UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA RAČUNARSTVO

SEMINARSKI RAD

High Availability kod MongoDB baze podataka

Master akademske studije
Studijski program: Računarstvo i informatika
Modul: Bezbednost računarskih sistema
Predmet: Sistemi za upravljanje bazama podataka

Profesor:
Prof. dr Aleksandar Stanimirović

Student:
Teodora Kalezić, 1929

Niš, novembar 2025. god.

Sadržaj

1. UVOD

Dostupnost podataka predstavlja jedan od najvažnijih aspekata savremenih informacionih sistema. Svaka organizacija teži ka sistemima koji mogu neprekidno pružati usluge čak i u slučaju nepredviđenih grešaka, otkaza ili problema unutar mreže. Ova sposobnost sistema da nastavi sa radom i u uslovima delimičnog otkaza poznata je pod nazivom **High Availability** (**HA**).

U kontekstu baza podataka, High Availability obuhvata skup tehnika, pristupa i mehanizama pri organizaciji i realizaciji sistema koji obezbeđuju da podaci budu stalno dostupni korisnicima, uz minimalno vreme prekida rada. Postizanje visoke dostupnosti postaje ključni zahtev u dizajnu modernih sistema s obzirom da gubitak pristupa podacima direktno utiče na poslovno okruženje i korisničko iskustvo.

Tema rada „High Availability kod MongoDB baze podataka“ ima za cilj da prikaže načine na koje MongoDB, kao jedna od najpopularnijih NoSQL baza podataka, implementira HA principe kroz mehanizme **replikacije**, **Sharding** i **Failover** procesa. Poseban akcenat će biti stavljen na praktične aspekte implementacije Replica Set-a.

Nakon uvoda, u drugom poglavlju će biti objašnjeni osnovni High Availability koncepti, ključne metrike, kao i srodni pojmovi relevantni za baze podataka.

Treće poglavlje će biti posvećeno konkretnim tehnikama za postizanje visoke dostupnosti, uključujući **replikaciju**, formiranje **klastera**, **mehanizme oporavka**, kao i **ravnomernu raspodelu opterećenja**. Dodatno, biće spomenuti i izazovi koji se mogu pojaviti pri radu sa ovakvim sistemima.

U okviru četvrtog poglavlja će detaljnije biti razmotren način na koji MongoDB primenjuje navedene koncepte kroz Replica Set i Sharding arhitekturu.

Na kraju, u petom poglavlju, biće prikazani praktični primeri implementacije Replica Set-a i demonstracije srodnih koncepata.

Zaključak će biti dat u šestom poglavlju, dok će u sedmom poglavlju biti navedene korišćene reference.

2. HIGH AVAILABILITY - OSNOVNI KONCEPTI

[1] **High Availability** (u nastavku **HA**) je koncept koji se odnosi na visoku dostupnost sistema i može se primeniti na računarske mreže, konkretne servise, ali i baze podataka.
U kontekstu baza podataka, HA označava sistem koji ostaje dostupan i funkcionalan čak i prilikom pada pojedinih komponenti sistema (servera, mrežnih komponenti, segmenata i sl). Ključni pristup postizanja HA jeste eliminacija **jedinstvenih tačaka otkaza** (eng. **Single Point of Failure - SPOF**) uvođenjem redundanse na nivou hardvera i softvera.
Implementacija HA sistema podrazumeva uvođenje više mašina (servera), koje sarađuju kroz **replikaciju** (eng. **Replication**) i **balansiranje opterećenja** (eng. **Load Balancing**), formirajući **distribuirani sistem**. Individualna mašina u ovakvom sistemu se naziva **čvor** (eng. **Node**), dok se grupa povezanih čvorova koji rade zajedno kao logički sistem naziva **klaster** (eng. **Cluster**).

2.1 Metrike - Uptime, Downtime, SLA

[1] Kada se govori o metrikama koje se odnose na HA, najčešće se uvode pojmovi **Uptime**, kao i **Service Level Agreement** (**SLA**).
Uptime predstavlja meru pouzdanosti sistema, izraženu kao procenat vremena u kojem je sistem bio funkcionalan i dostupan korisnicima. U praksi se Uptime najčešće iskazuje na godišnjem nivou korišćenjem mere pouzdanosti poznate kao „**devetke**” (eng. **Nines**) ili „**klasa devetki**” (eng. **Class of Nines**).
Ova mera se odnosi na broj cifara **9** u procentu dostupnosti sistema. Na primer, sistem sa dostupnošću od 99.9% pripada klasi „tri devetke”. Sa druge strane, uvodi se i pojam **Downtime**, period u kojem sistem nije bio dostupan.
Vrednosti osnovnih klasa devetki i dozvoljenog Downtime-a su vidljive u tabeli 1:

Naziv	Procenat Uptime	Downtime u godini	Downtime u mesecu	Downtime u danu
Jedna devetka („One nine”)	90%	36,53 dana	73,05h	2,40h
Dve devetke („Two nines”)	99%	3,65 dana	7,31h	14,40min
Tri devetke („Three nines”)	99.9%	8,77h	43,83min	1,44min
Četiri devetke („Four nines”)	99.99%	52,60min	4,38min	8,64s
Pet devetki („Five nines”)	99.999%	5,26min	26,30s	864,00ms
Šest devetki („Six nines”)	99.9999%	31,56s	2,63s	86,40ms
Sedam devetki („Seven nines”)	99.99999%	3,16s	262,98ms	8,64ms
Osam devetki („Eight nines”)	99.999999%	315,58ms	26,30ms	864,00µs
Devet devetki („Nine nines”)	99.9999999%	31.56ms	2,63ms	86,40µs

Tabela 1 - Class of Nines

Na primer, sistem koji pripada klasi „tri devetke” (99.9% Uptime) bi mogao biti nedostupan maksimalno 8,77 sati tokom godine, što se smatra standardom za mnoge realne sisteme.
Takođe, vrednosti iznad klase „šest devetki” su retko garantovani u realnim sistemima i predstavljaju teorijske metrike.

SLA je formalni ugovor između pružaoca usluge i korisnika koji definiše očekivani nivo usluge, uključujući garantovani Uptime, često izražen u nekoj od klasi devetki i na nivou meseca. SLA obično sadrži minimalni procenat dostupnosti, izuzetke koji se ne računaju u Downtime, obaveze oko rešavanja incidenata i slično.

2.2 Single Point of Failure (SPOF)

Single Point of Failure (**SPOF**) predstavlja tačku, tj. hardverski, softverski, ili čak ljudski resurs u sistemu čiji prestanak rada onemogućuje rad čitavog sistema. U kontekstu baza podataka,

najčešće predstavlja mašinu na kojoj se baza podataka ili neka njena komponenta nalazi. Na primer, ukoliko se baza podataka nalazi na jednoj fizičkoj mašini, njeno otkazivanje bi dovelo do nedostupnosti čitavog sistema.
Cilj je eliminisati ovakve pojave u sistemu, a to se najčešće vrši uvođenjem **redundanse** koja podrazumeva dodavanje novih resursa. Konkretno, u bazama podataka se najčešće koristi tehnika **replikacije** (eng. **Replication**) koja podrazumeva uvođenje novih servera koji sadrže duplikat baze podataka.
Dodatne HA tehnike uključuju uvođenje **klastera** (eng. **Database Clustering**) i **balansiranje opterećenja** (eng. **Load Balancing**), o kojima će biti reči u okviru poglavlja 3. Tehnike za postizanje High Availability.

2.3 CAP teorema

Pre definisanja **CAP teoreme**, potrebno je napraviti osvrt na **ACID svojstva** baze podataka. Relacione baze podataka moraju poštovati sledeća svojstva pri radu sa transakcijama:

- **A - Atomicity** (**Atomičnost**):

Sve operacije unutar transakcije se moraju izvršiti kako bi transakcija bila uspešna. Ukoliko se u toku neke operacije pojavi greška, transakcija se obustavlja i sistem se vraća u poslednje validno stanje

- **C - Consistency** (**Konzistentnost**):

Transakcija mora dovesti sistem iz jednog konzistentnog stanja u drugo. Konzistentno stanje podrazumeva da su zadovoljena sva definisana pravila i protokoli u vidu ograničenja na nivou modela podataka, kao i u vidu poslovnih pravila

- **I - Isolation** (**Izolacija**):

Odnosi se na rad konkurentnih transakcija. Nijedna transakcija ne može imati pristup drugoj transakciji koja nije završena. Efekti jedne transakcije nisu vidljivi drugoj dok se ona ne završi

- **D - Durability** (**Trajnost**):

Kada se transakcija završi, izmene koje je napravila su trajne

Podrška za ACID svojstvima ima najveću ulogu u sistemima koji zahtevaju visok nivo pouzdanosti i tačnosti podataka. Međutim, strogo pridržavanje ovim svojstvima uvodi značajan **overhead**, naročito u distribuiranim okruženjima gde su čvorovi geografski udaljeni. Kao rezultat, vremenom je došlo do popuštanja zahteva ACID svojstava u korist veće skalabilnosti i otpornosti na otkaze, što je dovelo do nastanka **CAP teoreme**.
CAP teorema se odnosi na to da distribuirani sistem baza podataka može zadovoljiti najviše dve od sledeće tri stavke:

1. **C - Consistency** (**Konzistentnost**): Pri svakom čitanju podataka, sistem vraća najskorije upisane podatke ili grešku
2. **A - Availability** (**Dostupnost**): Svaki zahtev koji je prihvaćen od strane dostupnog čvora u distribuiranom sistemu mora vratiti odgovor

3. **P - Partition Tolerance** (**Otpornost na otkaze**): Sistem funkcioniše za operacije čitanja i upisa, čak i u slučaju kada ne postoji konekcija između pojedinih delova sistema

U zavisnosti od tipa sistema se biraju različiti nivoi konzistentnosti, dostupnosti i otpornosti na otkaze. U teoriji je moguće zadovoljiti sve zahteve, ali je jako skupo. Moguće su sledeće kombinacije prioriteta:

1. **CA** - Kompromis oko **P**:

Ovakav vid kompromisa se odnosi na single-site Cluster rešenja. Ukoliko nastupi narušavanje topologije mreže, sistem se blokira.

[2] Jedan od mehanizama koji se koristi u CA sistemima je **dvofazni Commit** (eng. **Two-phase Commit - 2PC**). Ovaj algoritam uvodi koordinatora transakcije koji komunicira sa svim čvorovima u sistemu koji sadrže kopije relevantnih podataka i pokreće proces njihovog ažuriranja. Nakon što svi čvorovi izvrše lokalne promene, oni obaveštavaju koordinatora o tome da li je operacija uspešna ili ne.

Ako koordinator dobije potvrdu da su svi čvorovi uspešno sproveli izmene, on daje završnu potvrdu kojom se transakcija primenjuje u čitavom sistemu. U suprotnom, ako neki od čvorova prijavi grešku, koordinator šalje **Rollback** instrukciju svim ostalim čvorovima kako bi poništio prethodno izvršene izmene. Kao rezultat se obezbeđuje konzistentno stanje sistema, kao i dostupnost dok je mreža stabilna. Međutim, u slučaju particionisanja mreže, tj. prekida komunikacije između čvorova, 2PC blokira transakcije jer koordinator ne može potvrditi da su svi čvorovi uskladili podatke

2. **CP** - Kompromis oko **A**:

Kada se pravi kompromis oko dostupnosti, pristup pojedinim podacima može biti privremeno onemogućen ili ograničen, dok se ostatak sistema nalazi u konzistentnom stanju.

Kao primer se može navesti Sharding, proces pri kom se vrši horizontalno particionisanje podataka na većem broju servera, o kome će biti reči u poglavlju 4.2 Sharding

3. **AP** - Kompromis oko **C**:

Rešenje koje se odnosi na kompromis oko konzistentnosti predstavlja najčešći pristup u distribuiranim sistemima. Sistem je dostupan i prilikom narušavanja topologije iako pojedini čvorovi sistema postaju nedostupni. Takođe, važi da različiti čvorovi mogu sadržati različite kopije podataka, te neki od podataka koje sistem vraća mogu biti privremeno neažurni. Kao posledica, potrebno je pokrenuti proces razrešavanja konflikata.

Reprezentativni primer jeste DNS, gde postoji više DNS servera, ali je potrebno vreme za sinhronizovanje podataka. Sistem kao celina nastavlja sa radom iako neki podaci mogu biti nedostupni u datom trenutku

[3] Nakon pojave CAP teoreme je uveden i termin **BASE**, koji se nadovezuje na CAP teoremu i nalaže NoSQL sistemima da bi veći prirotet trebalo dati dostupnosti umesto konzistentnosti. Akronim BASE se sastoji od sledećih pojmova:

- **BA - Basically Available**: Sistem generiše odgovor na svaki zahtev, korisnici ne moraju čekati kraj transakcije drugih korisnika kako bi se njihova operacija izvršila

- **S - Soft State**: Sistem se menja tokom vremena, čak i u slučaju kada nema ulaznih podataka zbog postojanja sledeće stavke o konzistentnosti
- **E - Eventually Consistent**: Sistem će u nekom trenutku dostići konzistentno stanje, te će se izmene vremenom replicirati na ostale kopije podataka. Čak i u slučaju kada više korisnika utiče na vrednost nekog resursa u isto vreme, njihove lokalne kopije se mogu razlikovati, ali se garantuje da će se u nekom trenutku njihove promene propagirati i sinhronizovati i dovesti do konzistentnog stanja

Za razliku od ACID svojstava, koji nalažu konzistentnost, BASE prihvata da sistem ne može uvek biti u konzistentnom stanju. Ova karakteristika je značajna kod distribuiranih NoSQL sistema i predstavlja čest vid implementacije baza podataka.

