

Elektronski fakultet, Univerzitet u Nišu
Katedra za Računarstvo i informatiku

SEMINARSKI RAD

Replikacija baza podataka na primeru Redis NoSQL baze

Predmet: Sistemi za upravljanje bazama podataka

Student: Nikola Stanisavljević
Broj indeksa: 940

Sadržaj

Uvod	2
Replikacija podataka	3
Redis	7
Redis - In-Memory DB	7
Replikacija podataka u Redis-u	8
Redis Sentinel	11
Praktična ilustracija replikacije	14
Zaključak	21
Literatura	22

Uvod

Težnja ovog rada je da prikaže replikaciju baza podataka za Redis NoSQL rešenje. U početnom delu rada objašnjeni su način i važnost replikacija kod baza podataka generalno, kao i osnovne podele tipova replikacije. Dalje se obrađuje **Redis** kao *In-Memory* NoSQL predstavnik Key/Value baza, predstavljeni su osnovni tipovi i način funkcionisanja ove baze.

Značaj **replikacije baza podataka** ogleda se u poboljšanju dostupnosti tih podataka i otpornosti i održivosti sistema. Čuvanje istih podataka na više različitih lokacija je značajno pri oporavku baze podataka, kako bi i nakon pada nekog sistema podaci bili očuvani i dostupni za upotrebu u realnom vremenu. Osim toga, postojanje replika baze omogućava brži pristup podacima i smanjenje latentnosti.

Iako Redis pokreće jedna nit (engl. *single threaded*) on može procesirati par hiljadana zahteva u sekundi, sa veoma brzim odgovorom. Redis se može upotrebiti na tri različita načina: kao *cache* radi brže performanse upita, kao skladište podataka (engl. *data store*) i kao *message broker* koji je zadužen za distribuciju podataka kroz master/slave (*publish/subscribe*) sistem. Više detalja o pomenutim implementacijama nalazi se u posebnom poglavlju posvećenom Redis bazi.

Replikacija podataka

Replikacija podataka (engl. *data replication*) predstavlja proces smeštanja podataka na više različitih lokacija odnosno servera (čvorova) u svrhu povećanja dostupnosti podataka (engl. *availability*). Ovaj proces podrazumeva kopiranje podataka sa jednog servera na drugi, tako da podaci budu konzistentni i dostupni pri pristupanju bilo kom serveru u datom trenutku, a rezultat je distribuirana baza. Značaj replikacije ogleda se u povećanju dostupnosti podataka i bržem izvršenju upita. Odnosno osigurano je da u slučaju otkazivanja nekog servera imamo sačuvane podatke na nekoj drugoj lokaciji i upiti mogu brže doći do traženih podataka korišćenjem lokalnih kopija umesto pristupa *remote* serveru.

Replikacija podataka pruža različite pogodnosti u vidu:

- Konzistentne kopije podataka na različitim serverima;
- Povećanja dostupnosti podataka;
- Povećanja pouzdanosti podataka;
- Visoke performanse i podrška za veći broj korisnika;
- Uklanjanja redundantnosti, odnosno suvišnih podataka;
- Smanjenje kretanja podataka, budući da ima više replika podaci će se uglavnom naći tamo gde se obrađuje transakcija;
- Bržeg izvršavanja upita.

Dok su neke od mana replikacije:

- Potreba za više prostora za skladištenje istih podataka;
- Postaje skupa kada je potrebno ažurirati replike na svim serverima;
- Održavanje konzistentnosti na svim serverima je kompleksno.

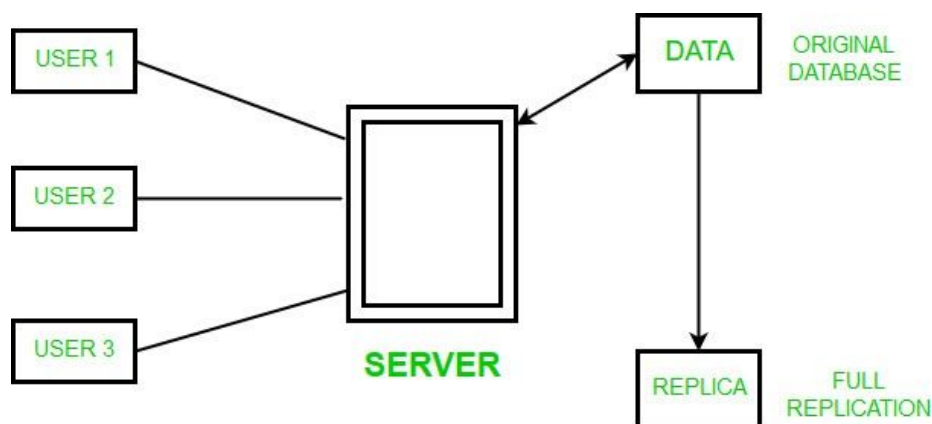
U osnovi razlikujemo **potpunu i parcijalnu replikaciju**. Potpuna replikacija (engl. *full replication*) podrazumeva da se na svakom serveru nalazi kopija celokupne baze, dok je parcijalna replikacija (engl. *partial replication*) znači da se čuvaju samo kopije određenih delova baze kojima se, na primer, najčešće pristupa.

Potpuna replikacija sa sobom donosi prednosti u vidu sledećih osobina:

- Visok stepen dostupnosti podataka;
- Poboljšanje performansi za globalne upite, jer se rezultat može naći sa bilo kog lokalnog servera;
- Brže izvršenje upita generalno.

Ali i određene mane:

- Teško je postići konkurentnost;
- Ažuriranje je veoma sporo.

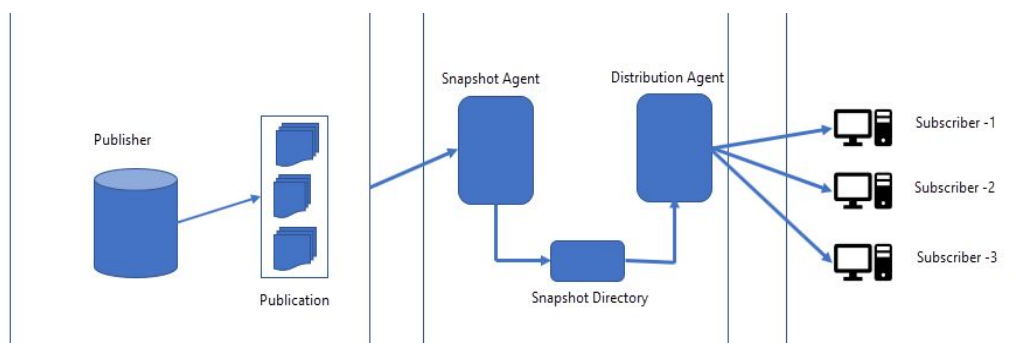


Slika1. Šema potpune replikacije

Parcijalna replikacija omogućava da se određeni fragmenti baze nalaze na nekom serveru samostalno, bez kopiranja celokupne baze. Upravo se prednost ovog načina replikacije ogleda se u tome što broj kopija fragmenata zavisi isključivo od važnosti tih podataka, tako da se oni koji se najčešće pretražuju mogu nalaziti u delovima na više lokacija umesto kopiranja svih podataka.

