*Elektronski fakultet Niš*



*Sistemi za upravljanje bazama podataka* **Replikacija MongoDB baze podataka**

Mentor:

prof. dr Aleksandar Stanimirović

Student:

David Bosnić 1289

Sadržaj:

[Uvod 3](#_Toc104914772)

[Replikacija i skup replika 4](#_Toc104914773)

[Primarni član 4](#_Toc104914774)

[Sekundarni član 6](#_Toc104914775)

[Članovi skupa replika nultog prioriteta 6](#_Toc104914776)

[Skriveni članovi skupa replika 7](#_Toc104914777)

[Odloženi članovi skupa replika 7](#_Toc104914778)

[Arbitar skupa replika 8](#_Toc104914779)

[Oplog 10](#_Toc104914780)

[Veličina oplog-a 10](#_Toc104914781)

[Minimalni period zadržavanja Oploga 11](#_Toc104914782)

[Radna opterećenja koja mogu zahtevati veću veličinu oplog-a 11](#_Toc104914783)

[Status oplog-a 12](#_Toc104914784)

[Kašnjenje replikacije i kontrola toka 12](#_Toc104914785)

[Spora Oplog aplikacija 12](#_Toc104914786)

[Ponašanje oplog kolekcija 12](#_Toc104914787)

[Sinhronizacija podataka skupa replika 14](#_Toc104914788)

[Tolerancija grešaka 14](#_Toc104914789)

[Početni izbor izvora sinhronizacije 15](#_Toc104914790)

[Arhitekture postavljanja skupa replika 16](#_Toc104914791)

[Strategije prilikom postavljanja arhitekture 16](#_Toc104914792)

[Balans opterećenja na implementacijama sa velikim opterećenjem 17](#_Toc104914793)

[Imenovanje skupa replika 17](#_Toc104914794)

[Implementacioni obrasci 17](#_Toc104914795)

[Tročlana arhitektura skupa replika 17](#_Toc104914796)

[Skupovi replika distribuirani u dva ili više centara podataka 19](#_Toc104914797)

[Semantika čitanja i pisanja skupa replika 21](#_Toc104914798)

[Primer kreiranja tročlanog skupa replika 23](#_Toc104914799)

[Zaključak 26](#_Toc104914800)

[Reference 27](#_Toc104914801)

Uvod

*MongoDB* je dokumentu orijentisana baza podataka koja se koristi u mnogim modernim veb aplikacijama. Klasifikovan je kao *NoSQL* baza podataka, jer se ne oslanja na tradicionalnu strukturu relacione baze podataka zasnovanu na tabeli. Umesto toga, koristi dokumente slične JSON-u sa dinamičkim šemama. To znači da, za razliku od relacionih baza podataka, *MongoDB* ne zahteva unapred definisanu šemu pre dodavanja podataka u bazu podataka.

Prilikom rada sa bazama podataka, često je korisno imati više kopija podataka. Ovo obezbeđuje redundantnost u slučaju kvara jednog od servera baze podataka i može poboljšati dostupnost i skalabilnost baze podataka, kao i smanjiti kašnjenje čitanja. Praksa sinhronizacije podataka u više zasebnih baza podataka naziva se replikacija. U *MongoDB* bazi podataka, grupa servera koji održavaju isti skup podataka kroz replikaciju se naziva skupom replika.

Ovaj rad pruža kratak pregled kako replikacija funkcioniše u *MongoDB*-u i opisuje kako se konfiguriše i pokreće skup replika. U ovom primeru konfiguracije, svaki član skupa replika biće posebna MongoDB instanca.

Replikacija i skup replika

Skup replika u *MongoDB-*u je grupa *mongod* procesa koji održavaju isti skup podataka. *Mongod* je primarni demon proces za *MongoDB* sistem. On obrađuje zahteve za podacima, upravlja pristupom podacima i obavlja operacije upravljanja u pozadini. Skupovi replika obezbeđuju redundantnost i visoku dostupnost i predstavljaju osnovu za sve primene u proizvodnji. Replikacija obezbeđuje redundantnost i povećava dostupnost podataka. Sa više kopija podataka na različitim serverima baze podataka, replikacija obezbeđuje nivo tolerancije na greške izazvane gubitkom jednog od servera baze podataka. U nekim slučajevima, replikacija može da obezbedi povećan kapacitet čitanja jer klijenti mogu da šalju operacije čitanja na različite servere. Održavanje kopija podataka u različitim centrima podataka može povećati lokalizaciju podataka i dostupnost za distribuirane aplikacije. Takođe moguće je održavati dodatne kopije za namenske svrhe, kao što su oporavak od katastrofe, izveštavanje ili pravljenje rezervnih kopija.

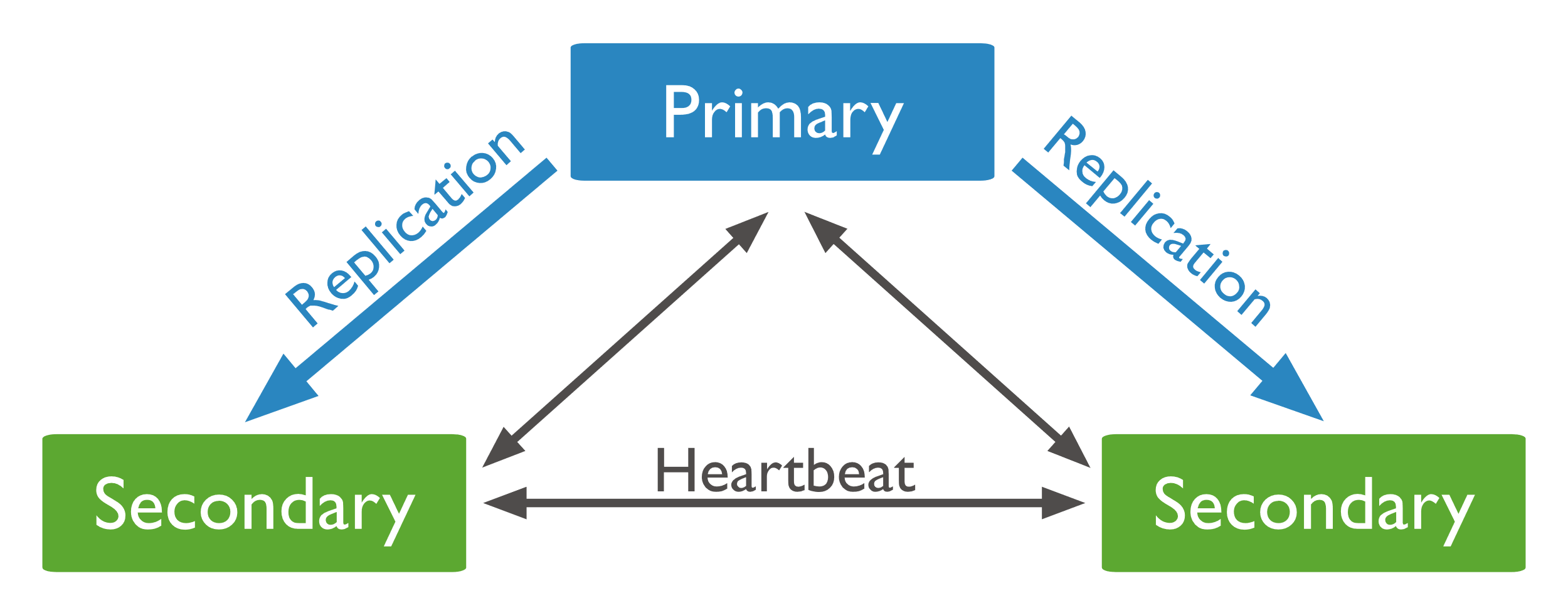
Članovi skupa replika mogu biti:

* Primarni – Primaju sve operacije pisanja;
* Sekundarni – Repliciraju operacije sa primarnog da bi održali identičan skup podataka. Sekundari mogu imati dodatne konfiguracije za posebne profile upotrebe. Na primer, sekundari mogu biti bez glasanja ili prioriteta 0;

