrojnih vrednosti
 - umesto generičkog algoritma, moguće je primeniti algoritam koji vrši agregaciju na nivou funkcije *map* i tako agregirane podatke šalje funkciji *reduce*
 - *Close = Combiner*

```
class Mapper
    method Initialize
        H = new AssociativeArray
    method Map(id n, object N)
        p = N.PageRank / N.OutgoingRelations.size()
        Emit(id n, object N)
        for all id m in N.OutgoingRelations do
            H{m} = H{m} + p
    method Close
        for all id n in H do
            Emit(id n, value H{n})

class Reducer
    method Reduce(id m, [s1, s2,...])
        M = null
        p = 0
        for all s in [s1, s2,...] do
            if IsObject(s) then
                M = s
            else
                p = p + s
        M.PageRank = p
        Emit(id m, item M)
```

122

122

MapReduce - napredni šabloni i primeri

- Brojanje jedinstvenih vrednosti
 - **problem**
 - neka svaka torka skupa ima polja F i G
 - potrebno je izračunati broj različitih vrednosti polja F za neku vrednost polja G
 - **primena**
 - analiza logova, prebrojavanje jedinstvenih korisnika

```
Record 1: F=1, G={a, b}
Record 2: F=2, G={a, d, e}
Record 3: F=1, G={b}
Record 4: F=3, G={a, b}
```

Result:

```
a -> 3    // F=1, F=2, F=3
b -> 2    // F=1, F=3
d -> 1    // F=2
e -> 1    // F=2
```

123

123

MapReduce - napredni šabloni i primeri

- Brojanje jedinstvenih vrednosti
 - **rešenje 1**
 - rešenje u dve faze
 - u prvoj fazi
 - funkcija *map* generiše broj 1 za svaki par vrednosti [G, F]
 - funkcija *reduce* izračunava ukupan broj pojavljivanja svakog od tih parova
 - osnovna svrha ove faze je da obezbedi jedinstvenost vrednosti polja F
 - u drugoj fazi
 - u funkciji *map* parovi su grupisani po vrednosti polja G
 - u funkciji *reduce* se izračunava ukupan broj vrednosti u svakoj grupi

124

124

MapReduce - napredni šabloni i primeri

```
--PHASE 1---
class Mapper
    method Map(null, record [value f, categories [g1, g2,...]])
        for all category g in [g1, g2,...]
            Emit(record [g, f], count 1)

class Reducer
    method Reduce(record [g, f], counts [n1, n2, ...])
        Emit(record [g, f], null )

---PHASE 2---
class Mapper
    method Map(record [f, g], null)
        Emit(value g, count 1)

class Reducer
    method Reduce(value g, counts [n1, n2,...])
        Emit(value g, sum( [n1, n2,...] ) )
```

125

125

MapReduce - napredni šabloni i primeri

- Brojanje jedinstvenih vrednosti
 - **rešenje 2**
 - rešenje u jednoj fazi
 - ali ne skalira dobro kao prvo rešenje i primena mu je ograničena
 - u slučaju kada broj istih vrednosti polja F u okviru iste vrednosti polja G nije prevelik
 - u slučaju kada postoji mali i ograničen broj vrednosti polja G
 - funkcija *map* daje na izlazu parove vrednosti F i G
 - funkcija *reduce*
 - odstranjuje duple vrednosti polja F u istoj kategoriji (za istu vrednost polja G) i uvećava brojač u okviru svake kategorije
 - nakon kompletног *reduce* koraka sumiraju se svi brojači generisani od strane *reducer-a*
 - moguće je koristiti funkcije *combine* umesto da se duplikati uklanjaju u *reduce* koraku

126

126

MapReduce - napredni šabloni i primeri

```

class Mapper
    method Map(null, record [value f, categories [g1, g2,...] )
        for all category g in [g1, g2,...]
            Emit(value f, category g)

class Reducer
    method Initialize
        H = new AssociativeArray : category -> count
    method Reduce(value f, categories [g1, g2,...])
        [g1', g2',..] = ExcludeDuplicates( [g1, g2,...] )
        for all category g in [g1', g2',...]
            H{g} = H{g} + 1
    method Close
        for all category g in H do
            Emit(category g, count H{g})

```

127

127

MapReduce - napredni šabloni i primeri

- Korelacija (engl. cross-correlation)
 - **problem**
 - snimljeni dokumenti sadrže skupove termina
 - za svaki par termina, potrebno izračunati broj dokumenata u kojima se oba termina pojavljuju
 - **primena**
 - analiza teksta, analiza tržišta

128

128

MapReduce - napredni šabloni i primeri

- Korelacija

- **rešenje 1 - zasnovano na parovima**
 - u funkciji *map* generišu se parovi termina kao ključ, a broj 1 kao vrednost
 - u funkciji *reduce* sabiraju se vrednosti za iste parove
 - manje rešenja
 - ograničen efekat korišćenja *combiner-a*
 - ne može se koristiti akumulacija vrednosti u memoriji (mapa)

```
class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            for all item j in [i1, i2,...]
                Emit(pair [i j], count 1)

class Reducer
    method Reduce(pair [i j], counts [c1, c2,...])
        s = sum([c1, c2,...])
        Emit(pair[i j], count s)
```

129

129

MapReduce - napredni šabloni i primeri

- Korelacija

- **rešenje 2 - zasnovano na mapama**
 - brže ali složenije
 - u funkciji *map* kreira se mapa gde se jedan termin koristi kao ključ, a svi ostali koji se pojavljaju sa njim kao vrednosti u mapi
 - u funkciji *reduce* se spajaju mape dobijene od različitih *map* koraka i generiše se isti rezultat kao i u prethodnoj verziji rešenja
- prednosti:
 - manji broj ključeva u koraku *map*
 - bolje iskorišćenje *combiner-a*
 - akumulacija vrednosti u memoriji

```
class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            H = new AssociativeArray : item -> counter
            for all item j in [i1, i2,...]
                H{j} = H{j} + 1
            Emit(item i, stripe H)

class Reducer
    method Reduce(item i, stripes [H1, H2,...])
        H = new AssociativeArray : item -> counter
        H = merge-sum( [H1, H2,...] )
        for all item j in H.keys()
            Emit(pair [i j], H{j})
```