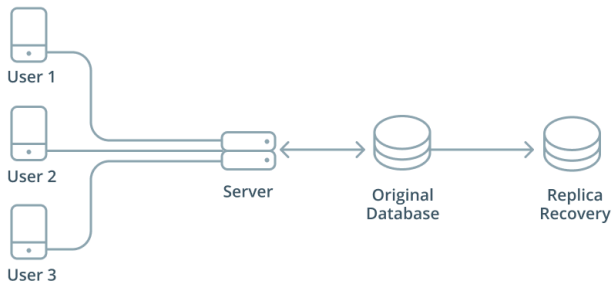
3. TEHNIKE ZA POSTIZANJE HIGH AVAILABILITY

Postizanje High Availability sistema zahteva primenu više tehnika koje zajedno eliminišu SPOF i obezbeđuju kontinuitet rada. Ove tehnike se mogu svrstati u sledeće kategorije:

- **Uvođenje većeg broja servera - replikacija** i **Database Clustering** obezbeđuju da podaci budu dostupni na više lokacija
- **Automatski oporavak - Failover** mehanizmi omogućuju automatsko prebacivanje na rezervne resurse u slučaju kvara
- **Distribucija opterećenja - Load Balancing** optimizuje iskorišćenost resursa i sprečava preopterećenje pojedinačnih komponenti sistema
- **Oporavak podataka i sistema** - ostvaruje se primenom **Backup** i **Disaster Recovery** mehanizama, respektivno

3.1 Replikacija

Osnovna tehnika postizanja HA sistema jeste **replikacija** (eng. **Replication**). Replikacija se odnosi na uvođenje više servera, gde svi sadrže istu bazu podataka, uključujući podatke, šemu, tabele, indekse i slično, ili samo njene podatke. U ovakvoj organizaciji se obično uvodi pojam **primarne baze podataka** (eng. **Primary database**) ili **primarni čvor**, gde se skladište originalni zapisi, dok svi ostali serveri predstavljaju **replike** (eng. **Replicas**) koji sadrže kopije zapisa i takođe se nazivaju **sekundarnim čvorovima**.
Replikacija podataka se vrši automatskim kopiranjem podataka iz primarne baze podataka u replike. Kao rezultat se uvodi veći nivo dostupnosti, skalabilnosti i otpornosti na greške u celokupnom sistemu. Na slici[1] 1 se može videti pojednostavljena šema replikacije:



Slika 1 - Replikacija

[1] Prisma - Introduction to database replication and its benefits:
https://www.prisma.io/dataguide/managing-databases/database-replication/database-replication-introduction

Iako uvođenje replikacije ima velike prednosti, ona dovodi i do višeg nivoa kompleksnosti zbog potrebe za sinhronizacijom svih čvorova u sistemu, gde može doći do pojave nekonzistentnosti podataka između replika. Dodatno, ovakva arhitektura može dovesti do pogoršanja performansi prilikom operacije upisa jer je potrebno izvršiti istu operaciju na svim replikama.

[4] Pored obezbeđivanja visoke dostupnosti, replikacija omogućuje i skaliranje performansi čitanja podataka. Kod većine aplikacija i sistema, češće se izvršavaju operacije čitanja. Uvođenjem više replika je moguće raspodeliti operacije čitanja na sekundarne čvorove, dok operacije upisa ostaju na primarnom čvoru. Ova tehnika, poznata kao Read/Write Separation, kombinuje replikaciju sa tehnikama balansiranja opterećenja i biće detaljnije obrađena u okviru poglavlja 3.4 Load Balancing.

3.2 Database Clustering

[5] **Database Clustering** predstavlja HA tehniku grupisanja više servera (čvorova) koji rade zajedno kao jedan logički sistem koji se naziva **klaster** (eng. **Cluster**). Za razliku od replikacije gde serveri rade nezavisno, u klasteru postoji koordinacija između čvorova pri obradi zahteva i deljenje resursa kao što je memorija.

Vrste klastera se mogu podeliti u sledeće kategorije, u zavisnosti od načina deljenja resursa:

- **Shared-Nothing** - svaki čvor u sistemu koristi svoje resurse, uključujući memoriju i procesorsku moć, i u potpunosti je izolovan od ostalih čvorova. Svaki čvor obrađuje poseban segment podataka
- **Shared-Disk** - svi čvorovi dele sekundarnu memoriju, ali imaju svoju procesorsku moć i primarnu memoriju. Čvorovi vide iste podatke, ali može doći do problema pri konkurentnim pristupima
- **Shared-Everything** - čvorovi dele sve resurse, uključujući sekundarnu memoriju

Dok se po aktivnosti čvorova mogu podeliti na:

- **Active-Active** - svi čvorovi aktivno obrađuju zahteve istovremeno, što omogućuje maksimalno iskorišćenje resursa i bolju distribuciju opterećenja
- **Active-Passive** - jedan čvor je aktivan i obrađuje sve zahteve, dok su ostali čvorovi u **Standby** režimu i preuzimaju obradu u slučaju otkaza aktivnog čvora. Ovaj pristup je jednostavnije implementirati, ali je iskorišćenost resursa manje efikasna

Uvođenje klastera se često kombinuje sa replikacijom i Load Balancing tehnikama za postizanje visokog nivoa dostupnosti i performansi.

3.3 Failover mehanizmi

Failover je proces koji predstavlja automatsko prebacivanje sa primarnih resursa na sekundarne, koji su uvedeni replikacijom i/ili formiranjem klastera. Ovaj proces pruža kontinutet rada sistema u slučaju otkaza primarne komponente. Ukoliko ovaj proces nije automatski i potrebna je intervencija, koristi se termin **Switchover** (manuelno prebacivanje).

Automatski Failover mehanizam je implementiran slanjem **Heartbeat** signala između povezanih mašina. Primarna mašina, koja je aktivna, u regularnim intervalima šalje signale koji označavaju „puls'' i obaveštavaju sekundarnu mašinu da je primarna još uvek funkcionalna. Ukoliko nastane prekid u slanju ovih signala, sekundarna mašina preuzima ulogu primarne, pretpostavljajući da je došlo do otkaza originalne primarne mašine.

3.4 Load Balancing

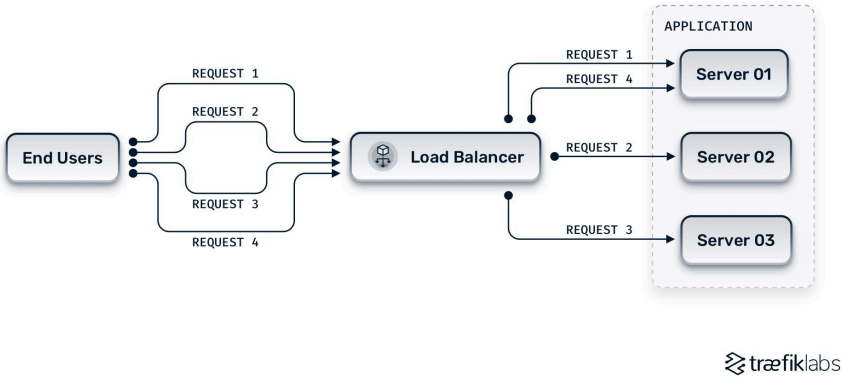
Tehnika kojom se dostiže HA u kombinaciji sa replikacijom i uvođenjem klastera je i **Load Balancing**, gde se vrši distribucija dolaznih zahteva na više servera u cilju optimizacije iskorišćenosti resursa, smanjenja latencije i sprečavanja preopterećenja pojedinačnih čvorova sistema.
Konkretno, replikacija i uvođenje klastera eliminišu pojavu SPOF, ali se dodavanjem Load Balancing tehnika eliminiše pojava „uskih grla'' u sistemu (eng. **Bottlenecks**), koji preuzimaju većinu opterećenja.

[6] Algoritmi koji se koriste za balansiranje opterećenja se mogu podeliti u dve kategorije:

- **Statički** - distribucija saobraćaja se vrši korišćenjem predeterminisanih pravila
- **Dinamički** - distribucija saobraćaja se vrši u skladu sa trenutnim opterećenjima

[7] Primer statičkog algoritma za balansiranje opterećenja je **Round Robin**, gde se zahtevi serverima dele ciklično. Prednost ovog pristupa je jednostavnost, dok je glavni nedostatak činjenica da se ne uzimaju u obzir individualna opterećenja servera u trenutku distribucije saobraćaja. Na slici[2] 2 se može videti demonstracija ovakvog pristupa:

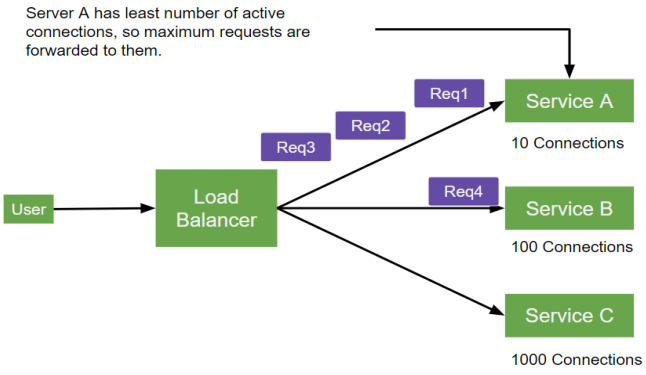
[2] Traefik - Everything You Need to Know About Round Robin Load Balancing: https://traefik.io/glossary/round-robin-load-balancing



Slika 2 - Round Robin Load Balancing

U cilju poboljšanja algoritma se može uvesti i dodavanje „težine" svakom serveru u skladu sa kapacitetima, gde se balansiranje opterećenja može optimizovati, posebno u slučajevima kada svi serveri nisu istog kapaciteta. Ovaj algoritam se naziva **Weighted Round Robin**.

Primer dinamičkog algoritma je **Least Connections**, gde se raspodela zahteva odlučuje na osnovu toga koji server u tom trenutku ima najmanje aktivnih konekcija. Ovim se eliminiše glavni nedostatak statičkih algoritama, ali je pretpostavka da su svi zahtevi podjednako zahtevni. Na slici[3] 3 se može videti primer ovakvog balansiranja opterećenja:



Slika 3 - Least Connections Load Balancing

Na sličan način se kao i kod Round Robin algoritma može uvesti i **Weighted Least Connections**, gde je moguće svakom serveru pripisati drugačiju „težinu" u skladu s tim koliko konekcija može prihvatiti.

[3] GeeksForGeeks - Network Load Balancing: Round Robin vs Least Connections: https://www.geeksforgeeks.org/computer-networks/network-load-balancing-round-robin-vs-least-connections/

Kombinovanjem replikacije i Load Balancing tehnika se omogućuje Read/Write Separation strategija. U ovom pristupu, operacije upisa izvršava primarni čvor, dok se operacije čitanja distribuiraju na sekundarne čvorove pomoću Load Balancing mehanizama. Ovakav pristup je efikasan u aplikacijama gde su operacije čitanja znatno češće, te se opterećenje primarnog čvora smanjuje i omogućuje horizontalno skaliranje performansi pri operacijama čitanja dodavanjem novih replika. Nedostatak Read/Write separacije je potencijalno kašnjenje pri repliciranju podataka, što može dovesti do čitanja zastarelih podataka.

3.5 Backup i Disaster Recovery

Backup predstavlja kreiranje kopije postojećih podataka u cilju zaštite od gubitka koji može nastupiti zbog hardverskog kvara, ljudske greške, napada ili drugih događaja. Za razliku od replikacije koja omogućuje kontinuitet rada sistema u realnom vremenu prelaskom na replike u slučaju otkaza, gde replike sadrže trenutno aktuelne podatke, Backup omogućuje vraćanje podataka iz određenog trenutka u prošlosti, tzv. **Point-in-time Recovery**. Backup kopije se kreiraju periodično i ne moraju biti sinhronizovane u realnom vremenu.

[8] Metode Backup-a se mogu podeliti na sledeće:

- **Full Backup -** Full Backup kreira kopiju čitave baze podataka i omogućuje potpun oporavak podataka u slučaju gubitka. Od svih metoda se izvršava najređe zbog memorijskih zahteva kao i potrebnog vremena za kopiranje. Sa druge strane, proces obnove podataka je najbrži i najjednostavniji
- **Incremental Backup -**Incremental Backup je metoda kojom se čuvaju samo promene nastale nakon poslednjeg Backup-a, bilo Full ili Incremental, te zahteva manje vremena i prostora za obradu jer se samo čuvaju modifikacije, tj. **inkrementi**. Vreme obnove podataka je sporije od Full Backup-a, jer je najpre potrebno obaviti Full Backup, a zatim sve prethodno kreirane inkremente redom
- **Differential Backup -** Za razliku od čuvanja inkremenata, Differential Backup predstavlja kompromis između prethodne dve metode jer čuva promene kreirane nakon Full Backup-a. Kao posledica, proces obnove je znatno jednostavniji jer je potrebno pokrenuti poslednje kreiran Full Backup, a zatim najnoviji Differential Backup

Frekvencija obavljanja Backup-a zavisi od važnosti i dinamičnosti podataka, kao i od korišćene metode. Na primer, kreiranje vremenski i memorijski zahtevnih kopija korišćenjem Full Backup metode se obično obavlja na nedeljnom ili mesečnom nivou, dok se Incremental ili Differential izvršavaju dnevno ili čak više puta dnevno.

Industrijski standard za Backup strategije je **3-2-1 pravilo**, koje nalaže sledeće:

- Potrebno je napraviti **3** kopije podataka, tj. originalni i dva Backup-a

- Korišćenjem **2** različita medija skladištenja, neki od kojih su korišćenje diskova, cloud rešenja i sl.
- Gde je potrebno da **1** kopija bude Off-site, tj. na drugoj lokaciji kao mera zaštite od fizičkih katastrofa kao što su poplave, požari, itd.

Dok Backup služi za očuvanje podataka, **Disaster Recovery** obuhvata strategiju oporavka celokupnog sistema nakon incidenta, bilo da je u pitanju hardverski kvar, napad na infrastrukturu ili prirodna katastrofa. Cilj je kreirati plan koji će vratiti sistem u funkcionalno stanje što je brže moguće, uz minimalan prekid rada i gubitak podataka. U tom kontekstu se uvode sledeće metrike:

- **RPO** (**Recovery Point Objective**) - Označava maksimalnu količinu podataka čiji se gubitak može tolerisati. izraženu kroz vremenski period. Na primer, RPO od jedan sat znači da sistem može tolerisati gubitak podataka nastalih u poslednjem satu. Manji RPO podrazumeva češće Backup-ove i bolju zaštitu. Osim periodičnih Backup kopija, moguće je koristiti **Continuous Data Protection** (**CDP**) strategiju koja beleži sve promene u realnom vremenu, čime se RPO skoro eliminiše, tj. približava nuli
- **RTO** (**Recovery Time Objective**) - Odnosi se na maksimalno prihvatljivo vreme oporavka sistema nakon incidenta. Manji RTO označava brži povratak sistema

U idealnom slučaju bi RPO i RTO trebalo da budu što manji, ali se na taj način povećavaju troškovi implementacije i održavanja sistema.
Efikasan Disaster Recovery plan obuhvata definisane procedure za aktiviranje rezervnih komponenti sistema, automatizovane Failover mehanizme, kao i redovno testiranje i reviziju čitavog procesa oporavka.