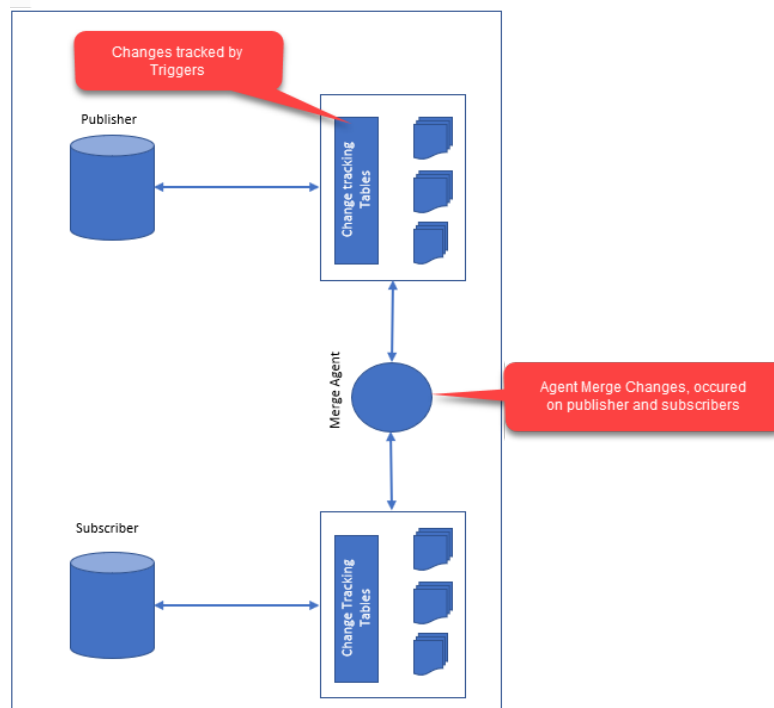
Postoji još tipova i podtipova replikacije prema raznim kriterijumima, na primer prema načinu ažuriranja čvorova postoje **sinhrona i asinhrona replikacija**. Sa strane upita, korisnici mogu pisati upite bez brige o tome kako su i gde smešteni podaci. Sa strane ažuriranja, transakcije moraju biti atomične, odnosno sve kopije se moraju ažurirati u jednoj transakciji. Ovo opisuje sinhronu replikaciju. Alternativno, kopije se mogu ažurirati periodično i transakcije koje čitaju različite kopije mogu videti različite podatke. Ovaj asinhroni način replikacije kompromituje podatke, ali se može efikasnije implementirati.

Replikaciju razlikujemo i po tipu i to snapshot, replikacija spajanja (engl. *merging*), transakciona (engl. *transactional*), heterogena (engl. *heterogeneous*) i P2P (engl. *peer-to-peer*) replikaciju. U nastavku su opisani pomenuti tipovi sa prikazanim vizuelnim šemama.



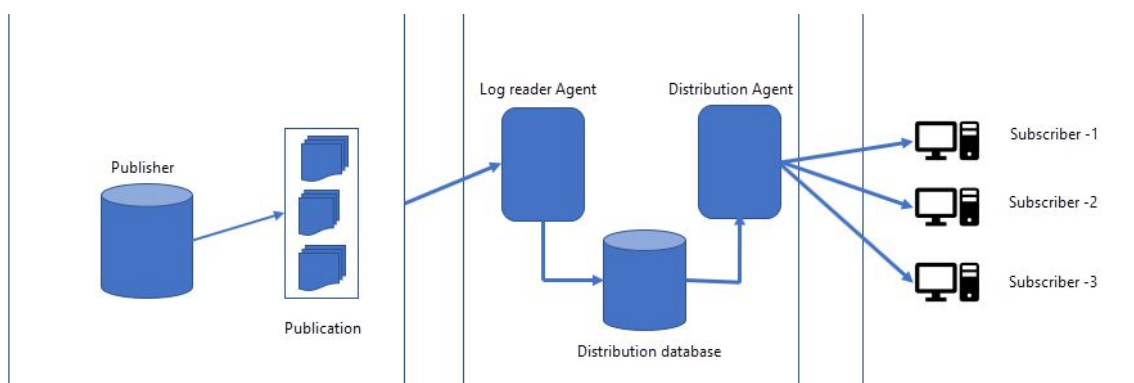
Slika2. Šema snapshot replikacije

Snapshot replikacija podrazumeva da se podaci sa jednog servera kopiraju na drugi server ili u drugu bazu na istom serveru u određenom trenutku nevezanom za trenutak ažuriranja. Koristi se kada nema čestih promena odnosno ažuriranja podataka. Snapshot replikacija je sporija od transakcione.



Slika3. Šema replikacije spajanja (engl. merge)

Replikacija spajanja označava spajanje dve ili više baza u jednu kombinovanu bazu. Ona predstavlja najsloženiji način replikacije jer dozvoljava i master čvoru i slave čvorovima da nezavisno menjaju bazu.



Slika4. Šema transakcione replikacije

Transakciona replikacija obezbeđuje korisničkom sistemu potpunu kopiju baze koja se periodično ažurira sa odgovarajućim promenama. Podaci se kopiraju u realnom vremenu, a distribuira ih master čvor (engl. *publisher*) svim svojim slave čvorovima (engl. *subscriber*) garantujući konzistentnost podataka.

U **heterogenoj replikaciji** serveri su deo heterogenog sistema, odnosno ne moraju biti u sistemu istog dobavljača. Tada je potrebno imati određeni servis za povezivanje i adaptiranje različitih servera kako bi se replikacija nesmetano dogodila.

Peer-to-Peer replikacija podrazumeva više servera i svaki od njih se ponaša i kao master i kao slave. Podaci se šalju i prihvataju od strane svakog čvora, a sinhronizuju između svih čvorova. Ovaj tip replikacije je nadogradnja na transakcioni tip. U ovom sistemu je bitno da čvor prepozna kada je promenjen, kako se ažuriranje ne bi nastavilo nakon što je svaki server ažuriran. Takođe, u ovom sistemu može doći do konflikta i nekonzistentosti podataka ako se istim podacima pristupa i oni se ažuriraju na različitim čvorovima sistema. Rešenje bi bilo da se operacije ažuriranja vrše samo na jednom čvoru, a promena se dalje propagira na ostatak sistema.

Redis

NoSQL je pokret koji je predstavljen 1998. godine, a obuhvata sve nerelacione baze podataka. Neki od primera *Not-Only-SQL* baza podataka su *Google BigTable* iz 2006. godine, Amazonov *Dynamo* iz 2007. godine, *Cassandra* u upotrebi Facebook-a iz 2008. godine i *Voldemort* koji koristi LinkedIn iz 2009. godine. Relacione baze podataka su korisne za efikasno skladištenje podataka, jer imaju podršku za ACID¹ transakcije i kompleksne SQL upite. Kada su u pitanju podaci na web-u i obrada tih podataka, pravila se menjaju i upotreba relacionih baza podataka gubi na značaju posebno zbog nedostatka fleksibilnosti.