130

130

MapReduce - napredni šabloni i primeri

- Relacioni operatori
 - pomoću MapReduce programa moguće je implementirati sledeće relacione operatore
 - selekcija
 - projekcija
 - unija
 - presek
 - razlika
 - grupisanje i agregacija
 - spajanje

131

131

MapReduce - napredni šabloni i primeri

- Relacioni operatori - selekcija
- Relacioni operatori - projekcija
 - funkcija *reduce* eliminiše duplike

```
---Selekcija---
class Mapper
  method Map(rowkey key, tuple t)
    if t satisfies the predicate
      Emit(tuple t, null)
```

```
---Projekcija---
class Mapper
  method Map(rowkey key, tuple t)
    // extract required fields to tuple g
    tuple g = project(t)
    Emit(tuple g, null)

class Reducer
  // n is an array of nulls
  method Reduce(tuple t, array n)
    Emit(tuple t, null)
```

132

132

MapReduce - napredni šabloni i primeri

- Relacioni operatori - unija
 - funkcija *reduce* eliminiše duplike
- Relacioni operatori - presek
 - funkcija *reduce* daje na izlazu samo sloganove koji se pojavljuju u oba skupa

```
---Unija---
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, null)

class Reducer
    // n is an array of one or two nulls
    method Reduce(tuple t, array n)
        Emit(tuple t, null)
```

```
---Presek---
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, null)

class Reducer
    // n is an array of one or two nulls
    method Reduce(tuple t, array n)
        if n.size() = 2
            Emit(tuple t, null)
```

133

133

MapReduce - napredni šabloni i primeri

- Relacioni operatori - razlika

```
---Razlika---
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, string tSetName)    // t.SetName is either 'R' or 'S'

class Reducer
    method Reduce(tuple t, array n) // array n can be ['R'], ['S'], ['R', 'S'], or ['S', 'R']
        if n.size() = 1 and n[1] = 'R'
            Emit(tuple t, null)
```

134

134

MapReduce - napredni šabloni i primeri

- Grupisanje i agregacija
 - funkcija map iz svake torke izdvaja obeležja po kojima se vrši grupisanje i obeležja po kojima se vrši agregacija
 - funkcija reduce izračunava agregat
 - za neke specifične funkcije možda je potrebno uraditi algoritam u dva koraka
 - npr. kako bi se filtrirali duplikati

```
class Mapper
    method Map(null, tuple [value GroupBy, value AggregateBy, value ...])
        Emit(value GroupBy, value AggregateBy)
class Reducer
    method Reduce(value GroupBy, [v1, v2,...])
        Emit(value GroupBy, aggregate( [v1, v2,...] ) ) // aggregate() : sum(), max(),...
```

135

135

MapReduce - napredni šabloni i primeri

- Spoj sa reparticionisanjem
 - engl. *repartition join, reduce join, sort-merge join*
 - spajaju se dva skupa torki po ključu k
 - funkcija *map*
 - označava svaku torku sa oznakom iz kojeg skupa je preuzeta
 - na izlazu ima vrednosti gde je ključ k po kojem se spaja, a vrednost označena torka
 - funkcija *reduce*
 - puni dve kolekcije torkama, za svaku oznaku skupa po jedna kolekcija
 - kreira dekartov proizvod torki jednog skupa sa torkama drugog skupa po ključu k
 - **mane:**
 - funkcija *map* na izlazu ima sve moguće torke
 - uključujući i torke za ključeve koji ne postoje u drugom skupu
 - funkcija *reduce* mora da drži sve torke jednog ključa u memoriji
 - ako ne može sve da stane u memoriju mora biti u mogućnosti da privremeno skladišti torke na disku

136

136

MapReduce - napredni šabloni i primeri

```

class Mapper
    method Map(null, tuple [join_key k, value v1, value v2,...])
        Emit(join_key k, tagged_tuple [set_name tag, values [v1, v2, ...] ] )

class Reducer
    method Reduce(join_key k, tagged_tuples [t1, t2,...])
        H = new AssociativeArray : set_name -> values
        for all tagged_tuple t in [t1, t2,...]      // separate values into 2 arrays
            H{t.tag}.add(t.values)
        for all values r in H{'R'}
            for all values l in H{'L'}           // produce a cross-join of the two arrays
                Emit(null, [k r l] )

```

137

137

MapReduce - napredni šabloni i primeri

- Spoj sa pomoćnom rasutom strukturom
 - engl. *hash join, map join, replicated join*
 - spajaju se dva skupa torki po ključu k
 - kod kojih je jedan skup znatno manji od drugog
 - funkcija map
 - uzima vrednosti manjeg skupa i kreira mapu vrednosti gde je ključ k ključ u mapi a vrednost torka iz tog manjeg skupa
 - prolazi kroz torke većeg skupa i vrši spajanje sa postojećim elementima u mapi
 - **prednost:** nema potrebe za slanjem većeg skupa preko mreže niti potrebe za sortiranjem

138

138

MapReduce - napredni šabloni i primeri

```
class Mapper
    method Initialize
        H = new AssociativeArray : join_key -> tuple from R
        R = loadR()
        for all [ join_key k, tuple [r1, r2,...] ] in R
            H{k} = H{k}.append( [r1, r2,...] )

    method Map(join_key k, tuple l)
        for all tuple r in H{k}
            Emit(null, tuple [k r l] )
```

139

139

Apache Spark (paketna obrada)



