**Replikacija podataka kod MongoDB baze podataka**

Kristina Antić, 934

Sadržaj

[1. uvod 3](#_Toc42618297)

[2. replikacija podataka kod mongodb baze podataka 5](#_Toc42618298)

[2.1. Članovi replica set-a 5](#_Toc42618299)

[2.2. Sinhronizacija podataka i semantika čitanja i upisa 7](#_Toc42618300)

[2.3. Visoka dostupnost replica set-a 11](#_Toc42618301)

[2.4. Kreiranje i podešavanje replica set-a 12](#_Toc42618302)

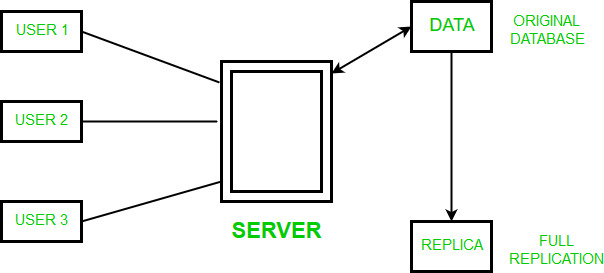
[3. zaključak 18](#_Toc42618303)

[literatura 18](#_Toc42618304)

# uvod

Replikacija podataka je proces čuvanja podataka na više lokacija, odnosno čvorova, zbog poboljšanja dostupnosti podataka. To je jednostavan proces kopiranja podataka iz baze sa jednog na drugi server tako da podaci budu konzistentni i dostupni bez obzira na to kom serveru se pristupa. Rezultat je distribuirana baza podataka.

Najekstremniji slučaj je ***replikacija cele baze podataka*** na svakom čvoru distribuiranog sistema, odnosno kreiranje potpuno replicirane distribuirane baze podataka. Ovo može značajno da poveća dostupnost jer sistem može da nastavi da radi dokle god je barem jedan čvor aktivan. Performanse globalnih upita se, takođe, poboljšavaju jer se rezultat upita može dobiti lokalno sa bilo kog čvora. Nedostatak potpune replikacije je drastično usporavanje operacija ažuriranja jer se bilo koje ažuriranje mora obaviti na svakoj kopiji baze podataka kako bi kopije ostale konzistentne. Kontrola konkurentnosti i tehnike oporavka su skuplje kod potpune replikacije. Šema potpune replikacije prikazana je na slici 1.1.



Slika 1.1 Šema potpune replikacije

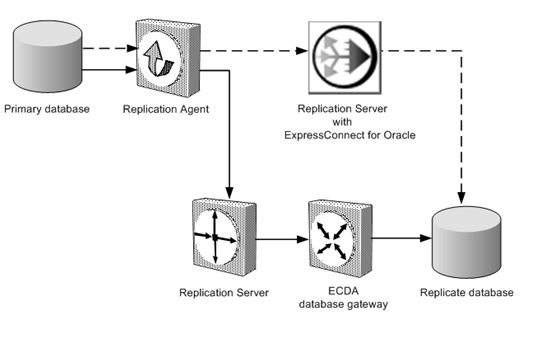
Drugi ekstremni slučaj je ***nepostojanje replikacija***. Zove se još i neredundantna alokacija. Ovo smanjuje dostupnost podataka i usporava izvršenje upita jer svi korisnici pristupaju istom serveru. Nasuprot tome, ukoliko nema replika baze podataka, oporavak je jednostavan i moguće je ostvariti konkurentnost.

Između prethodna dva ekstremna slučaja je ***parcijalna replikacija podataka***, odnosno repliciranje određenih delova baze podataka. Broj kopija delova baze može da varira od jedne, pa do broja čvorova u sistemu. Izbor čvorova i stepena replikacije zavisi od zahtevane dostupnosti i performansi, kao i od važnosti podataka koji se repliciraju.

U zavisnosti od načina ažuriranja distribuiranih podataka, postoji ***sinhrona*** i ***asinhrona*** replikacija. Ukoliko se radi o sinhronoj replikaciji, transakcije moraju biti atomične, odnosno sve kopije modifikovane relacije moraju biti ažurirane pre završetka transakcije koja vrši modifikaciju. Alternativni pristup, asnihrona replikacija, podrazumeva da se kopije modifikovane relacije ažuriraju periodično. Transakcije koje čitaju različite kopije iste relacije mogu videti različite vrednosti. Asinhrona replikacija kompromituje podatke, ali se može implementirati efikasnije od sinhrone replikacije.

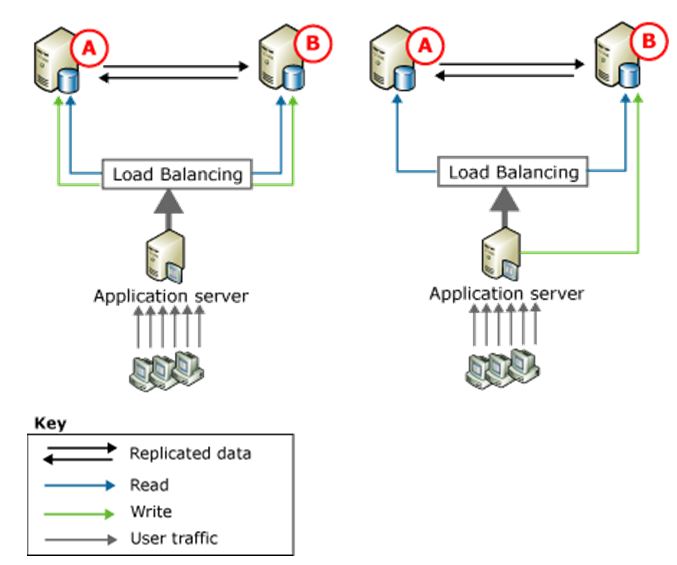
Razlikujemo više tipova replikacije:

1. ***Snapshot replikacija*** – podaci se šalju onako kako izgledaju u određenom trenutku, bez ikakvih promena. Koristi se kada nema čestih promena podataka. Snapshot replikacija je dobar način za inicijalnu sinhronizaciju između master i slave čvorova.
2. ***Transakciona replikacija*** – dobija se cela inicijalna kopija baze podataka, a potom se šalju samo promene koje se dešavaju u bazi. Ostali podaci ostaju nepromenjeni. Podatke šalje master čvor slave čvorovima. Ovaj način repliciranja je brži od snapshot repliciranja jer se sa jedne na drugu lokaciju šalju samo modifikovani podaci.
3. ***Merge replikacija*** – spajanje više baza u jednu, kombinovanu bazu. Ovo je najsloženiji način replikacije jer dozvoljava čvorovima da nezavisno menjaju bazu.
4. ***Heterogena replikacija*** – replikacija se dešava između heterogenih servera. Potrebno je da postoji određeni servis za povezivanje i adaptiranje različitih servera kako bi replikacija mogla da se obavi. Na slici 1.2 prikazana je šema heterogene replikacije.



Slika 1.2 Šema heterogene replikacije

1. ***Peer to Peer replikacija*** – uključuje više servera koji mogu da se ponašaju i kao master i kao slave čvorovi. Podaci se šalju i prihvataju od strane svakog čvora i sinhronizuju između svih čvorova. U ovakvom sistemu može doći do konflikta i nekonzistentnosti podataka ako se istim podacima pristupa i oni se ažuriraju na različitim čvorovima sistema. Rešenje bi bilo da se ažuriranje vrši samo na jednom čvoru, a promene se propagiraju na ostatak sistema. Šema peer to peer replikacije ilustrovana je na slici 1.3.



Slika 1.3 Šema peer to peer replikacije

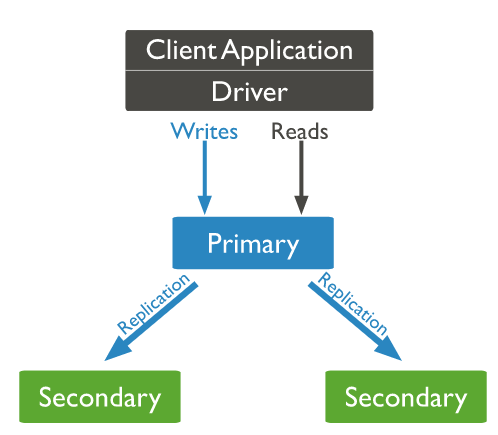
# replikacija podataka kod mongodb baze podataka

MongoDB koristi ***replica set***, koji predstavlja skup mongod instanci, za održavanje istog skupa podataka. Replica set-ovi obezbeđuju redundantnost i povećavaju dostupnost podataka. Sa više kopija podataka na različitim serverima, replikacija obezbeđuje određeni nivo tolerancije na otkaze servera. U nekim slučajevima, replikacija može da obezbedi povećanje kapaciteta za čitanje jer klijenti mogu da čitaju podatke sa rezličitih servera.

Replica set sadrži više čvorova sa podacima i, opciono, jedan čvor koji predstavlja ***arbitra***. Od čvorova koji čuvaju podatke, jedan čvor je ***primarni***, dok su ostali ***sekundarni***. Primarni čvor prima sve operacije za upis. Replica set može da ima samo jedan primarni čvor. Minimalna preporučena konfiguracija replica set-a je postojanje tri člana koji čuvaju podatke: jedan primarni čvor i dva sekundarna.

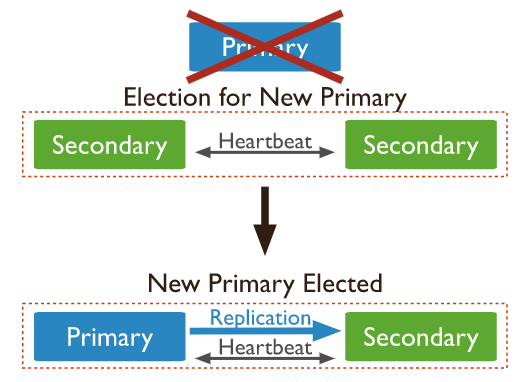
## Članovi replica set-a

***Primarni čvor*** je jedini član replica set-a koji prima operacije upisa. MongoDB primenjuje operacije upisa na primarni čvor i sve operacije čuva u oplog-u primarnog čvora. Sekundarni čvorovi repliciraju oplog i primenjuju ga nad svojim skupovima podataka. Svi članovi replica set-a mogu da prihvataju operacije čitanja. Po default-u aplikacija preusmerava sve operacije čitanja na primarni čvor. Default-no ponašanje za operacije čitanja se može promeniti. Na slici 2.1.1 prikazan je izgled jednog replica set-a sa defaul-nim ponašanjem.



Slika 2.1.1 Replica set

Ukoliko primarni čvor postane nedostupan, bira se novi primarni čvor među preostalim sekundarnim čvorovima. Detaljniji opis načina izbora primarnog čvora dat je u delu **2.3 Visoka dostupnost replica set-a**. Na slici 2.1.2 ilustrovan je proces odabira primarnog čalna.



