UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

MongoDB kao distribuirana baza podataka

Seminarski rad
Studijski program: Računarstvo i informatika
Modul: Softversko inženjerstvo

Student:

Đorđe Petković, br. ind. 1614

Mentor:

prof. Aleksandar Stanimirović

Niš, Jun 2024. godine

`

Sadržaj

`

Uvod

Distribuirane baze podataka imaju značajan uticaj u savremenom svetu informacionih tehnologija. Suštinski, distribuirana baza podataka sastoji se od više fizički odvojenih baza koje se nalaze na različitim računarima ili čvorovima u mreži, a povezane su i sinhronizovane radi efikasnog deljenja podataka među različitim lokacijama.

Postoje ključni razlozi za upotrebu distribuiranih baza podataka:

Skalabilnost	Distribuirane baze podataka pružaju mogućnost horizontalnog i vertikalnog skaliranja. Horizontalno skaliranje podrazumeva dodavanje novih čvorova za povećanje kapaciteta skladištenja i obrade podataka, dok vertikalno skaliranje podrazumeva povećanje resursa na postojećim čvorovima. Ova fleksibilnost omogućava efikasno upravljanje rastućim obimom podataka.
Visoka dostupnost	Distribuirane baze podataka omogućavaju replikaciju podataka na više lokacija. To znači da su podaci i dalje dostupni čak i ako dođe do kvara na jednom čvoru ili serveru. Ovo je posebno važno za kritične sisteme gde je neprekidan rad od suštinskog značaja.
Brza obrada podataka	Distribuirane baze omogućavaju paralelno izvršavanje upita i transakcija na različitim čvorovima, što rezultira bržom obradom u poređenju sa centralizovanim bazama podataka gde je sve ograničeno na jedan server.
Otpornost na kvarove	Repliciranje podataka na više lokacija u distribuiranim bazama podataka pruža visok nivo otpornosti na kvarove. Čak i ako dođe do problema sa jednim čvorom ili lokacijom, podaci ostaju dostupni preko drugih replika.
Geografska distribucija	Distribuirane baze omogućavaju geografsku raspodelu podataka na više lokacija, što je korisno za organizacije sa različitim kancelarijama ili filijalama. Ovakva distribucija smanjuje latenciju i poboljšava performanse pristupa podacima.

`

MongoDB pruža napredne mehanizme za distribuirane baze podataka kao što su replikacija i horizontalno skaliranje, što omogućava visoku dostupnost, skalabilnost i performanse u distribuiranim okruženjima. Takođe podržava izvršavanje ACID transakcija u distribuiranim okruženjima, što će biti detaljno opisano u seminarskom radu.

`

Distribuirana arhitektura kod MongoDB

MongoDB je distribuirana baza podataka koja podržava ključne mehanizme poput replikacije, horizontalnog skaliranja, distribuiranih upita i transakcija. Korisnici vide MongoDB kao jedinstvenu celinu, dok su navedeni mehanizmi sakriveni od njih.

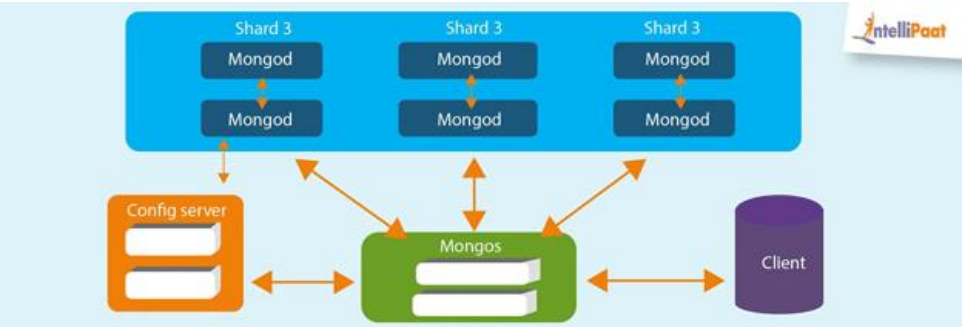
Replikacija u MongoDB-u omogućava podatke da postoje na više čvorova radi postizanja visoke dostupnosti i otpornosti na kvarove. Tipična replikacija uključuje primarni čvor koji prima sve upite i upisuje promene, dok se podaci takođe održavaju na sekundarnim čvorovima. Automatska sinhronizacija podataka obezbeđuje visoku dostupnost i u slučaju gubitka primarnog čvora, jedan od sekundarnih preuzima njegovu ulogu.

Sharding u MongoDB-u omogućava horizontalno skaliranje podataka distribuiranjem podataka preko više shardova, od kojih svaki može biti samostalna MongoDB baza ili skup čvorova. Sharding se postiže podešavanjem shard ključa koji određuje na osnovu kojih kriterijuma se podaci raspoređuju na shardove. Ovo omogućava ravnomerno raspoređivanje opterećenja i paralelno izvršavanje upita na različitim shardovima, pružajući elastičnost i bržu obradu podataka.

MongoDB kombinuje replikaciju i sharding kako bi postigao visoku dostupnost, skalabilnost i otpornost na kvarove u distribuiranim okruženjima, poznatim kao "MongoDB Replication & Sharding Architecture". Takođe podržava distribuirane transakcije koje se mogu izvršavati u sistemima sa replikama i shardovima na velikom broju dokumenata.

Pored distribucije podataka, MongoDB nudi napredne mogućnosti poput automatskog ispitivanja, balansiranja opterećenja, automatskog otkrivanja i oporavka od kvarova, kao i opcije za replikaciju i šifriranje podataka.

U ovom poglavlju biće objašnjeni glavni koncepti distribuirane arhitekture MongoDB baze podataka, njihova uloga u postizanju distribuirane arhitekture, kao i prednosti svakog od koncepata.



`

Replikacija kod MongoDB baze podataka

Skup replika u MongoDB-u je grupa **mongod** procesa koji održavaju isti skup podataka. Skupovi replika pružaju redundantnost i visoku dostupnost, i osnova su za sve korisničke (produkcijske) implementacije, pogotovo u distribuiranim arhitekturama.

U kontekstu distribuirane arhitekture kod MongoDB, replikacija pruža visoku dostupnost, kao i mogućnost oporavka od otkaza nekog od servera, na kojoj god geolokaciji se on nalazio. Omogućava da neki od sekundarnih servera preuzme ulogu primarnog, i održi distribuiranu arhitekturu baza podataka.

Redundantnost i Dostupnost Podataka

Replikacija pruža redundantnost i povećava dostupnost podataka. Sa više kopija podataka na različitim serverskim bazama podataka, replikacija pruža nivo tolerancije na otkaz u slučaju gubitka jednog serverskog baziranog sistema.

U nekim slučajevima, replikacija može pružiti povećani kapacitet čitanja, pošto klijenti mogu slati čitanje operacija na različite servere. Održavanje kopija podataka u različitim centrima može povećati lokalnost i dostupnost podataka za distribuirane aplikacije. Takođe mogu se održavati dodatne kopije za posebne svrhe, kao što su oporavak od katastrofe, izveštavanje ili rezervna kopija.

Skup replika sadrži nekoliko članova koji sadrže podatke i opciono jedan član arbitera (sudije). Od članova koji sadrže podatke, jedan član se smatra primarnim članom, dok se ostali članovi smatraju sekundarnim članovima.

Članovi seta replika

Članovi seta replika mogu biti:

- Primarni
- Sekundarni
- Arbitreri

Ukoliko se implementira replikacija, minimalna konfiguracija za skup replika je skup sa tri člana koji sadrže podatke, odnosno jedan primarni i dva sekundarna člana. U nekim slučajevima, ukoliko dodavanje drugog sekundarnog člana nije moguće, može se dodatiž arbiter. Arbiter učestvuje u izborima (glasanju), ali ne čuva podatke (tj. ne pruža redundanciju podataka). Ograničenje MongoDB-a je 50 člana, ali maksimalno 7 mogu da učestvuju u glasanju.

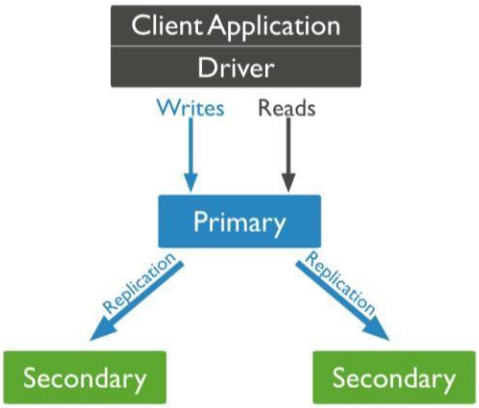
`

Set replika može da sadrži jedan primarni član. Ukoliko dođe do otkaza tog člana, glasa se za postavljanje novog primarnog člana, o čemu će biti reči u nastavku rada.

