Distribuirane baze podataka i Couchbase kao primer jedne takve baze podataka

Seminarski rad

Uroš Vukić

Broj indeksa: 1086

Sistemi za upravljanje bazama podataka

Elektronski fakultet u Niš

Sadržaj

1 Uvod

Iz godine u godinu povećava se količina podataka generisanih od strane ljudi (ovaj rast je eksponencijalne prirode). Pored ovoga, očekivanja od savremenih informacionih sistema rastu u svakom smislu. Jako je skupo (možda čak i nemoguće) napraviti centralizovan sistem koji će ispuniti sve zahteve tržišta. Iz tog razloga se sve više pribegava distribuiranoj implementaciji sistema pa i sistema za upravljanje bazama podataka. Prednosti jednog ovakvog sistema su sledeće:

- Velika količina glavne memorije
- Velika količina sekundarne memorije za trajno čuvanje podataka sa veoma brzim I/O operacijama
- Veliki broj procesnih jedinica i velika kompjuterska moć sistema (broj jezgara može biti i do nekoliko stotina/hiljada)
- Brza mrežna infrastruktura koja omogućava povezivanje klijenata uz jako malu latenciju (celokupna mreža gotovo nikad ne može otkazati)

2 Koncept distribuirane baze podataka (DDB)

Distribuirana baza podataka (distributed database – DDB) je kolekcija većeg broja logički povezanih baza podataka koje su povezane pomoću računarske mreže i kojima upravlja distribuirani sistem za upravljanje bazama podatka (ditributed database management system – DDBMS) pri tom čineći distribuciju transparentnu za krajnjeg korisnika.

2.1 Razlika između distribuiranih (DDB) i multi-procesorskih sistema

Postoji razlika između distribuiranih baza podataka i multi-procesorskih sistema koji koriste deljenu memoriju (primarnu ili sekundarnu). Da bi baza podataka bila distribuirana ona mora da zadovoljava sledeće uslove:

- Veza između čvorova distribuirane baze podataka se ostvaruje pomoću računarske mreže. Veći broj računara na jednom mestu nazivamo sajt (site) ili čvorovi (node). Ovi site-ovi/čvorovi moraju biti povezani računarskom mrežom kako bi razmenjivali podatke i naredbe međusobno.
- Postoji logička povezanost između baza podataka.
- Nepostojanje ograničenja homogenosti nad čvorovima koji učestvuju u vezi. Nije neophodno da svi čvorovi imaju identične podatke, hardver i softver.

Čvorovi mogu biti relativno blizu (npr. u istoj zgradi) i povezani LAN (Local Area Network) mrežom, ili se mogu nalaziti na velikoj međusobnoj udaljenosti i povezani WAN (Wide Area Network) mrežom.

2.2 Transparentnost

Transparentnost se zasniva na ideji skrivanja implementacije od krajnjeg korisnika. Visoko transparentni sistemi omogućavaju veliku fleksibilnost jer ne zahtevaju poznavanje unutrašnjosti samog sistema za uspešno korišćenje. Kod centralizovanih sistema baza podataka, transparentnost se uglavnom odnosi na logičku i fizičku nezavisnost podataka obezbeđenu programerima aplikacije. Sa druge strane, kod DDB podaci i softver su distribuirani na više lokacija koje su povezane računarskom mrežom. Iz tog razloga potrebne su nam nove vrste transparentnosti.

- Organizaciona transparentnost (distribuciona ili mrežna transparentnost) – Odnosi se na oslobađanje korisnika od detalja izvršenja operacija preko mreže i raspodele podataka u distribuiranom sistemu. Može se podeliti u dve podvrste – lokaciona transparentnost i transparentnost imena. Lokaciona transparentnost se odnosi na činjenicu da su komande/operacije upućene sistemu nezavisne od lokacije podataka i lokacije čvora na kome će biti izvršene. Transparentnost imena se označava da kada se objektu jednom dodeli ime, tom objektu se može pristupiti nedvosmisleno bez navođenja gde se objekat nalazi.
- Replikaciona transparentnost – Kopije istog podatka mogu biti čuvane na različitim čvorovima radi postizanja veće dostupnosti i pouzdanosti. Replikaciona transparentnost skriva informaciju o postojanju replika podataka.
- Fragmentaciona transparentnost – Postoje dve vrste fragmentacije relacije – horizontalna i vertikalna fragmentacija. Fragmentaciona transparentnost skriva informacije o postojanju

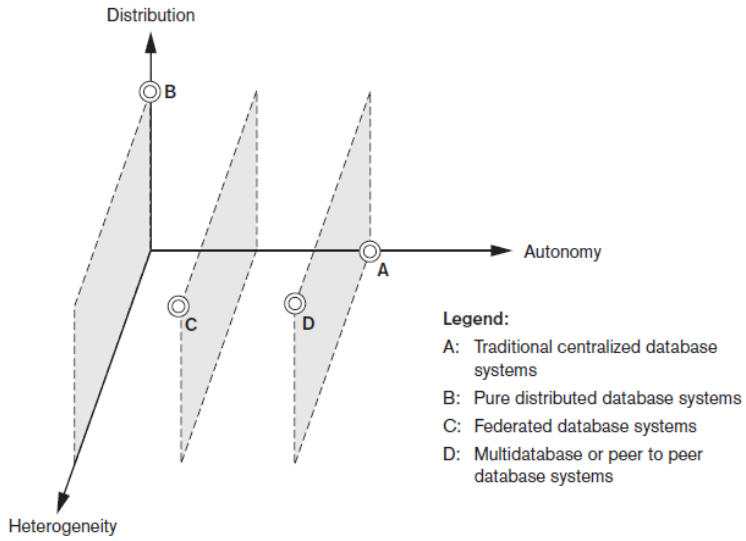
fragmenta od korisnika i prevodi upit prosleđen od strane korisnika u upite nad pojedinačnim fragmentima.

- Transparentnost dizajna – Ova transparentnost skriva saznanje o tome kako je dizajnirana distribuirana baza podataka.
- Izvršna transparentnost – Ova transparentnost skriva saznanje o tome gde se transakcija u distribuiranoj bazi izvršava.

2.3 Tipovi distribuiranih baza podataka

Pojam distribuirane baze podataka može opisivati raznovrsne sisteme koji se razlikuju po različitim aspektima. Jedan od aspekata je stepen homogenosti DDBMS softvera. Ako svi serveri koriste isti softver i svi klijenti koriste isti softver onda se DDBMS naziva homogenim sistemom. U suprotnom je heterogen sistem. Drugi aspekt je stepen lokalne autonomnosti. Ukoliko lokalni čvor (site) ne može da funkcioniše kao samostalni DBMS, onda takav sistem nema lokalnu autonomnost. Sa druge strane ako se lokalnom čvoru može pristupiti pomoću lokalnih transakcija, onda ovakav sistem ima određen nivo lokalne autonomnosti. Na osnovu stepena distribuiranja, autonomije i heterogenosti možemo uočiti sledeće vrste distribuiranih sistema:

A. Tradicionalne centralizovane sisteme – ovi sistemi imaju kompletnu autonomiju, ali nisu distribuirane i heterogene.
B. Potpuno distribuirane baze podataka– elementi ovog sistema nemaju nikakvu autonomiju. Postoji jedna konceptualna šema i sav pristup se obavlja kroz DDBMS.
C. Federativne baze podataka (FDBS) – elementi ovog sistema su autonomni i nezavisni. Postoji globalni pogled (view) ili šema federacije baza podataka koju deli veći broj aplikacija.
D. Sistemi više baza podataka – elementi ovih sistema imaju potpunu lokalnu autonomiju. Ne postoji globalna šema već se ona interaktivno kreira po potrebi aplikacija.

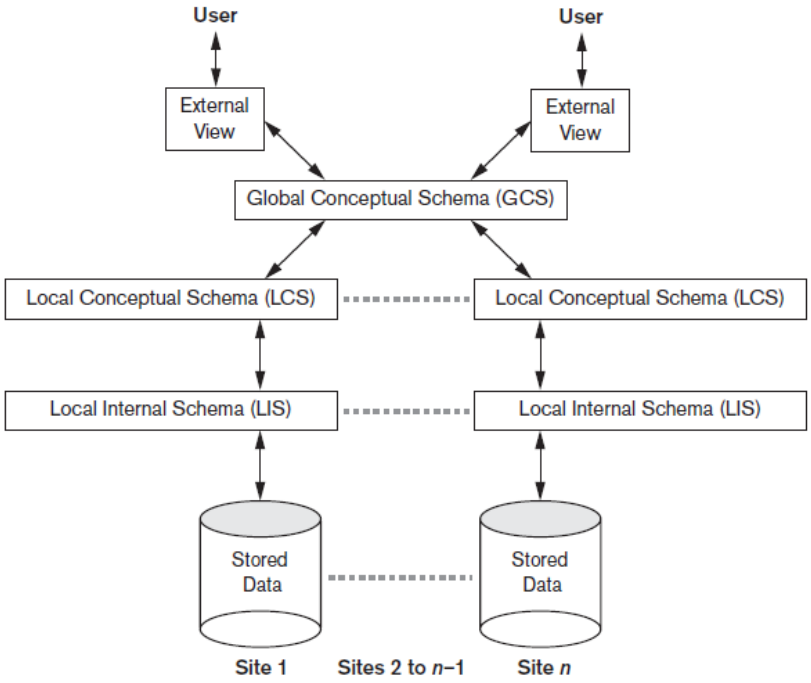


Ilustracija 1 - Podela sistema baza podataka na osnovu autonomije, stepena distribuiranja i heterogenosti