Osnovna podela NoSQL baza podataka data je u nastavku sa njihovim karakterističnim predstavnicima u zagradama:

- Key/Value Stores (*Dynamo, Redis, Voldemort*)
- Column stores (*BigTable, Cassandra*)
- Document stores (*MongoDB*)
- Graph databases (*Neo4j*)

Redis - In-Memory DB

Tipičan predstavnik Key/Value NoSQL baze je **REmote DIctionary Server** skraćeno Redis, koji predstavlja **in-memory key-value bazu podataka sa opcionom postojanošću** (engl. *durability*). Redis je pod BSD *open-source* licencom² i upotrebljava se kao baza podataka, *cache* i *message broker*. Podržava strukture podataka kao što su stringovi, heš tabele, liste, setovi, bitmape, hiperlogovi, geoprostorni indeksi, tokovi podataka (engl. *stream*) i drugi, a razvija ga kompanija Redis Labs.

Podaci u Redis-u se uvek ažuriraju i čitaju iz glavne memorije računara, ali su skladišteni i na disku u formatu koji ne dozvoljava običan pristup podacima, a koji se koristi za rekonstrukciju podataka nakon ponovnog pokretanja sistema (engl. *restart*). Korisničke komande ne opisuju upit koji pokreće DB engine, već specifične operacije koje se dešavaju na apstraktnim tipovima podataka. Redis podržava atomične operacije visokog nivoa kao što su presek, unija, razlika, sortiranje i slično. Priroda projektovane Redis baze nalaže da su tipični slučajevi korišćenja keširanje sesija, full page keširanje, red poruka i mnoge druge. Najpoznatija kompanija koja koristi Redis je *Twitter*, a njega nude između ostalog i *Amazon Web Services*, *Microsoft Azure* i *Alibaba Cloud* kroz njihovu implementaciju ApsaraDB.

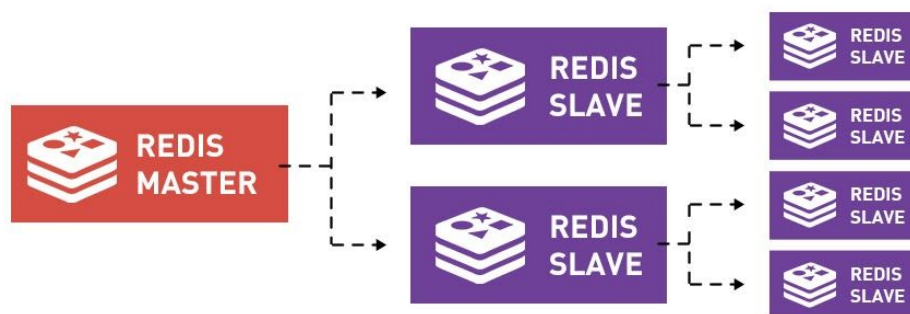
¹ *ACID* - *Atomicity Consistency Isolation Durability* - predstavlja skup svojstava transakcije u bazama podataka namenjenih da garantuju validnost čak i u slučaju grešaka, nestanka struje i slično.

² *Berkeley Software Distribution Licence* - predstavlja familiju dopusnih softverskih licenci

Replikacija podataka u Redis-u

Redis ima ugrađenu master/slave replikaciju, tako da podaci sa bilo kog Redis master servera mogu biti replicirani na veći broj slave servera. Slave čvorovi su potpune kopije instance master čvora. Određena replika može biti master nekoj drugoj replici, što omogućava implementaciju stabla replikacije sa jednim početnim čvorom (engl. *single-rooted*) što znatno smanjuje opterećenje (engl. *load balancing*). [Slika5]

U standardnom slučaju master čvor prihvata operacije upisa i čitanja, dok replike mogu samo da prihvate operaciju čitanja. Master čvor šalje komande koje dovode do modifikacija svih svojih replika odnosno slave čvorova. Međutim, replikama može biti konfigurisano da prihvataju i operacije upisa, što može dovesti do nekonzistentnosti između instanci. Replikacija je korisna za skalabilnost operacije čitanja i omogućava redundantnost podataka.



Slika5. Vizuelni prikaz stabla replikacije u Redis-u

Tri osnovna mehanizma Redis replikacije su:

1. Kada su master i njegova replika dobro povezani, **master ažurira repliku** slanjem **toka (engl. stream) komandi** kako bi iskopirao ono što se dešava podacima na master čvoru uzrokovano najčešće upisom ili nekom drugom akcijom.
2. Kada nestane konekcija između master čvora i replike, replika **automatski pokušava da se ponovo poveže** sa master čvorom i nakon toga zahteva **delimičnu sinhronizaciju** sa njim. Delimična sinhronizacija znači da će replika pokušati da obradi samo deo toka podataka koji je propustila usled nestanka konekcije.
3. U slučaju da delimična sinhronizacija nije moguća, replika će zahtevati **potpunu sinhronizaciju**, u kojoj master mora da napravi snapshot svih podataka i prosledi ga slave čvoru, a onda da nastavi da šalje tok komandi usled neke promene nad podacima.

Veoma je značajno to što je proces sinhronizacije slave čvora sa masterom ne blokirajući i master istovremeno može da nastavi normalno funkcionisanje.

Način replikacije u Redis-u je podrazumevano asinhroni, što znači da je latentnost niska, a performanse visoke. Master čvor u asinhronoj komunikaciji ne čeka svaki put repliku da procesira njegove komande iz toka, ali ukoliko mu je potrebno zna koja replika je već procesirala koji deo toka podataka odnosno koje od prosleđenih komandi. Ovime je omogućena opcionalna sinhrona replikacija koja se može ostvariti uz pomoć komande **WAIT**. Iako ovo ne dovodi Redis u jako konzistentno stanje, ipak se verovatnoća gubitka upisa nakon neke greške ili gubitka konekcije se drastično smanjuje.

Redis ima mogućnost da isključi opciju istrajnosti (engl. *persistence*) na master čvoru ili njegovim replikama, kako bi proces replikacije bio brži, ali je neophodno podesiti čvorove da se nakon ponovnog pokretanja automatski ne restartuju. Na primer, ukoliko je čvor A master čvor sa isključenom opcijom istrajnosti, a čvorovi B i C njegove replike, u slučaju da master čvor A otkáže, nakon restartovanja neće sadržati nikakve podatke, što će se dalje sinhronizovati na čvorove replike B i C i podaci će biti izgubljeni. U ovom slučaju neophodno je podesiti čvorove da ne mogu da se automatski restartuju.

