



**Univerzitet u Nišu
Elektronski fakultet
Katedra za računarstvo**

Seminarski rad

MongoDB kao distribuirana baza podataka

**Mentor:
Doc. dr. Aleksandar Stanimirović**

**Student:
Emilija Bićanin 1474**

Niš, 2023

Sadržaj

Sadržaj.....	2
1. Uvod	3
2. MongoDB-Distribuirana arhitektura	4
2.1 Horizontalno skaliranje podataka kod MongoDB baze-Sharding	5
2.1.1 Sharded cluster	5
2.1.2 Particionisanje podataka u chunk-ove	6
2.1.3 Ključevi za particionisanje podataka (Shard Keys).....	7
2.1.4 Odabir ključa za particionisanje podataka.....	7
2.1.5 Strategije sharding-a.....	8
2.1.6 Sharding zone	10
2.1.7 Prednosti sharding-a.....	10
2.2 Replikacija kod MongoDB-a.....	11
2.2.1 Članovi seta replike.....	11
2.2.2 Sinhronizacija podataka u setu replike	12
2.2.3 Arhitektura seta replike	12
2.2.4 Operacije čitanja u setovima replika	13
2.2.5 Operacije upisa u setovima replika.....	14
2.3 Distribuirane transakcije	14
2.3.1 Atomičnost distribuiranih transakcija.....	15
2.3.2 Dozvoljene operacije kod distribuiranih transakcija	15
2.3.3 Ograničenja distribuiranih transakcija.....	15
2.4 Konzistentnost operacija čitanja i pisanje kod MongoDB-a	16
2.4.1 Izolacija operacije čitanja (Read Concern).....	16
2.4.2 Potvrđivanje operacije upisa (Write Concern)	18
Kauzalna konzistentnost kod MongoDB-a.....	19
3. Praktična demonstracija distribuiranih koncepata u MongoDB-U.....	20
3.1 Setovi replika	20
3.1.1 Konfiguracija seta replike	20
3.1.2 Operacije upisa nad setom replike	21
3.2 MongoDB Sharding	21
3.2.1 Kreiranje šard klastera i šardovanje kolekcije	21
3.2.2 Šard zone	27
3.4 Distribuirane transakcije.....	30
4. Zaključak	31
5. Literatura	32

1. Uvod

Distribuirane baze podataka imaju veliki značaj u savremenom svetu informacionih tehnologija. U suštini, distribuirana baza podataka sastoji se od više fizički odvojenih baza podataka koje se nalaze na različitim računarima ili čvorovima u mreži. Ove baze podataka su povezane i sinhronizovane kako bi omogućile efikasno deljenje podataka između različitih lokacija.

Postoji nekoliko ključnih razloga za korišćenje distribuiranih baza podataka:

- **Visoka dostupnost:** Distribuirane baze podataka omogućavaju replikaciju podataka na više lokacija. To znači da, ako jedan čvor ili server doživi kvar, podaci su i dalje dostupni i operativni. Ova visoka dostupnost je posebno važna za kritične sisteme gde je kontinuitet poslovanja od ključnog značaja.
- **Skalabilnost:** Distribuirane baze podataka pružaju mogućnost skaliranja horizontalno i vertikalno. Horizontalno skaliranje podrazumeva dodavanje novih čvorova u mrežu kako bi se povećao kapacitet skladištenja i obrade podataka. Vertikalno skaliranje podrazumeva povećanje resursa (npr. procesora, memorije) na postojećim čvorovima. Ova skalabilnost omogućava efikasno upravljanje rastućim količinama podataka.
- **Brza obrada podataka:** Distribuirane baze podataka mogu omogućiti paralelno izvršavanje upita i transakcija na različitim čvorovima. To rezultira bržom obradom podataka u poređenju sa centralizovanim bazama podataka, gde su sve transakcije ograničene na jedan server.
- **Geografska distribucija:** Distribuirane baze podataka omogućavaju geografsku distribuciju podataka na različitim lokacijama. To je korisno kada organizacije imaju različite kancelarije ili filijale na više geografskih lokacija. Ovakva distribucija omogućava lokalni pristup podacima, što može smanjiti latenciju i poboljšati performanse.
- **Otpornost na kvarove:** U distribuiranim bazama podataka, podaci se često repliciraju na više lokacija. To pruža visok stepen otpornosti na kvarove, jer čak i ako jedan čvor ili celokupna lokacija dožive kvar, podaci su i dalje dostupni sa drugih replika.

MongoDB pruža moćne mehanizme za distribuirane baze podataka ko što su replikacija i horizontalno skaliranje podataka, koji omogućavaju visoku dostupnost, skalabilnost i performanse u distribuiranim okruženjima. Takođe pruža i mogućnost izvršavanja ACID transakcija u distribuiranim okruženjima. U seminarskom radu će biti opisani svi ovi mehanizmi i njihov značaj.

2. MongoDB-Distribuirana arhitektura

MongoDB predstavlja distribuiranu bazu podataka i kao takva podržava mehanizme poput replikacije, horizontalnog skaliranja, distribuiranih upita i transakcija. Svi ovi mehanizmi su sakriveni od krajnjeg korisnika i on bazu vidi kao jedinstvenu celinu.

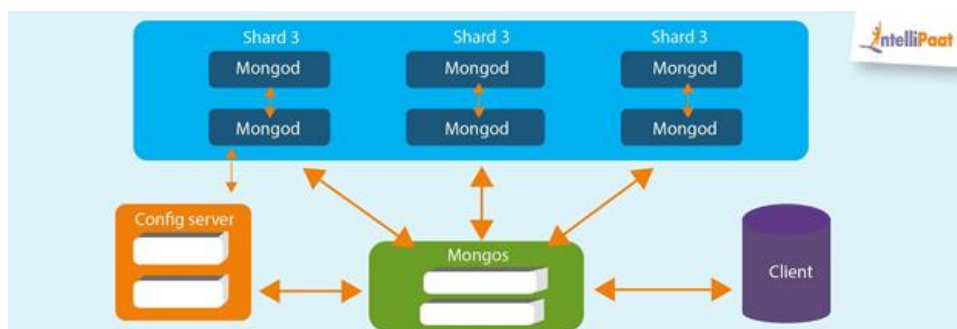
MongoDB replikacija omogućava replikaciju podataka na više čvorova kako bi se postigla visoka dostupnost i otpornost na kvarove. U tipičnoj replikaciji MongoDB-a, postoji primarni čvor koji prima sve upite i zapisuje promene u podacima, dok se kopije podataka održavaju na sekundarnim čvorovima. Replicirani podaci se automatski sinhronizuju sa primarnog čvora na sekundarne čvorove, pružajući tako visoku dostupnost čak i u slučaju kvara primarnog čvora. Kada primarni čvor postane nedostupan, jedan od sekundarnih čvorova preuzima ulogu primarnog čvora.

MongoDB Sharding omogućava horizontalno skaliranje podataka tako što raspoređuje podatke preko više čvorova u različite delove, nazvane shardovi. Svaki shard može biti samostalna MongoDB baza podataka ili skup čvorova. Sharding se postiže podešavanjem ključa za podele (shard key) koji određuje na osnovu kojeg kriterijuma se podaci raspoređuju na shardove. Ovo omogućava ravnomerno raspoređivanje opterećenja i bržu obradu podataka jer se upiti mogu paralelno izvršavati na različitim shardovima. Sharding takođe omogućava elastičnost, jer se novi shardovi mogu dodavati kako bi se povećao kapacitet sistema.

MongoDB takođe podržava kombinaciju replikacije i shardinga, koja omogućava postizanje visoke dostupnosti, skalabilnosti i otpornosti na kvarove u distribuiranim okruženjima. Ova kombinacija se naziva "MongoDB Replication & Sharding Architecture".

MongoDB ima i podršku za distribuirane transakcije koje se mogu izvršavati u sistemu sa replikama i šardovima i na većem broju dokumenata.

Uz distribuciju podataka, MongoDB pruža i druge napredne mogućnosti kao što su automatsko ispitivanje, balansiranje opterećenja, automatsko otkrivanje i oporavak od kvarova, kao i mogućnosti za replikaciju i šifriranje podataka.



Slika 1. MongoDB-Distribuirana arhitektura

2.1 Horizontalno skaliranje podataka kod MongoDB baze-*Sharding*

Horizontalno skaliranje uključuje podelu sistemskog skupa podataka 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 se nosi sa podskupom ukupnog radnog opterećenja, potencijalno pružajući bolju efikasnost od jednog servera velike brzine i velikog kapaciteta. Proširenje kapaciteta implementacije zahteva samo dodavanje dodatnih servera po potrebi, što može biti niži ukupni trošak od vrhunskog hardvera za jednu mašinu. Kompromis je povećana složenost infrastrukture i održavanja sistema.

MongoDB ima podršku za horizontalno skaliranje sistema koje se ovde naziva **sharding**. MongoDB skalira podatke na nivou kolekcije.

Jedan od ciljeva sharding-a je da klaster od nekoliko ili čak stotina čvorova (šardova) izgleda kao jedna mašina iz ugla aplikacije. Da bi sakrio detalje skaliranja od aplikacije, MongoDB pokreće jedan ili više procesa rutiranja koji se nazivaju **mongos** instance. Mongos predstavlja ruter i njemu se prosleđuju svi upiti a on uz pomoć konfiguracionih servera utvrđuje kojim šardovima treba proslediti upite.

Prilikom horizontalnog skaliranja, MongoDB deli podatke u **čankove** (engl. chunks). Na osnovu **ključa za particionisanje**. Svaki čank predstavlja podskup dokumenata čije polje/polja koja predstavljaju ključ imaju vrednosti u određenom opsegu.