2.4 Arhitekture distribuiranih baza podataka

2.4.1 Arhitektura potpuno distribuiranih baza podataka

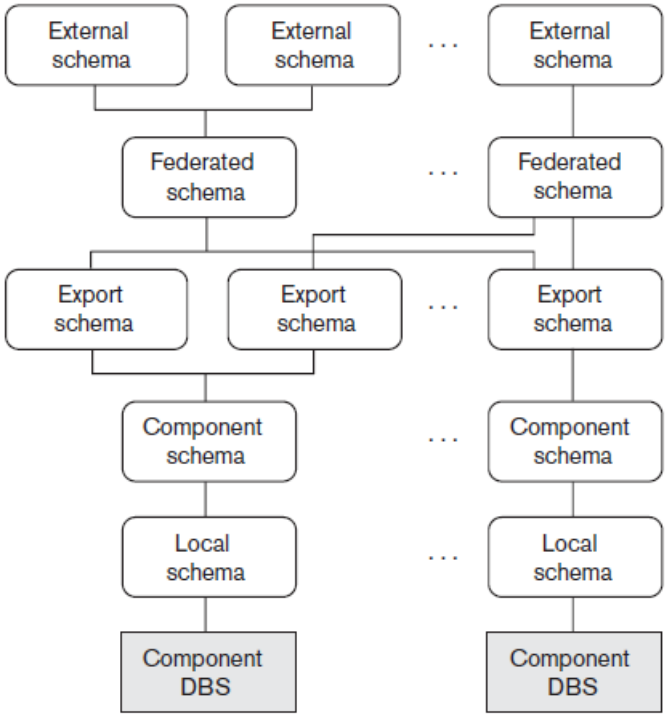
Na Ilustracija 2 se može videti generalna arhitektura potpuno distribuirane baze podataka (DDB). Sistem se prikazuje korisnicima kao konzistentan i objedinjen pogled (view) koji prikazuje logičku strukturu podataka distribuiranih po svim čvorovima sistema. Taj pogled je predstavljen konceptualnom globalnom šemom baze (global conceptual schema – GCS) koja obezbeđuje transparentnost u odnosu na računarsku mrežu. Kako bi se obezbedila potencijalna heterogenost u distribuiranoj bazi, svaki čvor je prikazan da poseduje svoju lokalnu internu šemu (LIS) određenu na osnovu organizacionih detalja konkretnog čvora. Logička organizacija podataka u svakom čvoru je definisana lokalnom konceptualnom šemom (local conceptual schema – LCS). GCS i LCS i njima odgovarajuća mapiranja obezbeđuju fragmentacionu i replikacionu transprentnost. Zbog jednostavnosti prikaza, neke od glavnih komponenti nedostaju na slici. Globalni kompajler upita referencira globalnu konceptualnu šemu iz globalnog sistemskog kataloga kako bi obezbedio ograničenja definisana nad globalnom šemom. Globalni optimizator upita referencira i globalne i lokalne konceptualne šeme kako bi generisao lokalno optimizovane upite koji se koriste za izvršenje globalnih upita. On procenjuje različite strategije izvršenja koristeći funkciju cene koja je zasnovana na vremenu odziva (CPU, I/O i mrežne latencije) i procenjuje veličinu posrednih rezultata. Nakon procene cene za svaku strategiju, on bira onu sa najnižom cenom izvršenja. Svaki lokalni DBMS poseduje svoje lokalni optimizator upita, menadžera transakcija, lokalni sistemski katalog koji čuva lokalnu šemu baze podataka. Globalni menadžer transakcija je odgovoran za koordinaciju izvršenja transakcija nad različitim čvorovima usklađujući svoj rad sa menadžerima lokalnih transakcija.



Ilustracija 2 - Arhitektura potpuno distribuirane baze podataka

2.4.2 Arhitektura federativnih baza podataka

Na Ilustracija 3 data je slika arhitekture federativnih baza podataka (FDBS). Kod ove arhitekture lokalna šema je konceptualna šema komponente baze podataka, a komponentna šema je izvedena prevođenjem lokalne šeme u kanonički model ili zajednički model podataka (common data model – CDM) za FDBS. Prevođenje lokalne šeme u komponentnu šemu postiže se mapiranjima koji transformišu komande nad komponentnom šemom u odgovarajuće komande nad lokalnom šemom. Eksportovana šema predstavlja podskup komponentne šeme koji je dostupan FDBS-u. Federativna šema je globalna šema ili pogled koja nastaje kao rezultat integracije svih eksportovanih šema. Eksterne šeme definišu šeme podataka za određenu grupu korisnika ili za određenu aplikaciju.



Ilustracija 3 - Arhitektura federativnih baza podataka

2.5 Fragmentacija podataka

Fragmentacija podataka predstavlja raspodelu podataka po čvorovima distribuirane baze podataka. Potrebno je doneti odluku na kojim čvorovima će se čuvati koji podaci.

2.5.1 Horizontalna fragmentacija

Horizontalna fragmentacija relacije predstavlja podskup torki te relacije. Torke koje pripadaju određenom horizontalnom fragmentu definišu se na osnovu uslova vrednosti jednog ili više atributa relacije. Horizontalna fragmentacija deli relaciju horizontalno grupišući redove u podskupove torki pri čemu svaki podskup ima određeno logičko značenje. Ovako dobijeni fragmenti se mogu dodeliti različitim čvorovima u distribuiranom sistemu. Izvedena horizontalna fragmentacija (derived horizontal fragmentation) predstavlja primenu particionisanja primarnog ključa na sekundarne relacije koje su

povezane sa primarnim ključem referenciranjem pomoću stranog ključa. Na ovaj način se povezani podaci primarnih i sekundarnih relacija fragmentišu na isti način.

2.5.2 Vertikalna fragmentacija

Vertikalna fragmentacija deli relaciju „vertikalno“ po kolonama. Svaki vertikalni fragment sadrži samo određene atribute relacije. Kako bi mogli da rekonstruišemo početnu relaciju na osnovu vertikalnih fragmenata relacije, potrebno je da svaki fragment sadrži primarni ključ ili ključ kandidat relacije. Kako bi rekonstruisali početnu relaciju na osnovu vertikalnih fragmenata, potrebno je primeniti OUTER UNION operaciju nad fragmentima. Pored toga, možemo upotrebiti i FULL OUTER JOIN operaciju kako bi dobili iste rezultate čak iako je pored vertikalne primenjena i horizontalna fragmentacija.

2.5.3 Kombinovana (hibridna) fragmentacija

Možemo da kombinujemo horizontalnu i vertikalnu fragmentaciju nad istom relacijom i na taj način dobijemo kombinovanu fragmentaciju. Početnu relaciju dobijamo primenom UNION i OUTER UNION operacija u odgovarajućem redosledu. Fragment relacije R može se specificirati pomoću SELECT-PROJECT kombinacije operacija $\pi_L(\sigma_C(R))$ (gde je L podskup skupa atributa/kolona relacije R, a C jedan od podskupova torki relacije R).

Fragmentaciona šema baze podataka je definicija skupa fragmenata koji uključuju sve atribute i torke u bazi podataka koji zadovoljavaju uslov da se kompletna baza podataka može rekonstruisati pomoću tih fragmenata primenom neke sekvence OUTER UNION i UNION operacija.

Alokaciona šema opisuje alokaciju fragmenata po čvorovima DDB i na taj način vrši mapiranje svakog fragmenta u čvor na kojem se on nalazi. Ukoliko se fragment nalazi na više čvorova, za taj fragment se kaže da je repliciran.

2.6 Replikacija i alokacija podataka kod distribuiranih baza podataka

Replikacija je metoda za povećanje dostupnosti i pouzdanosti sistema. U najekstremnijem slučaju cela baza podataka se nalazi na svakom čvoru distribuiranog sistema i na taj način kreira potpuno repliciranu distribuiranu bazu podataka. Na taj način se može značajno poboljšati pouzdanost sistema jer će sistem nastaviti da funkcioniše dok god postoji makar jedna replika koja funkcioniše. Takođe se poboljšavaju performanse pribavljanja podataka jer se upiti mogu proslediti nekoj lokanoj replici i na taj način očekivati brži odgovor. Mane potpune replikacije su drastično sporije operacije ažuriranja, jer se operacija mora izvršiti nad svakom replikom baze kako bi se one održavale u konzistentnom stanju. Ovo je posebno izraženo ukoliko postoji veliki broj replika. Potpuna replikacija čini kontrolu konkurentnosti i tehnike oporavka sistema skupljim nego što bi to bilo da nema replikacije.

Drugi ekstreman slučaj je nepostojanje replikacije, tj. svaki fragment se čuva na tačno jednom čvoru. U ovom slučaju svi fragmenti moraju biti disjunktni osim ponavljanja primarnog ključa u slučaju vertikalne ili kombinovane fragmentacije. Ovakav tip alokacije se naziva i ne redundantna alokacija.

Između ova dva ekstremna slučaja javlja se mogućnost primene delimične replikacije podataka, tj. neki fragmenti baze podataka mogu biti replicirani dok neki drugi ne moraju. Broj kopija svakog fragmenta može da varira od jedne kopije do broja čvorova u sistemu. Opis replikacije fragmenata se nekad naziva i šema replikacije.

Svaki fragment ili kopija fragmenta mora biti dodeljen nekom čvoru u distribuiranom sistemu. Ovaj proces se naziva distribucija podataka (ili alokacija podataka). Izbor čvorova i stepen replikacije za svaki fragment zavise od stepena performansi i dostupnosti koje želimo da postignemo u sistemu kao i frekventnosti transakcija na svakom čvoru. Ukoliko je potreban visok nivo dostupnosti, da se transakcije mogu proslediti svakom čvoru i ako većina transakcija služi samo za pribavljanje podataka, onda je potpuno replicirana baza podataka predstavlja dobar izbor. Sa druge strane, ukoliko transakcije koje se prosleđuju određenom čvoru pristupaju samo delovima baze podataka, onda se ti delovi baze mogu fragmentisati i alocirati na tom čvoru. Ukoliko se istim podacima pristupa sa više čvorova, isti podaci se mogu replicirati na svim čvorovima. Ukoliko se izvršava veliki broj ažuriranja, bolje performanse se mogu postići ograničavanjem broja replika. Pronalaženje optimalno ili dovoljno dobro rešenje alokacije podataka u distribuiranom sistemu je složen optimizacioni problem.

2.7 Obrada i optimizacija upita u distribuiranim bazama podataka

Upiti distribuiranih baza podataka se prosleđuju kroz sledeće faze:

1. Mapiranje upita – Upit se kod distribuiranih baza predstavlja formalno predstavlja upitnim jezikom. Takav upit se prevodi u algebarsku reprezentaciju upita nad globalnom konceptulanom šemom podataka i pri tom ne uzima u obzir distribuciju i replikaciju podataka. Ovo prevođenje je identično onom koje se dešava u centralizovanim DBMS-ovima.
2. Lokalizacija – Kod distribuiranih baza, podaci unutar relacija su fragmentisani i kao takvi se čuvaju na različitim čvorovima, pri čemu neki fragmenti mogu biti replcirani. U ovoj fazi se vrši mapiranje upita definisanog nad globalnom šemom u pojedinačne upite nad fragmentima koristeći informacije o distribuciji i replikaciji podataka.
3. Globalni optimizator upita – Optimizacija se sastoji od odabira strategije iz liste kandidata koja je najoptimalnija. Lista kandidata se dobija permutacijama operacija nad fragment upitima generisanih u prethodnoj fazi. U ukupnu procenu cene izvršenja uzimaju se u obzir CPU cena, I/O cena i cena komunikacije. Obzirom da su distribuirane baze povezane računarskom mrežom, ova cena ima veću težinu u odnosu na druge dve. Ovo je veoma izraženo ukoliko čvorovi komuniciraju preko WAN mreže.
4. Lokalna optimizacija upita – U ovoj fazi svaki čvor distribuirane baze optimizuje upit nad fragmentom koji se nalazi na tom čvoru. Optimizacija je slična onoj koja se odvija u centralizovanim sistemima.

2.7.1 Obrada distribuiranih upita primenom SEMIJOIN-a

Ideja pri distribuiranoj obradi upita primenom semijoin operacije je da se smanji broj torki relacije pre nego što se proslede drugom čvoru. Ideja je da se prosledi kolona relacije R na osnovu koje se vrši spoj (joining column) do čvora gde se nalazi druga relacija S i da se obavi spoj ove kolone i relacije S. Nakon toga atributi koji služe za spoj kao i atributi koji su potrebni u rezultatu se projektuju i prosleđuju čvoru koji poseduje relaciju R. Na ovaj način samo se kolona koja služi za spoj prosleđuje u jednom smeru i podskup podataka relacije S u drugom smeru. Ako samo mali broj torki relacije S učestvuje u spoju, ovakva obrada je veoma efikasna i minimizuje transfer podataka između čvorova.

