Elektronski fakultet Niš

Sistemi za upravljanje bazama podataka

Distribuirane baze podataka i MongoDB

Seminarski rad

Mentor:
Doc. dr Aleksandar Stanimirović

Student:
Julije Kostov 1026

Sadržaj

1. Uvod

Distribuirana baza podataka je baza podataka kod koje su podaci distribuirani na više lokacija. Čvorovi koji upravljaju i skladište podatke mogu da se nalaze na istom fizičkom čvoru ili više povezanih čvorova. Ti čvorovi zajedno rade na skladištenju i procesiranju podataka. Kod distribuiranih baza podataka sistem za upravljanje bazom podataka krajnjem korisniku omogućava jednostavan pristup koji sakriva detalje distribucije podataka [1].

Distribuirane baze podataka imaju sledeće karakteristike:

- Pouzdanost – kreiraju se kopije nad podacima pa pad jednog čvora ne dovodi do pada čitavog sistema, podaci se mogu preuzeti sa drugih čvorova.
- Performanse – kako se čvorovi mogu nalaziti na drugim lokacijama, moguće je da se traženi podaci nalaze na čvoru blizu klijenta što smanjuje vreme odziva.
- Skalabilnost – omogućava jednostavno dodavanje novog čvora, vertikalno skaliranje.
- Lokana autonomija čvorova
- Distribuirane transakcije i upiti

Pored navedenih prednosti, distribuirane baze podataka imaju sledeće nedostatke:

- Kompleksnost
- Sigurnost – potrebno je zaštititi pristup čvorovima koji se nalaze na različitim lokacijama.
- Održavanje – veći troškovi održavanja infrastrukture.
- Konzistentnost – potrebni su dodatni napori za održanje konzistentnih podataka na više čvorova.
- Komunikacija – zavisi od mreže. Povećan mrežni saobraćaj.

Distribuirana baza podataka treba da zadovolji sledeće principe:

- Lokalna autonomija čvorova – baza podataka svakog čvora se ponaša kao da nije deo distribuirane baze podataka
- Nepostojanje centralnog čvora – svi čvorovi su ravnopravni. Postojanje centralnog čvora predstavlja usko grlo sistema
- Transparentnost distribucije – krajnji korisnik pristupa bazi podataka na jednostavan način bez znanja o postojanju distribucije

Ovaj rad opisuje principe distribuiranih baza podataka, kao i primenu istih u Mongo bazi podataka. U trećem poglavlju dat je primer korišćenja Mongo baze u distribuiranom modu izvršenja.

2. Principi distribuiranih baza podataka

CAP teorema kaže da je kod distribuiranih baza podataka nemoguće obezbediti sve tri osobine:

- Konzistentnost (eng. *Consistency*) – svaki čitanje podataka dobija poslednju verziju podatka ili baca grešku
- Dostupnost (eng. *Availability*) – svaki zahtev dobija dogovor u garantovanom vremenskom roku, bez garancije da sadrži poslednju verziju podataka
- Tolerancija razdvojenosti (eng. *Partition tolerance*) – sistem mora da radi bez obzira na zakasnele ili izgubljene poruke. Sistem mora da radi i kada dođe do delimičnog otkaza

Kako CAP teorema kaže da nije moguće izgraditi sistem koji ima sve tri osobine, potrebno je napraviti kompromis oko jedne. Odbacivanjem tolerancije razdvojenosti obezbeđuje se konzistentnost i dostupnost. Kao alternativa toga je da ceo sistem radi na jednom računaru ili da se koriste redundantne veze što može da bude skupo. Kompromis oko osobine se ređe pravi. Ako se pravi kompromis oko dostupnosti onda se postiže konzistentnost i tolerancija razdvojenosti. Ovim se ne garantuje odziv sistema pa se i ovaj kompromis ređe pravi jer sistem koji nije dostupan nije ni upotrebljiv. Ako se pravi kompromis oko konzistentnosti onda sistem ima osobine dostupnosti i tolerancije razdvojenosti [2]. Zbog toga se ne garantuje čitanje poslednje vrednosti podataka. Kod ovakvih sistema ACID osobine zamenjuju se BASE osobinama:

- Suštinski raspoloživ – većina podataka je dostupna veći deo vremena
- Nekonzistentno stanje – baza podataka ne mora biti konzistentna u svakom trenutku
- Konvergentna konzistentnost – dodavanjem novog čvora podaci se repliciraju na njega. Ne postoji garancija da će u svakom trenutku svi čvorovi sadržati identične kopije podataka, ali se teži tome da svi sadrže konzistentne podatke

