Elektronski fakultet Niš

Seminarski rad iz predmeta

Sistemi za upravljanje bazama podataka

Tema:

Cluster rešenja kod **Redis** baze podataka

Student:

Milica Mladenović, 1061

Mentor:

Aleksandar Stanimirović

Sadržaj

Uvod

Klasterizacija baze podataka je proces kombinovanja više servera ili instanci koji su povezani sa jednom bazom podataka. Ponekad, jedan server nije dovoljan i adekvatan za upravljanje određenom količinom podataka ili brojem zahteva i to je situacija kada je potreban klaster podataka. Glavni razlozi za klasterizaciju baza podataka su: **redundantnost podataka**, **balansiranje opterećenja**, **velika dostupnost** i na kraju, **monitoring** i **automatizacija**. U nastavku je dato kraće objašnjenje svakog od njih.

Zahvaljujući klasterizaciji podataka, više računara učestvuje u skladištenju podataka i to daje prednost redundantnosti podataka. Svi računari su sinhronizovani, što znači da će svaki čvor imati iste podatke kao i svi ostali čvorovi. U bazi podataka se moraju izbegavati određene vrste ponavljanja koje dovode do nejasnoće podataka, a razlog zbog čega se ovo dešava je upravo sinhonizacija. U slučaju da računar prestane da radi, svi podaci će biti na raspolaganju na ostalim računarima.

Balansiranje opterećenja ili skalabilnost po default-u ne dolazi uz samu bazu podataka, već se može postići pravilnom klasterizacijom podataka. Pored toga, zavisi i od samog podešavanja. U osnovi, ono što skalabilnost radi jeste raspoređivanje radnog prostora među različitim računarima koji su deo klastera. Ovo ukazuje na činjenicu da više korisnika može biti podržano i ako iz nekog razloga dođe do naglog porasta saobraćaja, postoji veća sigurnost da će novi saobraćaj moći da se podrži. Bez skalabilnosti, određeni računar bi mogao postati preopterećen i došlo bi da usporavanja ili do potpunog smanjenja saobraćaja.

Ukoliko možete pristupiti bazi podataka, to znači da je ona dostupna. Međutim, velika dostupnost se odnosi na količinu vremena u kojoj se baza podataka smatra dostupnom. Količina dostupnosti koja je potrebna u velikoj meri zavisi od broja transakcija koje se vrše u samoj bazi i od toga koliko često se vrši analiza podataka. Zahvaljujući klasterizaciji podataka, može se postići izuzetno visok nivo dostupnosti, upravo zbog postojanja dodatnih mašina i skalabilnosti. U slučaju da se server ugasi, baza će i dalje biti dostupna.

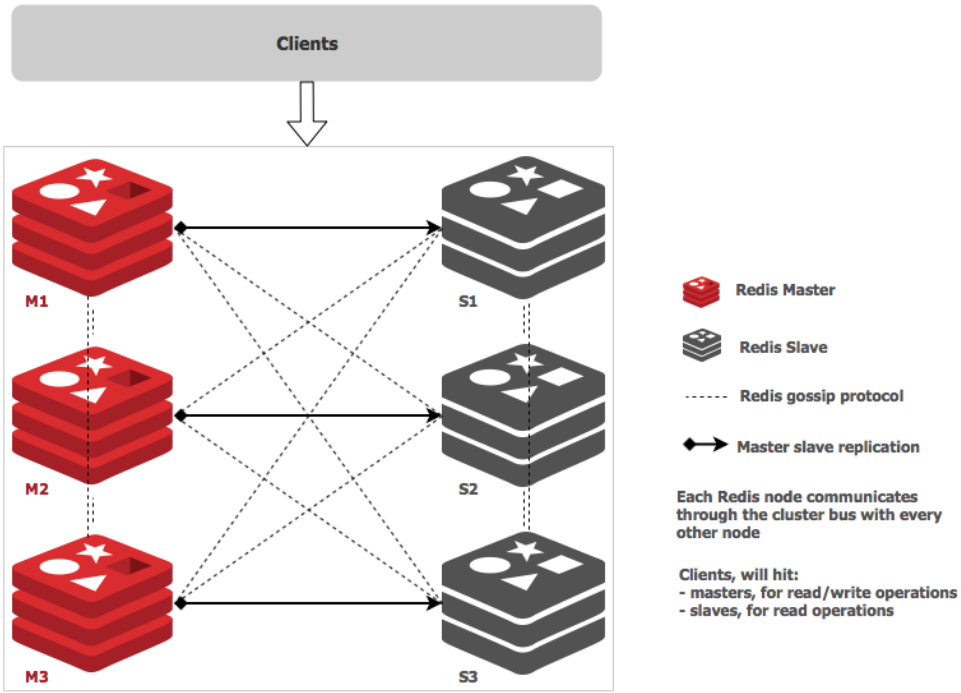
Za monitoring i automatizaciju se može korisiti uobičajena baza podataka jer se oni lako mogu izvršiti pomoću softvera, naročito ukoliko je prisutan klaster. Prednost je u tome što klasterizacija omogućava automatizaciju velikog broja procesa baze podataka u isto vreme kada omogućava postavljanje pravila radi upozoravanja na potencijalne probleme. Kod klasterovane baze podataka, automatizacija je korisna