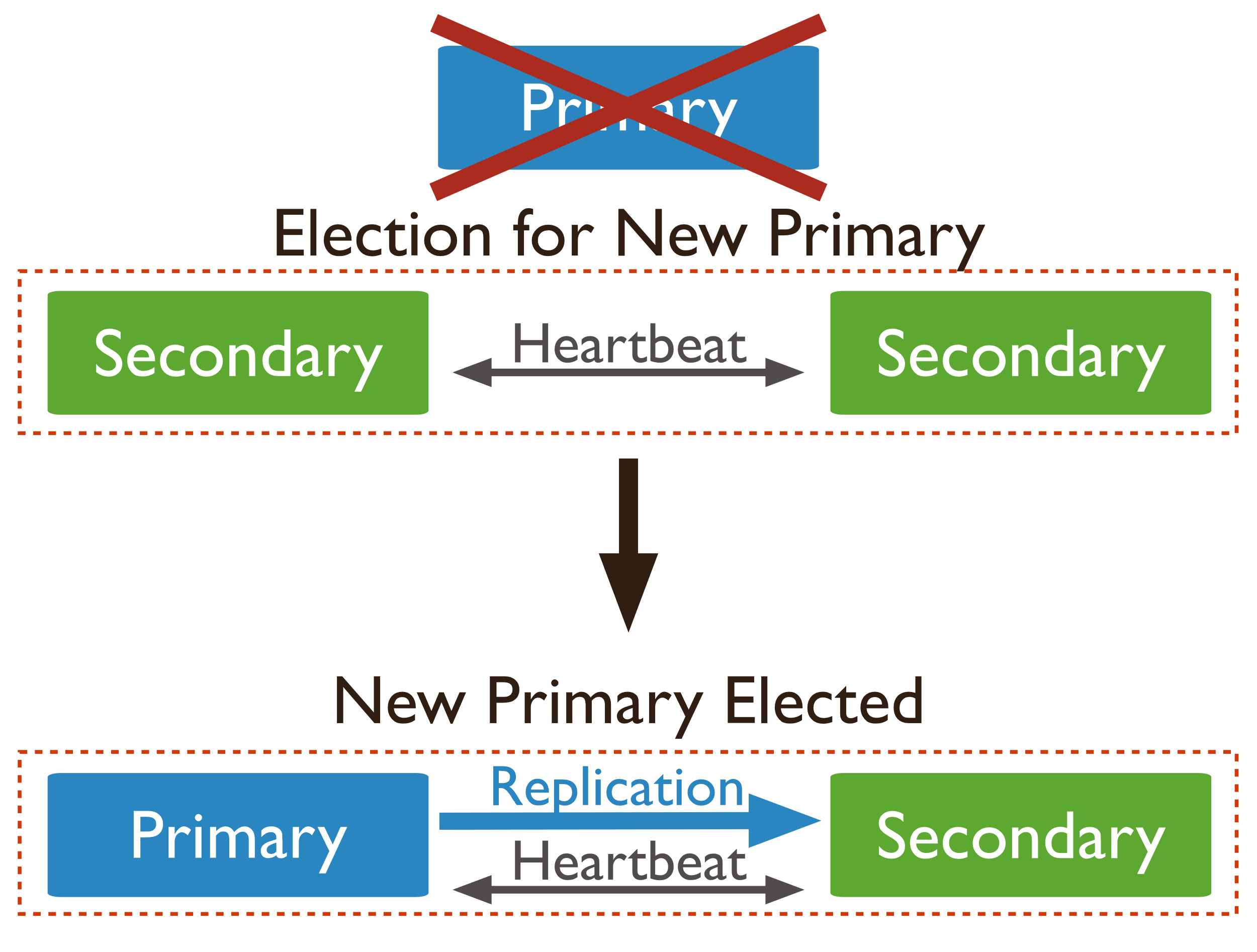
Minimalna preporučena konfiguracija za skup replika je tročlani skup replika, sa tri člana koji nose podatke: jednim primarnim i dva sekundarna člana. U nekim okolnostima (kao što je posedovanje primarnog i sekundarnog, ali ograničenja troškova zabranjuju dodavanje drugog sekundarnog), se može izabrati uključivanje arbitra. Arbitar učestvuje na izborima, ali ne poseduje podatke (tj. ne obezbeđuje redundantnost podataka). Skup replika može imati do 50 članova, ali samo 7 članova sa pravom glasa.

# **Primarni član**

Primarni član ili primar je jedini član u skupu replika koji prima operacije pisanja. *MongoDB* primenjuje operacije pisanja na primarnom, a zatim snima operacije na *oplog-*u primarnog, o kome će biti reči kasnije. Sekundarni članovi repliciraju ovaj dnevnik i primenjuju operacije na svoje skupove podataka. U tročlanom skupu replika prikazanom na slici 1, primar prihvata sve operacije pisanja. Zatim sekundari repliciraju *oplog,* da bi primenili te operacije na svoje skupove podataka. Iako klijenti ne mogu slati zahteve za upisom podataka sekundarima, klijenti mogu čitati podatke od sekundarnih članova. Sekundar može postati primar. Ako trenutni primar postane nedostupan, skup replika održava izbore da bi izabrao koji od sekundarnih jedinica postaje novi primar. U tročlanom skupu replika prikazanom na slici 2, primar postaje nedostupan. Ovo pokreće izbore gde jedan od preostalih sekundarnih kandidata postaje novi primar.



*Slika 1 – Primer tročlanog skupa replika*

**

*Slika 2 – Primer izbora prilikom otkaza primarnog člana*

Pod nekim okolnostima, dva čvora u skupu replika mogu privremeno da veruju da su primarni, ali najviše, jedan od njih će moći da izvrši upisivanje sa { w: " *majority* " } dozvolom za pisanje. Čvor koji ima { w: " *majority* " } dozvolu za upisivanje je trenutni primar, a drugi čvor je kvazi primar koji još uvek nije prepoznao svoje degradiranje, obično zbog mrežne particije. Kada se to dogodi, klijenti koji se povežu sa prethodnim primarnim čvorom mogu da primete zastarele podatke uprkos tome što su zahtevali primarnu preferenciju za čitanje, a nova upisivanja na bivši primar će se na kraju vratiti nazad.

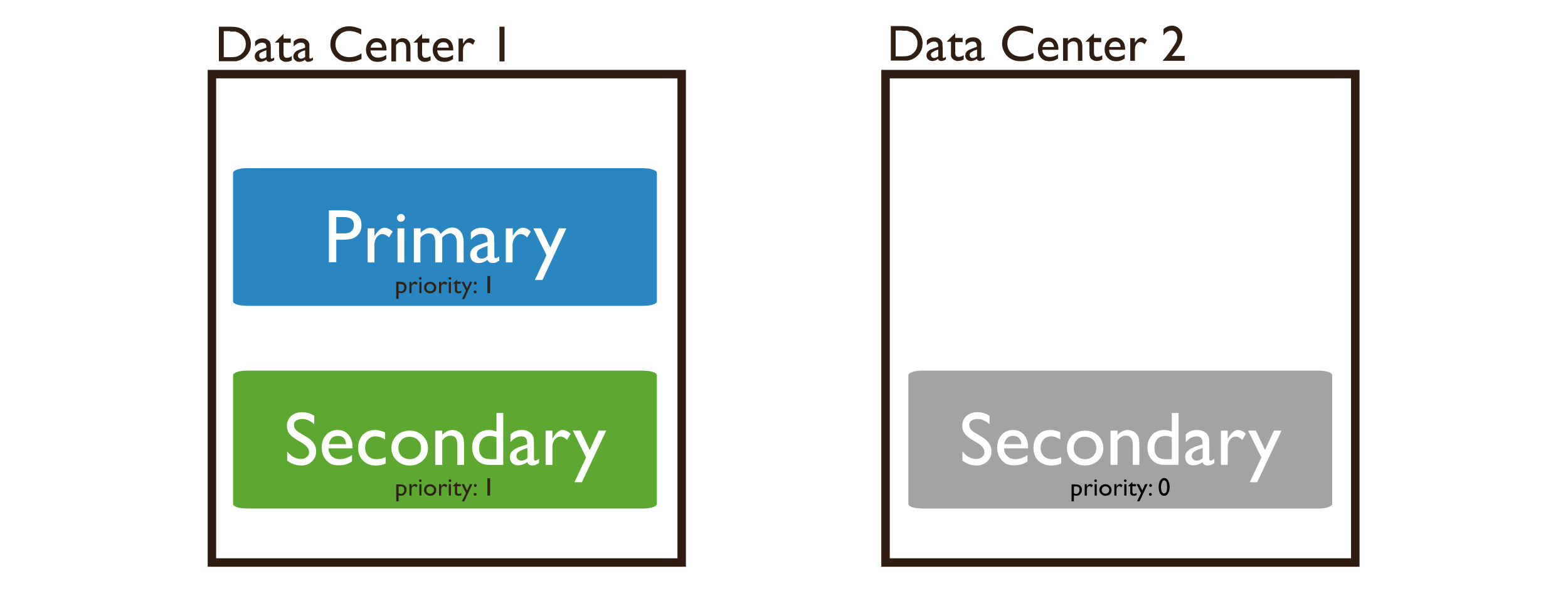
# **Sekundarni član**

Sekundarni čvorovi poseduju kopiju primarnog skupa podataka. Da bi replicirao podatke, sekundar primenjuje operacije iz *oplog*-a primarnog čvora na sopstveni skup podataka u asinhronom procesu. Skup replika može imati jednu ili više sekundarnih jedinica.

Sekundarni član se može konfigurisati za određene svrhe, kao što je:

* Sprečavanje da postane primarni na izborima, što mu omogućava da boravi u sekundarnom centru podataka ili da služi kao hladna pripravnost.
* Sprečavanje aplikacija da čitaju sa njega, što mu omogućava da pokreće aplikacije koje zahtevaju odvajanje od normalnog saobraćaja.
* Čuvanje „istorijskog“ zapisa za korišćenje u oporavku od određenih grešaka, kao što su nenamerno izbrisane baze podataka.

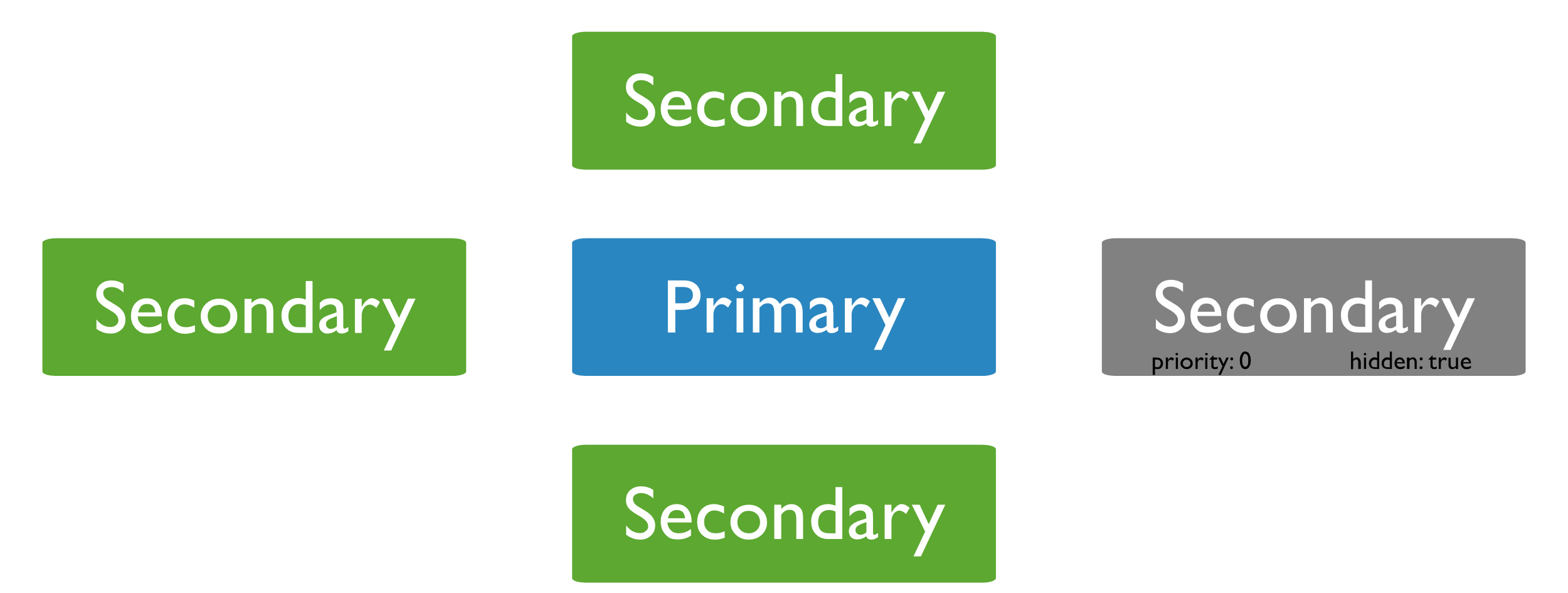
## Članovi skupa replika nultog prioriteta