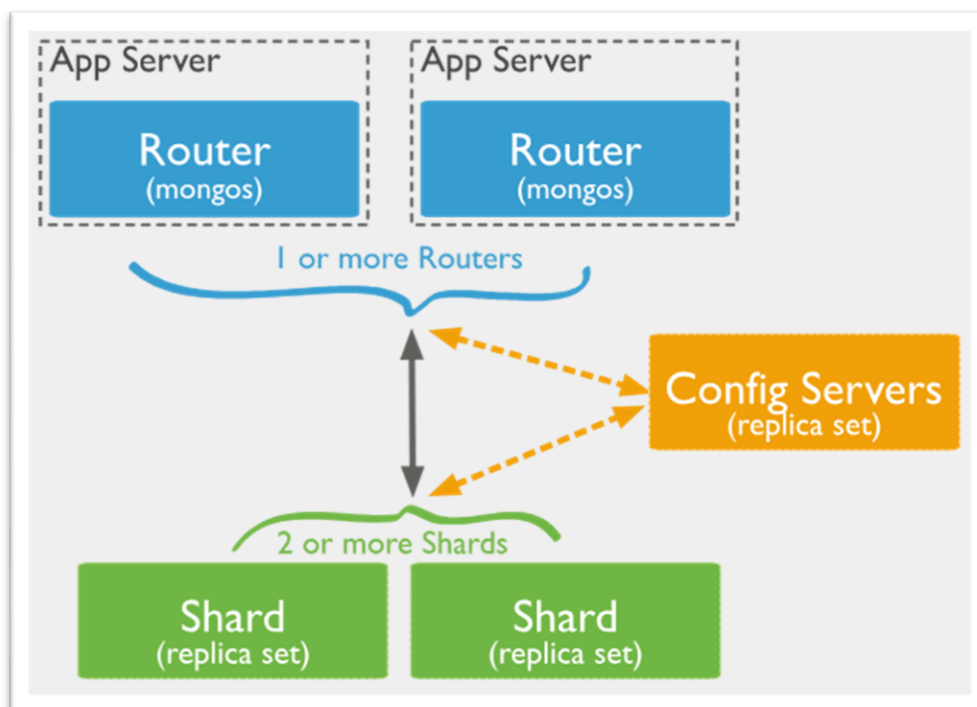
2.1.1 Sharded cluster

Sharded cluster kod MongoDB-ja predstavlja celinu čvorova na kojima su raspoređeni podaci iz kolekcija baze. Sharded cluster se sastoji iz sledećih komponenti:

- **shard**-Jedan shard predstavlja mongoDB instancu ili replica set koji sadrži deo ukupnih podataka na sharded 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.

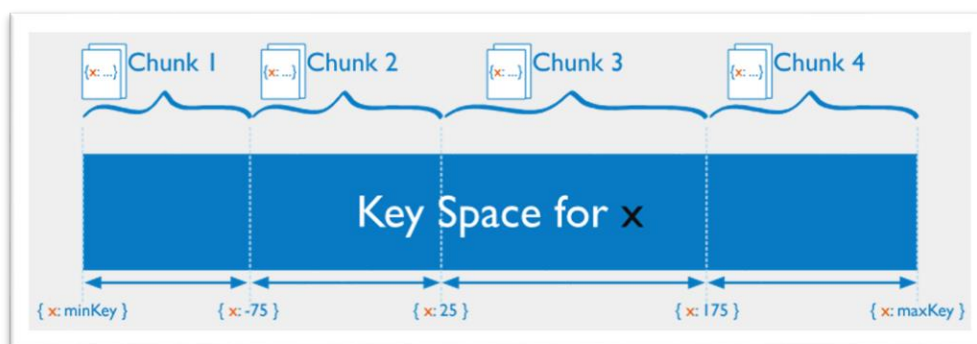
Baza podataka može sadržati i kolekcije koje su distribuirane po šardovima, ali i one koje nisu. Neparticionisane kolekcije se uvek skladište na **primarnom šardu**. Svaka baza podataka poseduje svoj primarni šard.



Slika 2. Sharded cluster

2.1.2 Partitionisanje podataka u *chunk*-ove

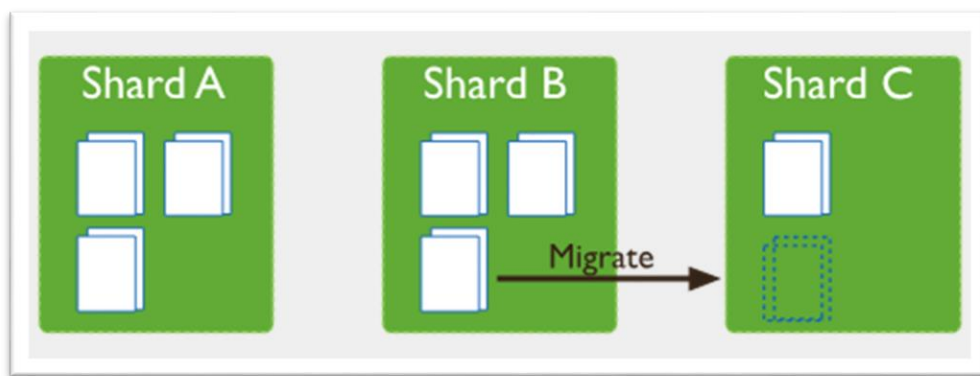
MongoDB koristi ključ za partitionisanje kako bi podelio podatke između šardova. Podaci se dele u delove koji se nazivaju čankovi. Jedan chunk obuhvata određeni opseg partitionisanih podataka. U odnosu na vrednost ključa za partitionisanje, svaki chunk ima svoju donju i gornju granicu. Na sledećoj slici prikazan je princip po kome se podaci dele u chunk-ove.



Slika 3. Podela podataka u chunk-ove

Podrazumevano, veličina jednog chunk-a je 128 megabajta. Ova veličina se promeniti u konfiguraciji.

Proces koji upravlja preraspodelom podataka u chunk-ove i migracijom podataka kod MongoDB-ja naziva se **balancer**. Kada razlika između najvećeg i najmanjeg šarda u količini podataka koje čuvaju dostigne određenu granicu, balancer vrši migraciju podataka kako bi obezbedio ravnomerniju distribuciju.



Slika 4. Uloga balancer-a

2.1.3 Ključevi za particionisanje podataka (*Shard Keys*)

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.

Pre mongo verzije 4.4, svi dokumenti iz kolekcije morali su sadržati polja od kojih se ključ za particionisanje sastoji. Međutim, počev od verzije 4.4, to više nije neophodno. Nedostajuća polja se tretiraju kao polja sa *null* vrednostima.

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.

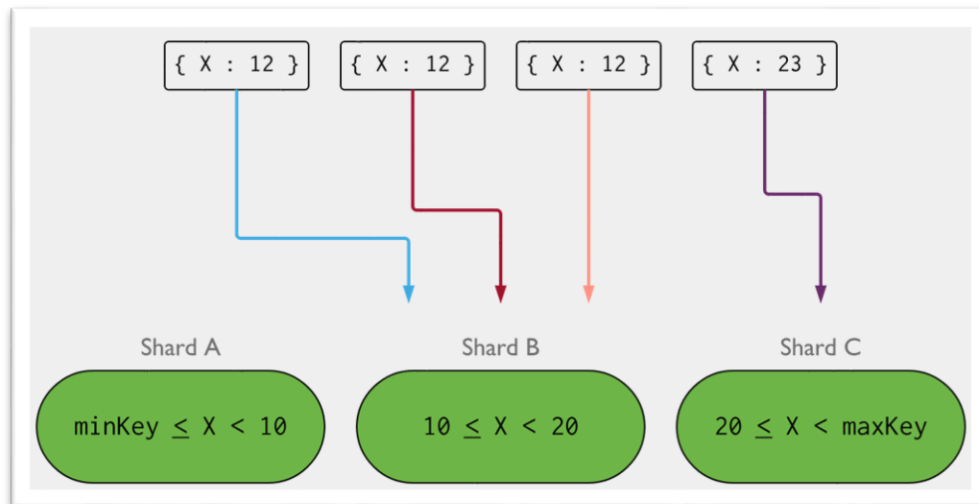
2.1.4 Odabir ključa za particionisanje podataka

Izbor ključa šarda utiče na performanse, efikasnost i skalabilnost podeljenog klastera. Klaster sa najboljim mogućim hardverom i infrastrukturuom može biti usko grlo kao posledica lošeg izbora ključa.

Prilikom biranja ključa za particionisanje, u obzir je neophodno uzeti sledeće činjenice:

- **Kardinalnost ključa-** Kardinalnost ključa ograničava maksimalan broj čankova koji može biti kreiran. To znači da dokumenti sa istom vrednošću ključa za particionisanje mogu postojati isključivo u okviru istog čanka. Ključ sa malom kardinalnošću će rezultovati malim brojem glomaznih čankova, što će značajno smanjiti efikasnost horizontalnog skaliranja klastera. Zato je dobro da se po mogućstvu za ključ particionisanja biraju polja koja imaju veliku kardinalnost. Ipak, ukoliko polje po kojem želimo da particionišemo podatke ima malu kardinalnost, treba razmisliti o ključu nad većim brojem polja, od kojih će prvo polje biti željeno, a ostala bolja mogu biti ubačena kako bi se povećala ukupna kardinalnost ključa.