Slika 2.1.2 Izbor primarnog člana

***Sekundarni čvorovi*** održavaju kopiju podataka primarnog čvora. Kako bi replicirao podatke, sekundarni čvor primenjuje operacije iz oplog-a primarnog čvora na svoj skup podataka. Replica set može da ima proizvoljan broj sekundarnih čvorova. Iako klijenti ne mogu da upisuju podatke u sekundarne čvorove, mogu da ih koriste za čitanje podataka. Sekundarni čvor može da postane primarni ukoliko primarni čvor postane nedostupan. Sekundarni čvorovi se mogu konfigurisati za neku specijalnu namenu:

* Sekundarni čvor se može sprečiti da postane primarni tokom izbora, kao i da započne proces izbora (***priority 0 član***). Osim ovih ograničenja, sekundarni čvorovi ovog tipa funkcionišu kao normalni sekundarni čvorovi: održavaju kopiju podataka, prihvataju operacije čitanja i glasaju tokom izbora. Sekundarni čvorovi ovog tipa mogu da funkcionišu kao standby, odnosno da čuvaju kopiju podataka kako bi mogli da zamene nedostupnog člana replica set-a.
* Aplikacije se mogu sprečiti da čitaju podatke sa tog čvora. Ovakvi čvorovi se zovu i ***sakriveni članovi***, jer nosu vidljivi klijentskim aplikacijama. Sakriveni članovi moraju biti *priority 0* članovi i ne mogu postati primarni, ali mogu da glasaju tokom izbora.
* ***Delayed članovi*** sadrže kopije podataka. Međutim, ti podaci predstavljaju raniju verziju podataka replica set-a. Članovi ovog tipa uglavnom služe za oporavak od raznih grešaka. Delayed članovi moraju biti *priority 0* članovi, a poželjno je i da budu *sakriveni*. Ovi članovi primenjuju operacije iz oplog-a primarnog člana, ali sa određenim zakašnjenjem.

***Arbitar*** ne sadrži kopiju podataka i ne može da postane primarni čvor. Međutim, arbitar može da glasa tokom procesa izbora primarnog čvora. Arbitri su *priority 0* članovi.

***Oplog*** je posebna kolekcija koja čuva sve operacije koje modifikuju podatke koji se čuvaju u bazi. MongoDB primenjuje operacije koje modifikuju podatke na primarni čvor. Te operacije se pamte u oplog-u primarnog čvora. Sekundarni čvorovi kopiraju i asinhrono primenjuju te operacije nad svojom kopijom podataka. Sve operacije u oplogu-u su idempotentne. Kako bi se olakšala replikacija, svi članovi replica set-a šalju jedni drugima ***heartbeat-ove***, kojima označavaju da su aktivni.

## Sinhronizacija podataka i semantika čitanja i upisa

Kako bi održavali važeće kopije podataka, sekundarni članovi replica set-a sinhronizuju ili repliciraju podatke od ostalih članova. MongoDB koristi dva načina za sinhronizaciju podataka: ***inicijalna sinhronizacija*** za dostavljanje svih podataka novim članovima i ***replikacija*** kako bi se primenile promene na ceo skup podataka.

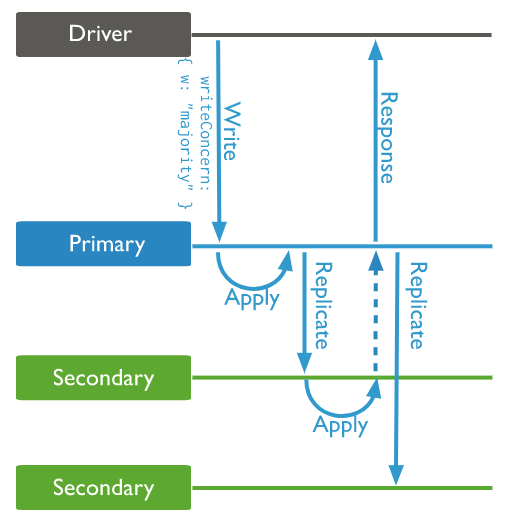
***Inicijalna sinhronizacija*** kopira sve podatke sa jednog na drugi čvor replica set-a. Proces inicijalne sinhronizacije je sledeći:

1. Kloniraju se sve baze podataka osim local baze, koja čuva podatke koji se koriste u procesu replikacije. Mongod skenira svaku kolekciju u svakoj bazi podataka i dodaje podatke u svoje kopije tih kolekcija.
2. Primenjuju se sve promene na skup podataka. Korišćenjem oplog-a, mongod ažurira svoj skup podataka kako bi reflektovao trenutno stanje replica set-a.

Sekundarni članovi ***repliciraju*** podatke kontinualno nakon inicijalne sinhronizacije. Oni kopiraju oplog iz svog *sync from* izvora i asinhrono primenjuju te operacije. Sekundarni članovi izbegavaju sinhronizaciju sa delayed i sakrivenim članovima.

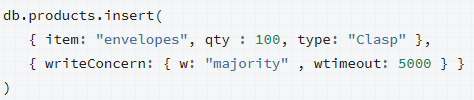
Prilikom upisa podataka, potrebno je da određeni broj članova replica set-a potvrdi operaciju upisa pre nego što se ona završi kao uspešna (***write concern***). Član može da potvrdi operaciju upisa samo nakon što je dobije i uspešno primeni. Za replica set-ove default-ni broj članova koji treba da potvrde upis je 1 (w: 1), odnosno samo primarna replika treba da potvrdi upis kako bi se on završio uspešno. Može se specificirati bilo koji broj veći od 1 kako bi se zahtevala potvrda od primarnog čvora i onoliko sekundarnih čvorova koliko je potrebno da se zadovolji specificirana vrednost. Operacije upisa kod kojih se traži potvrda većine zahtevaju potvrdu da je operacija obavljena od izračunate većine članova koji čuvaju podatke (***w: majority***). Većina se računa kao veća vrednost od broja članova koji čuvaju podatke i većina svih članova koji mogu da glasaju (uključujući arbitre).