Član sa nultim prioritetom je član koji ne može postati primarni i ne može pokrenuti izbore. Članovi sa prioritetom 0 mogu da priznaju operacije pisanja koje su izdate. Član sa prioritetom 0 takođe mora biti član sa pravom glasa da bi potvrdio pisanje. Osim već pomenutih ograničenja, sekundari koji imaju prioritet 0 funkcionišu kao normalni sekundari: održavaju kopiju skupa podataka, prihvataju operacije čitanja i glasaju na izborima. Konfigurisanje člana skupa replika sa prioritetom 0 može biti poželjno ako je određeni član raspoređen u centru podataka koji je udaljen od glavne implementacije i stoga ima veće kašnjenje. Možda dobro služi lokalnim zahtevima za čitanje, ali možda nije idealan kandidat za obavljanje dužnosti primara zbog svog kašnjenja. Za ovu situaciju, dijagram sa slike 3 prikazuje centar podataka sa leve strane koji hostuje primar i sekundar, i centar podataka sa desne strane koji hostuje sekundar koji je konfigurisan da ima prioritet 0 kako bi se sprečilo da postane primar. Zbog ove postavke, samo članovi levog centra podataka imaju pravo da postanu primarni na izborima. Uporedite ovo sa podrazumevanim prioritetom za članove skupa replika, prioritet 1, gde bi bilo koja od sekundarnih jedinica u ovom scenariju bila kvalifikovana da služi kao primarna.

*Slika 3 – Primer dva centra podataka*

## Skriveni članovi skupa replika

Skriveni član održava kopiju primarnog skupa podataka, ali je nevidljiv za klijentske aplikacije. Skriveni članovi su dobri za radna opterećenja sa različitijim obrascima korišćenja od ostalih članova u skupu replika. Skriveni članovi uvek moraju imati prioritet 0 i samim tim ne mogu postati primari. Skriveni članovi, međutim, mogu glasati na izborima. U sledećem skupu replika od pet članova prikazanih na slici 4, sva četiri sekundarna člana imaju kopije primarnog skupa podataka, ali jedan od sekundarnih članova je sakriven.



*Slika 4 – Primer skrivenog člana skupa replika*

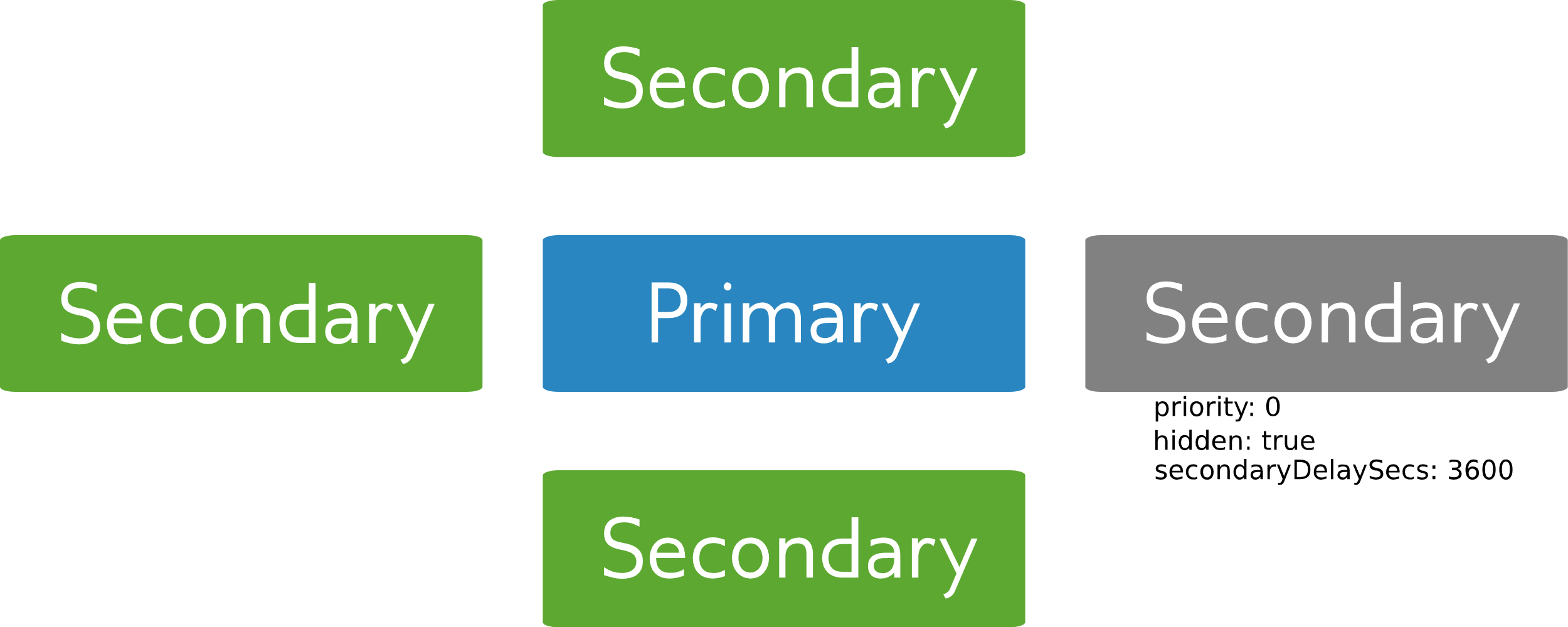
## Odloženi članovi skupa replika

Odloženi članovi sadrže kopije skupa podataka replika. Međutim, skup podataka odloženog člana odražava ranije ili odloženo stanje skupa. Na primer, ako je trenutno vreme 09:52 i član ima kašnjenje od sat vremena, odloženi član nema operaciju noviju od 08:52. Pošto su odloženi članovi „pokretna rezervna kopija“ ili tekući „istorijski“ snimak skupa podataka, oni mogu pomoći u oporavku od različitih vrsta ljudskih grešaka. Na primer, odloženi član može omogućiti oporavak od neuspešne nadogradnje aplikacije i grešaka operatera, uključujući izbrisane baze podataka i kolekcije.

Odloženi članovi:

* Moraju imati prioritet 0 . Postavlja se prioritet na 0 da bi se sprečilo da odloženi član postane primarni.
* Moraju biti skriveni članovi. Uvek treba sprečiti aplikacije da vide i pitaju odložene članove.

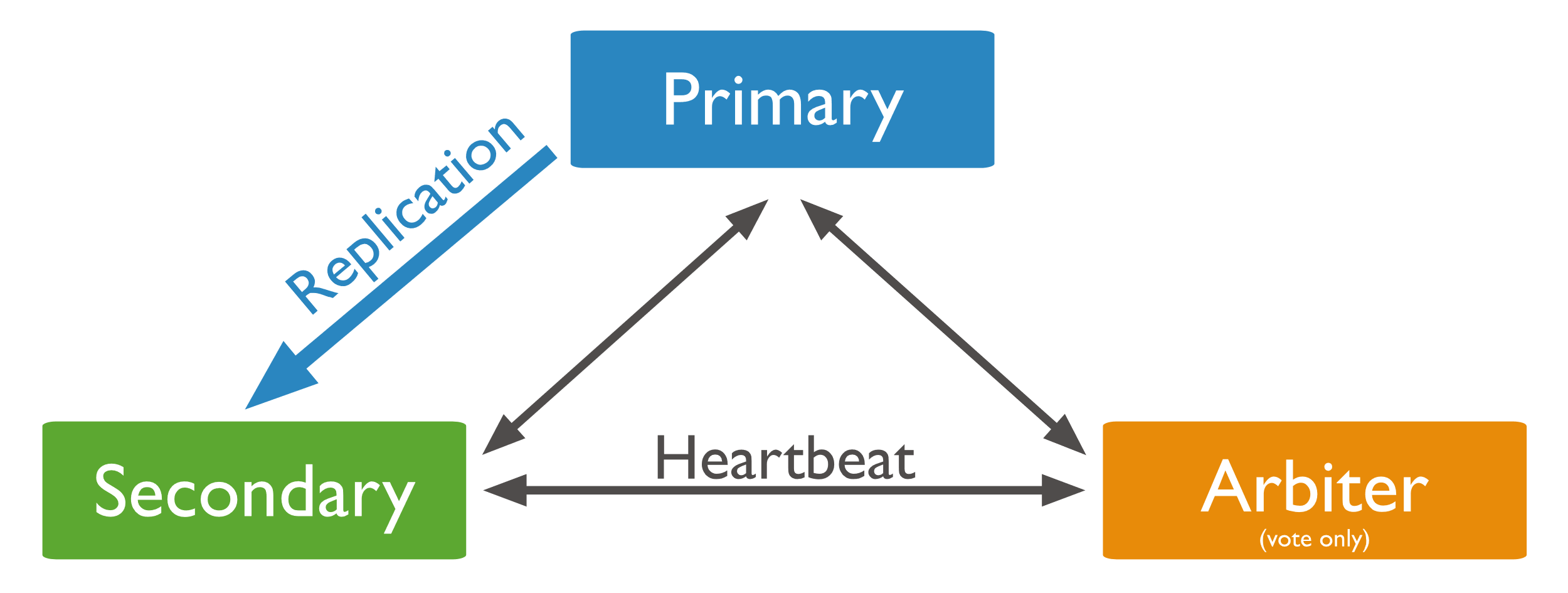
U sledećem skupu replika od 5 članova prikazanih na slici 5, primarni i svi sekundarni članovi imaju kopije skupa podataka. Jedan član primenjuje operacije sa zakašnjenjem od 3600 sekundi (jedan sat). Ovaj odloženi član je takođe sakriven i ima prioritet 0.