- **Učestalost sa kojom se različite vrednosti ključa pojavljuju-** Ključ sa visokom kardinalnošću obezbeđuje postojanje većeg broja čankova, ali ne garantuje da će podaci biti ravnomerno raspoređeni po čankovima. Kako bi osigurali ravnomernu raspodelu, treba obratiti pažnju na frekvencije različitih vrednosti ključa. Ukoliko većina dokumenata uzima vrednosti samo podskupa mogućih vrednosti ključa, onda će čankovi sa tim dokumentima biti glomazni i potencijalno predstavljati usko grlo sistema. Na sledećoj slici ilustrovano je kako veća frekvencija pojedinih vrednosti može da utiče na raspodelu podataka po čankovima.



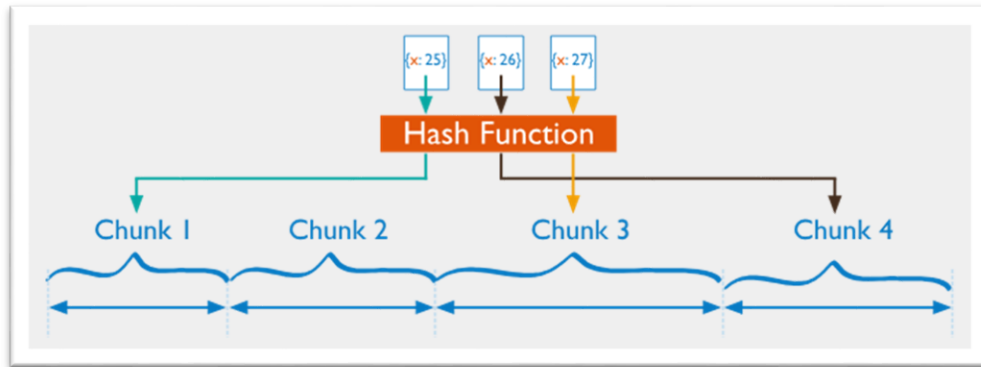
Slika 5. Uticaj frekvencije vrednosti ključa na particionisanje podataka

- **Da li vrednosti ključa monotono rastu-**Ključ za particionisanje sa vrednostima koje monotono rastu ili opadaju može da dovede do toga da se svi novokreirani dokumenti upisuju u jedan čank, koji onda može postati usko grlo. Ovo je posledica činjenice da svaki klaster poseduje čank sa gornjom granicom **maxKey** (maxKey predstavlja maksimalnu vrednost ključa) i jedan čank čija je donja granica **minKey** (minimalna vrednost ključa). Ukoliko vrednosti ključa monotono rastu, svaki novi dokument biće dodeljen čanku čija je gornja granica maxKey, odnosno čanku čija je donja granica minKey ukoliko vrednosti monotono opadaju. Taj čank će onda postati usko grlo za operacije kreiranja dokumenata.
- **Šablone upita-**Idealan ključ za particionisanje vrši ravnomernu distribuciju podataka ali i olakšava izvršavanje čestih upita. Zato prilikom odabira ključa, treba uzeti u obzir i česte obrasce pretrage podataka i da li ih potencijalni ključ podržava. Ukoliko upit sadrži ključ šarda, mongos ruter će odrediti relevantan šard i samo njemu poslati upit. Međutim, ukoliko upit ne sadrži ključ particionisanja, upit će biti prosleđen svim šardovima.
- **Ograničenja ključa-**U ranijim verzijama monga, veličina ključa je bila ograničena na 512 bajta. Počev od verzije 4.4, ovo ograničenje više ne postoji. U kolekciji mora postojati indeks čiji je prefiks šard ključ i koji je po polju/poljima ključa uređen po rastućem poretku. Ovaj indeks ne sme biti indeks na nizom podataka (*multikey* indeks), tekstualni indeks ili geoprostorni indeks.

2.1.5 Strategije *sharding*-a

MongoDB podržava dve osnovne strategije za particionisanje podataka:

1. **Heširano particionisanje**-Kod ove vrste particionisanja koristi se heširana vrednost ključa za particionisanje. Naime, opsezi koji se dodeljuju čankovima računaju u odnosu na heš vrednost a ne na plaintext vrednost ključa. Pripadnost dokumenta nekom čanku određuje se računanjem heš funkcije vrednosti ključa tog dokumenta. Na sledećoj slici je ilustrovan ovaj pristup.

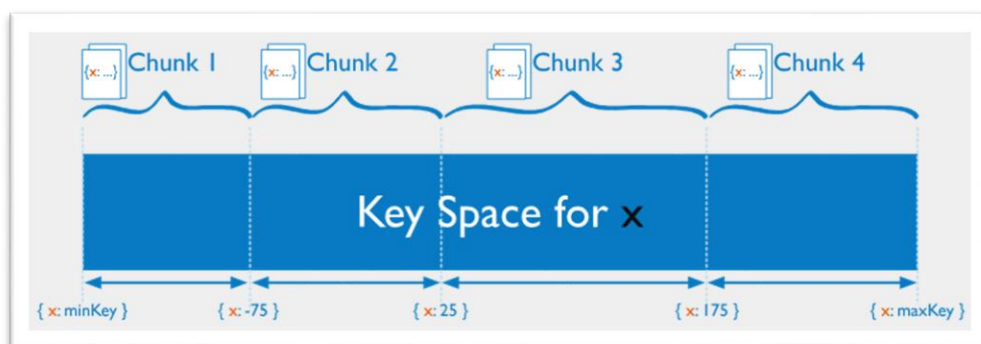


Slika 6. Heširano particionisanje

Pogodnosti heš funkcija su te što međusobno bliske vrednosti gotovo nikad neće biti ponovo bliske onda kada se heš funkcija primeni na njih. To znači da će distribuiranje podataka po heširanom ključu ravnomernije rasporediti podatke jer čak iako imamo veliki broj vrednosti ključa iz određenog opsega, one gotovo nikad neće biti raspoređene u istom čanku. Ovakva raspodela pogotovo je korisna kada imamo vrednosti ključa koje monotono rastu. Ona omogućava da umesto da se novi dokumenti uvek preslikaju u isti čank, sada mnogo ravnomernije raspodele.

Međutim, nedostatak ovakvog pristupa jesu slabije performanse kod upita opsega po ključu particionisanja. Naime, pošto će susedne vrednosti ključa biti uglavnom raspoređene na različitim šardovima, onda će i izvršenje upita biti mnogo sporije.

2. **Particionisanje po opsezima**-Za razliku od heširanog particionisanja, ovde se pripadnost čanku utvrđuje direktno u odnosu na vrednost ključa za particionisanje. Svakom čanku se dodeljuje opseg u zavisnosti od vrednosti ključa.



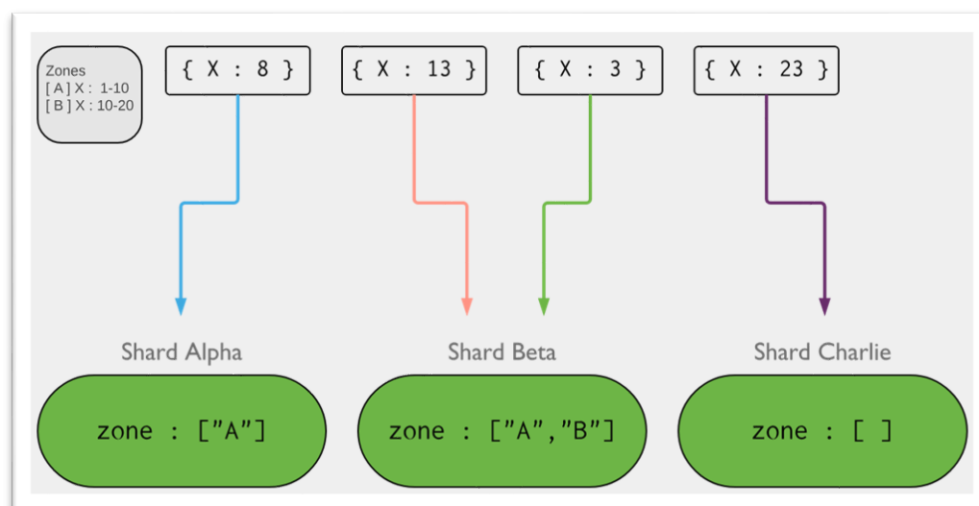
Slika 7. Particionisanje po opsezima

Kod ove vrste partitionisanja će dokumenti sa međusobno bliskim vrednostima ključa za particionisanje sa velikom verovatnoćom biti dodeljeni istom čanku.

2.1.6 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. Jedan šard može sadržati veći broj zona. Balanser vrši migraciju podataka sa jednog šarda koji pripadaju nekoj zoni isključivo u šardove kojima je dodeljena ta zona.

Sledeća slika ilustruje šardovani klaster koji se sastoji iz 3 šarda (Alpha, Beta i Charlie) i 2 zone (zona A i zona B). Zona A obuhvata dokumente kod kojih se ključ kreće u opsegu od 1 do 10, a zona B dokumente u opsegu od 10 do 20. Šardovima Alpha i Beta dodeljena je zona A, šardu Beta dodeljena je i zona B a šard Charlie nema dodeljenu zonu. To znači da će dokumenti sa ključevima u opsegu 1-10 moći da se nađu na šardovima Alpha i Beta, dokumenti sa ključevima u opsegu 10-20 će moći da se nađu samo na šardu Beta, a za sve ostale dokumente ne postoje ograničenja i oni se mogu naći na bilo kom šardu.



Slika 8. Podela po zonama

Neki uobičajeni slučajevi upotrebe koji mogu iskoristiti zone su:

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

2.1.7 Prednosti *sharding-a*

Sharding kod MongoDB-ja pruža sledeće pogodnosti:

- **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**-Pošto 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 (High Availability)**-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.