3.6 Izazovi

Prilikom projektovanja sistema visoke dostupnosti, potrebno je naći balans između performansi, konzistentnosti, kao i otpornosti na greške. U realnim distribuiranim sistemima su česte pojave hardverskih kvarova, mrežnih prekida, kašnjanja u komunikaciji, kao i neuspešne sinhronizacije između čvorova. Ovi problemi mogu značajno uticati na dostupnost i konzistentnost podataka. U nastavku su opisani ključni izazovi prilikom rada sa HA sistemima:

- **Network Partitioning (Mrežno particionisanje)**

Mrežno particionisanje nastaje kao posledica prekida u komunikaciji između delova distribuiranog sistema, ali svaka kreirana particija (segment u mreži) nastavlja da funkcioniše nezavisno od drugih. Za razliku od potpunog prekida rada čvorova, mrežna particija kreira situaciju kada čvorovi ne znaju da li je problem u mreži ili su drugi čvorovi zaista prestali sa radom. Neki od uzroka particionisanja mogu biti fizički prekidi (npr. oštećenje kablova i prekid rada mrežne opreme), preopterećenost mreže koja dovodi do gubitka paketa, izmene Firewall

pravila, kao i geografska udaljenost između Data centara. Kao posledice particionisanja se mogu javiti sledeći problemi:

- **Divergence** - Ukoliko particije nezavisno prihvataju različite operacije upisa, dolazi do neusaglašenosti podataka, koje je potrebno rešiti nakon oporavka mreže radi postizanja konzistentnog stanja sistema
- **Stale Reads** - Zastarele informacije pri čitanju gde se pojavljuju različite verzije istih podataka u zavisnosti od toga koji čvor (i mrežni segment) se kontaktira
- **Privremena nedostupnost sistema** - Ukoliko sistem implementira striktnu konzistentost, može odbiti sve operacije tokom particionisanja

Kao strategija za toleranciju na particionisanje se mogu koristiti **Quorum-based tehnike**, koje zahtevaju dozvolu od većine čvorova za bilo koju operaciju. Kao posledica se garantuje da samo jedna particija može nastaviti sa radom. Osim Quorum-based sistema se mogu koristiti i **Consesus algoritmi** koji automatski detektuju particije i biraju novi glavni čvor samo ako postoji **quorum**, tj. minimalan broj glasova za izvršenje odabira novog glavnog čvora.

- **Split-brain Problem**

[9] **Split-brain** problem je potencijalna posledica particionisanja mreže gde više delova sistema istovremeno smatraju da imaju kontrolu nad sistemom. Konkretno, moguće je da sve particije izvrše selekciju glavnog čvora, te i da izvršavaju različite akcije nad svojim skupom podataka gde kao posledica dolazi do nekonzistentnog stanja. Kada je potrebno obaviti oporavak sistema, potencijalno je potrebno poništiti neke od izvršenih transakcija i da rešavanje konflikata bude nemoguće.
Kao i u slučaju particionisanja mreže, ovaj problem je moguće sprečiti korišćenjem Quorum-based tehnika i Consensus algoritama.

- **Latencija**

U distribuiranim sistemima može značajno biti povećano kašnjenje u komunikaciji, posebno u slučajevima kada postoji veća geografska udaljenost između komponenti, koje se naziva **latencija**. Osim fizičke udaljenosti, kašnjenje može biti posledica operacija **replikacije**, kao što je propagacija efekata uspešne operacije upisa na sve čvorove sistema. Dodatno, ukoliko se vrše operacije čitanja i upisa nad **žurnalom** (eng. **Journal**) koji se nalazi u sekundarnoj memoriji, strukturi u kojoj se beleže operacije upisa.
Načini na koje je moguće smanjiti latenciju jeste korišćenje keširanja, **Batch** operacija koje izvršavaju više upita odjednom, kao i **Connection Pooling** mehanizma kojim se izbegava TCP handshake overhead tako što se koriste prethodno kreirane TCP koncekcije.

- **Data Consistency i Availability Trade-offs**

Prema CAP teoremi, realni sistemi ne mogu istovremeno garantovati potpunu konzistentnost, dostupnost i otpornost na otkaze, te je potrebno naći kompromis, najčešće između konzistentnost i dostupnosti podataka, u zavisnosti od tipa sistema.

Opisani izazovi nisu potpuno rešivi problemi, već predstavljaju ograničenja distribuiranih sistema. Svaki HA sistem mora doneti odluke o tome kako će razrešiti navedene situacije, u skladu sa zahtevima sistema.

4. HIGH AVAILABILITY U KONTEKSTU MongoDB-a

MongoDB je jedna od najpopularnijih NoSQL baza podataka i spada u kategoriju **Document-oriented** baza podataka. Za razliku od relacionih baza podataka koje koriste tabele sa fiksnom šemom, MongoDB omogućuje definisanje fleksibilne strukture podataka korišćenjem **dokumenata** (eng. **Document**) u BSON (Binary JSON) formatu. Svaki dokument predstavlja jednu instancu podataka, a više dokumenata se organizuje unutar **kolekcije** (eng. **Collection**), pandan tabeli u relacionim bazama podataka.

[10] Document-oriented baze podataka generalno imaju ugrađenu podršku za distribuciju podataka i omogućuju **Scaling Out**, tj. dodavanje novih čvorova u sistem. MongoDB to omogućuje kroz sledeće mehanizme:

1. **Replikacija** - korišćenjem skupova replika (eng. **Replica Sets**), što utiče na dostupnost i pouzdanost sistema kroz redundansu. U ovu svrhu se uvode **primarni** i **sekundarni** čvorovi, kao i automatski **Failover** pri prestanku rada primarnog čvora, gde jedan od sekundarnih čvorova preuzima ulogu. Dodatna prednost koju uvodi replikacija je distribucija operacija čitanja na sekundarne čvorove
2. **Sharding** - cilj ovog mehanizma je horizontalno particionisanje gde svaki čvor sadrži deo podataka umesto da svi podaci budu skladišteni na jednom čvoru. Konkretno, particionisanje podataka se vrši tako da svaka individualna komponenta, tzv. **Shard**, sadrži različit podskup podataka. Dodatno, svaki Shard je implementiran kao Replica Set

Procesu replikacije će biti posvećeno poglavlje 4.1 Replica Sets, dok će Sharding biti detaljnije opisan u poglavlju 4.2 Sharding.

4.1 Replica Sets

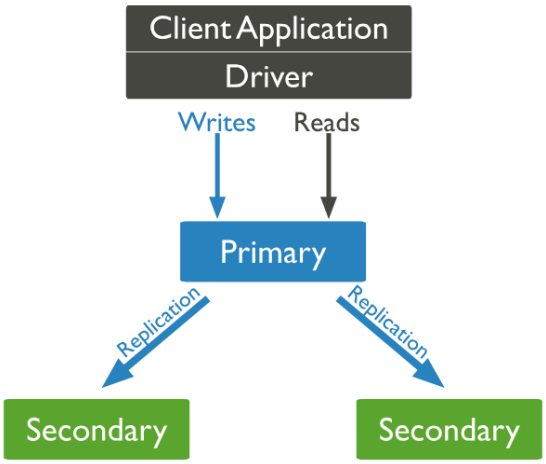
Replica Set u MongoDB-u je grupa **mongod** procesa koji održavaju isti skup podataka i predstavljaju sredstvo za dostizanje redundanse i HA. Jedna ovakva struktura se sastoji iz više čvorova koji skladište podatke, i eventualno jednog čvora koji se naziva **arbitar** (eng. **Arbiter Node**). Jedan od čvorova koji sadrži podatke ima ulogu **primarnog**, dok su ostali čvorovi **sekundarni**. Dodatno, čvor može pripadati samo jednom Replica Set-u.

Preporuka za minimalnu konfiguraciju Replica Set-a je postojanje tri čvora koja sadrže podatke, od kojih je jedan primarni i dva su sekundarni čvorovi. U pojedinim slučajevima je umesto dva sekundarna čvora moguće dodati jedan, kao i jedan arbitar koji učestvuje u procesu glasanja (eng. **Election Process**), ali ne sadrži podatke.

Jedan Replica Set može sadržati maksimalno 50 čvorova, od kojih njih 7 učestvuje u procesu glasanja.
U nastavku će najpre biti opisane uloge primarnih, sekundarnih, kao i arbitar čvorova, a zatim sam proces glasanja i selekcije novog primarnog čvora.

4.1.1 Primary, Secondary, Arbiter čvorovi

[11, 12] **Primarni čvor** je jedini koji preuzima sve operacije upisa i beleži sve izmene nad svojim skupom podataka unutar strukture koja se naziva **oplog** (**Operation Log**), koja će biti detaljnije opisana u poglavlju 4.1.3 Oplog i sinhronizacija. Sekundarni čvorovi repliciraju ovaj log i primenjuju operacije nad svojim skupom podataka. Opisana struktura se može videti na slici[4] 4:



Slika 4 - Organizacija čvorova u MongoDB-u

Osim operacija upisa, podrazumevano ponašanje je da primarni čvor prihvata sve operacije čitanja. Ovo je moguće promeniti tako da sekundarni čvorovi preuzmu ulogu čitanja, što će biti opisano u poglavlju 4.1.4 Operacije upisa i čitanja.

Replica Set može sadržati maksimalno jedan primarni čvor. Ukoliko on prestane sa radom, pokreće se proces selekcije kako bi se odabrao novi primarni čvor od postojećih sekundarnih. Ovaj proces će detaljnije biti opisan u poglavlju 4.1.2 Election proces.

[13] Uloga **sekundarnih čvorova** je da sadrže kopije skupa podataka primarnog servera korišćenjem **oplog** strukture primarnog čvora. Ovaj proces je asinhron, što znači da je moguće da u nekom trenutku nijedan od sekundarnih čvorova ne sadrži replike.

[4] MongoDB - Replication: https://www.mongodb.com/resources/products/capabilities/replication

Prilikom konfiguracije sekundarnih čvorova, moguće je dodeliti im sledeće uloge:

- [14] **Prioritet 0** - označava da sekundarni čvor neće biti kandidat za novog primarnog čvora prilikom procesa selekcije. Takođe, oni ne mogu pokrenuti ovaj proces, ali mogu učestvovati u glasanju. Ova uloga se obično koristi za udaljene čvorove koji imaju visoku latenciju, kao i za **Standby** čvorove, koji čuvaju replike i imaju ulogu da zamene čvor koji postane nedostupan
- [15] **Hidden member (Sakriven čvor)** - čvor koji neće biti vidljiv klijentskoj aplikaciji i uvek mora imati prioritet 0 opisan u prethodnoj stavci, te ne mogu postati kandidat za primarnog čvora. Kao i čvorovi sa prioritetom 0, oni mogu učestvovati u glasanju pri procesu selekcije. Jedina uloga im je čuvanje replike podataka. Kao rezultat, ne mogu se izvršavati operacije nad njima, osim ako se ne izvrši direktno povezivanje na ovakav čvor. Hidden čvorovi se obično koriste za Backup, tj. čuvanje rezervnih kopija, kao i generisanje analitičkih izveštaja
- [16] **Delayed Replica Set Member** - sadrže istorijske Snapshot-ove, tj. stanje podataka u nekom trenutku u prošlosti jer uvode namerna kašnjenja u repliciranju operacija iz **oplog** strukture. Ova uloga se koristi kada je potrebno izvršiti povratak na prethodno stanje u slučajevima greške, kao što je nenamerno brisanje podataka. Da bi čvor imao ovu ulogu, neophodno je da ima prioritet 0, da je sakriven (Hidden), kao i da nema mogućnost glasanja (**Non-voting**)

[17] U slučajevima kada je uvođenje dodatnog sekundarnog čvora skupo, moguće je uvesti **arbitar** čvor. Njegova uloga je učestvovanje u procesu selekcije novog primarnog čvora, ali ne sadrži kopiju podataka. Kao rezultat, ne može biti kandidat za primarnog čvora.
Preporučuje se korišćenje najviše jednog arbitra kako bi se izbegli problemi sa konzistentnošću podataka jer više njih otežava pouzdanu primenu mehanizma upisa na osnovu glasa većine, tzv. Majority Write Concern, koji će biti opisan u poglavlju 4.1.4 Operacije upisa i čitanja.

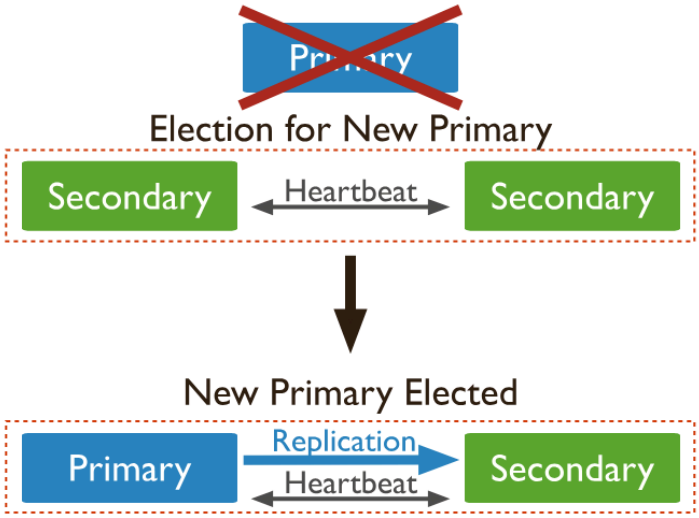
4.1.2 Election proces

[18] Election proces je proces selekcije novog primarnog čvora kada originalni primarni čvor prestane sa radom. Događaji koji mogu okinuti ovaj proces su:

- Dodavanje novog čvora u Replica Set
- Pokretanje Replica Set-a
- Pokretanje **Maintenance** procesa (održavanja) Replica Set-a
- [19] **Network Partition** (**Particionisanje mreže**), tip kvara koji deli distribuirani sistem na particije tako da čvorovi u jednoj particiji ne mogu komunicirati sa čvorovima iz druge. Kao posledica, moguće je da se primarni čvor nađe u delu mreže koji sadrži manjinu čvorova iz Replica Set-a. Kada primarni čvor otkrije da može da komunicira samo sa manjinom članova koji imaju pravo glasa, on automatski odstupa sa svoje uloge i prelazi u stanje sekundarnog čvora.

U međuvremenu, čvorovi koji se nalaze u drugom delu mreže i mogu da komuniciraju sa većinom čvorova koji imaju pravo glasa u procesu selekcije pokreću ovaj proces kako bi izabrali novi primarni čvor

- Kada sekundarni čvorovi ne dobiju **Heartbeat** signal od primarnog, koji se okida na svake dve sekunde i govori da je čvor funkcionalan. Podrazumevano vreme čekanja (**Configured Timeout**) je 10 sekundi, nakon čega se primarni čvor proglašava nedostupnim, a zatim pokreće **Automatic Failover** proces gde jedan od sekundarnih čvorova inicira proces selekcije i predlaže sebe za novog primarnog čvora. Ovaj proces je vidljiv na slici[5] 5:



Slika 5 - Election proces u Replica Set-u

Podrazumevano vreme trajanja procesa selekcije je 12 sekundi, ali latencija može uticati na vreme izvršenja, te je moguće izmeniti ovu vrednost u konfiguraciji, konkretno promenom **settings.electionTimeoutMillis** polja. Dodatno, prilikom evaluacije sekundarnih čvorova, algoritam obično prioritizuje one čvorove sa većom vrednošću **Priority** polja. Ukoliko ovakav čvor nije odabran, ostali čvorovi iniciraju proces selekcije kako bi se odabrao čvor sa najvećim prioritetom.
Pojam koji se koristi u konktestu procesa selekcije novog primarnog čvora je **term**, monotono rastuća vrednost koja označava broj pokušaja selekcije.