jer će pružiti dobijanje obaveštenja ukoliko se od sistema zahteva previše. Međutim, klaster će imati određenu mašinu koja će se koristiti kao sistem za upravljanje bazom podataka za ceo klaster. Ta mašina može imati skripte koje se automatski pokreću za čitav klaster i rade sa svim ostalim čvorovima.

Obim klastera varira od preduzeća do preduzeća, zavisno od vrste procesa i nivoa potrebnih performansi. Jedna važna karakteristika klastera je sistem za upravljanje bazama podataka, koji radi na preuzimanju povratnih zahteva iz mnogih čvorova unutar distribucije putem jednog zahteva korisnika. Ovo prevazilazi složena pitanja upita i optimizuje distribuirane odgovore na upit.

Redis klasterizacija

Redis klaster je skup Redis instanci, dizajniran za skaliranje baze podataka podelom na particije i na taj način je fleksibilniji. Ukoliko primarni (master) čvor postane nedostupan, tada neki od sekundarnih (slave) čvorova preuzimaju njegovu ulogu. Komunikacija unutar klastera se vrši putem interne magistrale, korišćenjem *gossip* protokola za širenje informacija o klasteru ili za otkrivanje novih čvorova.



Redis klaster je distribuirana imlementacija Redis baze podataka sa sledećim ciljevima, poređanim po značaju u dizajnu:

- Visoke performanse i linearna skalabilnost do 1000 čvorova. Koristi se asinhrona replikacija i ne obavljaju se operacije spajanja nad vrednostima.
- Prihvatljiv stepen sigurnosti pisanja: sistem na najbolji mogući način pokušava da zadrži sve zapise koji potiču od klijenata povezanih sa većinom master čvorova. Obično postoje mali prozori u kojima potvrđeni zapisi mogu biti izgubljeni. Ovi prozori su veći kada se klijenti nalaze u manjinskoj particiji.
- Dostupnost: Redis klaster može "preživeti" particije u kojima je većina master čvorova dostupna i gde postoji bar jedan slave čvor za svaki master čvor koji više nije dostupan.

Pregled glavnih komponenti Redis klastera

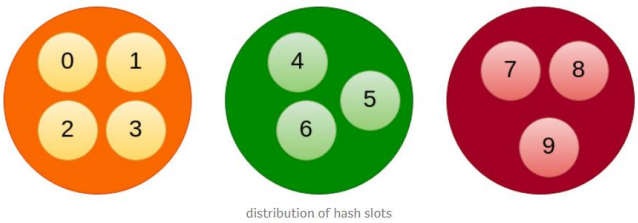
Model distribucije ključeva

Svaki ključ koji je sačuvan u Redis klasteru je povezan sa hash slotom. Prostor ključeva je podeljen na 16384 slota, postavljajući gornju granicu za veličinu klastera od 16384 master čvorova, iako je maksimalna predložena veličina do oko 1000 čvorova. Svaki master čvor u klasteru upravlja podskupom od 16384 hash slotova. Klaster je postojan kada rekonfiguracija klastera nije u toku, tj. kada se hash slotovi premeštaju sa jednog čvora na drugi. Kada je klaster stabilan, jedan čvor će opslužiti jedan hash slot. Naravno, taj čvor može imati jedan ili više podređenih čvorova koji ga mogu zameniti u slučaju da dođe do prekida mreže ili nekog drugog kvara.