2.7.2 Dekompozicija upita

Kod DDBMS-a koji nemaju transparentnost distribuiranja, korisnik mora sam obaviti parsiranje upita za svaki potreban fragment. Ukoliko su neki fragmenti replicirani, korisnik mora navesti sa kojih čvorova želi da pribavi te fragmente. Ukoliko DDBMS ne poseduje i replikacionu transparentnost, onda korisnik mora održavati i konzistentnost repliciranih podataka prilikom ažuriranja.

Sa druge strane, ukoliko DDBMS podržava potpunu distribuiranu, fragmentacionu i replikacionu transparentost, on omogućava korisniku da upit ili ažuriranje obavi kroz globalnu šemu na isti način kao i kod pristupa centralizovanim sistemima. Prilikom ažuriranja DDBMS je odgovoran za održavanje konzistentnosti među replikama podataka korišćenjem nekog algoritma za implementaciju konkurentnog pristupa. Prilikom prosleđivanja upita, modul za dekompoziciju upita razlaže upit na podupite koji se mogu izvršavati na pojedinačnim čvorovima. Pored ovoga, generiše se plan kombinovanja rezultata ovih pojedinačnih upita. Kako bi odredio koje replike sadrže određene podatke referencirane u upitu, DDBMS koristi informacije o fragmentaciji, replikaciji i distribuciji podataka iz DDBMS kataloga. Za vertikale fragmente u katalogu se čuva lista atributa koju svaki fragment poseduje, dok se za horizontalne fragmente čuva uslov za svaki fragment koji se još naziva i čuvar (guard). Prilikom dekompozicije upita, DDBMS određuje koji fragmenti mogu sadržati tražene torke poredeći uslov upita sa uslovom čuvara fragmenta.

2.8 Obrada transakcija u distribuiranim bazama podataka

Distribuirane baze poseduju globalnog menadžera transakcija kao dodatnu komponentu sistema kako bi obezbedio podršku za distribuirane transakcije. Globalni i lokalni menadžeri transakcija, zajedno sa menadžerima za upravljanje konkurentnim pristupom i oporavka DDBMS, zajedno obezbeđuju ACID svojstva DDBMS transakcija. Čvor na kome je transakcija započeta privremeno preuzima ulogu globalnog menadžera transakcija i koordiniše izvršenje operacije distribuirane baze podataka sa lokalnim menadžerima svakog čvora. Operacije svakog menadžera obezbeđene su kroz interfejs i uključuje sledeće operacije: BEGIN_TRANSACTION, READ ili WRITE, END_TRANSACTION, COMMIT_TRANSACTION i ROLLBACK. Globalni menadžer čuva informacije o svakoj transakciji poput jedinstvenog identifikatora transakcije, čvor na kome je transakcija započeta i tako dalje. Za READ operacije vraća lokalnu kopiju podataka ukoliko je ona validna i dostupna, a za WRITE operacije obezbeđuje da se ažuriranja obavljaju nad svim replikama podataka. Za ABORT operacije menadžer garantuje da se izmene transakcije neće sačuvati ni nad jednim čvorom, a za COMMIT operacije obezbeđuje da sve replike podataka budu ažurirane i sačuvane. Atomičnost COMMIT i ABORT operacije u distribuiranim transakcijama se uglavnom implementira primenom protokola dvofaznog commit-a. Menadžer transakcija prosleđuje kontroleru konkurentnosti neophodne informacije i kontroler pribavlja i oslobađa odgovarajuće lock-ove. Ukoliko je neka transakcija već zaključala traženi podatak, onda se trenutna transakcija odlaže sve dok se lock ne oslobodi. Kada se lock oslobodi, operacije se prosleđuju procesoru za izvršenje koji upravlja izvršenjem operacija u distribuiranom sistemu. Kada su operacije završene, lock-ovi se oslobađaju i menadžer transakcija se ažurira rezultatom operacija.

2.8.1 Protokol dvofaznog commit-a (Two-phase commit protocol)

Kako bi se obezbedila atomičnost kod distribuiranih baza podataka, potrebno je obezbediti mehanizam oporavka koji se obavlja kroz dve faze. Pored lokalnih menadžera oporavka koji se izvršavaju na svakom čvoru, neophodan je i globalni menadžer oporavka ili koordinator koji održava neophodne informacije za

oporavak globalnih transakcija. Koordinator uglavnom prati protokol dvofaznog commit-a koji se sastoji iz sledećih faza:

- Faza 1 – Svi čvorovi distribuiranog sistema signaliziraju koordinatoru da su operacije transakcije koje uključuju taj čvor završene. Koordinator šalje poruku svim čvorovima koji su učestvovali u transakciji da se pripreme za commit. Svaki od čvorova učesnika koji primi tu poruku će izvršiti upis svih log zapisa kao i neophodnih informacija za lokalni oporavak čvora na disk. Nakon toga čvor odgovara koordinatoru da je spreman za commit šaljući mu OK signal. Ukoliko dođe do otkaza prilikom upisa log operacija, čvor odgovara koordinatoru da ne može da obavi commit šaljući signal NOT OK. Ukoliko koordinator ne dobije odgovor od nekog čvora u predodređenom vremenskom intervalu, koordinator pretpostavlja NOT OK odgovor.
- Faza 2 – Svi čvorovi su odgovorili sa OK i koordinator odgovara sa OK, transakcija je uspešna i koordinator šalje commit signal svim čvorovima učesnicima u transakciji. Pošto su sve lokalne promene transakcija i neophodne informacije za oporavak već snimljene u log podacima, oporavak od otkaza je moguć. Svaki čvor učesnik upisuje u transakcioni log zapis commit uspešno završene transakcije i trajno menja podatke u lokalnoj bazi podataka. Ukoliko jedan ili više čvorova pošalju NOT OK signal koordinatoru, transakcija je neuspešna i koordinator šalje ROLLBACK signal svim čvorovima koji učestvuju u transakciji. U tom slučaju, svaki čvor rollback-uje promene obavljene tom transakcijom.

Sveukupni doprinos dvofaznog commit-a je da ili svi čvorovi koji učestvuju u transakciji zajedno commit-uju promene ili da nijedan to ne uradi. Ukoliko neki od učesnika ili koordinator otkaže, uvek je moguće izvršiti oporavak transakcije. Ukoliko dođe do greške prilikom faze 1 vrši se otkazivanje transakcije, dok se pri otkazu u fazi dva podrazumeva commit transakcije i vrši se oporavak.

Najveći nedostatak dvofaznog commit-a je to što je on blokirajući protokol. Otkaz koordinatora blokira sve čvorove učesnike i primorava ih da čekaju na njegov oporavak. Ovo dovodi do smanjenja performansi sistema pogotovo ako čvorovi drže lock-ove na neke od resursa. Drugi problematičan scenario je zajednički otkaz koordinatora i nekog čvora koji je commit-ovao promene. Čvor učesnik nema načina da obezbedi da su svi učesnici commit-ovali promene. Odluku o commit-u donosi koordinator u toku prve faze, a učesnici će izvršiti commit u toku druge faze nezavisno od toga da li su drugi čvorovi primili poruku. U slučaju da koordinator i čvor koji je commit-ovao promene otkažu u isto vreme, rezultati transakcije postaju neizvesni i nedeterministički. U ovom slučaju transakcija je već commit-ovana od strane jednog učesnika i ne može se otkazati nakon oporavka koordinatora. Takođe, transakcija ne može da se optimistički commit-uje jer je došlo do otkazivanja transakcije.

2.8.2 Protokol trofaznog commit-a (Three-phase commit protocol)

Problemi protokola dvofaznog commit-a rešavaju se protokolom trofaznog commit-a. Ovaj protokol deli drugu fazu dvofaznog commit-a u dve podfaze – fazu pripreme za commit i commit fazu. U fazi pripreme se rezultat glasanja prosleđuje svim učesnicima. Ako su svi učesnici izglasali OK, koordinator upućuje učesnicima da pređu u stanje pripreme za commit. Commit faza je identična onoj u dvofaznoj implementaciji protokola. U slučaju da koordinator otkaže u toku ove podfaze, drugi učesnik može da izvrši započetu transakciju do kraja tako što će proveriti sa čvorom koji je otkazao da li je primio poruku za pripremu commit-a. Ukoliko nije, otkazuje se započeta transakcija. Na ovaj način se stanje protokola može oporaviti nezavisno od toga koji učesnik je otkazao. Kada učesnik primi poruku za pripremu commit-a, on zna da su učesnici izglasali commit. Ako poruka za pripremu nije primljena, pretpostavlja

se otkazivanje transakcije. Takođe, ograničavanjem vremenskog perioda da transakcija commit-uje ili abort-uje svoje promene, obezbeđuje se da će svi lock-ovi biti oslobođeni u predviđenom vremenskom roku.

2.9 Konkurentnost pristupa i oporavak u distribuiranim bazama podataka

Problemi pri konkurentnom pristupu podacima u distribuiranim bazama podataka mogu biti sledeći:

- Upravljanje većim brojem kopija istog podatka
- Otkaz individualnih čvorova
- Otkaz komunikacionih kanala između čvorova
- Problemi usled distribuiranog commit-a
- Distribuirani deadlock

Kako bi se upravljalo replciranim podacima, tehnike koje se koriste za implementaciju konkurentnog pristupa nadograđuju tehnike koje se koriste u centralizovanim sistemima. Ideja je da se određena kopija podatka asocira sa odgovarajućom reprezentativnom kopijom podatka. Lock za ovaj podatak se pribavlja nad reprezentativnom kopijom tog podatka i sva zaključavanja i otključavanja se šalju čvoru koji sadrži tu kopiju. Različite metode su zasnovane na ovaj način pri čemu se ove metode razlikuju u načinu odabira reprezentativne kopije.

2.9.1 Tehnika primarnog čvora

Kod ove tehnike postoji jedan primarni čvor koji je postavljen kao čvor koordinator svih podataka u bazi. Svi lock-ovi se čuvaju na ovom čvoru pa se i svi zahtevi za pribavljanje i oslobađanje lock-a šalju njemu. Ova metoda je proširenje centralizovane metode. Ukoliko transakcije poštuju protokol dvofaznog commit-a onda se garantuje i serijabilnost. Prednost ove metode je jednostavno proširenje centralizovanog pristupa koje nije previše složeno. Mana ovog pristupa je to što se svi zahtevi šalju jednom čvoru i na taj način može doći do opterećenja čvora i on postaje usko grlo sistema. Pored toga, ukoliko dođe do otkaza koordinatora doći će do paralize celog sistema, jer se sve informacije čuvaju na jednom mestu. Ovo može ograničiti pouzdanost i dostupnost sistema. Iako se lock-ovi pribavljaju na jednom čvoru, samim podacima se može pristupiti sa bilo kojeg čvora. Ukoliko transakcija pribavi lock za upis i ažurira podatak, DDBMS je odgovoran za ažuriranje svih ostalih kopija podatka pre nego što dođe do oslobađanja lock-a.