[20] Važno je napomenuti da je jedino moguće odabrati novi primarni čvor pod uslovom da je većina čvorova iz Replica Set-a koji mogu glasati dostupna. Ovo pravilo postoji kako bi se izbegla mogućnost postojanja dva primarna čvora unutar particionisanog Replica Set-a, što bi dovelo do Split-Brain problema prethodno opisanog u okviru poglavlja 3.6 Izazovi.
[21] Da bi se izdvojili tipovi čvorova koji mogu učestvovati u procesu selekcije, najpre je potrebno razmotriti moguća **stanja čvorova** u Replica Set-u:

[5] MongoDB - Replica Set Elections: https://www.mongodb.com/docs/manual/core/replica-set-elections

- **STARTUP** - Početno stanje svakog čvora koje označava da čvor još uvek nije aktivan član Replica Set-a
- **PRIMARY** - Primarni čvor. Može prihvatati operacije upisa. **Može glasati u toku procesa** selekcije
- **SECONDARY** - Sekundarni čvor. čija je uloga replikacija podataka. **Može glasati u toku procesa selekcije**
- **RECOVERING** - Čvor se nalazi u stanju sinhronizacije ili proverava podatke i ne može izvršavati operacije čitanja. **Može glasati u toku procesa selekcije**
- **STARTUP2** - Čvor izvršava inicijalnu sinhronizaciju. **Može glasati u toku procesa selekcije, osim kada je tek dodat u Replica Set**
- **UNKNOWN** - Ostali čvorovi ne mogu utvrditi stanje čvora
- **ARBITER** - Čvor arbitar. Ne sadrži replike. **Može glasati u toku procesa selekcije**
- **DOWN** - Nedostupan iz perspektive ostalih čvorova
- **ROLLBACK** - U **Rollback** procesu, tj. poništava određene operacije upisa. Podaci se ne mogu čitati iz čvora. **Može glasati u toku procesa selekcije**
- **REMOVED** - Čvor je uklonjen iz Replica Set-a

Dakle, od spomenutih stanja, čvorovi koji su u stanju **PRIMARY**, **SECONDARY**, **RECOVERING**, **STARTUP2**, **ARBITER** i **ROLLBACK** mogu glasati u toku procesa selekcije novog primarnog čvora. Od navedenih, **STARTUP2** ne može glasati ukoliko je tek dodat u Replica Set.

Osim stanja, dodatno svojstvo koje obezbeđuje pravo glasanja čvoru je **members[n].votes**, koje mora imati vrednost **1** ukoliko je čvoru dozvoljeno glasanje, a vrednost **0** ukoliko nije. Ukoliko čvor nema pravo glasanja (**members[n].votes = 0)**, njegov prioritet (**Priority**) mora imati vrednost **0** i obrnuto: čvorovi koji imaju prioritet veći od vrednosti **0** ne smeju imati vrednost **0** u polju **members[n].votes**. Dodatno, svaki čvor koji ima pravo glasa može glasati za maksimalno jedan čvor-kandidat u jednom **term**-u, pod uslovom da kandidat ima podjednako ažuriran ili noviji log u poređenju sa glasačem.

Kandidat pobeđuje i postaje novi primarni čvor ako prikupi broj glasova većine čvorova, računajući sebe, a zatim dobija jedinstvenu **term** vrednost. Kada vrši

U toku procesa selekcije nije moguće izvršavati operacije upisa, s obzirom da ih samo primarni čvor može izvršavati u MongoDB-u. Moguće je izvršavati operacije čitanja ukoliko su sekundarni čvorovi tako podešeni, detaljnije opisano u poglavlju 4.1.4 Operacije upisa i čitanja.

[22] Dodatno, moguće je da je originalni primarni čvor izvršio određene operacije upisa koje sekundarni čvorovi Replica Set-a nisu uspeli da repliciraju pre nego što je došlo do izbora novog primarnog čvora (**Automatic Failover** procesa). U tom slučaju, nakon što se prethodno primarni čvor ponovo priključi Replica Set-u, pokreće se **Rollback** proces, koji podrazumeva poništavanje operacija upisa koje je izvršio bivši primarni čvor. Na ovaj način se vrši očuvanje konzistencije podataka između čvorova.

Opisani Rollback procesi su retki jer ih MongoDB izbegava, ali se mogu desiti kao posledica mrežnog particionisanja Replica Set-a. Rollback se ne pokreće ukoliko postoji čvor na kome su replikovane operacije upisa pre Automatic Failover procesa, pod uslovom da je taj čvor dostupan većini čvorova Replica Set-a.

[23] Tokom regularnog funkcionisanja Replica Set-a, primarni čvor održava keš sa često korišćenim podacima u RAM memoriji, što omogućuje brže operacije čitanja i koristi se termin „**topao keš**" (eng. **Warm Cache**). Kao rezultat, javlja se pojava „**hladnog keša**" (eng. **Cold Cache**) kod sekundarnih čvorova ukoliko nisu konfigurisani za izvršavanje operacija čitanja.
Posledica ovih svojstava je vidljiva nakon procesa selekcije novog primarnog čvora, jer je potrebno vreme za „zagrevanje" keš memorije novog primarnog čvora, što može privremeno uticati na performanse.
Rešenje za ovaj problem jesu **Mirrored Reads**, gde primarni čvor pri regularnom radu šalje kopije određenih operacija ka sekundarnim čvorovima koji su potencijalni kandidati za primarni čvor. Zatim, sekundarni čvorovi izvršavaju dobijene upite i održavaju „topao keš", ali ne vraćaju rezultate klijentu.

Nakon odabira primarnog čvora, replikacija podataka između čvorova se zasniva na **oplog** strukturi, koji je tema sledećeg poglavlja.

4.1.3 Oplog i sinhronizacija

[24, 25] **Oplog** (**Operations Log**) je struktura u kojoj se čuva hronološki zapis svih operacija koje menjaju podatke u bazi podataka. Primarni čvor izvršava operacije i beleži ih u svoj **oplog**, zajedno sa jedinstvenom **term** vrednošću dobijenom nakon Election procesa. Sekundarni čvorovi zatim asinhrono preuzimaju i primenjuju te zapise kako bi se postigla konzistentnost podataka u okviru Replica Set-a.

Svaki zapis u **oplog**-u sadrži par vrednosti: term i **vremensku oznaku** (**timestamp**), koji zajedno formiraju OpTime. OpTime identifikuje operaciju u okviru Replica Set-a i omogućuje poređenje zapisa između čvorova. Ukoliko dva čvora istovremeno veruju da su primarni, prednost se daje zapisima sa većom **term** vrednošću.

Svaki čvor Replica Set-a održava sopstvenu kopiju **oplog**-a i periodično razmenjuje **Heartbeat** signale sa ostalim čvorovima radi detekcije stanja i sinhronizacije. Sekundarni čvorovi mogu preuzimati zapise iz **oplog**-a bilo kog drugog člana Replica Set-a.

S obzirom da zapisi poseduju vremenske oznake, relevantan pojam je i **Oplog Window**, koji predstavlja vremenski raspon između najnovijeg i najstarijeg zapisa u **oplog**-u. Ako sekundarni

čvor izgubi vezu sa primarnim, može se ponovo sinhronizovati putem **replikacije** samo ako se veza obnovi unutar tog vremenskog raspona.

[26] MongoDB koristi dva tipa **sinhronizacije podataka**:

1. **Initial Sync** (**Inicijalna sinhronizacija**) - Kopira sve podatke sa izabranog čvora Replica Set-a na novi čvor. Izvorni čvor se bira na osnovu vrednosti **initialSyncSourceReadPreference** parametra:
    - **Primary** - Striktno koristi primarni čvor. Ukoliko je on nedostupan, beleži se greška i periodično proverava dostupnost primarnog čvora
    - **PrimaryPreferred** - Pokušava da koristi primarni čvor. Ukoliko je on nedostupan, bira se neki od preostalih čvorova iz Replica Set-a
    - **Ostale opcije** - Izvor se bira među preostalim čvorovima Replica Set-a na osnovu dostupnosti i statusa čvora

    Proces izbora izvora vrši dva prolaza kroz listu svih članova Replica Set-a kako bi se pronašao čvor koji ispunjava kriterijume (da je ili primaran ili sekundaran, dostupan, ažuran itd). Ako se odgovarajući čvor ne pronađe nakon dva prolaza, beleži se greška i proces selekcije se ponovo pokreće.
2. **Replication** (**Replikacija**) - Nakon inicijalne sinhronizacije, čvorovi vrše replikaciju podataka sa izabranog čvora korišćenjem **oplog**-a. Izbor izvornog čvora zavisi od podešavanja **Chaining**, tj. svojstva **Ulančavanja**:
    - **Chaining uključen** - Odabir se vrši između čvorova
    - **Chaining isključen** - Primarni čvor je glavni izvor. Ukoliko je nedostupan, beleži se greška i proverava njegova dostupnost

    Kao i kod inicijalne sinhronizacije, proces izbora izvornog čvora obuhvata dva prolaza kroz listu članova Replica Set-a. Ukoliko ne uspe, proces se ponovo pokreće.

[25] Održavanje konzistentnosti podataka u okviru Replica Set-a zasniva se na distribuiranom konsenzus protokolu koji je inspirisan **Raft** algoritmom. Klasičan Raft algoritam koristi **push-based** model, u kojem primarni čvor aktivno prosleđuje nove zapise replikama putem RPC zahteva.
Za razliku od ovog pristupa, MongoDB koristi **pull-based** model replikacije, gde sekundarni čvorovi preuzimaju nove zapise od drugih replika, a ne samo od primarnog čvora. Nakon preuzimanja novih zapisa, replike izveštavaju primarni čvor o svom statusu, tj. o najnovijem obrađenom zapisu u **oplog**-u. Na osnovu tih izveštaja, primarni čvor utvrđuje koji zapisi su potvrđeni od strane većine čvorova, nakon čega oni postaju **commit**-ovani, čime se garantuje konzistentnost sistema.

4.1.4 Operacije upisa i čitanja

Na osnovu CAP teoreme, MongoDB se može svrstati unutar CP kategorije[6], tj. sistema koji prave kompromis oko dostupnosti. Podrazumevano ponašanje u MongoDB Replica Set-u je da sve operacije izvršava primarni čvor, ali se ponašanje u kontekstu operacija čitanja može promeniti tako da ih izvršavaju sekundarni čvorovi. Ovakva promena dovodi do **Eventual Consistency** modela, gde se promene vremenom repliciraju na sve kopije podataka u sistemu.

U okviru MongoDB Replica Set-a, mehanizmi **Write Concern**, **Read Concern** i **Read Preference** predstavljaju osnovne parametre koji utiču na ponašanje sistema pri operacijama upisa i čitanja, što utiče na balans između konzistentnosti i dostupnosti sistema. Ovi mehanizmi će biti predstavljeni u nastavku.

Write Concern

[27] Pojam **Write Concern** definiše nivo potvrde koji MongoDB zahteva od Replica Set-a pre nego što operaciju upisa smatra uspešnom. Ovo podešavanje direktno utiče na balans između pouzdanosti podataka i performansi sistema: veći nivo Write Concern-a pruža veću sigurnost da neće doći do gubitka podataka u slučaju prestanka rada nekog čvora, ali uvodi dodatnu latenciju jer zahteva potvrdu od više čvorova.

Podešavanje Write Concern-a se svodi na sledeću strukturu:
`{ w: <value>, j: <boolean>, wtimeout: <number> }`
gde su polja:
-**w**: nivo propagacije operacije upisa i može imati sledeće vrednosti:

- **majority** - Zahteva potvrdu od većine čvorova koji mogu glasati u toku Automatic Failover procesa i koji skladište podatke. MongoDB izračunava većinu kao manji broj između:
  - Većine svih čvorova koji mogu glasati, računajući i čvorove arbitre
  - Broja svih čvorova koji skladište podatke (primarni čvor i sekundarni čvorovi čiji je prioritet veći od nule) i mogu glasati

  gde se potvrda od čvora dobija ukoliko je operacija upisa zabeležena u njegovom lokalnom **oplog**-u
- **<broj>** - nalaže da je potrebno da definisan broj **mongod** instanci potvrdi uspešnu operaciju upisa. Na primer:
  - vrednost **w: 0** - Ne zahteva se potvrda. Ova opcija je rizična i koristi se samo u specifičnim slučajevima, npr. kada je potrebno samo vratiti poruke o greškama
  - vrednost **w: 1** - Potvrda je potrebna samo od strane primarnog čvora
  - vrednost **w > 1** - Potrebna potvrda od primarnog čvora i (**w-1**) sekundarnih čvorova koji sadrže replike, za koje važi da ne moraju imati mogućnost glasanja.

[6] MongoDB Community Forums - Q. About MongoDB's CAP: https://www.mongodb.com/community/forums/t/q-about-mongodbs-cap/150499/2

Na primer, za slučaj **w: 2** u Replica Set-u koji sadrži primarni i dva sekundarna čvora je potrebno dobiti potvrdu od primarnog i jednog sekundarnog čvora

- **<proizvoljan naziv>** - koristi se u situacijama kada postoji veći broj Data centara i gde se zahteva potvrda od specifičnih geografskih lokacija

-j: potvrda da je operacija upisa zabeležena u žurnalu (eng. **Journal**) na disku, gde ponašanje zavisi od vrednost **w**

-wtimeout: definiše vremensko ograničenje u milisekundama u kojem se operacija upisa mora propagirati na dovoljan broj čvorova Replica Set-a nakon uspešnog izvršenja na primarnom čvoru kako bi se postigao zadati Write Concern. Ukoliko je vrednost **w <= 1**, **wtimeout** se ne primenjuje. U ostalim slučajevima, ako se premaši definisano **wtimeout**, MongoDB vraća Write Concern grešku.

[28] Kada se određuje podrazumevana opcija za Write Concern se koristi sledeća formula:

```
if [ (#arbiters > 0) AND (#non-arbiters <= majority(#voting-nodes)) ]
   defaultWriteConcern = { w: 1 }
else
   defaultWriteConcern = { w: "majority" }
```

Dakle, ukoliko se ispune sledeći uslovi:

- postoji jedan ili više čvorova sa ulogom arbitra
- broj čvorova koji nisu arbitri je manji ili jednak broju većine čvorova koji imaju pravo glasanja

onda važi da je podrazumevana vrednost Write Concern mehanizma **w: 1**. Ukoliko neki od uslova nije ispunjen, podrazumevana vrednost je **majority**.

