Studija slučaja Hadoop aplikacija nad skupom podataka NYSE

Sadržaj

Uvod

Knjiga "Big data with Hadoop MapReduce", kojom se vodim, kako bih objasnio Hadoop koncepte, pokriva osnovne terminologije i koncepte skupa velikih podataka, distribuiranog računarstva, kao i unutrašnji rad MapReduce koncepta. Akcenat je više na Hadoop v2 u poređenju sa Hadoop v1 zarad poštovanja trenda, mada je Hadoop v3 već dostupan. Skupovi velikih podataka se odnose na kolekciju skupova podataka koji su ogromni ili sa ogromnim protokom ili sa različitim tipovima podataka ili bilo kojom od ovih kombinacija, koje nadmašuju tradicionalnu memoriju, računarsku i algoritamsku sposobnost skladištenja, obrade, analize i razumevanja na ekonomski isplativ način."

Postoje dva načina primene Hadoop-a, implementacija jednog čvora (Hadoop u jednoj mašini) ili implementacija sa više čvorova (Hadoop na više od jednog čvora). Implementacija jednog čvora može biti u lokalnom (nijedan Hadoop demon ne radi u ovom režimu, a za sekvence izvršavanja brine Hadoop framework) ili pseudo-distribuiranom (MapRecduce i HDFS demoni se pokreću u zasebnom JVM-u simulirajući klaster unutar čvora korišćenjem više od jednog JVM-a.) modu.

Hadoop distribuirani sistem datoteka (HDFS) je primarni sistem za skladištenje podataka koji koriste Hadoop aplikacije. HDFS koristi arhitekturu "NameNode" i "DataNode" za implementaciju distribuiranog sistema datoteka koji obezbeđuje pristup podacima visokih performansi u visoko skalabilnim Hadoop klasterima. HDFS omogućava brz prenos podataka između računarskih čvorova. Usko je povezan sa MapReduce framework-om za obradu podataka koji filtrira i deli posao između čvorova u klasteru, i organizuje i kondenzuje rezultate u kohezivan odgovor na upit. Podaci se razbijaju u zasebne blokove i distribuiraju različitim čvorovima u klasteru. HDFS je dizajniran da bude veoma otporan na greške. Sistem datoteka kopira svaki deo podataka više puta i distribuira kopije na pojedinačne čvorove, postavljajući najmanje jednu kopiju na drugi serverski rack.

NYSE skup podataka

Njujorška berza, najpoznatija trgovačka kompanija na svetu, beleži sve transakcije po akciji koje su izlistane i aktivno trgovane na istoj. Parametri koji se beleže podrazumevaju, trenutnu cenu, cene otvaranja, zatvaranja, kao i prilagođena cena na zatvaranju. Takođe se beleži i volumen, ili količina transakcija u periodu od interesa, bilo to dan mesec ili sat. Svi indikatori koji se koriste od strane analitičara, mogu da se izvedu korišćenjem pomentuih podataka.

NYSE set podataka, koji je korišćen u ovoj knjizi [2], najpre kopiramo na HDFS sa lokalnog diska pomoću Hadoop komande u /input direktorijum:

hdfs dfs -copyFromLocal /nyse_data /input

Hadoop aplikacije razvijene za NYSE skup podataka

Vežba 1: Prosečan obim trgovanja za svaku akciju mesečno, NYSE skup podataka

S obzirom da ne postoji "datatype" u MapReduce, ulazni podatci moraju biti konvertovani u odgovarajući tip podataka. Konstruisana je java klasa sa get i set metodama primenjenim na NYSE skupu

podataka, čime jekreiran „Plain Old Java Object“ (POJO). POJO klasa može biti korišćena od strane bilo koje java aplikacije, jer nije vezana niti za jedan framework.