2.2 Replikacija kod MongoDB-a

Replikacija obezbeđuje redundantnost i povećava dostupnost podataka. Sa više kopija podataka na različitim serverima baze podataka, replikacija obezbeđuje nivo tolerancije grešaka na gubitak jednog servera baze podataka.

U nekim slučajevima, replikacija može da obezbedi povećan kapacitet čitanja jer klijenti mogu da šalju operacije čitanja na različite servere. Održavanje kopija podataka u različitim centrima podataka može povećati lokalizaciju podataka i dostupnost za distribuirane aplikacije. Takođe, mogu se održavati dodatne kopije za namenske svrhe, kao što su oporavak od katastrofe, izveštavanje ili pravljenje rezervnih kopija.

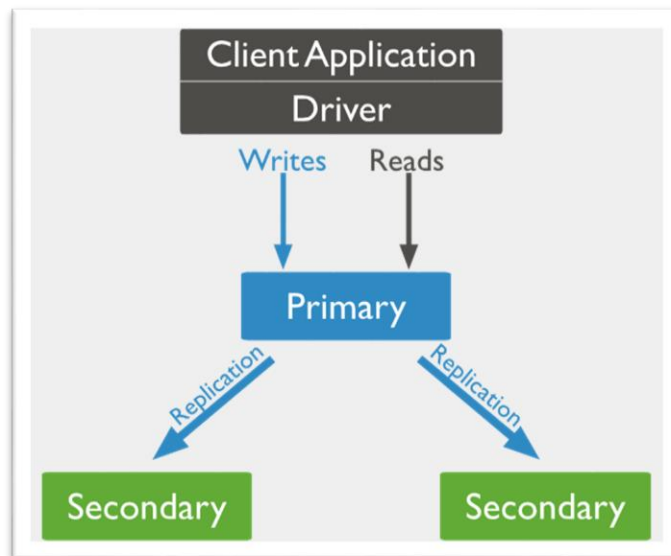
2.2.1 Članovi seta replike

Kod mongo baze postoje setovi replika. Set replike predstavlja skup *mongod* instanci koje čuvaju iste podatke. Set replika može opciono sadržati i jedan arbitražni čvor. Kod instanci sa podacima, samo je jedna jedina instanca označena kao primarni čvor, dok su sve ostale instance sekundarni čvorovi. Ukoliko dođe do otkaza primara, biće pokrenut proces selekcije novog primara među dostupnim sekundarima. U setu replike može postojati i arbitražni čvor, koji ne sadrži repliku podataka, ali ima pravo glasanja prilikom procesa biranja novog primara.

Za minimalnu konfiguraciju seta replike, preporučuje se set od makar tri člana: 1 primarnog čvora i 2 sekundarna. Jedan replika set može imati do 50 članova, ali samo 7 mogu učestvovati u glasanju novog primara.

Primarni čvor je zadužen za sve operacije upisa. On prima operacije upisa, izvršava ih i upisuje informacije o izvršenim operacijama u *oplog (operation log)* fajlu. Sekundarne replike dobijaju kopiju oplog-a i na osnovu njega izvršavaju nad svojim podacima iste operacije koje je izvršio primar.

Svi članovi seta replike mogu primiti i izvršavati *read* operacije. Podrazumevano se sve operacije čitanja prosleđuju primaru, ali se to može izmeniti na nivou svake operacije čitanja.



Slika 9. Replika set

2.2.2 Sinhronizacija podataka u setu replike

Kako bi sekundarne replike u skupu sadržale relevantne podatke, one moraju međusobno da komuniciraju i razmenjuju podatke. Kod MongoDB-a postoje dve osnovne forme sinhronizacije: **inicijalna sinhronizacija** pri kojoj se novi članovi replike ispunjavaju celim skupovima podataka i **replikacija** koja podrazumeva samo prosleđivanje promena koje su se u međuvremenu dogodile kako bi replike ažurirale svoje podatke.

Replike dobijaju informacije o operacijama koje moraju da izvrše nad svojim podacima u obliku oplog-a. Izvor replikacije od kojeg će sekundarna replika dobiti oplog može biti neka druga sekundarna replika ili striktno primarna replika. Izvor replikacije se navodi u konfiguraciji mongo instance u *chaining* opciji.

Ukoliko se odobri chaining opcija, to znači da će za izvor replikacije biti dozvoljeno da to bude bilo sekundarna bilo primarna replika. U tom slučaju, da bi se replika odabrala kao izvor replikacije za određenu repliku, ona mora imati noviji oplog od nje. Takođe, taj oplog mora biti najviše 30 sekundi stariji od najnovijeg oplog-a na primarnoj replici.

2.2.3 Arhitektura seta replike

Arhitektura seta replike utiče na kapacitet i mogućnosti seta. Preporučuje se da set ima najmanje tri člana-2 sekundarna i 1 primarni. Set replike može imati maksimalno 50 člana, od kojih maksimalno 7 mogu biti članovi sa pravom glasa pri izboru novog primara.

Poželjno je da broj replika koje imaju pravo glasa bude neparan, kako bi se izbegli nerešeni izbori. Ukoliko je broj članova paran, preporučuje se da se doda još jedna sekundarna replika sa podacima ili ,ukoliko je to nemoguće, arbitražni član.

Prilikom odabira broja članova replika, treba razmisliti o **otpornosti na greške** (fault tolerance). Otpornost na greške podrazumeva broj replika koje mogu postati nedostupne a da sistem

još uvek poseduje dovoljan broj replika da održi izbore i odabere novog primara. Ona predstavlja razliku između ukupnog broja čvorova i majority parametra koji određuje koliko najmanje čvorova mora da glasa za određeni čvor da bi on bio odabran kao primar. Zato bi dodavanje novih čvorova u set moglo da poveća toleranciju na greške.

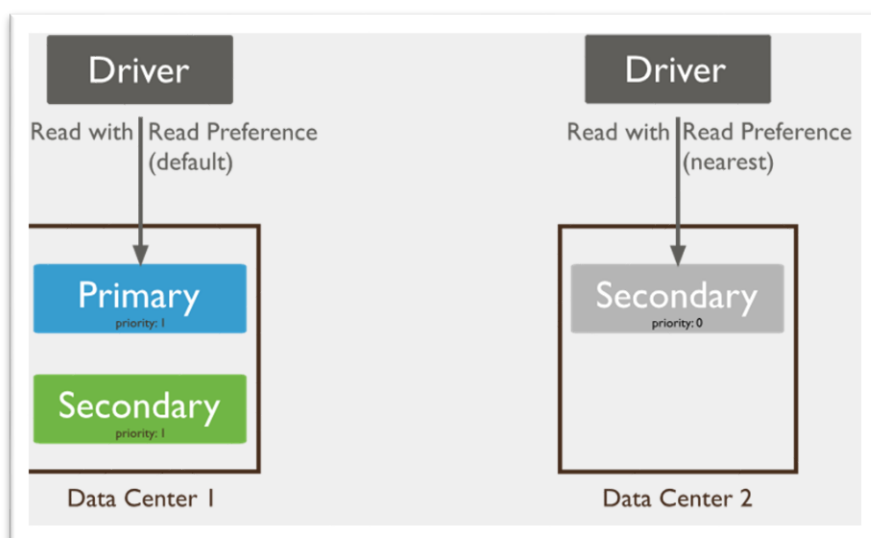
Poželjno je da članovi replike budu geografski distribuirani po različitim *data centre*-ima.

2.2.4 Operacije čitanja u setovima replika

U setovima replika sve operacije upisa izvršava primarna replika. Operacije čitanja se mogu izvršiti sa bilo koje replike, ali se podrazumevano i sve operacije čitanja prosleđuju samo primarnoj replici. Međutim, klijenti mogu da odaberu **preferencu čitanja**, odnosno da utiču na to kome se prosleđuje operacija čitanja. Preferenca čitanja može imati sledeće vrednosti:

- **Primary**-označava da se sve operacije čitanja prosleđuju primarnoj replici,
- **PrimaryPreferred**-sve operacije čitanja se prosleđuju primarnoj replici, sem onda kada ona nije dostupna, kada se prosleđuju nekoj od sekundarnih replika,
- **Secondary**-sve operacije čitanja se prosleđuju nekom sekundarnom članu
- **SecondaryPreferred**-sve operacije čitanja se prosleđuju sekundarnoj replici, sem onda kada ona nije dostupna. Tada se operacije čitanja prosleđuju primarnoj replici.
- **Nearest**-operacije čitanja se prosleđuju najbližoj replici, odnosno replici sa najmanjom latentnosti mreže, bez obzira da li je u pitanju primarna ili sekundarna replika.

Razlozi za menjanje preference čitanja mogu biti sledeći: smanjenje latentnosti u sistemima koji se prostiru nad većim brojem centara podataka, poboljšanje propusnosti operacije čitanja i omogućavanje čitanja onda kada je proces selekcije novog primara u toku i nema dostupnog primara.