Read Concern

[29] **Read Concern** definiše kada je moguće čitati podatke. Konkretno, moguće je podesiti tako da se operacije čitanja izvršavaju čak i ako podaci nisu sinhronizovani na svim replikama, ili tek nakon što su svi članovi Replica Set-a dobili sve promene. Na ovaj način je moguće odabrati balans između dostupnosti i konzistentnosti, jer čitanje bez sinhronizacije omogućuje brže odgovore, ali smanjuje garanciju konzistentnosti vraćenih podataka. Ključne Read Concern vrednosti su sledeće:

- **local** - Upit vraća podatke bez garancije da su podaci upisani u većinu čvorova Replica Set-a. Kao posledica, moguće je poništavanje promena podataka ukoliko se desi Rollback proces
- **majority** - Upit vraća podatke koji su potvrđeni od strane većine čvorova Replica Set-a, tj. replicirani na većini. Čak i u slučaju kvara, podaci ostaju trajni, čime dolazi do većeg nivoa konzistentnosti
- **linearizable** - Upit vraća podatke koji su posledica svih operacija upisa potvrđenih od većine koje su završene pre operacije čitanja. Kao posledica je moguće da će upit

sačekati da se efekti operacija upisa (koje se dešavaju konkurentno) propagiraju do većine čvorova u Replica Set-u pre nego što vrati rezultate

Read Preference

[30] Svi čvorovi mogu izvršavati operacije čitanja, ali je u MongoDB-u podrazumevano ponašanje da će se zahtev za čitanjem proslediti primarnom čvoru. Ovo svojstvo se naziva **Read Preference** i moguće je odabrati sledeće opcije:

- [31] **primary** - podrazumevano ponašanje, operacije čitanja izvršava primarni čvor. Takođe, ova opcija je obavezna kod **distribuiranih transakcija**, s obzirom je potrebno da sve operacije unutar jedne transakcije koriste identičan čvor kao izvor podataka u cilju ostvarivanja atomičnosti pri čitanju različitih dokumenata i kolekcija
- **primaryPreferred** - u većini slučajeva operacije čitanja izvršava primarni čvor. Ukoliko on nije dostupan, prosleđuje se sekundarnom čvoru
- **secondary** - sve operacije čitanja izvršavaju sekundarni čvorovi
- **secondaryPreferred** - operacije čitanja izvršavaju sekundarni čvorovi, ali ukoliko Replica Set poseduje samo primarni čvor, operacije čitanja izvršava on
- **nearest** - operacije čitanja se prosleđuju čvoru koji ima prihvatljivu latenciju, bez obzira da li je primarni ili sekundarni

Osim prve opcije, Primary, preostale opcije mogu vratiti zastarele podatke prilikom čitanja s obzirom da se replikacija izvršava asinhrono, te je moguće da u nekom trenutku samo primarni čvor sadrži ažurirane podatke.

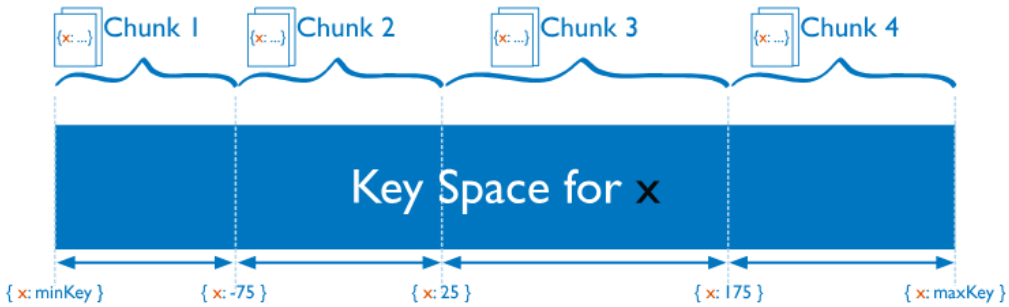
4.2 Sharding

[32] Pored Replikacije, MongoDB **Sharding** je dodatan mehanizam kojim se postiže HA. Sharding omogućuje horizontalno particionisanje podataka, gde se podaci raspodeljuju na više servera i cilj je smanjivanje opterećenja ukoliko se baza podataka oslanja na jedan server. U ovom kontekstu se svaki server naziva **Shard**, a ceo sistem **Sharded klaster** (eng. **Sharded Cluster**).

Particionisanje se vrši na osnovu vrednosti ključa, koji se u MongoDB-u naziva **Shard ključ** (eng. **Shard Key**), koji predstavlja polje (atribut) ili skup polja koji identifikuju svaki dokument. Svaka particija (Shard) unutar klastera sadrži određeni opseg ključeva. Od velike važnosti je pravilan odabir ključa kako bi podaci bili ravnomerno raspoređeni po Shard-ovima i omogućili se efikasni upiti. U suprotnom, može doći do preopterećenja nekog Shard-a jer će većina podataka biti raspoređena na njemu, što vremenom dovodi do nepoželjnih performansi.

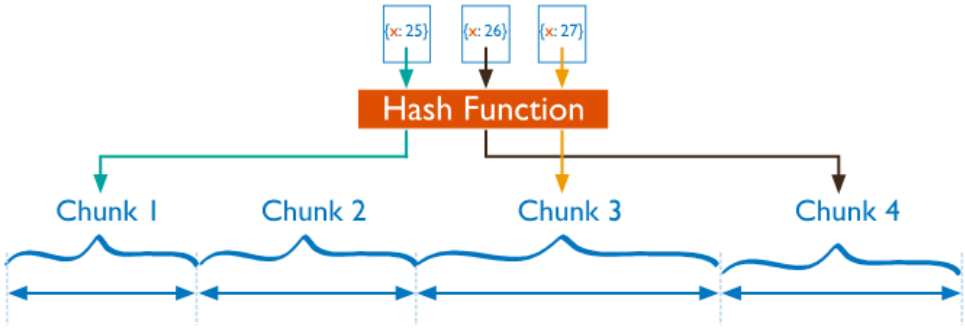
MongoDB vrši podelu svih postojećih vrednosti Shard ključeva u nepreklapajuće opsege ili heširane vrednosti Shard ključeva. Svaki opseg se dodeljuje određenom segmentu koji se naziva

Chunk, gde je cilj MongoDB-a izvršavanje ravnomerne distribucije segmenata u okviru Sharded klastera. Pojednostavljen princip je vidljiv na slici[7] 6:



Slika 6 - Distribucija vrednosti Shard ključeva u MongoDB-u

Ukoliko se koriste heširane vrednosti Shard ključeva, može se postići bolja distribucija vrednosti, ali je prilikom pretrage opsega vrednosti ključeva manja šansa da se željeni podaci nalaze unutar istog Shard-a. Pojednostavljen princip je vidljiv na slici[6] 7:



Slika 7 - Distribucija heširanih vrednosti Shard ključeva u MongoDB-u

[33] Dodatno, u okviru Sharded klastera postoji pozadinski proces koji se naziva **Sharded Cluster Balancer**, čija je uloga praćenje količine podataka na svakom Shard-u, ali i automatsko rebalansiranje pokretanjem migracije podataka između Shard-ova ukoliko se pojavi preopterećenje pojedinih Shard-ova.

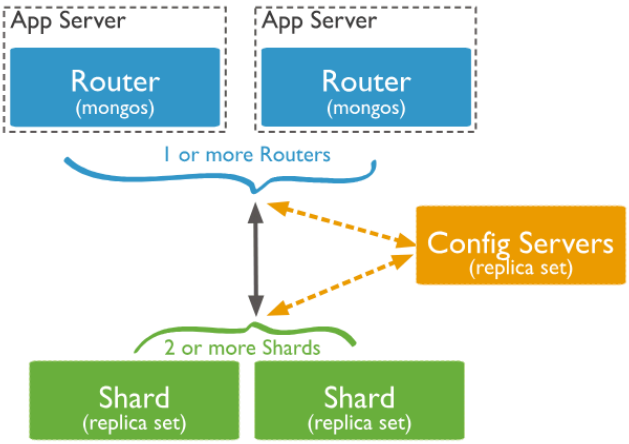
Sharded klaster se sastoji od sledećih komponenti:

1. **Shard-ovi** - serveri od kojih svaki sadrži podskup podataka. Svaki Shard predstavlja Replica Set jer sadrži i replike u pozadini, što znači da pruža redundansu i visoku dostupnost
2. **Config serveri** - čuvaju metapodatke, tj. koji opseg ključeva ili heš ide u koji Shard, kao i konfiguraciju klastera. Kao i sami Shard-ovi, predstavljaju Replica Set

[7] MongoDB - Sharding: https://www.mongodb.com/docs/manual/sharding

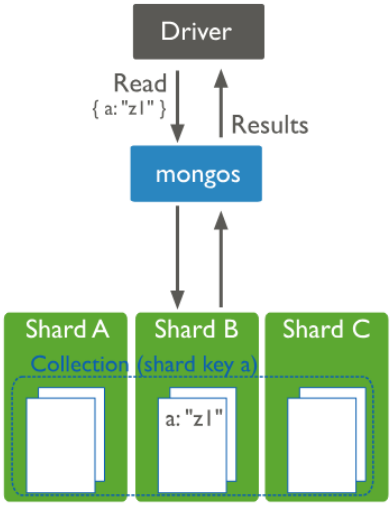
3. **Mongos instance** - komponente koje imaju ulogu rutiranja korišćenjem metapodatka sa Config servera uz pomoć kojih se vrši prosleđivanje klijentskih upita na odgovarajuće Shard-ove

gde se spomenuta arhitektura može videti na slici[6] 8:



Slika 8 - Komponente Sharded klastera

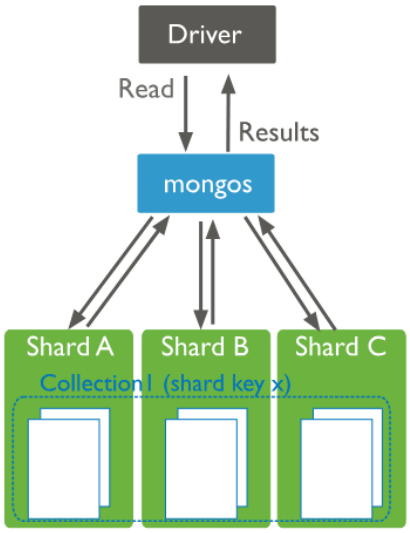
[34] Prednosti korišćenja Sharding-a je efikasnija distribucija operacija čitanja i upisa, s obzirom da svaki Shard sadrži određen podskup podataka sistema. Ukoliko se prilikom operacija koristi Shard Key, **mongos** instance mogu proslediti operacije na adekvatan Shard, vidljivo na slici[8] 9:



Slika 9 - Prosleđivanje operacije na konkretan Shard

Alternativa bi podrazumevala korišćenje **broadcast** komunikacije kojom se upit prosleđuje svim Shard-ovima, čeka njihov odgovor, a zatim se odgovori prosleđuju korisniku. Drugi naziv za ovaj pristup je **scatter/gather**, vidljivo na slici[7] 10:

[8] MongoDB - Routing with mongos: https://www.mongodb.com/docs/manual/core/sharded-cluster-query-router



Slika 10 - Broadcast komunikacija

Osim distribucije opterećenja, velika prednost je povećanje kapaciteta celokupnog sistema, kao i HA mehanizam jer se u ovakvim sistemima mogu izvršavati i **parcijalni upiti**. Konkretno, ukoliko jedan Shard (zajedno sa njegovim replikama) nije dostupan unutar klastera, operacije čitanja i upisa neće moći da pristupe tim podacima, ali će im biti omogućen pristup drugim podacima koji se nalaze na dostupnim Shard-ovima.

Raspodelom podataka na više servera sistem može efikasnije koristiti resurse i obrađivati veće količine podataka. Pravilno odabran Shard Key i balansirana distribucija podataka su ključni za optimalnu funkcionalnost Sharded klastera. Iako implementacija i održavanje ovakvog sistema uvode dodatnu složenost, Sharding u značajnoj meri doprinosi otpornosti sistema na otkazivanje pojedinačnih čvorova, kao i postizanje HA ciljeva.

5. PRAKTIČNI PRIMERI

U toku demonstracije će biti kreirano pet Docker instanci MongoDB baze, a zatim će biti manuelno kreiran Replica Set. Alternativno rešenje predstavlja korišćenje besplatne verzije **MongoDB Atlas**[9] servisa, fully managed cloud baze podataka koja automatski vrši zadatke kao što su:

- kreiranje rezervnih kopija korišćenjem Incremental Backup mehanizama i Point-in-Time oporavak, koji omogućuje povratak stanja baze podataka na određeno vreme u prošlosti
- skaliranje sistema prema nivoima opterećenja
- informacije o performansama upita, optimizaciju performansi i praćenje istih sa ugrađenim metrikama i alatima

Besplatna verzija MongoDB Atlas servisa automatski kreira Replica Set koji sadrži tri čvora, od kojih je jedan primarni i dva sekundarna. Ograničenja koja uvodi ovaj pristup jeste činjenica da se ne mogu izvršavati pojedine komande, konkretno one koje izazivaju automatski Failover, jer nije moguće ugasiti primarni čvor. Zbog veće fleksibilnosti pri demonstraciji će biti korišćen prvi pristup.

5.1 Kreiranje Replica Set-a

U okviru ovog poglavlja će biti prikazano kreiranje Replica Set-a od pet čvorova, jednog primarnog i četiri sekundarna, gde će za svaki čvor biti kreirana MongoDB instanca definisanjem odgovarajuće YAML konfiguracije i korišćenjem Docker servisa. Zatim će jednoj instanci, čvoru **mongo1**, biti prosleđena konfiguracija kojom se definiše Replica Set.
Prioritet čvora **mongo1** će biti jednak vrednosti **2**, dok će ostalim čvorovima prioriteti biti jednaki vrednosti **1**. Kao rezultat će Primarni čvor biti **mongo1**, dok će čvorovi **mongo2** - **mongo5** biti sekundarni.

Nakon kreiranja **.yaml** fajla[10] sa željenom konfiguracijom servisa, Docker okruženje se pokreće komandom:

```
docker-compose up -d
```

Kao rezultat se pokreću sve definisane MongoDB instance (**mongo1** - **mongo5**) i kreiraju odgovarajući kontejneri. Po završetku inicijalizacije kontejnera je potrebno pristupiti jednoj instanci, u ovom slučaju **mongo1**, kako bi se definisala konfiguracija Replica Set-a:

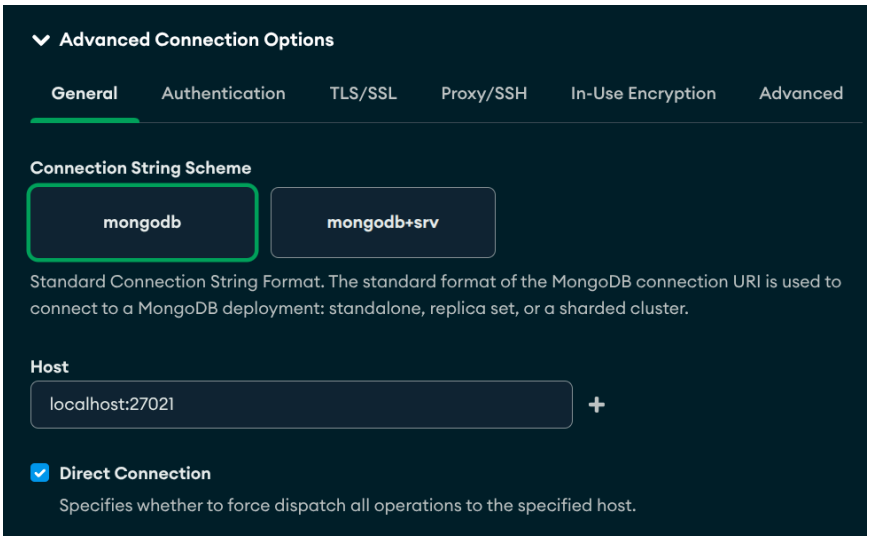
```
docker exec -it mongo1 mongosh
```

Ova komanda otvara interaktivnu **Mongo Shell** sesiju unutar **mongo1** kontejnera. Zatim se pokreće inicijalizacija Replica Set-a sledećom komandom:

[9] MongoDB Atlas - https://www.mongodb.com/products/platform/atlas-database
[10] YAML konfiguracija:
https://github.com/teo-kal/DBMS-3-mongodb-high-availability/blob/main/project/docker-compose.yml

```
rs.initiate({
   _id: "rs0",
   members: [
      { _id: 1, host: "mongo1:27017", priority: 2 },
      { _id: 2, host: "mongo2:27017", priority: 1 },
      { _id: 3, host: "mongo3:27017", priority: 1 },
      { _id: 4, host: "mongo4:27017", priority: 1 },
      { _id: 5, host: "mongo5:27017", priority: 1 }
   ]
})
```

čime se inicijalizuje Replica Set čiji je identifikator **rs0** i forsira **mongo1** kao primarni čvor zbog najvećeg prioriteta.
Jedan od načina povezivanja je grafički korišćenjem MongoDB Compass[11] okruženja. Na slici 11 je vidljivo povezivanje na **mongo1** instancu:



Slika 11 - Povezivanje na čvor u okviru MongoDB Compass grafičkog okruženja

Na ovaj način je moguće povezati se i na druge čvorove iz Replica Set-a.