Osnovni algoritam koji se koristi za mapiranje ključeva u hash slotove je sledeći:

```
HASH_SLOT = CRC16(key) mod HASH_SLOTS_NUMBER
```

Na primer, pretpostavimo da je prostor ključeva podeljen na 10 slotova (od 0 do 9). Svaki čvor će sadržati podskup hash slotova:



Atributi čvorova u klasteru

Svaki čvor u Redis klasteru ima pogled na trenutnu konfiguraciju klastera, status konekcije sa ostalim poznatim čvorovima, njihove flag-ove, karakteristike, dodeljene slotove i tako dalje. Svaki čvor ima jedinstveno ime u klasteru, koje je zapravo hexa reprezentacija 160-bitnog slučajnog broja i koje čvor dobije kada se prvi put pokrene. Čvor čuva svoj ID u konfiguracionom fajlu i koristi ga zauvek ili sve dok administrator sistema na obriše konfiguracioni fajl čvora. ID se koristi za identifikaciju svakog čvora u celom klasteru. Moguće je da određeni čvor promeni svoju IP adresu bez potrebe za promenom ID-ja. Takođe, klaster može detektovati promenu IP adrese ili porta i izvršiti rekonfiguraciju pomoću *gossip* protokola na klaster magistrali.

ID nije jedina informacija u vezi sa čvorom, ali je jedino ona uvek globalno konzistentna. Svakom čvoru je pridruženo još dodatnih informacija. Neke se odnose na detalje konfiguracije klastera datog čvora i konzistentne su u klasteru. Neke druge informacije, poput podatka kada je čvor poslednji put pingovan, su sa druge strane lokalne za svaki čvor.

Svaki čvor održava sledeće informacije o ostalim čvorovima u klasteru kojih je svestan: ID, IP adresu i port, skup flag-ova, ukoliko je u pitanju slave čvor, koji je njegov master čvor, poslednji put kada je pingovan, i poslednji put kada je primio *pong*, trenutno stanje konfiguracije, stanje veze i na kraju skup hash slotova.

Naredba **CLUSTER NODES** se može poslati bilo kom čvoru u klasteru i ona pruža uvid u stanje klastera i informacije o svakom čvoru u skladu sa lokalnim informacijama koje poseduje čvor kome je ova naredba upućena. U nastavku je dat primer ove naredbe koja se šalje masteru u malom klasteru od 3 čvora:

```
$ redis-cli cluster nodes
d1861060fe6a534d42d8a19aeb36600e18785e04 127.0.0.1:6379 myself - 0 1318428930 1 connected 0-1364
3886e65cc906bfd9b1f7e7bde468726a052d1dae 127.0.0.1:6380 master - 1318428930 1318428931 2 connected 1365-2729
d289c575dcbc4bdd2931585fd4339089e461a27d 127.0.0.1:6381 master - 1318428931 1318428931 3 connected 2730-4095
```

, pri čemu se svaka linija se sastoji od sledećih polja:

```
<id> <ip:port@cport> <flags> <master> <ping-sent> <pong-recv> <config-epoch> <link-state> <slot> <slot> ... <slot>
```

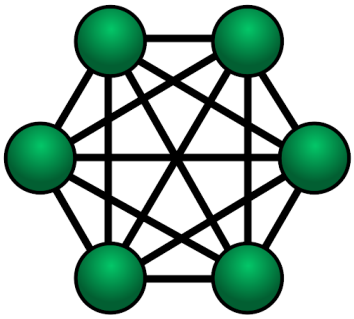
Klaster magistrala

Svaki čvor u Redis klasteru ima dodatni TCP port za prijem dolaznih konekcija od drugih čvorova u klasteru. Ovaj port je na fiksnom rastojanju od uobičajenog TCP porta koji se koristi za prijem dolaznih konekcija od klijenata. Da biste dobili Redis klaster port, potrebno je dodati 10000 na regularni port za naredbe. Na primer, ako Redis čvor osluškuje konekcije klijenata na portu 6379, na klaster magistrali će biti otvoren port 16379.

Komunikacija između čvorova se obavlja isključivo pomoću klaster magistrale i binarnog protokola koji je sastavljen od frejmova različitih vrsta i veličina. Binarni protokol klaster magistrale nije javno dokumentovan jer nije namenjen za komunikaciju eksternih softverskih uređaja sa čvorovima u Redis klasteru. U svakom slučaju, moguće je dobiti više detalja o ovom protokolu iz fajlova *cluster.h* i *cluster.c* koji se nalaze u izvornom kodu Redis klastera.

Topologija klastera

Redis klaster ima **full mesh** topologiju u kojoj je svaki čvor povezan sa svakim drugim čvorom u mreži pomoću TCP konekcije. U klasteru koji ima N čvorova, svaki ima N-1 izlaznih i N-1 ulaznih konekcija.



