UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Distribucija baze kod *MongoDB*

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Mentor:

Aleksandar Stanimirović

Student:

Petar Mančić

1. Uvod....3
2. Osnove distribucije podataka u *MongoDB*....4
2.1 Replikacija u *MongoDB*-u....4
2.1.1 Struktura *Replica Set*-a....4
2.1.2 Prednosti replikacije....5
2.2 Shardovanje (*Sharding*)....8
2.2.1 Zašto *sharding*?....9
2.3 Komponente distribuiranog sistema u *MongoDB*-u....9
2.4 *Shard Key*....11
2.5 *Balancer* i automatska redistribucija podataka....12
3. Arhitektura *Sharded Cluster*-a....13
3.1 Opšti pregled arhitekture....14
3.2. Upravljanje raspodelom podataka (*Chunk Map*)....15
3.3. Skaliranje klastera i dodavanje novih shardova....15
3.4 Otpornost na greške (*Fault Tolerance*)....16
4. Kako *MongoDB* deli podatke – *Shard Key* i *Chunkovi*....16
5. Strategije shardovanja....18
6. Kako upiti rade u distribuiranom okruženju....19
7. Praktični primeri....20
7.1. Pokretanje instanci....20
7.2. Demonstriranje principa replikacije....25
7.3. Otpornost na greške....28
7.3.1. Izbor nove *primary* instance....29
8. Zaključak....33
9. Literatura....34

1. Uvod

U savremenim informacionim sistemima količina podataka raste izuzetno brzo, a potrebe aplikacija sve češće zahtevaju obradu ogromnih datasetova u realnom vremenu. Tradicionalne relacione baze podataka, zasnovane na vertikalnom skaliranju, često nisu u stanju da efikasno podrže takav rast. Zbog ograničenja jedne mašine, sve je teže postići visok *throughput*, nisku latenciju i dostupnost u okruženjima sa velikim brojem korisnika. Upravo iz ovog razloga distribuirane baze podataka postaju ključna komponenta modernih sistema.

Distribuirana baza podataka predstavlja sistem u kojem se podaci fizički nalaze na više različitih servera, ali se aplikaciji predstavljaju kao jedinstvena logička celina. Distribucija može biti motivisana potrebom za skaliranjem, visokom dostupnošću, bržom obradom ili geografski raspoređenim korisnicima. Ključni izazovi uključuju konzistentnost podataka, replikaciju, toleranciju na greške i optimalnu raspodelu opterećenja.

MongoDB, kao najpopularnija *NoSQL* dokument-orijentisana baza podataka, razvijen je upravo sa idejom da nativno podrži horizontalno skaliranje i da omogući jednostavnu distribuciju podataka preko više čvorova. Za razliku od tradicionalnih baza koje se pre svega oslanjaju na vertikalno skaliranje, *MongoDB* omogućava ***sharding***, odnosno podelu kolekcija na manje delove (*shardove*) koji se raspoređuju preko više servera. Ovakav pristup omogućava da se kapacitet za skladištenje i obradu podataka praktično neograničeno proširuje dodavanjem novih mašina u klaster.

Distribucija u *MongoDB*-u kombinacija je dva ključna mehanizma:

1. **Replikacije (*Replication*)** – za obezbeđivanje visoke dostupnosti i tolerancije na greške.
2. **Shardovanja (*Sharding*)** – za raspodelu opterećenja i skaliranje u širinu (horizotalno).

Zahvaljujući ovoj arhitekturi, *MongoDB* se često koristi u sistemima gde se obrađuju **milioni** ili **milijarde** dokumenata – kao što su aplikacije za društvene mreže, *e-commerce* platforme, *IoT* sistemi ili analitički servisi.

U nastavku rada biće detaljno objašnjeni osnovni koncepti distribucije podataka u *MongoDB*-u, kao i arhitektura sharded klastera koji predstavlja najnapredniji vid horizontalnog skaliranja u

ovoj bazi. Ovo će pružiti jasnu sliku o tome zašto je *MongoDB* postao industrijski standard kada su u pitanju distribuirani *NoSQL* sistemi.

2. Osnove distribucije podataka u *MongoDB*

Distribucija podataka predstavlja osnovu skalabilnosti i dostupnosti u *MongoDB*-u. Za razliku od klasičnih relacionih baza koje se oslanjaju na centralizovanu strukturu, *MongoDB* je od samog početka dizajniran kao distribuirana baza podataka koja može raditi na desetinama ili stotinama čvorova. Distribucija se postiže korišćenjem dva ključna mehanizma:

1. **Replikacija (*Replication*)**
2. **Shardovanje (*Sharding*)**

Ova dva mehanizma zajedno omogućavaju horizontalno skaliranje, visoku dostupnost i otpornost na greške, što čini *MongoDB* pogodnim za sisteme velikog obima.

2.1 Replikacija u *MongoDB*-u

Replikacija je proces čuvanja identičnih kopija podataka na više servera. U *MongoDB*-u se replikacija ostvaruje kroz ***Replica Set***, koji predstavlja grupu *MongoDB* instanci koje održavaju iste podatke.

2.1.1 Struktura *Replica Set*-a

Replica Set tipično sadrži:

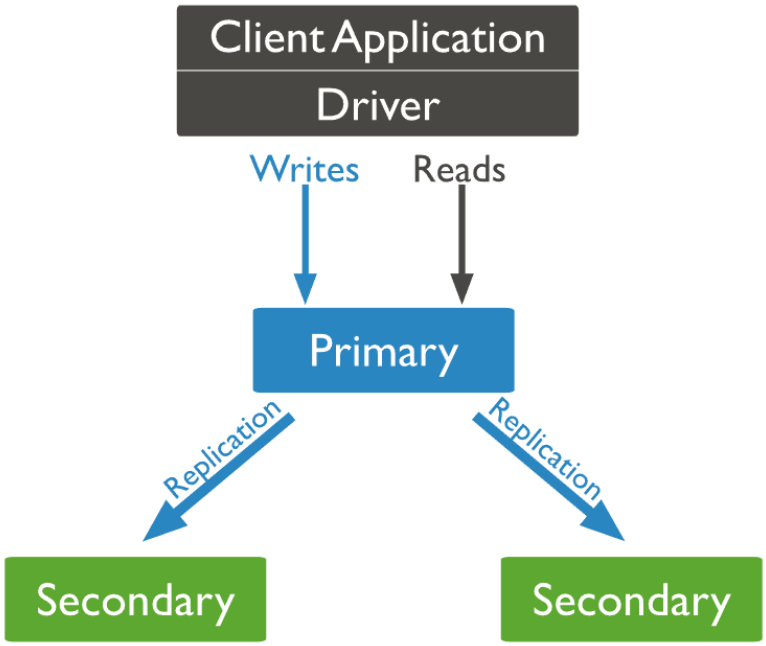
- ***Primary*** – glavni čvor koji prima sve upise (*write* operacije).
- ***Secondary*** – jedan ili više čvorova koji repliciraju podatke sa primarnog.
- ***Arbiter* (opciono)** – učestvuje u izboru novog primarnog čvora, ali ne čuva podatke.

Kada primarni čvor otkaže, sistem automatski bira novi primarni čvor (*failover* mehanizam), čime se obezbeđuje visoka dostupnost. Ovaj proces je potpuno automatizovan i transparentan za aplikacije.

2.1.2 Prednosti replikacije

- **Otpornost na greške** – gubitak jednog servera ne utiče na rad aplikacije.
- **Skaliranje čitanja** – neke read operacije se mogu slati na *secondary* čvorove.
- ***Backup* i sigurnost podataka** – podaci se nalaze na više fizičkih lokacija.

Međutim, sama replikacija ne rešava problem rasta podataka — svi čvorovi čuvaju kompletan *dataset*. Zato se koristi *sharding*.

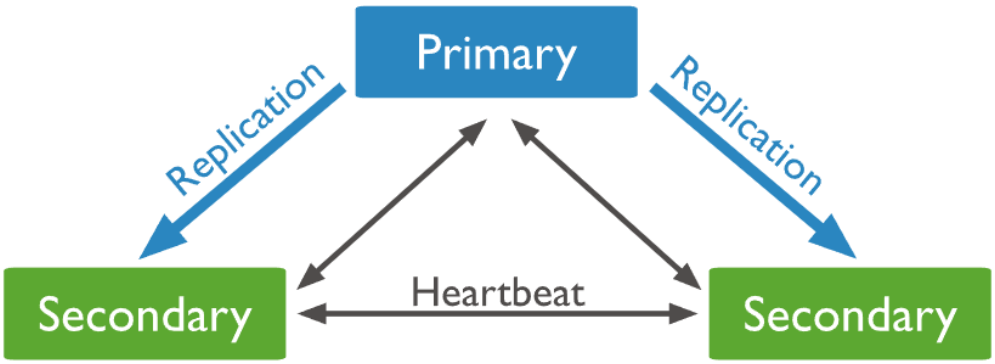


Slika 1 Osnovni princip replikacije (Primari + Secondary)

Ova slika prikazuje **osnovnu arhitekturu *MongoDB* replike**, odnosno kako izgleda komunikacija između primarnog i sekundarnih čvorova.