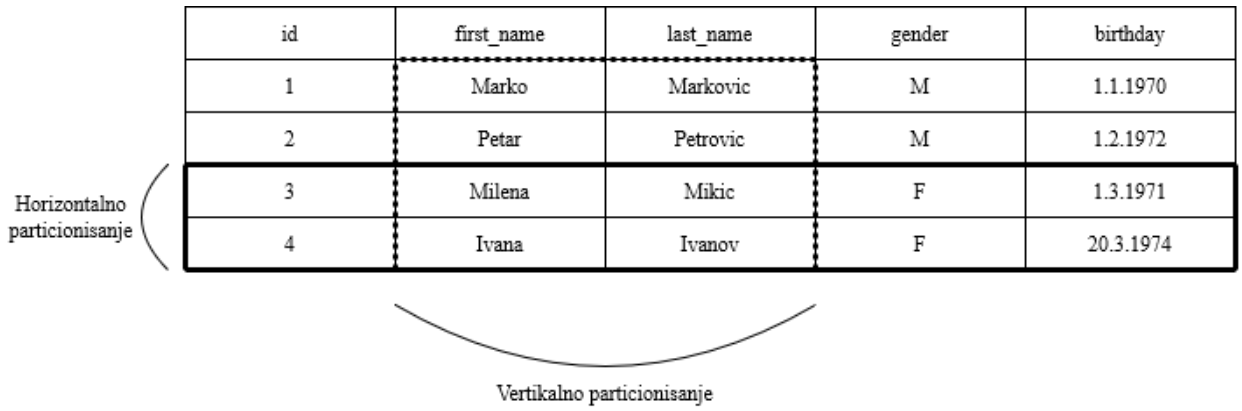
Kako bi se obezbedila konzistentnost podataka podaci u distribuiranim bazama podataka mogu da budu particionisani (eng. *partitioned*), ponovljeni (eng. *replicated*) ili i particionisani i ponovljeni. Kod particionisanja baza je podeljena na više delova koji se nalaze na različitim čvorovima. Skup podataka se deli na delove (fragmente) koji se raspoređuju po čvorovima. Kod replikacije podaci da imaju veći broj kopija na različitim lokacijama. Podaci mogu da budu potpuno ponovljeni gde svaki čvor sadrži sve podatke i delimično ponovljeni.

2.1 Particionisanje podataka

Kao što je već rečeno, podaci mogu biti replicirani gde postoji veći broj kopija podataka na različitim čvorovima ili particionisani što predstavlja deljenje podataka na određeni način i smeštanje na više čvorova. Replikacija podataka omogućava povećanje raspoloživosti podataka i efikasnost pristupa, ali otežava njihovo ažuriranje. Postoji horizontalno i vertikalno particionisanje [3].

Horizontalno particionisanje vrši podelu podataka po čvorovima, gde svaki red sadrži sve informacije. Podaci se grupišu po jednoj ili više koloni. Horizontalno particionisanje mora da bude odrađeno tako da se nakon spajanja fragmenata rekonstruiši svi podaci. Horizontalna fragmentacija može da bude primarna i izvedena. Kod primarne fragmentacije podela se vrši na osnovu jedne ili više kolona, dok se kod izvedene koristi način podele druge tabele koje se često zajedno koriste u upitima.

Vertikalno particionisanje grupiše kolone tabele u fragmente. Kao i kod horizontalnog particionisanja, spajanjem fragmenata mora se rekonstruisati baza podataka. Kako bi se izvršila rekonstrukcija svaki fragment mora da sadrži identifikatore podataka.



Slika 2.1 Horizontalno i vertikalno particionisanje

Hibridno particionisanje predstavlja kombinaciju horizontalnog i vertikalno particionisanja. Postoje dva pristupa. U prvom se prvo radi horizontalno particionisanje a nakon toga vertikalno, a u drugom pristupu je obrnuto. U oba slučaja rezultati su isti ali ovaj način particionisanja je najskuplji što se tiče rekonstrukcije podataka.

Najčešće se vrši particionisanje podataka na delove kojima upravlja replika set. Replika set je zadužena za upravljanje određenim delom podataka. Prilikom izvršenja upita, koordinator upita rutira upit do replika seta koji je zadužen za upis i čitanje tog dela podataka. Ovaj način particionisanja se zove **šarding** (eng. *sharding*). Šard (eng. *shard*) predstavlja horizontalnu particiju podataka. Šardovi se raspodeljuju čvorovima kako bi se smanjilo opterećenje. Svaki replika set predstavlja jedinstveni izvor za određeni skup podataka. Kako bi se šarding efektivno iskoristio kod dela podataka gde se vrši dosta operacija čitanja i upisa on se deli na manje particije kako bi se distribuiralo opterećenje. Prilikom dodavanja ili izbacivanja čvora iz klaster baze, potrebno je ponovo odraditi particionisanje kako bi se održao balans. Neke baze podataka automatski rade ponovno particionisanje i za to koriste određene algoritme kako bi se optimizovao taj proces. Kako bi se izvršilo rutiranje do pravog replika seta koji sadrži podatke moguće je raditi heš funkciju ključeva pomoću kojeg koji se kasnije uz određene operacije dobije identifikator čvora na kome su podaci.

2.2 Replikacija

Replikacija predstavlja postupak kopiranja i distribuiranja podataka iz jedne baze podataka u drugu koji sadrži mehanizam za sinhronizaciju. Ovo se koristi u slučaju kada je potrebna povećana dostupnost podataka i brzi pristup. Kreiranje većeg broja kopija smanjuje vreme pretrage ali povećava kompleksnost prilikom ažuriranja svih kopija. Zbog toga se replikacija bira u bazama gde se više vrše operacije pretrage podataka nego modifikacija podataka. Pored toga, ako deo sistema padne moguće je da u ostatku sistema postoji kopija podataka. Još jedna cena replikacije je i dodatan prostor za podatke [3].

