Seminarski rad

Predmet: Sistemi za upravljanje bazama podataka

Tema: MongoDB kao distribuirana baza podataka

Student: Jelena Đikić, br. ind. 1488

Niš, januar 2024

Sadržaj

1. Uvod

U poslednjih nekoliko godina, eksponencijalni rast količine generisanih podataka postao je veliki izazov za organizacije širom sveta. Od 2020. godine, za samo tri godine zetabajti podataka su se udvostručili, dostižući 55,8 zetabajta. Ovaj masivan porast podataka postavlja organizacijama potrebu za inovativnim rešenjima u skladištenju i upravljanju podacima visokih performansi.

U traganju za efikasnim rešenjima, distribuirane baze podataka se izdvajaju kao ključni faktor u savremenom svetu informacija. Njihova sposobnost da se skaliraju i prilagode rastućim potrebama organizacija čini ih neophodnim alatom u eri velikih podataka (engl. Big Data). Distribuirane baze podataka su ključne jer omogućavaju skladištenje raznovrsnih podataka, uključujući struktuirane i nestruktuirane podatke, očuvavajući istovremeno integritet podataka.

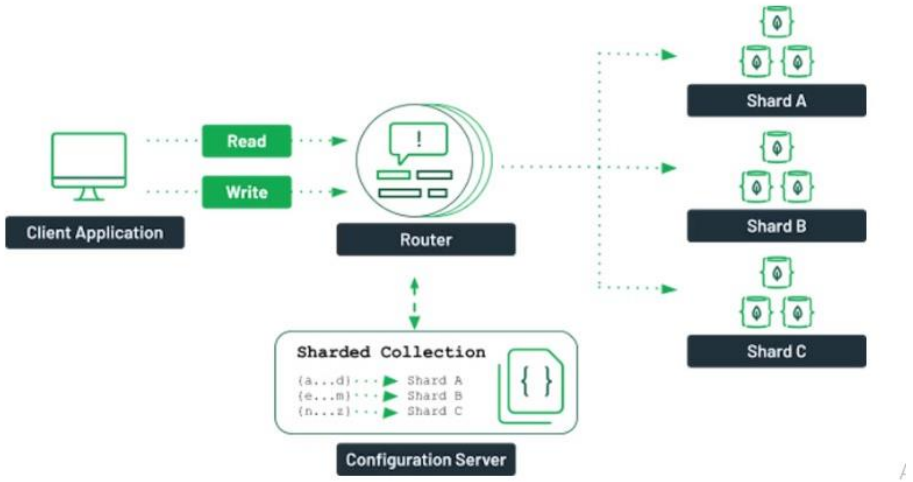
Jedan od lidera u ovom domenu je MongoDB, distribuirana baza podataka koja je stekla značajnu popularnost. Osnovne karakteristike MongoDB-a uključuju fleksibilnu semantiku JSON dokumenta, visok stepen skalabilnosti i podršku za horizontalno skaliranje. Ova baza podataka omogućava organizacijama da efikasno upravljaju ogromnim količinama podataka uz istovremeno održavanje performansi sistema.

Međutim, i pored brojnih prednosti, distribuirane baze podataka, uključujući MongoDB, nisu bez izazova. Povećana složenost upravljanja, potreba za adekvatnim obukama zaposlenih i izazovi u održavanju sinhronizacije između različitih čvorova sistema predstavljaju neke od mana koje organizacije moraju uzeti u obzir.

Distribuirane baze podataka postaju temelj za organizacije koje teže efikasnom upravljanju velikim količinama podataka. MongoDB, kao lider u ovom polju, pruža rešenja koja odražavaju potrebe savremenih organizacija, ali istovremeno zahtevaju pažljivo upravljanje kako bi se iskoristile sve prednosti ove tehnologije.

2. Distribuirane baze podataka

Distribuirana baza podataka predstavlja sistem za čuvanje podataka na više različitih lokacija umesto na jednom mestu. Drugim rečima, umesto centralizovanog skladištenja podataka na jednom serveru ili računaru, informacije su distribuirane na više servera ili u klaster računara sastavljenog od pojedinačnih čvorova. Ovi čvorovi često nisu samo geografski odvojeni, već se mogu manifestovati kao fizički računari ili virtuelne mašine smeštene unutar baze podataka u oblaku. Osim toga, distribuirane baze podataka često koriste različite tehnologije, kao što su protokoli za sinhronizaciju i replikaciju podataka, kako bi održale koherentnost između različitih delova sistema. [1]



Slika 1. Ilustracija klastera i čvorova [1]

Primer MongoDB klastera, koji je prikazan na slici 1 iznad, predstavlja samo jednu od mnogobrojnih opcija konfiguracije dostupnih pri izgradnji distribuirane baze podataka. Za razliku od klasičnih centralizovanih baza podataka, distribuirane baze podataka imaju zajedničku karakteristiku širenja podataka na više lokacija, bilo da su u pitanju fizički ili virtuelni resursi. Ova praksa znatno poboljšava otpornost i dostupnost podataka. Takođe, deljenje podataka na različitim mestima omogućava horizontalno skaliranje sistema. [1]

2.1. Osnovni koncepti distribuiranih baza podataka

Kao što je ranije pomenuto, čvorovi su pojedinačni serveri ili računari koji se nalaze u sistemu distribuirane baze podataka (npr. računari, virtuelne mašine, serveri koji ne dele fizičke komponente). Svaki čvor čuva skup podataka i radi na softveru sistema za upravljanje

distribuiranom bazom podataka (DDBMS). Da bi se odredilo koji će podaci biti uskladišteni među kojim čvorovima, mora se uzeti u obzir koncept distribucije podataka.

2.1.1. Podela podataka

Efikasnost, sigurnost i optimalan pristup korisnika u distribuiranoj bazi podataka zavise od precizne distribucije podataka. Ovaj proces, poznat i kao particionisanje podataka, može se postići primenom dve različite metode.

- **Horizontalno particionisanje**, kao prva metoda, podrazumeva cepanje tabela podataka u redove i distribuciju tih redova na različite čvorove sistema. Ova tehnika omogućava bolju raspodelu podataka i ravnotežu opterećenja između čvorova, pružajući efikasnost i optimizaciju pristupa podacima.
- **Vertikalno particionisanje** predstavlja drugu ključnu metodu, koja deli tabele na kolone i distribuira ih na različite čvorove sistema. Ova tehnika omogućava efikasno upravljanje specifičnim tipovima podataka i poboljšava performanse sistema kroz selektivnu distribuciju kolona prema potrebama korisnika.

Obe metode doprinose pravilnoj organizaciji podataka, što rezultira unapređenjem performansi, bezbednosti i pristupa podacima u okviru distribuirane baze podataka. [1]

Slika 2. Podela podataka na shard-ove [1]

Nastali skupovi podataka iz horizontalnog ili vertikalnog particionisanja originalne tabele se ponekad nazivaju delovima, šardovima (engl. **shards**).

2.1.2. Komunikacija sistema distribuirane baze podataka

Iako su čvorovi sposobni za potpuno nezavisno funkcionisanje, neophodno je da ostvaruju komunikaciju međusobno, jer, suprotno centralizovanim bazama podataka, ne dele iste fizičke komponente ili čak iste skupove podataka. Postoje tri varijante komunikacije unutar distribuirane baze podataka:

- Komunikacija putem emitovanja (engl. broadcast): Jedna poruka se šalje svim preostalim čvorovima u sistemu distribuirane baze podataka.
- Komunikacija putem grupnog emitovanja (engl. multicast): Jedna poruka se šalje određenim, ali ne svim drugim čvorovima u sistemu distribuirane baze podataka.
- Komunikacija putem ličnog emitovanja (engl. unicast): Poruka se šalje sa jednog pojedinačnog čvora ka drugom pojedinačnom čvoru.

2.1.3. Upravljanje transakcijama

Distribuirane baze podataka često moraju podržavati distribuirane transakcije koje obuhvataju više čvorova. Ova podrška je ključna za očuvanje svojstava **ACID**-a (atomičnost, konzistentnost, izolovanost, trajnost) transakcija u distribuiranim sistemima baza podataka. Osnovne karakteristike ACID-a obuhvataju:

- **Atomičnost**, gde se transakcija tretira kao jedinstvena jedinica, čime se održava integritet podataka. Ako transakcija nije potpuna, odbacuje se kao greška.
- **Konzistentnost**, koja se održava primenom predefinisanih pravila i ograničenja podataka u distribuiranim sistemima baza podataka.
- **Izolacija**, koja podrazumeva odvajanje transakcija radi sprečavanja sukoba podataka i očuvanja integriteta, posebno u višestrukim distribuiranim zapisima podataka na različitim lokacijama.
- **Trajnost**, koja obezbeđuje da se podaci čuvaju i u slučaju kvara sistema.

Ovo su ključne komponente koje transakcioni distribuirani sistemi upravljanja bazom podataka efikasno implementiraju kako bi osigurali pouzdanost i integritet podataka u distribuiranim okruženjima. [1]

2.1.4. Replikacija podataka

Replikacija podataka podrazumeva proces održavanja više kopija podataka na različitim čvorovima, serverima ili sajtovima. Dostupne su različite vrste šema replikacije baze podataka, među kojima su:

- Potpuna replikacija: U ovoj šemi, kompletna, funkcionalna kopija celokupne baze podataka distribuira se na sve lokacije unutar sistema distribuirane baze podataka, sa rutinskim ažuriranjima kopija baze podataka prema rasporedu. Postoje dva podtipa potpune replikacije.
- Transakciona replikacija: Ova šema obezbeđuje punu kopiju baze podataka na svakom čvoru, a promene podataka se ažuriraju u toj kopiji u stvarnom vremenu tokom obrade transakcija.
- Replikacija snimka: Ovde se pravi snimak baze podataka u određenom trenutku i distribuira se po čvorovima po potrebi, ali ne prati dosledno promene podataka. Preporučuje se samo za retko promenjene sadržaje.
- Delimična replikacija: U ovom slučaju, određeni čvorovi zahtevaju samo specifične delove baze podataka, pa se definisani deo replicira u izabranu grupu čvorova.
- Objedinjavanje replikacije: Ova najkompleksnija šema podrazumeva spajanje dve baze podataka u jednu. [1]

3. MongoDB kao distribuirana baza podataka

MongoDB predstavlja moćnu i visoko skalabilnu distribuiranu bazu podataka koja se ističe u savremenom izboru tehnologija zbog svoje sposobnosti efikasnog upravljanja podacima u distribuiranom okruženju. Ovaj sistem za upravljanje bazama podataka (DBMS) implementira ključne principe i karakteristike kako bi omogućio optimalno rukovanje podacima širom različitih čvorova, servera ili lokacija.

Jedan od osnovnih principa MongoDB-a je fleksibilnost u modeliranju podataka. MongoDB koristi JSON-sličan format, nazvan BSON, što omogućava lako skaliranje i brzo čitanje i pisanje podataka. Ova fleksibilnost olakšava agilno prilagođavanje promenama u strukturi podataka, što je od suštinskog značaja u dinamičkim i promenljivim okruženjima.