Full mesh topologija

Ove TCP konekcije se konstantno održavaju i ne kreiraju se na zahtev. Kada čvor očekuje *pong* kao odgovor na *ping* na klaster magistrali, umesto da čeka dovoljno dugo da bi označio čvor nedostupnim, on će pokušati da osveži konekciju sa drugim čvorom ponovnim povezivanjem od nule.

Iako čvorovi u Redis klasteru imaju ovu vrstu topologije, oni koriste *gossip* protokol i mehanizam za ažuriranje konfiguracije kako bi izbegli razmenu prevelikog broja poruka u normalnim uslovima, tako da broj razmenjenih poruka nije eksponencijalan.

Interakcija čvorova

Čvorovi uvek prihvataju konekcije na portu klaster magistrale, čak i odgovaraju na primljeni *ping*, i u slučaju da se pingovanom čvoru ne može verovati. Međutim, čvor će odbaciti sve ostale pakete ukoliko smatra da čvor koji ih je poslao nije deo klastera. Čvor će prihvatiti neki drugi čvor kao deo klastera samo na dva načina:

- Ako čvor predstavi samog sebe MEET porukom. Ona je identična *ping* poruci, s tim što primorava prijemnika da prihvati čvor kao deo klastera. Čvorovi šalju ovu poruku ostalim čvorovima samo ako administrator sistema to zatraži pomoću odgovarajuće komande
- Čvor će takođe registrovati drugi čvor kao deo klastera ukoliko mu već poznati čvor da neke informacije o njemu. Na primer, ako A zna za B, a B zna za C, B će poslati informacije čvoru A o čvoru C. Kada se to dogodi, A će priznati C kao deo klastera i pokušaće da se poveže sa njim.

Ovo znači da je klaster u stanju da automatski otkrije druge čvorove, ali samo ako postoji pouzdan odnos između čvorova, koga stvara administrator sistema. Ovaj mehanizam čini klaster robusnijim, ali sprečava da se različiti Redis klasteri mešaju nakon promene IP adrese ili drugih događaja na mreži.

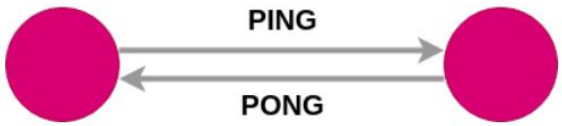
Tolerancija na greške

Heartbeat i gossip poruke

Čvorovi u Redis klasteru obično razmenjuju *ping* i *pong* pakete koji imaju istu strukturu i sadrže važne informacije. Jedina stvarna razlika jeste polje koje je namenjeno za tip poruke. Zajedničko ime za ove pakete je **heartbeat** paketi.

Heartbeat paketi sadrže zaglavlje koje je zajedničko svim tipovima paketa i posebnu gossip sekciju koja je specifična za ping i pong pakete. Ona nudi primaocu prikaz onoga što pošiljalac misli o drugim čvorovima u klasteru i sadrži samo informacije o nekoliko slučajnih čvorova poznatih pošiljaocu. Broj čvorova koji se spominju u ovoj sekciji je proporcionalan veličini klastera i za svaki od njih se prijavljuju sledeća polja: ID, IP adresa i port i flag-ovi.

Gossip sekcije omogućavaju primaocima da dobiju informacije o stanju drugih čvorova sa stanovištva pošiljaoca. Ovo je korisno kako za otkivanje grešaka, tako i za otkrivanje drugih čvorova u klasteru.



PING: Hi! Are you ok?

Gossip: Here are the info on other nodes I'm in touch with

A said he's Ok
I can't reach B. Can you reach B?

PONG: Hey! Yes, I'm Ok

Gossip: Here are the nodes I'm connected with

C & D both are Ok.
But B didn't reply to me as well!
I think B is down

Čvorovi uglavnom šalju *ping* pakete koji će aktivirati čvorove prijemnike da odgovore *pong* paketima. Međutim, nije uvek tako. Može se desiti da čvorovi jednostavno pošalju *pong* pakete kako bi ostalima poslali informacije o svojoj konfiguraciji. Ovo je, na primer, korisno za emitovanje nove konfiguracije što je pre moguće.

Čvor će obično pingovati nekoliko slučajnih čvorova svake sekunde tako da je ukupan broj poslatih *ping* paketa (a i primljenih *pong* paketa) konstantan po svakom čvoru, bez obzira na broj čvorova u klasteru. Svaki čvor obavezno pinguje svaki drugi koji nije poslao *ping* ili primio *pong* paket duže od polovine **NODE_TIMEOUT** intervala. Pre nego što ovaj vremenski interval istekne, čvorovi pokušavaju da osveže TCP konekciju sa drugim čvorom kako bi bili sigurni da se čvorovi ne smatraju nedostupnima samo zato što postoji problem u trenutnoj TCP vezi.