Na slici 1 prikazan je osnovni princip replikacije:

- ***Client Application*** i *MongoDB* **Driver** se nalaze na vrhu.
  Oni šalju upite bazi.
- **Sve *write* operacije (upisi)** idu direktno na ***Primary*** čvor.
  MongoDB uvek zapisuje samo na jedan glavni čvor radi konzistentnosti.
- ***Secondary*** čvorovi dobijaju podatke putem ***Replication*** procesa.
  Oni periodično preuzimaju (apliciraju) oplog zapis primarnog čvora.
- ***Read*** **operacije** mogu ići na:
  - ***Primary***, po default-u (najsvežiji podaci)
  - ***Secondary***, ako se uključi *readPreference* (koristi se za *load balancing*)



Slika 2 Replica Set sa heartbeat-om i detekcijom grešaka

Ova slika prikazuje kako *MongoDB* implementira **automatski failover** i komunikaciju između čvorova u replica set-u.

- Vidimo ***Primary*** i dva ***Secondary*** čvora.
- I dalje postoji replikacija od Primary ka *Secondary*.
- Novina: pojavljuju se ***heartbeat*** **poruke** između svih čvorova.

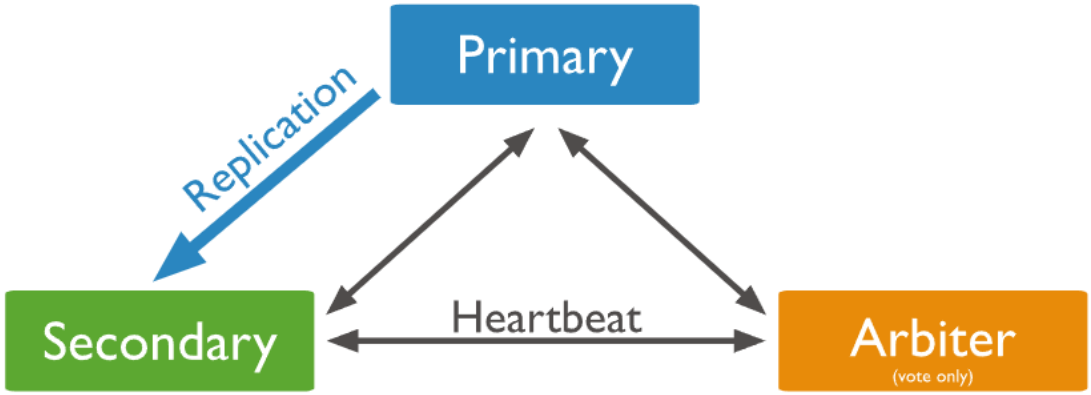
Šta su ***heartbeat*** poruke?

To su kratke periodične poruke koje čvorovi šalju jedni drugima da bi proverili:

- da li je drugi čvor živ,
- da li je primarni i dalje aktivan,
- da li treba pokrenuti izbor (*election*) za novog primarnog.

Zašto je važan *heartbeat*?

Ako sekundarni čvorovi ne čuju primarni čvor određeni vremenski period (10 sekundi po default-u), automatski pokreću izbor za novog primarnog.



Slika 3 Replica Set sa Arbiter čvorom

Ova slika uvodi novu komponentu u replica set — ***Arbiter***.

Replica set ima:

- ***Primary***
- ***Secondary***
- ***Arbiter*** (posebna instanca koja ne čuva podatke)
- *Arbiter* učestvuje u ***heartbeat* komunikaciji** — proverava dostupnost čvorova.
- *Arbiter* ne replicira podatke i ne učestvuje u čitanju ili pisanju.

- Njegova jedina funkcija je:
  Učestvovanje u glasanjima prilikom izbora novog primarnog (*election*).

Zašto se koristi *Arbiter*?

Arbiter se koristi kada:

- želimo **neparan broj glasova** u replica set-u,
- ne želimo trošak dodatnog servera koji čuva podatke,
- želimo minimizovati resurse (npr. u malim okruženjima).

Prednosti *Arbiter*-a

- Omogućava izbor primarnog bez gubitka većine.
- Ne troši mnogo resursa (nema podataka ni oploga).

Mane *Arbiter*-a

- Nema kopiju podataka — ne doprinosi dostupnosti.
- Predstavlja sigurnosni rizik (nema enkripciju podataka, ali glasa o izboru primarnog).

2.2 Shardovanje (*Sharding*)

Shardovanje je proces horizontalne podele podataka na više servera (shardova). Svaki shard čuva samo deo celokupne kolekcije, čime se postiže mogućnost obrade ogromnih datasetova.

U *MongoDB*-u shard nije pojedinačni server, već ***replica set***, tako da svaki *shard* ima svoju visoku dostupnost.

2.2.1 Zašto *sharding*?

Sharding rešava sledeće probleme:

- **Rast *dataset*-a** – kada jedna mašina ne može više da skladišti sve podatke.
- **Veliko opterećenje pri čitanju i pisanju** – *workload* se raspoređuje na više čvorova.
- **Smanjeno vreme odziva** – podaci i upiti se distribuiraju po shardovima.

Bez *sharding*-a, skaliranje bi bilo moguće samo vertikalnim dodavanjem resursa (*CPU, RAM, SSD*), što ima fizička i finansijska ograničenja.

2.3 Komponente distribuiranog sistema u *MongoDB*-u

Distribuirani sistemi u *MongoDB*-u zasnivaju se na arhitekturi šardovanog klastera (*sharded cluster*), koja omogućava horizontalnu skalabilnost i ravnomernu raspodelu podataka. Šardovani klaster se sastoji od više jasno definisanih komponenti, pri čemu svaka ima specifičnu ulogu u procesu čuvanja, pristupa i upravljanja podacima. Osnovne komponente koje čine ovakav distribuirani sistem su **shardovi**, ***mongos* ruteri** i **konfiguracioni serveri** (*config servers*).

1. Shardovi

Shard predstavlja osnovnu jedinicu skladištenja podataka u šardovanom klasteru. Svaki shard sadrži **samo deo ukupnog dataset-a**, pri čemu je raspodela dokumenata zasnovana na unapred definisanom shard ključu. Da bi se obezbedila visoka dostupnost i tolerancija na greške, svaki shard mora biti implementiran kao **replika set**. Na taj način, podaci se čuvaju na više replika, što omogućava oporavak u slučaju pada pojedinačnih čvorova i istovremeno povećava mogućnost paralelne obrade upita.

2. *Config Servers (Config Server Replica Set – CSRS)*

Config serveri (*Config Server Replica Set — CSRS*) skladište metapodatke o klasteru, uključujući:

- raspodelu podataka po shardovima,
- informacije o shard ključevima,
- stanja i konfiguraciju svih komponenti klastera.

S obzirom na to da su metapodaci kritični za korektan rad celog distribuiranog sistema, config serveri **moraju** biti implementirani kao replika set. Oni omogućavaju *mongos* instancama da u realnom vremenu imaju uvid u raspored podataka, čime se obezbeđuje pravilno rutiranje upita i održavanje konzistentnosti.

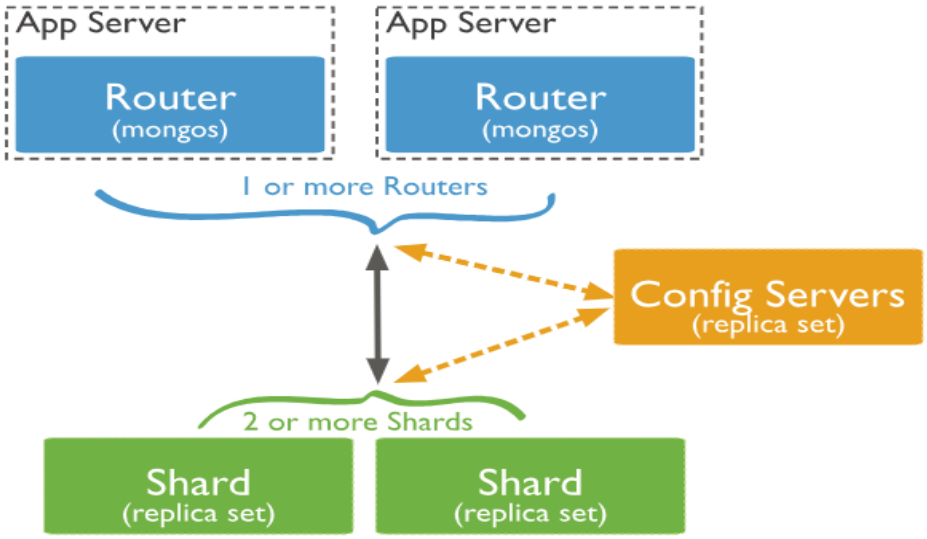
Bez *config* servera ceo sistem ne može da funkcioniše.

3.*Mongos Router*— komponenta za rutiranje upita

Mongos je „router“ koji prihvata upite od aplikacije i prosleđuje ih odgovarajućim shardovima.

On zna gde se nalaze podaci, zahvaljujući informacijama sa config servera.

Aplikacija ne komunicira direktno sa shardovima, već isključivo preko *mongos*-a. Na taj način aplikacija dobija iluziju da radi sa jednom, centralizovanom bazom.



Slika 4 Interakcija između komponenata u sharding clusteru

Međusobna interakcija komponenti

U tipičnom radu šardovanog klastera, interakcija komponenti funkcioniše na sledeći način:

1. Klijent šalje upit *mongos* ruteru.
2. *Mongos* konsultuje metapodatke sa config servera kako bi odredio koji shard sadrži tražene podatke.
3. Upit se prosleđuje odgovarajućim shardovima.
4. Shardovi obrađuju upit i vraćaju rezultate.
5. *Mongos* agregira pristigle rezultate i prosleđuje ih klijentu.