Distribuirana priroda MongoDB-a proizlazi iz sposobnosti **horizontalnog skaliranja**, gde se podaci raspoređuju na više čvorova radi postizanja veće otpornosti na otkaze i poboljšane

dostupnosti. Ovaj pristup omogućava MongoDB-u da se lako nosi sa rastućim obimom podataka i visokim zahtevima performansi.

MongoDB primenjuje strategije **replikacije** radi osiguravanja visoke dostupnosti i otpornosti na kvarove. Kroz replikaciju, podaci se održavaju u više kopija, što omogućava da se sistem održava u slučaju gubitka čvora ili servera. Takođe, MongoDB podržava horizontalnu particionisanje podataka, gde se podaci dele na manje delove radi ravnomernijeg raspoređivanja opterećenja.

Jedna od ključnih karakteristika MongoDB-a je njegova sposobnost podrške za **kompleksne upite** i analize putem fleksibilnih mehanizama upita. Ovo je od suštinskog značaja za aplikacije koje zahtevaju napredno pretraživanje i analizu podataka.

MongoDB se ističe kao distribuirana baza podataka zahvaljujući svojoj fleksibilnosti, horizontalnom skaliranju, podršci za replikaciju i particionisanje, kao i sposobnosti za obradu kompleksnih upita. Ovaj DBMS predstavlja ključni alat za organizacije koje žele efikasno rukovati velikim obimom podataka u distribuiranom okruženju.

3.1. Replikacija

Skup replika u MongoDB čini grupa mongod procesa koji zajedno održavaju identičan skup podataka. Ovi skupovi replika su od ključnog značaja za različite primene u proizvodnji, pružajući redundantnost i visoku dostupnost podataka. Redundantnost i dostupnost podataka su suštinski aspekti replikacije. Ova tehnika omogućava očuvanje više kopija podataka na različitim serverima baze podataka, što rezultira povećanom tolerancijom na greške u slučaju gubitka jednog od servera. Time se osigurava da podaci ostanu dostupni i funkcionalni, čak i u nepredviđenim situacijama.

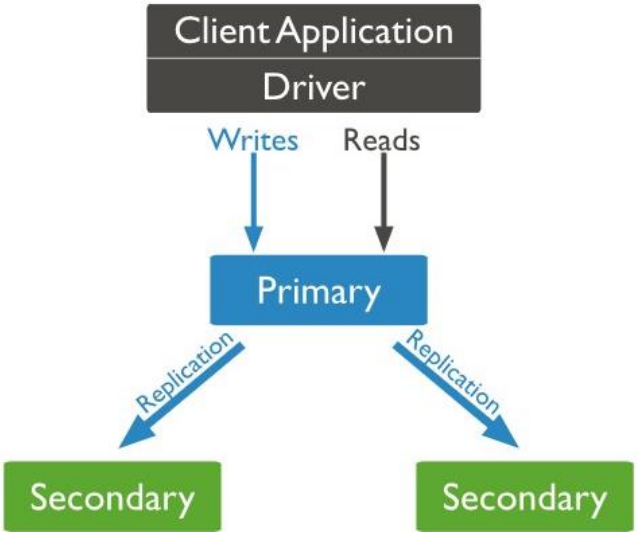
Pored toga, replikacija može doprineti povećanju kapaciteta za čitanje podataka, jer klijenti mogu upućivati operacije čitanja na različite servere. Ova fleksibilnost omogućava optimalno iskorišćenje resursa i poboljšava efikasnost sistema.

U nekim slučajevima, održavanje kopija podataka u različitim centrima podataka može povećati lokalizaciju podataka, poboljšavajući pristup i dostupnost za distribuirane aplikacije. Takođe, postoji mogućnost održavanja dodatnih kopija podataka za specifične namene, kao što su oporavak od katastrofe, izveštavanje ili pravljenje rezervnih kopija. Sve ove funkcionalnosti čine skupove replika ključnim elementom za stabilnost i efikasnost MongoDB sistema u proizvodnom okruženju.

3.1.1. Članovi skupa replika

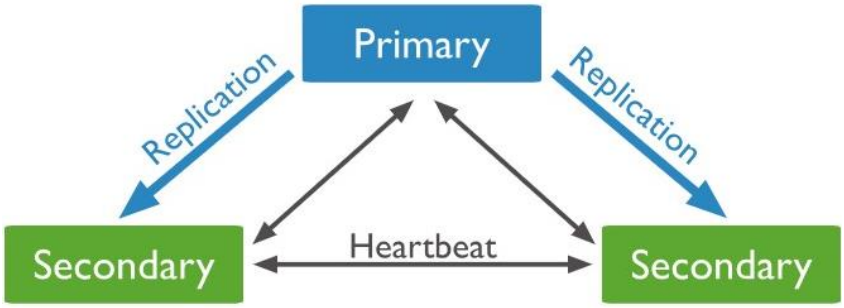
Skup replika predstavlja grupu instanci mongod-a koje održavaju isti skup podataka. Skup replika obuhvata nekoliko čvorova koji sadrže podatke i opciono jedan čvor arbitra. Od čvorova koji sadrže podatke, jedan i samo jedan član je označen kao **primarni** čvor, dok su ostali čvorovi označeni kao **sekundarni** čvorovi, što se može videti na slici 3.

Primarni čvor prima sve operacije pisanja. Skup replika može imati samo jedan primarni čvor sposoban da potvrdi upise sa { *w: "majority"* } sigurnosnim nivoom upisa, iako u nekim okolnostima, druga instanca mongod-a može privremeno smatrati sebe takođe primarnim. Primarni čvor beleži sve promene u svojim skupovima podataka u svom operacionom zapisu, tj. *oplog-u*. [10]



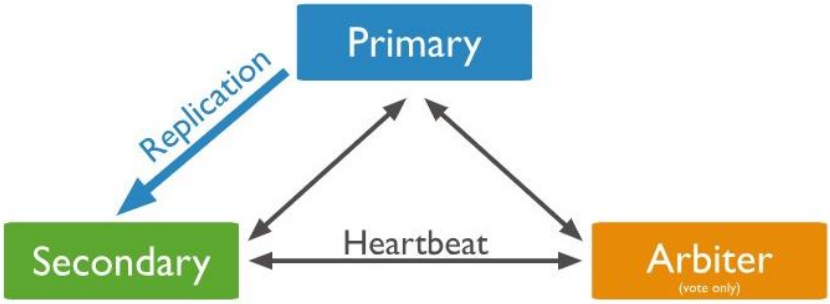
Slika 3. Replikacija kod MongoDB (primar, sekundari) [10]

Sekundarni čvorovi repliciraju *oplog* primarnog čvora i primenjuju operacije na svoje skupove podataka tako da se skupovi podataka sekundarnih čvorova odražavaju na skup podataka primarnog čvora, što se može videti na slici 4. Ukoliko primarni čvor nije dostupan, odabrani sekundarni čvor će organizovati izbore kako bi se sam izabrao za novi primarni čvor.



Slika 4. Replikacija oplog-a primarnog čvora [10]

U nekim situacijama, kada postoje ograničenja troškova (na primer, kada već postoje primarni i sekundarni čvor, ali finansijski faktori sprečavaju dodavanje još jednog sekundarnog čvora), moguće je odlučiti se za dodavanje instance mongod-a u skup replika u ulozi arbitra. Arbitar učestvuje u izborima, ali ne zadržava podatke, odnosno, ne obezbeđuje redundantnost podataka, kao što je prikazano na slici 5.



Slika 5. Dodavanje arbitera umesto sekundara [10]

Arbiter će uvek ostati arbiter, dok primarni čvor može odstupiti i postati sekundarni, a sekundarni može postati primarni tokom izbora. [10]

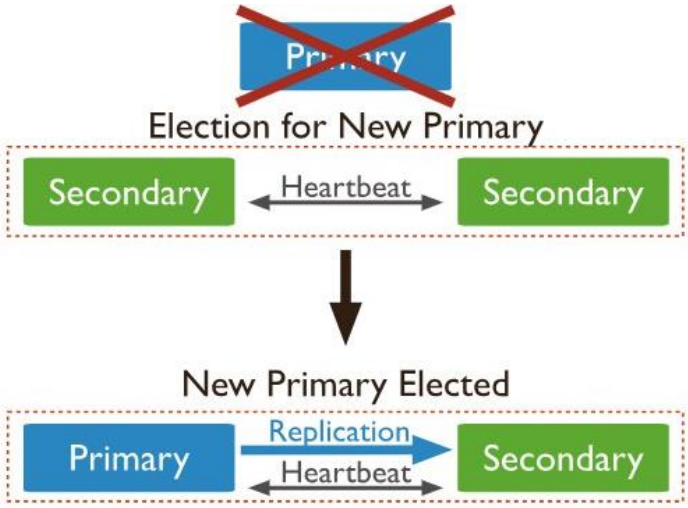
Sekundarni čvorovi repliciraju oplog primarnog čvora i primenjuju operacije na svoje skupove podataka asinhrono. Omogućavanjem da se skupovi podataka sekundarnih čvorova odražavaju na skup podataka primarnog čvora, skup replika može nastaviti s radom uprkos kvaru jednog ili više članova. Ova asinhrona replikacija doprinosi povećanoj otpornosti sistema na greške, jer omogućava rad skupa replika čak i kada neki članovi nisu trenutno dostupni ili su u kvaru. Takođe, pruža dodatni sloj bezbednosti i stabilnosti, čineći MongoDB bazu podataka pouzdanom i otpornom na potencijalne prekide ili izazove u radu sistema.

Još jedan bitan pojam jeste **replikaciono kašnjenje** koje se odnosi na vreme koje je potrebno da se kopira operacija pisanja s primarnog na sekundarni čvor. Iako je manje kašnjenje prihvatljivo, problemi se javljaju s povećanjem replikacionog kašnjenja, uključujući pritisak na izgradnju keša na primarnom čvoru.

Od verzije MongoDB 4.2, administratori mogu ograničiti brzinu primarnog čvora kako bi održali većinu potvrđenih kašnjenja ispod konfigurabilne maksimalne vrednosti *flowControlTargetLagSeconds*. Podrazumevano, kontrola protoka je aktivirana. Sa omogućenom kontrolom protoka, kako kašnjenje raste i približava se vrednosti *flowControlTargetLagSeconds*, upisi na primarnom čvoru moraju dobiti "tickets" pre nego što preuzmu zaključavanje kako bi primenili upise. Ograničavanjem broja "tickets"-a izdatih u sekundi, mehanizam kontrole protoka pokušava održati kašnjenje ispod ciljne vrednosti.

3.1.2. Automatski prelazak na rezervni čvor

Kada primarni čvor ne komunicira s ostalim članovima skupa duže od konfigurisanog vremena *electionTimeoutMillis* (podrazumevano 10 sekundi), odgovarajući sekundarni čvor pokreće izbore kako bi se sam nominovao da postane novi primarni čvor, prikazano na slici 6. Klaster nastoji da završi izbor novog primarnog čvora i nastavi normalno sa radom.



Slika 6. Automatski prelazak na rezervni čvor [10]

Skup replika ne može obraditi operacije pisanja dok se izbor ne završi uspešno. Skup replika može i dalje odgovarati na upite za čitanje ukoliko su takvi upiti konfigurisani da se izvršavaju na sekundarnim čvorovima dok je primarni nedostupan.