Svaki Redis čvor ima ID replikacije (engl. *replication ID*) koji je predstavljen random stringom. Takođe, postoji offset koji se povećava sa svakim bajtom koji se šalje u okviru stream-a replikama zbog vođenja računa o napretku replikacije. Ovaj offset se povećava čak i ako ni jedna replika nije povezana, pa nam **uređeni par (Replication ID, offset) služi kao identifikator verzije podataka na master čvoru**. Ako dve replike imaju ovakav potpuno isti uređeni par to znači da one sadrže potpuno iste podatke. Pri konektovanju replika čvorova na novi master čvor upotrebom komande **PSYNC** šalje se uređeni par (Replication ID, offset) od starog master čvora, tako da novi master zna gde je replikacija stala. U slučaju da ne postoje odgovarajući podaci ili istorija nije više poznata dolazi do potpune sinhronizacije.

Svaki put kada se master čvor restartuje ili replika bude unapređena u master čvor generiše se novi ID replikacije za tu instancu. Replike nasleđuju ID replikacije od svog mastera nakon *handshake* procedure. Dve instance sa istim ID-jem sadrže iste podatke, a offset je ukazatelj na to kada su te podatke dobile. Na osnovu toga se određuje koja replika ima najskorije ažurirane podatke. Na primer ukoliko imamo dve replike: čvor A i čvor B sa istim ID-jem replikacije, a offset čvora A je 1064, dok je offset čvora B 1086 znamo da čvoru A nedostaju određene komande kako bi došao do istog stanja kao i čvor B.

Od verzije Redis-a 2.6 podržan je read-only mod za replike kao podrazumevano podešavanje. U okviru "redis.conf" fajla pod opcijom *replica-read-only* se može promeniti ova mogućnost. Replike po default-u ne mogu da prihvate operacije upisa, ali u određenim slučajevima ima smisla da se ovo podešavanje promeni. Sa verzijom 4.0 omogućeno je da ključevi imaju podešeno vreme trajanja odnosno da budu kreirani kao *expiring keys*. Upisi koji se dešavaju u

čvoru replike su isključivo lokalni odnosno ne propagiraju se drugim replikama koje su povezane na njega. Recimo da je master čvor A, a njegove replike čvorovi B, C i D. Čvor B ima svoje replike B1 i B2. Čvorovi B, C i D primaju tok podataka sa master čvora A, a takođe i čvorovi B1 i B2 primaju ovaj tok podataka, iako u sloju iznad njih čvor B može primati operacije upisa.

Postoji opcija konfigurisanja operacija upisa odnosno da je moguće slati operacije upisa master čvoru samo ako on ima određen minimalni broj replika čvorova povezanih na njega. U ovom slučaju replike ping-uju master čvor svake sekunde, a master čvor je dužan da pamti u kom trenutku je poslednjem bio ping-ovan od svojih replika. Možemo podesiti da minimalni broj replika (*min-replicas-to-write* <N>) bude N, a one imaju kraći vremenski zaostatak (*min-replicas-max-lag* <M>) manji od maksimum M sekundi. U slučaju da postoji bar N replika master čvora i da je vremenski zaostatak manji od M sekundi operacija upisa će biti prihvaćena. Ovom opcijom se prozor u kome podaci mogu biti izgubljeni ograničava.

Replikacija ključeva sa rokom trajanja (engl. *expiring keys*) funkcioniše tako što replika čvorovi čekaju da ključ istekne u master čvoru, a potom iniciraju komandu za njihovo brisanje **DEL**. To znači da u određenom trenutku replika čvorovi sadrže istekle ključeve, do trenutka kada do njih stigne **DEL** komanda. Zbog ove situacije replike koriste svoj logički sat kako bi javile da ključ ne postoji. Za slučaj unapređenja replike u master čvor u novom master čvoru nezavisno kreće isticanje roka ključevima.

Pri upotrebi Redis-a sa Docker³-om i sličnim servisima može nastati problem, jer se IP adresa replike razlikuje koja se koristi između replike i mastera od logičke adrese koju koristi servis. Isto važi i za port iz `redis.conf` fajla. U ovom slučaju značaj imaju komande *replica-announce-ip* i *replica-announce-port* koje služe za otkrivanje IP adrese i porta master čvoru.

SHUTDOWN komanda obezbeđuje čuvanje i gašenje replike, tako da se podaci smeštaju u RDB fajl kako bi se replika uspešno sinhronizovala nakon ponovnog pokretanja. Ukoliko je replika isključena bez čuvanja ona neće uspeti da se parcijalno sinhronizuje nakon pokretanja, već će morati da zahteva potpunu sinhronizaciju sa master čvorom.

U slučaju da neki od slave čvorova prestane sa radom ne dolazi do promene, jer podaci još uvek postoje u drugim slave čvorovima i na master čvoru. Nakon što se čvor koji nije funkcionisao oporavi on se sinhronizuje sa master čvorom automatski. U slučaju da master čvor prestane sa radom i ne oporavi se nakon određenog vremena jedan od slave čvorova

³ *Docker* - Skup platforme u vidu servisa koji koristi virtuelizaciju na nivou OS-a za isporuku softvera u kontejnerima.

automatski bude unapređen u novi master čvor. Ovo je obezbeđeno uz pomoć Redis Sentinel-a, koji je potrebno instalirati na svakom čvoru u sistemu.

Tabela 1 u nastavku prikazuje korake koji se dešavaju pri povezivanju slave čvora na master čvor.

Korak	Master čvor	Slave čvor
1	Čeka komandu	Povezuje se na master, <i>PSYNC</i> komandom
2	Pokreće <i>BGSAVE</i> komandu, čuva trenutno stanje i memoriše operacije upisa koje će slati nakon uspešnog čuvanja	Obrađuje stare podatke (ako ih ima) ili vraća grešku (u zavisnosti od konfiguracije)
3	Započinje slanje <i>snapshot-a</i> slave čvoru	Briše sve stare podatke (ako postoje i u pitanju je potpuna sinhronizacija) i učitava podatke koje dobija iz mastera
4	Završava slanje <i>snapshot-a</i> i pokreće slanje toka podataka koji sadrži operacije upisa iz bafera	Završava učitavanje podataka u memoriju, započinje da izvršava komande koje pristižu iz toka podataka
5	Završava slanje komandi iz bafera i nastavlja uživo distribuciju operacija upisa u redosledu u kojem pristižu	Izvršava operacije iz toka podataka koji pristiže

Tabela 1. Koraci pri sinhronizaciji slave čvora sa masterom

Redis Sentinel

Redis Sentinel predstavlja Redis-ovo rešenje za **nadgledanje** (engl. *monitoring*) **instanci** koje **pruža automatski oporavak** nakon pada. Slično kao kod standardne Redis replikacije, Sentinel ima jedan master čvor koji ima prioritet pri odlučivanju o izboru Redis mastera. U slučaju nedostupnosti master čvora, Sentinel bira repliku koja će preuzeti ulogu mastera. Upotreba Redis Sentinela **omogućava visoku dostupnost** (engl. *high availability*) odnosno obezbeđuje automatizaciju za određene slučajeve bez potrebe za ljudskom intervencijom. Osim toga Sentinel pruža nadgledanje, obaveštavanje i služi kao snabdevač konfiguracija klijentima.

Prednosti korišćenja više instanci Sentinela su sledeće:

- Otkrivanje otkazivanja (engl. *failure detection*) se dešava kada se više Sentinel instanci složi oko činjenice da trenutni master više nije dostupan. Smanjena je verovatnoća greške sa više instanci Sentinela.