U MapReduce procesu, pre prosleđivanja podataka u maper, podatke treba prvo pretvoriti u parove ključ/vrednost pošto maper razume samo takve parove podataka. Podaci koje će obraditi pojedinačni maper su predstavljeni InputSplitom. RecordReader komunicira sa InputSplitom i pretvara Split u parove ključ/vrednost koji su pogodni za čitanje od strane mapera. Podrazumevano, RecordReader koristi TektInputFormat za pretvaranje podataka u par ključ/vrednost i komunicira sa InputSplit-om sve dok se čitanje datoteke ne završi. Funkcija mapiranja obrađuje određeni par ključ/vrednost i emituje određeni broj parova ključ/vrednost, a funkcija reduktora obrađuje vrednosti grupisane istim ključem i emituje drugi skup parova ključ/vrednost kao izlaz. Tipovi izlaza mapiranja treba da odgovaraju tipovima ulaza reduktora. Razvijeni su korisnički definisani tipovi podataka ključ i vrednost, a zatim se simbol akcije filtrira pomoću dva ključa, datum i simbol. Potom se za tip vrednost izabrane akcije, sumira volumen i prati totalni volumen. Klase implementiraju „WritableComparable“, pri čemu je potrebno implementirati „readFields“ i „write“ iz „Writable“ interfejsa. Kreirane su funkcije hashCode(), equals() i toString(), ali moraju biti pregažene jer već postoje u javi. Funkcija komparacije, compareTo(), takođe mora biti pregažena, jer nije u potpunosti paralelizovana u javi. MapReduce preko svog API interfejsa pruža RawComparator, koji je u potpunosti paralelizovan.

Nakon kreiranja ključ/vrednost parova, kreirana je funkcija mapiranja, koja za ulaz ima čitav red iz NYSE .csv fajla, a na izlazu daje modifikovani ključ sa dve tekstualne vrednosti, datum i simbol akcije. Izlazna vrednosti su suma i totalna suma volumena.

Na velikom skupu podataka, maper generiše velike komade međupodataka i ovi međupodaci se prosleđuju reduktoru na dalju obradu, što može dovesti do ogromnog zagušenja mreže. MapReduce framevork obezbeđuje funkciju „Combiner“, koja igra ključnu ulogu u smanjenju zagušenja mreže. kombinator u MapReduce-u je takođe poznat kao „mini-reduktor“. Primarni posao kombinatora je da obradi izlazne podatke iz mapera, pre nego što ih prosledi reduktoru, samim tim ubrzava izvršavanje reduktora. Startuje posle mapera i pre reduktora i njegova upotreba je opciona. Tako se u kombinatoru sumira totalni volumen, dok će reduktor obraditi prosečni volumen.

Izlaz reduktora je konačni izlaz, koji se čuva u HDFS-u. Obično, u reduktoru, vršimo vrstu izračunavanja agregacijom ili sumiranjem. Reduktor uzima parove ključ/vrednost koji proizvodi maper kao ulaz i pokreće funkciju redukovanja na svakom od njih. Ovi podaci se mogu agregirati, filtrirati i kombinovati na više načina za širok spektar obrade. Nakon obrade generiše izlaz (nula ili više para ključ/vrednost). Reduktori rade paralelno jer su nezavisni jedan od drugog, a o broju reduktora dlučuje sam programer. Svaki reduktor ima 3 faze. U prvoj fazi, sortirani izlaz iz mapera je ulaz u reduktor. Ova faza se naziva Shuffle i ona, uz pomoć HTTP-a, preuzima relevantnu particiju izlaza svih mapera. U drugoj fazi, zvanoj Sort, ulaz iz različitih mapera se ponovo sortira na osnovu sličnih ključeva u različitim maperima. Faze Shuffle i Sort se dešavaju istovremeno. U završnoj trećoj fazi, zvanoj Reduce, nakon mešanja i sortiranja, reduktor agregira parove ključ/vrednost, a zatim zapisuje rezultat u HDFS. Izlaz reduktora pritom nije sortiran.

Iako su implementacije mapera i reduktora sve što je potrebno za obavljanje MapReduce posla, postoji još jedan deo koda koji je neophodan, a to je drajver (driver class) koji komunicira sa Hadoop framework-om i definiše elemente konfiguracije potrebne za pokretanje MapReduce posla. Definiše se koje klase mapera i reduktora Hadoop treba da koristi, kao i mesta na kojima treba da pronađe ulazne podatke i u kom formatu, ali takođe i gde da postavi izlazne podatke i kako ih formatirati. Postoji dodatni niz drugih konfiguracionih opcija koje se mogu podesiti u klasi drajvera.