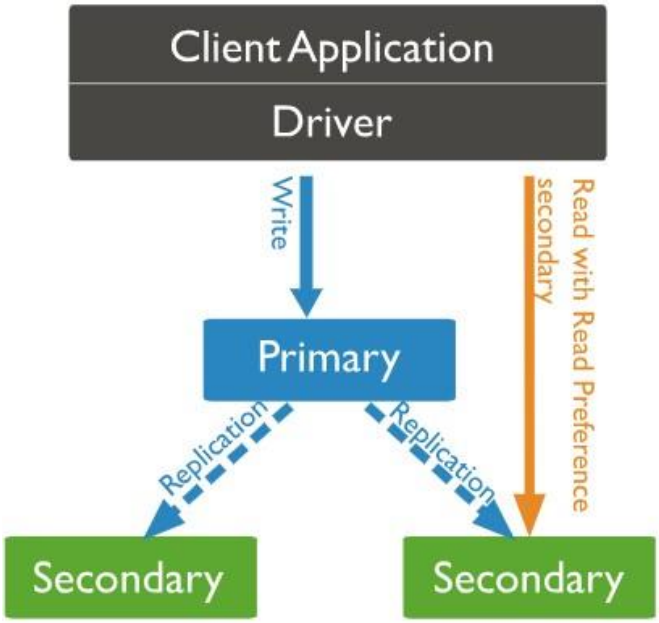
Srednje vreme pre nego što klaster izabere novi primarni čvor obično ne bi trebalo da prelazi 12 sekundi. Ovo uključuje vreme potrebno za označavanje primarnog čvora kao nedostupnog i pozivanje tekućih izbora. Faktori poput latencije mreže mogu produžiti vreme potrebno za završetak izbora u skupu replika, što, zauzvrat, utiče na vreme koje klaster može raditi bez primarnog čvora. Ovi faktori zavise od specifične arhitekture klastera.

Smanjenje opcije konfiguracije replikacije *electionTimeoutMillis* sa podrazumevanih 10 sekundi može rezultirati bržim otkrivanjem otkaza primara. Međutim, klaster može češće pokretati izbore usled faktora poput privremene latencije mreže, čak i ako je primarni inače ispravan. Ovo može dovesti do učestalijih povrataka za upisne operacije s w : 1.

Logika povezivanja aplikacije trebalo bi da uključuje toleranciju na automatske prelaze na rezervni čvor i nadolazeće izbore. MongoDB drajveri mogu detektovati gubitak primarnog i automatski ponovo pokušati određene upisne operacije jednom, pružajući dodatno ugrađeno upravljanje automatskim prelazima na rezervni čvor i izborima.

3.1.3. Operacije čitanja

Podrazumevano, klijenti vrše čitanja sa primarnog čvora. Međutim, klijenti imaju mogućnost specificiranja željenih opcija čitanja kako bi operacije čitanja bile upućene sekundarnim čvorovima, prikazano na slici 7.



Slika 7. Čitanje podataka [10]

Asinhrona replikacija na sekundarnim čvorovima znači da čitanja s tih čvorova mogu vratiti podatke koji ne odražavaju trenutno stanje podataka na primarnom čvoru. Distribuirane transakcije koje obuhvataju operacije čitanja obavezno moraju koristiti primarnu preferenciju čitanja. Sve operacije unutar jedne transakcije moraju biti usmerene ka istom članu. [10]

3.1.4. Vidljivost podataka

Na osnovu nivoa čitanja, klijenti mogu videti rezultate upisa pre nego što upisi postanu trajni.

Neki drugi klijenti koji koriste "local" ili "available" nivo čitanja mogu videti rezultate upisa pre nego što se potvrdi upis klijentu koji ga je izdao, bez obzira na write concern za taj upis. Klijenti sa "local" ili "available" nivoom čitanja mogu čitati podatke koji se kasnije mogu poništiti tokom prelaza na rezervni čvor skupa replika.

U operacijama u transakciji sa više dokumenata, kada se transakcija potvrdi, svi podaci promenjeni tokom transakcije su sačuvani i vidljivi izvan transakcije, bez mogućnosti da neki

delovi transakcije budu potvrđeni, a neki poništeni. Sve do potvrde transakcije, promene podataka napravljene tokom transakcije nisu vidljive izvan transakcije.

Međutim, kada transakcija upisuje na više shardova, ne sve vanjske operacije čitanja moraju čekati da rezultati potvrđene transakcije postanu vidljivi na svim shardovima. Na primer, ako je transakcija potvrđena i upis 1 je vidljiv na shardu A, ali upis 2 još nije vidljiv na shardu B, spoljašnje čitanje s nivoom čitanja "local" može pročitati rezultate upisa 1, a da ne vidi upis 2.

Reflektovana čitanja (engl. mirrored reads) smanjuju uticaj izbora primarnog čvora nakon prekida ili planiranih održavanja. Nakon prelaska na rezervni čvor u skupu replika, sekundarni čvor koji preuzima ulogu novog primarnog čvora ažurira svoj keš kako nove upite pristižu. Dok se keš zagreva, može doći do uticaja na performanse.

Počevši od verzije 4.4, reflektovana čitanja unapred zagrevaju keševe izabranih sekundarnih čvorova u skupu replika. Da bi unapred zagrejao keševe izabranih sekundarnih čvorova, primarni čvor oponaša uzorak podržanih operacija koje prima ka izabranim sekundarnim čvorovima.

Veličina podskupa izabranih sekundarnih čvorova skupa replika koji prima ogledna čitanja može se konfigurisati pomoću parametra *mirrorReads*. [10]

3.1.5. Praktičan primer kreiranja seta replika i rada sa njom

Konfiguracija seta replike

Svaki član replika seta u MongoDB-u je samostalan MongoDB server koji čuva kopiju podataka replika seta. Ovi članovi rade zajedno kako bi obezbedili visoku dostupnost, otpornost na kvarove i replikaciju podataka. U replika setu postoji jedan primarni član i nula ili više sekundarnih članova (maskimalno ih može biti 50). Kao što je ranije napomenuto primarni član je odgovoran za prijem svih upita za čitanje i pisanje. On je glavni server koji prima upite od klijenata i vrši pisanje podataka u bazu. Replikacijom podataka, promene na primarnom članu se automatski replikuju na sve sekundarne članove. Sekundarni članovi su replike podataka koje prate stanje primarnog člana. Oni pasivno primaju replikaciju podataka od primarnog člana.

Prvo je neophodno obaviti konfiguraciju jednog seta replika koji će se u ovom slučaju sadžati od jednog primara i dva sekundara. Potrebno je otvoriti tri odvojena terminala ili mongo shell-a i u njima redom izvršiti komande sa slika 8, 9 i 10. Recimo komanda na slici 8 pokreće MongoDB server sa replika setom *testReplicaSet*, postavlja putanju baze podataka na *C:\data\1*

i postavlja port na *27040* za prvu instancu. Slično, i komande sa slika 9 i 10, postavljaju putanje baze podataka i portove za drugu i treću instancu.

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --replSet testReplicaSet --dbpath "C:\data\1" --port 27040
```

Slika 8. Pokretanje prve instance seta replike

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --replSet testReplicaSet --dbpath "C:\data\2" --port 27041
```

Slika 9. Pokretanje druge instance seta replike

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --replSet testReplicaSet --dbpath "C:\data\3" --port 27042
```

Slika 10. Pokretanje treće instance seta replike

Zatim je neophodno povezati se sa bilo kojim od članova skupa replika koristeći komandu sa slike 11, koja se u ovom slučaju povezuje na prvu instancu sa portom *27040*.

```
C:\Program Files\MongoDB\Server\7.0\bin>mongosh --port 27040
Current Mongosh Log ID: 65c10c9defafd4e5499aaee4
Connecting to:          mongodb://127.0.0.1:27040/?directConn
Using MongoDB:          7.0.5
```

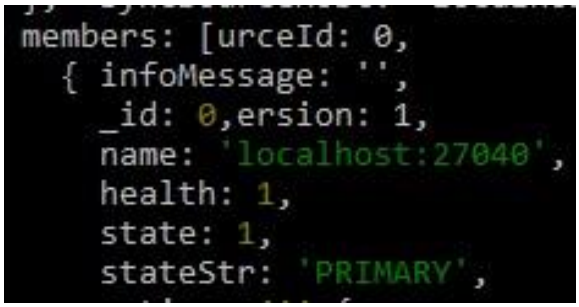
Slika 11. Povezivanje na prvu instancu skupa replika

Nakon toga izvršenjem komande *rs.initiate()*, kao na slici 12, na jednom od članova skupa replika pokreće se skup replika. Ova komanda inicijalizuje replika set sa nazivom *testReplicaSet* i konfiguriše tri člana replika seta, svaki sa svojim identifikatorom (_id), hostom i portom.