Aplikacija koja izda operaciju za upis čeka dok primarni čvor ne dobije potvrdu od potrebnog broja članova data set-a. Tek kada potreban broj sekundarnih čvorova potvrdi upis, primarni čvor vraća potvrdu uspešnog upisa. Ukoliko je taj broj 1, primarni čvor može da vrati potvrdu uspešnog upisa čim lokalno primeni operaciju upisa. Proces obrade operacije upisa ilustrovan je na slici 2.2.1. Što više članova treba da potvrdi upis, to se više povećava latentnost operacije jer klijent mora da čeka dok se ne dobije odgovarajući broj potvrda.



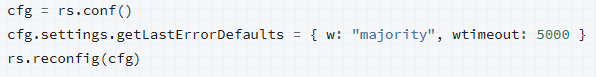
Slika 2.2.1 Obrada operacije upisa

Sledeći primer na slici 2.2.2 ilustruje upotrebu ***writeConcern*** opcije, kojom se specificira broj potrebnih potvrda za upis, u okviru *insert()* metode. Specificirana je većina (majority), kao i timeout parametar kako bi se izbeglo beskonačno blokiranje operacije. Ukoliko istekne specificirano vreme, to ne mora da znači da primarni čvor nije uspeo da primeni zadatu operaciju, već da zahtevani broj članova nije potvrdio upis u okviru zadatog vremena. Podaci mogu da postoje u određenom podskupu čvorova u trenutku isteka zadatog vremena i može se nastaviti replikacija dok svi čvorovi ne dobiju potrebne podatke.



Slika 2.2.2 Upotreba writeConcern opcije

Moguće je promeniti podrazumevanu vrednost za broj potvrda upisa u okviru konfiguracije replica set-a korišćenjem ***settings.getLastErrorDefaults*** opcije. Ovo je ilustrovano u primeru na slici 2.2.3 gde je potrebna potvrda upisa većine članova. Ukoliko se izda operacija upisa sa specificiranom *writeConcern* opcijom, podrazumevana vrednost se ignoriše.



Slika 2.2.3 Podešavanje podrazumevane vrednosti za broj potvrda upisa

Po default-u, aplikacija prosleđuje operacije čitanja primarnom članu replica set-a. Moguće je promeniti kome se prosleđuju operacije čitanja i čitati iz sekundarnih čvorova. Može se specificirati način čitanja (***read preference mode***) i, opciono, skup ***tagova*** i maksimalna starost podataka (***maxStalenessSeconds***). Podržano je više različitih načina čitanja:

* ***primary*** – sve operacije čitanja koriste samo primarni čvor replica set-a. Ovo je podrazumevani način čitanja. Ukoliko primarni čvor nije dostupan, opracija čitanja se završava greškom. Ovaj način čitanja ne podržava skup tagova ili maksimalnu starost podataka.
* ***primaryPreferred*** – ukoliko primarni čvor nije dostupan, čita se iz sekundarnog člana koji zadovoljava zadate tagove i maksimalnu starost podataka. Ukoliko je zadata maksimalna starost podataka, vrši se procena starosti podataka svakog sekundarnog čvora upoređivanjem vremena poslednje operacije upisa sa vremenom upisa čvora koji je najskorije izvršio upis. Operacija čitanja se prosleđuje čvoru čija je starost podataka manja ili jednaka zadatoj maksimalnoj vrednosti. Ukoliko su zadati tagovi, traže se sekundarni čvorovi čiji se tagovi poklapaju sa zadatim i operacija čitanja se prosleđuje nasumično izabranom čvoru iz grupe najbližih (način odabira čvora biće detaljnije opisan u nastavku).
* ***secondary*** – čita se isključivo iz sekundarnih čvorova. Ukoliko ni jedan sekundarni čvor nije dostupan, biće vraćena greška. Ukoliko je zadata maksimalna starost podataka, vrši se procena starosti podataka svakog sekundarnog čvora upoređivanjem vremena poslednjeg upisa sa vremenom poslednjeg upisa primarnog čvora. Operacija čitanja se prosleđuje čvoru čija je starost podataka manja ili jednaka zadatoj maksimalnoj vrednosti. Ukoliko primarni čvor nije dostupan, upoređivanje se vrši za vreme poslednjeg upisa sekundarnog čvora koji je najskorije izvršio operaciju upisa. Sa tagovima se radi na isti način kao u slučaju primaryPreferred.
* ***secondaryPreferred*** – ukoliko nema sekundarnih čvorova, čita se iz primarnog čvora. Ukoliko je zadata maksimalna starost podataka, procena starosti podataka se vrši na isti način kao u slučaju secondary. Ukoliko ne postoji ni jedan sekundarni čvor koji zadovoljava zadati uslov maksimalne starosti podataka, operacija čitanja se prosleđuje primarnom čvoru. Sa tagovima se radi na isti način kao u slučajevima *primaryPreferred* i *secondary*.
* ***nearest*** – čita se iz čvora čija se latentnost uklapa u zadati okvir. Primarni i sekundarni čvorovi se isto tretiraju. Ovaj način čitanja se koristi kako bi se minimizovala latentnost opracija čitanja. Ukoliko je zadata maksimalna starost podataka, vrši se procena starosti podataka svakog sekundarnog čvora upoređivanjem vremena poslednjeg upisa sa vremenom poslednjeg upisa primarnog čvora. Ukoliko primarni čvor nije dostupan, upoređivanje se vrši za vreme poslednjeg upisa sekundarnog čvora koji je najskorije izvršio operaciju upisa. Zatim se vrši izbacivanje sekundarnih čvorova čija je starost podataka veća od zadate maksimalne vrednosti. Od preostalih čvorova se nasumično bira jedan čija se letentnost uklapa u zadati okvir. Sa tagovima se radi na isti način kao u slučaju *primaryPreferred*.