5.1.1 Dodavanje i brisanje čvorova

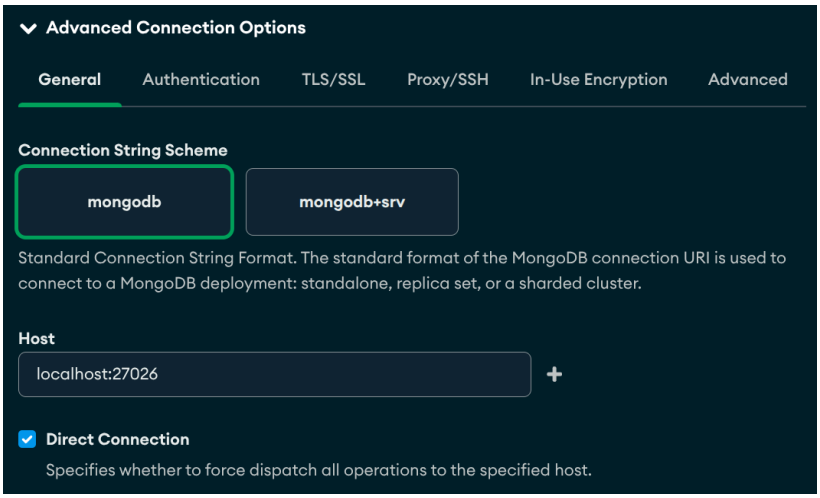
U ovom poglavlju će biti prikazano dodavanje nove **mongod** instance, **mongo6**, u postojeći Replica Set **rs0**.
Unutar YAML konfiguracije je potrebno dodati informacije o novoj instanci, a zatim izvršiti sledeću komandu:

```
docker-compose up -d mongo6
```

kojom se pokreće instanca **mongo6** bez uticaja na prethodno kreirane čvorove Replica Set-a.
Zatim je potrebno pristupiti čvoru **mongo1** sledećom komandom:

[11] MongoDB Compass: https://www.mongodb.com/products/tools/compass

```
docker exec -it mongo1 mongosh
```
i dodati **mongo6** instancu u Replica Set na sledeći način:
```
rs.add({ _id: 6, host: "mongo6:27017", priority: 0 })
```
Na ovaj način je u konfiguraciji Replica Set-a dodat čvor **mongo6** sa prioritetom **0**. U cilju jednostavnijeg testiranja je moguće dodati i konekciju u okviru MongoDB Compass okruženja, vidljivo na slici 12:



Slika 12 - Povezivanje na **mongo6** čvor

Ukoliko je potrebno ukloniti čvor **mongo6**, potrebno je izvršiti sledeću metodu nad primarnim čvorom:
```
rs.remove("<naziv_čvora>:27017")
```
Na ovaj se **mongo6** uklanja iz Replica Set-a.

5.2 Pregled informacija

[35, 36] Moguće iskoristiti **rs.conf()** metodu radi prikaza trenutne konfiguracije Replica Set-a. Osim osnovnih informacija kao što su naziv Replica Set-a (**_id**), **term** vrednosti, rezultat metode sadrži **members** niz u kome se mogu naći i informacije o članovima Replica Set-a, među kojima su:

- **_id** - identifikator čvora
- **host** - naziv čvora
- **arbiterOnly** - da li je čvor arbitar
- **hidden** - da li je čvor sakriven
- **priority** - prioritet čvora
- **secondaryDelaySecs** - ako je čvor Delayed Replica Set Member, ova opcija govori o broju sekundi kašnjenja čvora za primarnim
- **votes** - da li čvor glasa u procesu selekcije primarnog čvora ili ne

Na slici 13 se može videti primer ove strukture u kontekstu čvora **mongo1**:

```
members: [
  {
    _id: 1,
    host: 'mongo1:27017',
    arbiterOnly: false,
    buildIndexes: true,
    hidden: false,
    priority: 2,
    tags: {},
    secondaryDelaySecs: Long('0'),
    votes: 1
  },
```

Slika 13 - Element **members** niza

Osim navedenih vrednosti, vidljiva su i podešavanja koja se odnose na upisivanje u žurnal u kontekstu Write Concern mehanizma (**writeConcernMajorityJournalDefault**), da li je Chaining omogućen (**chainingAllowed**), metrike vezane za Heartbeat (**heartbeatIntervalMillis** i **heartbeatTimeoutSecs**) i proces selekcije (**electionTimeoutMillis**), vidljivo na slici 14:

```
protocolVersion: Long('1'),
writeConcernMajorityJournalDefault: true,
settings: {
  chainingAllowed: true,
  heartbeatIntervalMillis: 2000,
  heartbeatTimeoutSecs: 10,
  electionTimeoutMillis: 10000,
  catchUpTimeoutMillis: -1,
  catchUpTakeoverDelayMillis: 30000,
  getLastErrorModes: {},
  getLastErrorDefaults: { w: 1, wtimeout: 0 },
  replicaSetId: ObjectId('68f216c84c03e1ecdaa75e0c')
}
```

Slika 14 - Podešavanja Replica Set-a

[37, 38] Status Replica Set-a je moguće videti korišćenjem metode **rs.status()**. Neka od polja koje rezultujuća struktura sadrži su:

- **set** - naziv Replica Set-a
- **date** - datum
- **term -** vrednost koja identifikuje poslednji proces selekcije primarnog čvora
- **majorityVoteCount** - broj potrebnih čvorova za proces selekcije primarnog čvora
- **writeMajorityCount** - broj čvorova koji moraju potvrditi operaciju upisa kada je Write Concern vrednost definisana kao **majority**

- **writableVotingMembersCount** - broj čvorova koji sadrže podatke i imaju pravo glasanja, tj. definisanu vrednost **votes: 1**

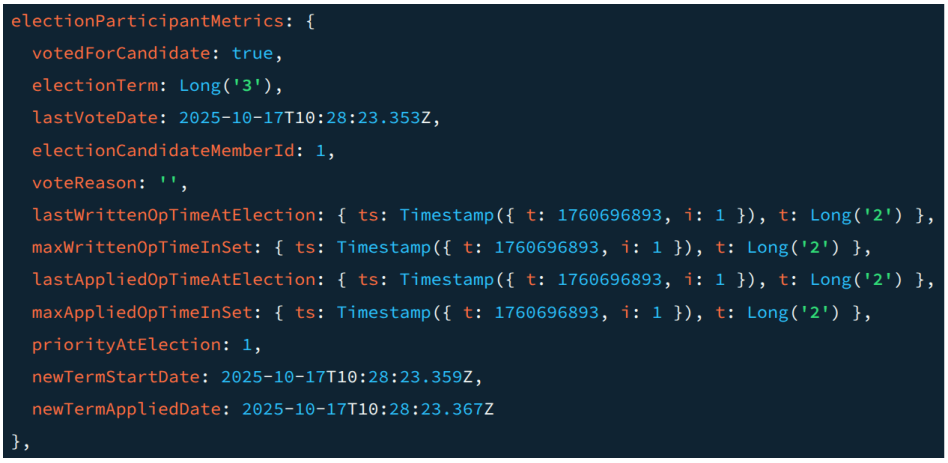
koja se mogu videti na slici 15:

Slika 15 - Status Replica Set-a

- **electionCandidateMetrics** - struktura koja sadrži informacije o procesu selekcije trenutnog primarnog čvora. Ovu strukturu samo sadrže primarni čvor i čvorovi kandidati, ali u slučaju kandidata metrike postaju nedostupne nakon procesa selekcije. Neke od informacija koje se mogu naći su razlog za pokretanjem procesa selekcije (**lastElectionReason**), vrednost prioriteta čvora u toku procesa selekcije (**priorityAtElection**) i vremenska oznaka početka novog **term**-a (**newTermStartDate**) vidljivo na slici 16:

Slika 16 - Informacije o procesu selekcije trenutnog primarnog čvora

- **electionParticipantMetrics** - slično prethodnoj strukturi, ali se pojavljuje kod čvorova koji su učestvovali u procesu selekcije novog primarnog čvora. Primer ovakve strukture je vidljiv na slici 17:



Slika 17 - Informacije o članu poslednjeg procesa selekcije

- **members** - lista čvorova sa njihovim informacijama:
  - **_id** - jedinstveni identifikator čvora u Replica Set-u
  - **name** - jedinstveni naziv čvora u Replica Set-u
  - **health** - vrednost **0** ukoliko čvor nije aktivan, **1** ukoliko jeste
  - **state** i **stateStr** - vrednost stanja čvora i odgovarajući naziv. Kod sekundarnih čvorova su ove vrednosti **2** i string **'SECONDARY'**, respektivno, dok su kod primarnog čvora vrednosti **1** i string **'PRIMARY'**
  - **syncSourceHost** i **syncSourceId** - naziv i identifikator čvora koji se koristi kao izvor pri sinhronizaciji. Kod primarnog čvora su ove vrednosti prazan string i vrednost **-1**, respektivno
  - **electionTime** i **electionDate** - vrednosti koje označavaju trenutak odabira, prisutne samo kod primarnog čvora

  gde se informacije o primarnom čvoru mogu videti na slici 18 a), dok se informacije jednog od sekundarnih čvorova mogu videti na slici 18 b):

a) b)

Slike 18 a) i b) - Elementi liste čvorova Replica Set-a

Dodatno, moguće je iskoristiti **filter()** funkciju nad **members** listom tako da se prikažu informacije samo o primarnom čvoru:

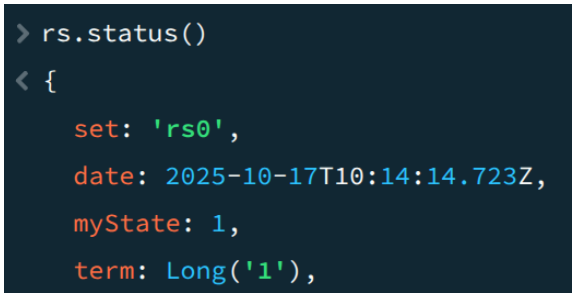
```
rs.status().members.filter(m => m.stateStr === "PRIMARY")
```

ili informacije o sekundarnim čvorovima:

```
rs.status().members.filter(m => m.stateStr === "SECONDARY")
```

5.3 Demonstracija Failover procesa

Najpre će biti prikazane pojedine informacije o Replica Set-u koje se odnose na proces selekcije primarnog čvora. Na slici 19 se može videti da je **term** vrednost **1**:



Slika 19 - Početna **term** vrednost

dok je na slici 20 vidljivo da je unutar **electionCandidateMetrics** strukture takođe vidljiva **term** vrednost (**electionTerm**), kao i vrednost prioriteta **mongo1** čvora u trenutku (**priorityAtElection**) primarnog čvora, u ovom slučaju **2**:

```
electionCandidateMetrics: {
  lastElectionReason: 'electionTimeout',
  lastElectionDate: 2025-10-17T10:13:38.701Z,
  electionTerm: Long('1'),
  lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  numVotesNeeded: 3,
  priorityAtElection: 2,
  electionTimeoutMillis: Long('10000'),
  numCatchUpOps: Long('0'),
  newTermStartDate: 2025-10-17T10:13:38.738Z,
  wMajorityWriteAvailabilityDate: 2025-10-17T10:13:39.227Z
},
```

Slika 20 - Pregled **term** informacija unutar čvora kandidata

Na kraju, u okviru slike 21 se mogu videti informacije o primarnom čvoru **mongo1**, konkretno stateStr: 'PRIMARY' vrednost:

```
{
  _id: 1,
  name: 'mongo1:27017',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 402,
  optime: [Object],
  optimeDate: 2025-10-17T10:14:08.000Z,
  optimeWritten: [Object],
  optimeWrittenDate: 2025-10-17T10:14:08.000Z,
  lastAppliedWallTime: 2025-10-17T10:14:08.749Z,
  lastDurableWallTime: 2025-10-17T10:14:08.749Z,
  lastWrittenWallTime: 2025-10-17T10:14:08.749Z,
  syncSourceHost: '',
  syncSourceId: -1,
  infoMessage: 'Could not find member to sync from',
  electionTime: Timestamp({ t: 1760696018, i: 1 }),
  electionDate: 2025-10-17T10:13:38.000Z,
  configVersion: 1,
  configTerm: 1,
  self: true,
  lastHeartbeatMessage: ''
},
```

Slika 21 - Informacije o primarnom čvoru

[39] Da bi se inicirao Failover proces, potrebno je pokrenuti metodu **rs.stepDown()** koja izaziva da primarni čvor odstupi sa svoje pozicije na definisano vreme, u ovom slučaju **60** sekundi. U toku definisanog vremenskog intervala čvor ne može biti kandidat za novog primarnog čvora i postaje sekundarni. Rezultati izvršenja su vidljivi na slici 22:

```
> rs.stepDown(60)
< {
    ok: 1,
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1760696828, i: 1 }),
      signature: {
        hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
        keyId: Long('0')
      }
    },
    operationTime: Timestamp({ t: 1760696828, i: 1 })
  }
```