Ovakav način organizacije omogućava efikasnu distribuciju podataka, skaliranje u širinu, kao i uravnotežen rad velikih sistema pod opterećenjem.

2.4 *Shard Key*

Shard key predstavlja centralni element sharding mehanizma u *MongoDB*-u i određuje način na koji se dokumenti raspoređuju između shardova u distribuiranom klasteru. Shard key je jedno polje ili kombinacija više polja u dokumentu, a njegova vrednost direktno utiče na to gde će se dokument fizički nalaziti. Zbog toga pravilno odabran shard key ima ključnu ulogu u performansama, skalabilnosti i ukupnoj efikasnosti sistema.

U trenutku kada se kolekcija sharduje, korisnik mora da definiše shard key. Nakon toga, svaki dokument dobija svoju poziciju u klasteru na osnovu vrednosti tog ključa, pri čemu *MongoDB* deli kolekciju na manje logičke jedinice – *chunkove* – formirane prema opsegu shard key vrednosti. Svaki chunk se zatim smešta na određeni shard, a *MongoDB*-ov balancer se stara da raspodela podataka bude ravnomerna.

Izbor shard key-a direktno utiče na tip sharding strategije koju *MongoDB* može da primeni, kao što su ***range sharding***, ***hash sharding*** ili ***zone-based sharding***. Takođe, shard key mora biti podržan indeksom koji počinje njegovim poljima, kako bi se omogućila efikasna obrada upita.

Kvalitet *shard key*-a presudno utiče na performanse. Dobar shard key poseduje visoku kardinalnost, ravnomerno raspoređuje podatke i često se koristi u filterima upita, čime se omogućava tzv. *targeted query* – upit koji se izvršava samo na jednom shardu. Loše izabran *shard*

key može dovesti do problema poput „*hot shard*-a“, neravnomerne raspodele podataka, povećanog broja „*scatter-gather*“ upita i opšte degradacije performansi.

Dobar *shard key* treba da obezbedi:

- ravnomernu raspodelu podataka,
- ravnomernu raspodelu upita,
- izbegavanje "*hotspot*" problema.

Loš *shard key* vodi do:

- neravnomerne raspodele,
- zagušenja jednog *shard*-a,
- sporog balanciranja.

Zato je izbor *shard key*-a jedna od najvažnijih arhitektonskih odluka u *MongoDB*-u.

Počev od verzije *MongoDB* 5.0, sistemi podržavaju i ***resharding*** – promenu shard *key*-a nakon što je kolekcija već shardovana, kao i **refiniranje *shard key*-a** dodavanjem dodatnih polja u njegov sufiks. Ovo značajno povećava fleksibilnost pri planiranju i skaliranju sistema.

2.5 *Balancer* i automatska redistribucija podataka

MongoDB poseduje *balancer* proces koji automatski premesti delove kolekcija, tzv. **chunks**, sa shardova koji imaju previše podataka na one koji imaju manje. Ovaj mehanizam je ključan za održavanje ravnomernog opterećenja u *sharded cluster*-ovima i omogućava sistemu da skalira horizontalno bez potrebe za ručnom intervencijom administratora.

Kako balancer funkcioniše:

- Svaka *shardovana* kolekcija je podeljena na više *chunk*-ova, koji predstavljaju intervale vrednosti ključnog polja (*shard key*).
- Balancer periodično proverava distribuciju *chunk*-ova među shardovima i identifikuje neravnotežu (npr. kada jedan shard ima znatno više *chunk*-ova od drugog).

- Kada se detektuje neravnoteža, balancer automatski premesti pojedinačne *chunk*-ove sa preopterećenog sharda na onaj koji ima manje podataka.
- Ova redistribucija se odvija u pozadini i ne utiče značajno na normalan rad aplikacija.

Zašto je ovo važno:

- Održava ravnomerno opterećenje svih shardova, što poboljšava performanse upita i upisa.
- Sprečava situaciju u kojoj jedan shard postaje „usko grlo“ zbog preopterećenja.
- Omogućava lakše skaliranje sistema dodavanjem novih shardova bez potrebe za ručnim premeštanjem podataka.

Praktičan primer:

Zamislimo da imamo *sharded cluster* sa tri *shard*-a, i da je kolekcija orders nejednako raspoređena — shard 1 sadrži 70% podataka, dok *shard* 2 i *shard* 3 po 15% svaki. Balancer automatski premešta *chunk*-ove sa shard-a 1 na shard 2 i 3, čime se postiže ravnomernija raspodela podataka i opterećenja.

3. Arhitektura *Sharded Cluster*-a

Arhitektura *MongoDB* sharded klastera predstavlja organizacioni model koji omogućava horizontalno skaliranje baze podataka raspodelom podataka na više nezavisnih serverskih jedinica (shardova). Iako su osnovne komponente šardovanog okruženja — shardovi, config serveri i mongos ruteri — već opisane u prethodnom poglavlju, ovaj deo rada fokusira se na način na koji se ove komponente međusobno povezuju, sarađuju i upravljaju podacima unutar jedinstvenog distribuiranog sistema.

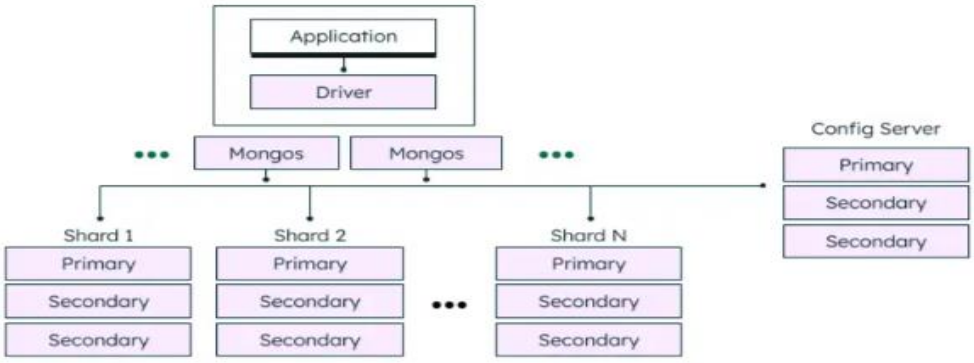
Sharded klaster obezbeđuje visoku dostupnost, skalabilnost i efikasnost izvršavanja upita, a njegova arhitektura je zasnovana na jasno definisanim odgovornostima svake komponente i centralizovanoj metapodatkoj strukturi koju održavaju *config* serveri.

3.1 Opšti pregled arhitekture

Na visokom nivou, sharded klaster se sastoji od tri logička sloja:

1. **Sloj skladištenja (*Shard Layer*)** – čuva stvarne podatke. Svaki shard je replica set, čime se obezbeđuje otpornost na greške i replikacija podataka.
2. **Sloj metapodataka *(Metadata Layer*)** – čine ga config serveri. Oni održavaju informacije o raspodeli podataka, lokaciji chunkova i konfiguraciji klastera.
3. **Sloj rutiranja upita (*Query Routing Layer*)** – implementiran preko mongos procesa, koji predstavlja interfejs između klijentskih aplikacija i celokupnog klastera.

Sve ove komponente zajedno formiraju celovit i distribuiran ekosistem koji se ponaša kao jedna logička baza podataka, iako fizički može biti raspoređen na desetine ili stotine servera.



Slika 5 Dijagram arhitekture sharded clustera

Prikazani dijagram ilustruje osnovnu arhitekturu *MongoDB* sharded klustera. Aplikacija komunicira sa sistemom preko *MongoDB* drivera i **mongos** procesa, koji služe kao ruteri za upite i određuju na koji shard treba poslati operaciju.

Svaki ***shard*** predstavlja deo ukupnih podataka i implementiran je kao ***replica set*** (*primary* + *secondary* čvorovi), što obezbeđuje i skaliranje i visoku dostupnost.

Sa desne strane nalazi se ***Config Server Replica Set***, koji čuva metapodatke o raspodeli podataka i omogućava correctno rutiranje upita. Mongos koristi ove metapodatke da pronađe shard koji sadrži tražene informacije i da vrati objedinjeni rezultat aplikaciji.

3.2. Upravljanje raspodelom podataka (*Chunk Map*)

Jedan od ključnih zadataka sharded arhitekture jeste održavanje tačnih i ažurnih informacija o tome **koji shard poseduje koji deo podataka**. Ova informacija se čuva u takozvanoj *chunk mapi* (eng. *chunk map*), koju održavaju *config* serveri.

Chunk mapa sadrži:

- ime kolekcije,
- *shard key*,
- opseg vrednosti chunk-a (npr. { "userId": 1000 – 1999 }),
- *shard* kojem *chunk* pripada,
- stanje migracije chunkova.

Ove informacije omogućavaju balanceru da detektuje neravnomernu raspodelu i pokrene migraciju chunkova.

3.3. Skaliranje klastera i dodavanje novih shardova

Velika prednost sharded arhitekture je mogućnost **dinamičkog skaliranja**. Dodavanje novog shard-a obavlja se u toku rada klastera, bez prekida sistema.

Nakon dodavanja sharda:

1. *Config* serveri beleže novi shard.
2. *Balancer* prepoznaje da novi shard nema chunkova.
3. Pokreće se automatska migracija chunkova ka novom *shard*-u.

4. Klaster dostiže novu ravnotežu u raspodeli podataka.

Ovaj proces je u potpunosti automatizovan, što administraciju čini jednostavnijom.