Vežba 2: Rad sa API-jima sistema datoteka (org.apache.hadoop.fs)

Uobičajeni zadatak u Hadoop-u je interakcija sa njegovim sistemom datoteka HDFS, bilo za prelistavanje, dodavanje novih, obradu, raščlanjivanje rezultata ili brisanje datoteka. Hadoop nudi Java API, a ista funkcionalnost se može ostvariti i preko komandne linije (uz odličnu integraciju sa Linux-om i njegovim Shell komandama).

hdfs dfs -cat /data/input.txt

Možemo da koristimo API u java programu za interakciju sa pokrenutim Hadoop demonima i prikaz informacija. Da bi se to postiglo, java program treba da koristi HDFS URL za povezivanje sa pokrenutim demonima.

yarn jar job.jar FileSystemAPI.GetFiles /input

API sistemskih datoteka je iskorišćen za štampanje naziva i putanja fajlova u folderu /input koji se nalazi na HDFS-u, a zatim i za konkatenaciju većeg broja fajlova u jedan, unutrar HDFS-a, potom i otpremanje u folder po izboru (pritom ne brišući izvorne fajlove).

Vežba 3: Filtriranje ulaznih datoteka i ulaznih zapisa

Filtriranje ulaznih datoteka koje se nalaze unutar HDFS-a je izvršeno uz pomoć API sistemskih datoteka, prostim izborom putanje do ulazne datoteke.

Filtriranje ulaznih zapisa, tj. simbola akcija na Njujorškoj berzi (stock ticker), je izvršeno u samoj funkciji mapiranja, jednostavnim kondicionim uslovom, “hardcoded”

if (parser.getStockTicker().equals(“GME”))

ili sa mogućnošću izbora simbola kroz argumente komandne linije

if(parser.getStockTicker().equals(context.getConfiguration().get(“filter.by.stockTicker”))))

yarn jar job.jar AvgStockMapperWithProperties -Dfilter.by.stockTicker=GME /input /output

Izbor simbola preko komandne linije jeste bolja opcija, jer je u suprotnom simbol “hardcoded” unutar map funcije, pa bismo svaki put kada moramo da izmenimo simbol morali da konvertujemo .java izvoni fajl u .jar.

Filtriranje unutar setup metode je najbolje rešenje, jer se setup metoda poziva jednom po JVM-u za razliku od funkcija mapiranja i redukcije, koje se pozivaju za svaki pojedinačni zapis. Nakon filtriranja po simbolu unutar setup metode, u map funkciji se pomoću ključa (mesec, simbol), prikupljaju zapisi za filtrirani simbol.

Vežba 4: Korisnički definisani particioner

Particioner funkcioniše kao uslov u obradi ulaznog skupa podataka. Faza particije se odvija nakon

faze mapiranja i pre faze redukovanja. Broj particionera je jednak broju reduktora. To znači da će particioner podeliti podatke prema broju reduktora. Prema tome, podatke prosleđene iz jednog particionera obrađuje jedan reduktor. Particioner deli parove ključ/vrednost posrednih izlaza mape, koristeći korisnički definisan uslov, koji funkcioniše kao heš funkcija.

U prethodnim slučajevima za filtriranje unutar map funkcije su korišćena 2 ključa, datum i simbol akcije. Sada se koristi samo drugi ključ, simbol, čime je omogućeno da se svi zapisi po filtriranom simbolu akcije upisuju u jednu izlaznu datoteku. Prednost korišćenja korisnički definisanog ključa, više od jednog, je što sada istovremeno možemo filtrirati zapise za više simbola, pri čemu dobijamo poseban izlaz u vidu datoteke po simbolu kao rezultat. Ako imamo 2 ili više simbola, svakom od simbola će biti dodeljen poseban reduktor (reducer). Podrazumevani particioner jednostavno koristi hashcode() metod ključa i izračunava particiju. Ovo daje priliku da implementiramo svoj hascode() kako bi podesili način na koji će ključevi biti particionisani.