*Slika 5 – Primer primenjivanja operacija sa zakašnjenjem*

# **Arbitar skupa replika**

Arbitar učestvuje na izborima za primare, ali arbitar nema kopiju skupa podataka i ne može postati primarni. Arbitar ima tačno 1 izborni glas. Podrazumevano, arbitar ima prioritet 0. Na primeru prikazanom na slici 6, u skupu replika sa 2 člana koji nose podatke (primarni i sekundarni), arbitar dozvoljava skupu da ima neparan broj glasova za prekid nerešenog rezultata.



*Slika 6 – Tročlana arhitektura sa arbitrom*

Ako se koristi tročlana arhitektura primarno-sekundarnog arbitra (PSA), treba uzmiti u obzir sledeće:

* Odluka većine za pisanje može da izazove probleme sa performansama ako je sekundar nedostupan ili kasni.
* Ako se koristi globalno podrazumevano većinsko odlučivanje o upisu i problem pisanja je manji od veličine skupa replika, upiti mogu da vrate zastarele (ne u potpunosti replicirane) podatke.

Neophodno je koristiti maksimalno jednog arbitra da bi se izbegli probleme sa konzistentnošću podataka. Višestruki arbitri sprečavaju pouzdanu upotrebu većine pisanja. Da bi se osiguralo da će se upis izvršiti nakon kvara primarnog čvora, većinsko pisanje zahteva da većina čvorova potvrdi operaciju pisanja. Arbitri ne čuvaju nikakve podatke, ali doprinose broju čvorova u skupu replika. Kada skup replika ima više arbitara, manje je verovatno da će većina čvorova koji nose podatke biti dostupna nakon kvara. Kada rade sa autorizacijom, arbitri razmenjuju akreditive sa drugim članovima skupa radi provere autentičnosti. *MongoDB* šifruje proces autentifikacije i razmena autentifikacije je kriptografski bezbedna. Pošto arbitri ne čuvaju podatke, oni ne poseduju internu tabelu mapiranja korisnika i uloga koja se koristi za autentifikaciju.

Oplog

*Oplog* (dnevnik operacija) je posebna kolekcija koja vodi evidenciju o svim operacijama koje modifikuju podatke uskladištene u vašim bazama podataka. *MongoDB* podržava navođenje minimalnog perioda zadržavanja *oplog*-a u satima, pri čemu *MongoDB* samo uklanja unos *oplog*-a u sledećim slučajevima:

* Oplog je dostigao maksimalnu konfigurisanu veličinu
* Oplog unos je stariji od konfigurisanog broja sati

*MongoDB* primenjuje operacije baze podataka na primarnim čvorom, a zatim beleži operacije na *oplog*-u pomenutog člana. Sekundarni članovi zatim kopiraju i primenjuju ove operacije u asinhronom procesu. Svi članovi skupa replika sadrže kopiju *oplog*-a, u kolekciji *local.oplog.rs*, što im omogućava da održavaju trenutno stanje baze podataka.

Da bi se olakšala replikacija, svi članovi skupa replika šalju takozvane otkucaje srca (pingove) svim ostalim članovima. Svaki sekundarni član može uvesti *oplog* unose od bilo kog drugog člana. Svaka operacija u *oplog*-u je idempotentna. To jest, *oplog* operacije daju iste rezultate bilo da se primenjuju jednom ili više puta na ciljni skup podataka.

# **Veličina oplog-a**

Kada se prvi put pokrene član skupa replika, *MongoDB* kreira *oplog* podrazumevane veličine ako se ne navede veličina *oplog*-a. Počevši od *MongoDB* 4.0, *oplog* može da raste preko svoje konfigurisane granice veličine da bi se izbeglo brisanje upamćenih tačaka. Za *Unix* i *Windows* sisteme podrazumevana veličina *oplog*-a zavisi od mehanizma za skladištenje. Ovi podaci su prikazani u tabeli 1.

Tabela 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Storage Engine | Podrazumevana veličina *oplog*-a | Donja granica | Gornja granica |
| In-Memor**y** Storage Engine | 5% fizičke memorije | 50 MB | 50 GB |
| WiredTiger Storage Engine | 5% slobodnog prostora na disku | 990 MB | 50 GB |

Kod 64-bitnih macOS sistema, podrazumevana veličina *oplog*-a je 192 MB ili fizičke memorije ili slobodnog prostora na disku u zavisnosti od mehanizma za skladištenje, što se može videti u tabeli 2.

Tabela 2:

|  |  |
| --- | --- |
| Storage Engine | Podrazumevana veličina *oplog*-a |
| In-Memor**y** Storage Engine | 192 MB fizičke memorije |
| WiredTiger Storage Engine | 192 MB slobodnog prostora na disku |

U većini slučajeva, podrazumevana veličina *oplog*-a je dovoljna. Na primer, ako *oplog* ima 5% slobodnog prostora na disku i popunjava se za 24 sata rada, onda sekundarni mogu da prestanu da kopiraju unose iz oplog-a do 24 sata, a da ne postanu previše zastareli da bi nastavili sa repliciranjem. Međutim, većina skupova replika ima mnogo manji obim operacija, a njihovi *oplog*-ovi mogu da sadrže mnogo veći broj operacija.

## Minimalni period zadržavanja Oploga

Počevši od MongoDB 4.4, može se odrediti minimalni broj sati za očuvanje unosa u *oplog*-u. *Mongod* uklanja *oplog* unos samo ako:

* Oplog je dostigao maksimalnu konfigurisanu veličinu
* Oplog unos je stariji od konfigurisanog broja sati na osnovu sata sistema hosta

Podrazumevano, *MongoDB* ne postavlja minimalni period zadržavanja *oplog*-a i automatski skraćuje *oplog* počevši od najstarijih unosa da bi održao konfigurisanu maksimalnu veličinu *oplog*-a. Da bi se konfigurisao minimalni period zadržavanja *oplog*-a na pokrenutom *mongod*-u, koristi se *replSetResizeOplog*. Postavljanje minimalnog perioda zadržavanja *oplog*-a dok *mongod* radi zamenjuje sve vrednosti postavljene pri pokretanju. Mora se ažurirati vrednost odgovarajuće postavke konfiguracione datoteke ili opcije komandne linije da bi se zadržale te promene kroz ponovno pokretanje servera.

## Radna opterećenja koja mogu zahtevati veću veličinu oplog-a

Ako se može predvideti da radno opterećenje skupa replika liči na jedan od sledećih obrazaca, možda se može kreirati *oplog* koji je veći od podrazumevanog. Nasuprot tome, ako aplikacija pretežno obavlja čitanje sa minimalnom količinom operacija pisanja, manji *oplog* može biti dovoljan.

Sledeći radni opterećenja mogu zahtevati veću veličinu *oplog*-a:

* Ažuriranja za više dokumenata odjednom – Oplog mora da prevede višestruka ažuriranja u pojedinačne operacije da bi održao idempotenciju. Ovo može koristiti veliku količinu *oplog* prostora bez odgovarajućeg povećanja veličine podataka ili upotrebe diska.
* Brisanja su jednaka istoj količini podataka kao i upisi – Ako se izbrišu otprilike ista količina podataka koja je upisana, baza podataka neće značajno rasti u upotrebi diska, ali veličina dnevnika operacija može biti prilično velika.
* Značajan broj *in-place* ažuriranja– Ako značajan deo posla čine ažuriranja koja ne povećavaju veličinu dokumenata, baza podataka beleži veliki broj operacija, ali ne menja količinu podataka na disku.

## Status oplog-a

Da bi se video status *oplog*-a, uključujući veličinu i vremenski opseg operacija, potrebno je izvršiti metodu *rs.printReplicationInfo()*.

## Kašnjenje replikacije i kontrola toka

U različitim izuzetnim situacijama, ažuriranja *oplog*-a sekundara mogu zaostajati za željenim vremenskim performansama. Korišćemjem db.getReplicationInfo() iz sekundarnog člana replikacije se može proceniti trenutno stanje replikacije i utvrditi da li postoji nenamerno kašnjenje replikacije. Podrazumevano, kontrola protoka je omogućena.

## Spora Oplog aplikacija

Počevši od verzije 4.2 (takođe dostupno počevši od verzije 4.0.6), sekundarni članovi skupa replika evidentiraju unose *oplog*-a za koje je potrebno više vremena od praga sporog rada da se primene. Ove poruke se evidentiraju za sekundare pod komponentom REPL sa primenjenim tekstom op: <oplog unos> trajao je <broj> ms.

## Ponašanje oplog kolekcija

Ne može se izbrisati kolekcija *local.oplog.rs* iz bilo kog člana skupa replika ako *MongoDB* implementacija koristi *WiredTiger* *Storage Engine*. Počevši od v4.2, ne može se izbrisati kolekcija *local.oplog.rs* sa samostalne *MongoDB* instance i preporučuje se da se ne briše kolekcija sa samostalne *MongoDB* v4.0 instance. *Mongod* zahteva oplog i za replikaciju i za oporavak čvora ako čvor padne.