Primarni član skupa replika

Primarni član je jedini član u skupu replika koji prima operacije upisa (write operacije) [4]. MongoDB primenjuje operacije upisa nad primarnim članom, a zatim beleži operacije u **oplog** (operacione logove) primarnog člana. Sekundarni članovi repliciraju ovaj **oplog** i primenjuju operacije nad svojim skupovima podataka.

U skupu replika sa tri člana, primarni član prihvata sve operacije upisa. Zatim se oplog replikuje na sekundarne članove kako bi se primenio na njihove skupove podataka.



Svi članovi skupa replika mogu prihvatiti operacije čitanja. Međutim, po default-u, aplikacija usmerava svoje operacije čitanja na primarni član (read preference mode = **primary**) [4]. To ponašanje se može promeniti ukoliko se odabere neka od drugih prefernci:

primary	Podrazumevani mod. Sve read operacije se šalju primarnom članu. Distribuirane transakcije koje sadrže read operacije moraju da koriste ovaj mod. Sve operacije u transakcijama moraju da se usmere na isti član.
primaryPreferred	Ukoliko je primarni član dostupan, read operacija će se izvršiti nad njim. Ukoliko nije, izvršiće se nad sekundarnim.
secondary	Sve read operacije primenjuju se nad sekundarnim članovima (replikama).

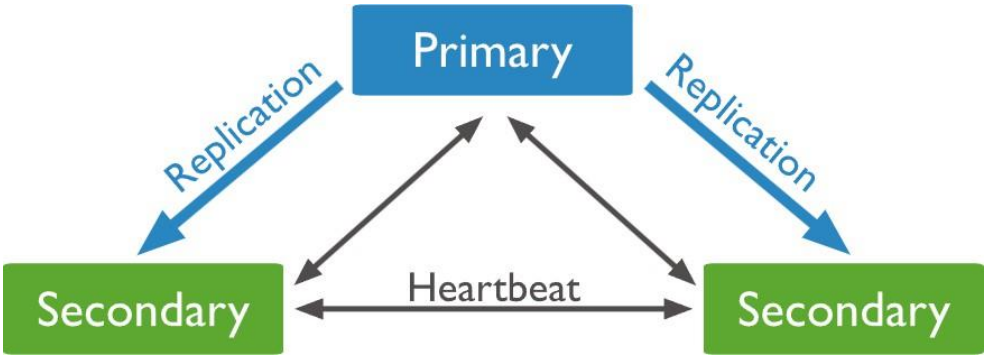
`

secondaryPreffered	Ukoliko je sekundarni član dostupan, read operacija će se izvršiti nad njim. Ukoliko nije, izvršiće se nad primarnim. Ukoliko postoji samo primarni član, bez sekundarnih, izvršiće se nad njim.
nearest	Operacija read se izvršava nad slučajnim članom, nevezano da li je primarni ili sekundarni, već se on bira na osnovnu praga latencije, koji se računa na osnovu prosleđenih opcija (vreme latencije za odabir instance – localThresholdMS, maksimalni replikacioni zaostatak – maxStalnessSeconds ..)

Sekundarni članovi

Sekundarni član održava kopiju primarnog skupa podataka. Za replikaciju podataka, sekundarni član primenjuje operacije iz **oploga** primarnog člana na svoj skup podataka u asinhronom procesu (MongoDB koristi asinhroni tip replikacije) [5]. Skup replika može imati jedan ili više sekundarnih članova.

Na ovom primeru, sa 2 sekundarna i jednim primarnim članom, vidimo da sekundarni članovi repliciraju podatke sa primarnog. Heartbeat (ping) se izvršava na svake 2 sekunde, i on govori članovima seta da li je neki član i dalje aktivan. Ukoliko se ne odazove na 5 heartbeat-a, članovi smatraju taj član kao nedostupan.



Iako se podaci ne mogu pisati direktno na sekundarne članove, mogu se čitati podatci iz sekundarnih članova, u zavisnosti od preferenci objašnjenim u prethodnom poglavlju.

Sekundarni član može postati primarni. Ako trenutni primarni član postane nedostupan, skup replika održava izbore (elections) kako bi odabrao koji od sekundarnih članova postaje novi primarni.

U zavisnosti od konfiguracije, možemo imati tri tipa sekundarnih članova [5]. To su:

- Članovi sa prioritetom 0
- Skriveni članovi
- Odloženi (delayed) članovi

`

Članovi sa prioritetom 0

Član sa prioritetom 0 je član skupa replika koji ne može postati primarni i ne može pokrenuti izbore (election). Članovi sa prioritetom 0 mogu potvrditi operacije upisa izdane sa postavkom **write concern** od **w: <broj>.** Za **"majority"** write concern, član sa prioritetom 0 mora biti glasački član kako bi mogao potvrditi upis (members[n].votes je veći od 0).

Osim gore pomenutih ograničenja, sekundarni članovi sa prioritetom 0 funkcionišu kao normalni sekundarni članovi. Oni održavaju kopiju skupa podataka, prihvataju operacije čitanja i glasaju na izborima.

Konfigurisanje člana skupa replika sa prioritetom 0 može biti poželjno ako je taj član implementiran u data centru koji je udaljen od glavne implementacije i stoga ima veću latenciju. Takav član može dobro služiti za lokalne zahteve za čitanje, ali nije idealan kandidat za obavljanje dužnosti primarnog člana zbog svoje latencije. Ukoliko bi takav član bio izabran za primarni čvor, rad samog sistema bi bio dosta sporiji.

Skriveni članovi

Skriveni članovi održavaju kopiju skupa podataka primarnog čvora, ali su nevidljivi za klijentske aplikacije. Klijenti neće distribuirati operacije čitanja sa preferencijom čitanja iz sekundarnih članova ka skrivenim članovima. Kao rezultat toga, ovi članovi ne primaju nikakve zahteve niti operacije, osim osnovne replikacije. Skriveni članovi koriste se za specijalizovane zadatke kao što su izveštavanje i pravljenje rezervnih kopija. Skriveni članovi moraju uvek biti članovi sa prioritetom 0 i stoga ne mogu postati primarni čvor. Skriveni članovi, takođe, mogu glasati na izborima.

Skriveni članovi skupa replika mogu potvrditi operacije upisa izdane sa postavkom **w: <broj>**. Međutim, za operacije upisa izdane sa write concern w: "majority", skriveni članovi moraju takođe glasati kako bi mogli potvrditi upis sa "majority" write concern. [9]

Odloženi (delayed) članovi

Odloženi članovi sadrže kopije skupa podataka skupa replika. Međutim, skup podataka odloženog člana odražava ranije, ili odloženo, stanje skupa. Na primer, ukoliko član ima kašnjenje od 2 sata, na njemu neće biti primenjena nijedna operacija izvršena u proteklih 2 sata.

Zbog toga što su odloženi članovi „rezerva“ ili tekući „istorijski“ snapshot skupa podataka, oni mogu pomoći u oporavku od različitih vrsta grešaka. Na primer, odloženi član može omogućiti oporavak od neuspešnih migracija i grešaka operatera, uključujući obrisane baze podataka i kolekcije.

`

Odloženi članovi takođe mogu da učestvuju u glasanju, i u obavezi su da učestvuju ukoliko je prosleđen write concern „majority“, a njihov members[n].votes je veći od 0. Česta je praksa postavljanja tog property-ja na 0, da bi rad odloženih članova bio efikasniji.