2.3 Distribuirane transakcije

Transakcije u bazama podataka predstavljaju logičku jedinicu posla koji se izvršava nad podacima u okviru sistema za upravljanje bazama podataka. Sastoje se od niza operacija koje se izvršavaju nad podacima. Transakcija nad bazom podataka mora da bude atomična (eng. *atomicity*), konzistentna (eng. *consistency*), izolovana (eng. *isolated*) i trajna (eng. *durable*), drugim rečima transakcija treba da ima ACID svojstva. Distribuirana transakcija se izvršava nad podacima koji se nalaze na više različitih čvorova koji imaju različite podatke. Problem koji je potrebno rešiti kod distribuiranih transakcija je upravljanje transakcijom koja se izvršava na više particija na različitim čvorovima, kao i potvrđivanje i vraćanje na prethodno stanje. Drugim rečima potrebno je postići da transakcije budu atomične i da vrše prevođenje sistema iz jednog u drugo stanje.. Upravljanje distribuiranim transakcijama rade menadžer i koordinator transakcija [3].

Menadžer transakcija kontroliše čitavu transakciju, vodi evidenciju o podacima i koordinira transakcijom na jednom ili više čvorova. Postoji lokalni menadžer transakcija koji se koristi kada je samo jedan izvor podataka i globalni koji se koristi kada ima više izvora.

Koordinator transakcije distribuira transakciju na odgovarajućim čvorovima i vrši koordinaciju na svakom čvoru. Kao rezultat transakcija može da se izvrši i da se modifikuju podaci ili da se podaci vrate na prvobitno stanje.

Kako bi operacije bile atomične koriste se algoritmi atomičnog potvrđivanja (eng. *atomic commitment algorithms*) kao što su dvofazni protokol potvrđivanja i trofazni protokol potvrđivanja.

Dvofazni protokol potvrđivanja sastoji se od dva koraka: pripreme i potvrđivanja ili vraćanja na prethodno stanje. U prvoj fazi koordinator šalje svima transakciju koju je potrebno izvršiti. Svaki čvor odgovara da li može da izvrši transakciju sa pozitivnim ili negativnim odgovorom koordinatoru. Sve odluke koordinator upisuje u log, a čvorovi je čuvaju lokalno. Ukoliko su svi čvorovi odgovorili pozitivno koordinator im šalje poruku za potvrđivanje, a ako je neko odgovorio negativno koordinator šalje poruku za vraćanje na prethodno stanje.

2.4 Detekcija greške

Kako bi sistem bio otporan na greške potrebna je pravovremena detekcija greške. Kako se distribuirane baze podataka izvršavaju na više čvorova postoji veći broj grešaka koje mogu da nastupe nego u centralizovanim. One obuhvataju sledeće:

- Otkaz čvora
- Otkaz mreže
- Gubitak poruke između čvorova

Distribuirana baza podataka mora da bude otporna na greške što zahteva da nastavi sa normalnim radom i očuva konzistentno stanje. Obično za detekciju grešaka postoji poseban deo sistema koji se bavi time. Jedan od načina za implementaciju detekcije greške jeste korišćenjem heartbeat i ping poruka.

Kako bi se proverilo stanje udaljenog čvora koriste se ove poruke koje se periodično šalju:

- Ping poruka – poruka koja se šalje udaljenom procesu kako bi se proverilo da li još uvek radi tako što se očekuje njegov odgovor u određenom intervalu
- Heartbeat poruka – proces šalje poruke drugim čvorovima kako bi ih obavestio da je aktivan

U oba slučaja svaki čvor ima listu svih ostalih čvorova sa njihovim stanjima i na osnovu poruke se održavanja njihovo stanje. Stanje čvora može da bude: živ, mrtav i sumnjiv. U slučaju da proces ne odgovara na ping poruke ili ne šalje heartbeat poruke njegovo stanje postaje sumnjiv nakon čega se pokreće odgovarajuća akcija.

2.5 Sistemske informacije

Baza podataka čuva određene metapodatke kao što su podaci o korisnicima, indeksima, tabelama, itd. Ove informacije u distribuiranim bazama podataka mogu da budu:

- Centralizovani – nalaze se samo na jednom čvoru
- Potpuno replicirani – postoje na svim čvorovima
- Particionisani – svaki čvor ima deo podataka koji se odnosi na podatke iz tog čvora
- Hibridni – koristi se particionisanje ali postoji i kopija koja sadrži podatke vezane za sve čvorove.

U zavisnosti od potreba treba izabrati način koji najviše odgovara.

2.6 Sinhronizacija podataka

Jedna od metoda koja se koristi za sinhronizaciju podataka u distribuiranim bazama je metoda logova. Log predstavlja datoteku u kojoj se čuvaju operacija ažuriranja nad podacima kao i rezultati tih operacija. Ovi podaci se koriste prilikom oporavka od greške kako bi se očuvalo konzistentno stanje baze. Pored primarne uloge logovi mogu da se koriste i za sinhronizaciju podataka u distribuiranim bazama.

Sinhronizacija može da bude sinhrona i asinhrona. Kod sinhrone ažuriranje svih kopija se vršu u okviru jedne transakcije koja ažurira i primarnu kopiju. Kod asinhrone se ažuriranje kopija vrši iz više koraka što može da dovede do konflikta u podacima.

Korišćenje logova zahteva komunikaciju sa ostalim čvorovima što može puno da košta što se tiče mrežnih resursa. Zbog toga neki poslovi u distribuiranim bazama pa i sinhronizacija mogu da izaberu jedan čvor koji treba da bude glavni i da izvrši koordinaciju poslova. Svi čvorovi su jednaki pa tako svaki može da preuzme ulogu vođe. Kada se jednom izabere vođa najčešće on ostaje sve dok ne dođe do greške zbog toga što i sam izbor vođe košta.