Počevši od *MongoDB* 5.0, više nije moguće izvoditi ručne operacije pisanja u *oplog* na klasteru koji radi kao skup replika. Izvođenje operacija pisanja u oplog kada se pokreće kao samostalna instanca treba da se vrši samo uz uputstva *MongoDB* podrške.

Sinhronizacija podataka skupa replika

Da bi kopije deljenog skupa podataka bile ažurne, sekundarni članovi skupa replika sinhronizuju ili repliciraju podatke drugih članova. *MongoDB* koristi dva oblika sinhronizacije podataka: početnu sinhronizaciju za popunjavanje novih članova punim skupom podataka i replikaciju za primenu tekućih promena na ceo skup podataka.

Inicijalna ili početna sinhronizacija kopira sve podatke sa jednog člana skupa replika u drugog člana. Počevši od *MongoDB* 4.4, može se navesti željeni izvor inicijalne sinhronizacije koristeći parametar *initialSincSourceReadPreference*. Ovaj parametar se može specificirati samo kada se pokreće *mongod*.

Kada izvršite početnu sinhronizaciju, MongoDB klonira sve baze podataka osim lokalne baze podataka. Da bi klonirao, *mongod* skenira svaku kolekciju u svakoj izvornoj bazi podataka i ubacuje sve podatke u sopstvene kopije ovih kolekcija. Inicijalna sinhronizacija gradi sve indekse kolekcije dok se dokumenti kopiraju za svaku kolekciju. U ranijim verzijama *MongoDB*-a, samo \_id indeksi se prave tokom ove faze. Početna sinhronizacija povlači novododate *oplog* zapise tokom kopiranja podataka. Potrebno je obezbediti da ciljni član ima dovoljno prostora na disku u lokalnoj bazi podataka da privremeno uskladišti ove *oplog* zapise tokom trajanja ove faze kopiranja podataka. Koristeći *oplog* iz izvora, *mongod* ažurira svoj skup podataka kako bi odražavao trenutno stanje skupa replika.

# **Tolerancija grešaka**

Ako sekundar koji obavlja početnu sinhronizaciju naiđe na neprolaznu (tj. trajnu) grešku u mreži tokom procesa sinhronizacije, sekundar ponovo pokreće početni proces sinhronizacije od početka. Počevši od *MongoDB* 4.4, početna sinhronizacija sekundara može pokušati da nastavi proces sinhronizacije ako je prekinuta prolaznom (tj. privremenom) greškom na mreži, ispuštanjem kolekcije ili preimenovanjem kolekcije. Izvor sinhronizacije takođe mora da pokreće *MongoDB* 4.4 da bi podržao početnu sinhronizaciju koja se može nastaviti. Ako izvor sinhronizacije pokreće *MongoDB* 4.2 ili stariju verziju, sekundar mora ponovo pokrenuti početni proces sinhronizacije kao da je naišao na neprolaznu mrežnu grešku. Podrazumevano, sekundar pokušava da nastavi početnu sinhronizaciju tokom 24 sata. *MongoDB* 4.4 dodaje parametar servera *initialSincTransientErrorRetriPeriodSeconds* za kontrolu količine vremena tokom kojeg sekundar pokušava da nastavi početnu sinhronizaciju. Ako sekundar ne može uspešno da nastavi početni proces sinhronizacije tokom konfigurisanog vremenskog perioda, on bira novi zdrav izvor iz skupa replika i ponovo pokreće početni proces sinhronizacije od početka. Sekundar pokušava da ponovo pokrene početnu sinhronizaciju do 10 puta pre nego što vrati fatalnu grešku.

# **Početni izbor izvora sinhronizacije**

Početni izbor izvora sinhronizacije zavisi od vrednosti parametra pokretanja *mongod*-a *initialSincSourceReadPreference*. Za *initialSincSourceReadPreference* postavljen na *primary* (podrazumevano ako je ulančavanje onemogućeno), potrebno je izabrati primar kao izvor sinhronizacije. Za *initialSincSourceReadPreference* postavljen na *primaryPreferred* (podrazumevano za glasanje članova skupa replika), treba pokušati sa izborom primarnog čvora kao izvora sinhronizacije. Ako je primar nedostupan, izvršava se izbor izvora sinhronizacije od preostalih članova skupa replika. Za sve ostale podržane režime čitanja, treba izvršiti izbor izvora sinhronizacije iz članova skupa replika. Članovi koji vrše početni izbor izvora sinhronizacije vrše dva prolaza kroz listu svih članova skupa replika.

Član primenjuje sledeće kriterijume na svakog člana skupa replika prilikom prvog prolaza za izbor početnog izvora sinhronizacije:

* Izvor sinhronizacije mora da bude u primarnom ili sekundarnom stanju replikacije.
* Izvor sinhronizacije mora biti onlajn i dostupan.
* Ako je *initialSincSourceReadPreference* *secondary* ili *secondaryPreferred*, izvor sinhronizacije mora biti sekundarni.
* Izvor sinhronizacije mora biti vidljiv.
* Izvor sinhronizacije mora biti unutar 30 sekundi od najnovijeg unosa u *oplog* na primaru.
* Ako član gradi indekse, izvor sinhronizacije mora da pravi indekse.
* Ako član glasa na izborima za repliku, izvor sinhronizacije takođe mora da glasa.
* Ako član nije odloženi član, izvor sinhronizacije ne sme biti odložen.
* Ako je član odloženi član, izvor sinhronizacije mora imati kraće konfigurisano odlaganje.
* Izvor sinhronizacije mora da bude brži (tj. manje kašnjenje) od trenutno najboljeg izvora sinhronizacije.
* Ako nema izvora sinhronizacije kandidata nakon prvog prolaza, član obavlja drugi prolaz sa blažim kriterijumima.

Član primenjuje sledeće kriterijume na svaki član skupa replika prilikom drugog prolaza za izbor početnog izvora sinhronizacije:

* Izvor sinhronizacije mora da bude u primarnom ili sekundarnom stanju replikacije.
* Izvor sinhronizacije mora biti onlajn i dostupan.
* Ako je *initialSincSourceReadPreference* sekundaran, izvor sinhronizacije mora biti sekundarni.
* Ako član gradi indekse, izvor sinhronizacije mora da pravi indekse.
* Izvor sinhronizacije mora da bude brži (tj. manje kašnjenje) od trenutno najboljeg izvora sinhronizacije.

Ako član ne može da izabere početni izvor sinhronizacije nakon dva prolaza, evidentira grešku i čeka 1 sekundu pre nego što ponovo pokrene proces selekcije. Sekundarni *mongod* može ponovo pokrenuti početni proces izbora izvora sinhronizacije do 10 puta pre nego što izađe sa greškom.

## Arhitekture postavljanja skupa replika

Arhitektura skupa replika utiče na kapacitet i sposobnost skupa. Standardno postavljanje skupa replika za proizvodni sistem je tročlani skup replika. Ovakva arhitektura obezbeđuju redundantnost i toleranciju na greške.

## Strategije prilikom postavljanja arhitekture

Skup replika može imati do 50 članova, ali samo 7 članova sa pravom glasa. Ako skup replika već ima 7 članova sa pravom glasa, dodatni članovi moraju biti članovi bez prava glasa.

Poželjno je da skup replika ima neparan broj članova sa pravom glasa. Kao što je napomenuto u prethodnom odeljku, skup replika može imati do 7 članova sa pravom glasa. Ako postoji paran broj članova sa pravom glasa, potrebno je priključiti dodatnog člana sa pravom glasa ili, ako ograničenja zabranjuju dodatnog člana sa pravom glasa, arbitra. Arbitar ne čuva kopiju podataka i zahteva manje resursa. Kao rezultat, arbitar se može pokrenuti na serveru aplikacija ili drugom deljenom resursu.

Tolerancija na greške za skup replika je broj članova koji mogu postati nedostupni i koji i dalje ostavljaju dovoljno članova u skupu da izaberu primar. Drugim rečima, to je razlika između broja članova u skupu i većine članova sa pravom glasa koja je potrebna da se izabere primar. Bez primara, skup replika ne može da prihvati operacije pisanja. Tolerancija na greške je efekat veličine skupa replika, ali veza nije direktna.

Tabela 3:

|  |  |  |
| --- | --- | --- |
| Broj članova | Potrebna većina da bi se izabrala nova primarna | Tolerancija na greške |
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 4 | 2 |

U tabeli 3 se vidi da dodavanje člana skupu replika ne povećava uvek toleranciju na greške. Međutim, u ovim slučajevima, dodatni članovi mogu pružiti podršku za namenske funkcije, kao što su rezervne kopije ili izveštavanje.

## Balans opterećenja na implementacijama sa velikim opterećenjem

U primeni sa veoma velikim saobraćajem čitanja, može se poboljšati propusnost čitanja distribucijom čitanja sekundarnim članovima. Kako primena raste, treba dodavati ili premeštati članove u alternativne centre podataka da bi se poboljšala redundantnost i dostupnost.

Postojeći članovi skupa replika moraju imati rezervni kapacitet da podrže dodavanje novog člana. Nove članovi se dodaju pre nego što trenutna potražnja zasiti kapacitet skupa.

## Imenovanje skupa replika

Ako se aplikacija povezuje na više od jednog skupa replika, svaki skup mora imati različito ime. Neki drajveri grupišu veze skupa replika prema imenu skupa replika.

# **Implementacioni obrasci**

Postoje dva osnovna tipa arhitektura:

* Setovi replika od tri člana – Tročlani skupovi replika obezbeđuju minimalnu preporučenu arhitekturu za skup replika.
* Skupovi replika raspoređeni u dva ili više centara podataka – Geografski raspoređeni skupovi uključuju članove na više lokacija radi zaštite od kvarova specifičnih za objekat, kao što su nestanci struje.