3.4 Otpornost na greške (*Fault Tolerance*)

Pošto je svaki shard zapravo replica set, klaster može nastaviti da radi čak i ako jedan ili više nodova unutar sharda otkažu. Fault tolerance se obezbeđuje kroz:

- automatski failover,
- replikaciju podataka,
- distribuciju podataka na više fizičkih mašina,
- obnavljanje nodova bez prekida rada klastera.

Iako *config* serveri imaju kritičnu ulogu, i oni su replica set, pa njihov pad ne dovodi do gubitka podataka ili raspada sistema — sve dok većina članova može da održava *quorum*.

4. Kako *MongoDB* deli podatke – *Shard Key* i *Chunkovi*

Raspodela podataka u *MongoDB* sharded klasteru zasniva se na dva ključna elementa: ***shard key*** i ***chunkovi***. *Shard key* je polje (ili kombinacija polja) koje određuje na kom shardu će se nalaziti određeni dokument. Pravilno izabran shard key obezbeđuje ravnomernu distribuciju podataka, dobro performansno ponašanje i efikasno izvršavanje upita. Loš izbor *shard key*-a može dovesti do neujednačenog opterećenja shardova i sporog sistema.

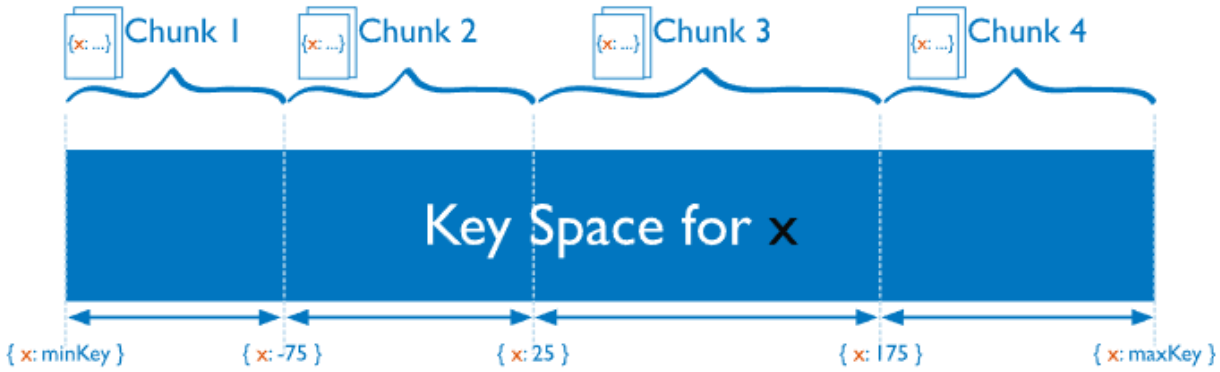
MongoDB može da raspoređuje podatke prema *range* principu (opsegu vrednosti) ili korišćenjem *hash*-iranog shard *key*-a. Range pristup je dobar za pretraživanje po opsezima, ali može dovesti do tzv. „vrućih“ shardova ako se vrednosti unose sekvencijalno. Hash pristup daje ravnomerniju raspodelu, ali otežava range upite.

Podaci se fizički ne raspodeljuju dokument-po-dokument, već se grupišu u veće logičke jedinice koje se nazivaju ***chunkovi***. Svaki chunk predstavlja određeni raspon vrednosti shard key-a. Tipična veličina chunk-a je oko 128 MB, a jedan *shard* može sadržati veći broj chunkova. Kako podaci rastu, *MongoDB* automatski deli prevelike chunkove (*split*), a kada neki shard ima više

chunkova nego drugi, balancer pokreće njihovu migraciju kako bi se postiglo ravnomerno opterećenje.

Migracija chunkova odvija se u pozadini i potpuno je transparentna za aplikaciju. Balancer kopira chunk na drugi shard, ažurira metapodatke na config serverima i briše staru kopiju, obezbeđujući da klaster ostane uravnotežen bez ručne intervencije administratora.

Upravo kombinacija shard *key*-a i *chunk* mehanizma omogućava *MongoDB*-u da efikasno distribuira podatke, skalira horizontalno i obezbedi stabilne performanse čak i kada baza raste do veoma velikih dimenzija.



Slika 6 Nacin na koji MongoDB deli kljucni opseg vrednosti na delove (chunks) I rasporedjuje in preko shardova

1.*Key space for x*

Ovo je ukupan opseg vrednosti shard ključa. U našem primeru se on zove x. Taj opseg ide od minKey do maxKey. Možemo zamisliti kao prava brojeva koja sadrži sve moguće vrednosti shard-key-a.

2.*Chunk-ovi (Chunk 1, Chunk 2, Chunk 3, Chunk 4)*

Ravni plavi blok se *seče* na više manjih celina – to su chunkovi**.**

MongoDB uobičajeno:

- drži *chunk* veličinu oko 128 MB (zavisi od verzije),
- svaki chunk pokriva određeni opseg ključeva.

U našem primeru *chunkovi* su razdvojeni na ovakve opsege:

1. ***Chunk 1*:**
   { x: minKey } → { x: -75 }
2. ***Chunk 2*:**
   { x: -75 } → { x: 25 }
3. ***Chunk 3*:**
   { x: 25 } → { x: 175 }
4. ***Chunk 4*:**
   { x: 175 } → { x: maxKey }

Kako se *chunkovi* koriste u *sharding*-u?

MongoDB pomoću ovih chunkova:

- raspoređuje podatke po *shard*-ovima (čvorovima),
- balansira opterećenje,
- premešta chunkove između shardova kada neki shard postane prepun.

Svaki dokument se smešta u chunk na osnovu **vrednosti *shard key*-a**.
Primer: ako dokument ima x = 10, upada u chunk gde je opseg:

{ x: -75 } → { x: 25 }

5. Strategije shardovanja

Hash-based sharding koristi hash funkciju nad vrednostima shard key-a kako bi podatke ravnomerno rasporedio između shard-ova. Ova metoda omogućava dobru balansiranost opterećenja, ali otežava izvršavanje upita po opsegu (*range queries*), jer susedne vrednosti shard key-a mogu biti fizički smeštene na različitim shard-ovima.

Range-based sharding deli podatke na osnovu opsega vrednosti shard *key*-a, što olakšava upite po rasponu, na primer pretrage po datumima ili ID-ovima. Međutim, postoji rizik od hot partition problema: ako određeni opseg vrednosti postane popularan, taj shard može biti preopterećen dok drugi ostaju skoro prazni.

Zoned sharding uvodi logičke ili geografske zone kako bi podaci bili raspoređeni prema lokaciji ili poslovnoj logici. Na primer, korisnici iz Evropske unije mogu biti smešteni na shard1, dok korisnici iz SAD-a idu na *shard2*. Ova strategija omogućava optimizaciju pristupa lokalnim podacima i bolje iskorišćenje mrežnih resursa.

Odabir strategije shardovanja zavisi od tipa aplikacije, obrasca pristupa podacima i željene ravnomernosti opterećenja, pa je ključna odluka pri dizajnu distribuirane baze.

6. Kako upiti rade u distribuiranom okruženju

U sharded okruženju, ***mongos*** funkcioniše kao router koji prima upit od klijenta i određuje na koje shard-ove upit treba da ide. Postoje dva osnovna tipa izvršenja upita: ***targeted*** i ***scatter-gather***.

***Targeted* upiti** se koriste kada vrednost shard *key*-a precizno identifikuje jedan *shard*. U tom slučaju, upit se šalje direktno na odgovarajući *shard*, što rezultira minimalnim opterećenjem i brzim izvršenjem.

***Scatter-gather* upiti** nastaju kada upit ne sadrži vrednost *shard key*-a ili zahteva opseg koji pokriva više shard-ova. *Mongos* tada šalje upit na sve shard-ove i agregira rezultate. Ovo povećava latenciju i mrežni saobraćaj, ali omogućava fleksibilnost kada precizna lokacija podataka nije poznata.

Kod ***sharded aggregation* pipeline**-a, deo agregacije može se izvršiti lokalno na svakom shard-u, a zatim se rezultati spajaju i finalizuju na *mongos*-u. Na ovaj način se smanjuje količina prenetih podataka i ubrzava obrada kompleksnih upita.

Razumevanje kako upiti funkcionišu u sharded okruženju je ključno za optimizaciju performansi, posebno u aplikacijama sa velikim brojem korisnika i masivnim količinama podataka.

7. Praktični primeri

U praktičnom delu ovog rada biće demonstriran rad distribuirane baze podataka na primeru *MongoDB* replica seta. Cilj je da se prikaže kako više instanci *MongoDB*-a može zajedno da funkcioniše, kako se podaci repliciraju između čvorova i kako sistem obezbeđuje visoku dostupnost.

Prvo, biće pokrenute tri lokalne instance *MongoDB*-a na različitim portovima, koje će činiti replica set. Zatim će se inicijalizovati replica set i odrediti koji čvor predstavlja *PRIMARY*, a koji *SECONDARY*. Nakon toga, na *PRIMARY* čvoru će se kreirati baza i kolekcija, i ubaciti testni dokumenti.

Praktični deo takođe uključuje demonstraciju replikacije – svi ubačeni podaci na PRIMARY čvoru automatski će se pojaviti i na *SECONDARY* čvorovima. Na kraju će biti pokazano ponašanje sistema u slučaju greške, tj. kada *PRIMARY* čvor prestane da radi, kako *SECONDARY* čvor preuzima ulogu PRIMARY i kako podaci ostaju dostupni.