2.9.2 Primarni čvor sa rezervnim čvorom (Pirmary site with backup site)

Ovaj pristup rešava mane pristupa sa primarnom kopijom tako što uvodi drugi čvor koji služi kao rezerva (backup). Sve informacije o lock-ovima se čuvaju na primarnom i rezervnom čvoru. U slučaju otkaza primarnog čvora, rezervni čvor preuzima funkciju primarnog čvora i vrši se odabir novog rezervnog čvora. Ovakav pristup pojednostavljuje proces oporavka od otkaza primarnog čvora jer sistem može nastaviti sa radom odmah čim se odabere novi rezervni čvor i sve informacije o lock-ovima budu iskopirane na njega. Mana ovog pristupa je što se usporava proces pribavljanja lock-ova jer se svi zahtevi za pribavljanje lock-a moraju sačuvati i na primarnom i na rezervnom čvoru pre slanja potvrde. Problem preopterećenja primarnog i rezervnog čvora zahtevima i usporenje sistema ostaje ne rešen kod ovog pristupa.

2.9.3 Tehnika primarne kopije

Ova tehnika teži da distribuira opterećenje pribavljanja lock-ova tako što se reprezentativne kopije distribuiraju različitim čvorovima u sistemu. Otkaz jednog čvora utiče samo na transakcije koje zahtevaju podatke čije se reprezentativne kopije nalaze na otkazanom čvoru, dok druge transakcije nisu ugrožene. Ova metoda takođe može koristiti rezervne čvorove kako bi se povećala pouzdanost i dostupnost sistema.

2.9.4 Odabir novog koordinatora u slučaju njegovog otkaza

Ukoliko dođe do otkaza koordinatora u DDBMS-u, preostali čvorovi koji još funkcionišu moraju odabrati novog koordinatora. Kod tehnike primarnog čvora bez rezervnog čvora, sve transakcije koje se izvršavaju se otkazuju i ponovo pokreću (što može biti previše sporo). U toku oporavka bira se novi primarni čvor, na njemu se kreira proces lock menadžera i zapisi svih lock informacija. Kod tehnika sa rezervnim čvorom, izvršenje transakcija se suspenduje dok se rezervni čvor ne preuzme ulogu primarnog i dok se ne odabere novi rezervni čvor i sve informacije budu iskopirane sa primarnog na rezervni čvor. Ukoliko nije postojao rezervni čvor ili ukoliko dođe do otkaza i primarnog i rezervnog čvora, održavaju se „izbori“ za novog koordinatora. Ukoliko čvor Y ne uspe da dobije odgovor od koordinatora u određenom periodu vremena, on pretpostavlja da je došlo do njegovog pada i započinje proces izbora prosleđujući svim čvorovima predlog da Y postaje novi koordinator. Kada prikupi većinu potvrdnih glasova, Y šalje svim čvorovima poruku da je Y izabran za novog koordinatora. Sam algoritam je prilično složen i obezbeđuje da ukoliko dva ili više čvora započnu izbore samo jedan bude izabran za koordinatora.

2.9.5 Upravljanje konkurentnim pristupom zasnovano na glasanju

Kod ovog pristupa ne postoji reprezentativna kopija kao u prethodno objašnjenim tehnikama implementacije konkurentnosti. Kod tehnike zasnovane na glasanju zahtev za lock-om nad podatkom se prosleđuje svim čvorovima koji poseduju kopiju tog podatka. Svaka kopija podatka održava svoj lock. Ukoliko transakcija pribavi lock od strane većine kopija podatka, ona drži lock nad svim kopijama tog podatka. Ukoliko transakcija ne uspe da pribavi većinu lock-ova u predviđenom vremenskom roku, ona otkazuje svoj zahtev i obaveštava čvorove o otkazu zahteva.

Metoda glasanja se smatra potpuno distribuiranom metodom za upravljanje konkurentnog pristupa, jer su u odluku o pribavljanju lock-a uključeni svi čvorovi. Mana ove tehnike je veliki saobraćaj između čvorova zbog stalnog glasanja. Ukoliko se uzme u obzir i mogućnost otkazivanja čvora, algoritam postaje jako složen.

2.9.6 Oporavak distribuiranih sistema

Proces oporavka distribuiranih baza podataka je jako složen. Jako je teško odrediti da li je čvor otkazao bez razmene velikog broja poruka sa ostalim čvorovima. Ukoliko čvor X pošalje poruku čvoru Y i očekuje odgovor od Y, ali ne dobije ga postoji nekoliko mogućih situacija:

- Poruka nije stigla do čvora Y zbog problema u komunikaciji
- Čvor Y je otkazao i ne može da odgovori na poruku
- Čvor Y ja aktivan i poslao je odgovor, ali je došlo do greške prilikom dostavljanja odgovora

Bez dodatnih informacija ili slanja dodatnih poruka jako je teško utvrditi šta se tačno desilo. Drugi problem je distribuirani commit. Kada transakcija ažurira podatak na nekoliko čvorova, ona ne sme da

commit-uje promene sve dok ne može da garantuje da su sve promene sačuvane na svakom čvoru. Ovaj problem se rešava tehnikom dvofaznog commit-a.

2.10 Katalog distribuiranih baza podataka

Odabir efikasne implementacije kataloga kod distribuiranih baza podataka može imati značajan uticaj na performanse sistema, distribuciju i replikaciju podataka. Katalozi su i sami baze podataka koje sadrže meta podatke o distribuiranoj bazi podataka. Postoje tri popularna načina implementacije distribuiranih kataloga: centralizovani katalozi, potpuno replicirani katalozi i delimično replicirani katalozi.

2.10.1 Centralizovani katalozi

Kod ove implementacije čitav katalog se čuva na jednom čvoru. Prednost ove metode je jednostavnost implementacije. Sa druge strane smanjuje se pouzdanost, dostupnost i sposobnost distribucije procesiranja u sistemu. Pri operacijama čitanja iz kataloga sa ne centralnog čvora, zahtev se šalje centralnom čvoru gde se ti podaci zaključavaju i zatim prosleđuju čvoru koji ih je tražio. Nakon završetka obrade, potvrda se šalje centralnom čvoru da otključa podatke. Sve operacije ažuriranja kataloga moraju se obraditi kroz centralni čvor. Ovo brzo dovodi do preopterećenja tog čvora ukoliko postoji veliki broj upisa u sistemu.

2.10.2 Potpuno replicirani katalozi

Kod ove implementacije potpuna kopija celokupnog kataloga se čuva na svakom čvoru. Ova šema omogućava brza čitanja iz kataloga omogućujući da zahtevima odgovore lokalni čvorovi sistema. Sa druge strane, ažuriranja se moraju emitovati (broadcast) svim čvorovima. Ažuriranja se tretiraju kao posebne transakcije, a konzistentnost se obezbeđuje primenom protokola dvofaznog commit-a. Kao i kod centralizovane implementacije, veliki broj upisa dovodi do povećanog broja poruka u sistemu usled emitovanja svake poruke.

2.10.3 Delimično replicirani katalozi

Centralizovani i potpuno replcirani pristup smanjuju autonomiju čvorova jer se mora obezbediti konzistentnost kataloga na globalnom nivou. Kod delimično replicirane implementacije, svaki čvor održava sve informacije iz kataloga o podacima koji se čuvaju na tom čvoru. Svaki čvor takođe kešira podatke o katalogu pribavljene od drugih čvorova, pri čemu se ne garantuje da su keširani podaci ažurni. Sistem prati zapise kataloga za svaki čvor sa koga je podatak kreiran i sa čvorova koji sadrže zapise kopije podatka. Sve promene se odmah propagiraju do čvora gde je kreiran podatak. Pribavljanje ažuriranih kopija podataka može biti odloženo dok se ne ostvari pristup čvorovima gde su oni kreirani.

3 Couchbase distribuirana baza podataka

Couchbase je open source distribuirana baza podataka, nastala spajanjem dva rešenja baze podataka – CouchDB i Membase. Podaci se čuvaju kao key-value parovi i spada u grupu document baza podataka. Obavlja brze operacije nad podacima zahvaljujući snažnim servisima za upite i indeksiranje. Za predstavljanje upita ka bazi koristi se bogat upitni jezik nazvan N1QL. Više instanci Couchbase servera se mogu zajedno formirati jedan klaster. Cluster Manager je program koji koordiniše aktivnosti svih čvorova klastera i obezbeđuje jednostavan interfejs prema svim korisnicima klastera. Individualni čvorovi se mogu dodavati i izbacivati iz klastera bez uticaja na performanse sistema.

Podaci se kod Couchbase servera mogu čuvati na dva načina - samo u glavnoj memoriji (RAM) ili kombinovano u glavnoj memoriji i na disku. Podaci se mogu replicirati među čvorovima klastera kako bi se obezbedila pouzdanost sistema. Takođe, podaci se mogu replicirati između više data centara kao backup (rezervna) verzija ili za implementaciju geografski distribuiranih aplikacija.

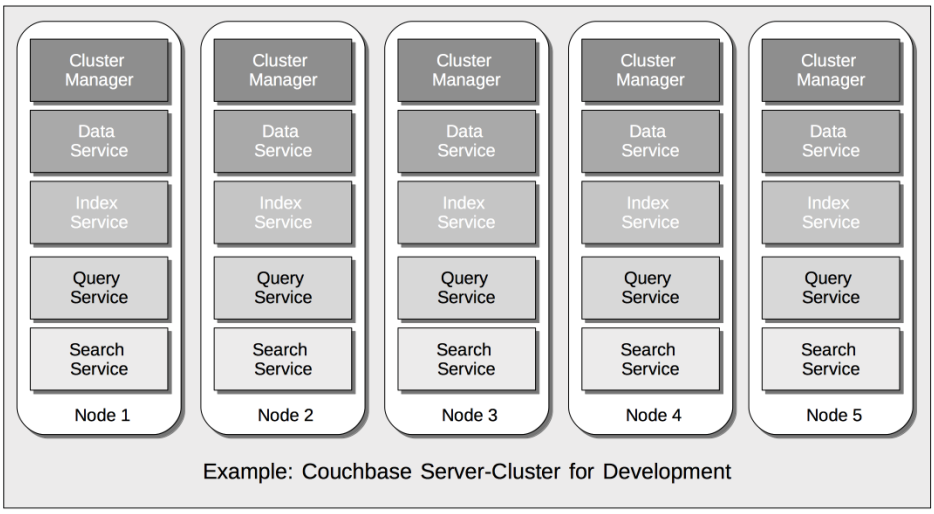
3.1 Couchbase klaster

Couchbase klaster se sastoji od jedne ili više instanci Couchbase servera pri čemu se svaki server pokreće na nezavisnom čvoru. Podaci i servisi se dele unutar klastera.