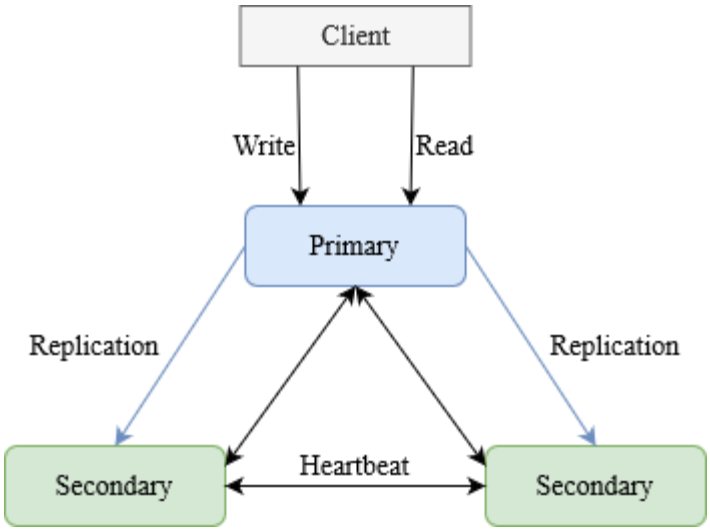
3. MongoDB

Mongo baza podataka predstavlja jednu od vodećih NoSQL baza podataka. Za razliku od relacionih baza podataka gde su podaci struktuirani, podaci koji se skladište u Mongo bazi predstavljeni su polustruktuiranim dokumentima. Za skladištenje podataka koristi se struktura slična JSON-u. Jedan podatak koji se skladišti u bazu predstavlja dokument koji se sastoji od parova ključ-vrednost. Vrednosti mogu da uključuje druge dokumente, nizove i nizove dokumenata. Prednosti korišćenja dokumenata su: dokumenti se koriste kao prirodni tipovi podataka kod mnogih programskih jezika, ugnježđeni dokumenti i nizovi smanjuju potrebu za spajanje više tabela i imaju dinamičku šemu [4]. Mongo smešta dokumente u kolekcije koje se mogu porediti sa tabelama kod relacionih baza. Glavne karakteristike Mongo baze su:

- Performanse
- Bogat jezik za upite nad bazom – podržava upite za upis i čitanje kao i za agregaciju podataka, pretragu teksta i geospacijalne upite
- Visoka dostupnost – omogućava replikaciju podataka na više servera i automatski oporavak od greške
- Horizontalna skalabilnost – koristi šarding za distribuiranje podataka na više mašina i vrši rutiranje upita na određeni server
- Izbor storage engine-a – omogućava laku zamenu storage engin-a

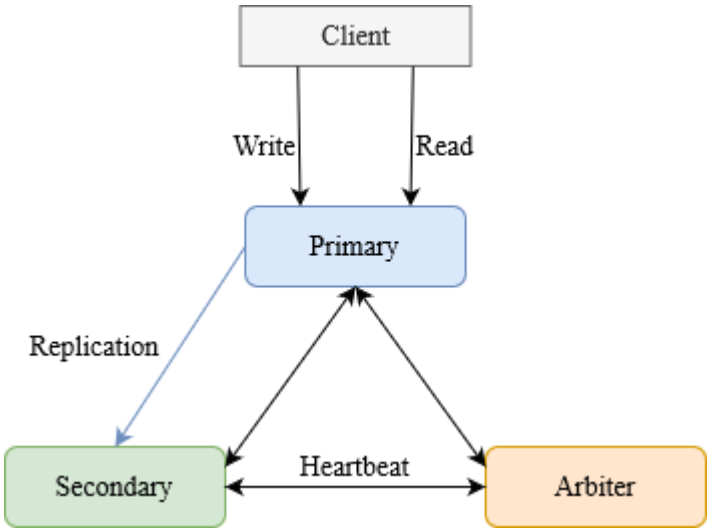
3.1 Replikacija

Replika set predstavlja grupu Mongo servera koji sadrže isti skup podataka. Koriste se kako bi pružile redundantnost podataka i visoku dostupnost. Replika se sastoji od nekoliko čvorova koji sadrže podatke. Od tih čvorova samo jedan je primarni čvor dok su ostali sekundarni. Sve operacije upisa prihvata primarni čvor. Nakon što primarni čvor dobije zahtev za upis podataka, promene nad podacima beleži u dnevniku operacija – oplog. Sekundarni čvorovi repliciraju dnevnik operacija primarnog čvora i rade nastale promene kako bi imale isti skup podataka kao i primarni čvor. Ukoliko primarni čvor nije dostupan, na osnovu sekundarnih čvorova se radi selekcija novog primarnog čvora. Podrazumevano vreme bez komunikacije primarnog čvora nakon kog sekundarni čvor pokreće izbor primarnog je 10 sekundi što se može promeniti. Kada se završi izbor novog čvora proces replikacije se nastavlja. Replika set može da odgovara na upite za čitanje podataka ukoliko je postavljeno da je moguće čitanje sa sekundarnih čvorova [4].