```
Enterprise test> rs.initiate(
...   {
...     _id: "testReplicaSet",
...     members: [
...     { _id : 0, host : 'localhost:27040' },
...     { _id : 1, host : 'localhost:27041' },
...     { _id : 2, host : 'localhost:27042' }
...     ]
...   }
... )
{ ok: 1 }
```

Slika 12. Inicijalizacija seta replika

Nakon ovih koraka, kreiran je MongoDB set replika sa tri člana koji radi na portovima *27040, 27041 i 27042*. Sada se može proveriti status replika seta i videti koji član je trenutno primarni. To se može učiniti korišćenjem komande *rs.status()* u MongoDB Shell-u. Među svim informacijama, ova komanda vraća članove skupa replika, kao što se vidi na slici 13, gde je instanca sa portom *27040* trenutno postavljena kao primarna.

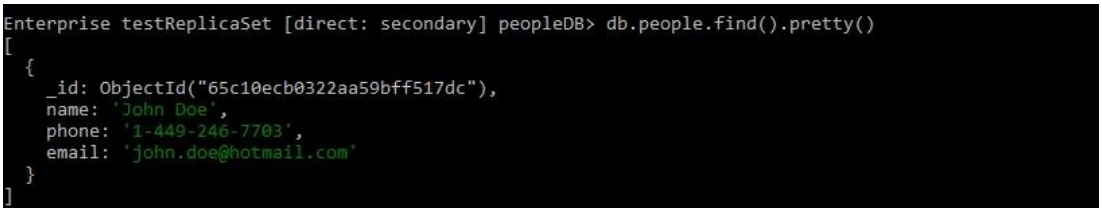


Slika 13. Provera statusa seta replika

Sada se može proveriti rad seta repilka upisvanjem nekog podatka na primar, kao na slici 14. Naredba *db.students.insertOne({...}, {...})* se koristi za dodavanje jednog dokumenta u kolekciju *people* unutar trenutne baze *peopleDB*. Argument komande predstavlja sam dokument koji se dodaje. U ovom slučaju, dokument ima tri polja: *name*, *phone*, i *email*, koji sadrže informacije o jednoj osobi. *writeConcern* je opcioni parametar koji omogućava konfigurisanje ponašanja upisa. U ovom slučaju, postavljene su opcije *w: "majority"* koja zahteva potvrdu od većine članova replika seta da je operacija uspešno izvršena pre nego što se smatra potvrđenom i *wtimeout: 5000* koja postavlja vreme u milisekundama koje server čeka na potvrdu od većine članova replika seta. U ovom slučaju, maksimalno vreme čekanja je 5000 milisekundi (5 sekundi).

Slika 14. Upis podataka na primarnu repliku

Ako se sada povežemo na neku drugu instancu seta replika i proverimo da li je podatak upisan i tamo. Izvršenjem komande *db.people.find.pretty()* dobija se rezultat da je student koji je dodat na primaru uspešno repliciran i na sekundarne replike.



Slika 15. Provera upisa na sekundarnim repilkama

3.2. Sharding

Sharding predstavlja tehniku distribucije podataka na više računara. MongoDB koristi sharding kako bi podržao primenu sa izuzetno obimnim skupovima podataka i operacijama velike propusnosti.

Sistemi baza podataka s obimnim skupovima podataka ili visokom propusnošću mogu postaviti izazov u vezi s kapacitetom pojedinog servera. Na primer, povećane stope upita mogu opteretiti CPU kapacitet servera. Grupisanje podataka koje premašuje RAM memoriju sistema može prouzrokovati opterećenje I/O kapaciteta disk jedinica.

Postoje dve metode koje se koriste za rešavanje rasta sistema : *vertikalno* i *horizontalno skaliranje*.

Vertikalno skaliranje podrazumeva povećanje kapaciteta jednog servera, što može uključivati upotrebu snažnijeg CPU-a, dodavanje više RAM-a ili proširenje prostora za skladištenje. Ograničenja u dostupnoj tehnologiji mogu sprečiti da jedna mašina bude dovoljno moćna za određeno radno opterećenje, a pružaoci usluga bazirani na oblaku često imaju čvrste granice u vezi s dostupnim hardverskim konfiguracijama. Iz toga proizlazi praktičan maksimum za vertikalno skaliranje.

Horizontalno skaliranje uključuje podelu podataka sistema i opterećenja na više servera, dodavanjem dodatnih servera radi povećanja kapaciteta po potrebi. Iako ukupna brzina ili kapacitet jedne mašine možda neće biti visoki, svaka mašina nosi se s podskupom ukupnog radnog opterećenja, potencijalno pružajući bolju efikasnost od jednog servera velike brzine i kapaciteta. Proširivanje kapaciteta implementacije zahteva samo dodavanje dodatnih servera po potrebi, što može rezultirati nižim ukupnim troškovima od nabavke vrhunskog hardvera za jednu mašinu. Kompromis uključuje povećanu složenost infrastrukture i održavanja za implementaciju. [2]

U MongoDB-u, sharding predstavlja tehniku za **horizontalnu distribuciju** baze podataka na više čvorova ili servera, poznatih kao "**šardovi**" (engl. shards). Svaki šard upravlja određenim segmentom podataka, formirajući podeljeni klaster koji omogućava MongoDB-u efikasno upravljanje velikim skupovima podataka i visokom konkurentnošću korisnika.

Unutar podeljenog klastera, svaki šard funkcioniše kao nezavisna baza podataka zadužena za skladištenje i upravljanje određenim podacima. Prilikom umetanja podataka u klaster, MongoDB-ov balanser automatski redistribuira podatke po šardovima, osiguravajući ravnotežu u distribuciji i opterećenju.

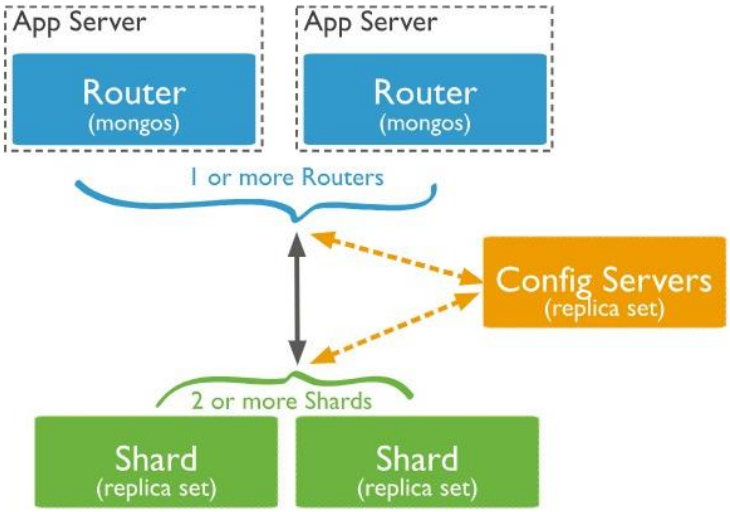
Šarding omogućava horizontalno skaliranje, gde se dodaju dodatni serveri ili čvorovi u klaster kako bi se obradili sve veći podaci i zahtevi korisnika. Horizontalno skaliranje poboljšava performanse i kapacitet jer se radno opterećenje raspodeljuje na više mašina, smanjujući rizik od jedne tačke otkaza (engl. Single Point Of Failure). Ovaj pristup je kontrast vertikalnom skaliranju, koje podrazumeva povećanje resursa jednog servera, što može biti ograničeno hardverskim kapacitetima i postati skupo kako zahtevi rastu. Sharding je poželjan za sisteme baza podataka koji doživljavaju značajan rast i zahtevaju visok nivo dostupnosti. [3]

3.2.1. Komponente sharding klastera

MongoDB podeljeni klaster sastoji se od sledećih ključnih komponenata:

- **šard (engl. shard)** - Svaki šard sadrži određeni deo podeljenih podataka, a svaki od njih može biti primenjen kao skup replika.
- **konfiguracioni serveri** - Konfiguracioni serveri čuvaju metapodatke i informacije o konfiguraciji za podeljeni klaster, uključujući detalje o distribuciji podataka po šardovima, opsegu čankova (engl. chunks) i ključu šarda. Konfiguracioni serveri olakšavaju koordinaciju upita i migracije podataka unutar podeljenog klastera.
- **mongos** - Mongos funkcioniše kao ruter upita, pružajući interfejs između klijentskih aplikacija i podeljenog klastera. Oni primaju zahteve, usmeravaju upite do odgovarajućih delova i objedinjuju rezultate kada je to potrebno. Mongos procesi sakrivaju osnovnu složenost šardovanja od aplikacije, čineći je kao jedinstvenu logičku bazu podataka.

Zajedno, ove komponente formiraju arhitekturu šardinga, koja omogućava MongoDB-u da se horizontalno skalira i efikasno upravlja ogromnim količinama podataka. Na slici 16 ispod prikazan je grafikon koji opisuje interakciju komponenti unutar podeljenog klastera. MongoDB deli podatke na nivou kolekcija, distribuirajući podatke kolekcije po delovima u klasteru. [2], [4]

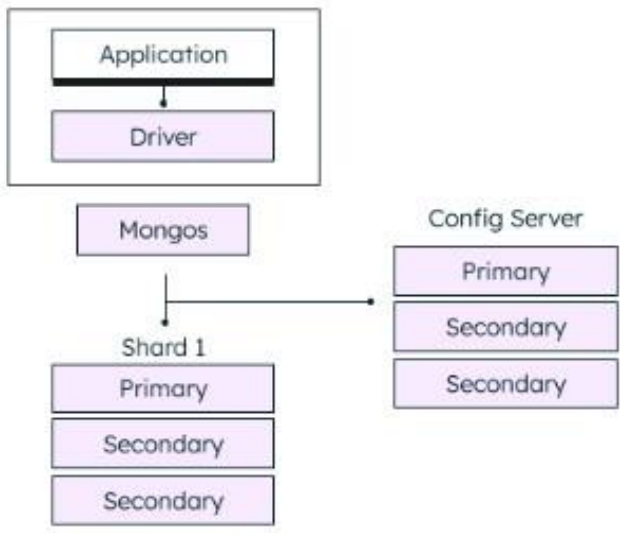


Slika 16. Interakcija komponenti unutar podeljenog klastera [2]

Za testiranje i razvoj, može se primeniti podeljeni klaster sa minimalnim brojem komponenti. Ovi neproizvodni klasteri imaju sledeće komponente:

- Jednu mongos instancu
- Jedan šard replika skup
- Konfiguracioni server skupa replika

Na slici 17 prikazana je arhitektura podeljenog klastera koja se koristi samo za razvoj:



Slika 17. Arhitektura podeljenog klastera [4]

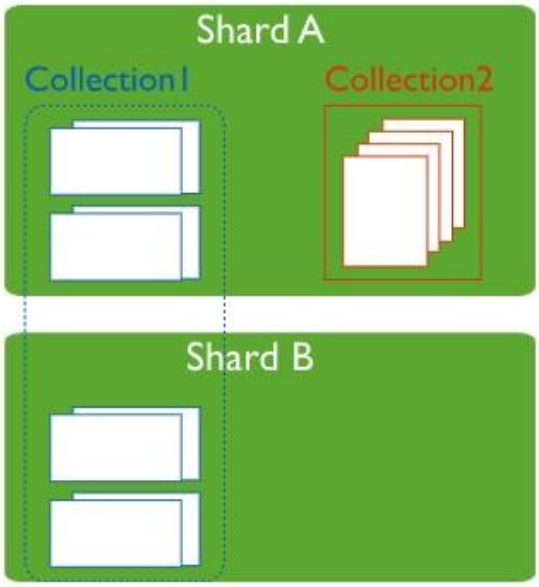
Šard (engl. Shard)

Šard sadrži podskup podeljenih podataka za podeljeni klaster. Zajedno, delovi klastera drže ceo skup podataka za klaster. Šardovi moraju biti raspoređeni kao skup replika da bi se obezbedila redundantnost i visoka dostupnost. U okviru podeljenog klastera, svaka baza podataka poseduje **primarni šard** koji sadrži sve kolekcije specifične za tu bazu.

Prilikom kreiranja nove baze podataka, mongos bira primarni šard tako što bira šard klastera koji sadrži najmanju količinu podataka. Za ovaj izbor, mongos koristi informacije o ukupnoj veličini (*totalSize*) koje su dobijene iz komande *listDatabases*.

Za promenu primarnog šarda baze podataka koristi se komanda *movePrimary*. Proces migracije primarnog šarda može potrajati dosta dugo, pa se preporučuje izbegavanje pristupa kolekcijama povezanim sa tom bazom podataka tokom trajanja migracije. U zavisnosti od obima podataka koji se prenose, migracija može uticati na celokupne operacije klastera.

Prilikom primene novog podeljenog klastera sa šardovima koji su prethodno funkcionisali kao skupovi replika, postojeće baze podataka i dalje ostaju u svojim originalnim skupovima replika. Baze podataka koje su kreirane nakon toga mogu se smestiti na bilo koji šard unutar klastera. [5]



Slika 18. Primarni šard [5]

Konfiguracioni serveri

Konfiguracioni serveri igraju ključnu ulogu u čuvanju metapodataka za distribuirani klaster. Ovi metapodaci detaljno opisuju stanje i organizaciju svih podataka i komponenti unutar klastera, uključujući listu čankova na svakom šardu i opsege koji precizno definišu te čankove.

Mongos instance efikasno keširaju ove metapodatke i koriste ih za precizno usmeravanje operacija čitanja i pisanja ka odgovarajućim šardovima. Važno je napomenuti da se Mongos redovno ažurira, posebno kada dođe do promena u metapodacima klastera, kao što su dodavanje novog šarda. Šardovi takođe aktivno čitaju metapodatke čankova sa konfiguracionih servera, što doprinosi efikasnom upravljanju podacima unutar klastera.

Pored toga, konfiguracioni serveri ne samo da čuvaju metapodatke, već i važne informacije o konfiguraciji autentifikacije. Ovi podaci obuhvataju kontrolu pristupa zasnovanu na ulozi (RBAC) i interna podešavanja autentifikacije za ceo klaster.

Značajno je napomenuti da svaki podeljeni klaster zahteva sopstvene servere za konfiguraciju radi optimalnog funkcionisanja. Upotreba istih servera za konfiguraciju za različite raščlanjene klastere nije preporučljiva, jer to može dovesti do problema u efikasnosti i upravljanju podacima.

Konfiguracioni serveri za podeljene klastere mogu se konfigurisati kao skup replika. Implementacija skupa replika za konfiguracione servere unapređuje konzistentnost među svim konfiguracionim serverima, s obzirom na to da MongoDB koristi standardni skup replika za protokole čitanja i upisivanja podataka o konfiguraciji. Osim toga, upotreba skupa replika omogućava podeljenom klasteru da koristi više od tri konfiguraciona servera, pošto skup replika može imati do 50 članova.

Postoji nekoliko ograničenja koja se odnose na konfiguraciju skupa replika kada se koristi za konfiguracione servere:

- Skup replika mora imati nula arbitara (arbitar je član skupa replika koji ne čuva kopiju podataka, već se koristi samo za odlučivanje u situacijama glasanja ili izbora)