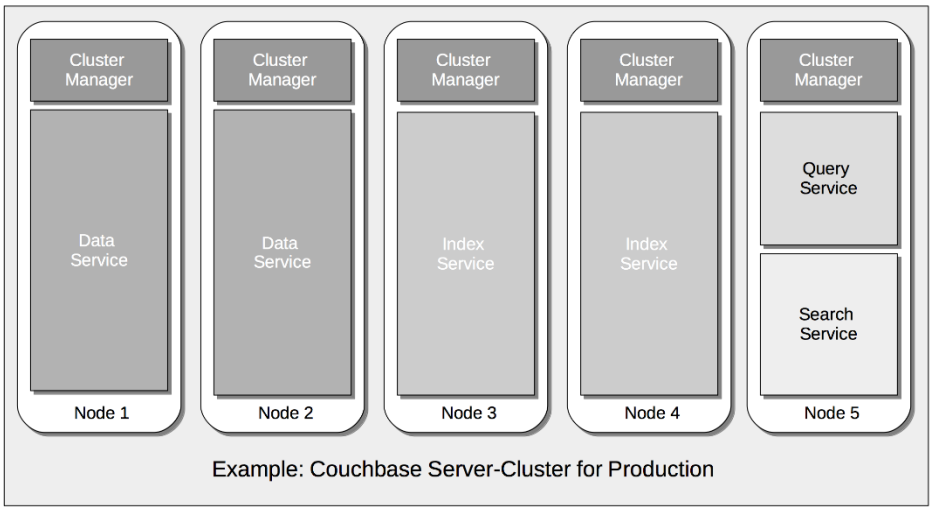
Prilikom pokretanja Couchbase servera, on se može konfigurisati kao samostalni server, kao novi klaster ili se može priključiti već postojećem klasteru. Kada klaster već postoji, čvorovi se mogu dodavati u njega pri čemu je na svakom pokrenuta jedna instanca Couchbase servera. Kada klaster sadrži više čvorova, na svakom čvoru je pokrenut Couchbase Cluster Manager koji upravlja komunikacijom između čvorova i obezbeđuje da su svi čvorovi stabilni. Cluster Manager obezbeđuje korisniku informacije o klasteru kroz Web interface.

3.2 Servisi Couchbase servera

Couchbase server uključuje više servisa (Services). Svaki servis se može deploy-ovati i održavati nezavisno od ostalih i na taj način obezbeđuje veliku sposobnost skaliranja sistema. Svaki čvor može pokrenuti po **jednu instancu** svakog servisa (Ilustracija 4) ili se pojedinačni čvorovi mogu posvetiti izvršenju samo jednog servisa (Ilustracija 5). Neki servisi poseduju zavisnosti u odnosu na druge i zahtevaju postojanje barem jedne instance servisa od kojih zavise. Na primer za pokretanje Query servisa potrebno je postojanje Index i Data servisa unutar klastera.



Ilustracija 4 - Klaster od 5 čvorova gde svaki čvor pokreće svaki servis.



Ilustracija 5 - Klaster od 5 čvorova gde postoje 2 specijalizovanih Data Service čvora, 2 Index čvora i jedan čvor na kome su pokrenuti Query i Search Service.

Couchbase podržava sledeće servise:

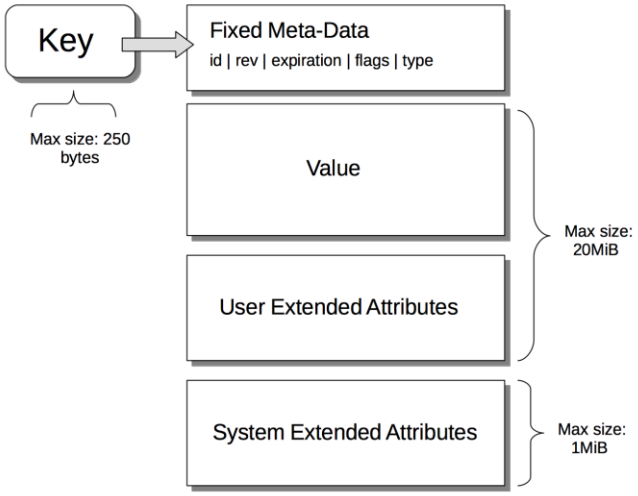
- Data: Podržava čuvanje i pribavljanje podataka po ključu
- Query: Obavlja parsiranje upita napisanog N1QL jezikom, izvršava upit i vraća rezultat. Query servis u toku izvršenja koristi usluge Data i Index serivsa.
- Index: Kreira indekse nad podacima koji se mogu koristiti od strane Query servisa.
- Search: Kreira indekse specifično za pretraživanje teksta dokumenata.
- Analytics: Obavlja operacije spoja (join operation), operacije nad skupovima (set operations), operacije agregacije i grupisanja podataka. Operacije koje obavlja se uglavnom smatraju zahtevnim i dugotrajnim operacijama koje zahtevaju veliku količinu memorije i CPU reursa.
- Eventing: Podržava skoro real-time obradu promene u podacima. Može se podesiti kod koji treba izvršiti nakon promene podataka ili na određen period vremena.

3.3 Struktura podataka u Couchbase bazi podataka

Podaci u Couchbase bazi se čuvaju u obliku key-value parova. Svaka vrednost podatka se identifikuje na osnovu jedinstvenog ključa (key) koji je kreiran od strane korisnika ili aplikacije prilikom čuvanja podataka. Svaki ključ mora biti UTF-8 string bez razmaka, ne sme biti duži od 250 byte-ova i mora biti jedinstven u okviru svog bucket-a.

Maksimalna veličina podatka je 20 MB. Podaci se mogu čuvati u dva formata:

- Binary: Bilo koja forma je prihvatljiva. Ovi podaci se ne mogu parsirati, indeksirati ili pretraživati već se mogu samo pribavljati po ključu.
- JSON: Za ove podatke se još kaže da imaju strukturu dokumenta (document). Oni se mogu parsirati indeksirati i pretraživati. Svaki dokument se sastoji od jednog ili više atributa pri čemu svaki ima svoju vrednost. Atributi mogu biti prostog tipa poput broja, string-a ili boolean vrednosti ili složenog tipa poput nizova ili ugnježdenih dokumenata. Couchbase podržava do 30 nivoa pod dokumenata.



Ilustracija 6 - Struktura podataka u Couchbase bazi podataka

3.4 Bucket-i

Couchbase server čuva podatke u Bucket-ima koji povezuju logički povezane grupe key-value parova. Svaki klaster može sadržati najviše 30 bucket-a. Postoje tri tipa backet-a:

- Couchbase bucket-i: Ova vrsta bucket-a čuva podatke perzistirane na disku kao i u glavnoj memoriji. Njima se omogućava visoka dostupnost podataka korišćenjem Databse Change Protocol-a (DCP) i dinamičko skaliranje više klastera korišćenjem Corss Datacenter Replication (XDCR). Ukoliko bucket prekorači predviđenu vrednost RAM memorije, dolazi do izbacivanja dokumenta (stavka podataka kod document baza podataka) koji su sačuvani se na disku i u memoriji. Ukoliko se traženi dokumenti (podaci) ne nalaze u glavnoj memoriji, oni se učitavaju u memoriju sa diska. Postoje dva načina izbacivanja dokumenata koji se podešavaju na samom kreiranju bucket-a:

    - Value-only: Kod Value-only izbacivanja dolazi do uklanjanja samo key-value parova dokumenata i time pospešuju performanse sistema po cenu memorije.
    - Full: Kod Full moda se izbacuju celokupni podaci (gde spadaju ključevi, key-value parovi i metadata podaci) i time daje prednost memoriji po ceni performansi.
- Ephemeral bucket-i: Predstavljaju alternativu Couchbase bucket-ima kada nije potrebna perzistencija podataka. Omogućavaju visoke performanse nad podacima koji se nalaze u glavnoj memoriji bez usporavanja zbog pristupa disku. Ovim se takođe pospešuje brzina rebalansa podataka između čvorova i brzina restart-ovanja čvorova. Ukoliko dođe do prekoračenja RAM memorije dodeljene ovoj vrsti bucket-a, može se deseti jedna od sledećih stvari:
    - Podaci sačuvani u RAM memoriji ostaju sačuvani i ne mogu se dodavati novi podaci. Pokušaj da se doda novi podatak prouzrokuje grešku.
    - Sačuvani podaci se izbacuju iz RAM memorije kako bi se napravio prostor za nove podatke. U tom slučaju se izbačeni podaci ne mogu pribaviti ukoliko su zatraženi od servera. Izbacivanjem se brišu celokupni podaci dokumenta i na njegovom mestu se određeno vreme čuva tombstone zapis.
- Memcached bucket-i: Ove vrste bucket-a su dizajnirani da se koriste u kombinaciji sa drugim sistemima baza podataka na primer relacionim bazama podataka. Keširanjem podataka u ovim bucket-ima smanjuje se broj upita koji se šalju serveru baze podataka. Svaki memcached bucket obezbeđuje direktno adresibilan distribuiran key-value keš podataka koji se čuvaju u glavnoj memoriji. Podaci se i kod ovih bucket-a ne čuvaju na disku. Ukoliko dođe do prekoračenja RAM memorije rezervisane za ove bucket-e, takođe dolazi do izbacivanja podataka.

Kod svih bucket-a se izbacivanje podataka iz glavne memorije obavlja po NRU (Not Recently Used – ne skoro korišćeni podaci) algoritmu i svi bucket-i su kompatibilni sa Memcached open sourced implementacijom key-value keša.

3.5 Couchbase i CAP teorema

Po CAP teoremi, sistemi za upravljanjem distribuiranim sistemima se uglavnom smatraju CP ili AP, odnosno uglavnom se pravi kompromis između konzistencije sistema C ili dostupnosti sistema A. U odnosu na CAP teoremu se Couchbase server ponaša kao CP (consistency, partitian tolerant) sistem po svojoj default konfiguraciji ili kada se pokrene kao jedan klaster. To je zato što se prilikom pribavljanja podataka uvek pristupa čvoru koji sadrži aktivnu kopiju zahtevanih podataka. Zahtevi za podatke se uz pomoć klijentske biblioteke transparentno distribuiraju do odgovarajućih čvorova i na taj način postaju vidljivi zahtevima ostalih klijenata. Takođe, svaki upis unutar klastera se distribuira svim replikama podataka. Ove replike nemaju u svakom trenutku najažurnije podatke, uglavnom se ne koriste za pribavljanje podataka sve dok ne postanu aktivne kopije podataka. Uglavnom se koriste za postizanje visoke pouzdanosti.

Prema CAP teoremi particionisanja mreže se ne mogu razlikovati u odnosu na otkaz dela sistema. U slučaju otkaza jednog čvora, upis podataka biće privremeno onemogućen dok se ne odredi sledeća primarna kopija nedostupnih podataka. Sa druge strane zahtevi za čitanje podataka se mogu opslužiti od strane neke od replika podataka koje se nalaze u klasteru.

3.6 Trajnost (Durability) upisa podataka

Klijenti koji upisuju podatke na Couchbase server navode uslove trajnosti podataka pri upisu na osnovu čega server može da odredi da li treba ažurirati podatke na više čvorova prilikom upisa u memoriju i/ili na disku. Sa povećanjem broja čvorova prilikom upisa povećava se i neophodan nivo trajnosti. Kada se promene commit-uju na serveru, klijent dobija potvrdu da su promene sačuvane. U suprotnom se klijentu vraća greška, a podaci zadržavaju vrednosti pre početka upisa.

Couchbase podržava trajnost upisa samo na nivou jednog dokumenta. Ova vrsta upisa se naziva i trajni upis ili sinhroni upis. Trajni upis je moguće obaviti nad najviše dve replike podataka.

