Apache Cassandra – pregled arhitekture i skladišta podataka

Seminarski rad iz predmeta Sistemi za upravljanje bazama podataka

Student: Aleksa Ravnihar, br indeksa: 684

Mentor: Prof. dr Aleksandar Stanimirović

April, 2020.

Sadržaj:

Uvod 3

SQL i NoSQL baze podataka 3

ACID svojstva i CAP teorema 3

Podela NoSQL baza podataka 4

Apache Cassandra 4

Osobine *Apache Cassandra* DBMS-a 5

Apache Cassandra i relacione baze podataka 5

Apache Cassandra model podataka 6

Apache Cassandra – distribuirana arhitektura 7

Protokoli za komunikaciju čvorova 9

Cassandra *storage engine* 9

Detalji implementacije *Cassandra* DBMS čvora 10

Upis podataka u *Cassandra* čvor 12

Čitanje podataka iz *Cassandra* čvora 13

Apache Cassandra i konzistentnost operacija čitanja i upisa 14

Zaključak 16

Reference 17

Uvod

Cilj ovog rada je da objasni unutrašnju organizaciju, arhitekturu i protokole koje implementira *Apache Cassandra* skladište podataka. Najpre ćemo dati kratak pregled i podelu popularnih skladišta podataka i opisaćemo ključne razlike između SQL i NoSQL baza podataka. Nakon toga ćemo dati detalje implementacije *Apache Cassandra* distribuirane baze podataka, objasnićemo fizičku strukturu čvora, način na koji čvorovi komuniciraju u okviru *Cassandra* klastera, kao i način na koji se vrši particionisanje, upis i čitanje podataka iz klastera, kao i kontrola konzistentnosti podataka u distribuiranom okruženju.

SQL i NoSQL baze podataka

Relacione baze podataka (RDBMS, engl. "*relational database management system*") su u prošlosti korišćene za skladištenje većine podataka koje su tadašnje aplikacije obrađivale. Relacioni model podataka predložen je prvi put od strane *E. F. Codd*-a u radu "*Relational model of Data for large Data banks*", objavljenom 1970. godine. Ovim počinje period u kome dominantnu ulogu imaju relacione baze podataka i u okviru kog se razvija većina poznatih RDBMS sistema. Neka od poznatijih i popularnijih RDBMS sistema su Oracle SQL server, MySQL, PostgreSQL, Microsoft SQL Server. Pored ovih DBMS sistema, razvio se i standard za pristup i manipulaciju podacima iz baze podataka u vidu SQL jezika (engl. "*structured query language*").

Iako su se relacione baze podataka pokazale kao odlično rešenje za većunu tadašnjih aplikacija, razvoj i evolucija Web i mobilnih aplikacija je zahtevao da se promeni pristup za skladištenje podataka. Moderne aplikacije rade sa mnogo većom količinom podataka kojima se pristupa sa svih uređaja koji imaju pristup Internetu. Takođe, aplikacije više nisu centralizovane, i vrlo često se izvršavaju na *Cloud*-u. Sve ovo je dovelo do razvoja nove vrste skladišta podataka, NoSQL baza podatala. U prvo vreme se pod ovim pojmom podrazumevalo "*Not SQL*" baze podataka, ali su vremenom NoSQL sistemi dobijali podršku za jezike sličene SQL upitnom jeziku, pa je izraz NoSQL prerasao u "*Not only SQL*" baze podataka.

ACID svojstva i CAP teorema

Glavne razlike između SQL i NoSQL baza podataka možemo videti kroz ACID svojstva i CAP teoremu.
ACID svojstva su vezana za relacione baze podataka, i predstavljaju skup osobina koje svaki DBMS mora da ispuni prilikom izvršavanja neke transakcije. Svako slovo predstavlja po jednu osobinu:

- Atomičnost (engl. „*Atomicity*"): Prilikom obrade transakcije, neophodno je da se svi delovi transakcije uspešno izvrše. Samo tada se smatra da je transakcija uspešno obavljena. Ukoliko neki deo transakcije ne uspe da se izvrši, smatra se da je transakcija neuspela, i sistem se vraća u stanje pre izvršenja transakcije.
- Konzistentnost (engl. „*Consistency*"): Svi podaci skladišteni u bazi podataka moraju da budu validni u svakom trenutku. Svaka transakcija prevodi stanje iz jednog u drugo validno stanje.
- Izolovanost (engl. „*Isolation*"): Izolovanost garantuje da se svaka transakcija izvršava u izolaciji, odnosno bez mogućnosti da bilo koja druga promena ili transakcija utiče na nju.

- Trajnost (engl.“*Durability“*): Svaka uspešno izvršena transakcija, ostavlja trajno promene u bazi podataka. Ukoliko se desi da nakon izvršenja transakcije, dođe do greške ili pada sistema, sigurno će sve promene iz transakcije biti vidljive nakon što se sistem oporavi.

NoSQL baze podataka su drugačije. Mahom su projektovane tako da osiguraju visoku dostupnost podataka kroz ceo klaster. Kako bi se to postiglo, obično se žrtvuju određene osobine DBMS-a iz ACID modela. Zato se za distribuirane baze podataka obično vezuje CAP teorema, koja kaže da u distribuiranim sistemima, kod kojih postoji mogućnost za otkaz nekog čvora, projektovani DBMS ne može u potpunosti da ispuni više od 2 osobine:

- Konzistentnost: Svaka operacija čitanja podatka mora da vrati validnu vrednost.
- Dostupnost: Svaki zahtev klijenta upućen ka DBMS-u mora biti obrađen, a odgovor vraćen klijentu. Iako je rezultat poslat klijentu, ovo ne označava da odgovor mora biti validan (konzistentan).
- Tolerancija na particionisanje (engl. „*partition tolerance*“): Ova osobina se odnosi na sposobnost sistema da nastavi sa radom ukoliko dođe do greške na mreži, i deo sistema/mreže bude nedostupan neko vreme. Dozvoljeno je da zahtevi koji su se dogodili u trenutku otkaza ne budu obrađeni, ali je neophodno da sistem može nastaviti sa radom.