Na slici 2.2.4 dat je primer specificiranja načina čitanja podataka korišćenjem ***readPref()*** metode. Kao parametri metode se prosleđuju način čitanja i skup tagova. Opcija za maskimalnu starost podataka nije podržana. Ova opcija se može specificirati kroz konekcioni string korišćenjem MongoDB drajvera.



Slika 2.2.4 Specificiranje načina čitanja podataka

MongoDB koristi ***Server Selection algoritam*** za odabir člana replica set-a koji će koristiti za čitanje podataka. Odabir servera se vrši jednom po operaciji i zavisi od odabranog načina čitanja.

* *primary* način čitanja – uvek se bira primarni čvor
* *secondary* način čitanja – uzima se lista sekundarnih čvorova koji zadovoljavaju zadate uslove (maksimalna starost i skup tagova). Ukoliko lista nije prazna, utvrđuje se koji član je nabliži (npr. član sa najmanjim povratnim vremenom) i računa okvir latentnosti. Iz liste članova koji se uklapaju u okvir latentnosti nasumično se bira jedan član.
* *nearest* način čitanja – uzima se lista čvorova koji zadovoljavaju zadate uslove (primarni i sekundarni čvorovi). Ukoliko lista nije prazna, utvrđuje se koji član je nabliži (npr. član sa najmanjim povratnim vremenom) i računa okvir latentnosti. Iz liste članova koji se uklapaju u okvir latentnosti nasumično se bira jedan član.
* *primaryPreferred* način čitanja – ukoliko je primarni čvor dostupan, bira se on. U suprotnom, prati se proces selekcije čvora kao kod secondary načina čitanja.
* *secondaryPreferred* način čitanja – praćenjem procesa selekcije čvora kao kod secondary načina čitanja, ukoliko lista sekundarnih članova nije prazna, bira se jedan sekundarni čvor. U suprotnom, ukoliko je lista prazna, bira se primarni čvor.

Opcija ***readConcern*** omogućava kontrolu konzistentnosti podataka koji se čitaju iz replica set-a. Postoji više nivoa za ovu opciju:

* ***local*** – upit vraća podatke sa instance bez garancije da su ti podaci upisani od strane većine članova replica set-a. Ovo je podrazumevana vrednost za čitanje iz primarnog čvora i za čitanje iz sekundarnih čvorova ukoliko su operacije čitanja povezane sa konzistentnim sesijama.
* ***available*** – upit vraća podatke sa instance bez garancije da su ti podaci upisani od strane većine članova replica set-a. Ovo je podrazumevana vrednost za čitanje iz sekundarnih čvorova ukoliko operacije čitanja nisu povezane sa konzistentnim sesijama.
* ***majority*** – upit vraća podatke koji su potvrđeni od strane većine članova replica set-a.
* ***linearizable*** – upit vraća podatke potvrđene od strane većine članova replica set-a pre početka operacije čitanja. Upit može da čeka da se operacije upita koje se konkurentno izvršavaju propagiraju do većine članova pre vraćanja rezultata.
* ***snapshot*** – vrednost dostupna samo za multi-document transakcije.

Ova opcija se može eksplicitno specificirati uz komande i metode koje vrše čitanje podataka. Na slici 2.2.5 ilustrovano je korišćenje *readConcern* opcije.



Slika 2.2.5 Specificiranje readConcern opcije

## Visoka dostupnost replica set-a

Replica setovi koriste ***izbore (elections)*** kako bi obezbedili visoku dostupnost. Replica setovi koriste izbore kako bi odlučili koji član će postati primarni. Izbori se mogu pokrenuti kao odgovor na više događaja, poput:

* dodavanje novog čvora u replica set
* inicijalizacija replica set-a
* održavanje replica set-a korišćenjem rs.reconfig() ili rs.stepDown() metoda
* sekundarni član je izgubio konekciju sa primarnim (primarni član je bio nedostupan više od konfigurisanog timeout-a)

Član koji je pokrenuo izborni proces glasa za sebe i pita sve ostale čvorove da li glasaju za njega. Čvorovi šalju svoj odgovor i, ukoliko je kandidat za primarni čvor dobio glasove od većine čvorova, on dobija izbore i postaje primarni član. Ukoliko se bivši primarni čvor oporavi i vrati u replica set, nastavlja rad kao sekundarni čvor.

Replica set ne može da obavlja operacije upisa dok se proces izbora uspešno ne završi. Operacije čitanja se mogu obavljati samo ukoliko se čitanje obavlja na sekundarnim čvorovima.

Članovi replica set-a šalju signale (***heartbeats***) jedni drugima na svake dve sekunde. Podaci sadržani u poslatom heartbeat-u uključuju verziju konfiguracije, id pošiljaoca, naziv replica set-a, adresu hosta pošiljaoca. Kada čvor primi heartbeat, on prvo procesira dobijene podatke, a zatim šalje odgovor. Ukoliko pošaljalac sadrži noviju verziju konfiguracije, odredišni čvor traži noviju verziju. Ukoliko se heartbeat odgovor ne vrati nakon 10 sekundi, ostali članovi označavaju tog člana kao nedostupnog. Podaci sadržani u odgovoru uključuju naziv replica set-a, vreme za izbore odredišnog čvora, stanje odredišnog čvora, verziju konfiguracije odredišnog čvora, da li je odredišni čvor primarni.