3.6.1 Vrste upisa

Trajnost omogućava korisnicima da biraju između dve vrste upisa:

- Regularni upis koji je asinhron i ne podržava trajnost. Pogodan je prilikom upisa podataka čiji gubici ne bi prouzrokovali velike probleme u aplikaciji. Na primer, jedna izgubljena vrednost senzora u skupu od nekoliko hiljada neće imati veliki značaj.
- Trajni upisi su sinhroni i podržavaju trajnost. Pogodan za upis podataka čiji gubitak bi doveo do negativnih posledica. Na primer, gubitak finansijskih transakcija.

3.6.2 Parametri trajnog upisa

Parametri koje klijent specificira pri trajnom upisu su:

- Level (nivo) – Predstavlja nivo trajnog upisa. Moguće vrednosti su:
    - majoirty (većina) – Izmena se mora proslediti u memoriju većine Data Service čvorova. Većina predstavlja konfigurisanu vrednost Data Service čvorova do kojih se promena mora proslediti. Određuje se na osnovu broja replika bucket-a.

Broj replika	Broj čvorova neophodnih za većinu
0	1
1	2
2	2
3	Nije podržano

    - majorityAndPersistActive – Izmena se mora proslediti većini Data Service čvorova. Pored toga, čvor koji sadrži aktivnu kopiju podatka (primarna kopija) mora sačuvati izmenu na disk.
    - persistToMajority – Izmena se mora proslediti većini Data Service čvorova i svaki čvor mora snimiti tu promenu na disk.
- Timeout – Vreme u milisekundama za koje se trajni upis mora obaviti. Default vrednost je 2500 ms.

3.7 Transakcije

Distribuirane ACID transakcije su operacije koje obezbeđuju da se prilikom istovremene promene većeg broja dokumenata ili sve promene uspešno izvrše do kraja, ili da se nijedna od promena ne izvrši.

Atomičnost je podržana kroz operacije dodavanja, ažuriranja i brisanja kroz bilo koji broj dokumenata. Trajnost je obezbeđena sinhronim upisom.

Couchbase podržava samo jedan tip izolacije transakcija i to je Read Commited nivo po SQL standardu. Prilikom izvršenja, transakcija čuva izmene obavljane nad dokumentom kao Extended atribut dokumenta u njegovim meta podacima. Na taj način se postiže da promene transakcije koje još nisu commit-ovane ne budu pročitane od strane neke druge transakcije. Nakon commit-a se ove promene čitaju iz meta podataka dokumenta i prepisuju sam sadržaj dokumenta i postaju vidljive drugim transakcijama. Kao posledica ove činjenice, dokumenti koji se mogu koristiti u transakcijama moraju biti manji od 10 MB.

Samo čvorovi koji sadrže podatke neophodne u transakciji učestvuju u transakciji. Više transakcija mogu da čitaju isti dokument u isto vreme. Ukoliko dve transakcije pokušaju da menjaju isti dokument, samo jedna može to uraditi u datom trenutku, dok druga dobija grešku i mora da pokuša ponovo.

3.7.1 Servisi i transakcije

Indeksi koji se koriste u Index, Search i Analytic servisu nisu konzistentni u trenutku commit-ovanja transakcije već postaju eventualno konzistentni sa ovim izmenama (Eventual Consistency). Ni Query ni Search servis ne vide ne commit-ovane promene. Međutim izolacija zasnovana na snapshot-u nije obezbeđena, pa se može desiti da pri simultanoj promeni i pretrazi podataka može doći do vraćanja dela podataka pre i dela podataka posle izvršenog commit-a.

3.7.2 Ograničenja

Couchbase transakcije imaju sledeća ograničenja:

- Samo dokumenti čija je veličina manja od 10 MB se mogu koristiti u transakcijama.
- Ne transakciona ažuriranja dokumenata bi trebalo izbeći nad dokumentima koji se već menjaju nekom transakcijom kako ne bi došlo do prepisivanja tih izmena od strane transakcije.
- Broj upisa koji se obavljaju prilikom ažuriranja dokumenta za vreme transakcije je veći od broja upisa ukoliko transakcije nema. Razlog tome su upisi neophodni za pripremu podataka za commit, sam commit kao i čuvanje izmena koje transakcija napravi u toku izvršenja.
- Replikacija među Data centrima (XDCR – Cross Data Center Replication) podržava eventualnu konzistentnost podataka i ne podržava atomičnost. Iz tog razloga, dokumente izmenjene transakcijom treba replicirati drugim Data centrima jedino ako se u njima ne može obaviti transakcija nad istim dokumentima.

3.8 Dostupnost (Availablity)

Podaci se automatski distribuiraju među čvorovima klastera. Svaki bucket se čuva na Data Service čvorovima kao 1024 vBucket (virutal bucket), koji se prenose do svih dostupnih Data Service čvorova. Dokumenti se čuvaju neizmenjeni unutar svakog vBucket-a. vBucket-i se takođe mogu replicirati unutar klastera upotrebom Database Change protokola između svih postojećih replika koje se razlikuju od originala.

Couchbase automatski obrađuje dodavanje i brisanje čvorova, kao i otkaz čvorova kako na taj način ne bi došlo do gubitka podataka. vBucket-i i njihove replike se redistribuiraju dostupnim čvorovima kad god dođe do promene u konfiguraciji.

Visok nivo dostupnosti se postiže pomoću replikacije podataka među Data centrima (Cross Datacenter Replication XDCR). Sadržaj bucket se može replicirati i održavati na udaljenom (remote) klasteru.

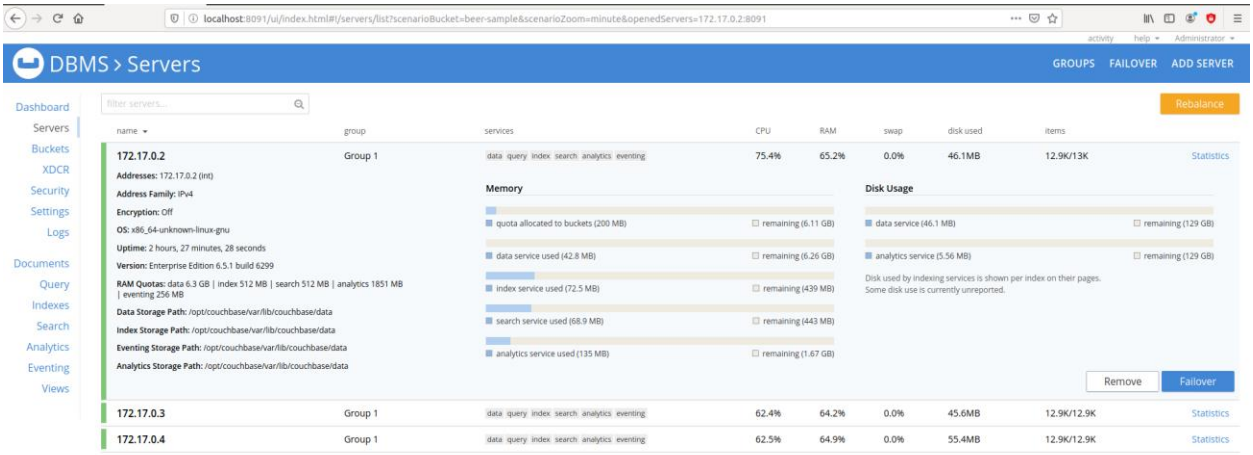
3.8.1 Lokalna (Local) i udaljena (Remote) replikacija

Couchbase podržava dve vrste replikacije:

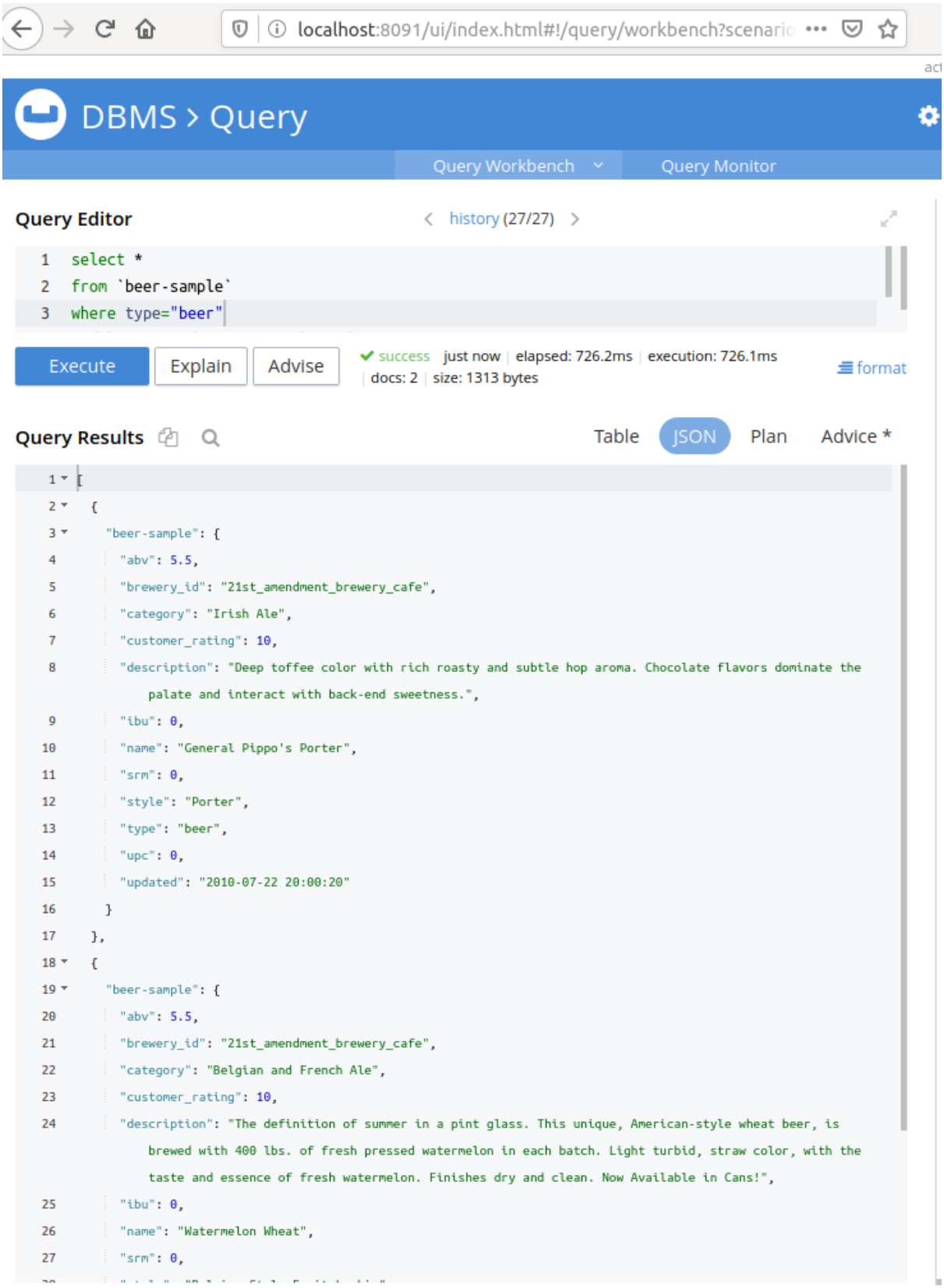
- Lokalna (Local) ili replikacija unutar klastera uključuje repliciranje podataka između čvorova klastera. Prilikom definicije bucket-a definiše se i broj njegovih replika. Maksimalan broj replika u jednom klasteru je 3, pri čemu realan broj replika može biti manju ukoliko postoji premalo čvorova u klasteru. Ažuriranjem aktivnih kopija podataka (primarna kopija podatka) dolazi i do ažuriranja replika podataka. U slučaju otkaza čvora, jedna od replika preuzima ulogu primarne replike. Lokalna replikacija podržava Data Change Protocol.
- Udaljena (Remote) ili replikacija između Data centara (Cross Data Center Replication – XDCR) podrazumeva razmenu podataka između različitih klastera, pri čemu se svaki nalazi u drugom Data centru. XDCR se konfiguriše nakon kreiranja bucket-a u lokalnom klasteru, koji će služiti kao izvorni klaster, i bucket-a na udaljenom kalsteru, koji će služiti kao odredišni klaster. Promene nad podacima u izvornom klasteru se automatski prosleđuju odredišnom klasteru. Pristup se može obaviti nad oba klastera. XDCR se može konfigurisati tako da replikacija funkcioniše u oba smera, odnosno promene u izvornom ili odredišnom klasteru ažuriraju podatke u drugom klasteru.