- Ne sme sadržavati odložene članove
- Obavezno je pravljenje indeksa (tj. nijedan član ne sme imati postavku members[n].buildIndexes postavljenu na false). [6]

Konfiguracioni serveri obavljaju ključne operacije **čitanja** i **pisanja** u MongoDB okruženju. Na tim serverima postoje *administratorska baza podataka* i *konfiguraciona baza podataka*, od kojih svaka ima svoju specifičnu ulogu i funkciju.

Administratorska baza podataka sadrži kolekcije koje se koriste za čuvanje podataka o autentifikaciji i autorizaciji, zajedno sa drugim *system.** kolekcijama predviđenim za internu upotrebu. Slično tome, konfiguraciona baza podataka sadrži kolekcije koje čuvaju metapodatke o klasteru. Kada dođe do promena u metapodacima, kao što su migracija ili podele čanka, MongoDB vrši upisivanje podataka u konfiguracionu bazu. Šardovi čitaju metapodatke čankova sa konfiguracionih servera kako bi održali ažuriranu sliku o klasteru.

U okviru podeljenog klastera, mongod i mongos instance nadgledaju različite skupove replika, uključujući skupove replika šardova i skup replika konfiguracionog servera. Ako svi konfiguracioni serveri postanu nedostupni, cela funkcionalnost klastera može biti ugrožena. Radi očuvanja dostupnosti i stabilnosti konfiguracionih servera, neophodno je imati rezervne kopije. Važno je napomenuti da podaci na serverima za konfiguraciju obično imaju malu veličinu u poređenju sa podacima smeštenim u klasteru, što rezultira relativno niskim opterećenjem aktivnosti na konfiguracionim serverima.

Metapodaci klastera čuvaju se u konfiguracionoj bazi podataka, koja je pod kontrolom konfiguracionih servera. Za pristup konfiguracionoj bazi podataka, potrebno je koristiti sledeću komandu u MongoDB shell-u: *use config*

Opšte pravilo je da se nikada ne treba direktno menjati sadržaj konfiguracione baze podataka. Ona obuhvata različite kolekcije, kao što su changelog, chunks, collections, databases, csrs.indexes, mongos, settings, shards, version,.. Ovaj skup kolekcija ključan je za pravilno funkcionisanje i održavanje podeljenog klastera, pa je važno pridržavati se predloženih procedura kako bi se izbegle neželjene posledice. [6]

Mongos

MongoDB mongos instance igraju ključnu ulogu u usmeravanju upita i operacija pisanja ka šardovima u podeljenom klasteru. Za aplikacije, mongos pruža jedini interfejs ka podeljenom klasteru. Važno je napomenuti da aplikacije nikada ne ostvaruju direktnu vezu ili komunikaciju sa šardovima.

Mongos prati koji se podaci nalaze na kom šardu keširajući metapodatke sa konfiguracionih servera. Ovi metapodaci omogućavaju mongosu da usmerava operacije od

aplikacija i klijenata do odgovarajućih mongod instanci. Važno je napomenuti da mongos nema trajno stanje i troši minimalne sistemske resurse, čineći ga efikasnim u pogledu performansi.

Uobičajena praksa je pokretanje mongos instanci na istim sistemima kao i servere aplikacija. Međutim, moguće je održavati mongos instance na šardovima ili drugim namenskim resursima, prilagođavajući ih specifičnim potrebama i zahtevima klastera. Ovaj pristup pruža fleksibilnost u optimizaciji performansi i raspodele resursa u skladu sa karakteristikama sistema.

Instanca mongos usmerava upit ka klasteru na sledeći način:

- Identifikuje liste šardova koji treba da prime upit.
- Postavlja kursore na svim ciljanim šardovima.

Nakon toga, mongos spaja podatke sa svakog ciljanog šarda i vraća rezultujući dokument. Određene modifikacije upita, kao što je sortiranje, vrše se na svakom šardu pre nego što mongos vrati rezultate.

Agregacione operacije koje se izvršavaju na više šardova mogu vratiti rezultate nazad mongosu radi spajanja, ukoliko nije potrebno da se izvrše na primarnom šardu baze podataka.

Rezultat uključuje tri JSON objekta:

- *mergeType* pokazuje gde se faza spajanja odvija (*primaryShard*, *anyShard* ili *mongos*).
- *splitPipeline* pokazuje koje operacije u pipelinu su izvršene na pojedinačnim šardovima.
- *shards* prikazuje rad koji svaki šard obavlja.

U nekim slučajevima, kada je ključ šarda ili prefiks ključa šarda deo upita, mongos izvodi ciljanu operaciju, usmeravajući upite ka podskupu šardova u klasteru.

Mongos izvodi emisiju operacije za upite koji ne uključuju ključ šarda, usmeravajući upite ka svim šardovima u klasteru. Neki upiti koji uključuju ključ šarda i dalje mogu rezultirati emisijom operacije, u zavisnosti od distribucije podataka u klasteru i selektivnosti upita. [7]

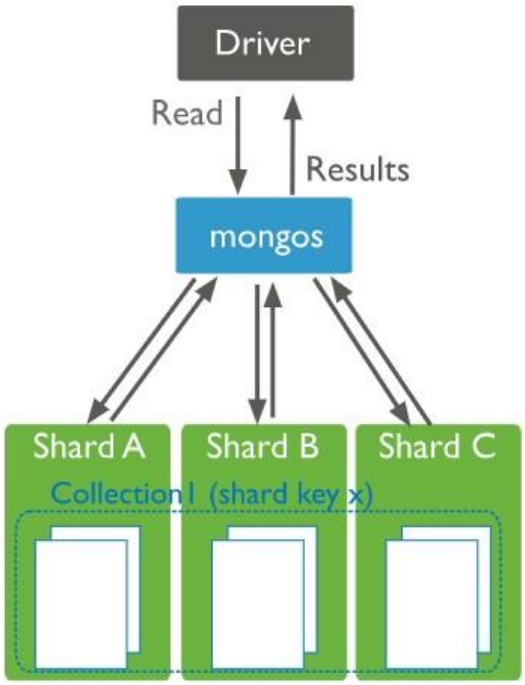
Postoje dva načina slanja upita: **operacije emitovanja** (engl. broadcast) i **ciljane operacije** (engl. targeted).

Najbrži upiti u podeljenom okruženju su oni koje mongos usmerava ka jednom šardu, koristeći ključ šarda i metapodatke klastera sa konfiguracionog servera. Ove ciljane operacije koriste vrednost ključa šarda kako bi locirale deo ili podskup delova koji zadovoljavaju kriterijume upita.

Za upite koji ne uključuju ključ šarda, mongos mora da pošalje upit svim šardovima, sačeka njihove odgovore, a zatim vrati rezultate aplikaciji. Ovi upiti scatter/gather

(raspršiti/prikupiti) mogu biti dugotrajne operacije koje zavise od ukupnog opterećenja klastera, kao i od faktora poput kašnjenja mreže, opterećenja pojedinačnih šardova i broja dokumenata vraćenih po šardu.

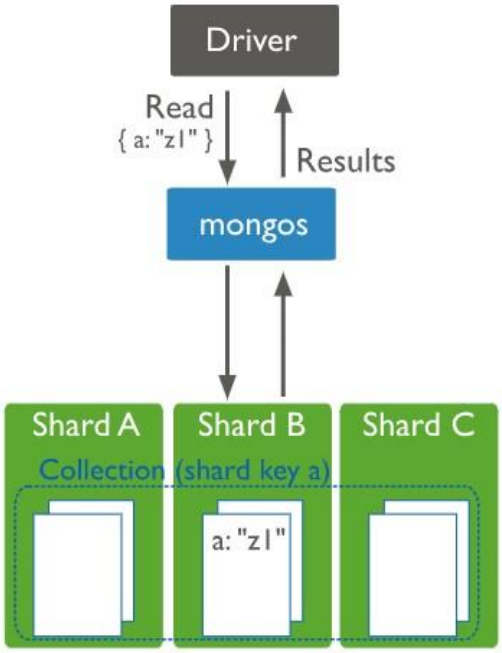
Što se tiče operacija **emitovanja** (engl. broadcast operations), mongos instance šalju upite svim šardovima za kolekciju kao na slici 19, osim ako mongos ne može precizno odrediti koji deo ili podskup delova čuva određene podatke. Kada mongos primi odgovore od svih šardova, spaja podatke i vraća rezultujući dokument. Efikasnost operacija emitovanja zavisi od opšteg opterećenja klastera, kao i od faktora poput kašnjenja mreže, opterećenja pojedinačnih šardova i broja dokumenata vraćenih po šardu. Uvek se preporučuje davanje prednosti ciljanim operacijama nad onima koje koriste operacije emitovanja, kada god je to moguće. [7]



Slika 19. Mongos – slanje upita svim šardovima [7]

Operacije višestrukog ažuriranja uvek predstavljaju operacije emitovanja. Metode updateMany() i deleteMany() su operacije emitovanja, osim ako dokument upita potpuno ne navede ključ šarda.

Mongos može usmeravati upite koji obuhvataju ključ šarda ili prefiks kompleksnog ključa šarda ka **ciljanom** šardu ili skupu šardova. U ovom procesu, mongos koristi vrednost ključa šarda kako bi pronašao deo čiji opseg uključuje navedenu vrednost ključa šarda, nakon čega usmerava upit na taj određeni deo.



Slika 20. Mongos – slanje upita ciljanom šardu [7]

U slučaju kada je ključ šarda, na primer, postavljen kao { a: 1, b: 1, c: 1 }, mongos ima sposobnost da usmerava upite koji obuhvataju ceo ključ šarda ili bilo koji od narednih prefiksa ključa šarda ka određenom segmentu ili skupu delova: { a: 1 }, { a: 1, b: 1 }. Ovo omogućava precizno usmeravanje upita na određene delove klastera, poboljšavajući efikasnost operacija.

Sve operacije *insertOne*() su usmerene ka jednom šardu. Međutim, važno je napomenuti da svaki dokument u nizu *insertMany*() cilja jedan deo, ali ne postoji apsolutna garancija da će svi dokumenti u nizu biti umetnuti u isti deo. Ovo se odnosi na specifičnosti raspodele podataka u klasteru.

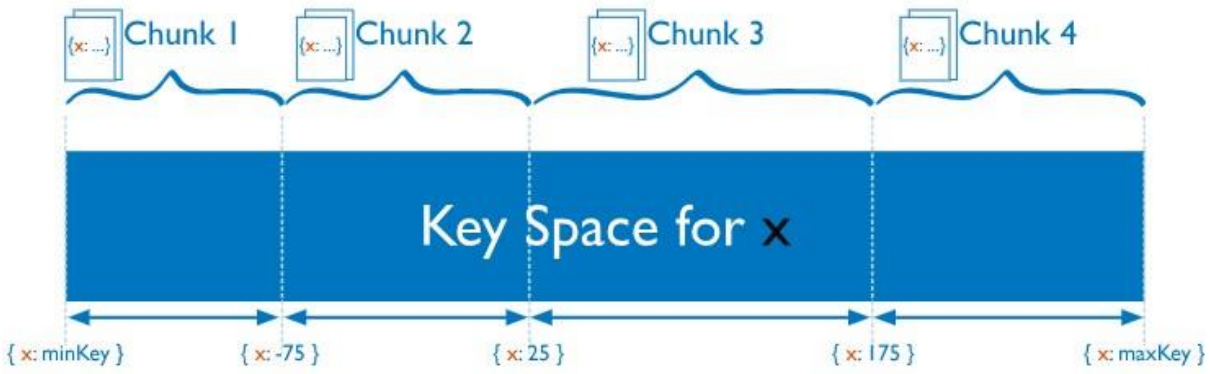
Za operacije *updateOne*(), *replaceOne*() i *deleteOne*(), neophodno je uključiti ključ šarda ili _id u dokumentu upita. MongoDB signalizira grešku ako se ove metode koriste bez ključa šarda ili _id, obezbeđujući doslednost i tačnost u manipulaciji podacima.

Zavisno o raspodeli podataka u klasteru i selektivnosti upita, mongos-i i dalje mogu primenjivati operaciju emitovanja kako bi zadovoljili zahteve ovih upita. Ova fleksibilnost omogućava prilagodljivost u radu sa različitim distribucijama podataka i vrstama upita koje se postavljaju u sistemu. [7]

Ključevi šarda i čankovi

MongoDB organizuje podatke koristeći ključ šarda povezan sa kolekcijom kako bi podelio podatke na čankove koji pripadaju određenom šardu. Svaki čank se sastoji od niza podeljenih podataka, pri čemu opseg može obuhvatati deo čanka ili celokupan čank.

Svaki deo čanka ima definisane inkluzivne donje i isključive gornje granice, a te vrednosti se određuju na osnovu ključa šarda. Ovaj pristup omogućava precizno određivanje granica i efikasno upravljanje podacima unutar klastera.



Slika 21. Sharded key [8]

Bitna karakteristika čanka je da predstavlja najmanju jedinicu podataka, a ta jedinica je određena jedinstvenom vrednošću ključa šarda. Ova jedinstvenost ključa šarda čini svaki čank jasno definisanim entitetom u sistemu, olakšavajući procese manipulacije podacima i upravljanja klasterom.

Ključ šarda je ili jedno indeksirano polje ili više polja pokrivenih složenim indeksom koji određuje distribuciju dokumenata kolekcije među delovima klastera.

MongoDB deli raspon vrednosti ključeva šarda (ili heširanih vrednosti ključeva šarda) na nepreklapajuće opsege vrednosti ključeva šarda (ili heširanih vrednosti ključeva šarda). Svaki opseg je povezan sa čankom, a MongoDB pokušava da ravnomerno rasporedi delove između delova u klasteru.

3.2.2. Praktičan primer konfiguracije sharding klastera kod MongoDB

Šarding klaster u MongoDB-u se sastoji od šardova, konfiguracionih servera i rutera, ili mongos procesa. U ovom slučaju je kreiran šarding klaster koji se sastoji od konfiguracionog servera koji ima 3 replike, 3 šard servera, od kojih svaki ima po 3 replike i mongos rutera.

Konfiguracija konfiguracionog servera

Prvo je potrebno kreirati i konfigurisati **konfiguracioni server klastera**, kao na slici 22. Konfiguracioni server pokrenut je kao set od 3 replike, pa opcija *--replSet configReplSet* postavlja ime tog seta replika, opcija *--configsvr* označava da će se pokrenuti konfiguracioni server, *--dbpath "C:\data\configdb1"* postavlja putanju do direktorijuma gde će prva replika konfiguracionog servera čuvati podatke konfiguracionog servera, *--port 27019* postavlja port na kojem će konfiguracioni server slušati konekcije. Potrebno je pokrenuti sve tri replike konfigruracionog servera kao na slici 22.

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --configsvr --replSet configReplSet
--port 27019 --dbpath "C:\data\configdb1"
```