Ako je potrebno, treba kombinovati karakteristike svake arhitekture u sopstvenoj primeni.

## Tročlana arhitektura skupa replika

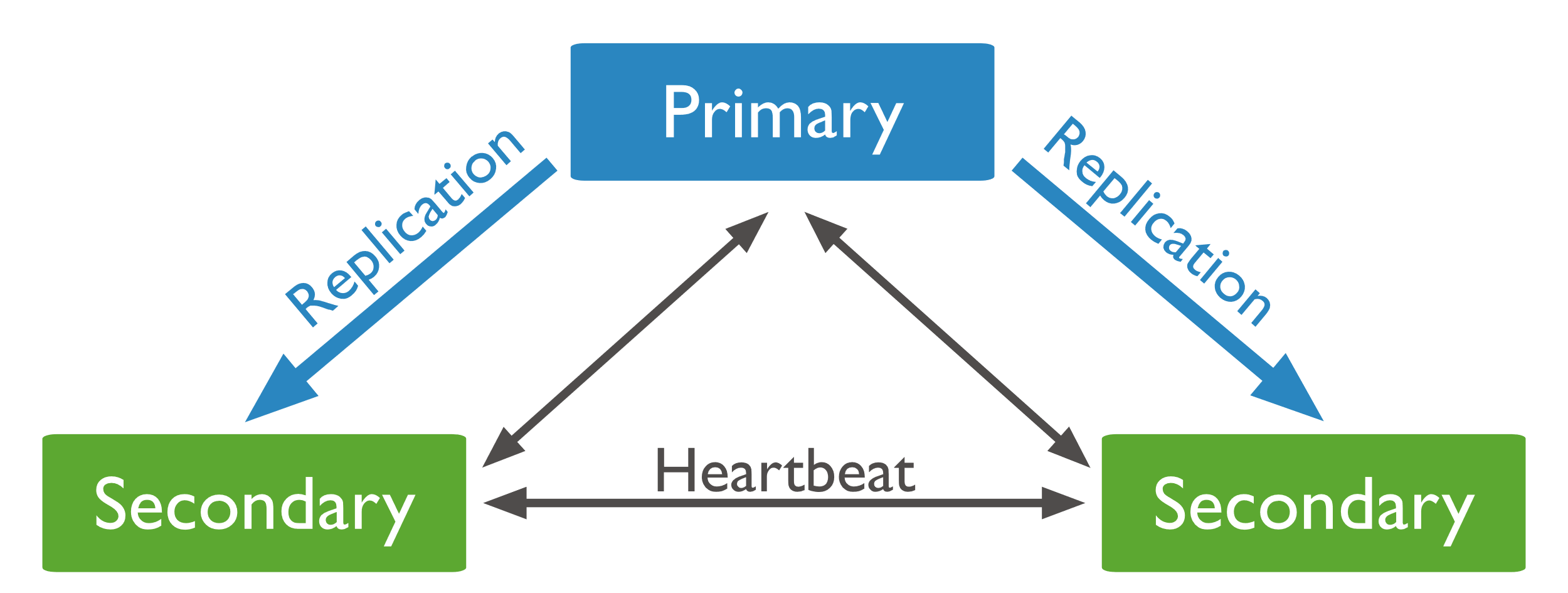
Minimalni broj članova skupa replika potrebnih za dobijanje pogodnosti skupa replika je tri člana. Tročlani skup replika može imati ili tri člana koji nose podatke (primarni-sekundarni-sekundarni) (preporučeno) ili ako okolnosti (kao što je trošak) zabranjuju dodavanje trećeg člana koji nosi podatke, dva člana koja nose podatke i arbitra (primarni- sekundarni-arbitar).

Primarni sa dva sekundarna člana (P-S-S)

Skup replika sa tri člana koji čuva podatke ima:

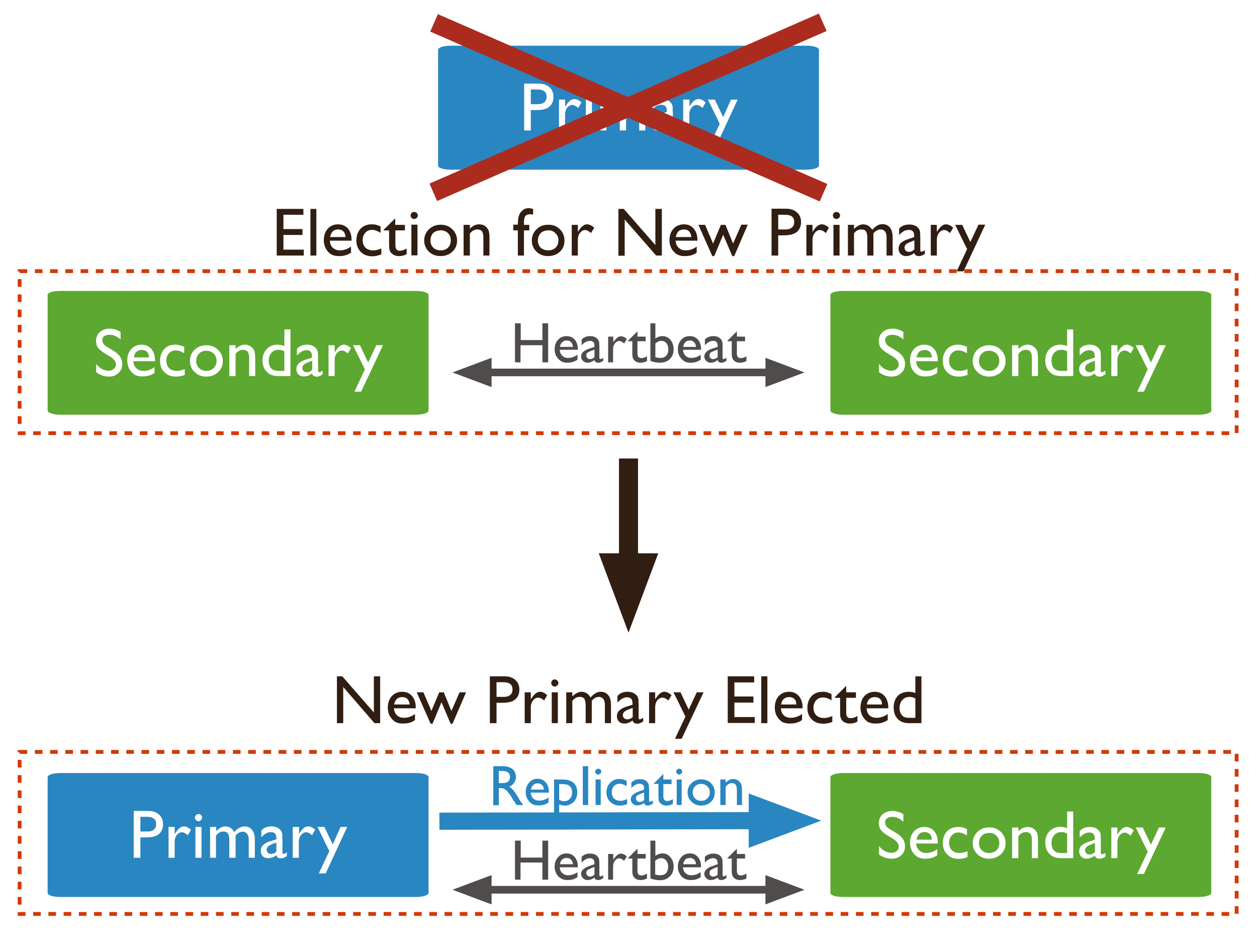
* Jedan primarni član.
* Dva sekundarna člana. Oba sekundarna mogu postati primarni na izborima.

Dijagram skupa replika od 3 člana koji se sastoji od primarnog i dva sekundara prikazan je na slici 7.



*Slika 7 – P-S-S arhitektura*

Ove primene obezbeđuju dve potpune kopije skupa podataka u svakom trenutku pored primarnog. Ovi skupovi replika pružaju dodatnu toleranciju na greške i visoku dostupnost. Ako primarni član nije dostupan, skup replika bira sekundar da bude primarni i nastavlja normalan rad. Stari primarni član se ponovo pridružuje skupu kada je dostupan. Ceo ovaj proces prikazan je na slici 8.



*Slika 8 – Primer izbora kod tročlane arhitekture*

Tročlani skup replika sa dva člana koji čuva podatke ima:

* Jedan primarni član.
* Jedan sekundarni član.
* Jednog arbitra.

Pošto arbitar ne poseduje kopiju podataka, ove primene pružaju samo jednu potpunu kopiju podataka. Arbitri zahtevaju manje resursa, ali na račun više ograničene redundancije i tolerancije grešaka. Međutim, primena sa primarnim, sekundarnim članovima i arbitrom obezbeđuje da skup replika ostane dostupan ako primar ili sekundar nisu dostupni. Ako primarni nije dostupan, skup replika će izabrati sekundar da bude primarni.

## Skupovi replika distribuirani u dva ili više centara podataka

Dok skupovi replika obezbeđuju osnovnu zaštitu od otkazivanja jedne instance, skupovi replika čiji se svi članovi nalaze u jednom centru podataka podložni su greškama centra podataka. Prekidi struje, prekidi u mreži i prirodne katastrofe su svi problemi koji mogu uticati na skupove replika čiji se članovi nalaze u jednom objektu. Distribucija članova skupa replika u geografski različite centre podataka dodaje redundantnost i obezbeđuje toleranciju grešaka ako je jedan od centara podataka nedostupan.

Da bi podaci bili zaštićeni u slučaju kvara centra podataka, potrebno je da postoji bar jednog član u alternativnom centru podataka. Ako je moguće, dobra praksa je koristiti neparan broj centara podataka i distribucija članova koja maksimizira verovatnoću da čak i uz gubitak centra podataka, preostali članovi skupa replika mogu da čine većinu ili, u najmanju ruku, da obezbede kopiju podataka .