Broj globalno razmenjenih poruka može biti povećan ako je NODE_TIMEOUT postavljen na malu vrednost, a broj čvorova (N) veliki jer će svaki čvor pokušati da pinguje sve ostale za koje nije dobio sveže informacije u svakoj polovini ovog vremenskog intervala. Na primer, u klasteru koji ima 100 čvorova i gde je timeout čvora 60 sekundi, svaki čvor će pokušati da pošalje 99 pingova na svakih 30 sekundi, odnosno 3.3 pinga u sekundi. Pomnoženo sa 100 čvorova, to je 330 pingova u sekundi u celom klasteru. Postoje načini i za smanjenje broja poruka, ali treba imati na umu da je u ovom primeru, razmenjenih 330 paketa u sekundi ravnomerno podeljeno između 100 različitih čvorova, tako da je saobraćaj koji svaki čvor prima prihvatljiv.

Otkrivanje grešaka

Otkrivanje grešaka u Redis klasteru je zapravo utvrđivanje da master ili slave čvor više nije dostupan većini čvorova i tada je potrebno reagovati na način da slave čvor preuzme ulogu mastera u klasteru. Ukoliko to nije moguće učiniti, klaster prelazi u stanje greške kako bi prestao da prima upite od klijenata.

Kao što je već pomenuto, svaki čvor ima listu flag-ova koja se odnosi na ostale poznate čvorove. Postoje dva flag-a koja se koriste za detekciju grešaka i nazivaju se **PFAIL** (Posible Failure) i **FAIL**. PFAIL označava moguću grešku i to je nepriznati tip greške. Sa druge strane, FAIL znači da čvor sigurno propada i ovo stanje potvrđuje većina mastera u klasteru u određenom vremenskom roku.

Čvor označava drugi čvor PFAIL flag-om kada on nije dostupan duže nego što traji timeout interval. I master i slave čvorovi mogu označiti neki čvor ovim flag-om bez obzira na njegov tip.

Koncept nedostupnosti nekog čvora u Redis klasteru je kada postoji aktivni *ping*, odnosno ping koji je poslat ali za koji još uvek nije stigao odgovor, i koji čeka duže od vrednosti timeout intervala. Da bi ovaj mehanizam funkcionisao, NODE_TIMEOUT mora biti veliki u poređenju sa vremenom putovanja kroz mrežu. Da bi se povećala pouzdanost tokom normalnih operacija, čim istekne polovina timeout intervala bez dobijanja odgovora na *ping*, čvorovi će ponovo pokušati da se povežu sa ostalima. Na taj način se osigurava da će veze ostati aktivne tako da one koje su prekinute uglavnom neće rezultirati lažnim izveštajima o greškama između čvorova.

PFAIL flag je samo lokalna informacija koju svaki čvor ima o drugima, ali nije dovoljna da bi se neki od njih izbacio iz klastera. Da bi se to desilo, PFAIL stanje mora preći u FAIL pri čemu moraju biti ispunjeni sledeći uslovi:

- Neki čvor, recimo A, ima drugi čvor B koji je označen kao PFAIL
- Čvor A je prikupio informacije o B sa stanovišta većine mastera u klasteru, preko *gossip* sekcije
- Većina mastera je signalizirala jedno od ova dva stanja u vremenskom intervalu od NODE_TIMEOUT * FAIL_REPORT_VALIDITY_MULT (gde je drugi član u izrazu faktor validnosti koji je u trenutnoj implementaciji jednak dvojci)

Ako su svi prethodni uslovi ispunjeni, A ce označiti čvor kao FAIL i poslati FAIL poruku svim ostalim čvorovima u klasteru nakon koje su i oni primorani da ga označe na isti način.

FAIL flag se koristi kao okidač za pokretanje algoritma u kom slave preuzima ulogu mastera. Medjutim, u praksi, slave može delovati nezavisno i samostalno i zauzeti mesto svog mastera ukoliko je on nedostupan za njega. Ukoliko je dostupan za ostale master čvorove u klasteru, oni neće podržati taj isti slave u odluci koju je doneo. Zahvaljujući ovim mehanizmima, obično će svi čvorovi u isto vreme prestati da prihvataju zahteve ako se klaster nalazi u stanju greške, što je vrlo poželjna funkcija sa stanovišta aplikacija koje koriste Redis klaster. Takođe, izbegavaju se pogrešne odluke slave čvorova koji, zbog lokalnih problema, ne mogu doći do svog master čvora (koji je inače dostupan većini drugih mastera u klasteru).