3.9 Primer izvršenja distribuirane transakcije na Couchbase klasteru

Couchbase server poseduje integrisan Web interfejs koji je dostupan na portu 8091. Ovaj interfejs omogućava pregled statusa svih čvorova u klasteru, pregled statusa svih servisa u klasteru, izvršavanje N1QL upita, podešavanje konfiguracije klastera i dodavanje i brisanje čvorova.



Ilustracija 7 - Informacije od čvorovima koji se nalaze u klasteru



Ilustracija 8 - Primer izvršenja N1QL upita na Web klijentu i njegov rezultat

Za demonstraciju izvršenja distribuiranih transakcija biće iskorišćena demo Java aplikacija i Web interfejs. Aplikacija će se izvršavati nad beer-sample podacima koji se dobijaju uz Couchbase server. Glavna logika aplikacije je u longRunningTransaction funkciji (Ilustracija 10) koja ažurira customer_rating atribut dokumenata koji se dobijaju na osnovu upita:

```
select *
from `beer-sample`
where type="beer"
and brewery_id="21st_amendment_brewery_cafe"
and abv=5.5
```

Izvršni kod klijenta dat je na Ilustracija 9 i Ilustracija 10.

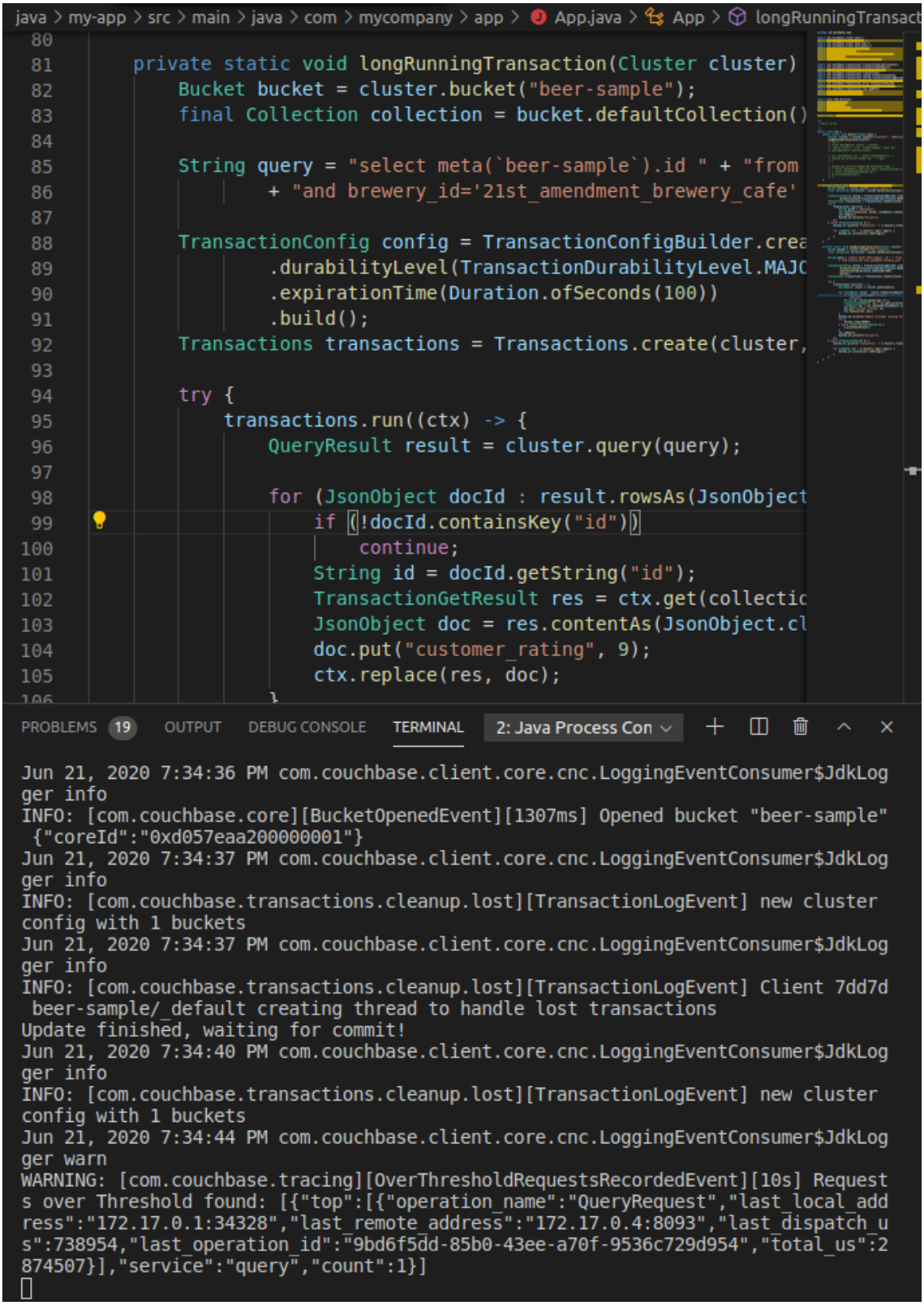
```
39  public class App {
        Run | Debug
40      public static void main(String[] args) {
41          if(args.length != 1) {
42              System.out.println("Expecting one parameter as argument");
43              return;
44          }
45          int value;
46          try {
47              value = Integer.parseInt(args[0]);
48          } catch (NumberFormatException ex) {
49              System.out.println(ex.getStackTrace());
50              return;
51          }
52          Cluster cluster = Cluster.connect("localhost", "Administrator", "Administrator");
53          longRunningTransaction(cluster, value);
54      }
55
```

Ilustracija 9 - Main funkcija demo aplikacije

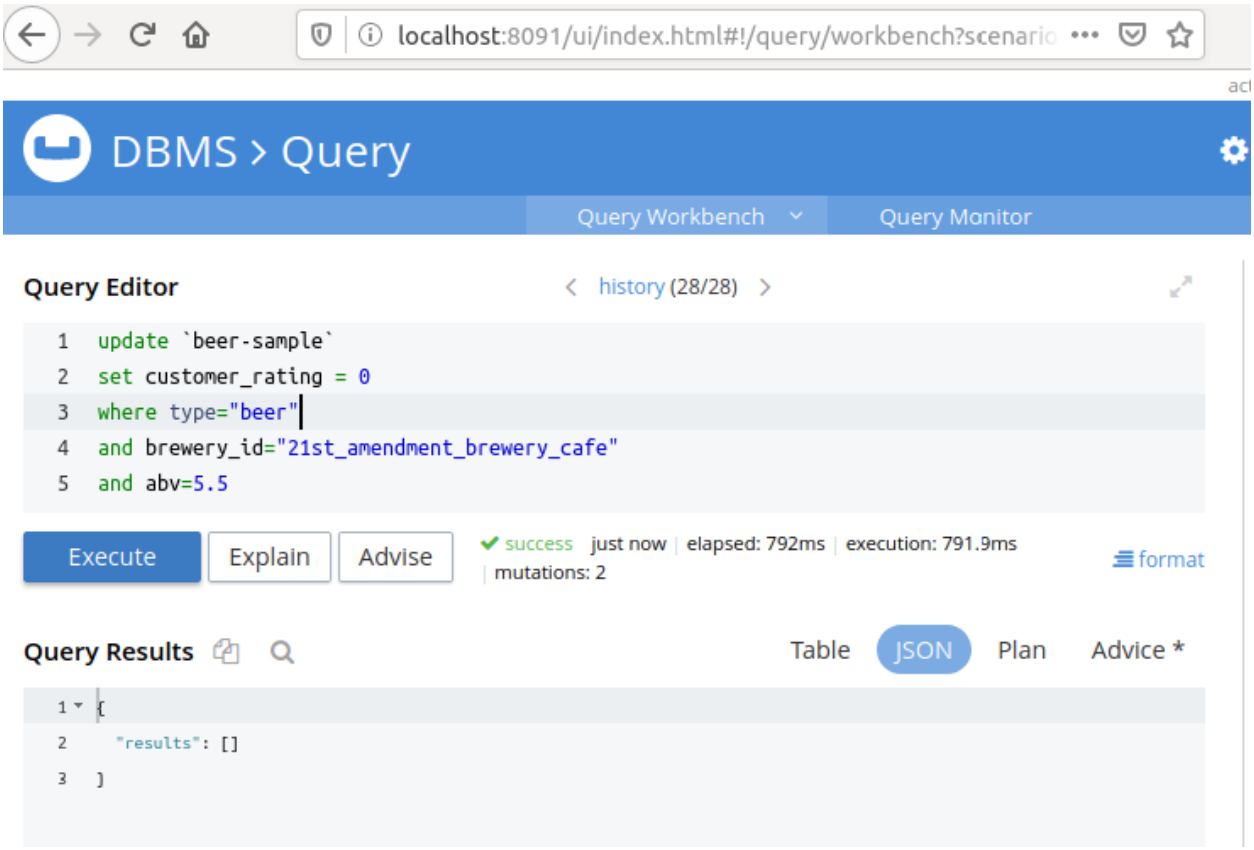
```java
private static void longRunningTransaction(Cluster cluster, int value) {
    System.out.println("Starting long running transaction.");
    Bucket bucket = cluster.bucket("beer-sample");
    final Collection collection = bucket.defaultCollection();

    String query = "select meta(`beer-sample`).id " + "from `beer-sample`" + "where type='beer'"
            + "and brewery_id='21st_amendment_brewery_cafe' " + "and abv=5.5";

    TransactionConfig config = TransactionConfigBuilder.create()
            .durabilityLevel(TransactionDurabilityLevel.MAJORITY)
            .expirationTime(Duration.ofSeconds(100))
            .build();
    Transactions transactions = Transactions.create(cluster, config);

    try {
        transactions.run((ctx) -> {
            QueryResult result = cluster.query(query);

            for (JsonObject docId : result.rowsAs(JsonObject.class)) {
                if (!docId.containsKey("id"))
                    continue;
                String id = docId.getString("id");
                TransactionGetResult res = ctx.get(collection, id);
                JsonObject doc = res.contentAs(JsonObject.class);
                doc.put("customer_rating", value);
                ctx.replace(res, doc);
            }
            System.out.println("Update finished, waiting for commit!");
            try {
                Thread.sleep(20000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            ctx.commit();
            System.out.println("Success");
        });
    } catch (TransactionFailed e) {
        System.err.println("Transaction " + e.result().transactionId() + " failed");

        for (LogDefer err : e.result().log().logs()) {
            System.err.println(err.toString());
        }
    }
}
```