140

Apache Spark

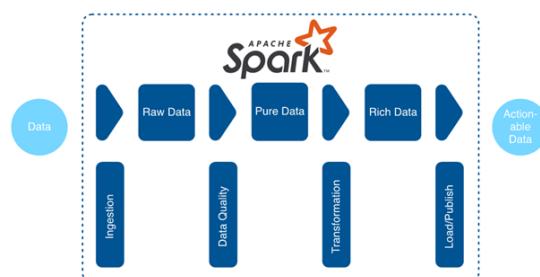
- Apache Spark je radni okvir otvorenog koda
 - za paketnu obradu podataka
 - za obradu podataka u realnom vremenu
 - koji vrši izračunavanja **u radnoj memoriji** računara koji pripadaju klasteru
- Osobine alata Apache Spark
 - **brzo procesiranje podataka** - zasnovano na čuvanju podataka u radnoj memoriji kao i na strukturama podataka optimizovanim za ovakvu vrstu obrade
 - značajno brzi od Hadoop MapReduce
 - **fleksibilnost** - podržani su programski jezici Java, Scala, R i Python i veliki broj primitiva za upravljanje podacima
 - **podrška za više paradigm obrade podataka**
 - **podrška za različite distribuirane sisteme** - kompatibilnost sa Hadoop-om i Mesos-om
 - postojanje sopstvene infrastrukture

141

141

Apache Spark - upotreba

- Tipični scenario upotrebe Apache Spark-a
 - učitavanje podataka (engl. *ingestion*)
 - iz različitih izvora
 - prečišćavanje podataka (engl. *data quality*)
 - obezbeđenje kvaliteta podataka koji se obrađuju
 - obrada podataka (engl. *transformation*)
 - objava rezultata obrade (engl. *load/publish*)
 - upis rezultata obrade podataka u skladište podataka

izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-1/v-13/43>

142

142

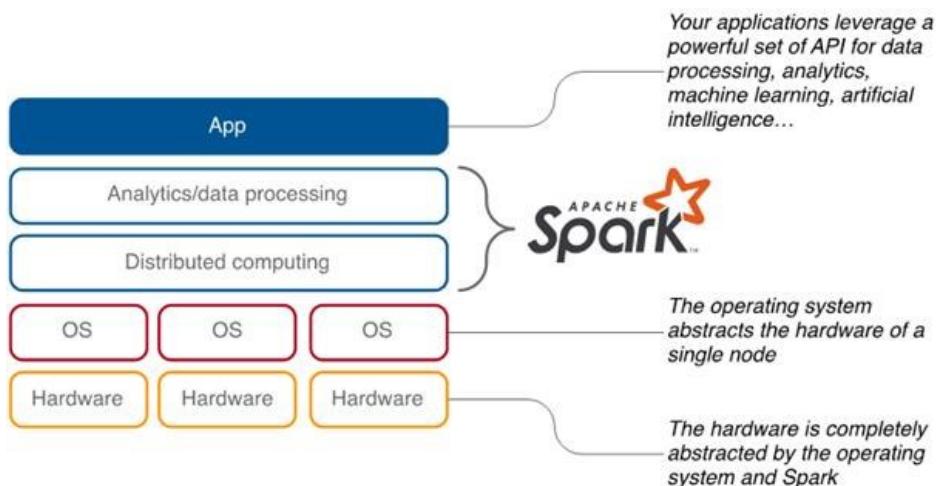
Apache Spark - upotreba

- Tipične primene radnog okvira Apache Spark
 - mašinsko učenje
 - upotrebom komponente Apache MLlib pruža mogućnost skaliranja distribuiranih algoritama mašinskog učenja
 - *fog computing*
 - u IoT sistemima, izdvajanje procesne moći iz *cloud-a* u klastere bliže izvoru podataka
 - sračunati podaci se potom često skladište u *cloud-u*
 - detekcija događaja
 - identifikacija šablona i događaja sa ciljem detekcije anomalija, identifikacija rizika i obaveštavanja korisnika
 - interaktivna analiza podataka
 - izvršavanje *ad-hoc* upita bez redukcije osnovnog skupa podataka

143

143

Apache Spark - arhitektura

izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-1/v-13/43>

144

144

Apache Spark - arhitektura

- Osnovni elementi arhitekture
 - Spark Core
 - Spark SQL
 - Spark Streaming
 - Spark MLlib
 - Spark GraphX
 - Spark R



izvor: <https://www.edureka.co/blog/spark-architecture/>

145

145

Apache Spark - arhitektura

- Spark Core
 - osnovni izvršni sistem za paralelnu i distribuiranu obradu velike količine podataka
 - predstavlja osnovu za sve ostale module
 - obuhvata i distribuirane skupove podataka otporne na otkaze
 - engl. *Resilient Distributed Datasets (RDD)*
 - particonisani skupovi podataka
 - struktura nad kojom se vrši obrada u Spark-u

146

146

Apache Spark - arhitektura

- Spark SQL
 - omogućava pisanje SQL upita radi obrade podataka
 - direktno naleže na Spark Core komponentu
 - koristi i podatke organizovane u obliku okvira podataka
 - engl. *DataFrame*
 - pandan tabele iz relacione baze podataka u Spark SQL-u
 - poseduje mogućnost definisiranja šeme podataka
 - svaka kolona ima ime i tip
 - poseduje ugrađeni optimizator upita
 - omogućava pristup putem standardnih aplikativnih mehanizama
 - moguće se konektovati preko JDBC ili ODBC
 - interno se svodi na RDD i DAG

147

147

Apache Spark - arhitektura

- Spark Streaming
 - omogućava obradu podataka u realnom vremenu
 - visoke propusne moći
 - visoke tolerancije na greške
 - obuhvata podatke iz toka podataka u paketima male veličine
 - posmatra ih kao RDD
 - mikro-paketna obrada podataka
- Spark GraphX
 - komponenta za distribuiranu obradu podataka organizovanih putem strukture tipa grafa
 - omogućava obradu grafova
 - apstrakcija sistema Pregel
 - za distribuiranu obradu grafova
 - proširenje RDD podrškom za formiranje grafova