Konfiguracioni parametri Redis klastera

U nastavku je dato objašnjenje konfiguracionih parametara koje Redis klaster uvodi u konfiguracioni fajl *redis.conf*, radi boljeg razumevanja primera stvaranja samog klastera:

- **cluster-enabled** <yes/no>: U *yes* slučaju, omogućava se podrška klasterizaciji u određenoj Redis instanci, u suprotnom instanca je samostalna kao i obično.
- **cluster-config-file** <filename>: Treba imati na umu da, uprkos nazivu ove opcije, ovo nije konfiguracioni fajl koga može editovati korisnik, već fajl u kome čvor iz Redis klastera automatski pamti konfiguraciju klastera (stanje, konkretno) svaki put kada dođe do promene. Fajl navodi niz stvari poput liste čvorova u klasteru, njihovih stanja, postojanih promenljivih i tako dalje.
- **cluster-node-timeout** <milliseconds>: Maksimalni vremenski interval u kome čvor može biti nedostupan, a da se ne smatra ugašenim. Ako master čvor nije dostupan duže od vrednosti trajanja timeout-a, njegovi slave čvorovi zameniti. Ovaj parametar kontroliše mnoge važne stvari u Redis klasteru.
- **cluster-slave-validity-factor** <factor>: Ako je vrednost pozitivna, maksimalno vreme prekida konekcije se računa kao vrednost timeout-a čvora pomnožena faktorom koji je određen ovom opcijom, a ako je u pitanju slave čvor, on neće pokušati da započne failover nad master čvorom ako je konekcija prekinuta duže od specificiranog vremena.

- **cluster-migration-barrier** <count>: Minimalni broj slave čvorova sa kojima je master ostao povezan kako bi drugi slave čvor migrirao na master čvor koji je ostao bez ijednog slave-a.
- **cluster-require-full-coverage** <yes/no>: ako je ovaj parametar postavljen na *yes*, kao što je i podrazumevano, klaster prestaje sa prihvatanjem zahteva ako neki deo prostora ključeva nije pokriven nijednim čvorom. Ako je postavljen na *no*, klaster će i dalje opsuživati upite čak iako se mogu obraditi samo pojedini zahtevi.
- **cluster-allow-reads-when-down** <yes/no>: ako je ovaj parametar postavljen na *no*, kao što je i podrazumevano, čvor će prestati da opslužuje čitav saobraćaj kada u klasteru postoji neka greška. Ovo sprečava čitanje potencijalno nekonzistentnih podataka sa čvora koji nije svestan stanja u klasteru. Ova opcija se može postaviti na *yes*, kako bi se omogućilo čitanje sa čvora tokom stanja neuspeha, što je korisno za aplikacije koje žele da odrede prioritet dostupnosti čitanja, ali ipak žele da spreče nedosledno pisanje.

Kreiranje i korišćenje Redis klastera

Da bi se kreirao klaster, prvo što je potrebno je nekoliko praznih Redis instanci koje su pokrenute u klaster modu. To praktično znači da klasteri nisu stvoreni korišćenjem normalnih instanci jer je potrebno konfigurisati poseban mod kako bi Redis instanca omogućila određene karakteristike i naredbe klastera.

Ovo je minimalno što je potrebno dodati u Redis datoteku za konfiguraciju klastera:

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

Ono što omogućava način rada u klaster modu je jednostavna direktiva *cluster-enabled*. Svaka instanca sadrži i putanju do fajla u kome se čuva njena konfiguracija, a to je po default-u fajl *nodes.conf*. Korisnici nikada ne menjaju ovaj fajl. Njega generišu instance u Redis klasteru pri pokretanju i ažuriraju ga kad god je to potrebno.

Treba imati na umu da minimalni klaster, da bi radio onako kako se očekuje, treba da sadrži najmanje 3 master čvora. U cilju testiranja, preporuka je pokretanje klastera sa 6 čvorova, od kojih su 3 master, a 3 slave čvorovi. Da bi se to desilo, potrebno je napraviti novi direktorijum i unutar njega kreirati 6 direktorijuma nazvanih brojevima koji slede nakon broja porta instance koju ćemo pokrenuti unutar bilo kog od njih. Na primer:

```
mkdir cluster-test
cd cluster-test
mkdir 7000 7001 7002 7003 7004 7005
```

Potrebno je napraviti konfiguracioni fajl *redis.conf* unutar svakog direktorijuma, od 7000 do 7005. Kao templejt za minimalnu konfiguraciju se može iskoristiti gornji primer, s tim što je potrebno zameniti broj porta (koji je u primeru 7000) brojem koji odgovara nazivu direktorijuma.