Vežba 5: Razvijanje prilagođenog particionera

Ako metoda hashCode() ne distribuira ravnomerno podatke drugih ključeva u opsegu particija, tada podaci neće biti ravnomerno poslati reduktorima. Loše particionisanje podataka znači da će neki reduktori imati više unosa podataka od drugih, tj. imaće više posla od drugih reduktora. Dakle, ceo posao će čekati da jedan reduktor završi svoj izuzetno veliki deo opterećenja, otud i potreba za prilagođenim particionerima. Prilagođeni particioneri se pišu u MapReduce poslu kad god postoji zahtev da se skup podataka podeli više od dva puta. Particioner je proces koji omogućava da se rezultati sačuvaju u različitim reduktorima, na osnovu uslova korisnika. Particioniranjem po ključu, možemo garantovati da će zapisi za isti ključ ići u isti reduktor. Particioner obezbeđuje da samo jedan reduktor primi sve zapise za taj određeni ključ. Na primer, ako postoji zahtev da se pronađe najstarija osoba, sa svakog leta avio-kompanije, moramo da koristimo prilagođeni particioner.

Ideja je u tome da, prilagođeni particioner, može koristiti za grupisanje podataka koje treba sortirati. Ako treba sortirati po mesecima, može se podeliti svaki podatak sa stringom „februar“ u 2. particiju, „decembar“ u 12. itd. Izmenom koda može se precizirati da određeni simbol ili određeni datum trgovine idu na određeni reduktor. Ovde, umesto da komentarišemo hashCode drugog polja u ključu, kreiran je prilagođeni heš particioner koji uzima samo drugo polje prilagođenog ključa.

Vežba 6: Odluka o particiji na osnovu argumenata vremena izvršavanja

Nadogradnja prethodnog slučaja prilagođenog particionera, izvršena je u vidu, dodate mogućnosti argumenata putem komandne linije

yarn jar job.jar AvgStockMonthPartitionerDriver –D partition.by=ticker /nyse_data/.* /out*

čime je omogućeno fleksibilno particioniranje, uz slobodan izbor korisnika, da to bude prema simbolu (ticker), ili mesecu (trademonth).

Vežba 7: Particionisanje i sortiranje na osnovu prvog polja u prilagođenom ključu pomoću komparatora

Proces prenosa podataka iz mapera u reduktore poznat je kao mešanje, tj. proces kojim sistem vrši sortiranje i prenosi izlaz mape reduktoru kao ulaz. MapReduce shuffle faza je neophodna za

reduktore, inače ne bi imali nikakav ulaz. Pošto mešanje može da počne čak i pre nego što se faza mape završi, ovo štedi vreme i završava zadatke za kraće vreme.

Ključevi koje generiše maper automatski se sortiraju pomoću MapReduce Framevork-a, odnosno pre pokretanja reduktora, svi parovi ključ/vrednost koji su generisani od strane mapera se sortiraju po ključu, a ne po vrednosti. Vrednosti prosleđene svakom reduktoru se ne sortiraju već mogu biti bilo kojim redosledom. Sortiranje u Hadoop-u pomaže reduktoru da lako razlikuje kada treba da počne novi zadatak redukcije. Ovo štedi vreme reduktoru. Reduktor pokreće novi zadatak redukcije kada se sledeći ključ u sortiranim ulaznim podacima razlikuje od prethodnog. Svaki zadatak redukcije uzima parove ključ/vrednost kao ulaz i generiše par ključ/vrednost kao izlaz.

Ako želimo da sortiramo vrednosti reduktora, onda se koristi tehnika sekundarnog sortiranja, koja nam omogućava da sortiramo vrednosti (u rastućem ili opadajućem redosledu) prosleđene svakom reduktoru.

Vežba 8: Izdvajanje 3 najviše trgovane akcije po danu iz NYSE skupova podataka

U ovom primeru reduktor neće biti korišćen za agregaciju, niti sumiranje, kao u prethodnim slučajevima, već za rangiranje. Reduktor dobija ulazne podatke, sortirane po datumu i volumenu u opadajućem redosledu, ali se grupisanje odvija samo prema datumu. Ova funkcionalnost se postiže podešavanjem particionera i komparatora, koji se logički nalaze, između funkcija mapiranja i redukcije.