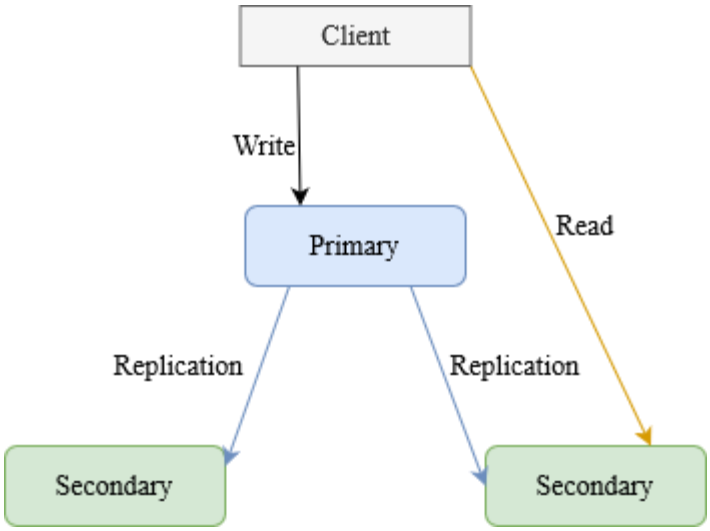
Slika 3.1 MongoDB replikacija

U određenim uslovima kada nije moguće koristiti čvor kao sekundarni (primer zbog troškova) moguće je izabrati da čvor bude arbiter. Takav čvor učestvuje u izboru čvorova ali ne sadrži podatke.



Slika 3.2 MongoDB arbiter

Podrazumevano čitanje podataka se radi od primarnog čvora, ali klijent može da promeni podešavanja čitanja tako da dozvoli i čitanje sa sekundarnog čvora. Kako se replikacija podataka od primarnog čvora do sekundarnog vrši asinhrono može doći čitanja podataka koji nisu isti kao na primarnom čvoru. Kod transakcija se zbog toga obavezno za čitanje koristi primarni čvor.



Slika 3.3 Čitanje sa sekundarnog čvora

Od verzije 4.0 u Mongo bazu uvedene su multi-dokument transakcije. Kod ovih transakcija operacije čitanja moraju da čitaju podatke sa primarnog čvora. Pre nego što transakcija bude potvrđena, promene nad podacima nisu vidljive izvan konteksta transakcije. Međutim ne moraju sve operacije čitanja da čekaju da potvrđivanje transakcije bude vidljivo u svim šardovima.

3.2 Šarding

Šarding predstavlja način za distribuiranje podataka na više mašina. Mongo koristi šarding kod sistema sa velikom količinom podataka i velikog broja operacija nad podacima. Baze podataka koje skladište velike količine podataka mogu da imaju problema sa performansama ukoliko rade na jednom serveru. Primer je veliki broj upita koji zahtevaju CPU vreme, kao i veliki set podataka koji vraća neki upit koji je potrebno smestiti u RAM memoriju mašine. Kako bi se omogućio rad u ovakvim uslovima potrebno je skalirati resurse i to može biti veritkalno i horizontalno [4].

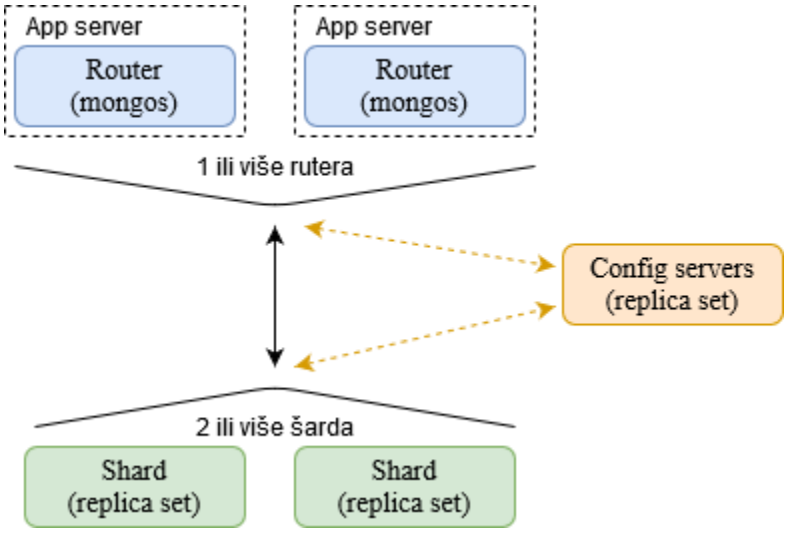
Vertikalno skaliranje uključuje povećanje resursa jednog servera, kao što jači CPU, dodavanje RAM memorije, dodavanje storage-a. Postoji određeni limit do kojeg je moguće vertikalno skalirati jednu mašinu koji je ograničen današnjom tehnologijom. Pored toga cena dodavanja jači resursa je sve veća u zavisnosti od njihovih karakteristika.

Horizontalno skaliranje omogućava dodavanje dodatnog servera koji se koristi za obradu. Jedan server može da ne bude dovoljno jak za obradu celokupnih podataka pa se zbog toga posao deli na više delova gde svaki deo vrši obradu jednog dela podataka. Prednost horizontalnog skaliranja je i to što je moguće dodavati dodatne servere samo po potrebi i to je jeftinije nego dodavanje jačih resursa na jednu mašinu. Mana ovoga je što je infrastruktura kompleksnija.

Mongo baza podataka podržava horizontalno skaliranje preko šardinga.

MongoDB klaster šardova se sastoji od:

- Shard – svaki šard sadrži deo šardovanih podataka. Svaki šard može da se postavi kao replika set.
- Mongos – predstavlja ruter upita. Pruža interfejs između klijenata i klastera šardova.
- Config servers – sadrže metadata i konfiguracione podatke. Postavljaju se kao replika set.