- Sentinel funkcioniše čak i kada ne rade svi njegovi procesi, što znači da je sistem u robustnom stanju odnosno otporan na otkazivanje.



Slika6. Vizuelni prikaz strukture sistema replikacije uz Redis Sentinel

Da bi Sentinel omogućio robustnost sistema potrebno je da su pokrenute bar tri njegove instance koje je neophodno smestiti na one čvorove za koje postoji veća verovatnoća da će otkazati na neočekivan način. Potrebne su bar tri instance, jer Sentinel koristi kvorum kako bi donosio odluke. Takođe, potrebna je podrška za Sentinel kod klijenata.

Redis Sentinel se može pokrenuti na dva načina, i to uz pomoć komandi:

redis-sentinel.exe <putanja do sentinel.conf fajla>

ili

redis-server.exe <putanja do sentinel.conf fajla> --sentinel

Obe komande imaju podjednako dejstvo. Podrazumevani port koji Sentinel koristi za osluškivanje konekcija je 26379, tako da on mora biti slobodan pri pokretanju kako bi instanca Sentinela bila funkcionalna. Konfiguracija Sentinela je prikazana linijama koda u nastavku:

sentinel monitor <naziv_mastera> <adresa_hosta> <port> <kvorum>

(*kvorum* predstavlja broj Sentinela koji odlučuju po principu većine (engl. majority vote) o unapređenju replike u novi master čvor, neophodno je da postoji minimum 3 Sentinela kako bi konfiguracija imala smisla)

sentinel down-after-milliseconds <naziv_mastera> <vreme_u_milisekundama>

sentinel failover-timeout <naziv_mastera> <vreme_u_milisekundama>

sentinel parallel-sync <naziv_mastera> <broj_replika>

(*broj_replika* označava koliko replika može biti rekonfigurisano i konektovano na novog mastera nakon otkazivanja starog mastera istovremeno. Najbolje postaviti na 1.

Komande koje se koriste za konfiguraciju se odnose na master čvorove, dok će replike biti automatski otkrivene. Konfiguraciju prepisuje Sentinel za slučaj da se neka od replika unapredi u novi master čvor.

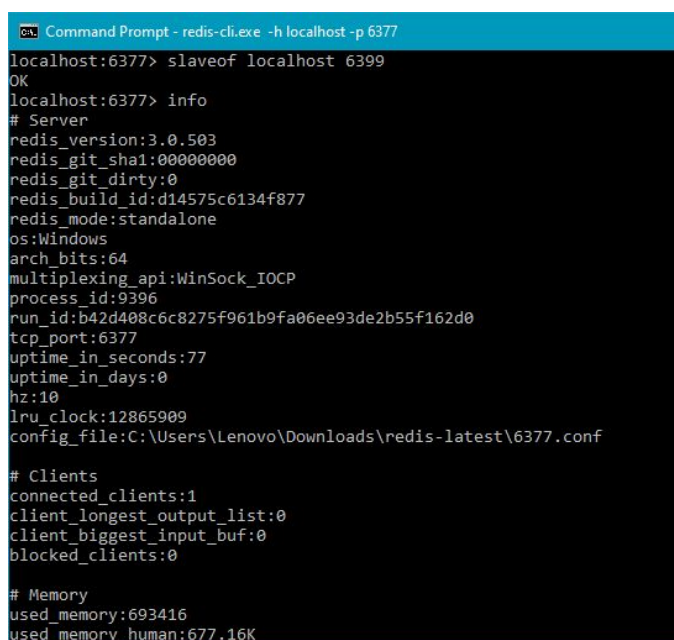
U nastavku su prikazane neke od komandi koje se koriste uz Sentinel:

- *sentinel masters* - prikazuje spisak mastera koje nadgleda sentinel i njihovo stanje (postoji i komanda *sentinel master <naziv_mastera>* za konkretan master čvor);
- *sentinel slaves* - analogno prethodnoj komandi za slave čvorove (postoji i *sentinel slave <naziv_slave_čvora>*);
- *sentinel get-master-addr-by-name <naziv_mastera>* - vraća IP adresu i port za konkretnog mastera koji je naveden;
- *sentinel ckquorum <naziv_mastera>* - koristi se za proveru podešavanja Sentinela, proverava da li trenutno podešavanje omogućava Sentinelu da postigne kvorum.

Dodavanje Sentinela u sistem je lako, preporučuje se dodavanje jednog za drugim sa pauzom od 30 sekundi, kako bi imali vremena da otkriju ceo sistem kome se priključuju. Ali uklanjanje Sentinela je komplikovanije i obuhvata zaustavljanje Sentinel procesa i slanje *sentinel reset ** komande svim Sentinelima u sistemu uz obaveznu proveru stanja Sentinela koji ostaju u sistemu.

Praktična ilustracija replikacije

Osnovna konfiguracija replikacije u Redis-u omogućava se uz pomoć komande **replicaof** *<hostname ili IP adresa>* *<port>* za verziju veću od 5.0, dok je njena alternativa za starije verzije **slaveof** kako je prikazano na Slika6. U ranijim verzijama Redis-a **replicaof** komanda se zvala **slaveof**. Replikacija bez diska (engl. diskless replication) može biti omogućena uz pomoć **repl-diskless-sync** konfiguracionog parametra. Ako je neka od replika već imala master čvor, pokretanjem nove **replicaof** komande ona će dobiti novog mastera i krenuće sinhronizaciju sa njim uz brisanje ranije sačuvanih podataka. Komandom **replicaof no one** uklanja se veza replike sa master čvorom kome je bila dodeljena. U ovom slučaju podaci neće biti obrisani sa replike.



```
Ca. Command Prompt - redis-cli.exe -h localhost -p 6377
localhost:6377> slaveof localhost 6399
OK
localhost:6377> info
# Server
redis_version:3.0.503
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:d14575c6134f877
redis_mode:standalone
os:Windows
arch_bits:64
multiplexing_api:WinSock_IOCP
process_id:9396
run_id:b42d408c6c8275f961b9fa06ee93de2b55f162d0
tcp_port:6377
uptime_in_seconds:77
uptime_in_days:0
hz:10
lru_clock:12865909
config_file:C:\Users\Lenovo\Downloads\redis-latest\6377.conf

# Clients
connected_clients:1
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0

# Memory
used_memory:693416
used_memory_human:677.16K
```

Slika6. Ilustracija povezivanja replike sa master čvorom

Za slučaj da je potrebna autentikacija master čvor može zahtevati lozinku za sve operacije sinhronizacije, a instanca replike uz komandu **config set masterauth** *<lozinka>* može biti konfigurisana za taj master čvor.