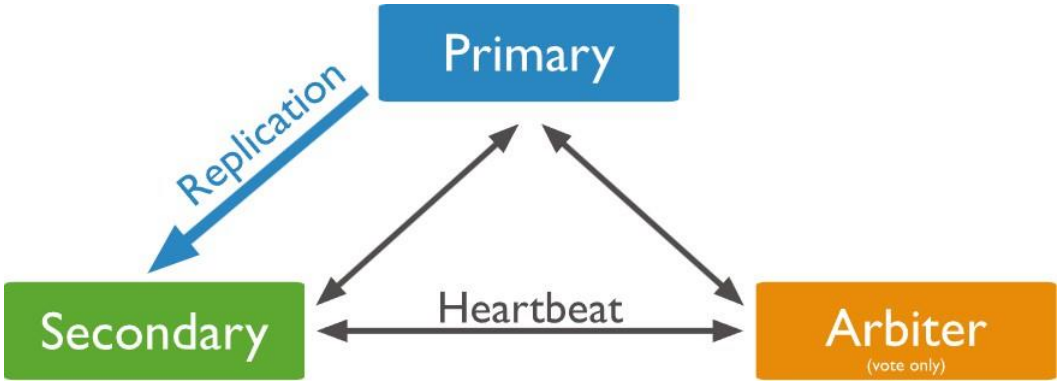
Arbiteri

U nekim slučajevima, ukoliko je moguće postojanje samo jednog primarnog i sekundarnog člana, može se dodati arbiter u skup replika. Arbiter učestvuje u izborima za primarni čvor, ali nema kopiju skupa podataka i ne može postati primarni čvor [6].

Arbiter ima tačno 1 glas u izborima. Po defaultu, arbiter ima prioritet 0.

Uglavnom je preporučeno korišćenje jednog arbitera kako bi se izbegli problemi sa konzistentnošću podataka. Više arbitera sprečava pouzdano korišćenje "majority" write concern.

Da bi se osiguralo da će upis opstati nakon pada primarnog čvora, "majority" write concern zahteva da većina čvorova potvrdi operaciju upisa. Arbitri ne čuvaju nikakve podatke, ali doprinose ukupnom broju čvorova u skupu replika. Kada skup replika ima više arbitera, manja je verovatnoća da će većina čvorova koji nose podatke biti dostupna nakon pada čvora.



Oplog

Log operacija (oplog) je specijalna ograničena kolekcija koja sadrži zapise svih operacija koje menjaju podatke koje se čuvaju u bazi podataka. Ukoliko write operacije ne utiču na podatke u bazi podataka, one se ne čuvaju u oplog-u.

Za razliku od ostalih kolekcija ograničenih veličinom, oplog može da naraste iznad specificiranog limita veličine, da bi se izbeglo brisanje nekih nereplikovanih operacija. MongoDB primenjuje operacije nad primarnim članom, i upisuje ih u oplog primarnog člana.

`

Sekundarni članovi iz oploga čitaju operacije, i primenjuju ih u procesu asinhrone replikacije. Svi sekundarni članovi moraju imati kopiju oplog-a, u **local.oplog.rs** kolekciji, koji im omogućava sinhronizaciju sa primarnim članom [7].

Svi članovi šalju heartbeat-ove (pingove) međusobno. Bilo koji sekundarni član može preuzeti oplog od bilo kog drugog člana.

Svaka operacija u oplog-u je idempotentna. Oplog operacije daju iste rezultate, ma koliko puta se primenjuju na bazu podataka.

Može se specifirati minimalni broj sati tokom kojih će sve operacije u oplog-u biti sačuvane. U tom slučaju, operacije će biti izbrisane iz oplog-a samo ukoliko je oplog popunjen do maksimalne konfigurisane veličine, i ukoliko je ta operacija starija od minimalnog broja sati očuvanja oplog-

Replikacija i sinhronizacija skupa replika

Da bi se održale ažurirane kopije deljenog skupa podataka, sekundarni članovi skupa replika sinhronizuju ili repliciraju podatke od drugih članova. MongoDB koristi dva oblika sinhronizacije podataka: početnu sinhronizaciju za popunjavanje novih članova kompletnim skupom podataka i replikaciju za primenu tekućih promena na ceo skup podataka [8].

Početna sinhronizacija

Početna sinhronizacija kopira sve podatke sa jednog člana skupa replika na drugog člana. Mogu se navesti preferirani izvor početne sinhronizacije pomoću parametra **initialSyncSourceReadPreference**. Ovaj parametar se može navesti samo prilikom pokretanja **mongod**.

Mehanizmi oporavka uz pomoć replikacije

MongoDB koristi nekoliko mehanizama oporavka i replikacije kako bi osigurao visoku dostupnost i otpornost na greške. Ovi mehanizmi obezbeđuju da baze podatka, i pri otkazu nekod od čvorova, nastavi da radi i pruža odgovore na zahteve i operacije.

Automatki failover

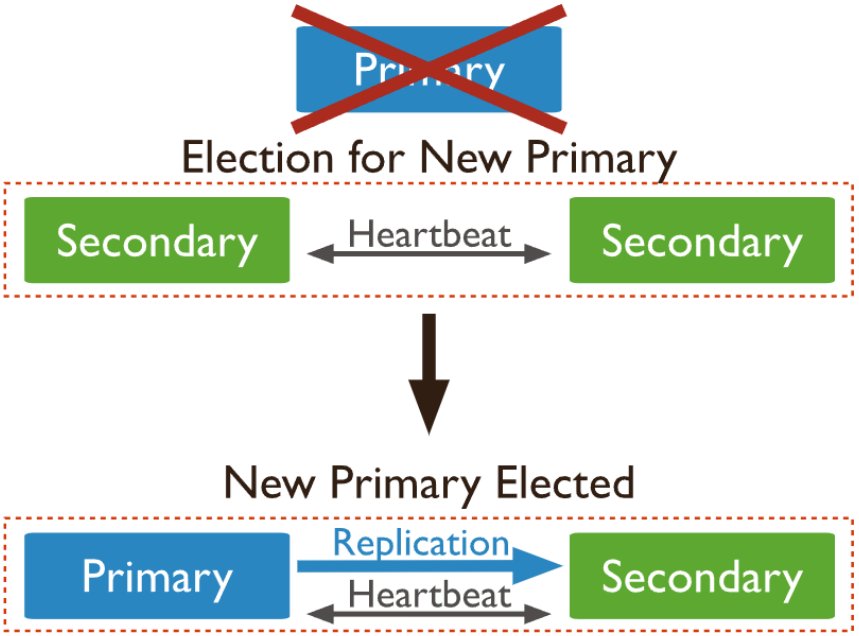
Replika setovi koriste izbore (elections) kako bi odredili koji član skupa će postati primarni. U ovim izborima učestvuju svi članovi koji imaju pravo glasa (members[n].votes > 0). Replika skupovi pokreću izbore kao odgovor na razne događaje, kao što su:

- Dodavanje novog čvora u skup replika
- inicijalizacija replika skupa

`

- izvođenje održavanja (mainentance) skupa replika korišćenjem metoda kao što su rs.stepDown() ili rs.reconfig()
- sekundarni članovi koji gube konekciju sa primarnim članom na više od konfigurisanog vremenskog ograničenja (podrazumevano 10 sekundi – odnosno 5 hearthbeat-ova)

Ukoliko je primarni čvor je bio nedostupan duže od konfigurisanog vremenskog ograničenja, pokreće se proces automatskog failover. Jedan od preostalih sekundarnih članova poziva na izbore kako bi se izabrao novi primarni član i automatski nastavilo normalno funkcionisanje.



Sve do završetka izbora, set replika ne može da procesira write operacije. Može da izvršava read operacije, ukoliko su one konfigurisane da se čitaju sa sekundarnih čvorova (secondary ili secondaryPreferred postavljeno na read preference mode).

`

Horizontalno skaliranje kod MongoDB

Horizontalno skaliranje podrazumeva podelu celokupnog skupa podataka i opterećenja na više servera, dodavanjem dodatnih servera kako bi se povećao kapacitet prema potrebi. Iako pojedinačna brzina ili kapacitet svakog servera možda neće biti visoki, svaki server se bavi podskupom ukupnog radnog opterećenja, potencijalno pružajući bolju efikasnost u poređenju sa jednim serverom visokih performansi i kapaciteta. Proširenje kapaciteta se jednostavno postiže dodavanjem novih servera po potrebi, što može rezultirati nižim ukupnim troškovima u poređenju sa nabavkom vrhunskog hardvera za pojedinačnu mašinu. Ipak, ovaj pristup nosi povećanu složenost infrastrukture i zahteva održavanje sistema.

MongoDB podržava horizontalno skaliranje podataka kroz mehanizam poznat kao sharding. Sharding u MongoDB-u skalira podatke na nivou kolekcije.

Jedan od glavnih ciljeva sharding-a je da klaster od nekoliko stotina čvorova (shardova) izgleda kao jedna mašina iz perspektive aplikacije. Da bi sakrili detalje o skaliranju od aplikacije, MongoDB koristi jedan ili više ruter procesa koji se nazivaju mongos instance. Mongos ruter prima sve upite i, uz pomoć konfiguracionih servera, određuje koje shardove treba uključiti u obradu upita.