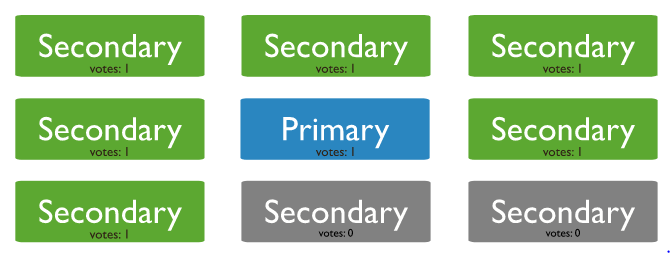
Poželjno je obezbediti da sekundarni čvor sa najvećim prioritetom bude dostupan da pokrene izborni proces. Prioritet člana utiče na vreme i rezultate izbora. Sekundarni članovi sa većim prioritetom pokreću izborni proces pre nego sekundarni čvorovi sa nižim prioritetom i veća je verovatnoća da oni pobede na izborima. Instance sa nižim prioritetom mogu biti izabrane za primarne na kratak vremenski period, čak iako je sekundarni čvor sa većim prioritetom dostupan. Članovi replica set-a nastavljaju da pokreću izborne procese sve dok sekundarni čvor sa najvećim prioritetom ne postane primarni. Članovi sa prioritetom 0 ne mogu da postanu primarni i ne traže izbore.

Kada je u pitanju distribuirani replica set, gubitak data centra može da utiče na mogućnost ostalih članova u drugim data centrima da izaberu primarnog člana. Ukoliko je moguće, bilo bi dobro distribuirati članove replica set-a po data centrima kako bi se maksimizovala verovatnoća da čak i sa gubitkom data centra, jedan od preostalih članova replica set-a može da postane novi primarni član.

Kada primarni čvor detektuje da može da vidi manjinu čvorova u replica set-u, on postaje sekundarni čvor. Nezavisno, član koji može da komunicira sa većinom čvorova (uključujući sebe) pokreće izbore kako bi postao novi primarni čvor.

Konfiguraciona opcija ***members[n].votes*** i stanje (***state***) člana određuju da li član može da glasa tokom procesa izbora. Članovi koji imaju *members[n].votes* vrednost 1 glasaju tokom izbora. Da bi se član isključio iz procesa glasanja, ova opcija se postavlja na 0. Članovi koji ne glasaju moraju imati prioritet 0, a članovi koji imaju prioritet različit od 0 ne mogu imati votes parametar postavljen na 0. Članovi u sledećim stanjima mogu da glasaju: PRIMARY, SECONDARY, STARTUP2 (konfiguracija je učitana i član odlučuje da li će pokrenuti inicijalnu sinhronizaciju), RECOVERING (član replica set-a nije spreman da prihvata operacije upisa), ARBITER, ROLLBACK.

Iako neki članovi ne glasaju tokom procesa izbora, ovi članovi čuvaju kopije podataka i mogu da prihvate operacije čitanja od klijentskih aplikacija. Replica set može da ima do 50 članova, ali samo 7 mogu da glasaju, članovi koji ne mogu da glasaju omogućavaju da replica set može da ima više od 7 članova. Članovi koji ne mogu da glasaju imaju i prioritet i votes vrednost 0. Na slici 2.3.1 prikazan je replica set od 9 članova, od kojih 7 mogu da glasaju, a 2 ne mogu.



Slika 2.3.1 Replica set od 9 članova

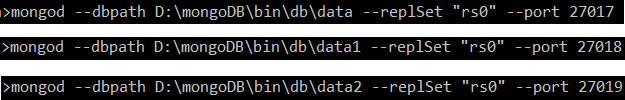
***Rollback*** poništava operacije upisa na bivšem primarnom članu kada se ponovo vrati u replica set nakon greške. Rollback je neophodan samo ako je primarni član prihvatio operacije upisa koje sekundarni članovi nisu uspešno replicirali pre nego što je primarni član postao nedostupan. Kada se bivši primarni član vrati u replica set kao sekundarni, on radi rollback opracija upisa kako bi održao konzistentnost baze podataka sa ostalim članovima. Rollback se ne dešava ukoliko je operacija upisa replicirana na drugi član replica set-a pre nego što je primarni član postao nedostupan i ako je taj član ostao dostupan većini članova replica set-a. Ukoliko se koristi default-ni write concern {w:1}, rollback će se odraditi ukoliko primarni član postane nedostupan pre nego što se podaci repliciraju na bilo koji sekundarni čvor.

Podrazumevano je da, ukoliko se desi rollback, MongoDB upisuje rollback podatke u BSON fajlove. Za svaku kolekciju za čije podatke se radi rollback, rollback fajlovi se čuvaju u <dbpath>/rollback/<db>.<collection> direktorijumu. Ukoliko je operacija čiji se rollback radi brisanje dokumenta ili kolekcije, ne kreiraju se rollback fajlovi sa podacima. Klijenti koji koriste local ili available read concern mogu da vide rezultate operacije upisa pre nego što se operacija potvrdi klijentu koji je izdao. Klijenti koji koriste local ili available read concern mogu da čitaju podatke nad kojima se naknadno može uraditi rollback zbog otkaza u replica set-u.

## Kreiranje i podešavanje replica set-a

U ovom delu biće dat primer kreiranja replica set-a od tri člana. Replica set-ovi bi uvek trebalo da imaju neparan broj članova kako bi proces glasanja prošao bez problema.

Najpre je potrebno stratovati svaki član kao mongod instancu sa odgovarajućim opcijama. Kao opcije se navode putanja do direktorijuma sa podacima, naziv replica set-a, kao i port na kome će instanca slušati. Na slici 2.4.1 je primer kreiranja tri instanci replica set-a “rs0”, koje slušaju na portovima 27017, 27018 i 27019.



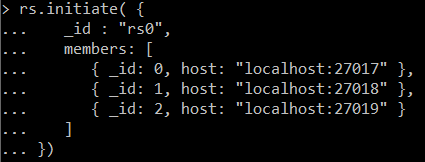
Slika 2.4.1 Kreiranje instanci replica set-a

Nakon toga je potrebno konektovati se na jednu od pokrenutih instanci. U ovom primeru ćemo se konektovati na instancu koja sluša na portu 27017, kao što je prikazano na slici 2.4.2.