148

148

Apache Spark - arhitektura

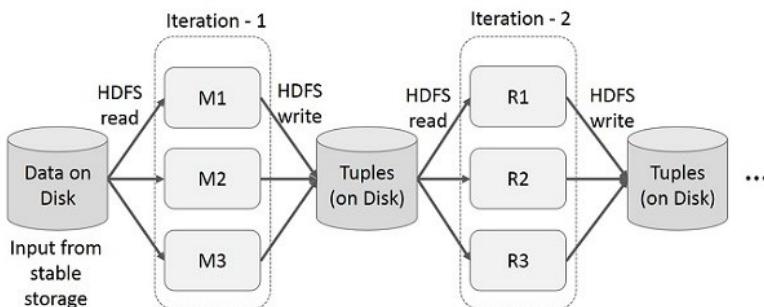
- Spark MLlib
 - omogućava implementaciju algoritama mašinskog učenja nad podacima
 - distribuirani radni okvir
- Spark R
 - omogućava integraciju programskog jezika R sa Apache Spark sistemom

149

149

Apache Spark - RDD

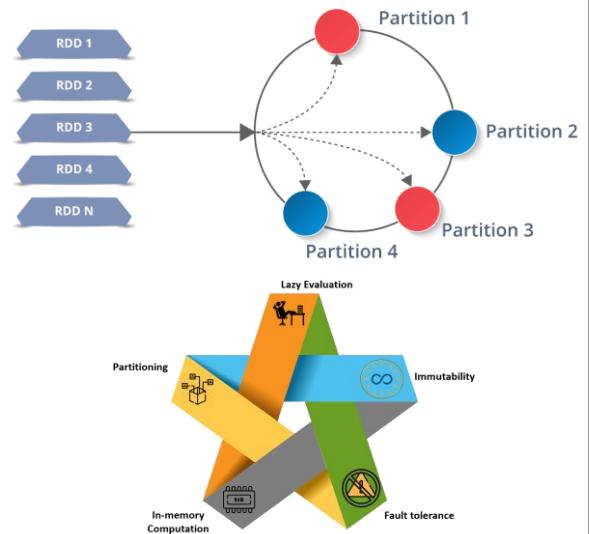
- Problem sa interativnim *map-reduce* programima
 - sporo deljenje memorije
 - upis na disk nakon svakog *map-reduce* koraka
 - previše (za krajnji rezultat) nepotrebnih I/O operacija



150

Apache Spark - RDD

- Distribuirani skup podataka otporan na otkaze
 - engl. *Resilient Distributed Datasets* (RDD)
 - podaci su distribuirani po čvorovima klastera
 - otporan na otkaze usled particonisanja i replikacije podataka
 - mogućnost rekreiranja podataka nakon otkaza
 - nepromenljiv skup podataka
 - nije moguće menjati stanje nakon kreiranja
 - skladišten u radnoj memoriji tokom obrade



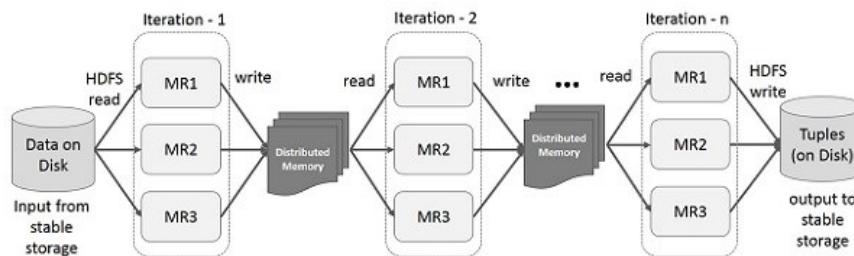
izvor: <https://www.edureka.co/blog/spark-architecture/> i <https://intellipaat.com/blog/tutorial/spark-tutorial/programming-with-rdds/>

151

151

Apache Spark - RDD

- Distribuirani skup podataka otporan na otkaze
 - Eliminacija velikog broja I/O poziva značajno povećava brzinu izvršavanja iterativnih *map-reduce* programa



152

Apache Spark - RDD

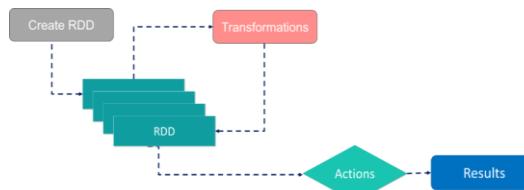
- Distribuirani skup podataka otporan na otkaze
 - omogućava ponovno iskorišćenje međurezultata obrade
 - njihovim snimanjem u radnoj memoriji ili na disku
 - moguće je podešavati particionisanje radi optimizacije skladištenja podataka
 - poseduje veze ka RDD od kojih je nastao
 - primenom niza transformacija
 - pogodno za oporavak skupa podataka u slučaju otkaza
 - dva načina kreiranja RDD-a
 - paralelizacijom kolekcije podataka iz programa
 - referenciranjem strukture sa podacima u eksternom skladištu
 - HDFS, HBase, itd.
 - particije su atomičke jedinice obrade podataka
 - Spark automatski izvršava zadatke nad particijama RDD koji je predmet iste

153

153

Apache Spark - RDD

- Distribuirani skup podataka otporan na otkaze
 - podržava operacije
 - **transformacije** (engl. *transformation*)
 - kreiranje novih RDD
 - odloženo izvršavanje transformacija nad podacima
 - sve tražene transformacije se snimaju i izvršavaju tek kada su podaci potrebni u obradi
 - **obrade / akcije** (engl. *action*) podataka
 - sračunavanja nad podacima radi kreiranja rezultata koji se prikazuju korisniku