Prilikom horizontalnog skaliranja, MongoDB deli podatke na čunke (chunks) na osnovu ključa za deljenje. Svaki čunak predstavlja podskup dokumenata čija polja koja predstavljaju ključ imaju vrednosti u određenom opsegu.

Prednosti korišćenja šardinga:

Efikasnije operacije upisa i čitanja	Deljenje podataka na šardove u klasteru omogućava da svaki šard bude zadužen za podskup operacija na klasteru, tačnije za one operacije koje se tiču podataka dodeljenim tom šardu. Na taj način omogućeno je horizontalno skaliranje operacija čitanja i upisa.
Kapacitet skladišta	S obzirom na to da se podaci dele po šardovima, sa porastom podataka moguće je dodati nove šardove u sistem i na taj način povećati njegov kapacitet skladištenja.
Visoka dostupnost	Ukoliko neki šard u sistemu bude nedostupan, upisivanje podataka može da se vrši na dostupnim šardovima. Takođe je omogućeno i čitanje podataka na dostupnim šardovima.

Shardovani klaster (sharded cluster)

Shardovani klaster predstavlja celinu čvorova na kojima su raspoređeni podaci iz kolekcija baze. Sharder cluster se sastoji iz sledećih komponenti:

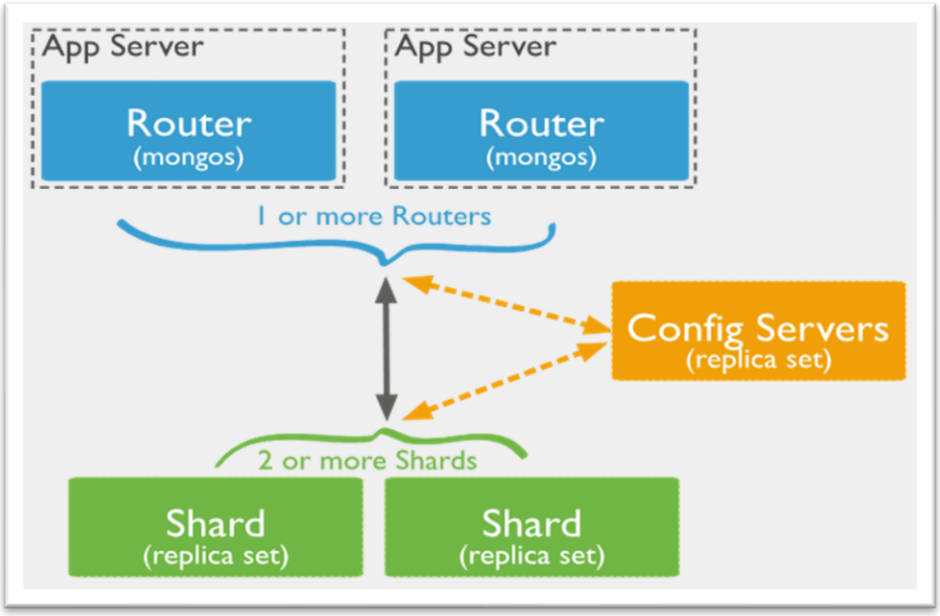
shard-Jedan shard predstavlja mongoDB instancu ili replica set koji sadrži deo ukupnih podataka na sharder cluster-u,
mongos – Mongos je ruter koji usmerava upite i pruža interfejs između klijentskih aplikacija i čvorova u cluster-u. Sakriva od korisnika detalje skaliranja, tako da korisnik bazu vidi kao jednu nepodeljenu celinu,

`

konfiguracioni serveri – Ovi serveri skladište meta podatke i konfiguraciona podešavanja za cluster. Na osnovu podataka iz konfiguracionih servera mongos ruter zna kojim šardovima da prosledi izvršavanje operacije.

Na sledećoj slici ilustrovane su veze između komponenti šardovanog klastera.

Baza podataka može sadržati i kolekcije koje su distribuirane po šardovima, ali i one koje nisu. Neparticionisane kolekcije se uvek skladište na **primarnom šardu**. Svaka baza podataka poseduje svoj primarni šard.



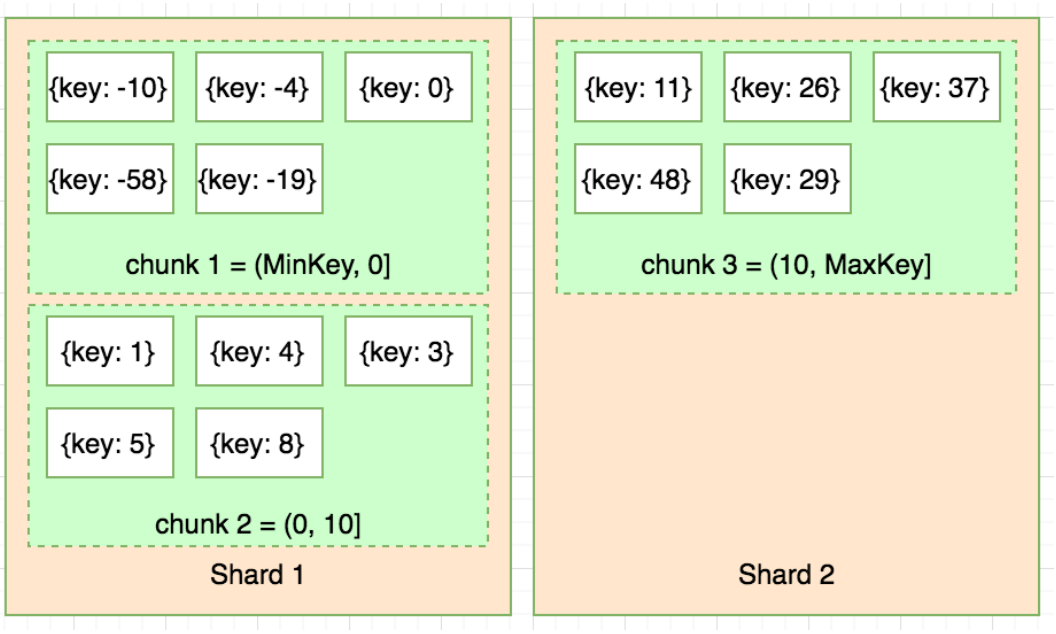
Particija podataka u chunk-ove

MongoDB koristi ključ za particionisanje kako bi podelio podatke između šardova. Podaci se dele u delove koji se nazivaju čankovi. Svaki čank obuhvata određeni opseg particionisanih podataka, definisan na osnovu vrednosti ključa za particionisanje. Svaki čank ima svoju donju i gornju granicu u odnosu na vrednosti ključa.

Podrazumevana veličina jednog čanka u MongoDB-u je 128 megabajta, ali ova veličina se može promeniti putem konfiguracije sistema.

Proces koji upravlja preraspodelom podataka u čankove i migracijom podataka kod MongoDB-a naziva se balanser. Kada se razlika između najvećeg i najmanjeg šarda u količini podataka dostigne određeni prag, balanser vrši migraciju podataka kako bi se postigla ravnomernija distribucija opterećenja između šardova.

`



Ključevi za particiju podataka

MongoDB koristi ključeve za particionisanje za utvrđivanje načina na koji se podaci dele između shard-ova. Ključ za particionisanje (shard key) se sastoji iz jednog ili više polja dokumenta.

Ključ za particionisanje se selektuje prilikom poziva operacije particionisanja kolekcije (*sh.shardCollection*).

Prilikom particionisanja kolekcije, ukoliko se radi o kolekciji koja nije prazna, mora da postoji indeks u kolekciji čiji prefiks predstavlja ključ za particionisanje. Ukoliko je kolekcija prazna, pozivom operacije particionisanja će se kreirati indeks kolekcije koji sadrži ista polja kao i ključ za particionisanje.

Izbor ključa za particiju

Izbor ključa za particionisanje ima značajan uticaj na performanse, efikasnost i skalabilnost distribuiranog klastera. Čak i klaster sa najboljom mogućom infrastrukturom može postati usko grlo zbog lošeg izbora ključa za particionisanje.