7.1. Pokretanje instanci

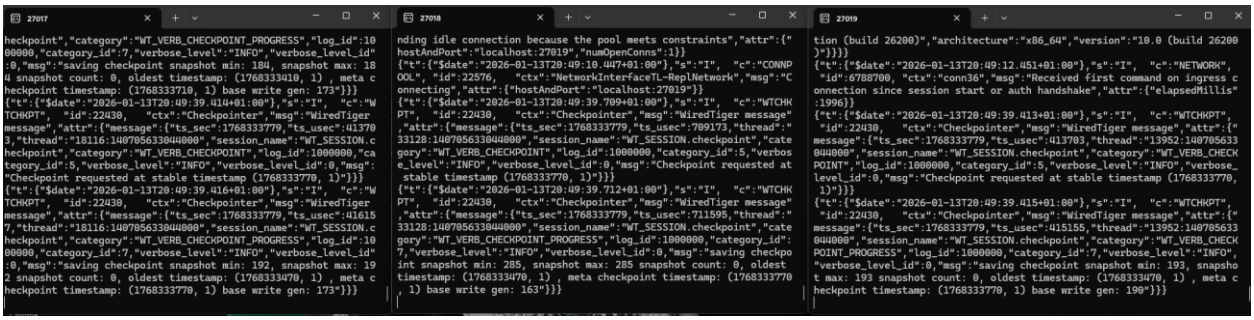
Prvi korak u implementaciji replica set arhitekture jeste pokretanje tri *MongoDB* instance, od kojih jedna ima ulogu *primary* čvora, dok preostale dve predstavljaju *secondary* čvorove. *Primary* instanca koristi port **27017**, dok se secondary instance pokreću na portovima **27018** i **27019**.

Kako bi bilo moguće istovremeno pokrenuti više *MongoDB* instanci na istoj mašini, neophodno je automatizovati proces pokretanja. U tu svrhu kreirana je ***PowerShell* skripta**, koja omogućava pokretanje tri zasebne mongod instance, pri čemu svaka instanca koristi sopstveni port i odgovarajući direktorijum za skladištenje podataka. Na ovaj način obezbeđuje se jasna razdvojenost čvorova unutar replica seta, kao i njihovo pravilno funkcionisanje u distribuiranom okruženju.

```powershell
# ===== CONFIG =====
$mongoBin = "C:\Program Files\MongoDB\Server\8.2\bin\mongod.exe"
$baseData = "C:\mongo-data"
$replSet  = "rs0"

$nodes = @(
    @{ port = 27017; path = "$baseData\rs0-27017" },
    @{ port = 27018; path = "$baseData\rs0-27018" },
    @{ port = 27019; path = "$baseData\rs0-27019" }
)

# ===== CREATE DATA DIRS =====
foreach ($n in $nodes) {
    if (!(Test-Path $n.path)) {
        New-Item -ItemType Directory -Path $n.path | Out-Null
    }
}

# ===== START NODES =====
foreach ($n in $nodes) {
    Start-Process `
        -FilePath $mongoBin `
        -ArgumentList "--port $($n.port) --dbpath `"$($n.path)`" --replSet $replSet" `
        -WindowStyle Normal
}

Write-Host "MongoDB replica set nodes started."
Write-Host "Connect with: mongosh --port 27017"
```

Slika 7 Powershell skripta za instanciranje 3 razlicite mongod instance

Nakon pokretanja 3 razlicite instance, to izgleda ovako.



Slika 8 Pokrenute istance na različitim portovima

Možemo primetiti da su nazivi otvorenih tabova promenjeni kako bi se u budućnosti omogućilo lakše snalaženje i jasna identifikacija svake pokrenute instance.

Nakon pokretanja svih *MongoDB* instanci, neophodno je izvršiti inicijalizaciju replica seta. Ovaj postupak se obavlja pomoću sledeće komande:

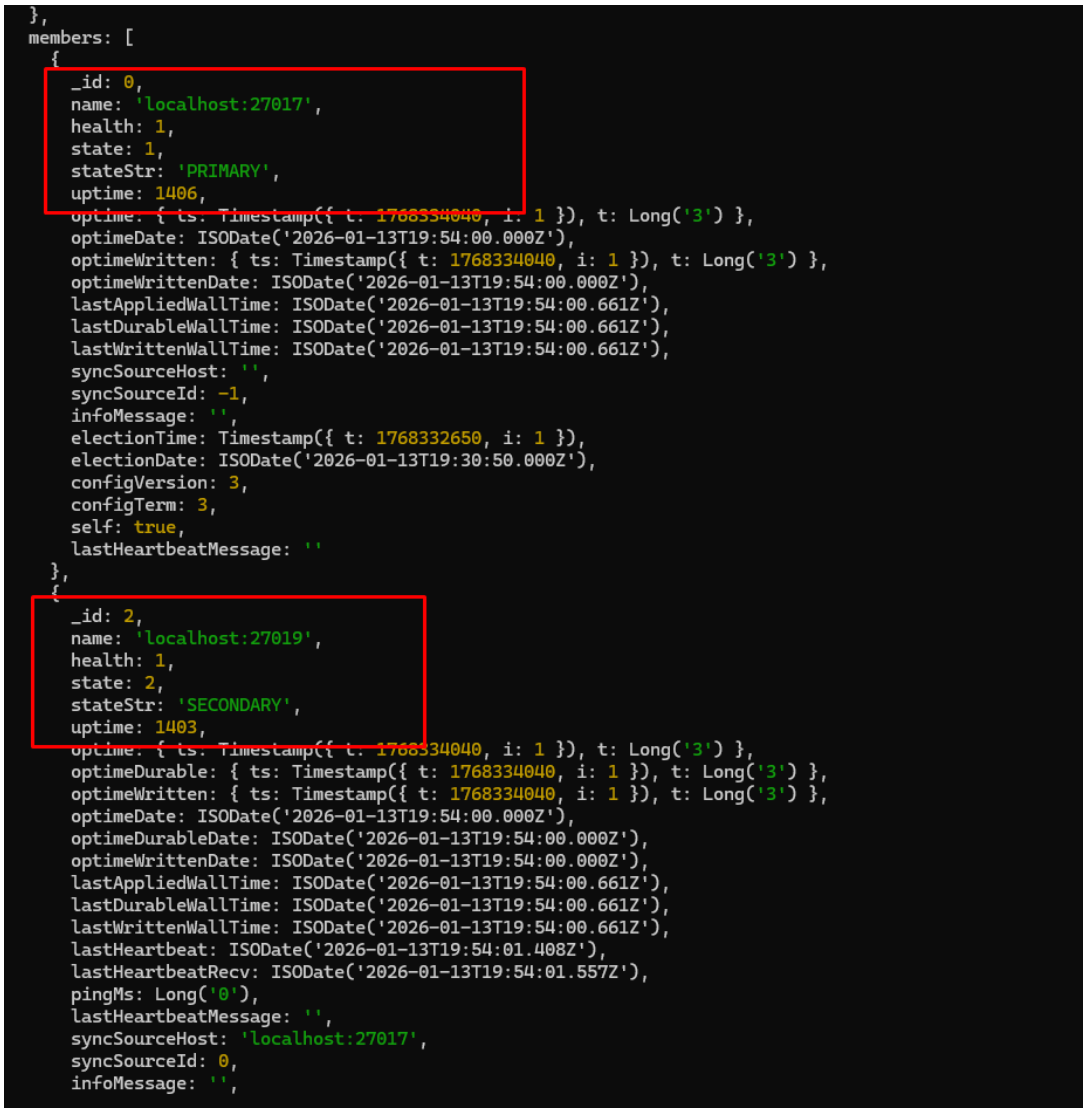
```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "localhost:27017" },
    { _id: 1, host: "localhost:27018" },
    { _id: 2, host: "localhost:27019" }
  ]
})
```

Navedenom komandom definiše se novi replica set sa nazivom **rs0**. Parametar _id predstavlja jedinstveni identifikator replica seta, dok se u okviru polja members navode svi čvorovi koji učestvuju u replica set arhitekturi.