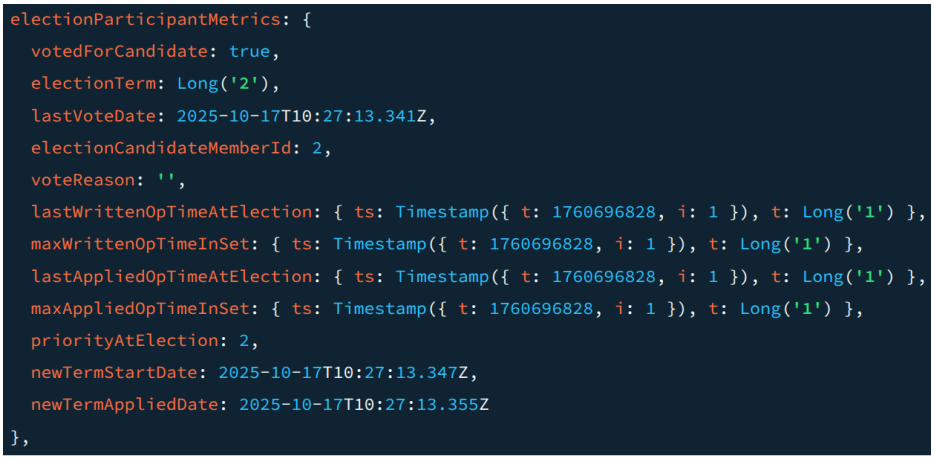
Slika 22 - Iniciranje Failover procesa

Neposredno nakon izvršavanja spomenute metode, ostali čvorovi iniciraju proces selekcije novog primarnog čvora, gde je deo rezultata metode **rs.status()** vidljiv na slici 23:

```
> rs.status()
< {
    set: 'rs0',
    date: 2025-10-17T10:27:14.321Z,
    myState: 2,
    term: Long('2'),
    syncSourceHost: 'mongo2:27017',
    syncSourceId: 2,
    heartbeatIntervalMillis: Long('2000'),
    majorityVoteCount: 3,
    writeMajorityCount: 3,
    votingMembersCount: 5,
    writableVotingMembersCount: 5,
```

Slika 23 - Iniciranje procesa selekcije novog primarnog čvora

Primećuje se da je **term** vrednost inkrementirana, što označava novi proces selekcije, dok je na slici 24 primetno da **mongo1** sada sadrži **electionParticipantMetrics** strukturu, s obzirom da zbog izvršene **stepDown** metode nije bio kandidat za novi primarni čvor iako mu je vrednost prioriteta bila najveća, vidljivo na osnovu vrednosti priorityAtElection: 2:



Slika 24 - Informacije o čvoru **mongo1**

Na slici 25 je vidljivo da je novi primarni čvor **mongo2** na osnovu vrednosti stateStr: 'PRIMARY':

Slika 25 - Informacije o čvoru **mongo2**

Nakon definisanog vremena od 60 sekundi od pokretanja **rs.stepDown()** metode, **mongo1** ponovo dobija pravo da bude kandidat u procesu selekcije. S obzirom da je njegov definisan prioritet prilikom konfiguracije veći od ostalih čvorova u Replica Set-u, inicira se novi proces

selekcije, iako je **mongo2** čvor izabran kao novi primarni, indikovano inkrementiranom **term** vrednošću na slici 26:

```
> rs.status()
< {
    set: 'rs0',
    date: 2025-10-17T10:28:30.338Z,
    myState: 1,
    term: Long('3'),
    syncSourceHost: '',
    syncSourceId: -1,
    heartbeatIntervalMillis: Long('2000'),
    majorityVoteCount: 3,
    writeMajorityCount: 3,
    votingMembersCount: 5,
    writableVotingMembersCount: 5,
```

Slika 26 - Novi proces selekcije

Dodatno, **electionCandidateMetrics** struktura sadrži **priorityTakeover** kao razlog iniciranja novog procesa selekcije unutar polja **lastElectionReason**, vidljivo na slici 27:

```
electionCandidateMetrics: {
  lastElectionReason: 'priorityTakeover',
  lastElectionDate: 2025-10-17T10:28:23.346Z,
  electionTerm: Long('3'),
  lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1760696893, i: 1 }), t: Long('2') },
  lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1760696893, i: 1 }), t: Long('2') },
  lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1760696893, i: 1 }), t: Long('2') },
  numVotesNeeded: 3,
  priorityAtElection: 2,
  electionTimeoutMillis: Long('10000'),
  priorPrimaryMemberId: 2,
  numCatchUpOps: Long('0'),
  newTermStartDate: 2025-10-17T10:28:23.359Z,
  wMajorityWriteAvailabilityDate: 2025-10-17T10:28:23.364Z
},
```

Slika 27 - Pregled novog procesa selekcije

gde je na slikama 28 a) i b) na osnovu **stateStr** vrednosti vidljivo da je **mongo1** ponovo primarni čvor, dok je **mongo2** ponovo sekundarni čvor, respektivno:

a) b)

Slike 28 a) i b) - Pregled čvorova **mongo1** i **mongo2**

5.4 Operacije čitanja i upisa

U nastavku će biti prikazan način na koji je moguće interagovati sa parametrima koji se tiču Write Concern, Read Concern i Read Preference mehanizama, najpre pregled vrednosti, a zatim i podešavanje istih, praćeno demonstracijom.

5.4.1 Pregled vrednosti

Pregled Read i Write Concern podešavanja se može obaviti korišćenjem sledeće komande:
`db.adminCommand({ getDefaultRWConcern: 1 })`
gde se primer rezultata može videti na slici 29:

Slika 29 - Pregled Read i Write Concern podešavanja

Konkretno, polja koja ukazuju na Read i Write Concern respektivno su **defaultReadConcern**, koji sadrži nivo **local** i **defaultWriteConcern** koja sadrži vrednosti `{ w: 'majority', wtimeout: 0 }`. Vrednost `w: 'majority'` označava da je potrebna potvrda većine čvorova da bi se operacija upisa smatrala uspešnom, dok se **wtimeout** ne primenjuje.

Ukoliko je potrebno proveriti podešavanja za Read Preference, može se iskoristiti sledeća komanda:
`db.getMongo().getReadPref()`
gde je primer rezultata dat na slici 30:

Slika 30 - Pregled Read Preference podešavanja

gde vrednost **mode** polja ukazuje na to da se operacije čitanja obavljaju preko primarnog čvora.

5.4.2 Izmena vrednosti

Osim pristupanja vrednostima, moguće je podesiti vrednosti za spomenute mehanizme na nivou jedne komande, ali i globalno.
Izmene na nivou jedne komande se mogu izvršiti na sledeći način:

- **Read Concern**:

Korišćenjem **readConcern()** metode

```
db.myCollection.find({ test: 1 }).readConcern("<vrednost>")
```

na primer, ukoliko je potrebno definisati da upit vrati podatke koji su potvrđeni od strane većine čvorova Replica Set-a (**majority**), komanda bi glasila:

```
db.myCollection.find({ test: 1 }).readConcern("majority")
```

- **Write Concern**:

Korišćenjem **writeConcern** polja

```
db.myCollection.insertOne(
   { test: 1 },
   { writeConcern: { w: <vrednost>, wtimeout: <vrednost> } }
)
```

na primer, definisanje Write Concern mehanizma koji zahteva da primarni i jedan sekundarni čvor potvrde upis (`w: 2`), kao i vreme 5000 milisekundi koje predstavlja maksimalno vreme propagacije operacije upisa nakon uspešnog izvršenja na primarnom čvoru:

```
db.myCollection.insertOne(
   { test: 1 },
   { writeConcern: { w: 2, wtimeout: 5000 } }
)
```

- **Read Preference**:

Korišćenjem **readPref()** metode

```
db.myCollection.find({ test: 1 }).readPref("<vrednost>")
```

na primer, promena da operaciju čitanja izvrši neki od sekundarnih čvorova umesto primarnog bi se postigla na sledeći način:

```
db.myCollection.find({ test: 1 }).readPref("secondary")
```

Moguće je globalno podesiti Read i Write Concern podešavanja na nivou klastera korišćenjem **db.adminCommand()** komande, kao i **defaultReadConcern** i **defaultWriteConcern** polja respektivno:

- **Read Concern**:

```
db.adminCommand({
   setDefaultRWConcern: 1,
   defaultReadConcern: { level: "<vrednost>" }
})
```

na primer, ukoliko je potrebno definisati da upit vrati podatke koji su potvrđeni od strane većine čvorova Replica Set-a (**majority**), komanda bi glasila:

```
db.adminCommand({
   setDefaultRWConcern: 1,
   defaultReadConcern: { level: "majority" }
})
```

- **Write Concern**:

```
db.adminCommand({
   setDefaultRWConcern: 1,
```

```
  defaultWriteConcern: { w: <vrednost>, wtimeout: <vrednost> }
})
```
na primer, definisanje Write Concern mehanizma koji zahteva da primarni i jedan sekundarni čvor potvrde upis (`w: 2`), kao i vreme 5000 milisekundi koje predstavlja maksimalno vreme propagacije operacije upisa nakon uspešnog izvršenja na primarnom čvoru:
```
db.adminCommand({
  setDefaultRWConcern: 1,
  defaultWriteConcern: { w: 2, wtimeout: 5000 }
})
```

Za razliku od prethodnih, globalno podešavanje Read Preference mehanizma se vrši na nivou konekcije korišćenjem **setReadPref()** metode na sledeći način:
```
db.getMongo().setReadPref("<vrednost>")
```
na primer, promena da operaciju čitanja izvrši neki od sekundarnih čvorova umesto primarnog bi se postigla na sledeći način:
```
db.getMongo().setReadPref("secondary")
```

5.4.3 Testiranje Write Concern i Read Preference svojstava

1. **Write Concern**

U okviru ovog segmenta će biti demonstrirano ponašanje kreiranog Replica Set-a koji sadrži pet čvorova: jedan primarni (**mongo1**, čiji je prioritet **2**) i četiri sekundarna (**mongo2-mongo5**, čiji su prioriteti **1**). U cilju preglednosti, iskorišćena je komanda
```
rs.status().members.map(m => [m.name, m.stateStr])
```
koja prikazuje nazive i stanja čvorova, vidljivo na slici 31:

```
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'PRIMARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', 'SECONDARY' ],
    [ 'mongo4:27017', 'SECONDARY' ],
    [ 'mongo5:27017', 'SECONDARY' ]
  ]
```

Slika 31 - Pregled statusa čvorova **mongo1-mongo5**

Podrazumevana vrednost za Write Concern u ovom Replica Set-u je `w: 'majority'`, vidljivo na slici 32 kao rezultat komande `db.adminCommand({ getDefaultRWConcern: 1 })`:

Slika 32 - Pregled Read Concern vrednosti

Zatim će Write Concern biti postavljen tako da zahteva potvrdu od tri čvora, tj. primarnog i dva sekundarna čvora:

```
db.adminCommand({
   setDefaultRWConcern: 1,
   defaultWriteConcern: { w: 3 }
})
```

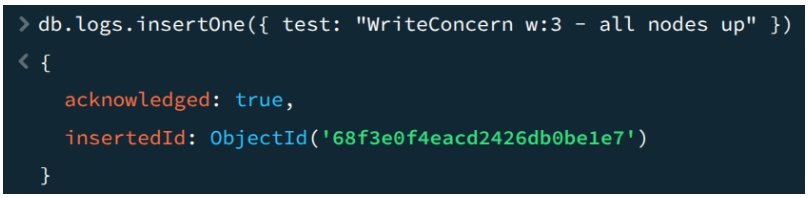
gde je isečak vidljiv na slici 33:

Slika 33 - Postavljanje nove Write Concern vrednosti

Kao provera, unosi se dokument:

```
db.logs.insertOne({ test: "WriteConcern w:3 - all nodes up" })
```

gde se uspešnost operacije vidi na slici 34:



Slika 34 - Unos vrednosti

Zatim se koristi komanda kojom se čvorovi **mongo3**, **mongo4** i **mongo5** zaustavljaju, tj. njihove Docker Container instance:

```
docker stop mongo3 mongo4 mongo5
```

gde se na slici 35 može videti status čvorova i da su samo čvorovi **mongo1** i **mongo2** aktivni:

```
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'PRIMARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', '(not reachable/healthy)' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

Slika 35 - Pregled statusa čvorova

ali, s obzirom da većina čvorova Replica Set-a nije dostupna, sledeća operacija upisa na čvoru **mongo1** nije moguća, vidljivo na slici 36:

```
> db.logs.insertOne({ test: "WriteConcern w:3 - mongo3,4,5 down" })
✖ ▸ MongoServerError[NotWritablePrimary]: not primary
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'SECONDARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', '(not reachable/healthy)' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

Slika 36 - Neuspešna operacija upisa

Kao posledica pada većine čvorova, **mongo1** ne može održati svoj status primarnog, te je ponovnim pregledom statusa Replica Set-a vidljivo da su čvorovi **mongo1** i **mongo2** sekundarni čvorovi. Trenutno ne postoji primarni čvor koji će preuzeti operacije upisa.

Zatim je pokrenuta komanda

```
docker start mongo3
```

kako bi **mongo3** ponovo postao aktivan čvor. Na slikama 37 a), b) i c) se može videti progresivna razlika izvršavanja **status()** metode:

```
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'SECONDARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', 'SECONDARY' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

a)

```
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'SECONDARY' ],
    [ 'mongo2:27017', 'PRIMARY' ],
    [ 'mongo3:27017', 'SECONDARY' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

b)

```
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'PRIMARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', 'SECONDARY' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

c)

Slike 37 a), b) i c) - Pregled statusa čvorova u toku novog procesa selekcije

Slika 37a) pokazuje inicijalno stanje nakon povratka čvora **mongo3**, gde su svi prisutni čvorovi sekundarni. Zatim se na slici 37b) može videti da je čvor **mongo2** postao primarni čvor. Nakon nekog vremena se proces selekcije ponovo inicira, vidljivo na slici 37c), s obzirom da čvor **mongo1** sadrži najveću vrednost prioriteta: **2**.

Na kraju je moguće izvršiti operaciju upisa jer su tri čvora prisutna u Replica Set-u, vidljivo na slici 38:

```
> db.logs.insertOne({ test: "WriteConcern w:3 - mongo4,5 down" })
< {
    acknowledged: true,
    insertedId: ObjectId('68f3e183eacd2426db0be1e9')
  }
```

Slika 38 - Uspešna operacija upisa

2. Read Preference

U okviru demonstracije Read Preference mehanizma će biti iskorišćena **mongosh** komanda u kombinaciji sa Read Preference parametrom pri specificiranju Connection String-a, s obzirom da se ovo svojstvo primenjuje nad individualnom konekcijom, za razliku od Read i Write Concern svojstava koji se podešavaju na nivou čitavog klastera.
Povezivanje na primarni čvor se izvršava sledećom komandom:

```
docker exec -it mongo1 mongosh
"mongodb://mongo1:27017/?replicaSet=rs0&readPreference=primary"
```

Gde opcija `readPreference=primary` označava da će primarni čvor izvršavati operacije čitanja.
Rezultat izvršenja ove komande je povezivanje na primarni čvor, konkretno **mongo1**, gde se status svakog čvora može videti na slici 39, kao i podešena Read Preference vrednost korišćenjem metode **getReadPref()**:

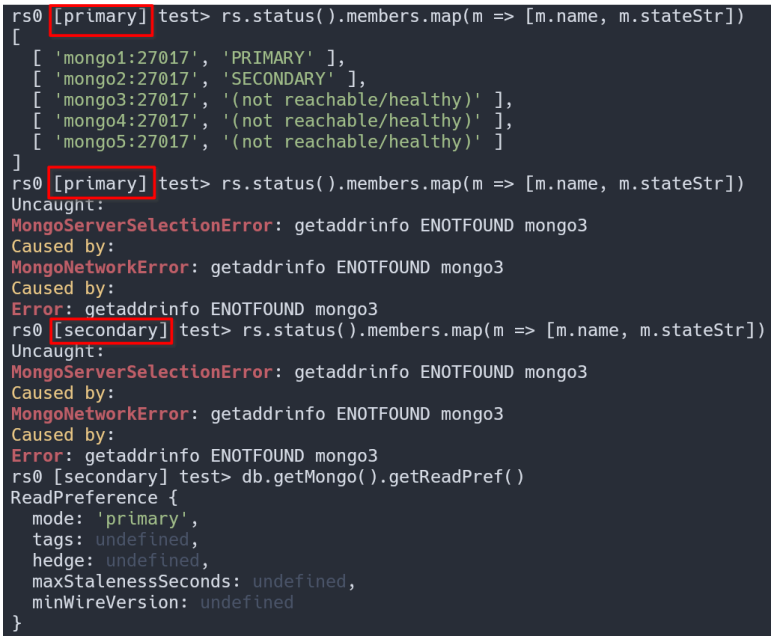
```
rs0 [primary] test> rs.status().members.map(m => [m.name, m.stateStr])
[
  [ 'mongo1:27017', 'PRIMARY' ],
  [ 'mongo2:27017', 'SECONDARY' ],
  [ 'mongo3:27017', 'SECONDARY' ],
  [ 'mongo4:27017', 'SECONDARY' ],
  [ 'mongo5:27017', 'SECONDARY' ]
]
rs0 [primary] test> db.getMongo().getReadPref()
...
ReadPreference {
  mode: 'primary',
  tags: undefined,
  hedge: undefined,
  maxStalenessSeconds: undefined,
  minWireVersion: undefined
}
```

Slika 39 - Pregled statusa čvorova i Read Preference vrednosti

Nakon toga se čvorovi **mongo3 - mongo5** zaustavljaju korišćenjem komande:

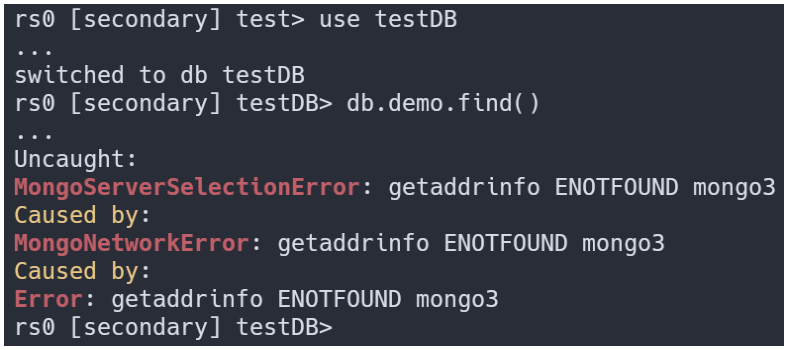
```
docker stop mongo3 mongo4 mongo5
```

gde se na slici 40 može videti proces povlačenja **mongo1** čvora sa mesta primarnog, naznačeno crvenim pravougaonicima:



Slika 40 - Proces povlačenja **mongo1** čvora sa uloge primarnog

Novi primarni čvor se ne može odabrati s obzirom da ne postoji većina koja bi učestvovala u procesu glasanja. Kao rezultat, čvorovi **mongo1** i **mongo2** postaju sekundarni čvorovi. Dodatno, kao posledica navođenja **primary** Read Preference svojstva, nije moguće izvršavati operacije čitanja nad preostalim čvorovima, vidljivo na slici 41:



Slika 41 - Neuspešna operacija čitanja

Ukoliko bi se povezivanje na neki od preostalih čvorova izvršilo korišćenjem konekcionog stringa u kome se navodi Read Preference svojstvo kao što je **secondary**, operacije čitanja bi bile uspešno izvršene, gde je primer komande u cilju povezivanja na čvor **mongo2** sledeća:

```
docker exec -it mongo2 mongosh
"mongodb://mongo2:27017/?replicaSet=rs0&readPreference=secondary"
```

Rezultat izvršenja operacije čitanja, kao i statusa čvorova je vidljiv na slici 42:

```
rs0 [secondary] TestDB> use testDB
switched to db testDB
rs0 [secondary] testDB> db.demo.find()
[ { _id: 1, v: 'new' } ]
rs0 [secondary] testDB> rs.status().members.map(m => [m.name, m.stateStr])
[
  [ 'mongo1:27017', 'SECONDARY' ],
  [ 'mongo2:27017', 'SECONDARY' ],
  [ 'mongo3:27017', '(not reachable/healthy)' ],
  [ 'mongo4:27017', '(not reachable/healthy)' ],
  [ 'mongo5:27017', '(not reachable/healthy)' ]
]
```

Slika 42 - Uspešna operacija čitanja i pregled statusa čvorova

6. ZAKLJUČAK

U drugom poglavlju ovog rada opisani su osnovni koncepti vezani za **High Availability** (**HA**), uključujući ključne metrike kao što su **Uptime** i **Downtime**, kao i pojmove **Single Point of Failure** (**SPOF**) i **Service Level Agreement** (**SLA**). Takođe je objašenjena povezanost HA pristupa sa **CAP teoremom**, koja opisuje kompromis između konzistentnosti, dostupnosti i tolerancije na particionisanje u distribuiranim sistemima.

Treće poglavlje je bilo posvećeno različitim tehnikama za postizanje visokog nivoa dostupnosti kao što su: **replikacija**, uvođenje **klastera**, **Failover mehanizmi**, **Load Balancing** i **Backup** i **Disaster Recovery strategije**. Posebno su istaknuti izazovi implementacije HA rešenja, poput **sinhronizacije podataka** i **Split-brain** problema.

U četvrtom poglavlju analiziran je način na koji se High Availability koncepti primenjuju u okviru MongoDB baze podataka. Detaljno su obrađeni pojmovi **Replica Set**-a, uloge **primarnih**, **sekundarnih** i **Arbiter** čvorova, proces izbora novog primarnog čvora (**Election proces**), kao i mehanizmi sinhronizacije korišćenjem **oplog**-a. Obrađene su i kontrole operacija čitanja i upisa kroz **Write Concern**, **Read Concern** i **Read Preference** podešavanja. Takođe, predstavljen je i koncept **Sharding**-a kao metoda za horizontalno particionisanje podataka i skaliranje sistema.

U petom poglavlju prikazani su praktični primeri konfiguracije i testiranja mehanizama MongoDB Replica Set-a, uključujući kreiranje i upravljanje čvorovima, pregleda njihovih vrednosti, simulacije **Failover procesa**, kao i demonstraciju ponašanja prilikom upisa i čitanja podataka.

MongoDB spada u kategoriju baza podataka koje podrazumevano poseduju visok nivo konzistentnosti, ali i otpornost na otkaze pri particionisanju. Međutim, kombinacijom **replikacije**, **Sharding**-a i ugrađenih **Failover mehanizama**, MongoDB omogućuje visok nivo dostupnosti i skalabilnosti. Ovakav sistem omogućuje podešavanje balansa između dostupnosti i konzistentnosti. Konkretno, **Write Concern** definiše nivo potvrde pri operacijama upisa, **Read Preference** određuje kojim čvorovima se prosleđuju zahtevi za čitanje, dok **Read Concern** reguliše trenutak pristupanja podacima.
Zaključuje se da postizanje željenih performansi i optimalne konfiguracije zahteva pažljivo planiranje topologije sistema, kao i razumevanje kompromisa između dostupnosti, konzistentnosti, troškova i nivoa redundanse.

7. LITERATURA

[1] Wikipedia - High availability: https://en.wikipedia.org/wiki/High_availability (Poslednji pristup: 31.10.2025. godine)
[2] Wikipedia - Two-phase commit protocol: https://en.wikipedia.org/wiki/Two-phase_commit_protocol (Poslednji pristup: 31.10.2025. godine)
[3] Amazon - What's the Difference Between an ACID and a BASE Database?: https://aws.amazon.com/compare/the-difference-between-acid-and-base-database (Poslednji pristup: 31.10.2025. godine)
[4] Chang Wan - High Performance Architecture: Read-Write Separation: https://www.cwblogs.com/posts/read-write-separation (Poslednji pristup: 31.10.2025. godine)
[5] NETDATA - What Is Database Clustering? Types & Benefits: https://www.netdata.cloud/academy/whatisdatabaseclusteringtypesbenefits (Poslednji pristup: 31.10.2025. godine)
[6] Cloudflare - Types of load balancing algorithms: https://www.cloudflare.com/learning/performance/types-of-load-balancing-algorithms (Poslednji pristup: 31.10.2025. godine)
[7] TraefikLabs - Everything You Need to Know About Round Robin Load Balancing: https://traefik.io/glossary/round-robin-load-balancing (Poslednji pristup: 31.10.2025. godine)
[8] N2W - Database Backup: Types, Methods, and Backup in Common DB Systems: https://n2ws.com/blog/database-backup (Poslednji pristup: 31.10.2025. godine)
[9] DZone - Split-Brain in Distributed Systems: https://dzone.com/articles/split-brain-in-distributed-systems (Poslednji pristup: 31.10.2025. godine)
[10] MongoDB - A Guide to Horizontal vs Vertical Scaling: https://www.mongodb.com/resources/basics/horizontal-vs-vertical-scaling (Poslednji pristup: 31.10.2025. godine)
[11] MongoDB - Replica Set Members: https://www.mongodb.com/docs/manual/core/replica-set-members (Poslednji pristup: 31.10.2025. godine)
[12] MongoDB - Replica Set Primary: https://www.mongodb.com/docs/manual/core/replica-set-primary (Poslednji pristup: 31.10.2025. godine)
[13] MongoDB - Replica Set Secondary Members: https://www.mongodb.com/docs/manual/core/replica-set-secondary (Poslednji pristup: 31.10.2025. godine)
[14] MongoDB - Priority 0 Replica Set Members: https://www.mongodb.com/docs/manual/core/replica-set-priority-0-member (Poslednji pristup: 31.10.2025. godine)
[15] MongoDB - Hidden Replica Set Members: https://www.mongodb.com/docs/manual/core/replica-set-hidden-member (Poslednji pristup: 31.10.2025. godine)
[16] MongoDB - Delayed Replica Set Members: https://www.mongodb.com/docs/manual/core/replica-set-delayed-member (Poslednji pristup: 31.10.2025. godine)

[17] MongoDB - Replica Set Arbiter: https://www.mongodb.com/docs/manual/core/replica-set-arbiter (Poslednji pristup: 31.10.2025. godine)
[18] MongoDB - Replica Set Elections: https://www.mongodb.com/docs/manual/core/replica-set-elections (Poslednji pristup: 31.10.2025. godine)
[19] MongoDB - Glossary: https://www.mongodb.com/docs/manual/reference/glossary (Poslednji pristup: 31.10.2025. godine)
[20] MongoDB Community Forums - Three-member replica set across Data center: https://www.mongodb.com/community/forums/t/three-member-replica-set-across-data-center/127932/3 (Poslednji pristup: 31.10.2025. godine)
[21] MongoDB - Replica Set Member States: https://www.mongodb.com/docs/manual/reference/replica-states/#replica-set-member-states (Poslednji pristup: 31.10.2025. godine)
[22] MongoDB - Rollbacks During Replica Set Failover: https://www.mongodb.com/docs/manual/core/replica-set-rollbacks/#std-label-replica-set-rollback (Poslednji pristup: 31.10.2025. godine)
[23] MongoDB - Replication - Mirrored Reads: https://www.mongodb.com/docs/manual/replication/#std-label-mirrored-reads (Poslednji pristup: 31.10.2025. godine)
[24] MongoDB - Replica Set Oplog: https://www.mongodb.com/docs/manual/core/replica-set-oplog (Poslednji pristup: 31.10.2025. godine)
[25] USENIX - Siyuan Zhou, MongoDB Inc.; Shuai Mu, Stony Brook University - Fault-Tolerant Replication with Pull-Based Consensus in MongoDB: https://www.usenix.org/system/files/nsdi21-zhou.pdf (Poslednji pristup: 31.10.2025. godine)
[26] MongoDB - Replica Set Data Synchronization: https://www.mongodb.com/docs/manual/core/replica-set-sync (Poslednji pristup: 31.10.2025. godine)
[27] MongoDB - Write Concern: https://www.mongodb.com/docs/manual/reference/write-concern (Poslednji pristup: 31.10.2025. godine)
[28] MongoDB - Default MongoDB Read Concerns/Write Concerns - Default Write Concern: https://www.mongodb.com/docs/manual/reference/mongodb-defaults/#default-write-concern (Poslednji pristup: 31.10.2025. godine)
[29] MongoDB - Read Concern: https://www.mongodb.com/docs/manual/reference/read-concern (Poslednji pristup: 31.10.2025. godine)
[30] MongoDB - Read Preference: https://www.mongodb.com/docs/manual/core/read-preference (Poslednji pristup: 31.10.2025. godine)
[31] MongoDB - Transactions: https://www.mongodb.com/docs/manual/core/transactions/#std-label-transactions (Poslednji pristup: 31.10.2025. godine)
[32] MongoDB - Sharding: https://www.mongodb.com/docs/manual/sharding (Poslednji pristup: 31.10.2025. godine)
[33] MongoDB - Sharded Cluster Balancer: https://www.mongodb.com/docs/manual/core/sharding-balancer-administration (Poslednji pristup: 31.10.2025. godine)

[34] MongoDB - Routing with mongos - Targeted Operations vs. Broadcast Operations: https://www.mongodb.com/docs/manual/core/sharded-cluster-query-router/#targeted-operations-vs.-broadcast-operations (Poslednji pristup: 31.10.2025. godine)
[35] MongoDB - rs.conf() (mongosh method): https://www.mongodb.com/docs/manual/reference/method/rs.conf (Poslednji pristup: 31.10.2025. godine)
[36] MongoDB - Self-Managed Replica Set Configuration: https://www.mongodb.com/docs/manual/reference/replica-configuration (Poslednji pristup: 31.10.2025. godine)
[37] MongoDB - replSetGetStatus (database command): https://www.mongodb.com/docs/manual/reference/command/replSetGetStatus (Poslednji pristup: 31.10.2025. godine)
[38] MongoDB - rs.status() (mongosh method): https://www.mongodb.com/docs/manual/reference/method/rs.status (Poslednji pristup: 31.10.2025. godine)
[39] MongoDB - rs.stepDown() (mongosh method): https://www.mongodb.com/docs/manual/reference/method/rs.stepDown (Poslednji pristup: 31.10.2025. godine)