Mi ćemo se u ovom radu fokusirati na *Apache Cassandra* DBMS, i kroz pregled njene arhitekture i načina funkcionisanja DBMS-a pokazaćemo da ona ispunjava AP svojstva CAP teoreme (dostupnost i toleranciju na particionisanje), dok se konzistentnost može podešavati u zavisnosti od zahteva.

Podela NoSQL baza podataka

U zavisnosti od vrste podataka koje se skladište u bazi, razvijene su različite NoSQL baze podataka:

- Baze orijetntisane za skladištenje ključ-vrednost parova podataka (engl. “*key-value store*”) kao što su *Redis* i *Memcached*.
- Baze koje rade sa tabelama sličnim SQL tabelama, ali koje su dosta fleksibilnije, i podržavaju fleksibilan broj kolona (engl. “column-oriented”) kao što su *Google Big Table*, *HBase*, *Apache Cassandra*.
- Beze koje rade sa dokumentima (engl. „*document-oriented*“) kao što su *MongoDB*, *CouchDB*.

Apache Cassandra

Cassandra predstavlja NoSQL *column-store* DBMS razvijen od strane *Apache* kao projekat otvorenog koda (engl. “*open-source*”). Ideja za *column-store* DBMS javila se zapravo u kompaniji *Facebook* gde je razvijena prva verzija ovog DBMS-a - razvili su ga *Avinash Lakshman* i *Prashant Malik*. Potreba za novim DBMS-om se javila jer dotadašnja SQL rešenja nisu davala dobre rezultate u sistemu za razmenu i pretragu poruka u *Messanger* aplikaciji. Ideja je bila da se dizajnira sistem koji kombinuje dobre osobine

Amazon Dynamo DBMS-a i Google BigTable rešenja, kako bi se implementirao distribuirani, visoko dostupni DBMS koji se lako skalira.

Neke od osobine kojima se teži prilikom dizajniranja DBMS sistema poput *Apache Cassandra* su:

- Mala latenciju prilikom operacija čitanja i upisa
- Fleksibilnost podataka koji se čuvaju u bazi podataka
- Balansiranje opterećenja i jednostavno proširenje klastera (skalabilnost)
- Visoka dostupnost i redudantnost podataka (otpornost na otkaz)
- Linearna zavisnost količine obrađenih podataka i broja čvorova (procesorskih jedinica) u klasteru

Osobine *Apache Cassandra* DBMS-a

U nastavku su date neke od glavnih osobina *Apache Cassandra* DBMS-a:

- Particionisana i distribuirana baza podataka: DBMS radi sa klasterom računara, gde su svi podaci particionisani i dodeljeni određenom čvoru u klasteru. Ne postoji glavni (engl. "*master*") čvor, već je sistem dizajniran kao „*masterless*" sistem. Ovo znači da klijent koji komunicira sa bazom podataka (bilo da je to administrator, ili aplikacija koja koristi bazu podataka), može da se poveže na bilo koji čvor iz klastera, a taj čvor će propagirati zahtev do čvora na kom se nalaze podaci koji su zahtevani.
- Kako bi se postigla otpornost na otkaz, svaki podatak se čuva na većem broju čvorova, da otkaz čvora ne bi doveo do gubljenja podataka.
- Distribuiranost podataka: *Cassandra* obezbeđuje sistem za automatsko distribuiranje podataka. Na ovaj način korisnik ne treba da vodi računa kako i gde skladištiti podatke, već sistem sam odlučuje kako će distribuirati podatke na veći broj čvorova.
- Skalabilnost sistema: Performanse sistema, koje se najčešće ogledaju u broju operacija koje baza podataka izvrši u sekundi, ili u količini podataka koja se može obraditi u sekundi (engl. „*throughput*"), *Cassandra* je dizajnirana tako da se dodavanjem novih čvorova u klasteru, linearno povećava i broj obrada koje sistem izvrši u sekundi.
- CQL – „*Cassandra Query Language*": CQL je jezik koji predstavlja glavni način za komunikaciju klijenta i *Cassandra* DBMS-a. Ne samo da ime podseća na široko poznati SQL, već je i sintaksa veoma slična. Najlakši način za komunikaciju sa sistemom je preko CQL terminala (*cqlsh* – „*cql shell*") preko koga mogu da se definišu i pretražuju korisničke tabele. Pored ovoga, razvijeni su i grafički alati za interakciju sa DBMS-om.

Apache Cassandra i relacione baze podataka

Iako i u *Cassandra* terminologiji korisnik manipuliše podacima koji su smešteni u tabelama (kasnije u radu će biti detaljnije objašnjeno na kakve tabele se misli), implementacija ovih tabela se veoma razlikuje od klasičnih RDBMS tabela. Takođe, *Cassandra* je distribuirana baza podataka, i različite tabele se čuvaju na različitim čvorovima u oviru klastera. Kako CQL jezik podržava samo operacije koje se

izvršavaju lokalno, na jednom čvoru, nije moguće implementirati operacije poput spajanja tabela (engl. "*join*"). Pored spajanja tabela, bilo kakve operacije koje se odnose na veći broj particija (odnosno veći broj klastera), nisu dozvoljene. Ovo ima za posledicu da *Cassandra* takođe ne dozvoljava postojanje stranih ključeva u okviru svoje šeme. Iako sam DBMS ne podržava ovakve operacije, korisnik ima slobodu da ovakve operacije implementira na klijentskoj strani, nakon pribavljanja podataka iz različitih tabela.

Apache Cassandra model podataka

Pre nego što detaljno opišemo arhitekturu *Cassandra* DBMS-a, ukratko ćemo opisati model podataka koji je implementiran. Jedan od načina za definisanje i manipulaciju sa podacima je korišćenje CQL jezika. Sintaksa i način funkcionisanja ovog jezika dosta podsećaju na SQL jezik: korisnik kreira određene tabele, definiše vrste i kolone za svaku tabelu, može da pretražuje tabele. Čak se i rezultati koje vraća server obično prikazuju tabelarno. Iako krajnja reprezentacija podataka i način definisanja podataka sugerišu da u pozadini DBMS radi sa tabelama kao i RDBMS sistemi, *Cassandra* u pozadini koristi drugačiji sistem kako bi manipulisala podacima.