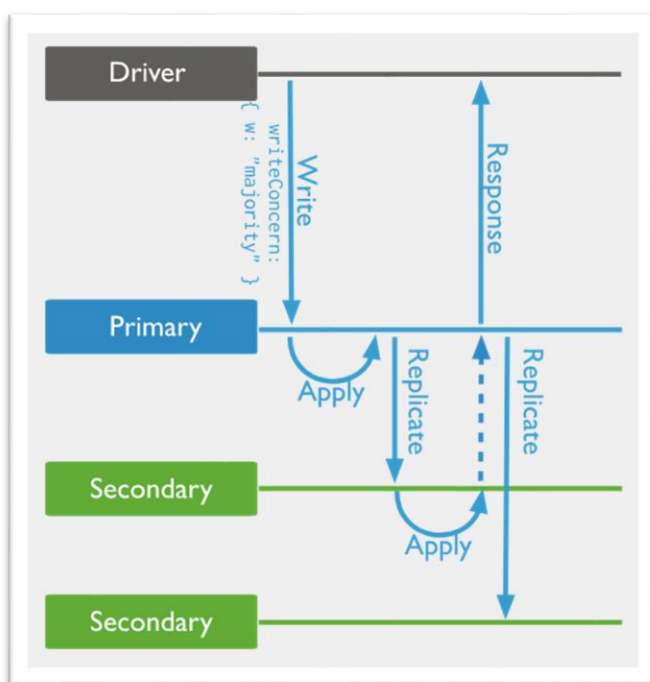


Slika 10. Različite preference čitanja

2.2.5 Operacije upisa u setovima replika

U setovima replika, operacije upisa može obaviti samo primarna replika. Primarna replika informacije o operacijama upisa koje je obavila upisuje u oplog koji se onda prosleđuje sekundarnim replikama kako bi one ažurirale svoje skupove podataka.

Kod MongoDB-a je moguće podesiti *writeConcern* parametar. On govori o tome koliko članova u skupu replika mora da potvrdi operaciju upisa da bi ona mogla da se izvrši. Vrednost ovog parametra može biti "majority" što onda znači da većina članova sa pravom glasa mora da odobri operaciju upisa pre nego što se ona izvrši. Vrednost *writeConcern* parametra može biti i numerička, ukoliko želimo da specificiramo tačan broj replika od kojih želimo potvrdu.



Slika 11. Operacija upisa kod skupa replike

2.3 Distribuirane transakcije

Transakcija predstavlja niz operacija baze podataka koji će uspeti samo ako je svaka operacija unutar transakcije ispravno izvršena. Transakcije su bile važna karakteristika relacionih baza podataka dugi niz godina, ali su do nedavno uglavnom izostajale iz baza podataka orijentisanih na dokumente. Priroda dokumenata orijentisanih baza podataka — gde jedan dokument može biti robusna, ugnežđena struktura, koja sadrži ugrađene dokumente i nizove, a ne samo jednostavne vrednosti — pojednostavljuje skladištenje povezanih podataka unutar jednog dokumenta. Kao takvo, izmena više dokumenata kao dela jedne logičke operacije često je nepotrebna, ograničavajući potrebu za transakcijama u mnogim aplikacijama.

Postoje, međutim, aplikacije za koje je potrebno pristupiti i modifikovati više dokumenata u jednoj operaciji sa zagarantovanim integritetom čak i sa bazama podataka orijentisanih na dokumente. MongoDB je uveo ACID transakcije u verziji 4.0 koje podržavaju transakcije nad više dokumenata u setovima replika (multi-document transactions). Od verzije 4.2 mongoDB uvodi podršku za distribuirane transakcije nad šardovanim klasterima.

2.3.1 Atomičnost distribuiranih transakcija

Distribuirane transakcije kod MongoDB-a su atomske, odnosno obezbeđuju “sve ili ništa” mehanizam izvršavanja. Kada se transakcija izvrši, sve operacije izvršene u njoj su sačuvane i vidljive celom sistemu. Dok se sve operacije u transakciji ne izvrše, sistemu neće biti vidljiv efekat nijedne od njih. Ukoliko se desi neka greška ili otkaz pri izvršenju transakcije i ona ne uspe da uspešno izvrši sve operacije u okviru nje, sve operacije koje se jesu izvršile biće poništene.

2.3.2 Dozvoljene operacije kod distribuiranih transakcija

Operacije koje je dozvoljeno navesti unutar transakcije kod MongoDB-a su:

- CRUD operacije nad postojećim kolekcijama. Pod CRUD operacijama kod MongoDB-a podrazumevaju se operacije agregacije, operacija *countDocuments* nad kolekcijom, operacija *distinct* (ali samo nad kolekcijama koje nisu šardovane), operacija *find*, operacije kreiranja, ažuriranja i brisanja kako pojedinačnih tako i grupe dokumenata.
- Počev od MongoDB verzije 4.4 dozvoljne su i operacije kreiranja indeksa i novih kolekcija u transakcijama.
- Informacione operacije poput *hello*, *buildInfo* i *connectionStatus* takođe su podržane u distribuiranim transakcijama. Međutim, one se ne mogu navesti kao prva operacija transakcije.

Za transakcije kod MongoDB-a važi i da se operacije unutar njih mogu izvršavati i nad različitim bazama podataka.

2.3.3 Ograničenja distribuiranih transakcija

Neke operacije nisu podržane u distribuiranim transakcijama kod MongoDB-a. To su:

- operacije *listCollections* i *listIndexes*,
- Administrativne operacije poput kreiranja korisnika, uloga..
- Operacija kreiranja kolekcije u distribuiranim šardovima onda kada transakcija podrazumeva i operaciju upisa podataka u neku drugu kolekciju. Nije moguć upis podataka u jednom šardu i kreiranje kolekcije na drugom šardu u istoj transakciji.

Ono što takođe nije dozvoljeno u transakcijama jeste upis/čitanje iz baza local, config i admin. Nije dozvoljen ni u upis u sistemskim kolekcijama. Nije dozvoljeno ni izvršiti upit explain koji vraća plan izvršenja nekog upita.

2.4 Konzistentnost operacija čitanja i pisanje kod MongoDB-a

2.4.1 Izolacija operacije čitanja (*Read Concern*)

Kontrola konzistentnosti i izolacije podataka postiže se u MongoDB-ju podešavanjem *readConcern* parametra. Podešavanjem ovog parametra, kao i *writeConcern* parametra (o kome će biti reči kasnije), pruža se mogućnost podešavanje nivoa konzistentnosti i dostupnosti po potrebi. Pri podešavanju ovih parametara, moramo napraviti određenu vrstu kompromisa, jer stroži zahtevi konzistentnosti imaju za posledicu sporije izvršenje operacija i samim tim smanjenu dostupnost sistema. Važi i obrnuto: povećanje dostupnosti sistema se postiže smanjivanjem kriterijuma konzistentnosti podataka.

Nivoi konzistentnosti koji se mogu obezbediti podešavanjem *readConcern* parametra su sledeći:

- Lokalni nivo ("local")-Ovde upit vraća podatke sa mongo instance bez ikakve 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. Ukoliko imamo set od tri replike: 2 sekundara i 1 primar i *readConcern* podešen na "local", neka je izvršena operacija upisivanja podataka W0. Neka pri tome važi sledeće: svi upisi pre W0 su uspešno izvršeni i poslednji upis podataka pre W0 je Wprev. Neka je dat sledeći timeline:

T0-Primar izvršava operaciju W0,

T1-Prva sekundarna replika čita oplog i izvršava operaciju W0,

T2-Druga sekundarna replika čita oplog i izvršava operaciju W0,

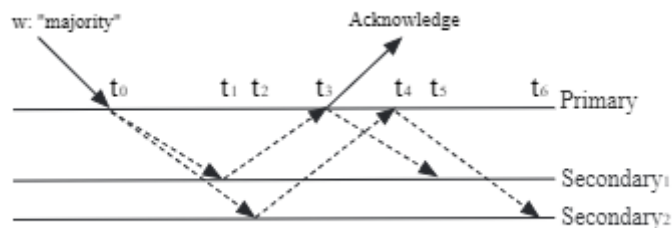
T3-Primar postaje svestan da je prva sekundarna replika uspešno izvršila operaciju W0 i šalje potvrdu klijentu,

T4- Primar postaje svestan da je druga sekundarna replika uspešno izvršila operaciju W0,

T5-Prva sekundarna replika dobija obaveštenje da je operacija W0 uspešno izvršena u većini čvorova u sistemu i to pamti u svom najnovijem snapshot-u,

T6-Druga sekundarna replika dobija obaveštenje da je operacija W0 uspešno izvršena u većini čvorova u sistemu i to pamti u svom najnovijem snapshot-u.

Na sledećoj slici prikazani su ovi događaji.



Slika 12. Timeline

U sledećoj tabeli data 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
Primar	Nakon T0	Podaci sadrže efekte upisa W0
Prvi sekundar	Pre T1	Podaci ne sadrže efekte upisa W0, već Wprev
Prvi sekundar	Nakon T1	Podaci sadrže efekte upisa W0
Drugi sekundar	Pre T2	Podaci ne sadrže efekte upisa W0, već Wprev
Drugi sekundar	Nakon T2	Podaci sadrže efekte upisa W0

Tabela1.

Kao što se vidi iz tabele, svaki član replike odmah pri izvršenju operacije W0 lokalno, smatra da je operacija validno izvršena i bez obzira na to da li će korisnik da zatraži 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 prethodnom nivou. Podaci se vraćaju sa instance bez traženja potvrde od ostalih instanci. Jedina razlika izmedju ove i opcije local jeste ponašanje kod distribuiranih (sharded) kolekcija. Tada available nivo može da vrati i takozvane "orphaned" dokumente, koji predstavljaju dokumente zaostale na šardu nakon neuspele migracije. Kod local nivoa ne postoji opasnost od vraćanja orphaned dokumenata.
- Nivo većine ("majority")-Upit vraća podatke koji su potvrđeni od strane većine članova u setu. Podaci koji se sada vrata su trajni, čak i u slučaju otkaza.

Neka je dat isti redosled događaja kao u slučaju opisanom u primeru za lokalni nivo. Međutim, kako sada replike moraju da čekaju potvrdu da se većinskom delu sistema operacija izvršila uspešno,

rezultati operacije W0 će tek nakon toga biti vidljivi u operacijama čitanja. Pregled stanja podataka u replikama prikazan je u sledećoj tabeli.