Ulazni ključ u funkciji mapiranja je red u .csv fajlu, a ulazna vrednost podaci trenutne linije. Izlazni ključ, tipa Long, je par formatiranog datuma i volumena, a vrednost običan text sa svim informacijama o simbolu akcije.

Postoje dva načina upoređivanja ključeva: implementacijom “WritableComparable” interfejs-a ili primenom “RawComparator” interfejs-a. U prvom pristupu, vršis se poređenje deserijalizovanih objekte, dok se u drugom pristupu, upoređivanje ključeva zapravo odvija poređenjem njima odgovarajućih bajtova.

Particioner, vrši grupisanje podataka, prema datumu, i to u rastućem redosledu. Ovim je garantovano da će izlaz iz particionera za isti ključ ići u isti komparator na dalje sortiranje. Komparator sortiranja odlučuje kako će ključevi biti sortirani. Komparator prema ključu, datum, sortira vrednosti volumena u opadajućem redosledu za svaki dan. Grupni komparator odlučuje koji izlazni ključevi će biti grupisani u jedan ključ, a naravno i sve kolekcije vrednosti će takođe biti grupisane, dakle određuje koji izlazni ključevi idu na koji reduktor.

Kako je reduktoru, nakon svih prethodnih modifikacija posao znatno olakšan (na ulazu dobija podatke o akciji sortirane opadajućim redosledom, grupisane prema datumu), sve što treba je da se obrade prva 3 zapisa, zarad izdvajanje 3 najviše trgovane akcije po danu.

Vežba 9: Kompresija

Kompresovanje podataka u HDFS-u omogućava bolju hardversku alokaciju, jer komprimovani podaci često iznose 25% veličine originalnih podataka, što je ogromna ušteda prostora imajući na umu za koje količine podataka je Hadoop skrojen. Samim tim što znamo, da su MapReduce poslovi skoro uvek IO vezani tj. ograničeni, skladištenje kompresovanih podataka znači da je potrebno manje ukupne

IO propusnosti, iz čega sledi da će poslovi raditi brže. Međutim, postoje dva problema: neki formati kompresije se ne mogu podeliti za paralelnu obradu, a drugi su toliko spori u dekompresiji da poslovi postanu CPU ograničeni, eliminišući samim tim prethodne dobitke na IO propusnosti. Postoji veći broj kodeka za kompresiju, dostupnih u hadoop-u: bzip2, gzip, lz4, lzo, snappy. Međutim moguće je implementirati ili ponovo koristiti već implementirane algoritme. Kao primer, LZMA algoritam, koji poseduje brzinu kompresije sličnu bzip2 (ili još bolju), ali i mnogo veću brzinu dekompresije. Dostupnost kodeka se može proveriti sledećom komandom:

hadoop checknative –a

Da bismo precizirali koji od kodeka želimo da koristimo, moramo editovati core-site.xml i mapred-site.xml fajlove, ili da navedemo parametre unutar .java koda (naziv kodeka se dodaje na naziv fajla).

Vežba 10: Brojači

Brojači pružaju način za merenje napretka ili broja operacija koje se dešavaju u okviru posla mapiranja i redukcije. Brojači u Hadoop MapReduce-u su koristan kanal za prikupljanje statistike o MapReduce poslu ili za kontrolu kvaliteta ili za nivo aplikacije. Takođe su korisni za dijagnozu problema. Brojači predstavljaju Hadoop globalne brojače, definisane od strane MapReduce framework-a ili aplikacija. Svaki Hadoop brojač je imenovan „enumom“ i ima “long” kao vrednost. Brojači su grupisani, pri čemu se svaka grupa sastoji, od brojača iz određene Enum klase. Hadoop brojači omogućavaju validaciju: da je tačan broj bajtova pročitan i zapisan, tačan broj zadataka pokrenut, količina potrošnje CPU-a i memorije odgovara trenutnom poslu i čvorovima klastera. Sintaksa pozivanja brojača preko komandne linij je sledeća:

mapred job -counter jobid groupname countername

ili preko API interfejsa:

yarn jar job.jar CounterDriver jobid groupname countername

Brojači su podeljeni u grupe. Svaka grupa sadrži brojače zadataka (koji se ažuriraju kako napreduje zadatak) ili brojače poslova (koji se ažuriraju kao napredak posla). Hadoop brojač zadataka prikuplja specifične informacije (kao što je broj pročitanih i napisanih zapisa) o zadacima tokom njegovog izvršavanja. Na primer, brojač koji broji ulazne zapise koje je pročitao svaki zadatak mapiranja. FileSystem brojači prikupljaju informacije kao što je broj bajtova koje čita i upisuje FileSystem. FileInputFormat brojači prikupljaju informacije o broju bajtova koje čitaju zadaci mapiranja preko FileInputFormat-a. FileOutputFormat brojači prikupljaju informacije o broju bajtova napisanih od strane zadataka mapiranja ili reduktora preko FileOutputFormat-a.

Korisnički definisan brojač korišćenjem enum-a je kreiran zarad pronalaženja dana bez trgovine. Najbolje mesto za kreiranje korisnički definisanog brojača je u reduktoru ili nakon završetka posla u glavnoj metodi. Za kreirali brojače u toku rada, koristi se dinamički brojač.

Vežba 11: Spajanje skupa podataka o trgovini sa skupom podataka liste zaglavlja kompanija

Lista zaglavlja kompanija, koja se nalazi na lokalnom disku, se dovodi na ulaz pomoću –files opcije i spaja sa datotekama iz /input foldera unutar HDFS-a.

yarn jar job.jar AvgStockVolMonthDriver –files companylist_noheader.csv /input /out

Hadoop “Distributed cache“ metoda “adCacheFile()” uzima putanju do fajla, iz ulaznih argumenata i konvertuje u URI format, koji se konačno koristi za dodavanje fajla unutar Hadoop distribuirane keš memorije.

DistributedCache.addCacheArchive(new URI("/input.txt", job);

Kada korisnik pokrene posao, Hadoop kopira datoteku kao argument u –files, -archives, -libjars u HDFS. Pre pokretanja zadatka, NodeManager kopira ove datoteke sa HDFS-a na lokalni disk — keš. To radi tako da zadatak sada može da pristupi ovim datotekama. U ovom trenutku, datoteka se označava kao lokalizovana. Sa tačke gledišta zadatka, datoteke su simbolički povezane sa radnim direktorijumom zadatka. Datoteke navedene pod opcijom –libjars se kopiraju u putanju klase zadatka pre nego što započnu izvršavanje.

NodeManager održava brojanje za broj zadataka koristeći ove lokalizovane datoteke. Pre pokretanja zadatka brojač se povećava za jedan. I nakon što se zadatak završi, brojač se smanjuje za jedan. Datoteka je podobna za brisanje samo kada nema zadataka koji je koriste. Datoteka se briše kada veličina keša pređe određeno ograničenje kao što je 10 GB (podrazumevano). Hadoop briše datoteke da bi napravio mesta za druge datoteke koje se trenutno koriste. Datoteka se briše pravilom najmanje korišćenog algoritma, a veličina keš memorije može se podešavati po potrebi.

Kada se fajl/folder doda u distribuirani keš, datoteka je dostupna svim maperima. Ovo je jedna od tehnika optimizacije u hadoop-u ako je datoteka male veličine. Možete se učiniti dostupnim svim čvorovima za dosta brži pristup podacima. Postoji još nekoliko tehnika optimizacije koje pomažu u optimizaciji poslova u MapReduce-u. Korišćenje kombinatora između mapera i reduktora, korišćenje LZO kompresije (nemaju Java implementaciju, ali ima implementaciju u C programskom jeziku), kao i pravilno podešavanje broja MapReduce zadataka.

Literatura

1. Big data with Hadoop MapReduce: a classroom approach - Rathinaraja Jeyaraj, Ganeshkumar Pugalendhi, Anand Paul
2. https://github.com/rathinaraja/MapReduce
3. Map Reduce-Hadoop - Prof.dr Natalija Stojanovic
4. https://hadoop.apache.org/docs/stable
5. Hadoop: The Definitive Guide - Tom White