Osnovne komponente kojima se manipuliše prilikom rada sa DBMS-om su:

- Prostor ključeva (engl. „*key space*"): Služi da definišemo kako će se pamtiti podaci u distribuiranom sistemu (koliko će kopija da čuvano u sistemu). Svaki prostor ključeva sadrži jednu ili više tabela. Kako bi se napravila analogija sa RDBMS sistemima, prostor ključeva konceptualno odgovara pojmu jedne baze podatala.
- Familija kolona (engl. „*Column family*"): Definiše šemu za određenu kolekciju koja se pamti u bazi podataka (odgovara tabelama u RDBMS sistemu).
- Particija: Particija definiše obavezan deo primarnog ključa za sve vrste unutar jedne particije. Ključ particije se koristi prilikom distribuiranja podataka kroz klaster (određuje čvor gde će podatak biti smešten).
- Vrsta: Skup kolona i vrednosti za svaku kolonu. Identifikuje se pomoću primarnog ključa koji se sastoji iz ključa particije i opciono ključa klastera.
- Kolone: Svakoj koloni je pridružen određeni tip podatka, ključ kolone (ime konkretne kolone) i vrednost upisana u kolonu.

Cassandra model podatka definiše nekoliko vrsta ključa:

- Ključ particije: Naziva se još i primarni ključ. Služi kako bi se odredila lokacija particije u klasteru. Ključ sam po sebi može da predstavlja i podatak. Moguće je definisati ključ koji se sastoji iz jedne kolone, ali i ključ koji se sastoji iz više kolona.
- Kompozitni ključ: Ključ particije koji se sastoji iz više kolona.
- Složeni ključ (engl. „*Compound key*"): Ključ particije koji se sastoji od više kolona, ali se ne koriste vrednosti svih kolona prilikom particionisanja (kao kod kompozitnog ključa).

- Ključ klastera (engl. „*Cluster key*“): Ukoliko prilikom definisanja particije, definišemo nekoliko ključeva particije, prvi ključ se tretira kao ključ particije, dok su ostali ključevi klaster ključevi.

U pozadini,sistem ne radi sa tabelama, već sa sortiranim *hash* mapama. Sortiranost se postiže ključem particije, i klaster ključevima. Ovo omogućava da se brzo skenira opseg vrednosti za neku particiju. Tabelama (familijama kolona) može u svakom trenutku da se dodaju nove kolone (nije strogo definisana struktura tabele).

Neke od prednosti korišćenja ovakvog modela podataku su:

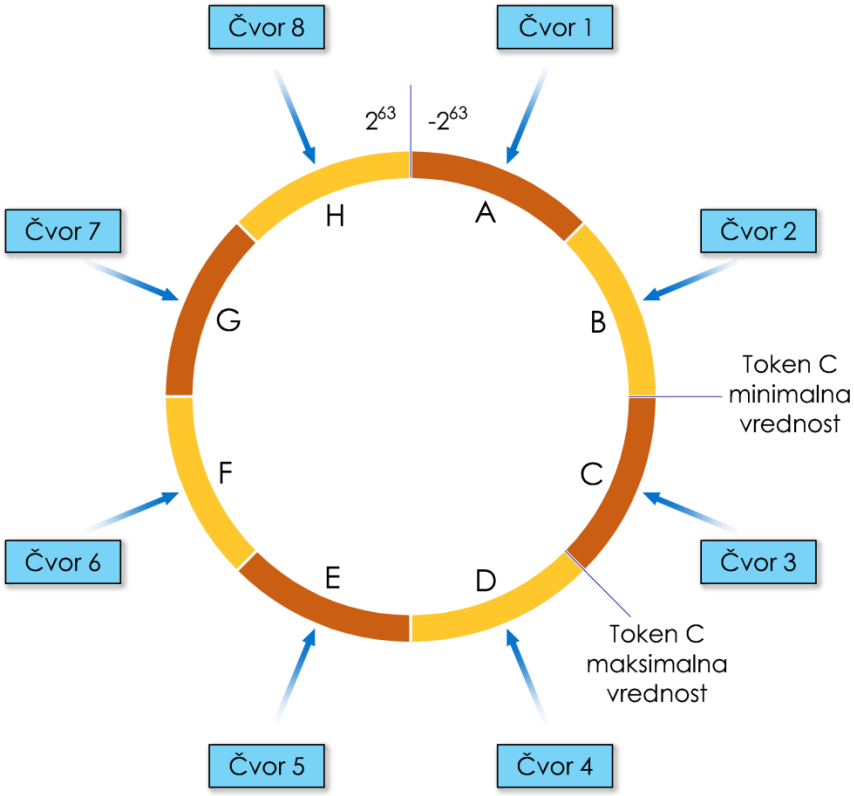
- Postiže se dobra distribuiranost particija kroz ceo klaster.
- Balans opterećenja: Podaci se ravnomerno raspoređuju kroz ceo klaster, a operacije nad particijom se izvršavaju lokalno, samo na jednom čvoru. Prilikom čitanja particije, dovoljno je čitanje sa jedne particije, pa nema potrebe za spajanjem podataka sa različitih čvorova.

Apache Cassandra – distribuirana arhitektura

Apache Cassandra je distribuirana baza podataka koja se izvršava na klasteru računara. DBMS je projektovan tako da svaki čvor u klasteru može da odgovori na zahtev od strane klijenta (takozvana “*multi master*”, “*serverless*” arhitektura). Postoji nekoliko komponenata koje se izvršavaju na svakom *Cassandra* čvoru koje određuju kako se pronalazi čvor koji je odgovoran za određeni podatak. Čvor koji primi zahtev od klijenta i koji je zadužen za kontrolu izvršenja zahteva i vraćanje rezultata klijentu obično se naziva čvor koordinator.

Kako bi osigurao dostupnost podataka i otpornost na otkaze, *Cassandra* DBMS vrši repliciranje podataka i njihovu distribuciju na različite čvorove u okviru klastera. Ideja je da ukoliko dođe do otkaza nekog čvora u klasteru, kopije podataka mogu biti pročitane iz nekog drugog čvora koji takođe sadrži kopiju datog podatka. Kada čvor u okviru klastera primi zahtev za upis nekog podatka, mora “nekako” na osnovu tog zahteva da “zaključi” kom čvoru u klasteru treba da prosledi zahtev za upis, odnosno da odredi čvor gde će podatak biti upisan.