Čvor	Vreme	Podaci koji se vraćaju pri čitanju
Primar	Pre T3	Podaci ne sadrže efekte upisa W0, već Wprev
Primar	Nakon T3	Podaci sadrže efekte upisa W0
Prvi sekundar	Pre T5	Podaci ne sadrže efekte upisa W0, već Wprev
Prvi sekundar	Nakon T5	Podaci sadrže efekte upisa W0
Drugi sekundar	Pre T6	Podaci ne sadrže efekte upisa W0, već Wprev
Drugi sekundar	Nakon T6	Podaci sadrže efekte upisa W0

Tabela 2.

- “Linearizable”- Upit vraća podatke koji odražavaju sve uspešne većinsko 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 ima sposobnost linearizacije konkurentnih operacija.
- “Snapshot”-Upit vraća podatke koji su većinski potvrđeni u sistemu u nekom određenom trenutku u vremenu. Sa podešavanjem ovog parametra, možemo podesiti i parametar *atClusterTime* gde se može navesti timestamp vremenskog trenutka ukoliko želimo da vidimo kakvo je bilo stanje podataka u tom trenutku.

2.4.2 Potvrđivanje operacije upisa (*Write Concern*)

Način potvrđivanja operacije upisa kod MongoDB-a u setovima replika i distribuiranim šardovima određuje se parametrom “writeConcern”. Ovaj parametar se navodi na nivou pojedinačne operacije upisa, osim kod transakcija kada se writeConcern parametar postavlja na nivou cele svih operacija transakcije.

Parametar writeConcern sadrži nekoliko polja:

- Polje **w**-Ova opcija opisuje količinu potvrde koja je neophodna od sistema da bi operacija bila smatrana uspešnom. Može imati vrednost "majority" i tada se podrazumeva da je neophodna potvrda većine članova u sistemu. Vrednost polja w može biti i numerička i tada ona 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**-Ovim poljem se specificira vremenski limit za dobijanje potvrde o uspešnosti operacije upisa. Ukoliko se potvrda ne dobije od broja čvorova specificiranih u w polju do isteka *wtimeout* vremenskog perioda, operacija će se smatrati neuspešnom i klijentu će se vratiti greška. Ukoliko ne postavimo ovaj parametar, a u sistemu 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 operacije koja joj prethodi, kažemo da između te dve operacije postoji kauzalna zavisnost. Na primer operacija koja briše sve dokumente iz kolekcije i operacija čitanja nakon nje kojom se proverava da li je ta ista kolekcija prazna imaju ovakvu vrstu relacije. MongoDB podržava kauzalno konzistentne sesije koje garantuju izvršenje operacije u redosledu koji poštuje njihovu kauzalnu zavisnost.

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

Kauzalna konzistentna sesija pruža sledeće garancije:

- **Čitanje sopstvenih upisa** (*Read your writes*)-garancija da će svaka operacija čitanja sadržati efekte poslednje operacije upisa izvršene od strane istog klijenta.
- **Monotona čitanja** (*Monotonic reads*)-garancija da operacija čitanja može da vrati samo novije ili iste podatke u odnosu na operaciju čitanja koja joj je prethodila. Na primer, neka u sesiji operacija upisa w1 prethodi operaciji upisa w2, operacija čitanja r1 prethodi operaciji čitanja r2 i operacija čitanja r1 vraća podatke koji sadrže efekte operacije w2. Onda operacija čitanja koje se izvršava nakon nje r2, ne može da vrati podatke koji sadrže samo efekte operacije w2.
- **Monotoni upisi** (*Monotonic writes*)-garancija da će operacija upisa jedne sesije da se izvrši pre svih sukcesivnih operacija upisa iste sesije. Ukoliko na primer imamo operaciju upisa w1 koja se izvršava pre operacije upisa w2, operacija w2 mora da se izvrši nad podacima koji sadrže efekte operacije w1.
- **Čitanje nakon upisa** (*Writes follow reads*)-garancija da će operacija upisa u jednoj sesiji kojoj prethodi operacija čitanja u istoj sesiji da se izvrši nad podatkom koji je isti ili noviji u odnosu na onaj koji je bio pročitao u operaciji čitanja.

Ove garancije važe za sve čvorove u sistemu. Ukoliko, na primer, na primaru replike izvršimo operaciju upisa kojom se traži potvrda od većine članova u sistemu da se operacija izvršila i nakon toga izvršimo operaciju čitanja sa nekog sekundarnog člana replike, operacija čitanja će vratiti efekte prethodne operacije upisa.

3. Praktična demonstracija distribuiranih koncepata u MongoDB-U

3.1 Setovi replika

3.1.2 Konfiguracija seta replike

Set replike kod MongoDB-a sastoji se iz jedne primarne replike i više sekundarnih replika. Članova u setu replike može biti do 50, a preporuka je da ih bude makar 3. U nastavku je dat postupak za konfigurisanje jednog seta replike koji se sastoji iz jednog primara i dva sekundara. Koraci su sledeći:

1. **Pokretanje mongod instanci.** Prvi korak pri kreiranju seta replike jeste pokretanje instanci. Na slici je prikazana naredba kojom se jedan čvor seta replike pokreće:

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

Slika 13xPokretanje instance u setu replike

Parametri koje je potrebno navesti su: *--dbpath* kojim se navodi putanja do foldera na kom će se čuvati podaci, *--port* kojim se navodi broj porta na kome će se izvršavati instanca i pošto instanca pripada replika setu neophodno je navesti *--replSet* parametar iza kojeg je potrebno navesti ime seta replike koji se kreira.

Pošto će se set replike sastojati iz 3 člana, potrebno je pokrenuti još sve mongod instance, na isti način kao i prethodnu. Komande kojim se pokreću ostale instance prikazane su ispod. Bitno je za sve tri instance navesti istu vrednost *--replSet* parametra.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data1\db2" --port 27018 --replSet replicas
```

Slika 14. Pokretanje druge instance

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data1\db3" --port 27019 --replSet replicas
```

Slika 15. Pokretanje treće instance

2. **Konfiguracija seta replike.** Nakon što su replike pokrenute, neophodno je izvršiti konfiguraciju seta replike. To se radi tako što se prvo konektujemo na instancu za koju želimo da bude primarna replika (u ovom slučaju to će biti instanca na portu 27017). Zatim je potrebno pozvati *rs.initiate* naredbu kojoj se kao parametar prosleđuje konfiguraciona promenljiva. Konfiguraciona promenljiva treba da sadrži podatke o imenu seta replike u polju *_id* i adrese svih instanci koje će biti članovi seta u polju *members*. Na sledećoj slici prikazana je konfiguracija seta replike.

```

{
  _id: 'replicas',
  members: [
    { _id: 0, host: 'localhost:27017' },
    { _id: 0, host: 'localhost:27018' },
    { _id: 0, host: 'localhost:27019' }
  ]
}
test> rs.initiate(rsconf)

```

Slika 16. Konfiguracija seta replike

3.1.2 Operacije upisa nad setom replike

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.

Recimo da imamo tri člana seta replike, dva sekundarna i 1 primarni. 1 sekundarni čvor otkaže. Ukoliko imamo writeConcern postavljen na "majority" (što znači da većina čvorova mora da potvrdi 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. Ovo ponašanje prikazano je na sledećim slikama.

```

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

```

Slika 17. Write concern-"majority"

```

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

```

Slika 18. Write concern-"3"

3.2 MongoDB Sharding

3.2.1 Kreiranje šard klastera i šardovanje kolekcije

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

U nastavku će biti dat primer kreiranja jednog šard klastera, kao i šardovanje kolekcije na lokalnoj mašini koji se sastoji iz svih gorepomenutih članova.

Kreiranje i konfiguracija shardovanog klastera i distribucija kolekcije na klasteru sastoji se iz sledećih koraka:

1. **Kreiranje i konfiguracije konfiguracionog servera klastera.** Da bi kreirali konfiguracioni server treba kreirati mongo instancu na sledeći način:

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --configsvr --replSet ConfigReplSet --bind_ip localhost
```

Slika 19. Pokretanje konfiguracionog servera

Kao što se vidi sa slike potrebno je pokrenuti *mongod* instancu sa postavljenim `--configsvr` flegom. Takođe, poželjno je da konfiguracioni server bude pokrenut kao set replike, kako bi se poboljšala konzistentnost sistema. Zato je ovde postavljen `--replSet` fleg nakon koga sledi naziv seta replike. Što se tiče IP adrese servera, ovde se radi o lokalnoj mašini, a broj porta nije naveden u naredbi jer mongo automatski dodeljuje broj porta 27019 konfiguracionom serveru.

Pošto je konfiguracioni server pokrenut kao set replike, potrebno je izvršiti konfiguraciju te replike pozivom `rs.initiate()` metode. Pre poziva ove metode, potrebno je konektovati se sa pokrenutom serverskom instancom na portu 27019. Konfiguracija servera prikazana je na sledećoj slici.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongosh localhost:27019
Current Mongosh Log ID: 64982d8893a18f6699736be0
Connecting to:  mongod://localhost:27019/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.0
Using MongoDB:  6.0.6
Using Mongosh:  1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2023-06-25T13:41:47.417+02:00: Access control is not enabled for the database. Read and write access to data and configuration is
-----

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
ConfigReplSet [direct: primary] test> rs.initiate({
...   "_id" : "ConfigReplSet",
...   "configsvr" : true,
...   "members" : [
...     { "_id" : 0, "host" : "localhost:27019" }
...   ]
... })
```