Slika 3.4 Mongo klaster

Mongo koristi šard ključeve kako bi distribuirala podatke iz kolekcije. Šard ključ se sastoji od polja koje svaki dokument iz kolekcije sadrži. Šard ključ može da bira korisnik i može da bude samojedan. Nakon izbora šard ključa nije moguće promeniti ga. Od izbora šard ključa zavise performanse, efikasnost i skalabilnost šardovanog klastera. Šardovi su podeljeni u čankove (eng. *chunk*). Svaki čank ima definisan opseg na osnovu šarding ključa. Kako bi se postigla balansirana distribucija čankova po šardovima, balaser radi u pozadini migraciju čankova po šardovima.

Prednosti šardinga su sledeće:

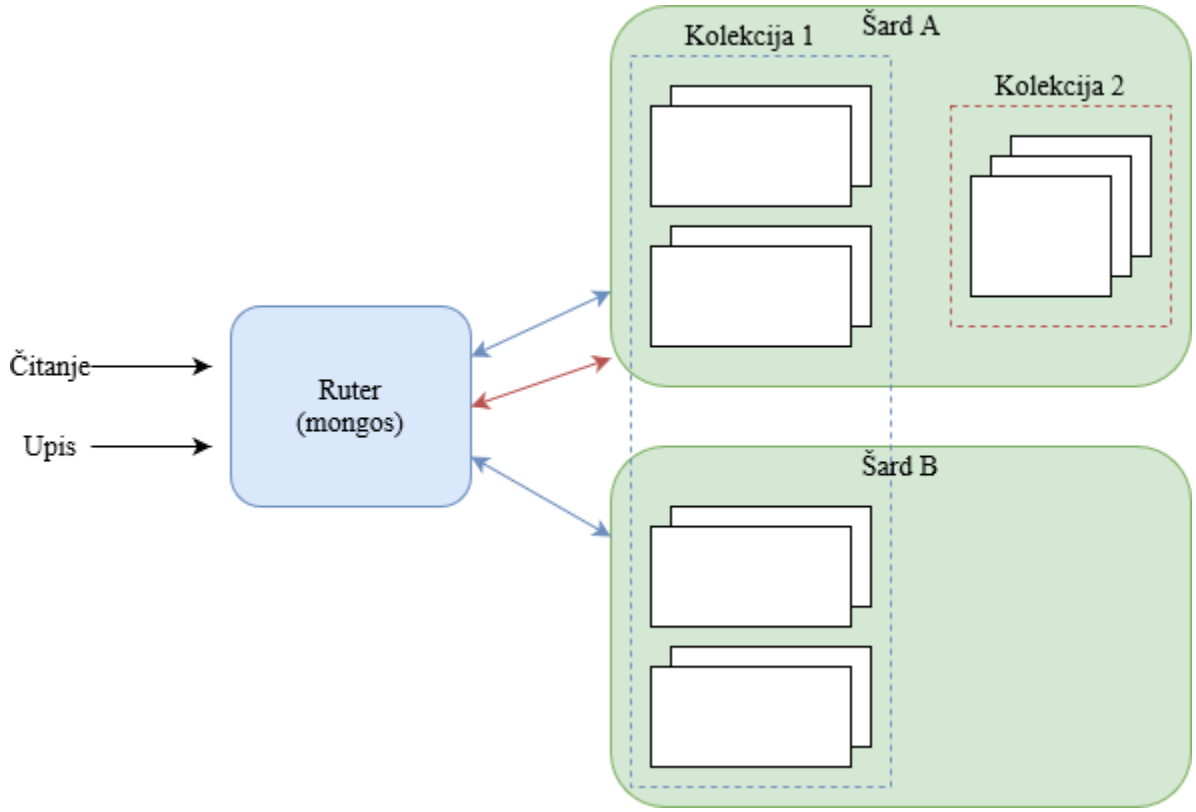
- Čitanje/upis – Mongo distribuira operacije čitanja i upisa po šardovima i time se omogućava da svaki šard radi deo operacija. Dodavanjem novih šardova postiže se horizontalna skalabilnost operacija. Na osnovu šarding ključa operacije se šalju samo određenim šardovima.
- Kapacitet prostora – Šarding distribuira deo podatke po šardovima u klaasteru.
- Visoka dostupnost – Šardovan klaster može da izvršava parcijalni upis i čitanje čak iako više šardova nije dostupno. Delom podataka koji nije dostupan nije moguće izvršavati operacije, ali na deo podataka koje je dostupan je moguće raditi upis i čitanje. U kombinaciji sa replika setovima dostupnost se povećava.

Šarding zahteva dobro planiranje, održavanje i izbor šarding ključa. Nakon izbora šarding ključa njega nije moguće promeniti, isto tako nakon što se odradi šarding kolekcije nje nije moguće spojiti. Takođe postoje i neke operacije koje nije moguće izvršiti kada se koristi šarding. Ako upit ne sadrži šarding ključ onda se radi broadcast operacije i pretraživanje svih šardova.

Postoje 2 moda za šarding:

- Uz pomoć heš funkcije – izračunava se heš vrednost polja i na osnovu te vrednosti se smešta u određeni čank
- Na osnovu opsega – čankovi se dele na osnovu opsega vrednosti šarding ključa

Mongo može da ima u klasteru kolekcije koje su šardovane i koje nisu. Kolekcije koje nisu šardovane se skladište na primarnom šardu. Za operacije sa kolekcijama potrebno je povezati se na ruter a ne direktno na šard.