Svaki podatak koji se upisuje u bazu podataka mora da ima svoj ključ particije (engl. “*partition key*”). Na osnovu vrednosti ključa određuje se čvor u okviru klastera kome se prosleđuje dati podatak. Postoji više različitih politika particionisanja, koje se mogu implementirati u zavisnoti od nekih specijalnih zahteva, topologije i veličine klastera. Zajedničko za sve politike je da se na osnovu ključa generiše jedna *long* (64-bitna vrednost) koja se u *Cassandra* terminologiji naziva *token*. Skup svih vrednosti tokena obično se predstavlja kao “*token ring*”, i prikazan je na sledećoj slici:



Slika 1: "Token ring" struktura

Svakom fizičkom čvoru u klasteru se dodeljuje određeni skup vrednosti sa *token-ring* strukture, pa je tako za svaki čvor definisana miniumalna i maksimalna vrednost tokena koji pripada datom fizičkom čvoru. U prvim verzijama DBMS-a, prilikom konfigurisanja klastera, manuelno su se podešavali vrednosti tokena za svaki čvor u klasteru.Na ovaj način je svaki fizički čvor dobijao jedan token, odnosno jedan deo vrednosti iz *token-ring* strukture. Ovakav sistem bi davao dobre performanse ukoliko bi postojao veliki broj fizičkih čvorova u klasteru. Ukoliko imamo mali broj čvorova, skaliranje sistema, odnosno dodavanje novih čvorova dovelo bi do dizbalansa. To se dešava zato što ne bi bilo moguće da se ravnomerno raspodele svi tokeni novim čvorovima u klasteru. Kako bi se zadržala izbalansirana podela tokena među čvorovima u klasteru, morali bismo da dodamo specifičan broj novih čvorova, koji nam možda nije potreban. Novije verzije *Cassandra* DBMS-a, implementiraju takozvane vertuelne čvorove (engl. "*virtual nodes*") kako bi rešile ovaj problem. Ključni pojmovi prilikom implementacije virtuelnih čvorova su:

- *Token*: Predstavlja skup vrednosti sa *token-ring* strukture.
- *Endpoint*: Fizička IP adresa i port na mreži u okviru klastera.
- *Host* ID: Jedinstveni identifikator za svaki fizički čvor u mreži. Obično se sastoji od *Endpoint*-a i jednog ili više tokena.
- Virtualni čvor ("*virtual node, vnode*"): Predstavlja token u token krugu kome je pridružen određeni fizički čvor.

U pozadini sistem koristi mapu tokena (engl. "*token map*") koja definiše mapiranja svakog tokena u određeni *Endpoint*. Na ovaj način postižemo da je za jedan fizički čvor vezan veći broj tokena. Prednost

ovakvog sistema je da se prilikom dodavanja ili uklanjanja čvorova iz klastera podaci ravnomerno distribuiraju po svim čvorovima. Takođe, ukoliko dodje do otkaza nekog čvora, zahtevi se ravnomernije propagiraju drugim čvorovima (postoje mehanizmi gde je sistem koji razrerašava situacije otkaza čvora „svestan“ postojanja *tokena*, pa može optimalnije da raspodeli zahteve čvorovima). Sa druge strane, postoje i određene mane sistema sa virtuelnim čvorovima:

- Potrebno je za svaki *token* dodati još 2 *(*RF* - 1) dodatnih *tokena* na token-*ring* strukturi, kako bi se ispoštivao faktor replikacije (*RF*) i kako bi ostao dodatni token za dalju alokaciju.
- Operacije održavanja klastera postaju složenije.
- Performanse operacija nad tokenima mogu biti degradirane.

Protokoli za komunikaciju čvorova

Kako bi čvorovi u okviru *Cassandra* klastera mogli da pravilno funkcionišu i razmenjuju informacije, svaki čvor u okviru klastera učestvuje u određenim protokolima za komunikaciju, i to u:

- *Gossip* protokolu
- Sni*t*ch protokolu

Cilj Gossip protokola je da otkrije da li je došlo do greške na nekom čvoru u okviru klastera, odnosno da otkrije trenutno nedostupne čvorove. Kako bi se što više redukovao mrežni saobraćaj u klasteru, komunikacija se obavlja između dva čvora (engl. „*peer to peer*“). Protokol se zasniva na tome da čvor (obično se naziva „*seed node*“) izabere do tri proizvoljna čvora u klasteru, i na svaku sekundu pošalje informacije o svom statusu, i informacije koje je dobio od drugih čvorova putem *Gossip* protokola. Svaka poruka nosi u sebi verziju, tako da svaki čvor može na osnovu nje da odbaci informacije koju se „zastarele“. Ovakvom komunikacijom se promene brzo propagiraju kroz klaster bez opterećivanja mreže (ne koriste se *broadcast* poruke). Nije potrebno da svi čvorovi u klasteru šalju informacije o trenutnom statusu. Obično se bira jedna grupa čvorova koji će inicirati protokol i slati poruke. Preporučeno je da to uvek budu isti čvorovi kako bi se izbegli problemi u komunikaciji. Takođe, ukoliko klaster sačinjava više *data* centara, makar jedan čvor iz svakog *data* centra treba da bude „*seed-node*“. Ukoliko bi svaki čvor bio „*seed-node*“, došlo bi do degradiranja perforrmansi protokola.

Drugi protokol za komunikaciju je *Snitch* protokol, koji se koristi kako bi svaki čvor dobio informaciju o lokaciji i udaljenosti ostalih čvorova u klasteru (odnosno da čvorovi nauče topologiju klastera). Ove informacije se koriste kada se raspoređuju replike podataka, i prilikom prosleđivanja zahteva za upis i čitanje podataka. *Cassandra* obično pokušava da najviše jednu kopiju drži u jedoj polici (engl. „*rack*“). U zavisnosti od mrežne topologije i vrste *Cloud* sistema na kome radi *Cassandra* klaster, razvijeni su različiti *Snitch* protokoli koji se mogu podesiti prilikom konfigurisanja baze.