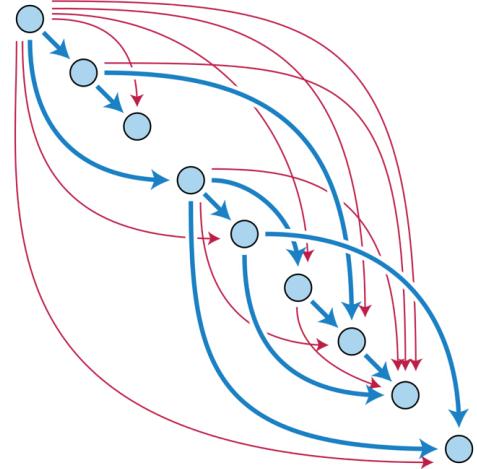
izvor: <https://www.edureka.co/blog/spark-architecture/>

154

154

Apache Spark - DAG

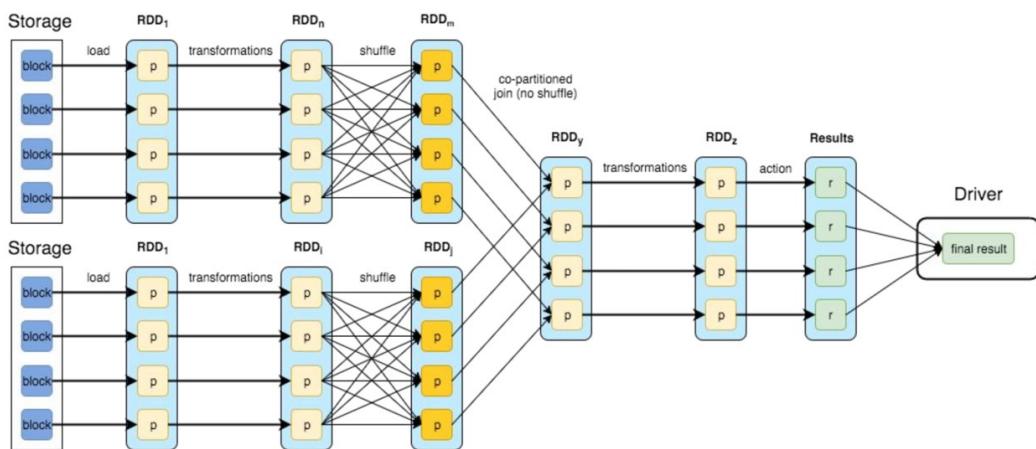
- Usmereni aciklični graf operacija
 - engl. *directed acyclic graph* (DAG)
 - predstavlja niz međusobno povezanih operacija nad podacima
 - predstavlja internu reprezentaciju programa za obradu podataka
 - i osnovu za distribuiranje i raspodelu RDD i zadataka u klasteru
 - za razliku od MapReduce modela podržava postojanje više od dve faze obrade



izvor: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167> 155

155

Apache Spark - DAG



izvor: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167>

156

156

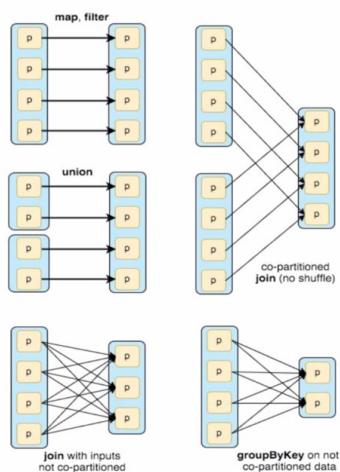
Apache Spark - RDD

- Tipovi međuzavisnosti RDD
 - **direktna zavisnost** (engl. narrow, *pipelineable*)
 - posledica direktnih/uskih transformacija nad podacima
 - svaka particija roditeljskog RDD-a je korišćena u najviše jednoj particiji potomka
 - dozvoljava izvršenje kompletognog stabla particija na jednom čvoru u klasteru
 - oporavak od gubitka particije potomka je jednostavniji
 - usled izračunavanja samo nedostajućih predaka
 - **proširena zavisnost** (engl. wide, *shuffle*)
 - posledica proširenih/širokih transformacija nad podacima
 - više particija potomaka zavisi od jedne particije roditeljskog RDD-a
 - zahteva prebacivanje svih roditeljskih particija prilikom promene čvora u klasteru
 - kompletno ponovno izračunavanje na osnovu svih roditeljskih particija je potrebno u slučaju gubitka particije potomka

157

157

Apache Spark - RDD

izvor: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167>

158

158

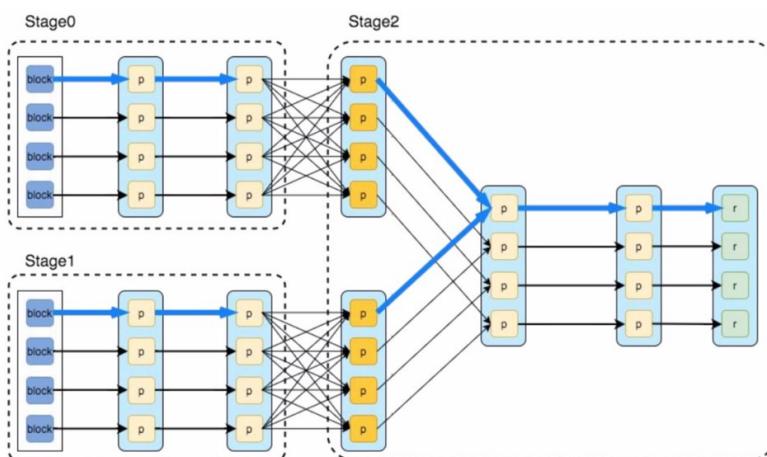
Apache Spark - RDD

- Identifikacija faza izvršavanja
 - na osnovu zavisnosti sa prethodnim RDD-vima
 - krenuvši od poslednjeg RDD-a istoj fazi pripadaju svi RDD sa kojima je u direktnoj zavisnosti
- Ograničenja RDD
 - nepostojanje ugrađenog optimizatora operacija sa RDD
 - sva optimizacija prepustena programeru
 - obrada strukturiranih podataka
 - RDD nema mehanizme da prepozna šemu ulaznog skupa podataka
 - performanse
 - RDD su JVM objekti -> preterana upotreba GC-a i Java serijalizacije
 - Skladištenje
 - u slučaju da RDD ne može stati u radnu memoriju, sledi smeštanje jednog dela na disk