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --configsvr --replSet configReplSet
--port 27020 --dbpath "C:\data\configdb2"
```

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --configsvr --replSet configReplSet
--port 27021 --dbpath "C:\data\configdb3"
```

Slika 22. Kreiranje replika konfiguracionog servera

Zatim je potrebno inicijalizovati ovaj set replika za konfiguracioni server kao na slici 23. Postupak je opisan u poglavlju 3.1.5, jedina razlika je parametar *configsvr: true* koji označava da je u pitanju konfiguracioni server.

```
Enterprise test> rs.initiate(
...   {
...     _id: "configReplSet",
...     configsvr: true,
...     members: [
...       { _id: 0, host: "localhost:27019" },
...       { _id: 1, host: "localhost:27020" },
...       { _id: 2, host: "localhost:27021" }
...     ]
...   }
... )
{ ok: 1 }
```

Slika 23. Inicijalizacija seta replika konfiguracionog servera

Konfiguracija šardova

Zatim je potrebno konfigurisati šardove. Ovde su kreirana 3 šarda, od kojih svaki ima po 3 replike. Konfiguracija se obavlja na sledeći način. Za svaki šard neophodno je prvo pokrenuti 3 replike, kao na slici 24. Prvi šard ima 3 replike na portovima *27022*, *27023* i *27024*, i svaka replika čuva podatke u folderima, respektivno, *"C:\data\shard1\1"*, *"C:\data\shard1\2"* i *"C:\data\shard1\3"*.

```
C:\Program Files\MongoDB\Server\7.0\bin>mongod --shardsvr --replSet shard1ReplSet
--port 27022 --dbpath "C:\data\shard1\1"

C:\Program Files\MongoDB\Server\7.0\bin>mongod --shardsvr --replSet shard1ReplSet
--port 27023 --dbpath "C:\data\shard1\2"

C:\Program Files\MongoDB\Server\7.0\bin>mongod --shardsvr --replSet shard1ReplSet
--port 27024 --dbpath "C:\data\shard1\3"
```

Slika 24. Konfiguracija prvog šarda

Nakon toga potrebno je povezati se za jednu repliku ovog šarda i inicijalizovati set replika, povezivanje vrši se komandom *mongosh --host localhost:27022*, a inicijalizacija je prikazana na slici 25.

```
Enterprise test> rs.initiate(
...   {
...     _id: "shard1ReplSet",
...     members: [
...       { _id: 0, host: "localhost:27022" },
...       { _id: 1, host: "localhost:27023" },
...       { _id: 2, host: "localhost:27024" }
...     ]
...   }
... )
{ ok: 1 }
```

Slika 25. Inicijalizacija seta replika prvog šarda

Ovaj postupak potrebno je obaviti i za preostala dva šarda. Drugi šard takođe ima 3 replike na portovima *27025*, *27026* i *27027*, njegov replika set naziva se *shard2ReplSet,* a podaci pamte se respektivno u folderima, *"C:\data\shard2\1"*, *"C:\data\shard2\2"* i *"C:\data\shard2\3"*. Treći šard ima 3 replike na portovima *27028*, *27029* i *27030*, njegov replika set naziva se *shard3ReplSet,* a podaci pamte se respektivno u folderima, *"C:\data\shard3\1"*, *"C:\data\shard3\2"* i *"C:\data\shard3\3"*.

Konfiguracija mongos rutera

Nakon što su konfigurisani šard i konfiguracioni serveri, potrebno je konfigurisati mongos ruter upita, kao na slici 26. Ova komanda se koristi za pokretanje mongos procesa, koji je MongoDB server tipa sharded router (šard rutera). *--configdb configReplSet/localhost:27019, localhost:27020, localhost:27021* opcija specificira konfiguracionu replika set koja sadrži informacije o raspodeljenim ključevima i lokacijama šardova, koji je kreiran u tački 1. Adrese replika u konfiguracionom replikacionom setu su *localhost:27019, localhost:27020*, i *localhost:27021*. mongos koristi ovu konfiguraciju da bi znao kako da rutira upite između različitih šardova i kako da upravlja podacima u klasteru. *--port 27031* opcija postavlja port na kojem će mongos osluškivati za dolazne konekcije.

```
C:\Program Files\MongoDB\Server\7.0\bin>mongos --configdb configReplSet/localhost:27019
,localhost:27020,localhost:27021 --port 27031
```

Slika 26. Konfiguracija mongos rutera

Ovom komandom se uspostavlja MongoDB sharded klaster gde `mongos` služi kao posrednik između aplikacije i šardova. Kada aplikacija šalje upit, mongos koristi informacije iz konfiguracione replike *configReplSet* da bi znao koje šardove treba kontaktirati kako bi dobio rezultate upita. Osim toga, mongos obavlja posao rutiranja upita, prateći šard ključeve i usmeravajući upite na odgovarajuće shardove kako bi optimizovao distribuciju podataka.

Dodavanje šardova u klaster

Sada je potrebno povezati se na mongos server komandom *mongosh --port 27031* i na njemu dodati sve kreirane šardove u klaster. Na slici 27 prikazan je postupak dodavanja prvog šarda.