Cassandra *storage engine*

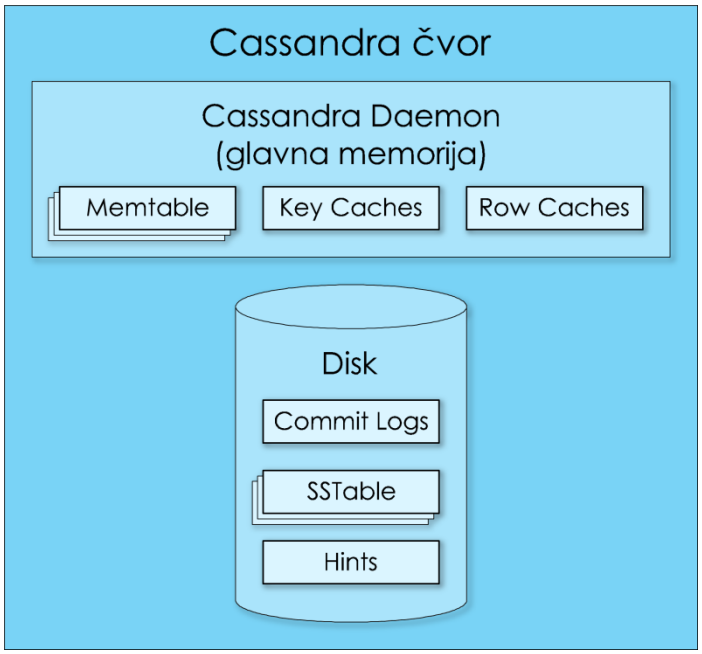
Glavna struktura podataka koju koristi *Cassandra* su LSM stabla (engl. „*Long structred Merge Tree*“). LSM stabla se koriste kada nam je potreban brzi pristup podacima (najčeće se podacima pristupa preko indeksa, radi brze pretrage), u sistemima gde je dominantna operacija upisa podataka. LSM stabla

zapravo predstavljaju verziju stabla za pretragu (engl. „*search tree*“), a struktura interno održava listu ključ-vrednost kako bi omogućio brzu pretragu. Za razliku od *Cassandre*, relacione baze za čuvanje podataka koriste B stabla.

Još jedna karakteristika *Cassandra* DMBS-a je da ne koristi „čitanje pre upisa“ politiku za rešavanje konflikta prilikom obrade većeg broja transakcija. Upisi se kod *Cassandre* obavljaju potpuno nezavisno, odnosno bez obzira na stanje podataka u bazi. Umesto toga, *Cassandra* svaku operaciju posmatra kao jednu novi promenu, i svaku promenu dodaje u jednu *in-memory* strukturu, u koju se samo nadovezuju nove promene, bez obzira na prethodno stanje podataka. Kada ta struktura dostigne određenu veličinu, ceo njen sadržaj se prebacuje na disk, i postaje nepromenljiv (engl. „*immutable*“). Ovakva implementacija omogućava da se upis podataka kod *Cassandre* obvavlja veoma brzo. Prilikom čitanja podataka, potrebno je izvršiti određenu obradu nad sačuvanim podacima, kako bi se utvrdilo korektno stanje podataka koji se zahtevaju. Postoji i mogućnost da se proveri stanje određenog podatka pre upisa. To su takozvane CAS opearcije(engl. “*check and* set”) koje su u DBMS-u implementirane pomoću PAXOS protokola.

Detalji implementacije *Cassandra* DBMS čvora

Već je rečeno da se Cassandra izvršava na klasteru računara, prilikom čega ne postoji glavni (master) čvor, već svaki čvor može da prihvati zahtev od klijenta, i da kontaktira neki drugi čvor iz klastera koji sadrži tražene podatke. Na svakom čvoru se izvršava *Cassandra* demon (engl. “*daemon*”) koji održava stanje određenih struktura (u glavnoj memoriji i na disku) koje se koriste prilikom rada DBMS-a, i koji komunicira sa ostalim čvorovima iz klastera. Na sledećoj slici je prikazana arhitektura jednog *Cassandra* čvora:



Slika 2: Arhitektura *Cassandra* čvora

Commit Log

Commit Log je struktura koja se nalazi na disku svakog čvora. Svaki upis koji izvršava određeni čvor, beleži se u Commit Log-u. Postoji jedinstveni Commit log na nivou čvora, tako da se tu nalaze zapisi o promenama za sve particije koje se nalaze na datom čvoru. Brzina upisa kod *Cassandra* DBMS-a se ogleda u tome da se upis smatra izvršenim čim se doda određena stavka u *Cimmit Log*. *Commit Log* je komponenta DBMS-a koja služi kao mehanizam za oporavak: ukoliko dođe do pada mreže ili nekog čvora, moguće je da se izvrši rekonstrukcija stanja čvora na osnovu sadržaja Commit log-a.

SSTables

SSTable (engl. „*Sorted String Table*“) je struktura u kojoj se trajno čuvaju podaci na disku svakog čvora. Za razliku od *Commit log*-a koji je jedinstven na nivou čvora, za svaku tabelu koja se pamti na određenom čvoru se generiše po jedna *SSTable* struktura. Nije dozvoljena izmena podataka u okviru ove strukture, već samo nadovezivanje novog sadržaja. Kada se vrši upis, podaci se ne upisuju direktno u *SSTable*, nego periodično iz *Memtables* strukture. Svaka *SSTable* struktura sadrži više fajlova, i to:

- *Data.db*: Sadržaj *SSTable* strukture, odnosno podaci koji se dobijaju iz *Memtables* strukture.
- *Index.db*: Indeks za svaki ključ particije u okvru Data.db fajla.
- *Summary.db*: Redukovana verzija Index.db fajla (obično svaki 128-i indeks).
- *Filter.db*: *Bloom* filter za svaki ključ particije iz *SSTable* strukture. *Bloom* filter je struktura koja se nalazi u glavnoj memoriji i pomoću koje se određuje da li se određena vrsta iz baze nalazi u *Memtables* strukturi, da se ne bi učitavali podaci sa diska (iz *SSTable* strukture).
- *CompressionInfo.db*: Meta podaci o komprimovanom Data.db fajlu.
- *Statistics.db*: Meta podaci o *SSTable* strukturi (ključ za particije, ključ za klastere, „*timestamps*“ – vreme o svakoj unetoj promeni nad podacima, „*tombstones*“ – podaci koji su izbrisani iz baze).
- *Digest.crc32*: Kontrolna suma Data.db fajla (CRC-32).
- *TOC.txt* (engl. „*table of content*“): Lista svih fajlova koji sačinjavaju *SSTable)*.
-