159

159

Apache Spark - RDD

izvor: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167>

160

160

Apache Spark - Dataframe

- Tabelarno organizovan skup podataka
 - engl. *Dataframe*
 - oslanja se na RDD i od njega nasleđuje
 - nepromenljivost
 - distribuirano izvršavanje u radnoj memoriji
 - otpornost na otkaze
 - poboljšanja u odnosu na RDD
 - efikasnije upravljanje memorijom (engl. *Custom Memory Management*)
 - optimizacija upita
 - daje se prednost *DataFrame*-u u odnosu na RDD prilikom obrade **strukturiranih podataka**

161

161

Apache Spark - Dataset

- Spark skup podataka
 - engl. *Spark dataset*
 - predstavljaju proširenja tabelarno organizovanog skupa podataka i RDD
 - pružaju objektno orijentisani interfejs
 - rad sa klasama i objektima
 - kolekcija JVM objekata

162

162

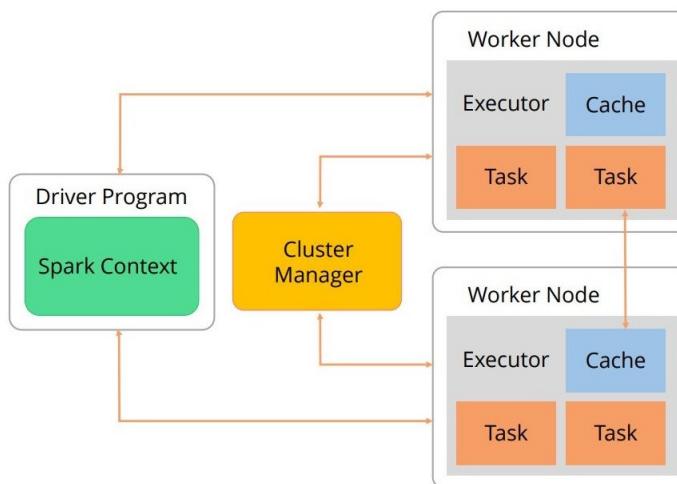
Apache Spark - RDD, Dataframe, Dataset

	RDD	Dataframe	Dataset
šta je?	API niskog nivoa apstrakcije	API visokog nivoa apstrakcije	Kombinacija RDD i Dataframe
optimizacija	Ne podržava	Optimizuje izvršavanje upita	Optimizuje izvršavanje upita
predstava podataka	Particionisani i distribuirani podaci	Kolekcija redova i imenovanih kolona	Kombinacija RDD i Dataframe
prednosti	Jednostavnost API-ja	Postojanje šeme distribuiranih podataka	Unapređeno korišćenje memorije
nepromenljivost podataka i interoperabilnost	RDD uvezuje podatke i omogućava oporavak od greške	Nakon transformacije u tabelarni oblik, nije moguće rekonstruisati originalni objekat	Moguće je rekonstruisati originalni RDD iz skupa podataka
performanse	Ograničene zbog serijalizacije u Javi i GC-a	Značajna unapređenja u odnosu na RDD	Operacije se izvode nad serijalizovanim podacima

163

163

Apache Spark - arhitektura

izvor: <https://www.simplilearn.com/basics-of-apache-spark-tutorial>

164

164

Apache Spark - arhitektura

- Glavni čvor (engl. *master node*)
 - sadrži dajver koji upravlja izvršavanjem aplikacije
 - omogućava kreiranje konteksta (engl. *Spark Context*)
 - zajedno sa kontekstom rastavlja traženi posao na zadatke
- Upravljač klasterom (engl. *cluster manager*)
 - upravlja distribucijom zadataka i RDD-a u klasteru
- Radni čvorovi (engl. *worker node*)
 - izvršavaju zadatke koji su im dodeljeni
 - nad particionisanim RDD
 - vraćaju rezultate nadležnom kontekstu

165

165

Apache Spark - arhitektura

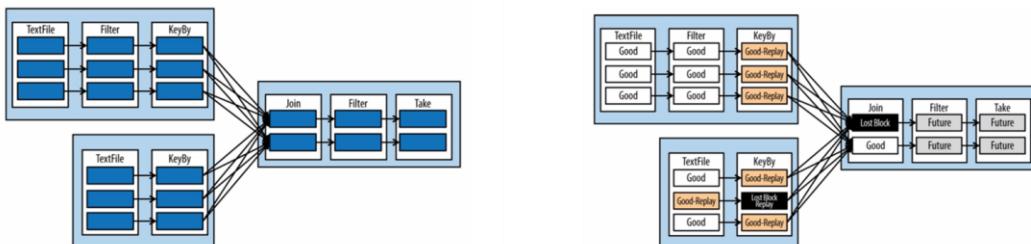
- Izvršavanje poslova
 - **korak 1:** konverzija klijentskog programa u usmereni aciklični graf operacija
 - takođe optimizuje operacije koje je potrebno izvršiti
 - **korak 2:** kreiranje fizičkog plana izvršavanja usmerenog acikličnog grafa operacija
 - sa identifikacijom čvorova u klasteru koji učestvuju u obradi
 - podela posla na manje jedinice - zadatke
 - **korak 3:** identifikacija i zauzimanje potrebnih resursa u klasteru i slanje zadataka na izvršenje
 - dajver upravlja ovim procesom
 - **korak 4:** nadgledanje izvršenja zadataka i preuzimanje rezultata obrade