Uz pomoć komande **role** (srb. uloga) možemo saznati da li je trenutna instanca master ili slave čvor, što je prikazano na Slika7.


```
Command Prompt - redis-cli.exe -h localhost -p 6377
localhost:6377> role
1) "slave"
2) "localhost"
3) (integer) 6399
4) "connecting"
5) (integer) -1
localhost:6377>
```

Slika7. Ilustracija upotrebe komande "role"

Redis klijent (engl. *client*) predstavlja neku mašinu ili softver koji se povezuje sa serverom radi pristupanja nekom servisu. Komanda **client list** vraća skup trenutno povezanih klijenata na određeni server, na Slika8 prikazano je šta vraća ova komanda, a u nastavku se nalazi objašnjenje svakog od polja:

- **id** - jedinstveni 64-bitni ID klijenta
- **name** - naziv konekcije, može se postaviti putem client setname komande
- **addr** - adresa i port trenutne konekcije klijenta
- **fd** (engl. *file descriptor*) - opisuje soket preko koga je povezan klijent
- **age** - ukupno trajanje klijentske konekcije u sekundama
- **flags** - skup jednog ili više flag-a koji pružaju dodatne informacije o klijentu
- **db** - ID trenutne baze na koju je povezan klijent (između 0 i 15)
- **sub** - broj kanala na koje je klijent povezan
- **multi** - broj komandi koje klijent ima u redu čekanja u okviru transakcije (biće -1 za slučaj da transakcija nije u toku ili 0 ako je transakcija započeta ali nijedna komanda nije pristigla)
- **qbuf** - dužina klijentskog bafera za upite (0 znači da nema upita na čekanju)
- **qbuf-free** - količina slobodnog prostora u klijentskom baferu za upite (0 znači da je prostor u baferu popunjen)
- **obl** - dužina klijentskog izlaznog bafera
- **oli** - dužina klijentske izlazne liste (u njoj se smeštaju odgovori za slučaj da je bafer popunjen)
- **omem** - memorijski prostor koji zauzima klijentski izlazni bafer
- **events** - opisuje klijentske akcije ("r" za čitanje, "w" za upis ili oba)
- **cmd** - poslednja komanda koja je pokrenuta na klijentu

```
Command Prompt - redis-cli.exe -h localhost -p 6377
localhost:6377> client list
id=2 addr=127.0.0.1:56720 fd=9 name= age=528 idle=0 flags=N db=0
sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0
events=r cmd=client
localhost:6377>
```

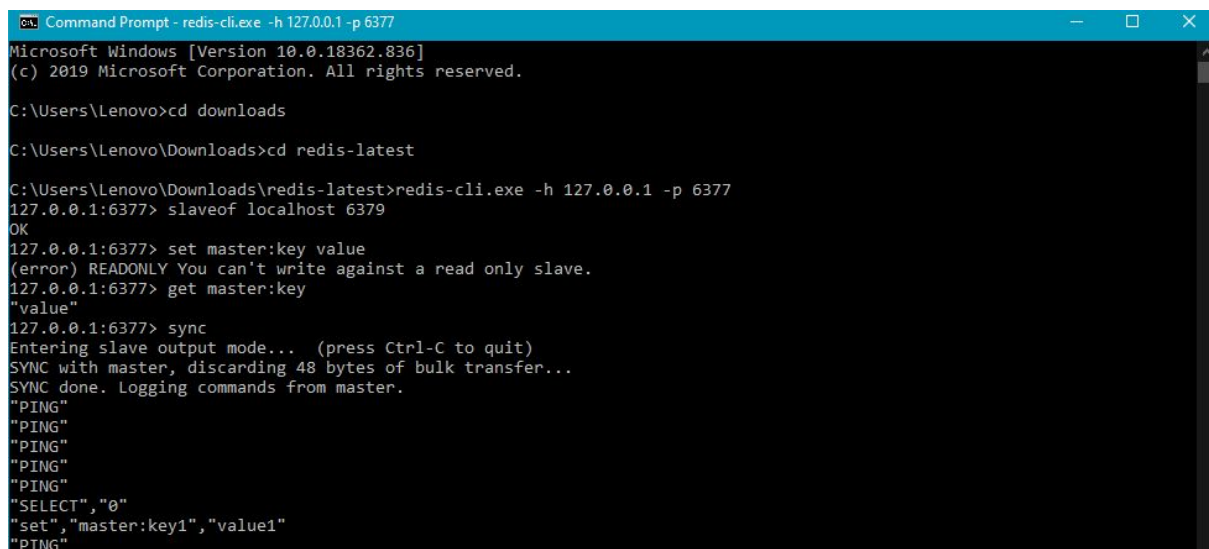
Slika8. Prikaz klijenta nakon pokretanja komande "client list"

Podrazumevan način komunikacije master čvora sa replikama je asinhroni, ali se sinhroni način može simulirati uz pomoć komande **wait** koja blokira konekciju sa trenutnim klijentom za određeno vreme izraženo u milisekundama sve dok se sve operacije upisa ne izvrše uspešno za određeni broj replika. Ako na primer želimo da blokiramo klijentsku konekciju sve dok se upis ne završi uspešno kod bar 2 replike sa pauzom od 30 milisekundi sintaksa će glasiti ovako: **wait 2 30**. Ova komdanda vraća integer vrednost koja predstavlja broj replika koje su primile operaciju upisa.

Kako bi se klijentska konekcija odblokirala neophodno je pokrenuti komandu **client unblock <client_ID>**. Za trenutno pauziranje klijentske konekcije upotrebljava se **client pause <vreme u milisekundama>**.

Za zatvaranje klijentske konekcije koristi se komanda **client kill <IP adresa i port i/ili ID klijenta i/ili tip klijenta i/ili skipme (yes/no)>**. Određivanjem tipa klijenta mogu se na primer zatvoriti konekcije svih slave čvorova, a *skipme* opcija uz vrednosti “yes” ili “no” ukazuje na to da li će komanda **kill** uticati na trenutnog klijenta koji je pokreće.

Kao što je prethodno pomenuto **potpuna sinhronizacija master čvora i replike realizuje se preko toka podataka** (engl. *stream*). Master čvor pokreće proces čuvanja podataka u pozadini, a istovremeno smešta u bafer sve operacije upisa koje mu pristižu. Čim se proces čuvanja završi, master čvor šalje bazu replikama, koje je čuvaju na disku i zatim učitavaju u memoriju. Master će onda proslediti sve operacije koje je čuvao u baferu svojim replikama kako bi ažurirali podatke. **SYNC** komanda omogućava da se pokrene sinhronizacija master čvora i replike. Ova komanda i dalje postoji, ali se umesto nje koristi **PSYNC** koja omogućava i parcijalnu sinhronizaciju. Slika9 prikazuje izgled klijenta replike nakon pokretanja **SYNC** komande, dok Slika10 prikazuje glavni server.



```
CS Command Prompt - redis-cli.exe -h 127.0.0.1 -p 6377
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd downloads
C:\Users\Lenovo\Downloads>cd redis-latest
C:\Users\Lenovo\Downloads\redis-latest>redis-cli.exe -h 127.0.0.1 -p 6377
127.0.0.1:6377> slaveof localhost 6379
OK
127.0.0.1:6377> set master:key value
(error) READONLY You can't write against a read only slave.
127.0.0.1:6377> get master:key
"value"
127.0.0.1:6377> sync
Entering slave output mode... (press Ctrl-C to quit)
SYNC with master, discarding 48 bytes of bulk transfer...
SYNC done. Logging commands from master.
"PING"
"PING"
"PING"
"PING"
"PING"
"SELECT", "0"
"set", "master:key1", "value1"
"PING"
```