Memtables

Memtable je struktura koja se nalazi u glavnoj memoriji i koja se koristi za baferovanje upisa u *SSTable* strukturu. Za svaku aktivnu tabelu u čvoru održava se posebna *Memtable* struktura. Kada stigne zahtev za određeni upis u tabelu, on se ne beleži u *SSTable* direktno, već u *Memtable*. Kada *Memtable* struktura dostigne određenu veličinu, njen sadržaj se dodaje u odgovarajući *SSTable* fajl, a sadržaj *Memtable* strukture se prazni. Na ovaj način se ubrzava proces upisa, jer je upis u glavnu memoriju mnogo brži, nego upisu na disk, koji se u ovom slučaju odigrava samo periodično.

Hints

Hints struktura predstavlja još jedan mehanizam za oporavak. Kada neki čvor iz klastera primi određeni zahtev od klijenta, on najpre treba da pronađe čvor iz klastera kome treba da prosledi zahtev. Ukoliko dođe do greške prilikom rada klastera (zbog otkaza traženog čvora, velikog mrežnog saobraćaja saobraćaja, itd.) tako da čvor koji je primio zahtev ne može da prosledi zahtev željenom čvoru, taj čvor beleži u svoj *Hint* bafer dati zahtev. Kada se pomoću protokola za komunikaciju između čvorova otkrije

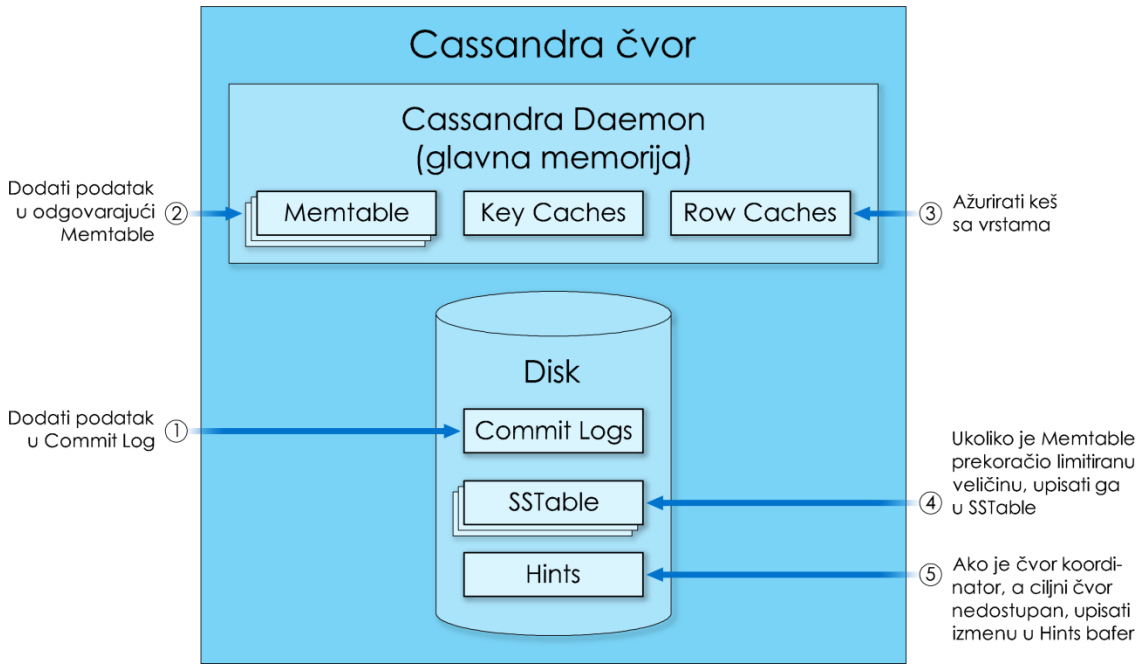
da je željeni čvor ponovo dostupan, čvor koji je koordinisao zahtev će poslati čvoru baferovani zahtev, tako da željeni čvor može da ažurira podatke.

Key Caches i Row Caches

Strukture koje se koriste za keširanje podataka kojima se skoro pristupalo kako bi se povećale performanse prilikom čitanja.

Upis podataka u *Cassandra* čvor

Na sledećoj slici je prikazan redosled operacija prilikom upisa podataka u *Cassandra* tabelu:

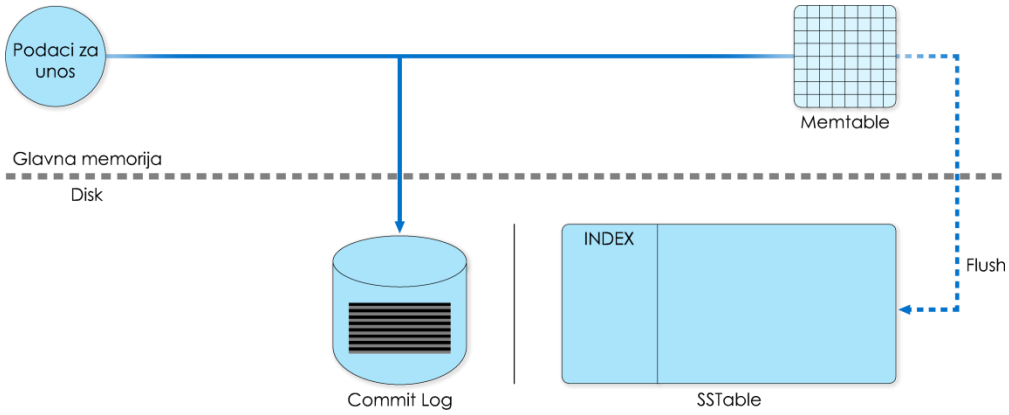


Slika 3: Upis u *Cassandra* čvor

Prilikom svakog upisa podataka u čvor, izvršavaju se sledeći koraci:

1. Najpre se podaci upisuju u *Commit Log* kako bi se osiguralo da ne dodje do gubljenja podataka u slučaju otkaza.
2. Kopija podataka se takođe upisuje u *Memtable* strukturu.
3. Za slučaj da se koristi keširanje podataka, proverava se da li kopije podataka postoje u kešu, i invalidiraju se. Novi podaci se upisuju ponovo u keš.
4. (Opciono) Ukoliko se dogodi da veličina *Memtable* strukture prekorači odrađenu vrednost, sadržaj *Memtable* strukture se upisuje na disk kao *SSTable* struktura. Može da se dogodi da se kreiraju više *SSTable* struktura za jednu particiju na čvoru. Postoje protokoli koji se koriste kako bi se se ove strukture objedinile u jednu strukturu.

5. (Opciono) Ukoliko je dozvoljena opcija „*hinted handoffs*“ , proverava se da li je dostupan čvor kome treba da se prosledi klijentski zahtev. Ukoliko nije, zahtev se upisuje u *Hints* strukturu i prosleđuje se ciljnom čvoru u trenutku kada postane dostupan.

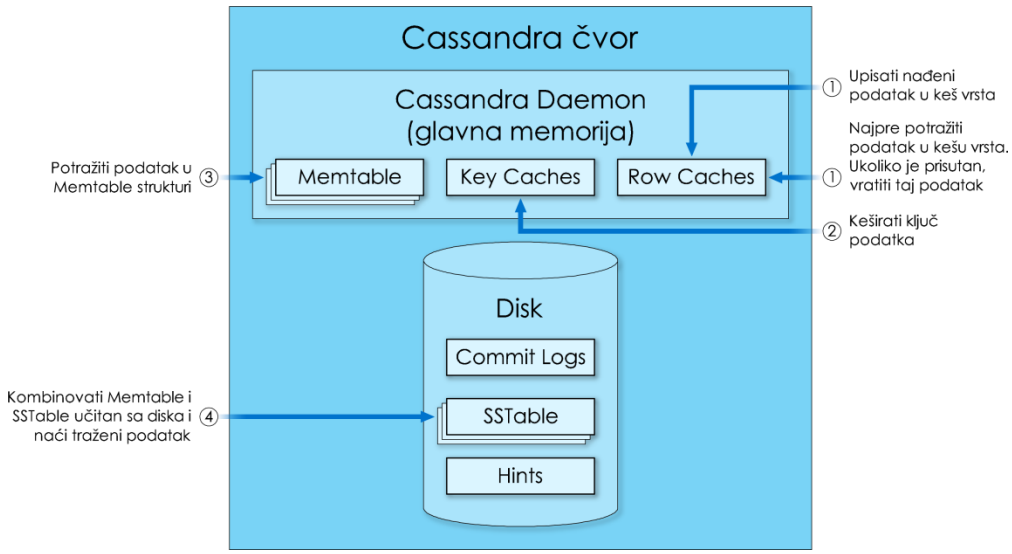


Slika 4: Redosled operacija pri upisu u *Cassandra* čvor

Čitanje podataka iz *Cassandra* čvora

Za razliku od operacije upisa, koja se izvršava veoma brzo, operacija čitanja podataka je dosta složenija i zavisi od podešene politike za konzistentnost podataka prilikom čitanja. U zavisnosti od podešene politike, čvor koji je primio zahtev za čitanje mora da pribavi određeni broj replika datog podatka kako bi utvrdio vrednost koju treba da vrati.

Bez obzira na podešenu politiku za konzistentnost podataka, prilikom čitanja podatka sa čvora, uvek se izvršavaju koraci prikazani na sledećoj slici:



Slika 5: Čitanje iz *Cassandra* čvora

1. Najpre se podatak traži u kešu vrsta, ako je keširanje dozvoljeno. Ukoliko je prisutan, taj podatak se vraća i pretraga se završava.
2. Ukoliko podatak nije prisutan u kešu vrsta, pretražuje se keš ključeva. Ukoliko je ključ prisutan, čvor može zaključiti gde se nalazi podatak pomoću određenih struktura u glavnoj memoriji.
3. Čvor proverava da li se podatak nalazi u *Memtable* strukturi.
4. Ukoliko se podatak ne nalazi u *Memtable* strukturi, čvor učitava sa diska *SSTable* strukturu i vrši pretragu podataka u njoj. Obično se kombinuju podaci iz *Memtable* i *SSTable* struktura kako bi se utvrdio koji podatak je validan.
    4.1. Ukoliko je uključeno keširanje podataka, čvor upisuje u keš vrsta pribavljeni podatak iz *SSTable* strukture.

Apache Cassandra i konzistentnost operacija čitanja i upisa

Konzistentnost podataka je jedna od glavnih osobina koja se posmatra kod distribuiranih baza podataka. Konziostentnost se odnosi na to da li su sve kopije nekog podatka sinhronizovane, i imaju istu validnu vrednost. Već smo napomenuli da *Apache Cassandra* ispunjava AP svojistva iz CAP teoreme o distribuiranim bazama podataka. Ne možemo da kažemo da Cassandra ispunjava i osobinu konzistentnosti podataka, ali ima implementirane mehanizme koji omogućavaju DBMS-u određenu fleksibilnost u odnosu na konzistentnost, pa je moguće podešavati željenu konzistentnost na račun nekih drugih osobina.

Cassandra implementira model eventualne konzistentnosti (engl. „*eventually consistent*“). Termin se najčešće odnosi na distribuirane računarske sisteme i označava da ukoliko ne dođe do nove promene nekog podatka, vrednost datog podatka će se nakon nekog vremena sinhronizovati sa ostalim kopijama u distribuiranom sistemu (kada sve replike nekog podatka sinhronizuju svoje vrednosti, kaže se da je replika konvergirala). Iako je ovo prihvatljivo rešenje za problem konzistentnosti podataka, ovakav sistem dosta usložnjava arhitekturu i funkcionisanje distribuiranih sistema, a da nam nikada ne pruža sigurnost da su pročitani validni podaci (da se konvergencija desila pre čitanja podatka), pa je često kritikovan.