Prilikom odabira ključa za particionisanje, važno je uzeti u obzir sledeće faktore:

Kardinalnost ključa	Kardinalnost ključa određuje maksimalan broj čankova koji mogu biti kreirani. Dokumenti sa istom vrednošću ključa moraju biti smešteni u isti čank. Ključ sa niskom kardinalnošću može dovesti do manjeg broja, ali većih čankova, što smanjuje efikasnost horizontalnog skaliranja. Zbog toga je poželjno odabrati ključ sa visokom kardinalnošću za bolju distribuciju opterećenja.
Frekvencija vrednosti ključa	Ključ sa visokom kardinalnošću može generisati veći broj čankova, ali to ne garantuje ravnomernu raspodelu podataka. Treba obratiti pažnju na frekvenciju pojavljivanja različitih vrednosti ključa. Ako su neke vrednosti mnogo češće od drugih, čankovi koji sadrže ove vrednosti mogu postati preopterećeni, što može

`

	predstavljati usko grlo sistema.
Monotono rastuće vrednosti ključa	Ključ za particionisanje sa vrednostima koje monotono rastu ili opadaju može dovesti do toga da svi novi dokumenti završe u istom čanku, što može stvoriti usko grlo. Ovo je zbog načina na koji MongoDB deli podatke u čankove sa definisanim gornjim i donjim granicama na osnovu vrednosti ključa.
Šabloni upita	Idealno, ključ za particionisanje treba da omogući ravnomernu distribuciju podataka i olakšava izvršavanje čestih upita. Prilikom izbora ključa, važno je uzeti u obzir uobičajene obrasce pretrage podataka i da li ih odabrani ključ podržava.
Ograničenja ključa	U MongoDB-u, ključ za particionisanje ne sme premašiti 512 bajta u ranijim verzijama, ali ovo ograničenje je uklonjeno od verzije 4.4. Takođe, postoji zahtev za indeksom koji pokriva ključ za particionisanje i uređen je po rastućem poretku, ne sme biti multikey, tekstualni ili geoprostorni indeks.

Sve ove faktore treba pažljivo razmotriti prilikom planiranja ključa za particionisanje u MongoDB-u radi optimalne distribucije i performansi klastera.

Strategije za sharding

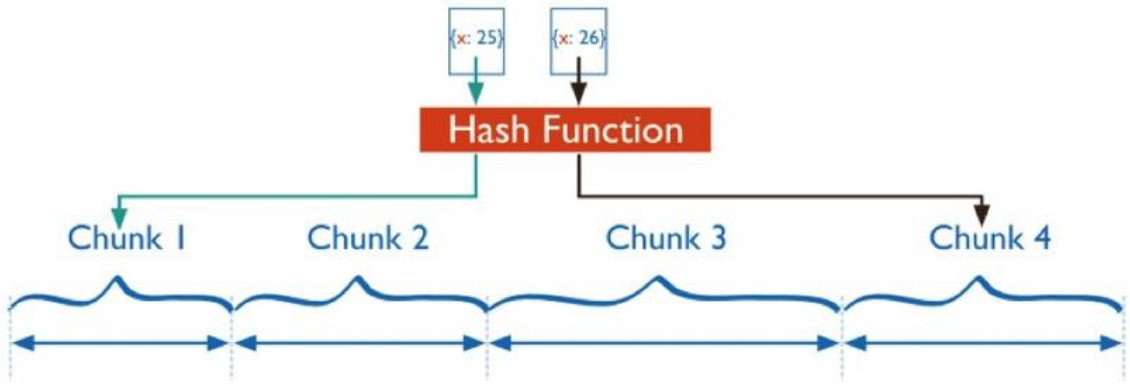
MongoDB podržava dve osnovne strategije za particionisanje podataka:

Heširano particionisanje

Ova strategija koristi heširanu vrednost ključa za particionisanje. Umesto da se opsezi čankova određuju direktno na osnovu plaintext vrednosti ključa, pripadnost dokumenta čanku određuje se primenom heš funkcije na vrednost ključa tog dokumenta. Heš funkcije imaju prednost u tome što bliske vrednosti gotovo nikada neće rezultirati sličnim heš vrednostima, što doprinosi ravnomernijem rasporedu podataka po čankovima. Ova karakteristika je posebno korisna kada su vrednosti ključa monotono rastuće, što omogućava da se novi dokumenti ravnomernije rasporede umesto da se uvek smeštaju u isti čank.

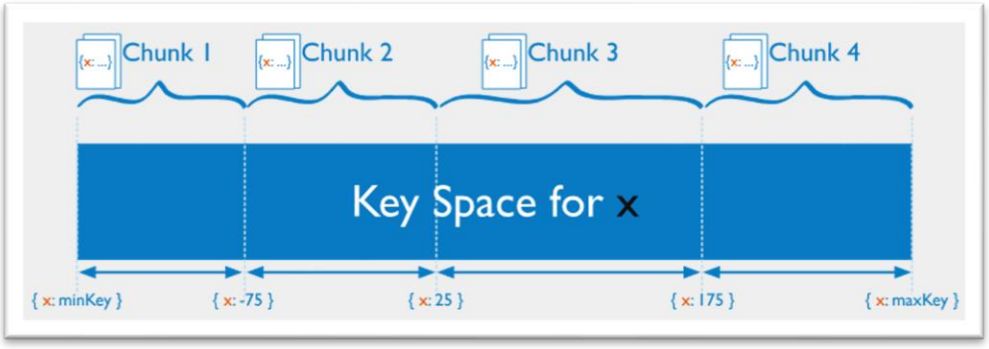
Međutim, heširano particionisanje može imati slabije performanse kod upita koji zahtevaju operacije nad opsezima vrednosti ključa. Zbog toga što su susedne vrednosti ključa često raspoređene na različite šardove, izvršenje ovakvih upita može biti sporije.

`



Particionisanje po opsezima

Za razliku od heširanog particionisanja, ovde se određuje pripadnost čanku direktno na osnovu vrednosti ključa za particionisanje. Svaki čank dobija opseg vrednosti ključa, pa će dokumenti sa sličnim vrednostima ključa verovatno biti smešteni u isti čank.

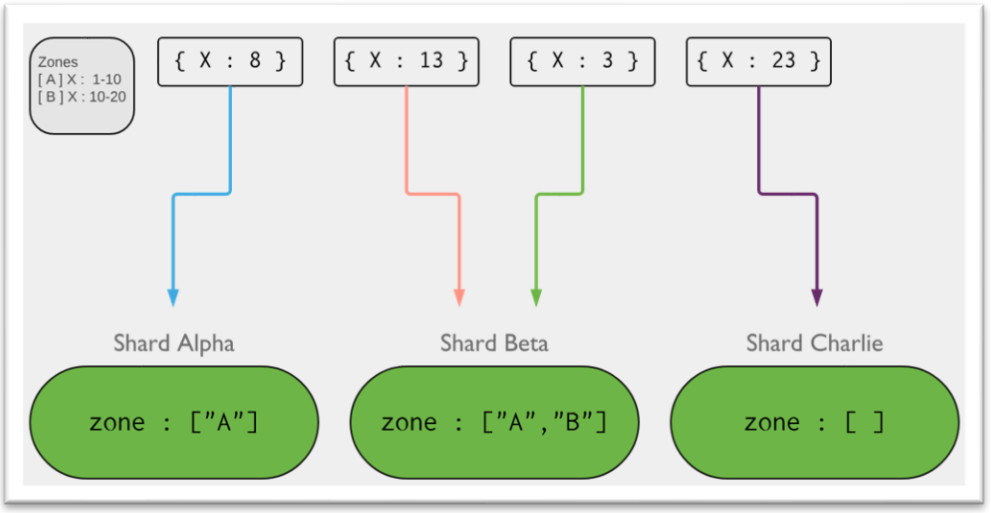


Sharding zone

U šardovanim klasterima moguće je kreirati zone šardovanih podataka na osnovu ključa za particionisanje. Svaka zona može biti povezana sa jednim ili više šardova u klasteru, pri čemu jedan šard može obuhvatiti više zona. Balanser obavlja migraciju podataka sa jednog šarda koji pripadaju određenoj zoni isključivo u šardove kojima je dodeljena ta zona.

U ilustraciji šardovanog klastera koji se sastoji od 3 šarda (Alpha, Beta i Charlie) i 2 zone (zona A i zona B), zona A obuhvata dokumente čiji se ključevi kreću u opsegu od 1 do 10, dok zona B obuhvata dokumente čiji se ključevi kreću u opsegu od 10 do 20. Šardovima Alpha i Beta je dodeljena zona A, dok je šardu Beta dodeljena i zona B, dok šard Charlie nema dodeljenu zonu. To znači da će dokumenti sa ključevima u opsegu 1-10 biti smešteni na šardovima Alpha i Beta, dok će dokumenti sa ključevima u opsegu 10-20 biti smešteni samo na šardu Beta. Za sve ostale dokumente nema specifičnih ograničenja i oni se mogu naći na bilo kom šardu.