Slika9. Prikaz replike sa pokrenutom komandom “sync”

```
Command Prompt - redis-server.exe 6377.conf
[16824] 19 May 18:41:28.680 # Server started, Redis version 3.0.503
[16824] 19 May 18:41:28.682 * DB loaded from disk: 0.000 seconds
[16824] 19 May 18:41:28.682 * The server is now ready to accept connections on port 6377
[16824] 19 May 18:42:50.002 * SLAVE OF localhost:6379 enabled (user request from 'id=2 addr=127.0.0.1:52954 fd=9 name= a
ge=29 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0 events=r cmd=slaveof')
[16824] 19 May 18:42:50.243 * Connecting to MASTER localhost:6379
[16824] 19 May 18:42:50.259 * MASTER <-> SLAVE sync started
[16824] 19 May 18:42:50.259 * Non blocking connect for SYNC fired the event.
[16824] 19 May 18:42:50.260 * Master replied to PING, replication can continue...
[16824] 19 May 18:42:50.261 * Partial resynchronization not possible (no cached master)
[16824] 19 May 18:42:50.275 * Full resync from master: ce36159999c5e6c6f1b33fa3ce3892d04a78e143:1
[16824] 19 May 18:42:50.418 * MASTER <-> SLAVE sync: receiving 48 bytes from master
[16824] 19 May 18:42:50.425 * MASTER <-> SLAVE sync: Flushing old data
[16824] 19 May 18:42:50.425 * MASTER <-> SLAVE sync: Loading DB in memory
[16824] 19 May 18:42:50.426 * MASTER <-> SLAVE sync: Finished with success
[16824] 19 May 18:43:38.583 * Slave 127.0.0.1:unknown-slave-port> asks for synchronization
[16824] 19 May 18:43:38.584 * Starting BGSAVE for SYNC with target: disk
[16824] 19 May 18:43:38.587 * Background saving started by pid 8444
[16824] 19 May 18:43:38.772 # fork operation complete
[16824] 19 May 18:43:38.774 * Background saving terminated with success
[16824] 19 May 18:43:38.787 * Synchronization with slave 127.0.0.1:unknown-slave-port> succeeded
[16824] 19 May 18:45:53.330 # Connection with slave 127.0.0.1:unknown-slave-port> lost.
[16824] 19 May 18:46:13.560 # Connection with master lost.
[16824] 19 May 18:46:13.562 * Caching the disconnected master state.
[16824] 19 May 18:46:13.694 * Connecting to MASTER localhost:6379
[16824] 19 May 18:46:13.694 * MASTER <-> SLAVE sync started
[16824] 19 May 18:46:15.698 * Non blocking connect for SYNC fired the event.
[16824] 19 May 18:46:15.793 # Sending command to master in replication handshake: -Writing to master: Unknown error
```

Slika10. Prikaz servera sa pokrenutom komandom "sync" na klijentu

Razlog zbog koga Redis instance imaju dva ID-ja replikacije je to što u nekom trenutku replika može biti unapređena u master čvor. Novi master čvor se i dalje seća svog ID-ja replikacije, odnosno ID-ja prethodnog master čvora. Ovo omogućava ostalim replikama da se pri pokretanju sinhronizacije sa novim master čvorom pokuša parcijalna sinhronizacija na osnovu ID-ja starog master čvora. Nakon uspešne sinhronizacije ili parcijalne sinhronizacije sa replikama, novi master čvor će izmeniti svoj ID replikacije. Moguć je i scenario u kome stari master čvor nastavlja da funkcioniše kao master čvor u mreži, pa je zbog toga neophodno da novi master čvor zameni svoj ID i izbegne konflikt.

Kao što je pomenuto u prethodnim poglavljima, **Redis podržava master/slave replikaciju u kojoj je dozvoljeno da replika odnosno slave ima svoje replike**. Ovo ne znači da je moguća multiple master replikacija i u slučaju da je neka od instanci replika omogućila prihvatanje operacija upisa, takav upis biće isključivo lokalna na toj instanci tj. neće se propagirati na njene slave instance. U nastavku je ilustrovan ovaj scenario sa 4 čvora: M na portu 6379, S1 na portu 6377, S2 na portu 6385 i S3 na portu 6381. Inicijalno je postavljeno da je M master čvorovima S1 i S2, a istovremeno je S2 master čvoru S3.

```
Command Prompt - redis-cli.exe -h localhost -p 6379
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd downloads
C:\Users\Lenovo\Downloads>cd redis-latest
C:\Users\Lenovo\Downloads\redis-latest>redis-cli.exe -h localhost -p 6379
localhost:6379> slaveof no one
OK
localhost:6379> role
1) "master"
2) (integer) 0
3) (empty list or set)
localhost:6379> flushdb
OK
localhost:6379> keys *
(empty list or set)
localhost:6379> set master:key master
OK
localhost:6379> set master:key1 master1
OK
localhost:6379> keys *
1) "master:key1"
2) "master:key"
localhost:6379>
```

Slika11. Inicijalizovan master čvor M na portu 6379

```
Command Prompt - redis-cli.exe -h localhost -p 6377
Microsoft Windows [Version 10.0.18362.836]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd downloads
C:\Users\Lenovo\Downloads>cd redis-latest
C:\Users\Lenovo\Downloads\redis-latest>redis-cli.exe -h localhost -p 6377
localhost:6377> slaveof localhost 6379
OK
localhost:6377> role
1) "slave"
2) "localhost"
3) (integer) 6379
4) "connected"
5) (integer) 1
localhost:6377> keys *
1) "master:key"
2) "master:key1"
localhost:6377>
```

Slika12. Inicijalizovan slave čvor S1 na portu 6377

Čvor M kao master obrađuje operacije upisa i propagira promene svojim replikama S1 i S2, a S2 dalje svom slave čvoru S3. Podrazumevano podešavanje replika je da ne prihvataju operacije upisa, ali se to može promeniti u konfiguraciji uz komandu *slave-read-only no*. Na taj način je konfigurisan čvor S2 koji je master čvora S3. Slika11, Slika12, Slika13 i Slika14 ilustruju početno stanje sistema za svaki od čvorova. U čvoru M su upisana dva ključa *master:key* i *master:key1*, a taj upis se može videti i kod replika S1, S2 i S3. Na čvoru S2 vršimo upis još dva ključa *slave:key* i *slave:key1* uz pomoć komande *set <key> <value>* kako je ilustrovano na Slika13. Pošto je čvor S2 jedna od replika, ovaj upis koji se dešava na njemu je lokalna i neće se propagirati dalje na njegove replike, konkretno S3 u ovom slučaju. Propagira se isključivo sadržaj koji dolazi sa master čvora u korenu stabla replikacije. Ukoliko se dogodi da master čvor otkáže replike će uporno pokušavati da se konektuju i sinhronizuju sa njim, ali bezuspešno.

```

Command Prompt - redis-cli.exe -h localhost -p 6385
localhost:6385> keys *
1) "slave:key"
2) "master:key"
3) "master:key1"
localhost:6385> keys *
1) "slave:key"
2) "master:key"
3) "master:key1"
localhost:6385> role
1) "master"
2) (integer) 358
3) 1) 1) "1800:0:0:0"
   2) "6381"
   3) "358"
localhost:6385> slaveof localhost 6379
OK
localhost:6385> keys *
1) "slave:key"
2) "master:key"
3) "master:key1"
(2.10s)
localhost:6385> set slave:key1 slave2
OK
(2.21s)
localhost:6385> keys *
1) "slave:key"
2) "master:key"
3) "slave:key1"
4) "master:key1"
localhost:6385>