Slika 20. Konfiguracija konfiguracionog servera

2. **Kreiranje i konfiguracija šard servera.** Nakon inicijalizacije konfiguracionog servera, možemo kreirati i konfigurisati instancu šard servera. Poželjno je i da šard bude set replika kako bi se povećala bezbednost i dostupnost podataka. Na sledećoj slici prikazana je komanda kojom se pokreće šard server. Ono što je ovde ključno jeste postavljanje `--shardsvr` flega. Port nije neophodno navesti, mongo po difoltu dodeljuje port broj 27018 šard serveru.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongod --shardsvr --replSet ShardReplSet --bind_ip localhost
```

Slika 21. Pokretanje šard servera

Nakon pokretanja šard servera, pošto je on pokrenut kao replika set, neophodno je izvršiti inicijalizaciju tog seta replike, slično kao kod konfiguracionog servera. Konfiguraciju šard servera prikazana je na sledećoj slici.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongosh localhost:27018
Current Mongosh Log ID: 649836b145fb4cabef7a6661
Connecting to:      mongodb://localhost:27018/?directConnection=true&serverSelectionTimeoutMS=30000
Using MongoDB:      6.0.6
Using Mongosh:      1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2023-06-25T14:14:16.324+02:00: Access control is not enabled for the database. Read more here: https://docs.mongodb.com/manual/reference/secure-replication/#access-control
-----

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test> rs.initiate({
...   "_id" : "ShardReplSet",
...   "members" : [
...     { "_id" : 0, "host" : "localhost:27018" }
...   ]
... })
{ ok: 1 }
```

Slika 22. Konfiguracija replika seta šarda

Na ovaj način kreiran je jedan šard server. Istim postupkom možemo kreirati i ostale šard servere.

3. **Kreiranje i konfiguracija *mongos* rutera.** Nakon kreiranja šard instanci i konfiguracionog servera, preostaje još samo kreiranje rutera operacija u šard klasteru. Ruter predstavlja posebnu mongo instancu koja se kreira *mongos* naredbom. Na sledećoj slici prikazno je kreiranje *mongos* instance.

```
C:\Program Files\MongoDB\Server\6.0\bin>mongos --configdb ConfigReplSet/localhost:27019 --bind_ip localhost
```

Slika 23. Kreiranje *mongos* rutera

Ono što je važno pri kreiranju *mongos* rutera jeste navođenje adrese konfiguracionog servera u `--configdb` parametru. Podrazumevan broj porta mongo rutera jeste 27017. Sva komunikacija sa klasterom ide preko rutera i sve operacije se prvo upućuju njemu.

4. **Dodavanje šardova u klaster.** Nakon što su svi potrebni članovi klastera kreirani, potrebno je povezati se sa instancom rutera i povezati šard servere sa klasterom. To se postiže `sh.addShard` operacijom koja za parametar uzima adresu šarda koji se dodaje u klaster.

```
> sh.addShard("ShardReplSet/localhost:27018")
{
  shardAdded: 'ShardReplSet',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687698304, i: 2 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687698304, i: 2 })
}

> sh.addShard("ShardReplSet1/localhost:27028")
{
  shardAdded: 'ShardReplSet1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687698336, i: 3 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687698336, i: 3 })
}
```

Slika 24 Dodavanje šardova u klaster

Na slici iznad prikazano je dodavanje prethodno kreiranih šardova na portovima 27018 i 27028 u šard klaster.

5. **Omogućavanje šardinga nad bazom podataka.** Da bi proces šardinga nad podacima bio omogućen, neophodno je komandom *sh.enableSharding* eksplicitno omogućiti šarding nad određenom bazom podataka.

```
> use populations
< switched to db populations
> sh.enableSharding("populations")
< {
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687715095, i: 2 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687715095, i: 1 })
}
```

Slika 25. Omogućavanje šardinga nad bazom podataka

6. **Šardovanje kolekcije.** Šardovanje, odnosno distribucija podataka po šardovima, se izvršava na nivou kolekcija u MongoDB-u. Da bi izvršili šardovanje kolekcije potrebno je i 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 navodi 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, mongo će je kreirati. Takođe će kreirati i indeks nad poljem navedenim kao ključ.

```
> sh.shardCollection("geodata.cities", { "country": 1, "name": "hashed" })
< {
  collectionsharded: 'geodata.cities',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687858842, i: 33 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687858842, i: 29 })
}
```

Slika 26. Šardovanje kolekcije

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 kolekciju i ključ za šardovanje.

7. **Dodavanje dokumenata u šardovanu kolekciju.** Nakon što je izvršena šarding operacija nad kolekcijom, ukoliko je kolekcija prazna možemo dodati dokumente koji će sada biti raspoređeni po šardovima. Naredbom na sledećoj prikazano je dodavanje 20 dokumenata sa podacima o gradovima u kolekciju `populations.cities`.

```
> db.cities.insertMany([
  {"name": "Seoul", "country": "South Korea", "population": 25.674 },
  {"name": "Mumbai", "country": "India", "population": 19.980 },
  {"name": "Belgrade", "country": "Serbia", "population": 1.374 },
  {"name": "Beijing", "country": "China", "population": 19.618 },
  {"name": "Shanghai", "country": "China", "population": 25.582 },
  {"name": "Osaka", "country": "Japan", "population": 19.281 },
  {"name": "Paris", "country": "France", "population": 2.161 },
  {"name": "Tokyo", "country": "Japan", "population": 37.400 },
  {"name": "Karachi", "country": "Pakistan", "population": 15.400 },
  {"name": "Dhaka", "country": "Bangladesh", "population": 19.578 },
  {"name": "Madrid", "country": "Spain", "population": 3.223 },
  {"name": "Oslo", "country": "Norway", "population": 0.634 },
  {"name": "Warsaw", "country": "Poland", "population": 1.765 },
  {"name": "Delhi", "country": "India", "population": 28.514 },
  {"name": "Barcelona", "country": "Spain", "population": 1.62 },
  {"name": "Kolkata", "country": "India", "population": 14.681 },
  {"name": "Berlin", "country": "Germany", "population": 3.465 },
  {"name": "Manila", "country": "Philippines", "population": 13.482 },
  {"name": "Chongqing", "country": "China", "population": 14.838 },
  {"name": "Istanbul", "country": "Turkey", "population": 14.751 }
])
```

Slika 27. Dodavanje dokumenata u šardovanu kolekciju

Nakon što smo dodali dokumente, pozivom operacije `db.cities.getShardDistribution()` moguće je videti kako su dokumenti raspoređeni po šardovima. Rezultat ove naredbe nakon dodavanja dokumenata prikazan je na sledećoj slici.

```

> db.cities.getShardDistribution()
<
  Shard ShardReplSet1 at ShardReplSet1/localhost:27028

  {
    data: '943B',
    docs: 9,
    chunks: 2,
    'estimated data per chunk': '471B',
    'estimated docs per chunk': 4
  }

  Shard ShardReplSet at ShardReplSet/localhost:27018

  {
    data: '1KiB',
    docs: 11,
    chunks: 2,
    'estimated data per chunk': '567B',
    'estimated docs per chunk': 5
  }

```

Slika 28. Distribucija dokumenata po šardovima

Kao što se vidi sa slike, od ukupno 20 dokumenata, 9 je raspoređeno na prvom šardu, a 11 na drugom. To govori o prilično ravnomernoj raspodeli koja je rezultat dobro odabranog ključa.

8. **Izvršenje upita nad šardovanom kolekcijom.** Sada kada imamo dokumente distribuirane po šardovima, možemo izvršiti neke upite nad kolekcijom kako bi videli na koji način ruter upita prosleđuje operacije. Na slici ispod dat je rezultat *explain* operacije upita koji pretražuje kolekciju po vrednosti poljima koji su prefiks ključa šardinga.

```

> db.cities.find({country:"Serbia",name:"Belgrade"}).explain()
< {
  queryPlanner: {
    mongosPlannerVersion: 1,
    winningPlan: {
      stage: 'SINGLE_SHARD',
      shards: [
        {
          shardName: 'ShardReplSet',
          connectionString: 'ShardReplSet/localhost:27018',
          serverInfo: {
            host: 'DESKTOP-2039D2R',
            port: 27018,
            version: '6.0.6',
            gitVersion: '26b4851a412cc8b9b4a18cdb6cd0f9f642e06aa'
          }
        }
      ]
    }
  }
}

> db.cities.find({country:"Spain"}).explain()
< {
  queryPlanner: {
    mongosPlannerVersion: 1,
    winningPlan: {
      stage: 'SINGLE_SHARD',
      shards: [
        {
          shardName: 'ShardReplSet',
          connectionString: 'ShardReplSet/localhost:27018',
          serverInfo: {
            host: 'DESKTOP-2039D2R',
            port: 27018,
            version: '6.0.6',
            gitVersion: '26b4851a412cc8b9b4a18cdb6cd0f9f642e06aa7'
          }
        }
      ]
    }
  }
}

```

Slika 29. Upit po prefiksu ključa šardinga

Kao što se sa slike vidi planer upita za izvršenje planira pristupanje samo jednom od šardova (stage: SINGLE_SHARD). Ovo je prednost izvršavanja upita po vrednosti ključa za šarding jer na osnovu te vrednosti ruter klastera može tačno da odredi na kom šardu se nalaze željeni podaci i nema potrebe da pristupa ostalim šardovima.