166

166

Apache Spark - arhitektura

- Otpornost na otkaze
 - svaki RDD poseduje celokupnu istoriju transformacija
 - koje su izvršene nad prethodnim RDD u cilju kreiranja novog
 - počevši od trenutka akvizicije podataka

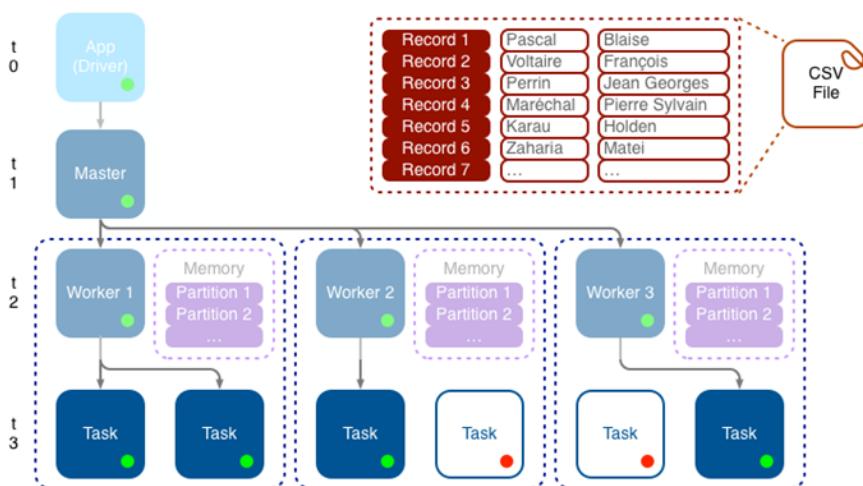


izvor: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167>

167

167

Apache Spark - arhitektura

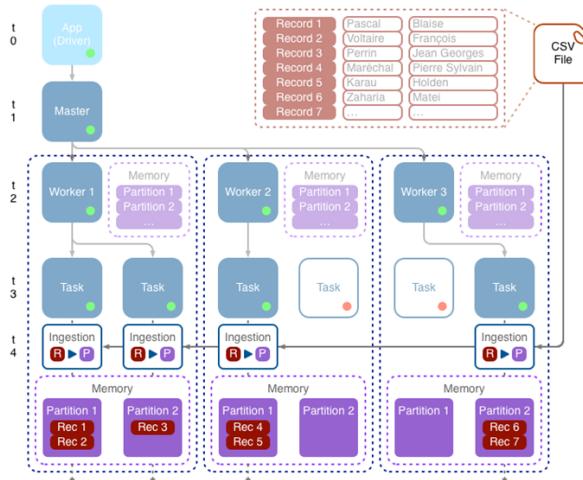


izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-2/v-13/64>

168

168

Apache Spark - arhitektura

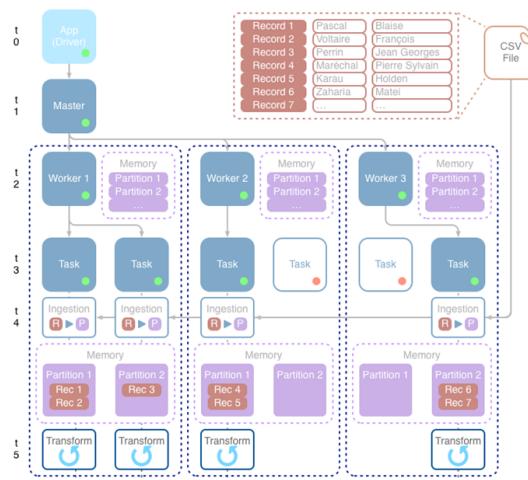


izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-2/v-13/64>

169

169

Apache Spark - arhitektura

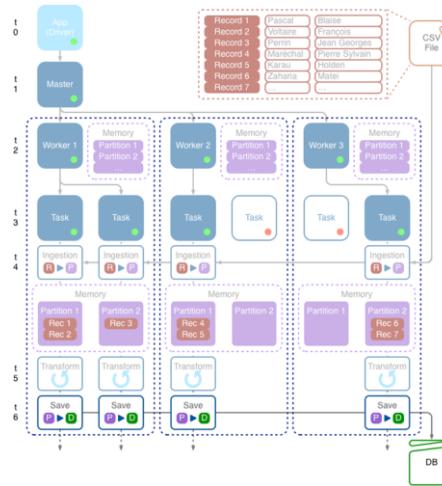


izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-2/v-13/64>

170

170

Apache Spark - arhitektura



izvor: <https://livebook.manning.com/book/spark-in-action-second-edition/chapter-2/v-13/64>

171

171

Apache Spark - primer RDD

- Primer u programskom jeziku Java
 - prebrojati pojavljivanje reči u tekstualnom dokumentu

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");

JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);

counts.saveAsTextFile("hdfs://...");
```

172

172

Apache Spark - primer Dataframe

- Primer u programskom jeziku Java
 - identifikovati sve logove sa greškom u log datoteci

```
// Creates a DataFrame having a single column named "line"
JavaRDD<String> textFile = sc.textFile("hdfs://.../");
JavaRDD<Row> rowRDD = textFile.map(RowFactory::create);
List<StructField> fields = Arrays.asList(
    DataTypes.createStructField("line", DataTypes.StringType, true));
StructType schema = DataTypes.createStructType(fields);
DataFrame df = sqlContext.createDataFrame(rowRDD, schema);