```

```

Command Prompt - redis-cli.exe -h localhost -p 6381
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0

# CPU
used_cpu_sys:0.06
used_cpu_user:0.09
used_cpu_sys_children:0.00
used_cpu_user_children:0.00

# Cluster
cluster_enabled:0

# Keyspace
db0:keys=2,expires=0,avg_ttl=0
localhost:6381> role
1) "slave"
2) "localhost"
3) (integer) 6385
4) "connected"
5) (integer) 393
localhost:6381> keys *
1) "master:key"
2) "master:key1"
localhost:6381>

```

Slika13. Inicijalizovan slave čvor S2 na portu 6385 Slika14. Inicijalizovan slave čvor S3 na portu 6381

U slučaju da master čvor otkáže, njegove slave replike će pokušavati da se povežu na njega, prikazano na Slika15. Čim se master čvor ponovo pojavi i uspešno se povežu slave će pokušati da pokrene parcijalnu sinhronizaciju, a ukoliko ne uspe zatražiće potpunu sinhronizaciju sa master čvorom, prikazano na Slika16.

```

Command Prompt - redis-server.exe 6377.conf
[6176] 20 May 20:20:33.621 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:35.624 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:35.624 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:35.636 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:35.640 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:37.644 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:37.644 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:37.654 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:37.657 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:39.658 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:39.658 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:39.670 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:39.670 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:41.674 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:41.674 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:41.685 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:41.688 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:43.691 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:43.691 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:43.701 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:43.704 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:45.706 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:45.707 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:45.724 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:45.728 * MASTER <-> SLAVE sync started
[6176] 20 May 20:20:47.730 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:20:47.730 # Sending command to master in replication handshake: -Writing to master: Unknown error
[6176] 20 May 20:20:47.742 * Connecting to MASTER localhost:6379
[6176] 20 May 20:20:47.745 * MASTER <-> SLAVE sync started

```

Slika15. Slave pokušava da se poveže sa masterom nakon izgubljene konekcije


```
Command Prompt - redis-server.exe 6377.conf

Running in standalone mode
Port: 6377
PID: 6176

http://redis.io

[6176] 20 May 20:19:18.927 # Server started, Redis version 3.0.503
[6176] 20 May 20:19:18.928 * DB loaded from disk: 0.000 seconds
[6176] 20 May 20:19:18.928 * The server is now ready to accept connections on port 6377
[6176] 20 May 20:19:47.240 * SLAVE OF localhost:6379 enabled (user request from 'id=2 addr=127.0.0.1:60211 fd=9 name= ag
e=9 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0 events=r cmd=slaveof')
[6176] 20 May 20:19:48.124 * Connecting to MASTER localhost:6379
[6176] 20 May 20:19:48.131 * MASTER <-> SLAVE sync started
[6176] 20 May 20:19:48.131 * Non blocking connect for SYNC fired the event.
[6176] 20 May 20:19:48.132 * Master replied to PING, replication can continue...
[6176] 20 May 20:19:48.133 * Partial resynchronization not possible (no cached master)
[6176] 20 May 20:19:48.140 * Full resync from master: 6627308d9d2f88159c2be178234da15312f0fd03:1
[6176] 20 May 20:19:48.297 * MASTER <-> SLAVE sync: receiving 105 bytes from master
[6176] 20 May 20:19:48.305 * MASTER <-> SLAVE sync: Flushing old data
[6176] 20 May 20:19:48.305 * MASTER <-> SLAVE sync: Loading DB in memory
[6176] 20 May 20:19:48.306 * MASTER <-> SLAVE sync: Finished with success
```

Slika16. Slave koji je ponovo uspešno konekovan na master čvor

Zaključak

Redis kao in-memory NoSQL Key/Value rešenje pruža ugrađenu replikaciju po principu master/slave. U radu su opisane metode replikacije uopšteno, a zatim na konkretnom sistemu u okviru Redis-a. Objašnjene su osnovne komande za konfiguraciju i principi master/slave replikacije. Takođe, Redis-ovo rešenje za oporavak od otkazivanja i nadgledanje Redis instanci, Sentinel je koristan dodatak jednom takvom sistemu.

U Redisu je moguće podesiti da određena replika bude master nekoj drugoj replici, što omogućava implementaciju stabla replikacije sa jednim početnim čvorom, ali se operacije upisa propagiraju samo sa mastera u korenu stabla replikacije. Na ovaj način obezbeđena je konzistentnost podataka i omogućava se smanjenje opterećenja sistema.

Na kraju je kroz praktičnu ilustraciju obrađeno više situacija i simulacija jednog sistema replikacije u Redis okruženju.

Literatura

- [1] Ramakrishnan, Gehrke - "Database management systems", Treće izdanje
- [2] H. Pandey - "Data Replication in DBMS", Članak sa sajta GeeksForGeeks
<https://www.geeksforgeeks.org/data-replication-in-dbms/>
- [3] Redis - Zvanična dokumentacija
<https://redis.io/documentation>
- [4] Redis - Stranica na izvoru Wikipedia
<https://en.wikipedia.org/wiki/Redis>
- [5] M. Drake - How to manage replicas and clients in Redis, Članak sa sajta DigitalOcean
<https://www.digitalocean.com/community/cheatsheets/how-to-manage-replicas-and-clients-in-redis>
- [6] A. Goswami - DBMS Data Replication and it's Types, Članak sa sajta IncludeHelp
<https://www.includehelp.com/dbms/data-replication-and-its-types.aspx>
- [7] F. Pavlović - Scaling with Redis, Članak sa bloga kompanije INGsoftware
<https://www.ingsoftware.com/scaling-with-redis>
- [8] S. Voy - Redis replication overview, Članak sa sajta RTFM
<https://rtfm.co.ua/en/redis-replication-part-1-overview-replication-vs-sharding-sentinel-vs-cluster-redis-topology/>
- [9] Redis Sentinel - Redis blog na zvaničnoj internet stranici Redis-a
<https://redis.io/topics/sentinel>
- [10] Sentinel Clients - Redis blog na zvaničnoj internet stranici Redis-a
<https://redis.io/topics/sentinel-clients>