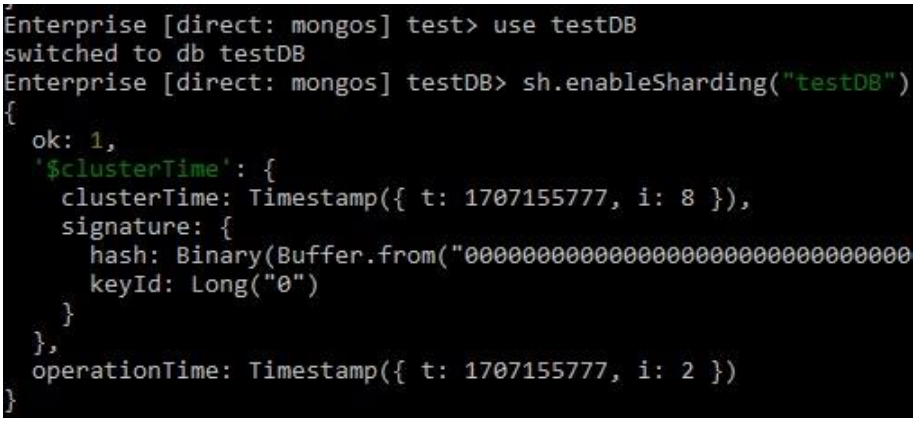
```
Enterprise [direct: mongos] test> sh.addShard("shard1ReplSet/localhost:27022")
{
  shardAdded: 'shard1ReplSet',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1707155575, i: 5 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1707155575, i: 5 })
}
```

Slika 27. Dodavanje šarda u klaster

Na isti način potrebno je dodati i preostala dva šarda komandama: *sh.addShard("shard2ReplSet/localhost:27025")* i *sh.addShard("shard3ReplSet/localhost:27028")*

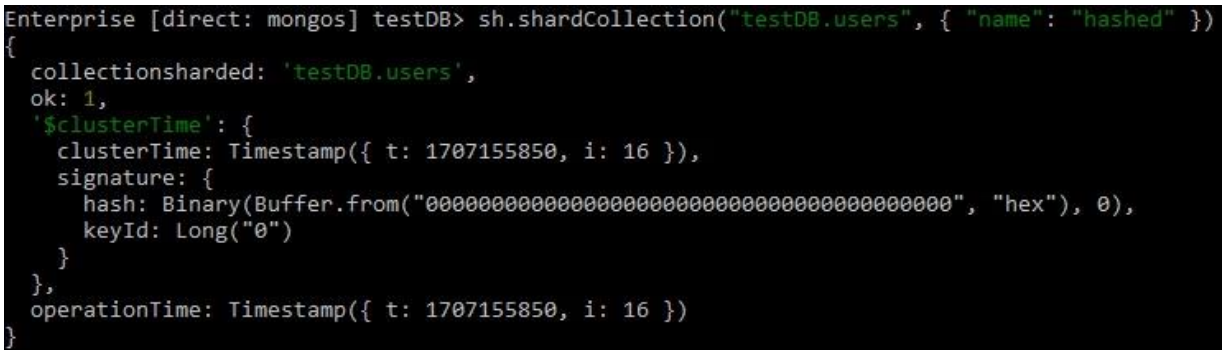
Omogućavanje šardinga nad izabranom bazom podataka i šardovanje kolekcije

Komande sa slike 28 se koriste za postavljanje šardinga za određenu kolekciju u MongoDB bazi podataka. *use testDB* komanda se koristi za prebacivanje trenutnog konteksta baze podataka na testDB. Ako baza podataka ne postoji, MongoDB će je automatski kreirati. Komanda *sh.enableSharding("testDB")* omogućava šarding za odabranu bazu podataka, u ovom slučaju, za testDB. Šardovanje se omogućava kako bi se povećala horizontalna skalabilnost i distribuirala baza podataka na više šardova. Kada se ova komanda izvrši, MongoDB će postaviti sharding metapodatke za bazu podataka testDB.



Slika 28. Omogućavanje šardinga

Na kraju komanda *sh.shardCollection("testDB.users", { "name": "hashed"* }), sa slike 29, postavlja šarding za određenu kolekciju, u ovom slučaju, za kolekciju *users* unutar baze podataka *testDB*. Šarding se vrši na osnovu polja *name*, a konkretno se koristi funkcija *hashed* kako bi se vrednosti polja name haširale pre nego što se koriste za raspodelu podataka između šardova. Korišćenje haširanja pomaže u ravnomernijoj distribuciji podataka između šardova, čime se postiže bolja skalabilnost.



Slika 29. Šardovanje kolekcije users

Nakon što je sve podešeno, mogu se dodati podaci u testDB bazu podataka, kao na slici 30.

```
Enterprise [direct: mongos] testDB> var usersDataWithId = [];

Enterprise [direct: mongos] testDB> for (var i = 1; i <= 50; i++) {
...     usersDataWithId.push({
...         id: i,
...         name: "User" + i,
...         email: "user" + i + "@example.com",
...         phone: "123456789" + i
...     });
... }
50
Enterprise [direct: mongos] testDB> db.users.insertMany(usersDataWithId);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("65c122bc86585843ea36bd7c"),
```

Slika 30. Dodavanje podataka u šarding klaster

Zatim se komandom *db.users.getShardDistribution()*, kao na slici 31, dobija pregled raspodele podataka (chunk-ova) unutar kolekcije users na različitim šardovima u šarding okruženju. Ova komanda prikazuje informacije o tome koliko podataka pripada svakom šardu, koliko čankova postoji, i druge relevantne informacije vezane za šarding.

```
Shard shard2ReplSet at shard2ReplSet/localhost:27025,localhost:27026,localhost:27027
{
  data: '1KiB',
  docs: 13,
  chunks: 2,
  'estimated data per chunk': '650B',
  'estimated docs per chunk': 6
}
---
Shard shard3ReplSet at shard3ReplSet/localhost:27028,localhost:27029,localhost:27030
{
  data: '1KiB',
  docs: 15,
  chunks: 2,
  'estimated data per chunk': '747B',
  'estimated docs per chunk': 7
}
---
Shard shard1ReplSet at shard1ReplSet/localhost:27022,localhost:27023,localhost:27024
{
  data: '2KiB',
  docs: 22,
  chunks: 2,
  'estimated data per chunk': '1KiB',
  'estimated docs per chunk': 11
}
```

```
Totals
{
  data: '4KiB',
  docs: 50,
  chunks: 6,
  'Shard shard2ReplSet': [
    '26.14 % data',
    '26 % docs in cluster',
    '100B avg obj size on shard'
  ],
  'Shard shard3ReplSet': [
    '30.04 % data',
    '30 % docs in cluster',
    '99B avg obj size on shard'
  ],
  'Shard shard1ReplSet': [
    '43.81 % data',
    '44 % docs in cluster',
    '99B avg obj size on shard'
  ]
}
```

Slika 31. Pregled rasporedele podataka

Podaci su podeljeni na tri šarda: shard1ReplSet, shard2ReplSet, i shard3ReplSet. Na *shard3ReplSet* nalazi se oko 30.04% ukupnih podataka u klasteru, *shard2ReplSet* šard zauzima oko 26.14%, na kraju, *shard1ReplSet* sadrži oko 43.81% ukupnih podataka u klasteru. Može se zaključiti da su podaci relativno ravnomerno raspoređeni između šardova, svaki šard ima isti broj čankova, a prosečna veličina objekta na šardovima je slična.

3.3. Distribuirane transakcije kod MongoDB skladišta podataka

U MongoDB-u, operacije nad pojedinačnim dokumentima su atomične. Ova atomičnost se odnosi na to da se operacije nad jednim dokumentom izvršavaju u potpunosti ili se ne izvršavaju uopšte. Korišćenjem ugnježdenih dokumenata i nizova, moguće je zabeležiti odnose između podataka unutar jedne strukture dokumenata, umesto normalizacije preko više dokumenata i kolekcija. Ova atomičnost operacija na jednom dokumentu eliminiše potrebu za distribuiranim transakcijama u mnogim praktičnim scenarijima.

U situacijama gde je potrebna atomičnost čitanja i pisanja nad više dokumenata (u jednoj ili više kolekcija), MongoDB podržava distribuirane transakcije. Distribuirane transakcije omogućavaju korišćenje transakcija preko više operacija, kolekcija, baza podataka, dokumenata i šardova. Ovo pruža mogućnost održavanja konzistentnosti podataka u složenim okruženjima gde je neophodno obezbediti integritet podataka širom sistema. [11]

3.3.1. Atomičnost distribuiranih transakcija

U situacijama koje zahtevaju atomičnost čitanja i pisanja nad više dokumenata (u jednoj ili više kolekcija), MongoDB podržava distribuirane transakcije, uključujući transakcije na setovima replika i sharding klasterima.

Distribuirane transakcije su atomične. One pružaju "sve ili ništa" opciju, što znači da:

- Kada se transakcija potvrdi, sve promene podataka napravljene tokom transakcije su sačuvane i vidljive izvan same transakcije. Do trenutka kada se transakcija potvrdi, promene podataka napravljene tokom transakcije nisu vidljive izvan same transakcije. Međutim, kada transakcija zapisuje podatke na više šardova, ne sve operacije čitanja izvan transakcije moraju čekati rezultat potvrđene transakcije da bi bile vidljive širom šardova. Na primer, ako je transakcija potvrđena i zapis 1 je vidljiv na šardu A, ali zapis 2 još nije vidljiv na šardu B, spoljni upit sa nivoom čitanja "local" može čitati rezultate zapisa 1 bez uvida u zapis 2.
- Kada se transakcija prekine, sve promene podataka napravljene tokom transakcije se odbacuju, i nikada ne postanu vidljive. Na primer, ukoliko bilo koja operacija u transakciji ne uspe, transakcija se prekida i sve promene podataka napravljene tokom transakcije se odbacuju, a nikada ne postanu vidljive.

3.3.2. Operacije u MongoDB transakcijama

MongoDB pruža snažan model za upravljanje podacima putem distribuiranih transakcija, omogućavajući programerima izvođenje različitih operacija unutar ovog okvira. Podržane operacije u MongoDB distribuiranim transakcijama obuhvataju:

- **CRUD operacije nad kolekcijama** - Unutar transakcije, MongoDB podržava osnovne CRUD operacije nad kolekcijama. To uključuje operacije agregacije *db.collection.aggregate(), db.collection.countDocuments()* za dobijanje broja dokumenata u kolekciji, *db.collection.distinct*() operaciju (primenjivu samo na kolekcije koje nisu šardovane), kao i *db.collection.find()* operaciju. Takođe, operacije *kreiranja*, *ažuriranja* i *brisanja* mogu se primenjivati kako na pojedinačne tako i na grupe dokumenata.
- **Operacije administracije** - Od verzije 4.4 MongoDB-a, moguće je izvršiti operacije kreiranja indeksa *db.collection.createIndex()*, *db.collection.createIndexes()* i novih kolekcija unutar distribuiranih transakcija *db.createCollection().* Ova funkcionalnost dodatno povećava fleksibilnost pri upravljanju strukturom podataka tokom transakcija.
- **Informacione operacije** - Distribuirane transakcije podržavaju operacije koje pružaju informacije, poput *hello*, *buildInfo* i *connectionStatus*. Bitno je napomenuti da se ove operacije ne smeju navesti kao prva operacija u transakciji.
- **Rad sa različitim bazama podataka** - MongoDB omogućava izvođenje operacija unutar transakcije nad različitim bazama podataka, što olakšava integraciju i sinhronizaciju podataka između različitih delova sistema.