Na slici ispod prikazan je rezultat operacije explain onda kada za pretragu ne koristimo ključ šardinga već neko drugo polje dokumenata. U tom slučaju se mora pristupiti svim šardovima i izvršenje upita nije najefikasnije.

```
> db.cities.find( { population: { $gt: 1 } } ).explain()
< {
  queryPlanner: {
    mongosPlannerVersion: 1,
    winningPlan: {
      stage: 'SHARD_MERGE',
      shards: [
        {
          shardName: 'ShardReplSet',
          connectionString: 'ShardReplSet/localhost:27018',
          serverInfo: {
            host: 'DESKTOP-2039D2R',
            port: 27018,
            version: '6.0.6',
            gitVersion: '26b4851a412cc8b9b4a18cdb6cd0f9f642e06aa7'
          }
        }
      ]
    }
  },
```

Slika 30. Upit po polju koje nije prefiks ključa šardinga

3.2.2 Šard zone

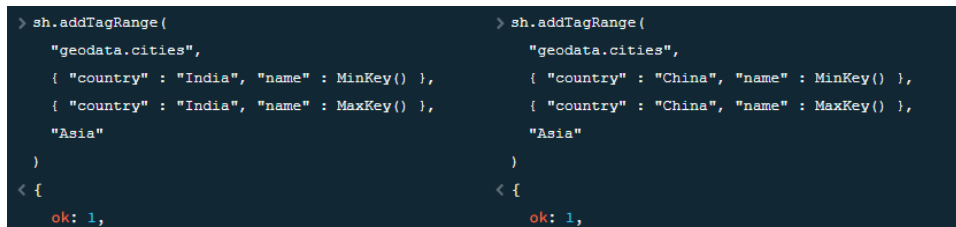
Nekada je pogodno šardovima dodeliti određene opsege ključa particije kako bi se određeni dokumenti skladištili samo na određenim šardovima. Posebno je pogodno rasporediti podatke geografski po šardovima tako da podaci koji su relevantni za jednu geografsku oblast budu raspoređeni u čvorovima najbližim toj oblasti.

Kolekcija korišćena u prethodnom primeru predstavlja kolekciju gradova raspostranjenim u dva kontinenta: Evropi i Aziji. Zbog toga su kreirane dve zone: Evropa i Azija. Pokrenuta su 3 šarda. Prvom šardu (*ShardReplSet*) dodeljena je zona Europe, a drugom i trećem šardu (*ShardReplSet1* i *ShardReplSet2*) dodeljena je zona Asia. Ova podela prikazana je na slici ispod.

```
> sh.addShardTag("ShardReplSet", "Europe")
< {
  ok: 1,
}
> sh.addShardTag("ShardReplSet1", "Asia")
< {
  ok: 1,
}
> sh.addShardTag("ShardReplSet2", "Asia")
< {
  ok: 1,
}
```

Slika 31. Dodeljivanje zona šardovima

Nakon što se šardovima dodele zone, potrebno je za svaku zonu dodeliti rangove ključa distribucije. Ovde je kao ključ distribucije korišćen kompozitni ključ koji se sastoji iz dva polja i ima sledeću formu: {"country":1,"name":"hashed"}. Prilikom dodeljivanja opsega zoni potrebno je navesti donju i gornju granicu ključa za particiju. Jednoj zoni može biti dodeljen veći broj opsega. Ta činjenica je iskorišćena i ovde, pa su zoni Evropa dodeljeni svi ključevi čiji prefiks, odnosno polje *country*, predstavlja neku Evropsku zemlju (Srbija, Francuska, Norveška,..) a zoni Azija su dodeljeni dokumenti u kojima vrednost *country* predstavlja neku Azijsku zemlju. Primeri dodeljivanja opsega zonama prikazani su na sledeće dve slike.



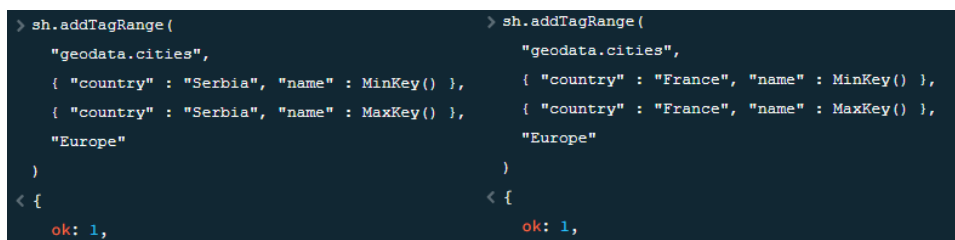
```

> sh.addTagRange(
  "geodata.cities",
  { "country" : "India", "name" : MinKey() },
  { "country" : "India", "name" : MaxKey() },
  "Asia"
)
< {
  ok: 1,
}

> sh.addTagRange(
  "geodata.cities",
  { "country" : "China", "name" : MinKey() },
  { "country" : "China", "name" : MaxKey() },
  "Asia"
)
< {
  ok: 1,
}

```

Slika 32. Dodeljivanje opsega zoni Evropa



```


> sh.addTagRange(
  "geodata.cities",
  { "country" : "Serbia", "name" : MinKey() },
  { "country" : "Serbia", "name" : MaxKey() },
  "Europe"
)
< {
  ok: 1,
}

> sh.addTagRange(
  "geodata.cities",
  { "country" : "France", "name" : MinKey() },
  { "country" : "France", "name" : MaxKey() },
  "Europe"
)
< {
  ok: 1,
}

```

Slika 33. Dodeljivanje opsega zoni Azija

Sada kada su zone definisane možemo izvršiti šarding kolekcije *cities*.



```

> sh.shardCollection("geodata.cities", { "country": 1, "name": "hashed" })
< {
  collectionsharded: 'geodata.cities',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687858842, i: 33 }),
    signature: {
      hash: Binary(Buffer.from("00000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687858842, i: 29 })
}

```

Slika 34. Šardovanje kolekcije

Dokumenti dodati u kolekciju su dokumenti isti kao u prethodnom primeru. Podela po šardovima u ovom primeru prikazana je na sledećoj slici. Prvom šardu koji pripada zoni Evropa dodeljeno je 7 dokumenata što je ukupan broj evropskih gradova u kolekciji. Ostalih 13 gradova koji pripadaju Aziji raspoređeno je u druga dva šarda.

```
> sh.getShardedDataDistribution()
< {
  ns: 'geodata.cities',
  shards: [
    {
      shardName: 'ShardReplSet1',
      numOrphanedDocs: 0,
      numOwnedDocuments: 7,
      ownedSizeBytes: 560,
      orphanedSizeBytes: 0
    },
    {
      shardName: 'ShardReplSet2',
      numOrphanedDocs: 0,
      numOwnedDocuments: 6,
      ownedSizeBytes: 474,
      orphanedSizeBytes: 0
    },
    {
      shardName: 'ShardReplSet',
      numOrphanedDocs: 0,
      numOwnedDocuments: 7,
      ownedSizeBytes: 553,
```

Slika 35. Distribucija po šardovima

3.4 Distribuirane transakcije

MongoDB pruža podršku izvršavanja atomičnih transakcija u šardovanim klasterima i setovima replika. Svaka transakcija postoji u okviru jedne klijentske sesije i kreira se iz nje. Sve operacije koje se navedu u transakciji izvršavaju se po principu “sve ili ništa”:ili će sve operacije biti uspešno izvršene ili neće nijedna.

Ovde je kreirana jedna C# konzolna aplikacija koja koristi *MongoDB.Driver*. Tu je demonstrirana transakcija koja se sastoji dodavanja nekoliko dokumenata u kolekciju i nakon toga ažuriranja tih istih dokumenata. Operacije od kojih želimo da se transakcija sastoji navode se između poziva operacija *startTransaction* i *endTransaction*. Na slici ispod prikazana je transakcija dodavanja novih dokumenata u vidu proizvoda u kolekciju *products* i zatim njihovo ažuriranje u istoj transakciji.

```
// Begin transaction
session.StartTransaction();

try
{
    // Insert the sample data

    Console.WriteLine("hil");
    await products.InsertOneAsync(session, tv);
    await products.InsertOneAsync(session, book);
    await products.InsertOneAsync(session, dogBowl);
    Console.WriteLine("hil");
    var resultsBeforeUpdates = await products
        .Find<Product>(session, Builders<Product>.Filter.Empty)
        .ToListAsync();
    Console.WriteLine("Original Prices:\n");
    foreach (Product d in resultsBeforeUpdates)
    {
        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 UpdateDefinitionBuilder<Product>()
        .Mul<Double>(r => r.Price, 1.1);
    await products.UpdateManyAsync(session,
        Builders<Product>.Filter.Empty,
        update); //,options);

    // Made it here without error? Let's commit the transaction
    await session.CommitTransactionAsync();
}
```

Slika 36. Transakcija dodavanja i ažuriranja proizvoda

4. Zaključak

Kao što je u seminarskom pokazano, MongoDB zaista ima veliku podršku za distribuirane baze podataka u vidu replikacije podataka, horizontalnog skaliranja podataka, distribuiranih upita i transakcija,... MongoDB je izuzetno pogodan za velike sisteme gde jedna mašina nije dovoljna. Međutim, treba dobro razmisliti o projektovanju i konfiguraciji mongo koncepata jer na performanse baze će najviše uticati izbori koje sami napravimo. Tako u procesu replikacije treba voditi računa o broju replika i načinu na koji su raspoređene, kod sharding-a treba na pravi način odabrati ključ za podelu podataka, kod transakcija treba voditi računa o broju dokumenata kojima se pristupa... Zato se treba dobro upoznati sa ovim konceptima pre nego što krenemo da ih primenjujemo.

5. Literatura

1. [What is MongoDB? — MongoDB Manual](#)
2. [How To Use Sharding in MongoDB | DigitalOcean](#)
3. [The “Majority” WriteConcern of MongoDB Replica Sets | by OnlyKiosk Dev Tech | Level Up Coding](#)
4. [Causal Consistency in Mongo. Written by Kert Piatkin | by Outfunnel Tech Blog | Medium](#)
5. [How To Use Transactions in MongoDB | DigitalOcean](#)