`



Neki uobičajeni slučajevi upotrebe zona uključuju:

- Izolovanje specifičnog skupa podataka na određeni podskup šardova
- Raspoređivanje važnih podataka na šardove koji su geografski najbliži aplikacionom serveru
- Raspoređivanje podataka po šardovima uzimajući u obzir njihove hardverske kapacitete.

Distribuirane transakcije

Transakcija predstavlja niz operacija nad bazom podataka koje će biti uspešne samo ako svaka operacija unutar transakcije bude ispravno izvršena. Relacione baze podataka su tradicionalno podržavale transakcije kao ključnu karakteristiku, dok su baze podataka orijentisane na dokumente, poput MongoDB-a, često izbegavale ovu funkcionalnost zbog prirode dokumenata koji mogu sadržati složene strukture i ugrađene pod-dokumente [10].

Iako većina aplikacija sa bazama podataka orijentisanim na dokumente ne zahteva transakcije zbog jednostavnijeg skladištenja povezanih podataka unutar jednog dokumenta, postoje slučajevi gde je neophodno garantovati integritet pri operacijama koje modifikuju više dokumenata u jednoj transakciji. MongoDB je u verziji 4.0 uveo ACID transakcije koje podržavaju manipulaciju sa više dokumenata unutar replikacionih setova (multi-document transactions). Verzija 4.2 dodatno je proširila podršku na distribuirane transakcije u šardovanim klasterima.

Atomičnost distribuiranih transakcija

Distribuirane transakcije u MongoDB-u su atomične, što znači da obezbeđuju "sve ili ništa" mehanizam izvršavanja. Kada se transakcija izvrši, sve operacije unutar nje su trajno sačuvane i vidljive celom sistemu. Sve dok se sve operacije u transakciji ne završe uspešno, nijedna od njih neće imati vidljiv uticaj na sistem. U slučaju greške ili otkaza tokom izvršenja transakcije, sve operacije koje su prethodno izvršene biće poništene.

Dozvoljene operacije u distribuiranim transakcijama

Operacije koje su dozvoljene unutar MongoDB-ovih distribuiranih transakcija obuhvataju CRUD operacije nad postojećim kolekcijama, kao i operacije kao što su agregacije, countDocuments, distinct (samo nad ne-šardovanim kolekcijama), find, kao i operacije kreiranja, ažuriranja i brisanja kako pojedinačnih tako i grupa dokumenata. Od verzije 4.4 MongoDB-a, podržane su i operacije

`

kreiranja indeksa i novih kolekcija unutar transakcija. Informacione operacije poput hello, buildInfo i connectionStatus takođe su podržane, ali ne mogu biti prva operacija u transakciji.

Ograničenja distribuiranih transakcija

Postoje neke operacije koje nisu podržane u distribuiranim transakcijama MongoDB-a, kao što su listCollections, listIndexes, administrativne operacije poput kreiranja korisnika ili uloga, kao i operacija kreiranja kolekcije u šardovanim klasterima ako transakcija uključuje upis u drugu kolekciju. Takođe, nije dozvoljeno čitanje ili pisanje u lokalnim (local), konfiguracionim (config) i administratorskim (admin) bazama podataka, niti izvršavanje explain upita koji vraća plan izvršenja nekog upita.

Ove specifikacije i ograničenja omogućavaju MongoDB-u da efikasno podrži distribuirane transakcije uz očuvanje integriteta podataka i sigurnosti operacija u kompleksnim okruženjima.

Konzistentnost operacija čitanja i pisanja kod MongoDB-a

Izolacija operacija čitanja (Read Concern)

U MongoDB-ju se kontrola konzistentnosti i izolacije podataka postiže podešavanjem parametra readConcern. Pored njega, značajan je i writeConcern parametar (o čemu ćemo govoriti kasnije), koji omogućava fino podešavanje nivoa konzistentnosti i dostupnosti prema potrebama. Prilikom podešavanja ovih parametara, potrebno je napraviti kompromis: stroži zahtevi za konzistentnošću često usporavaju operacije i smanjuju dostupnost sistema, dok se povećanje dostupnosti može postići smanjivanjem kriterijuma konzistentnosti podataka.

Nivoi konzistentnosti koje možemo obezbediti podešavanjem readConcern parametra su sledeći:

Lokalni nivo (“local”)

Upit vraća podatke sa konkretne Mongo instance bez garancije da su ti podaci prethodno upisani na većinskom broju instanci u skupu replika. To znači da u slučaju otkaza većine čvorova ili neuspele transakcije, ovi podaci mogu biti povučeni.

> **Primer**: U setu od tri replike (2 sekundarne i 1 primarna), ako je readConcern podešen na “local” i izvršena je operacija upisivanja W0, pri čemu su svi prethodni upisi uspešno izvršeni. Sledeći je timeline:
>
> - T0: Primarna izvršava operaciju W0.
> - T1: Prva sekundarna replika čita oplog i izvršava operaciju W0.
> - T2: Druga sekundarna replika čita oplog i izvršava operaciju W0.
> - T3: Primarna postaje svesna da je prva sekundarna replika uspešno izvršila operaciju W0 i šalje potvrdu klijentu.
> - T4: Primarna postaje svesna da je druga sekundarna replika uspešno izvršila operaciju W0.
> - T5: Prva sekundarna replika dobija obaveštenje da je operacija W0 uspešno izvršena u većini čvorova i to pamti u svom najnovijem snapshot-u.
> - T6: Druga sekundarna replika dobija obaveštenje da je operacija W0 uspešno izvršena u većini čvorova i to pamti u svom najnovijem snapshot-u.
>
> U sledećoj tabeli prikazana su stanja podataka u određenim vremenskim intervalima za prethodno opisane događaje ukoliko je readConcern podešen na “local”:

`

Čvor	Vreme	Podaci koji se vraćaju pri čitanju
Primarna	Nakon T0	Podaci sadrže efekte upisa W0
Prva sekundarna	Pre T1	Podaci ne sadrže efekte upisa W0, već Wprev
Prva sekundarna	Nakon T1	Podaci sadrže efekte upisa W0
Druga sekundarna	Pre T2	Podaci ne sadrže efekte upisa W0, već Wprev
Druga sekundarna	Nakon T2	Podaci sadrže efekte upisa W0

Kao što se vidi iz tabele, svaki član replike odmah po izvršenju operacije W0 lokalno smatra da je operacija validna, bez obzira da li će korisnik zatražiti podatke pre nego što je čvor dobio potvrdu od ostalih čvorova o uspešnosti operacije ili nakon toga, biće mu vraćeni najnoviji podaci.

Dostupni nivo (“available”)

Sličan lokalnom nivou, ali podaci se vraćaju sa instance bez traženja potvrde od ostalih instanci. Razlika između ove i opcije local je u tome što kod distribuiranih (sharded) kolekcija available nivo može vratiti i takozvane “orphaned” dokumente, koji su ostali na šardu nakon neuspele migracije. Kod local nivoa nema rizika od vraćanja orphaned dokumenata.

Nivo većine (“majority”)

Upit vraća podatke koje je potvrdila većina članova u setu. Takvi podaci su trajni, čak i u slučaju otkaza.

Primer: Redosled događaja je isti kao u primeru za lokalni nivo, ali sada replike moraju čekati potvrdu da je većina čvorova uspela u izvršavanju operacije pre nego što podaci postanu vidljivi u operacijama čitanja. Pregled stanja podataka u replikama prikazan je u sledećoj tabeli:

Čvor	Vreme	Podaci koji se vraćaju pri čitanju
Primarna	Pre T3	Podaci ne sadrže efekte upisa W0, već Wprev
Primarna	Nakon T3	Podaci sadrže efekte upisa W0
Prva sekundarna	Pre T5	Podaci ne sadrže efekte upisa W0, već Wprev
Prva sekundarna	Nakon T5	Podaci sadrže efekte upisa W0
Druga sekundarna	Pre T6	Podaci ne sadrže efekte upisa W0, već Wprev
Druga sekundarna	Nakon T6	Podaci sadrže efekte upisa W0

Linearizable

Upit vraća podatke koji odražavaju sve uspešne većinski potvrđene operacije upisa koje su završene pre početka operacije čitanja. Ima efekte slične prethodno opisanom nivou, uz to što

`

omogućava većem broju niti da izvršavaju upise i čitanja nad istim podacima u realnom vremenu i omogućava linearizaciju konkurentnih operacija.