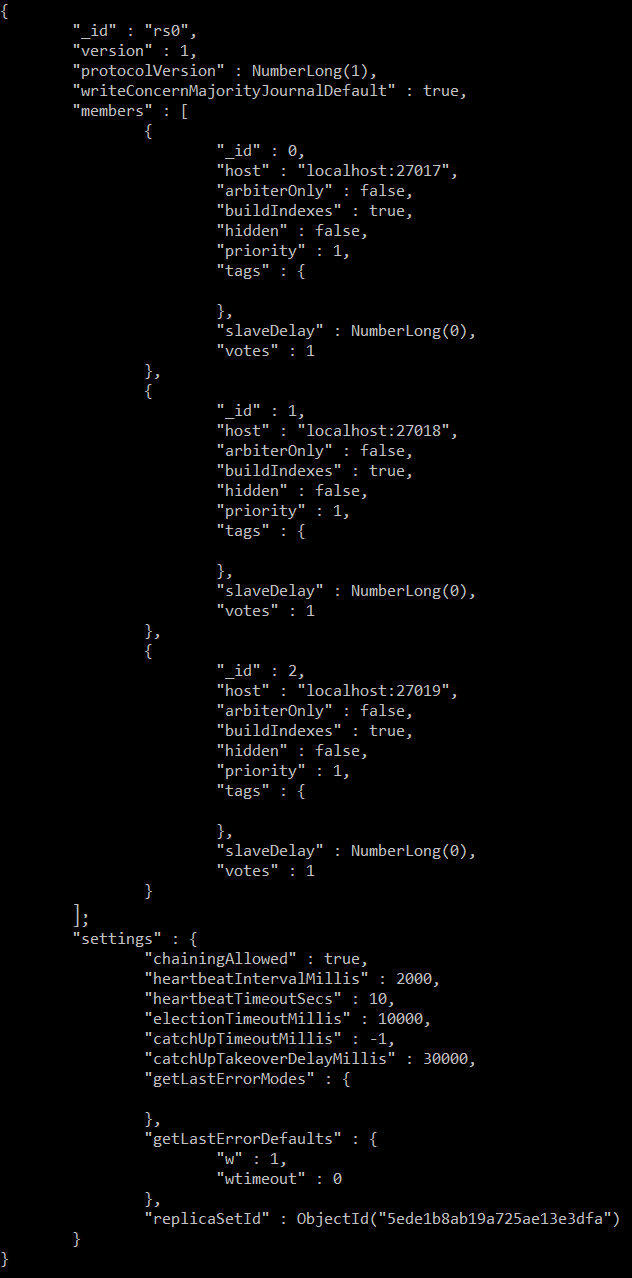
Slika 2.4.2 Konektovanje na instancu

Potom je potrebno izvršiti inicijalizaciju replica set-a sa instance na koju smo se konektovali. Inicijalizacija se vrši korišćenjem ***rs.initiate()*** metode, kojoj se prosleđuju odgovarajući parametri za inicijalizaciju (naziv replica set-a, identifikatori i adrese hostova na kojima se izvršavaju članovi). Inicijalizacija replica set-a je ilustrovana na slici 2.4.3.



Slika 2.4.3 Inicijalizacija replica set-a

Pozivom ***rs.conf()*** funkcije može se videti konfiguracioni objekat replica set-a. Ovaj objekat sadrži informacije o replica set-u (id, heartbeat interval, itd.) i njegovim članovima (id, adresa hosta na kome se izvršava, da li je arbitar, da li je sakriven član, prioritet, votes, tagovi). Na slici 2.4.4 je prikazan izgled konfiguracionog objekta. Pozivom ***rs.status()*** funkcije može se, pored osnovnih informacija o replica set-u, videti i koji član replica set-a je primarni. Ova informacija se može videti u members delu. Na slici 2.4.5 prikazane su informacije o primarnom i sekundarnim članovima kreiranog replica set-a.

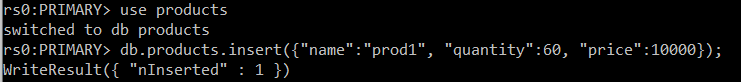


Slika 2.4.4 Konfiguracioni objekat replica set-a

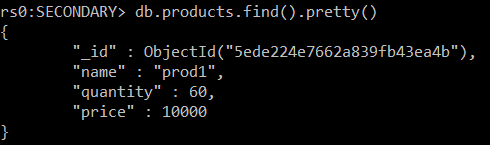


Slika 2.4.5 Primarni i sekundarni članovi replica set-a

Da bismo testirali rad kreiranog replica set-a, dodaćemo dokument u *products* kolekciju korišćenjem primarnog člana, što je ilustrovano na slici 2.4.6. Nakon toga ćemo pročitati podatke korišćenjem bilo koje sekundarne replike (slika 2.4.7).

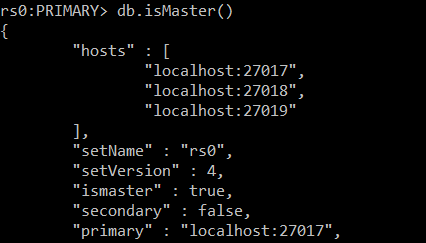


Slika 2.4.6 Dodavanje podataka korišćenjem primarnog člana



Slika 2.4.7 Čitanje podataka korišćenjem sekundarnog člana

U postojeći replica set mogu se dodavati novi članovi. Replica set može da sadrži najviše 7 članova koji mogu da glasaju. Ukoliko se dodaje novi član u replica set koji već sadrži 7 članova koji mogu da glasaju, potrebno je ili da novi član ne može da glasa ili da se nekom već postojećem članu oduzme pravo glasa. Dodavanje novog člana počinje tako što se prvo startuje nova mongod instanca na isti način kao u primeru na slici 2.4.1. Zatim se vrši konekcija na primarni član replica set-a. Ukoliko je potrebno utvrditi koji član je primarni, poziva se metoda ***db.isMaster()*** na bilo kom članu replica set-a. Rezultati poziva ove etode prikazani su na slici 2.4.8.