DataFrame errors = df.filter(col("line").like("%ERROR%"));
// Counts all the errors
errors.count();
// Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count();
// Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect();
```

173

173

Pomoćni alati

174

Apache Pig

- Apache Pig
 - podiže nivo apstrakcije za obradu velikih skupova podataka
 - usled previše niskog nivoa na kojem se implementiraju map i reduce koraci
 - poseduje ekspresivnije operatore na višem nivou apstrakcije nego u MapReduce programima
 - namenjen istraživanju podataka u paketnoj obradi podataka
 - često korišćen usled prevelike kompleksnosti i dugog vremena potrebnog da se MapReduce program implementira
 - sastavljen iz dva dela
 - *Pig Latin* - jezik u kojem se implementiraju algoritmi obrade podataka
 - proširiv korisnički definisanim funkcijama
 - izvršno okruženje za *Pig Latin*
 - lokalno okruženje - za izvršavanje na lokalnoj mašini
 - distribuirano izvršavanje - za izvršavanje na Hadoop klastru



.75

175

Apache Pig

- Apache Pig
 - *Pig Latin* programi
 - opisuju tokove podataka
 - koje alat Pig prevodi u izvršive MapReduce programe koje potom i izvršava u sopstvenom izvršnom okruženju
 - predstavljaju sekvensu operacija (transformacija) koje se primenjuju na ulazne podatke u cilju dobijanja izlaznih podataka u odgovarajućem obliku
 - moguće ih je izvršiti samo nad delom svih podataka
 - i na taj način uočiti potencijalne greške pre nego što se krene sa obradom celog skupa podataka
 - generisani izvršni kod nije optimalan kao ručno napisan MapReduce kod
 - sa svakom novom verzijom, razlika je sve manja

176

176

Apache Pig

- Apache Pig i RSUBP
 - upitni jezik
 - Pig Latin je programski jezik za opis tokova podataka
 - SQL je deklarativni programski jezik
 - šema podataka
 - kod alata Pig, šema je opciona
 - strukture podataka
 - alat Pig podržava kompleksne strukture podataka, RDBMS ne
 - transakcije i indeksi
 - RSUBP podržava indekse i transakcije, alat Pig ne

177

177

Apache Pig - primer

- Analize meteoroloških podataka - Apache Pig
 - najveća izmerena temperatura

```
-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
(quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
MAX(filtered_records.temperature);
DUMP max_temp;
```

178

178

Apache Hive

- Apache Hive
 - radni okvir koji treba da omogući funkcionalnosti tradicionalnih skladišta podataka nad Hadoop-om
 - omogućava korišćenje izraza koji liče na SQL
 - mešavina alata Pig i tradicionalnih RDBMS
 - sličnost sa alatom Pig:
 - koristi HDFS kao mehanizam za skladištenje podatka
 - ne podržava brze upite koji koriste indeksne strukture i transakcije
 - sličnost sa RDBMS
 - svi podaci se smještaju u tabele i postoji jasno definisana šema
 - HiveQL je zasnovan na SQL-u



179

Apache Hive

- Apache Hive i RSUBP
 - upitni jezik
 - napravljen na osnovu SQL-92 standarda
 - ali ne podržava apsolutno sve operacije
 - šema podataka
 - RSUBP zahteva da podaci odgovaraju šemi prilikom učitavanja (engl. *schema on load*)
 - Hive zahteva da podaci odgovaraju šemi prilikom čitanja (engl. *schema on read*)
 - strukture podataka
 - alat Hive podržava kompleksne strukture podataka, RDBMS ne
 - ažuriranja, transakcije i indeksi
 - Hive ne podržava modifikaciju i brisanje postojećih podataka
 - Hive podržava samo kompaktne i *bitmap* indekse
 - zaključavanje je moguće na nivou particije i tabele

180

180

Apache Hive - primer

- Analize meteoroloških podataka - Apache Hive
 - najveća izmerena temperatura

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;

SELECT year, MAX(temperature)
FROM records
WHERE temperature != 9999
AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)
GROUP BY year;
```

181

181

Apache HBase

- Apache HBase
 - distribuirana, kolonski orijentisana baza podataka
 - koja koristi HDFS kao skladište podataka
 - koristi se kada je u okviru Hadoop rešenja potrebno u realnom vremenu dozvoliti čitanje i pisanje slučajno odabranog sloga
 - linearno skalira
 - dodavanjem novih čvorova u klaster
 - za razliku od RSUBP
 - podržava distribuirano skladištenje veoma velikih, retko popunjениh tabela
 - na svima dostupnom hardveru



182

182

Apache HBase

- Apache HBase i RSUBP
 - HBase
 - ne podržava indekse
 - omogućava automatsko particionisanje tabela
 - automatsko linearno skaliranje
 - radi na svima dostupnom hardveru
 - omogućava paketnu obradu
 - u potpunosti paralelnu
 - izvršavanjem MapReduce programa nad skladištem

183

183

Apache Zookeeper

- Apache Zookeeper
 - distribuirani sistem za koordinaciju servisa u Hadoop okruženju
 - sastoji se od skupa alata za upravljanje parcijalnim otkazima
 - osobine:
 - jednostavnost
 - pojednostavljen sistem datoteka koji pruža osnovne usluge rada sa datotekama
 - ekspresivnost
 - poseduje primitive za kreiranje velikog broja struktura podataka i protokola
 - npr. distribuirani redovi ...
 - visoka dostupnost
 - omogućava komunikaciju slabo spregnutih modula
 - moduli ne moraju da budu svesni jedni drugih
 - implementiran u vidu biblioteke



184

184

Literatura

- Tom White: Hadoop - The Definitive Guide, O'Reilly
- Mark Grover, Ted Malaska, Jonathan Seidman, Gwen Shapira: Hadoop Application Architectures, O'Reilly
- Ilya Katsov, MapReduce patterns, Online:
<https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/>
- Matei Zaharia, Bill Chambers, Spark: The Definitive Guide, O'Reilly
- Ty Shaikh, Batch Processing — Apache Spark, Online: <https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167>

185