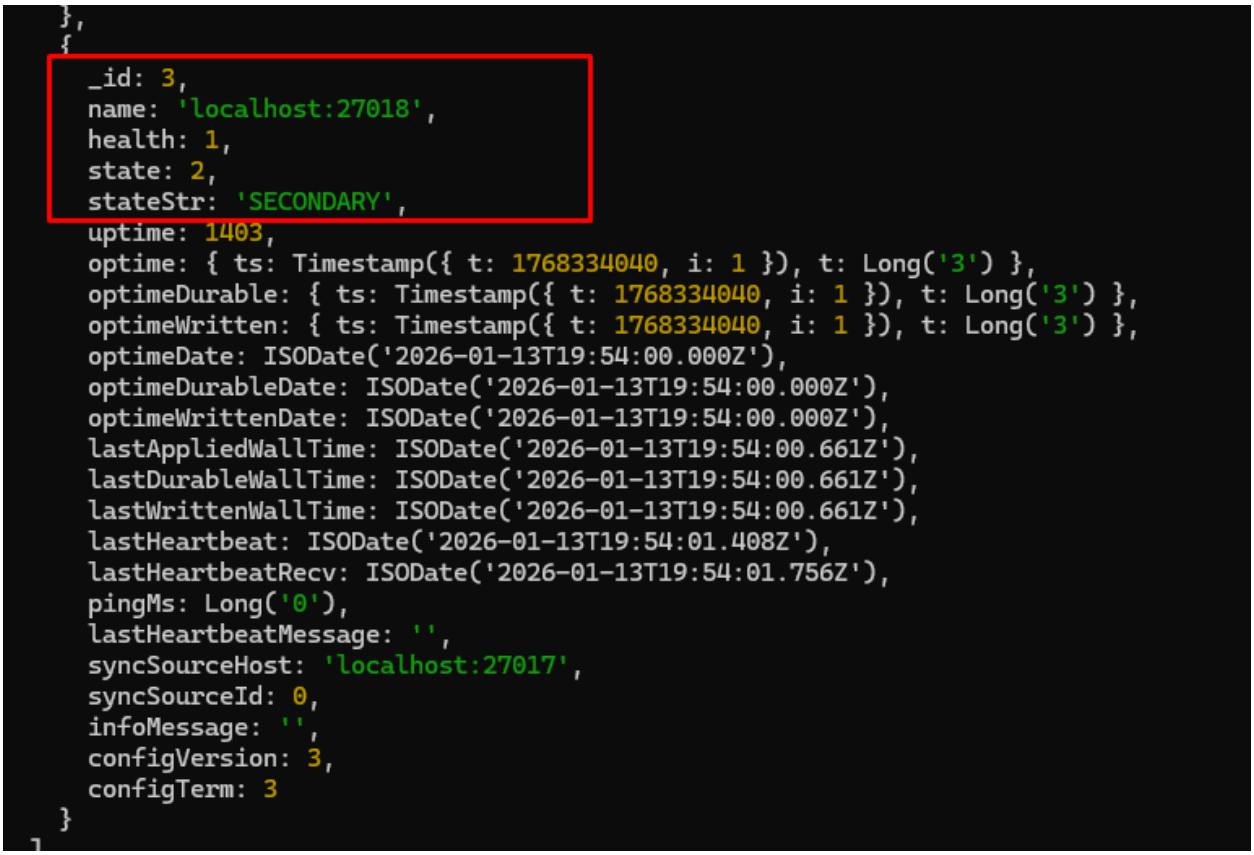
Svaki član replica seta ima sopstveni identifikator (*_id*) i adresu (*host*), koja se sastoji od naziva hosta i porta na kome je instanca pokrenuta. Prvi navedeni čvor (*localhost:27017*) inicijalno preuzima ulogu *primary* instance, dok se preostali čvorovi automatski konfigurišu kao *secondary* instance.

Kako bismo proverili da li su sve replike uspešno pokrenute i pravilno povezane unutar replica seta, potrebno je izvršiti sledeću komandu: *rs.status*()

Rezultat izvršene komande:



Slika 9 Primary i secondary instanca



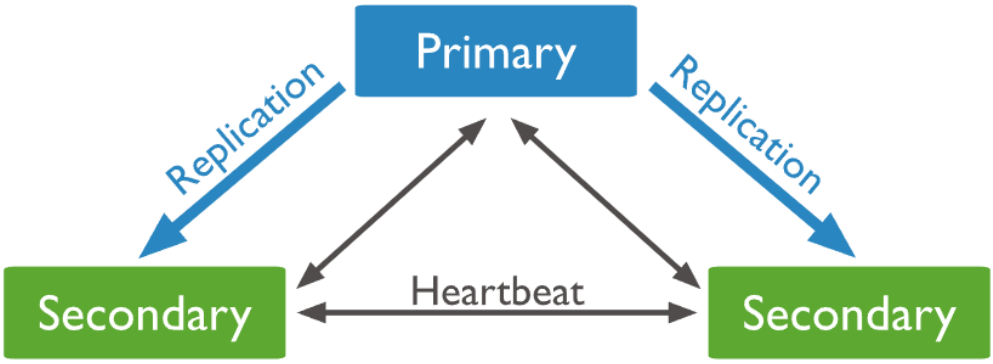
Slika 10 Poslednja secondary instanca

Na prikazanim slikama može se uočiti da su replike uspešno kreirane i pravilno konfigurisane. Instanca koja je pokrenuta na portu **27017** ima ulogu *primary* čvora, dok se instance na portovima **27018** i **27019** nalaze u ulozi *secondary* čvorova.

7.2. Demonstriranje principa replikacije

U nastavku rada demonstrira se princip replikacije podataka u okviru *MongoDB* replica seta. Podaci će se upisivati preko *primary* instance, dok će se njihovo čitanje obavljati sa *secondary* instanci, čime se jasno potvrđuje osnovni princip rada replica seta — svi podaci koji se upišu na primarni čvor automatski se i gotovo trenutno propagiraju na sve sekundarne čvorove.

Podsetimo se slike 2, na kojoj je prikazana struktura replica seta sa jednom *primary* i dve *secondary* instance. Ova ilustracija jasno prikazuje način na koji se podaci upisani na primarnoj instanci distribuiraju i sinhronizuju sa sekundarnim instancama.



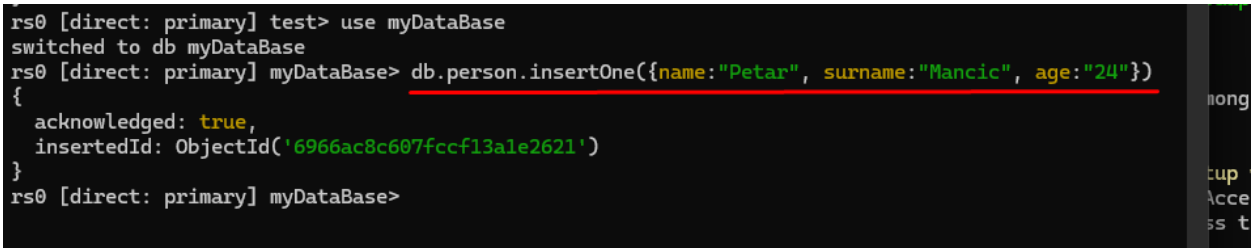
Slika 11 Replikacija

U sledećem koraku izvršićemo upisivanje podataka na primarnu instancu. Najpre je potrebno kreirati bazu podataka. U ovom primeru korišćena je baza podataka pod nazivom **myDataBase**.).

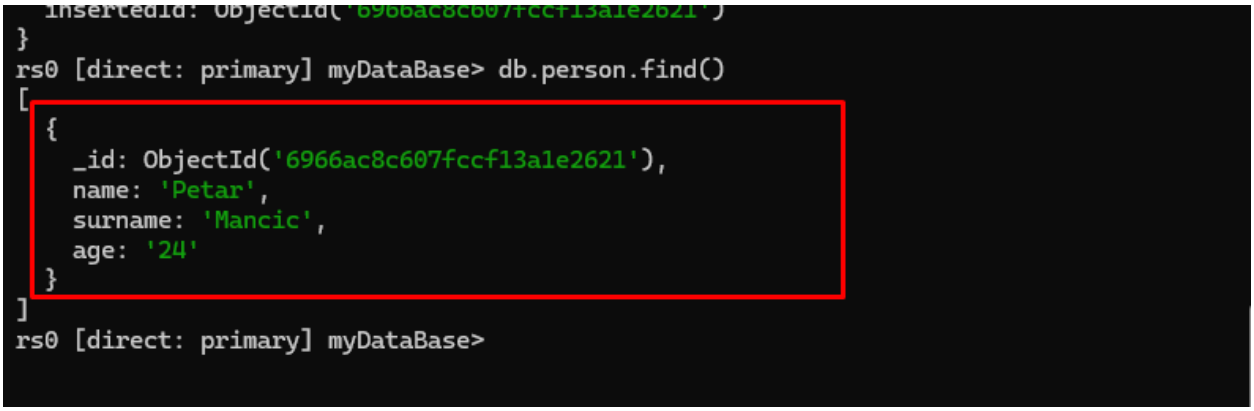
```
rs0 [direct: primary] test> use myDataBase
switched to db myDataBase
rs0 [direct: primary] myDataBase>
```

Slika 12 Kreiranje baze

Nakon kreiranja baze podataka, izvršeno je upisivanje jednog dokumenta u kolekciju **person**. Ova operacija se obavlja isključivo na *primary* instanci, budući da *MongoDB* dozvoljava upisivanje podataka samo na primarnom čvoru replica seta.