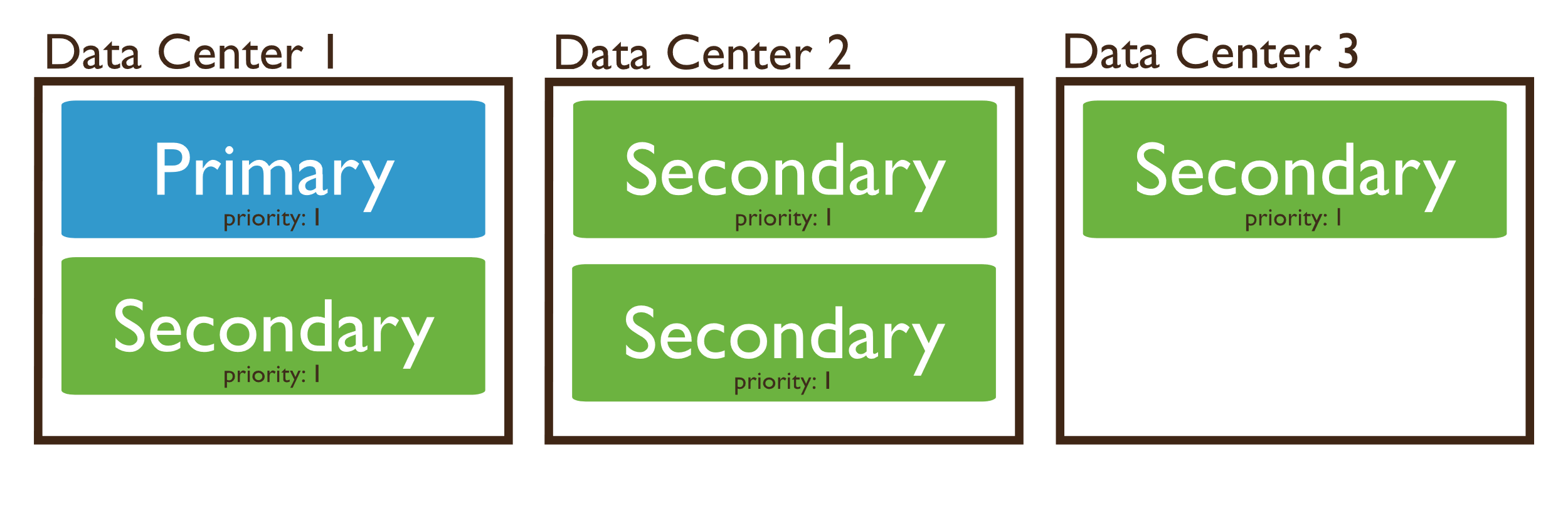
Na primer, za skup replika od tri člana, neke moguće distribucije članova uključuju:

* Dva centra podataka: dva člana u centru 1 i jedan član u centru 2. Ako je jedan od članova skupa replika arbitar, treba distribuirati arbitra u centru 1 sa članom koji nosi podatke. Ako centar 1 padne, skup replika postaje samo za čitanje. Ako centar 2 padne, skup replika ostaje za pisanje jer članovi u centru 1 mogu održati izbore.
* Tri data centra: jedan član u centru 1, jedan član u centru 2 i jedan član u centru 3. Ako se neki centar podataka pokvari, skup replika ostaje upisan jer preostali članovi mogu održati izbore.

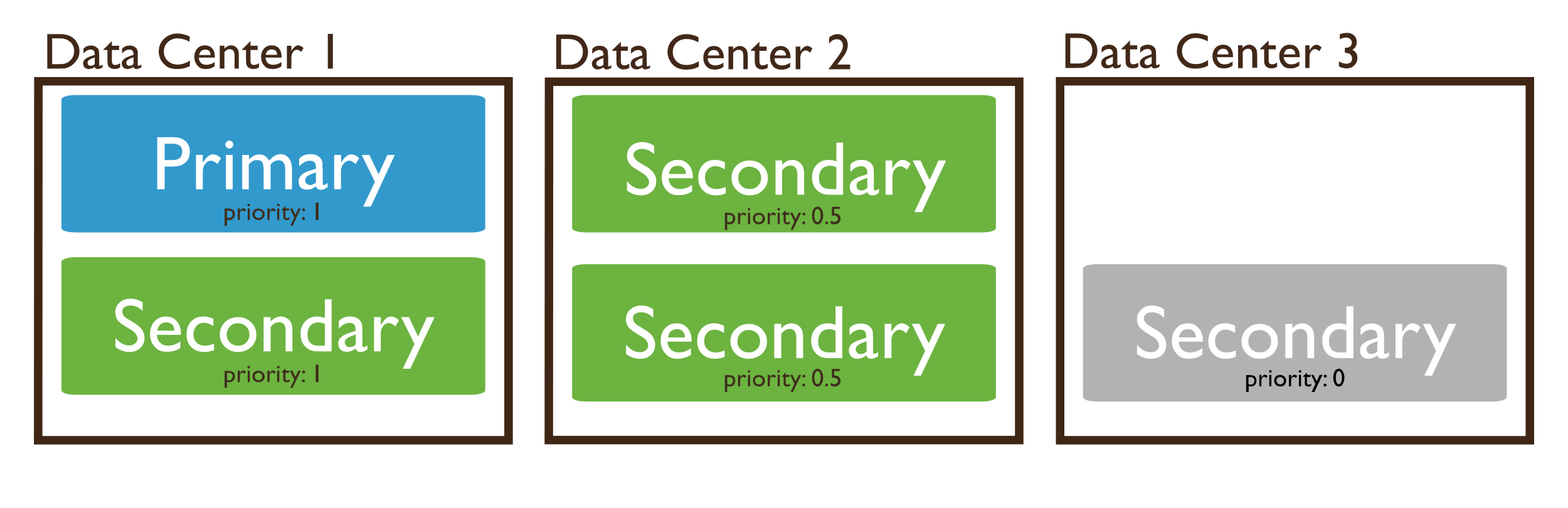
Za skup replika sa 5 članova, neke moguće distribucije članova uključuju:

* Dva centra podataka: tri člana u centru 1 i dva člana u centru 2. Ako centar 1 padne, skup replika postaje samo za čitanje. Ako centar 2 padne, skup replika ostaje za pisanje jer članovi u centru 1 mogu stvoriti većinu.
* Tri centra podataka: dva člana u centru1, dva člana u centru 2 i jedan član u centru 3. Ako se neki centar podataka pokvari, skup replika ostaje za pisanje jer preostali članovi mogu održati izbore.

Na primer, sledeći skup replika od 5 članova koji je prikazan na slici 9, distribuira svoje članove u tri centra podataka.

*Slika 9 – Primer skupa replika distribuiranog u tri centra podataka*

Neki članovi skupa replika, kao što su članovi koji imaju ograničenje mreže ili ograničene resurse, ne bi trebalo da budu u mogućnosti da postanu primari u slučaju greške. Treba konfigurisati članove koji ne bi trebalo da postanu primari da imaju prioritet 0. U sledećem primeru prikazanom na slici 10, članovi skupa replika u centru podataka 1 imaju veći prioritet od članova u centru podataka 2 i 3; članovi u centru 2 imaju veći prioritet od člana u centru 3.



*Slika 10 – Primer skupa replika različitog prioriteta distribuiranog u tri centra podataka*

Na dijagramu na slici 10 prikazan je skup replika od 5 članova raspoređenih u tri data centra. Skup replika uključuje članove sa prioritetom 0,5 i prioritetom 0.

# **Semantika čitanja i pisanja skupa replika**

Iz perspektive klijentske aplikacije, transparentno je da li *MongoDB* instanca radi kao jedan server (tj. „samostalan“) ili skup replika. Međutim, *MongoDB* obezbeđuje dodatne konfiguracije za čitanje i pisanje za skupove replika. Briga o pisanju za skupove replika opisuju broj članova koji nose podatke (tj. primarnih i sekundarnih, ali ne arbitra) koji moraju da potvrde operaciju pisanja pre nego što se operacija vrati kao uspešna. Član može da potvrdi operaciju pisanja samo nakon što je primio i uspešno primenio upis.

Za skupove replika, upis zahteva potvrdu da su se operacije pisanja proširile na izračunatu većinu članova sa pravom glasa koji nose podatke. Za većinu konfiguracija skupa replika, ovo je podrazumevana način za upis. Aplikacija koja izdaje operaciju pisanja koja zahteva potvrdu o pisanju čeka dok primar ne primi potvrdu od potrebnog broja članova. Što je više članova koji priznaju upis, manja je verovatnoća da će se pisani podaci vratiti nazad ako primar ne uspe. Međutim, ako veliki broj članova skupa replika treba da potvrdi pisanje, to može povećati kašnjenje jer klijent mora da sačeka dok ne dobije traženi nivo potvrde za pisanje. Izbor idealnog stepena potvrde za pisanje za bilo koju datu operaciju pisanja zavisi od ciljeva performansi aplikacije.

Sledeća operacija uključuje opciju *writeConcern* za metod *insertOne()*. Operacija specificira stepen potvrde za pisanje i vremensko ograničenje od 5 sekundi. Parametar za upisivanje *wtimeout* obezbeđuje da se operacija ne blokira na neodređeno vreme.