Snapshot

Upit vraća podatke koji su većinski potvrđeni u sistemu u određenom trenutku u vremenu. Možemo podesiti i parametar atClusterTime kako bismo videli stanje podataka u određenom trenutku.

Potvrđivanje operacije upisa (Write Concern)

Način potvrđivanja operacije upisa u MongoDB-u za setove replika i distribuirane šardove određuje se parametrom “writeConcern”. Ovaj parametar se može postaviti na nivou pojedinačne operacije upisa, osim kod transakcija gde se postavlja na nivou svih operacija transakcije.

Parametar writeConcern sadrži nekoliko polja:

Polje w	Ova opcija opisuje nivo potvrde potrebne od sistema da bi operacija bila smatrana uspešnom. Može imati vrednost “majority”, što znači da je potrebna potvrda većine članova sistema, ili numeričku vrednost koja predstavlja minimalan broj instanci od kojih je potrebna potvrda.
Polje j	Predstavlja boolean vrednost koja određuje da li se zahteva da operacija bude upisana u žurnalu na disku.
Wtimeout	Specifikuje vremenski limit za dobijanje potvrde o uspešnosti operacije upisa. Ako potvrda ne stigne od broja čvorova specificiranih u w polju pre isteka wtimeout perioda, operacija će se smatrati neuspešnom i klijent će dobiti grešku. Ukoliko ovaj parametar nije postavljen, a nema uslova za potvrđivanje operacije od strane dovoljnog broja instanci, operacija upisa će biti trajno blokirana.

Kauzalna konzistentnost kod MongoDB-a

Kada jedna operacija logički zavisi od prethodne operacije, kažemo da između njih postoji kauzalna zavisnost. Na primer, operacija koja briše sve dokumente iz kolekcije i operacija čitanja koja proverava da li je kolekcija prazna nakon brisanja imaju takvu vrstu zavisnosti. MongoDB podržava kauzalno konzistentne sesije koje obezbeđuju da se operacije izvršavaju redosledom koji poštuje njihovu kauzalnu zavisnost.

Da bi klijentska sesija bila kauzalno konzistentna, potrebno je da operacije čitanja i upisa u toj sesiji imaju parametre readConcern i writeConcern (koji su objašnjeni u prethodnim odeljcima) postavljene na vrednost “majority”. Takođe, potrebno je da klijentska aplikacija izvršava samo jednu nit u jednom trenutku.

Kauzalno konzistentna sesija pruža sledeće garancije:

`

Čitanje sopstvenih upisa (Read your writes)	Garantuje da će svaka operacija čitanja prikazivati efekte poslednje operacije upisa izvršene od strane istog klijenta.
Monotona čitanja (Monotonic reads)	Garantuje da operacija čitanja može vratiti samo novije ili iste podatke u odnosu na prethodnu operaciju čitanja. Na primer, ako u sesiji operacija upisa w1 prethodi operaciji upisa w2, operacija čitanja r1 prethodi operaciji čitanja r2 i operacija čitanja r1 vraća podatke koji uključuju efekte operacije w2, tada operacija čitanja r2 ne može vratiti podatke koji sadrže samo efekte operacije w1.
Monotoni upisi (Monotonic writes)	Garantuje da će se operacija upisa jedne sesije izvršiti pre svih sledećih operacija upisa u istoj sesiji. Na primer, ako imamo operaciju upisa w1 koja se izvršava pre operacije upisa w2, operacija w2 mora biti izvršena nad podacima koji uključuju efekte operacije w1.
Čitanje nakon upisa (Writes follow reads)	Garantuje da će operacija upisa u jednoj sesiji, kojoj prethodi operacija čitanja u istoj sesiji, biti izvršena nad podatkom koji je isti ili noviji u odnosu na onaj koji je pročitan u operaciji čitanja.

Ove garancije važe za sve čvorove u sistemu. Na primer, ako na primaru replike izvršimo operaciju upisa koja zahteva potvrdu od većine članova sistema, a zatim izvršimo operaciju čitanja sa sekundarnog člana replike, operacija čitanja će prikazati efekte prethodne operacije upisa.

`

Praktična primena nad skupom podataka

Setovi replika

U ovom poglavlju biće predstavljen i implementiran set replika, kao i operacije nad njim.

Konfiguracija replika seta

Na početku potrebno je pokrenuti **mongod** instance

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data1\db1" --port 27017 --replSet replike
```

Ukoliko pravimo set replika sa 3 instance, potrebno je ovaj postupak ponoviti jos dva puta, sa različitim vrednostima za port number. Možemo pokrenuti na portovima 27018 i 27019.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data1\db1" --port 27018 --replSet replike
```

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data1\db1" --port 27019 --replSet replike
```

Nakon pokretanja, potrebno je konfigurisati set replika. Prvo se konektujemo na instancu koja želimo da bude primarna (27017 u ovom slučaju), i nakon toga proslediti konfiguracioni objekat komandi **rs.initiate()**

```
const rsconf = {
	_id: 'replike',
	members: [
		{_id: 0, host: 'localhost:27017' },
		{_id: 0, host: 'localhost:27018' },
		{_id: 0, host: 'localhost:27019' }
	]
}
rs.initiate(rsconf)
```

Operacije upisa nad setom replike se uvek izvršavaju na primaru seta replike, a kasnije putem oploga ostale replike ažuriraju svoje skupove podatka. Međutim, da bi osigurali uspešnost operacije upisa nad celim sistemom, možemo tražiti da određen broj replika da potvrdu o tekućoj operaciji upisa kako bi ona uopšte mogla da se izvrši. To se čini podešavanjem **writeConcern** parametra.

Ukoliko imamo tri člana seta replike, dva sekundarna i 1 primarni 1 sekundarni čvor otkaže. Ako se postavi writeConcern na “majority” (što znači da većina čvorova mora da potrvrdi upis), kako većina čvorova čini 2 čvora a u sistemu i jesu ostala 2 aktivna čvora, operacija će se izvršiti. Međutim, ukoliko je writeConcern postavljen na vrednost 3, što znači da je potrebna potvrda svih članova, operacije će biti bezuspešna.

`

```
> db.products.insertOne(
    { item: "bags", qty : 100, type: "Clasp" },
    { writeConcern: { w: "majority" , wtimeout: 5000 } }
)
< {
    acknowledged: true,
    insertedId: ObjectId("64997f74222e5e13dd45a374")
  }
```

```
> db.products.insertOne(
    { item: "books", qty : 100, type: "Clasp" },
    { writeConcern: { w: 3,wtimeout:3000 } }
)
MongoWriteConcernError: waiting for replication timed out
```

MongoDB Sharding

Kreiranje klastera i šardovanje kolekcija

Jedan šard klaster u MongoDB-u sastoji se od sledećih komponenti: šardova, koji su čvorovi na kojima su distribuirani podaci iz kolekcija, konfiguracionih servera, koji čuvaju informacije o tome koji se podaci nalaze na kom šardu, i mongos rutera, čiji je zadatak da prima upite i usmerava ih ka odgovarajućem šardu.

U nastavku će biti dat primer kreiranja jednog šard klustera i šardovanja kolekcije na lokalnoj mašini. Za ivršenje komandi potrebno je pozicionirati se u delu fajl sistema na kome se nalazi MongoDB.

Pr. `C:\Program Files\MongoDB\Server\6.0\bin>`

Da bi kreirali konfiguracioni server treba kreirati mongo instancu na sledeći način:

```
mongod-configsvr--replSet ConfigReplSet-bind_ip localhost
```

Potrebno je pokrenuti instancu mongod sa postavljenim --configsvr flegom. Takođe, preporučljivo je da konfiguracioni server bude pokrenut kao set replike kako bi se poboljšala konzistentnost sistema. Zbog toga je ovde postavljen --replSet fleg, nakon čega sledi naziv seta replike. Što se tiče IP adrese servera, ovde se radi o lokalnoj mašini, a broj porta nije naveden u naredbi jer MongoDB automatski dodeljuje broj porta 27019 za konfiguracioni server.

Pošto je konfiguracioni server pokrenut kao replica set, potrebno je izvršiti konfiguraciju replike pozivom metode rs.initiate(). Pre poziva ove metode, potrebno je konektovati se na pokrenutu serversku instancu na portu 27019.