Slika 13 Insertovanje objekta u kolekciji person



Slika 14 Čitanje sa primary instance

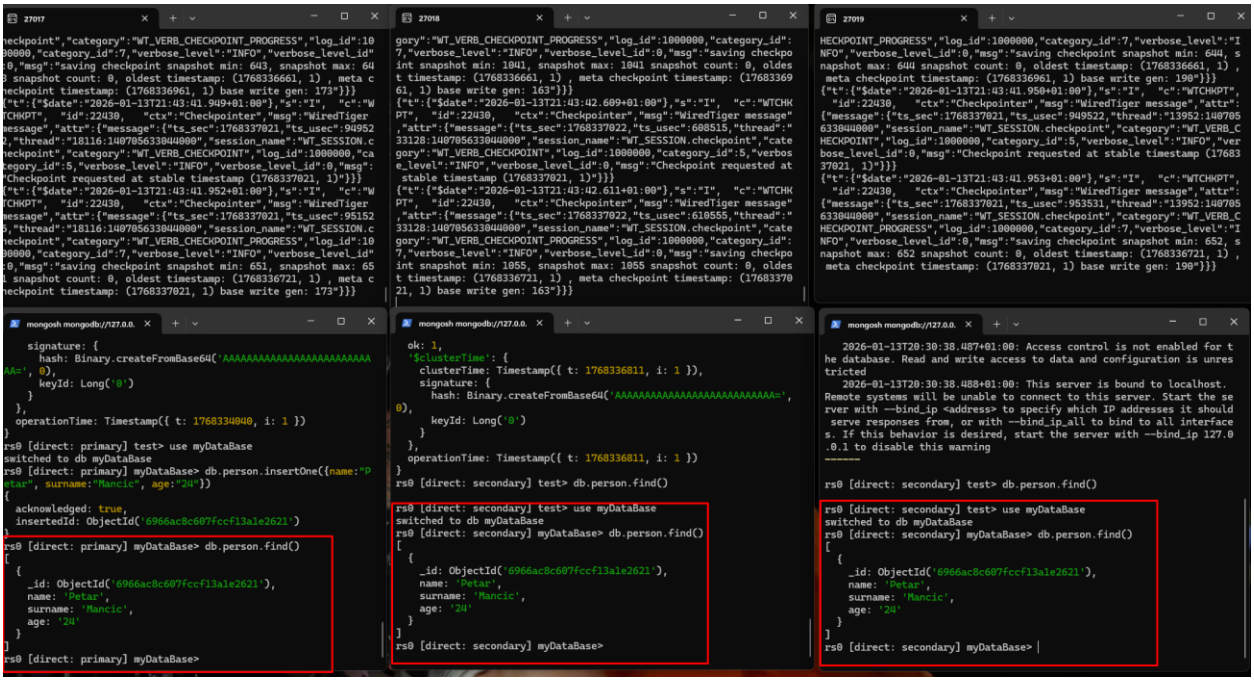
Kako bismo se dodatno uverili da je proces replikacije uspešno izvršen, isti podatak se zatim čita i sa preostale dve *secondary* instance.

```
rs0 [direct: secondary] test> use myDataBase
switched to db myDataBase
rs0 [direct: secondary] myDataBase> db.person.find()
[
  {
    _id: ObjectId('6966ac8c607fccf13a1e2621'),
    name: 'Petar',
    surname: 'Mancic',
    age: '24'
  }
]
rs0 [direct: secondary] myDataBase>
```

Slika 15 Čitanje sa secondary instance na portu 27018

```
rs0 [direct: secondary] test> use myDataBase
switched to db myDataBase
rs0 [direct: secondary] myDataBase> db.person.find()
[
  {
    _id: ObjectId('6966ac8c607fccf13a1e2621'),
    name: 'Petar',
    surname: 'Mancic',
    age: '24'
  }
]
rs0 [direct: secondary] myDataBase>
```

Slika 16 Čitanje sa secondary instance na portu 27019



Slika 17 Primary I secondary instance na jednoj slici

Dobijeni rezultati pokazuju da se dokument upisan na primarnoj instanci nalazi i na sekundarnim instancama, što potvrđuje ispravnost i efikasnost mehanizma replikacije podataka u okviru *MongoDB* replica seta.

7.3. Otpornost na greške

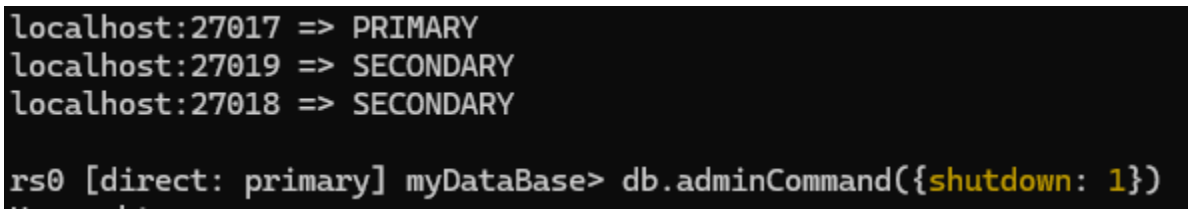
Nakon što je demonstriran mehanizam replikacije podataka, u nastavku rada fokus je stavljen na prikaz sposobnosti *MongoDB* sistema da obezbedi visoku dostupnost i otpornost na greške (*fault tolerance*). Ova osobina je od ključnog značaja u distribuiranim sistemima, jer omogućava kontinuitet rada čak i u slučaju otkaza pojedinih komponenti.

Kao što je prethodno objašnjeno, *MongoDB* replica set se sastoji od jedne *primary* instance i proizvoljnog broja *secondary* instanci. Izbori za novu *primary* instancu pokreću se isključivo u situacijama kada trenutna primarna instanca, iz bilo kog razloga, prestane sa radom. U slučaju otkaza *secondary* čvora, sistem nastavlja sa radom bez prekida, budući da *secondary* instance nemaju ulogu u obradi upisa podataka. Iz tog razloga, scenario otkaza sekundarne instance neće biti posebno razmatran u okviru ovog rada.

7.3.1. Izbor nove *primary* instance

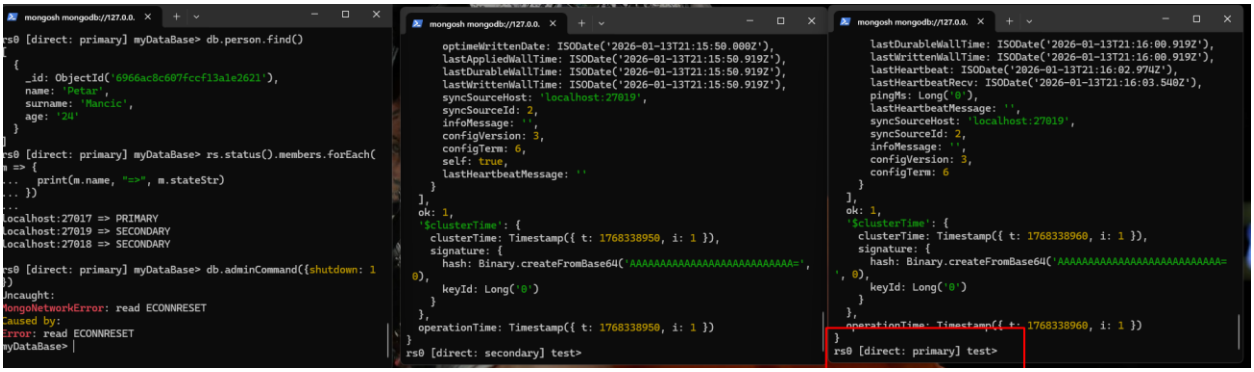
U narednom koraku izvršiće se namerno gašenje *primary* instance kako bi se posmatrao proces automatskog oporavka sistema. Posebna pažnja biće posvećena tome koja od preostalih *secondary* instanci će, kroz mehanizam izbora, preuzeti ulogu nove *primary* instance. Radi lakšeg praćenja eksperimenta, napominje se da se u trenutnoj konfiguraciji *primary* instanca nalazi na portu **27017**.

Gašenje *primary* instance:



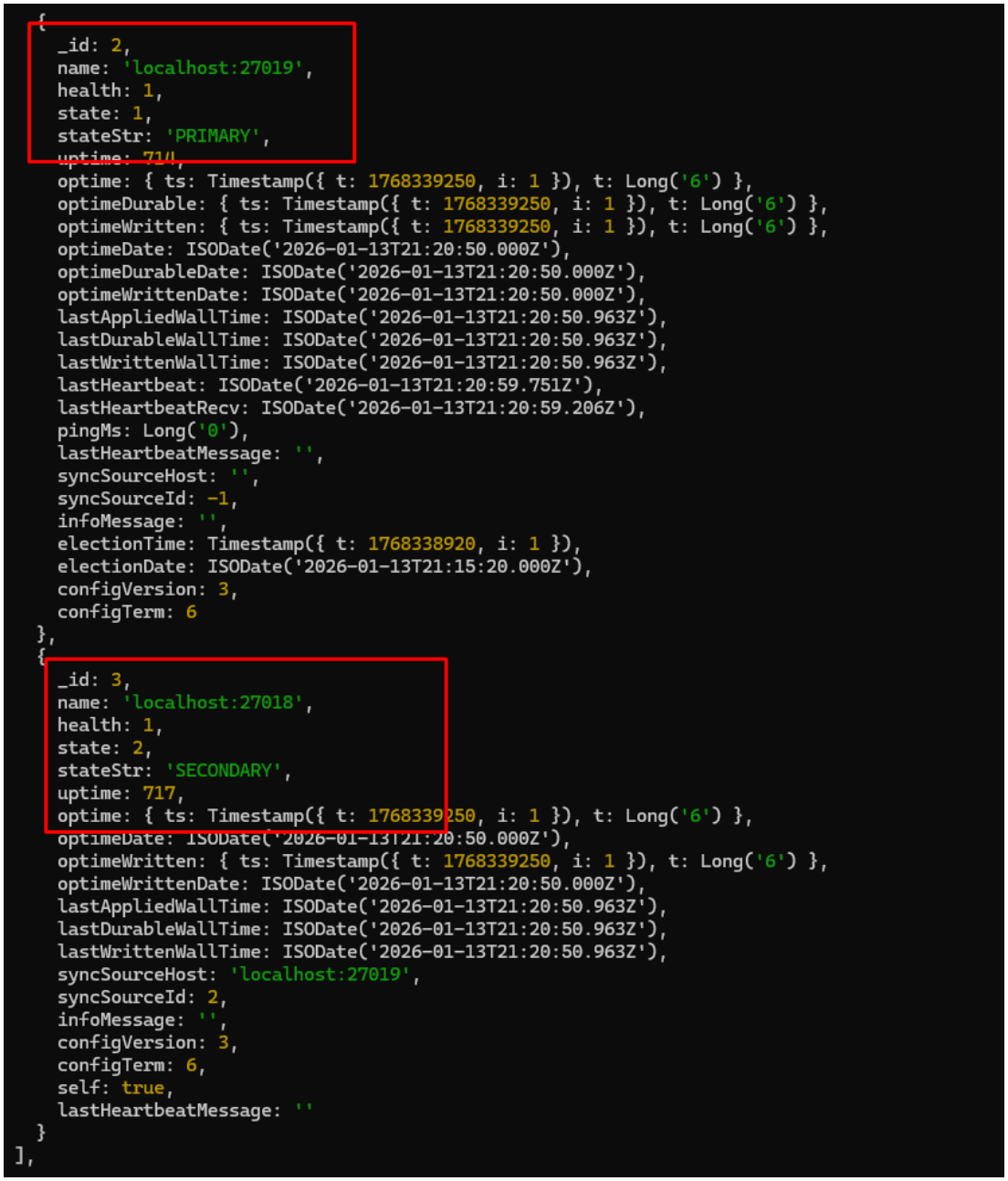
Slika 18 Gašenje primary instance

Nakon gašenja primarne instance, *MongoDB* replica set automatski pokreće proces izbora (*election process*) kako bi se obezbedio nastavak normalnog rada sistema. Preostale *secondary* instance međusobno komuniciraju i, na osnovu unapred definisanih kriterijuma, biraju novu *primary* instancu. Ovaj proces se odvija bez potrebe za ručnom intervencijom korisnika i obično traje svega nekoliko sekundi.



Slika 19 Izbor novog primary noda

U konkretnom primeru, jedna od *secondary* instanci uspešno je izabrana za novu *primary* instancu, dok je druga ostala u ulozi *secondary* čvora. Ovakvo ponašanje potvrđuje da *MongoDB* poseduje efikasan mehanizam automatskog oporavka, koji omogućava visoku dostupnost sistema čak i u slučaju otkaza ključnih komponenti.



Slika 20 Izbog novog primary noda

```
rs0 [direct: primary] myDataBase> rs.status().members.forEach(m => {
...   print(m.name, "=>", m.stateStr)
... })
...
localhost:27017 => (not reachable/healthy)
localhost:27019 => PRIMARY
localhost:27018 => SECONDARY

rs0 [direct: primary] myDataBase>
```

Slika 21 Provera novog primary noda

Proces izbora nove primarne instance zasniva se na više faktora, među kojima su najvažniji sinhronizovanost podataka, dostupnost čvora i njegova uloga u prethodnim izbornim procesima. Na taj način se osigurava da instanca koja preuzima ulogu *primary* čvora raspolaže najnovijim podacima i može bezbedno da nastavi obradu upisa.

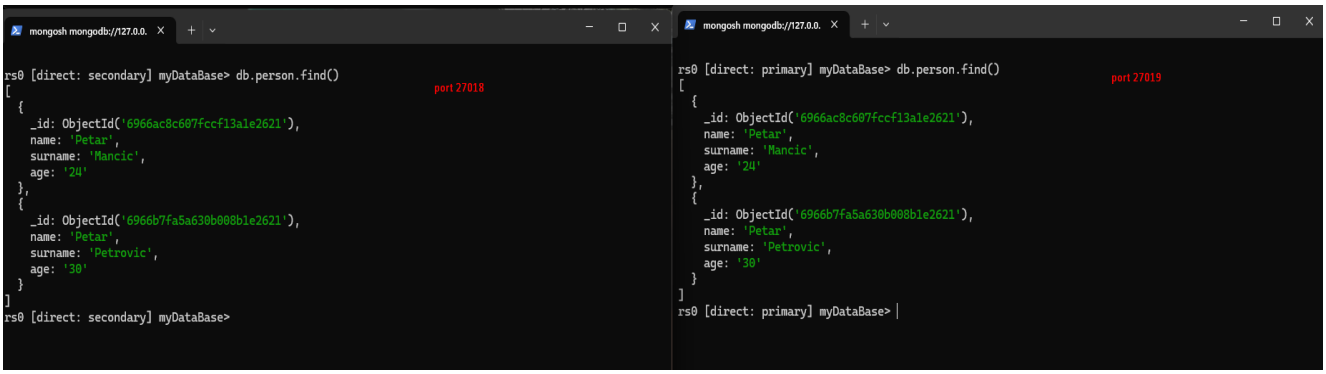
Na ovaj način demonstriran je osnovni princip otpornosti na greške (*fault tolerance*) u *MongoDB* sistemu. Čak i u slučaju potpunog prestanka rada primarne instance, sistem nastavlja sa funkcionisanjem uz minimalan prekid, čime se postiže visok stepen pouzdanosti i dostupnosti podataka.

Sledeće što ćemo uraditi jeste da ćemo izvršiti upis na novoj *primary* instanci i time se uveriti da je ceo sistem nastavio nesmetano da radi iako je doslo do gašenja prvobitne *primary* instance.

```
}
rs0 [direct: primary] test> use myDataBase
switched to db myDataBase
rs0 [direct: primary] myDataBase> db.person.insertOne({name:"Petar", surname:"Petrovic", age:"30"}
)
{
  acknowledged: true,
  insertedId: ObjectId('6966b7fa5a630b008b1e2621')
}
rs0 [direct: primary] myDataBase>
```

Slika 21 Insertovanje novog objekta preko nove primary instance

Nakon uspešnog upisa, izvršena je provera da li je novi podatak pravilno repliciran na *secondary* instanci, koja u ovom primeru koristi port **27018**.



Slika 22 Upoređivanje rezultata sa instanci na portu 27018 I portu 27019

Rezultati čitanja potvrđuju da se upisani objekat nalazi i na sekundarnoj instanci, što pokazuje da mehanizam replikacije nastavlja da funkcioniše ispravno i nakon promene primarnog čvora.

Na osnovu dobijenih rezultata može se zaključiti da *MongoDB* replica set obezbeđuje visok stepen dostupnosti i pouzdanosti sistema, čak i u situacijama kada dođe do otkaza primarne instance.

Na kraju je važno napomenuti i scenario u kome dolazi do gašenja poslednje preostale *primary* instance u okviru *MongoDB* replica seta. U tom slučaju, u sistemu ostaje samo jedna aktivna *secondary* instanca.

Usled nedostatka većine (*quorum*), izbori za novu *primary* instancu ne mogu biti održani. Kao posledica toga, preostala instanca ne može preuzeti primarnu ulogu, te sistem ne dozvoljava izvršavanje upisa podataka, čime se sprečava potencijalni gubitak ili nekonzistentnost podataka. Oporavak sistema moguć je tek nakon ponovnog pokretanja najmanje jednog dodatnog čvora, čime se ponovo uspostavljaju uslovi za održavanje izbora.

8. Zaključak

U ovom radu prikazan je princip rada *MongoDB replica set* arhitekture, sa posebnim fokusom na mehanizme replikacije podataka i otpornosti sistema na greške (*fault tolerance*). Kroz praktičan primer sa jednom *primary* i dve *secondary* instance, demonstrirano je kako se podaci upisuju na primarnoj instanci i automatski propagiraju na sekundarne instance, čime se obezbeđuje konzistentnost podataka i visoka dostupnost sistema.

Posebna pažnja posvećena je mehanizmu izbora (*election*), koji omogućava automatsko biranje nove *primary* instance u slučaju prestanka rada trenutne primarne instance. Na taj način *MongoDB* obezbeđuje kontinuitet rada sistema bez potrebe za manuelnom intervencijom, što je od ključnog značaja za pouzdanost distribuiranih sistema.

Takođe je razmotren i teorijski scenario u kome dolazi do gubitka većine čvorova u replica set-u, pri čemu sistem prelazi u režim ograničene funkcionalnosti i ne dozvoljava upis podataka, čime se sprečava nekonzistentnost i gubitak informacija. Ovaj mehanizam dodatno potvrđuje robusnost *MongoDB* arhitekture i njenu prilagođenost radu u realnim, promenljivim uslovima.

Na osnovu prikazanih primera i analize može se zaključiti da *MongoDB replica set* predstavlja efikasno i pouzdano rešenje za obezbeđivanje visoke dostupnosti, otpornosti na greške i sigurnog upravljanja podacima u savremenim distribuiranim aplikacijama.

9. Literatura

1. ***MongoDB* Documentation**, *Replication*, dostupno na: https://www.*MongoDB*.com/docs/manual/replication/.
2. ***MongoDB* Documentation**, *Sharding*, dostupno na: https://www.*MongoDB*.com/docs/manual/sharding/.
3. **Bradshaw, S., Brazil, E., & Chodorow, K.** (2019). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage* (3rd Edition). O'Reilly Media. (Poglavlja o replikaciji i shardingu).
4. **Pritchett, D.** (2008). *BASE: An Acid Alternative*, ACM Queue. (Kontekstualna literatura za distribuirane sisteme i eventualnu konzistentnost).
5. **Membrey, P., Plugge, E., & Hawkins, T.** (2014). *The Definitive Guide to MongoDB: A NoSQL Database for Cloud and Desktop Computing*. Apress.
6. ***MongoDB* University**, *M103: Basic Cluster Administration*, materijali sa kursa o konfiguraciji replica set-ova i sharded klastera.
7. **White, T.** (2015). *Hadoop: The Definitive Guide*. O'Reilly Media. (Za uporednu analizu distribuiranog skladištenja podataka).
8. **Zvanični primeri iz seminarskog rada** zasnovani na konfiguraciji primary i secondary instanci u lokalnom okruženju.