Zatim treba kopirati redis-server izvršnu datoteku, preuzetu iz najnovijih izvora sa GitHub-a, u *cluster-test* direktorijum i otvoriti 6 novih tabova u terminalu (Command Prompt). Svaku instancu treba pokrenuti u posebnom tabu na sledeći način (ukoliko se prethodno nalazimo u folderu *cluster-test*):

```
cd 7000
../redis-server ./redis.conf
```

U logovima svake od instanci se vidi da svaki čvor dodeljuje sebi ID, obzirom da fajl *nodes.conf* ne postoji:

```
[82462] 26 Nov 11:56:55.329 * No cluster configuration found, I'm 97a3a64667477371c4479320d683e4c8db5858b1
```

Kada instanca dobije svoj ID, koristiće ga zauvek kako bi imala jedinstveno ime u kontekstu klastera. Svaki čvor pamti svaki drugi koristeći upravo ove ID-eve, umesto IP adrese ili broja porta jer se oni mogu promeniti, dok se jedinstveni identifikator ne menja nikada dok god čvor postoji.

Sada, kada postoji nekoliko pokrenutih Redis instanci, potrebno je krerati klaster pisanjem neke smislene konfiguracije za svaku od njih. U Redis verziji 5, to je veoma lako ostvariti pomoću uslužnog programa Redis klijenta koji se može koristiti za kreiranje novih klastera, proveru ili promenu postojećeg i tako dalje. Da bismo stvorili klaster, u redis-cli je potrebno ukucati sledeće:

```
redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 \
127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \
--cluster-replicas 1
```

Ovde se koristi naredba *create* zbog toga što je potrebno kreirati klaster. Opcija *--cluster-replicas 1* znači da želimo slave čvor za svaki kreirani master. Ostali argumenti su lista adresa instanci koje će se koristiti za kreiranje klastera.
Očigledno da je jedini setup u skladu sa navedenim zahtevima kreirati master koji ima 3 master i 3 slave čvora. Redis-cli će sam predložiti konfiguraciju koju treba prihvatiti, upisivanjem *yes* u terminal. Posle svega ovoga, klaster će biti konfigurisan i pridružen. Konačno, ako sve prođe kako treba, pojavljuje se sledeća poruka što znači da postoji bar jedan master čvor koji opslužuje svaki od 16384 dostupnih slotova:

```
[OK] All 16384 slots covered
```

Podela podataka (*resharding*) u klasteru

Resharding generalno znači premeštanje hash slotova iz jednog skupa čvorova u drugi skup čvorova, i poput kreiranja klastera, to se izvodi pomoću uslužnog programa *redis-cli*. Potrebno je upisati sledeće:

```
redis-cli --cluster reshard 127.0.0.1:7000
```

Potrebno je navesti samo jedan čvor, redis-cli će ostale čvorove pronaći automatski. Trenutno redis-cli može da obavi resharding samo uz podršku administratora. Ne možemo reći, na primer, da želimo da prebacimo 5% slotova sa jednog čvora na drugi, već će nam se pojavljivati pitanja od kojih je prvo koliko slotova želimo da prebacimo:

```
How many slots do you want to move (from 1 to 16384)?
```

Zatim je potrebno navesti čvor koji će primiti hash slotove, odnosno njegov jedinstveni identifikator. On postoji u listi, ali uvek ga je moguće dobiti unosom sledeće komande (pod pretpostavkom da želimo prvi master čvor, na portu 7000):

```
$ redis-cli -p 7000 cluster nodes | grep myself
97a3a64667477371c4479320d683e4c8db5858b1 :0 myself,master - 0 0 0 connected 0-5460
```

Sledeće pitanje je iz kojih čvorova želimo da uzmemo hash slotove. Na kraju ovog procesa, može se istestirati klaster pomoću sledeće naredbe:

```
redis-cli --cluster check 127.0.0.1:7000
```

Resharding se može izvršiti i automatski bez potrebe za ručnim unošenjem parametara na interaktivan način. Ovo je moguće uraditi pomoću komandne linije:

```
redis-cli reshard <host>:<port> --cluster-from <node-id> --cluster-to <node-id> --cluster-slots <number of slots> --cluster-yes
```

Ovo omogućava izgradnju nekog automatizma ukoliko je verovatno da će se u budućnosti opet vršiti resharding jer trenutno ne postoji način da redis-cli automatski rebalansira klaster proveravajući raspored ključeva po čvorovima i prebacivajući slotove po potrebi na pametan način. Ova karakteristika će biti dodata u budućnosti.