Takođe, MongoDB transakcije podržavaju uobičajene operacije poput unosa novih dokumenata, ažuriranja postojećih, brisanja, pretrage, kreiranja i brisanja kolekcija te upravljanje indeksima. Ovi tipovi operacija mogu se izvoditi unutar transakcija kako bi se garantovala atomičnost, doslednost i izolacija podataka. Ova široka podrška omogućava programerima bezbedno upravljanje podacima u distribuiranim okruženjima, čime se osigurava doslednost i integritet podataka. [11], [12]

3.3.3. Ograničenja distribuiranih transakcija

Transakcije u MongoDB-u su čvrsto povezane sa sesijom. Važno je napomenuti da u svakom trenutku moguće imati najviše jednu otvorenu transakciju po sesiji. Prilikom korišćenja drajvera, bitno je da svaka operacija unutar transakcije bude povezana sa sesijom.

Svaka transakcija, koja se sprovodi u okviru sesije, obezbeđuje ključne karakteristike kao što su atomičnost, doslednost i izolacija podataka. Atomskost garantuje da se sve operacije u transakciji izvršavaju ili nijedna, čime se osigurava doslednost podataka. Doslednost znači da transakcija prelazi iz jednog doslednog stanja u drugo, dok izolacija obezbeđuje da rezultati transakcije ne budu vidljivi drugim transakcijama pre nego što se završi.

Ako sesija završi, a ima otvorenu transakciju, MongoDB će automatski poništiti tu transakciju kako bi očuvao integritet podataka. Sesije, inicijalizovane pomoću drajvera, igraju ključnu ulogu u upravljanju transakcijama, omogućavajući praćenje stanja transakcije, uključujući početak, potvrdu ili poništavanje.

3.3.4. Konzistentnost operacija čitanja i pisanja

Podešavanja čitanja

Operacije u transakciji koriste read preference na nivou transakcije.

Pomoću drajvera, možete postaviti read preference na nivou transakcije prilikom početka transakcije:

- Ako read preference na nivou transakcije nije postavljena, transakcija koristi read preference na nivou sesije.
- Ako nisu postavljene ni read preference na nivou transakcije ni na nivou sesije, transakcija koristi read preference na nivou klijenta. Podrazumevano, read preference na nivou klijenta je postavljen na *primary*.
- Distribuirane transakcije koje sadrže čitanje moraju koristiti read preference primary. Sve operacije u datoj transakciji moraju biti usmerene ka istom članu baze podataka.

Read concern – izolacija operacije čitanja

Operacije unutar transakcije koriste read concern na nivou transakcije. To znači da će bilo koji read concern postavljen na nivou kolekcije i baze podataka biti zanemaren unutar same transakcije.

Moguće je postaviti read concern na nivou transakcije prilikom početka same transakcije.

- Ukoliko nije postavljen read concern na nivou transakcije, on se podrazumevano preuzima sa read concern-om na nivou sesije.
- Ako nisu postavljeni ni read concern na nivou transakcije ni na nivou sesije, read concern na nivou transakcije će se podrazumevano preuzeti sa read concern-om na nivou klijenta. Po defaultu, read concern na nivou klijenta za čitanje od primarnog čvora je "local”

Transakcije podržavaju sledeće nivoe read concern-a:

local

- Read concern *local* vraća najnovije dostupne podatke sa čvora, ali ti podaci mogu biti poništeni.
- Za transakcije na sharded klasteru, *local* read concern ne može garantovati da su podaci iz istog snapshot-a preko različitih shardova. Ako je potrebna izolacija snapshot-a, koristi se *snapshot* read concern.
- Moguće je kreirati kolekcije i indekse unutar transakcije. Ako se eksplicitno kreira kolekcija ili indeks, transakcija mora da koristi read concern *local*. Ako se kolekcija implicitno kreira, može se koristiti bilo koji od read concern-a dostupnih za transakcije.

majority

- Ukoliko se transakcija potvrdi sa write concern *majority*, read concern *majority* vraća podatke koji su potvrđeni od strane većine članova seta replika i ne mogu se poništiti. Inače, read concern *majority* ne pruža garancije da operacije čitanja čitaju podatke većinski potvrđenih podataka.
- Za transakcije na sharded klasteru, read concern *majority* ne može garantovati da su podaci iz istog pregleda snimka preko različitih shardova. Ako je potrebna izolacija snimka, koristite read concern *snapsho*t.

snapshot

- Read concern *snapshot* vraća podatke sa snimka većinski potvrđenih podataka, ukoliko se transakcija potvrdi sa write concern *majority*.
- Ukoliko transakcija ne koristi write concern *majority* prilikom potvrde, *snapshot* read concern ne garantuje da operacije čitanja koriste snimak većinski potvrđenih podataka.
- Za transakcije na sharded klasterima, *snapshot* pogled podataka je sinhronizovan između shardova. [11]

Write concern – izolacija operacije upisa

Transakcije koriste write concern na nivou transakcije kako bi potvrdile operacije upisa. Operacije upisa unutar transakcija moraju se izvršiti bez eksplicitne specifikacije write concerna i koriste podrazumevani write concern. Prilikom potvrde, upisi se zatim potvrđuju koristeći write concern na nivou transakcije.

Može se postaviti write concern na nivou transakcije prilikom početka same transakcije:

- Ako write concern na nivou transakcije nije postavljen, podrazumevano će se koristiti write concern na nivou sesije za potvrdu.
- Ako nisu postavljeni write concern ni na nivou transakcije ni na nivou sesije, write concern na nivou transakcije će se podrazumevano koristiti prema write concern-u na nivou klijenta:
    - *w: "majority"* u MongoDB 5.0 i novijim verzijama, sa razlikama za postavke koje sadrže arbitere
    - w: 1

Transakcije podržavaju sve vrednosti za write concern w, uključujući:

w: 1

- Write concern w: 1 pruža potvrdu nakon što je upis primenjen na primarni čvor. Kada se potvrđuje sa w: 1, *majority* read concern na nivou transakcije ne garantuje da operacije čitanja u transakciji čitaju podatke većinskog potvrđenog upisa.
- Kada potvrđujete sa w: 1 write concern, *snapshot* read concern na nivou transakcije ne garantuje da operacije čitanja u transakciji koriste snimak većinskog potvrđenog upisa

w: "majority"

- Write concern *w: "majority"* pruža potvrdu nakon što je upis primenjen na većinu (M) od glasačkih članova, što znači da je upis primenjen na primarni čvor i (M-1) glasačkih sekundarnih čvorova.
- Kada se potvrđuje sa *w: "majority"* write concern, *majority* read concern na nivou transakcije garantuje da su operacije čitanja vršene nad podacima većinskog potvrđenog upisa. Za transakcije na shard klasterima, ovaj prikaz većinskog potvrđenog upisa nije sinhronizovan između shardova.
- Kada se potvrđuje sa *w: "majority"* write concern, *snapshot* read concern na nivou transakcije garantuje da su operacije čitanja vršene iz sinhronizovanog snimka većinskog potvrđenog upisa. [11]

3.3.5. Primer distribuirane transkacije

Da bi implementirali distribuirane transakcije u MongoDB-u, potrebno je koristiti transakcijski API, koji omogućava izvođenje transakcija sa više operacija nad jednim ili više MongoDB čvorova. Na slici 32 je prikazan primer distribuirane transakcije pomoću MongoDB transakcijskog API-ja koristeći MongoDB Node.js.

```
const { MongoClient } = require('mongodb');
async function performDistributedTransaction() {
  const client = new MongoClient('mongodb://localhost:27017',{ useNewUrlParser: true, useUnifiedTopology: true });
  try {
    await client.connect();

    // Početak transakcije
    const session = client.startSession();
    session.startTransaction();

    try {
      const database = client.db('testDB');
      const users = database.collection('users');
      const students = database.collection('students');

      // Operacije unutar transakcije
      await users.updateOne({ _id: 1 }, { $set: { email: 'newemail@gmail.com' } }, { session });
      await students.insertOne({ _id: 1, name: 'John Doe', email: 'john@gmail.com' }, { session });

      // Zavrsetak transakcije
      await session.commitTransaction();
      console.log('Transakcija uspešno završena.');
    } catch (error) {
      // Ako se dogodi greška, poništite transakciju
      await session.abortTransaction();
      console.error('Transakcija nije uspela:', error);
    } finally {
      // Zatvaranje sesije
      session.endSession();
    }
  } finally {
    // Zatvaranje MongoDB konekcije
    await client.close();
  }
}
performDistributedTransaction();
```

Slika 32. Distribuirana transakcija

U ovom primeru, transakcija obuhvata ažuriranje dokumenta u kolekciji users i umetanje dokumenta u kolekciju students. Ukoliko neka od ovih operacija ne uspe, transakcija će biti poništena, inače će biti uspešno potvrđena.

4. Zaključak

MongoDB, kao distribuirana baza podataka, se izdvaja kao ključno rešenje u savremenom svetu informacija. U ovom kontekstu, distribuirane baze podataka postaju ključna komponenta, odgovarajući na izazove savremenog društva koje generiše i manipuliše enormnim količinama podataka. Raznolikost podataka, od struktuiranih do nestruktuiranih, i potreba za očuvanjem integriteta podataka postavljaju zahtev za skalabilnim i fleksibilnim rešenjima, a MongoDB ispunjava te zahteve na impresivan način.

MongoDB se ističe kroz svoje ključne karakteristike, kao što su replikacija i sharding. Replikacija omogućava visoku dostupnost i otpornost na kvarove, koristeći članove skupa replika koji automatski prelaze na rezervni čvor u slučaju problema. Ovo poboljšava operacije čitanja i osigurava vidljivost podataka. S druge strane, sharding omogućava horizontalno skaliranje baze podataka, efikasno raspodeljujući podatke na više servera.

U kontekstu distribuiranih transakcija, MongoDB pruža rešenja koja adresiraju izazove atomičnosti, konzistentnosti i izolacije. Međutim, ograničenja distribuiranih transakcija su neizbežna, a pažljivo upravljanje i implementacija su neophodni kako bi se očuvala konzistentnost operacija čitanja i pisanja.

Na kraju bitno je reći da se MongoDB profilira kao moćan alat u domenu distribuiranih baza podataka, pružajući organizacijama efikasna i skalabilna rešenja za upravljanje ogromnim količinama podataka. Međutim, važno je razumeti i upravljati izazovima koji prate ovu tehnologiju, čime se obezbeđuje optimalna iskoristivost njenih prednosti u savremenom digitalnom okruženju.

5. Literatura

[1] https://www.mongodb.com/basics/distributed-database
[2] https://www.mongodb.com/docs/manual/sharding/
[3] https://www.percona.com/blog/when-should-i-enable-mongodb-sharding/
[4] https://www.mongodb.com/docs/manual/core/sharded-cluster-components/
[5] https://www.mongodb.com/docs/manual/core/sharded-cluster-shards/
[6] https://www.mongodb.com/docs/manual/core/sharded-cluster-config-servers/
[7] https://www.mongodb.com/docs/manual/core/sharded-cluster-query-router/
[8] https://www.mongodb.com/docs/manual/core/sharding-shard-key/
[9] https://www.percona.com/blog/when-should-i-enable-mongodb-sharding/
[10] https://www.mongodb.com/docs/manual/replication/
[11] https://www.mongodb.com/docs/manual/core/transactions/
[12] https://www.mongodb.com/docs/manual/core/transactions-operations/#std-label-transactions-operations-ref