Slika 3.5 Šardovane i nešardovane kolekcije

3.3 Transakcije

U situacijama kada operacije čitanja i upisa moraju da budu atomične nad više dokumenata u jednoj ili više kolekcija korisite se multi-dokument transakcije. Multi-dokument transakcije su atomične i omogućavaju da se izvrše ili sve operacije ili nijedna. Kada se transakcije potvrdi

promene nastale u njoj su vidljive izvan nje. Kada transakcija vrši upis u više šardova u zavisnosti od klijentove konfiguracije čitanja (u slučaju da je ‘local’) može da vidi upisane podatke iako transakcija nije potvrđena. Kada se transakcija odbaci ili dođe do greške prilikom izvršenja sve promene u okviru nje se odbace. Transakcije su vezane za sesiju i moguće je imati samo jednu otvorenu transakciju za sesiju. Ako se sesija zatvori a postoji otvorena transakcija u njoj onda se ona odbacuje. Operacije čitanja u okviru multi-dokument transakcije moraju da budu konfigurisane tako da koristi ‘primary’ [4].

Nivo čitanja u okviru transakcije

Postoje sledeći nivoi čitanja:

- local – vraća najnovije podatke na čvoru ali oni mogu biti vraćeni na prethodno stanje.
- majority – vraća podatak koji je potvrđen od strane većine čvorova.
- snapshot – vraća podatke iz snepšota kreiranog od podatka koji je potvrdila većina čvorova

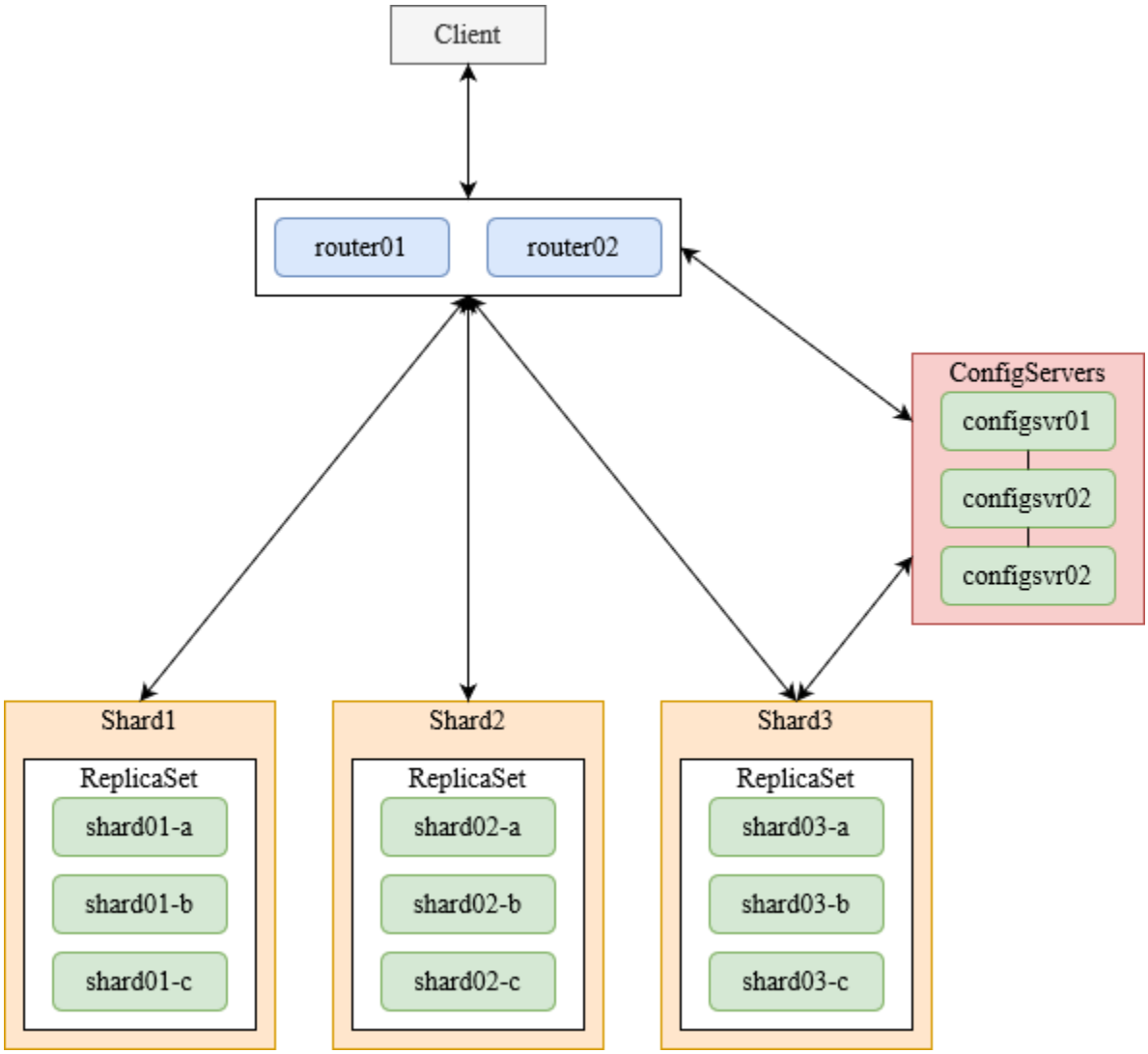
Nivo upisa u okviru transakcije

Kako bi se operacije smatrale potvrđenim transakcije koriste određeni nivo za upis:

- 1 – potvrđuje se upis nakon što se izvrši upis na primarnom čvoru. Ukoliko se dogodi da otkaže primarni čvor moguće je izgubiti promene.
- majority – potvrđuje se upis nakon što su podaci upisani na primarnom čvoru i na većini čvorova – 1.