```
mongosh localhost:27019
```

`

```
rs.initiate({
"id": "ConfigReplSet",
"configser": true,
members ": [
        ("id": 0, "host": "localhost:27019")
]
})
```

Nakon inicijalizacije konfiguracionog servera, možemo kreirati i konfigurisati instancu šard servera. Preporučljivo je da šard bude set replika kako bi se povećala bezbednost i dostupnost podataka. Na sledećoj slici prikazana je komanda za pokretanje šard servera. Ključno je postavljanje --shardsvr flega. Port nije potrebno navesti, jer MongoDB po difoltu dodeljuje port broj 27018 za šard server.

```
mongod-shardsvr --replSet ShardReplSet --bind_ip localhost
```

Nakon pokretanja šard servera kao replika seta, potrebno je izvršiti inicijalizaciju tog seta replike, slično kao kod konfiguracionog servera, samo menjamo port na 27018.

Ovim postupkom možemo kreirati više šard servera.

Kreiranje i konfiguracija mongos rutera

Nakon kreiranja šard instanci i konfiguracionog servera, preostaje još samo kreiranje rutera operacija u šard klasteru. Ruter predstavlja posebnu MongoDB instancu koja se kreira mongos naredbom. Na sledećoj slici prikazano je kreiranje mongos instance.

```
mongos --configdb ConfigReplSet/localhost:27019 --bind_ip localhost
```

Važno je pri kreiranju mongos rutera navesti adresu konfiguracionog servera u --configdb parametru. Podrazumevani broj porta mongo rutera je 27017. Sva komunikacija sa klasterom ide preko rutera i sve operacije se prvo upućuju njemu.

Dodavanje šardova u klaster se radi korišćenjem sh.addShard operacije, koja uzima adresu šarda koji se dodaje.

```
sh.addShard ("ShardReplSet/localhost:27018")
```

Omogućavanje šardinga nad bazom podataka vrši se operacijom sh.enableSharding, kojom eksplicitno govorimo da je dozvoljeno vršenje šardinga nad bazom podataka, koju mu prosleđujemo kao parametar.

```
sh.enableSharding("ShardReplSet ")
```

Šardovanje kolekcije

Šardovanje, odnosno distribucija podataka po šardovima, se izvršava na nivou kolekcija u MongoDB-u. Da bismo izvršili šardovanje kolekcije, potrebno je da definišemo koje polje dokumenata će se koristiti kao šarding ključ. Na slici ispod prikazano je izvršenje operacije **sh.shardCollection** kojom se vrši šardovanje kolekcije. Kao parametri ove naredbe navode se kolekcija nad kojom se vrši šarding i polje koje će biti izabrano kao ključ po čijim se vrednostima

`

vrši distribucija podataka po čvorovima. Ukoliko navedena kolekcija ne postoji, MongoDB će je kreirati. Takođe će kreirati i indeks nad poljem navedenim kao ključ.

```
sh.shardCollection("baza.kolekcija, {“pretraga” : 1})
```

Kada smo odredili kolekciju koja će biti šardovana i kreirali indeks nad poljem koje će se koristiti kao ključ šardinga, možemo pozvati komandu **sh.shardCollection** kojom se vrši šardovanje kolekcije. Ovoj komandi kao parametre treba proslediti naziv kolekcije i ključ za šardovanje.

Nakon dodavanja podataka u kolekciju, korišćenjem **insertMany** operacije, možemo da izvršimo operaciju **db.baza.getShardDistribution()** koja nam vraća seledeće rezultate.

```
Shard ShardReplSet1 at ShardReplSet1/localhost:27028
{
        data: 781B',
        docs: 13,
        chunks: 2,
        'estimated data per chunk': '548B',
        'estimated docs per chunk': 7
}
Shard ShardReplSet at ShardReplSet/localhost:27018
{
        data: '1KiB',
        docs: 17,
        chunks: 2,
        'estimated data per chunk': '636B',
        'estimated docs per chunk': 8
}
```

Vidimo da je od 30 dokumenata, koliko ih je ukupno, 13 raspoređeno u prvom šardu, dok je 17 raspoređeno u drugom šardu. Što je raspodela ravnomernija, to znači da je odabrani ključ bolji. U idealnom slučaju, dokumenti će biti raspoređeni podjednako između svih šardova.

Distribuirane transakcije

MongoDB omogućava izvršavanje atomičnih transakcija u šardovanim klasterima i setovima replika. Svaka transakcija se odvija unutar jedne klijentske sesije i kreira se iz nje. Sve operacije unutar transakcije se izvršavaju po principu "sve ili ništa": ili će sve operacije biti uspešno izvršene, ili neće nijedna.

U nastavku je prikazana C# konzolna aplikacija koja koristi MongoDB.Driver, u kojoj je demonstrirana transakcija koja dodaje nekoliko dokumenata u kolekciju i zatim ažurira te iste dokumente. Operacije koje čine transakciju navode se između poziva funkcija startTransaction i endTransaction. Na slici ispod prikazana je transakcija koja dodaje nove dokumente (proizvode) u kolekciju products i zatim ih ažurira u istoj transakciji.

```
try
{
        await products. InsertOneAsync(session, tv);
        await products.InsertOneAsync(session, book);
```

`

```
        await products. InsertOneAsync(session, dogBowl);
        var resultsBeforeUpdates await products
                .Find<Product>(session, Builders<Product>.Filter.Empty)
                .ToListAsync();
        foreach (Product d in resultsBefore Updates)
        {
                Console.WriteLine(
                String.Format("Product Name: {0}\tPrice: (1:0.00}", d. Description, d. Price)
        );
        }
        // Increase all the prices by 10% for all products

        var update = new Update DefinitionBuilder<Product>()
                .Mul<Double>(r => r. Price, 1.1);
        await products. UpdateManyAsync(session, Builders<Product>
                .Filter.Empty, update); //,options);
        // Ukoliko ne postoji greska, komitujemo transakciju
        session.CommitTransactionAsync();
}
```

`

Zaključak

MongoDB kao distribuirana baza podataka nudi veliki skup funkcionalnosti koje omogućavaju skaliranje i visoku dostupnost podataka. Kroz replikaciju, MongoDB obezbeđuje otpornost na greške i poboljšava dostupnost sistema tako što koristi više kopija podataka koje se nalaze na različitim čvorovima. Replikacioni setovi omogućavaju automatsko prebacivanje u slučaju pada glavnog čvora, čime se smanjuje vreme otkaza.

Šardovanje je ključna tehnika koja omogućava horizontalno skaliranje velikih baza podataka, tako što deli podatke po različitim šardovima. Korišćenjem šardovanja, MongoDB može efikasno upravljati velikim količinama podataka i distribuirati teret upita, što rezultira poboljšanom performansom i bržim vremenima odgovora.

Distribuirane transakcije pružaju mogućnost izvršavanja više operacija nad različitim dokumentima i kolekcijama kao jednu atomski operaciju. Ove transakcije osiguravaju konzistentnost podataka u kompleksnim scenarijima, omogućavajući "sve ili ništa" pristup, što je od suštinskog značaja za kritične poslovne aplikacije.

Kombinovanjem ovih tehnika, MongoDB omogućava izgradnju skalabilnih, pouzdanih i visokodostupnih aplikacija koje mogu da podrže velike količine podataka i visok nivo konkurentnih zahteva. Kroz fleksibilnost i snažne mogućnosti koje pruža, MongoDB se pokazuje kao moćan alat za moderno upravljanje podacima u distribuiranim okruženjima.

`

Reference

[1] https://www.mongodb.com/company/what-is-mongodb

[2] https://rivery.io/data-learning-center/complete-guide-to-data-replication/

[3] How To Use Transactions in MongoDB | DigitalOcean

[4] https://www.mongodb.com/docs/manual/core/replica-set-primary

[5] https://www.mongodb.com/docs/manual/core/replica-set-secondary

[6] https://www.mongodb.com/docs/manual/core/replica-set-arbiter

[7] https://www.mongodb.com/docs/manual/core/replica-set-oplog/

[8] Causal Consistency in Mongo. Written by Kert Pjatkin | by Outfunnel Tech Blog | Medium

[9] The “Majority” WriteConcern of MongoDB Replica Sets | by OnlyKiosk Dev Tech | Level Up Coding (gitconnected.com)

[10] https://www.digitalocean.com/community/tutorials/how-to-use-transactions-in-mongodb