Model eventualne konzistentnosti je kod *Apache Cassandra* DBMS-a implementiran tako da sistem može podešavati nivo željene konzistentnosti na račun dostupnosti podataka. Zbog toga se često navodi da u odnosu na CAP teoremu, Cassandra ispunjava AP svojistva, ali po potrebi može da ispunjava i CP svojstva. DBMS nam omogućava da definišemo dve vrste konzistentnosti:

- Konzistent na nivou svake operacije: Za svaku naredbu koju izdajemo preko CQL jezika možemo da definišemo željenu konzistentost.
- Globalna konzistentnost: Možemo da definišemo jedan nivo konzistentnosti za podatke u okviru jednog klastera ili u okviru jednog *data* centra.

Ovakva osobina nam omogućava da modelujemo konzistentnost u zavisnosti od konkretnog problema koji rešavamo. Ukoliko radimo sa „osetljivim“ podacima, gde nam je veoma bitno da podaci budu validni, obezbedićemo konzistentnost podataka na račun latencije odgovora: DBMS-u će trebati više vremena da obradi zahtev dok ne utvrdi validnost podataka, ali će na kraju pročitani podaci sigurno biti

validni. Sa druge strane, ukoliko nam nije od presudne važnosti da podaci u istom trenutku budu validni, možemo da osigurno da je latencija pri obradi veoma mala, tako da korisnik ne mora da čeka previše na rezultat obrade.

Cassandra definiše različite nivoe konzistentnosti koji se mogu koristiti prilikom podešavanja klastera ili prilikom slanja upita. U zavisnosti od operacija koje DBMS obavlja, razlikujemo konzistentnost prilikom čitanja podataka, i konzistentnost prilikom upisa podataka. Konzistentnos prilikom čitanja podataka se odnosi na to koliko replika nekog podatka mora da se složi (da imaju istu vrednost), kako bi se podatak smatrao validnim. U zavisnosti na ovu osobinu, *Cassandra* definiše sledeće nivoe konzistentnosti:

- ONE: Dovoljno je da samo jedna replika odgovori.
- TWO: Dovoljno je da dve replike odgovore.
- THREE: Dovoljno je da tri replike odgovore.
- QUORUM: Potrebna je većina replika, i to: $(\frac{RF}{2} + 1)$ replika, gde je *RF* faktor replikacije.
- ALL: Potrebno je da se sve replike slože oko jedne validne vrednosti.
- LOCAL_QUORUM: Potrebna je većina, ali samo među replikama koje se nalaze u *data* centru čvora koji je primio zahtev od strane klijenta (čvora koordinatora).
- EACH_QUORUM: Potrebna je većina u svakom *data* centru.
- LOCAL_ONE: Potrebno je da samo jedna replika odgovori. Ovaj nivo se koristi kada je klaster podeljen na više *data* centara, i osigurava da koordinator ne šalje zahteve za čitanje replikama iz udaljenih *data* centara.

Prilikom operacije čitanja podatka, čvor koordinator šalje zahtev jednom čvoru koji sadrži traženi podatak. Ukoliko je podešen nivo konzistentnosti veći od ONE, čvor koordinator šalje zahtev za podatkom i ostalim replikama, kako bi utvrdio validnost podatka. Ukoliko uslov konzistentnosti nije ispunjen, pokreće se mehanizam koji ažurira replike u okviru klastera („*read repair*“ proces): čvor koordinator će poslati zahtev svakoj replici (osim onoj kojoj je inicijalno poslao zahtev za podatak), izvršiće poređenje i u svaku repliku će da upiše podatak koji je najskorije dodat u sistem.

Konzistentnost operacije upisa podatka se razlikuje od operacije čitanja. *Cassandra* uvek pri operaciji upisa vrši upis u sve replike koje postoje u sistemu. Konzistentnost se odnosi samo na to koliko replika treba da vrati potvrdu da je upis uspešno obavljen. Pored prethodno definisanih nivoa konzistentnosti, za operaciju upisa postoji još jedan dodatni nivo:

- ANY: Ovom opcijom se dozvoljava čvoru koordinatoru da upiše podatak u svoj *Hint* bafer ukoliko je čvor kome treba da prosledi zahtev nedostupan. Ovim je osigurano da ili jedna replika vrati potvrdu o upisu podatka, ili koordinator upisuje podataka u *Hint* bafer (koji će kasnije da prosledi odgovarajućem čvoru kada postane dostupan).

Zaključak

Apache Cassandra je distribuirana, „*column-store*“ NoSQL baza podataka otvorenog koda, razvijena najpre u kompanije *Facebook*, a danas je razvija i održava *Apache* fondacija. DBMS je particionisan, distribuiran, skalabilan, i ima mogućnost za rad na klasteru računara koji se prostire kroz veći broj *data* centara. Svaki *Cassandra* čvor je projektovan tako da može da se izvršava na bilo kom hardveru, tako da nema specijalnih zahteva prilikom pokretanja sistema. DBMS je pogodan za korišćenje ukoliko nam je potrebna skalabilnost i visoka dostupnost podataka, koje ne utiču na performanse sistema.

Cassandra važi za jednu od najpopularnijih NoSQL rešenja, i pokazuje odlične performanse prilikom rada na velikim klasterima. U ovom radu smo detaljno proučili organizaciju i strukturu *Apacha Cassandra* skladišta podataka, način na koji se čitaju i upisuju podaci, opisali smo komunikaciju između čvorova i kako možemo da podešavamo željenu konzistentnost podataka.

Reference

1. „*Apache Cassandra Documentation v4.0*“[Na mreži] http://cassandra.apache.org/doc/latest/

2. “*Apache Cassandra™ Architecture -Inside DataStax Distribution of Apache Cassandra™*” (Maj, 2019)

3. “*Apache Cassandra Docs*” [Na mreži] https://docs.datastax.com/en/landing_page/doc/landing_page/cassandra.html

4. A. Stanimirović – “Napredne baze podataka – Column stores” (2015)