Dodavanje i izbacivanje čvora

Dodavanje novog čvora je u osnovi proces dodavanja praznog čvora u klaster i potom premeštanja nekih podataka u njega, u slučaju da je on novi master, ili nagoveštavanja da je potrebno da se postavi kao replika poznatog čvora, u slučaju da je on slave.

Recimo, da je potrebno dodati novu master instancu. Prvi korak je dodavanje praznog čvora (postupak je isti kao i prethodno opisani prilikom stvaranja inicijalnih 6 instanci). Zatim se koristi redis-cli kako bi se novi čvor dodao postojećem klasteru:

```
redis-cli --cluster add-node 127.0.0.1:7006 127.0.0.1:7000
```

Za to se koristi naredba *add-node* kojoj se kao prvi argument navodi adresa novog čvora i kao drugi adresa slučajnog čvora koji već postoji u klasteru. Praktično, *redis-cli* nije mnogo pomogao, međutim pre nego što nešto uradi, on proverava stanje klastera, tako da je dobra ideja izvoditi operacije klasterizacije pomoću njega.

Nakon celog postupka, moguće je povezati se sa novim čvorom da bi se utvrdilo da li se zaista pridružio klasteru:

```
redis 127.0.0.1:7006> cluster nodes
3e3a6cb0d9a9a87168e266b0a0b24026c0aae3f0 127.0.0.1:7001 master - 0 1385543178575 0 connected 5960-10921
3fc783611028b1707fd65345e763befb36454d73 127.0.0.1:7004 slave 3e3a6cb0d9a9a87168e266b0a0b24026c0aae3f0 0 1385543179583 0 connected
f093c80dde814da99c5cf72a7dd01590792b783b :0 myself,master - 0 0 0 connected
2938205e12de373867bf38f1ca29d31d0ddb3e46 127.0.0.1:7002 slave 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 0 1385543178072 3 connected
a211e242fc6b22a9427fed61285e85892fa04e08 127.0.0.1:7003 slave 97a3a64667477371c4479320d683e4c8db5858b1 0 1385543178575 0 connected
97a3a64667477371c4479320d683e4c8db5858b1 127.0.0.1:7000 master - 0 1385543179080 0 connected 0-5959 10922-11422
3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 127.0.0.1:7005 master - 0 1385543177568 3 connected 11423-16383
```

Novom čvoru je moguće dodeliti hash slotove pomoću resharding-a na prethodno opisan način.

Za uklanjanje slave čvora koristi se naredba *del-node* u *redis-cli* gde je prvi argument slučajni čvor iz klastera, a drugi ID čvora koji se uklanja:

```
redis-cli --cluster del-node 127.0.0.1:7000 `<node-id>`
```

Moguće je ukloniti i master čvor na isti način, ali da bi se uklonio, on mora biti prazan. Ukoliko to nije slučaj, prvo je potrebno sve njegove podatke prebaciti na ostale master čvorove.

Zaključak

Redis klaster je jednostavno strategija raspodele podataka. On automatski deli podatke između više Redis čvorova. To je napredna verzija Redisa koja postiže distribuirano skladištenje i sprečava bilo koju vrstu greške.
Zaključno, Redis klaster ima sledeće karakteristike:

- Horizontalna skalabilnost: možemo nastaviti da dodajemo nove čvorove dok god se povećava potreba za kapacitetom
- Automatsko podešavanje podataka: može automatski particionisati i podeliti podatke među čvorovima
- Tolerantnost na greške: iako dođe do gubitka čvora, i dalje se može raditi bez gubitka podataka
- Decentralizovano upravljanje klasterom: nijedan čvor se ne ponaša kao dirigent celog klastera, već svaki čvor učestvuje u konfiguraciji klastera putem *gossip* protokola

Reference

1. https://redis.io/topics/cluster-tutorial, Redis cluster tutorial

2. https://redis.io/topics/cluster-spec, Redis cluster specification

3. https://ndimensionz.com/kb/what-is-database-clustering-introduction-and-brief-explanation/, What is database clustering - Introduction and brief explanat