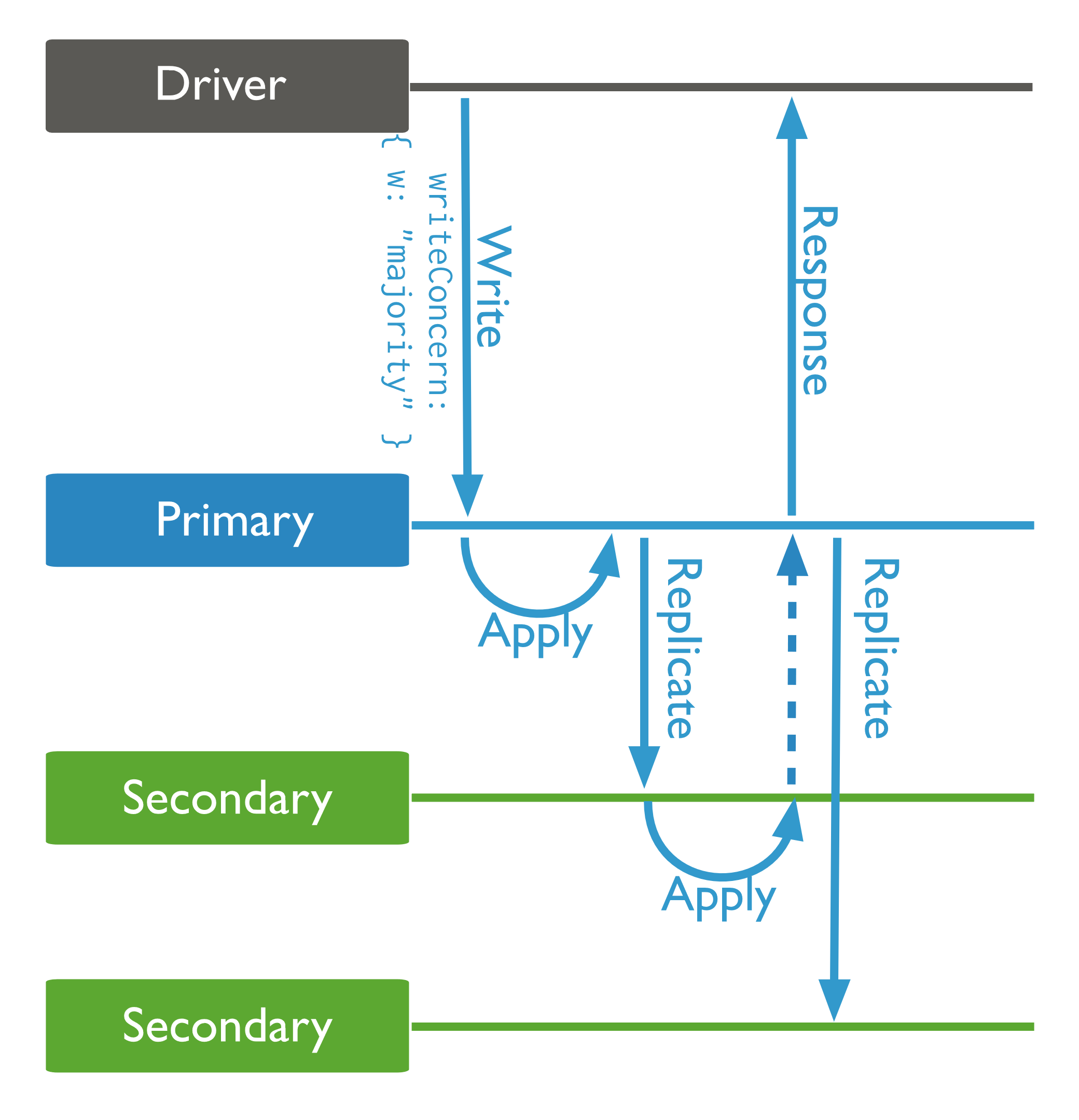
db.products.insertOne(

{ item: "Test", qty : 100, type: "Test" },

{ writeConcern: { w: "majority" , wtimeout: 5000 } }

)

Aplikacija čeka dok primar ne vrati potvrdu upisa, što ukazuje na to da je izračunata većina članova sa glasanjem koji nose podatke potvrdila operaciju pisanja. Na primer, u skupu replika od 3 člana (P-S-S), operacija bi zahtevala potvrdu 2 od 3 člana. Ako je skup replika kasnije skaliran tako da uključuje dva dodatna sekundarna člana sa pravom glasa, ista operacija bi zahtevala potvrdu 3 od 5 članova skupa replika. Ako primarni član ne vrati potvrdu o upisu unutar ograničenja *wtimeout*-a, operacija pisanja ne uspeva i kao rezultat generiše se greška.



*Slika 11 – Primer procesa upisa*

Operacija pisanja kojoj istekne vreme čekanja za navedenu granicu upisivanja samo ukazuje na to da potreban broj članova skupa replika nije potvrdio operaciju pisanja u vremenskom periodu *wtimeout*. To ne znači nužno da primarni član nije uspeo da primeni upis. Podaci mogu postojati na podskupu čvorova skupa replika u vreme greške pri pisanju i mogu nastaviti sa repliciranjem sve dok svi čvorovi u klasteru ne budu imali te podatke. Prijave treba da uzmu u obzir potencijalnu dostupnost upisa podataka bez obzira na stanje potvrde upisa.

Primer kreiranja tročlanog skupa replika

Nakon instalacije *Docker desktop*-a, pribavljena je najnovija verzija *mongo*-a, izvršavanjem sledeće komande u terminalu:

*docker pull mongo*

Potom su kreirana tri kontejnera sa *Mongo* slikama, koji su svi unutar sopstvene *Docker* mreže kontejnera. Nazvani su: mongo1, mongo2 i mongo3. Ovo su tri *Mongo* instance skupa replika. Takođe svaki od njih je podignut na lokalnoj mašini. Svaki od tri Mongo kontejnera treba da bude u stanju da komunicira sa svim ostalim kontejnerima u mreži.

Diagram

Description automatically generated

*Slika 12 – Arhitektura kreiranog mongo klastera*

Komandom *docker network create my-mongo-cluster* kreirana je nova mreža za klaster. Sa naredne 3 komande kreirane su instance *mongo* replika.

*docker run -p 30001:27017 --name mongo1 --net my-mongo-cluster   
mongo mongod --replSet my-mongo-set*

*docker run -p 30002:27017 --name mongo2 --net my-mongo-cluster   
mongo mongod --replSet my-mongo-set*

*docker run -p 30003:27017 --name mongo3 --net my-mongo-cluster   
mongo mongod --replSet my-mongo-set*

Gde su:

* *docker run –* komanda za startovanje kontejnera *docker* slike
* *-p 3000X:27017 –* port 27017 u postojećem kontejneru, kao port 30001 na lokalnom hostu
* *--name mongox –* ime kontejnera
* --*net my-mongo-cluster –* mreža na koju se dodaje novokreirani kontejner
* *mongo -* naziv slike koja se koristi za stvaranje novog kontejnera
* *mongod --replSet my-mongo-set* - pokretanje *mongod-*a dok se ista ta instanca dodaje u skup replika pod nazivom „my-mongo-set“

Kreirane instance se mogu videti na slici 13.

*A screenshot of a computer

Description automatically generated with medium confidence*

*Slika 13 – Tri instace mongo-a*

Nakon kreiranja i pokretanja Mongo instanci, komandom *docker exec -it mongo1 mongo* vršse povezivanje sa mongo terminalom. Kreirana je sledeća konfiguracija:

*> db = (new Mongo('localhost:27017')).getDB('test')*

*test*

*> config = {*

*"\_id" : "my-mongo-set",*

*"members" : [*

*{*

*"\_id" : 0,*

*"host" : "mongo1:27017"*

*},*

*{*

*"\_id" : 1,*

*"host" : "mongo2:27017"*

*},*

*{*

*"\_id" : 2,*

*"host" : "mongo3:27017"*

*}*

*]*

*}*

Prvi ključ \_id u konfiguraciji treba da bude isti kao --replSet fleg koja je postavljena za mongod instance, što je u ovom slučaju my-mongo-set. Zatim se navode svi članovi koji pripadaju skupu replika. Skup replika se pokreće komandom *rs.initiate(config)*. Zatim ubacivanjem test podataka u primar, taj isti podatak se može pročitati u ostalim čvorovima, što se može i videti iz narednog skupa komandi.

> db.mycollection.insert({name : 'sample'})

WriteResult({ "nInserted" : 1 })

> db.mycollection.find()

{ "\_id" : ObjectId("111fegszhr527fhd56436"), "name" : "sample" }

> db2 = (new Mongo('mongo2:27017')).getDB('test')

test

> db2.setSecondaryOk()

> db2.mycollection.find()

{ "\_id" : ObjectId("111fegszhr527fhd56436"), "name" : "sample" }

Zaključak

Kada se čuvaju podaci u *MongoDB*-u, može biti više *Mongo* servera (zvanih replike) koji drže kopiju uskladištenih podataka. Na ovaj način, čak i ako se jedan od servera pokvari, imamo rezervnu kopiju za preuzimanje podataka. Ova tehnika se koristi da bi aplikacije bile otpornije na kvarove baze podataka, pošto aplikacija može nastaviti da radi čak i ako jedan server baze podataka nije u funkciji.

Reference

[1] https://www.mongodb.com/docs/manual/core/replica-set-members/

[2] https://www.mongodb.com/docs/manual/core/replica-set-secondary/

[3] https://www.mongodb.com/docs/manual/core/read-preference/

[4] https://www.mongodb.com/docs/manual/core/replica-set-priority-0-member/

[5] https://www.mongodb.com/docs/manual/core/replica-set-hidden-member/

[6] https://www.mongodb.com/docs/manual/core/replica-set-delayed-member/

[7] https://www.mongodb.com/docs/manual/core/replica-set-arbiter/

[8] https://www.mongodb.com/docs/manual/core/replica-set-oplog/

[9] https://www.mongodb.com/docs/manual/core/replica-set-sync/

[10] https://www.mongodb.com/docs/manual/core/replica-set-architectures/

[11] https://www.mongodb.com/docs/manual/core/replica-set-high-availability/

[12] https://www.mongodb.com/docs/manual/applications/replication/

[13] https://www.bmc.com/blogs/mongodb-replication/

[14] https://www.tutorialspoint.com/mongodb/mongodb\_replication.htm

[15] https://hevodata.com/learn/mongodb-replica-set-3-easy-methods/

[16] https://www.sohamkamani.com/docker/mongo-replica-set/