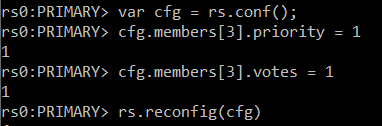
Slika 2.4.8 Rezultati poziva isMaster() metode

Nakon toga se, u okviru primarnog člana, poziva funkcija za dodavanje novog člana replica set-a, ***rs.add()***. Kao parametri ove funkcije prosleđuju se adresa hosta na kome se izvršava član koji se dodaje, njegov prioritet i da li može da glasa (votes opcija). Primer dodavanja novog člana u replica set prikazan je na slici 2.4.9.



Slika 2.4.9 Dodavanje novog člana u replica set

Pozivom ***rs.status()*** metode se može proveriti da li je dodati član sekundarni. Ukoliko je potrebno promeniti prioritet i votes opciju dodatog člana (ili bilo kog drugog člana), to se može uraditi korišćenjem ***rs.reconfig()*** metode. U postojećoj konfiguraciji se menjaju ove opcije za određenog člana (potrebno je znati lokaciju člana u members nizu) i pozivom ove metode se ažurira konfiguracija replica set-a. Na slici 2.4.10 ilustrovan je način promene prioriteta i votes opcije poslednje dodatog člana replica set-a.



Slika 2.4.10 Rekonfiguracija člana replica set-a

Ukoliko se u replica set dodaje član koji je arbitar, umesto *rs.add()* metode, koristi se ***rs.addArb()*** metoda, kojoj se kao parametar prosleđuje adresa hosta na kome se član izvršava, kao što je prikazano na slici 2.4.11.



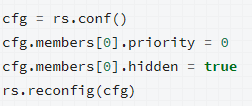
Slika 2.4.11 Dodavanje arbitra u replica set

Uklanjanje člana iz replica set-a se postiže pozivom ***rs.remove()*** metode. Kao argument se prosleđuje adresa hosta na kome se izvršava član koga treba izbaciti. Pre poziva metode je potrebno ugasiti mongod instancu odgovarajućeg člana i konektovati se na primarnu instancu, odakle će se pozvati funkcija. Na slici 2.4.12 prikazan je način uklanjanja člana iz replica set-a.



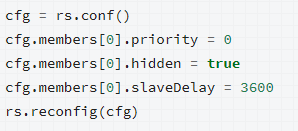
Slika 2.4.12 Uklanjanje člana iz replica set-a

Ukoliko je potrebno konfigurisati neki član replica set-a tako da ne može da postane primarni, samo mu je potrebno postaviti *prioritet* na 0, na isti način kao što je ilustrovano na slici 2.4.10. Ukoliko se neki član konfiguriše kao sakriven, prioritet mu se postavlja na 0, a opcija hidden na true, kao što je ilustrovano na slici 2.4.13.



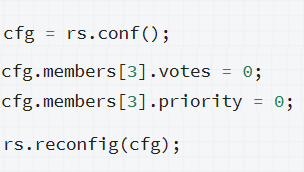
Slika 2.4.13 Konfiguracija sakrivenog člana

Za konfiguraciju delayed člana, *prioritet* se postavlja na 0, *hidden* opcija na true, a *slaveDelay* opcija na broj sekundi kašnjenja. Konfiguracija delayed člana prikazana je na slici 2.4.14.



Slika 2.4.14 Konfiguracija delayed člana

Za konfiguraciju člana replica set-a tako da ne može da glasa, i *prioritet* i *votes* opcije mu se postavljaju na 0. Konfiguracija člana koji ne može da glasa prikazana je na slici 2.4.15.



Slika 2.4.15 Konfiguracija člana koji ne može da glasa

Ukoliko imamo sekundarnog člana replica set-a koji ne mora više da čuva podatke, ali treba da ostane zbog procesa glasanja, može se konverovati u arbitra. Sekundarni član se isključuje, uklanja se iz replica set-a, prazni se folder sa podacima, restartuje se mongod instanca i ona se dodaje kao arbitar.

# zaključak

Replikacija podataka je bitna kako bi se obezbedila redundantnost, povećala dostupnost i olakšali neki administrativni zadaci poput pravljenja rezervnih kopija. MongoDB nudi mehanizam replikacije koji omogućava sinhronizaciju podataka između većeg broja instanci.

Između više instanci se uspostavlja mehanizam replikacije. Jedna instanca je primarna i u njoj se upisuju podaci koji se sinhronizuju sa ostalim, sekunadrnim instancama. Čitanje se može vršiti pribavljanjem podataka sa bilo koje instance. Takođe, MongoDB obezbeđuje i failover mehanizam ukoliko dođe do otkaza primarne instance. Mehanizmom glasanja jedna od sekundarnih instanci postaje primarna.

MongoDB predstavlja dobro rešenje za čuvanje podataka, uz mogućnost njihove replikacije. Na taj način se može obezbediti dostupnost podataka, čak i ako dođe do otkaza neke instance.

# literatura

1. Database Management Systems, R. Ramakrishnan and J. Gehrke, McGraw-Hill; 3rd edition, 2002
2. Fundamentals of Database Systems, Addison Wesley, 6th edition, 2010
3. <https://www.geeksforgeeks.org/data-replication-in-dbms/>
4. <https://www.includehelp.com/dbms/data-replication-and-its-types.aspx>
5. Dokumentacija MongoDB baze podataka

<https://docs.mongodb.com/manual/replication/>

1. <https://github.com/mongodb/mongo/blob/master/src/mongo/db/repl/README.md>