Ilustracija 10 - Kod longRunningTransaction funkcije

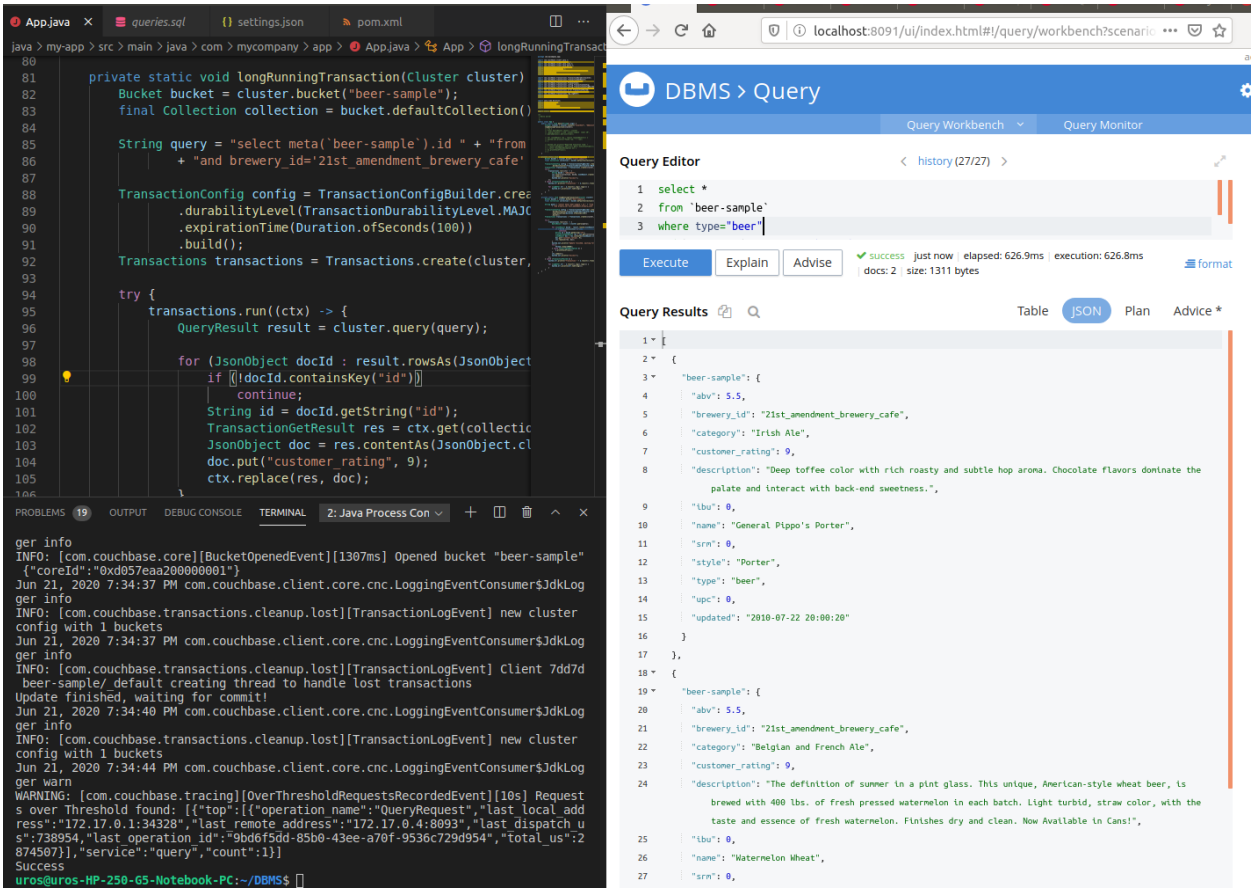
Ukoliko pokrenemo u toku izvršenja transakcije (Ilustracija 11) dođe do ažuriranja istih podataka van transakcije (Ilustracija 12), Couchbase će dozvoliti izmenu podataka, ali će oni biti prepisani nakon izvršenja transakcije. Rezultat paralelnog izvršenja ove dve operacije se može videti na Ilustracija 13.



Ilustracija 11 - Izvršenje demo aplikacije koja postavlja vrednost customer_rating atributa na 9

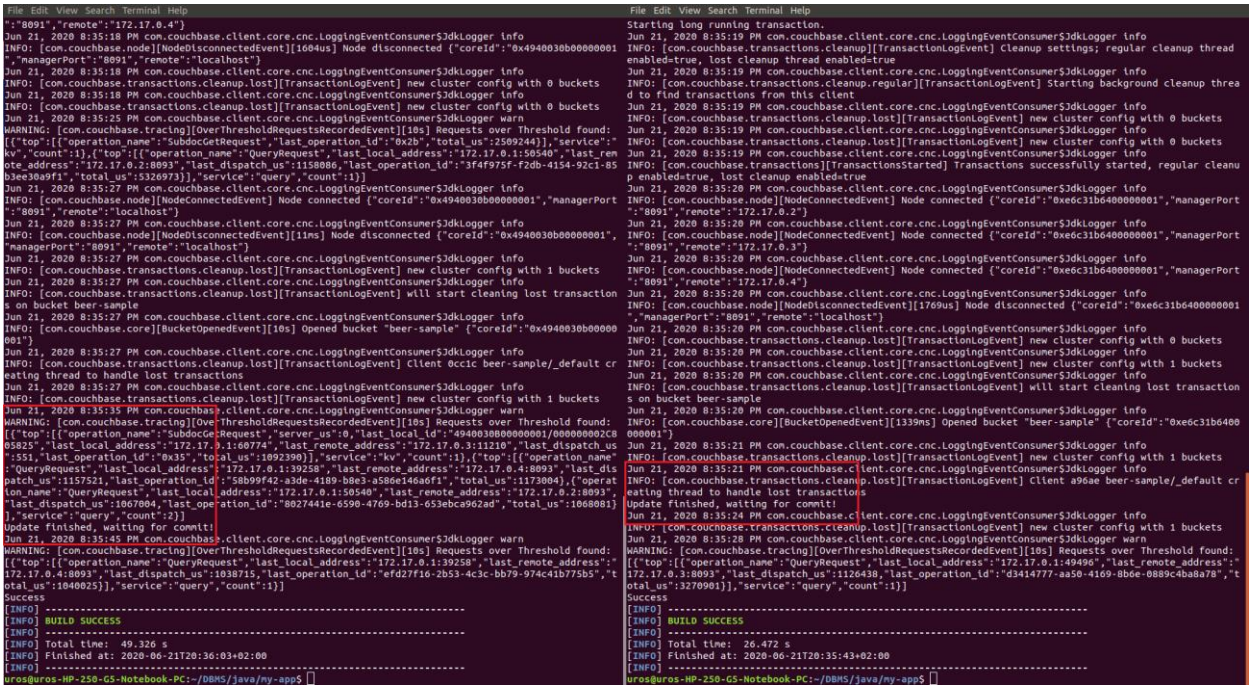


Ilustracija 12 - Ažuriranje koje obavlja demo aplikacija napisano preko upita izvršeno uz pomoć Web klijenta bez upotrebe transakcija



Ilustracija 13 - Rezultat nakon paralelnog izvršenja Web klijenta i demo aplikacije

Ukoliko se imamo dve transakcije koje istovremeno ažuriraju iste podatke, transakcija koja prva pristupi podacima će prva obaviti ažuriranje. Rezultat paralelnog pristupa istim podacima od strane dva klijenta se može videti na Ilustracija 14. Desni klijent je prvi pristupio podacima i obavio ažuriranje. Za to vreme levi klijent čeka na završetak transakcije desnog klijenta i odmah nakon toga obavlja svoje ažuriranje.



Ilustracija 14 - Paralelno izvršenje dve transakcije pomoću dva demo klijenta

4 Zaključak

Distribuirane baze podataka su nezamenjiv alat za rešavanje problema savremenih informacionih sistema. Ovakvim sistemima postižu se „state of the art“ rezultati u pogledu skaliranja, pouzdanosti i dostupnosti sistema.

U ovom radu dat je pregled osnovnih koncepata distribuiranih sistema za upravljanje baza podataka i kao primer jednog takvog sistema predstavljen je Couchbase server. Videli smo podelu sistema na osnovu autonomije, distribuiranosti i heterogenosti podataka. Kao dva istaknuta primera distribuiranih baza podataka, predstavljene su arhitekture potpuno distribuiranih sistema i federativnih sistema baza podataka. Videli smo vrste fragmentacije podataka u distribuiranim sistemima – vertikalna, horizontalna, kombinovana, i načine replikacije i alokacije podataka - potpuna replikacija, delimična replikacija i sistemi bez replikacije podataka. Dali smo pregled obrade upita u distribuiranim sistemima i kako možemo smanjiti količinu podataka koji se razmenjuju između čvorova distribuiranih sistema primenom tehnike SEMIJOIN-a. Načini za implementaciju transakcija u distribuiranim sistemima uključuju dvofazni i trofazni commit, pri čemu trofazni commit rešava probleme koji se javljaju kod dvofaznog commit-a, ali zahteva veći mrežni saobraćaj. Videli smo načine za implementaciju pristupa podacima kod distribuiranih baza podataka, pri čemu su dve glavne varjante tehnika primarne kopije i metoda zasnovana na glasanju. Nakon toga, dali smo načine za implementaciju kataloga distribuirane baze podataka - centralizovani, potpuno replicirani i delimično replicirani.

Couchbase je jedan od primera distribuiranih baza podataka. Spada u grupu document baza podataka, mada se može koristiti i kao „in memory cache“. Couchbase se može podići kao jedinstveni server, klaster čvorova ili mega-klaster (međusobno povezani klasteri). Na svakom čvoru se izvršavaju jedna ili više vrsta servisa. Po CAP teoremi spada u CP baze ako se podigne kao jedan klaster ili u AP ukoliko se podigne kao više međusobno povezanih klastera. Couchbase ima podršku Read Commited distribuirane transakcije.

5 Literatura

[1] "Couchbase dokumentacija," [Online]. Available: https://docs.couchbase.com/server/6.5/introduction/intro.html.

[2] "Couchbase blog o transakcijama," [Online]. Available: https://blog.couchbase.com/distributed-multi-document-acid-transactions-in-couchbase/.

[3] S. B. N. Ramez Elmasri, Fundamentals of Database Systems 6th edition.