4. Primer kreiranja MongoDB klastera

MongoDB klaster podignut je korišćenjem Docker-a. Na jednom hostu podignuti su sledeći čvorovi mongo klastera:



Slika 4.1 Čvorovi MongoDB klastera

Prilikom pokretanja čvorova svaki Mongo čvor se pokreće sa određenom ulogom. Ruter čvorovi pokreću Mongos proces koji je zadužen za šarding podataka i rutiranje upita. Ostali čvorovi pokreću primarni Mongod proces koji je zadužen za upravljanje podacima. Kod tih čvorova potrebno je proslediti kao fleg ulogu čvora (--configsvr za konfig servere ili –shardsvr šardove). Kako jedan ReplicaSet ima više čvorora potrebno je navesti fleg sa imenom replika set (--repliSet <ime>). Nakon startovanja svih čvorova potrebno je izvršiti komande za dodavanje u klaster. Komanda za inicijalizaciju konfig servera:

```
rs.initiate(
  {
     _id: "rs-config-server",
     configsvr: true,
     version: 1,
     members: [
        { _id: 0, host : "configsvr01:27017" },
        { _id: 1, host : "configsvr02:27017" },
        { _id: 2, host : "configsvr03:27017" }
     ]
  }
)
```

Ova komanda se izvršava na jednom od čvorova koji spadaju u servere konfiguracije. U members nizu se navode hostovi koji čine replika set, kako se instance izvršavaju u Docker mreži navodi se hostname u okviru docker mreže.

Nakon inicijalizacije servera konfiguracije potrebno je izvršiti inicijalizaciju šardova. Komanda za inicijalizacijzu šardova je:

```
rs.initiate(
  {
     _id: "rs-shard-01",
     version: 1,
     members: [
        { _id: 0, host : "shard01-a:27017" },
        { _id: 1, host : "shard01-b:27017" },
        { _id: 2, host : "shard01-c:27017" },
     ]
  }
)
```

Ova komanda se izvršava na jednom od servera koji čine šard. Za svaki šard potrebo je izršiti ovu komandu gde se ime šard i njegovi članovi.

Nakon inicijalizacije šardova radi se inicijalizacija rutera. Komanda za inicijalizaciju rutera je:

```
sh.addShard("rs-shard-01/shard01-a:27017")
sh.addShard("rs-shard-01/shard01-b:27017")
sh.addShard("rs-shard-01/shard01-c:27017")

sh.addShard("rs-shard-02/shard02-a:27017")
sh.addShard("rs-shard-02/shard02-b:27017")
sh.addShard("rs-shard-02/shard02-c:27017")

sh.addShard("rs-shard-03/shard03-a:27017")
sh.addShard("rs-shard-03/shard03-b:27017")
sh.addShard("rs-shard-03/shard03-c:27017")
```

Ova komanda se izvršava na jednom ruteru. Nakon izvršenja komandi za inicijalizaciju imamo spreman MongoDB klaster. Iako Mongo radi u klaster modu, šardovanje kolekcije se radi eksplicitno i tada se bira šarding ključ. Potrebno je napraviti kolekciju i dodati podatke. Nakon toga dozvoljava se šardovanje te kolekcije i nakon dodavanja novih podataka Mongo automatski balansira raspodeljenost podataka.

Kao primer klijenta za dodavanje i čitanje podataka kreirane su 2 NodeJS aplikacije. Kod aplikacije za dodavanje podataka:

```
const mongoose = require('mongoose');
const faker = require('faker');
const cliProgress = require('cli-progress');
const { User } = require('./user');

const bar = new cliProgress.SingleBar({}, cliProgress.Presets.shades_classic);
const toInsert = 100000;
bar.start(toInsert, 0);

function start() {
  mongoose.connect(`mongodb://router01:27017,router02:27017/testingDb`, {
useNewUrlParser: true });

  const db = mongoose.connection;

  db.createCollection('users');

  db.on('error', console.error.bind(console, 'connection error:'));
  db.once('open', async () => {
    for (let i = 0; i < toInsert; i++) {
      const test = new User({ name: faker.name.findName(), about:
faker.lorem.sentences() });
      await test.save();
      bar.update(i+1);
    }

    await db.collection('users').ensureIndex({ name: 'hashed' });
    console.log('done');
    bar.stop();

    process.exit();
  });
}

start();
```

Klijentska aplikacija za dodavanje podataka se takođe izvršava u okviru Docker-a i u istoj mreži kao i ostali čvorovi klastera pa zbog toga zna za hostove router01 i router02. Kreira se konekcija ka ruterima nakon toga se kreira kolekcija **users** u bazi **testingDb**. Šema kolekcije **users** se kreira na sledeći način:

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
  about: String,
});

const User = mongoose.model('users', userSchema);

module.exports = { User, userSchema }
```

Nakon dodavanja 100000 podataka u kolekciju kreira se index za **name** tipa heš koji se kasnije koristi za šardovanje. Kako bi se dozvolilo šardovanje kolekcije potrebno je izvršiti sledeće komande:

```
sh.enableSharding('<ime_baze>')

db.adminCommand( { shardCollection: '<ime_baze>.<ime_kolekcije>', key: {
<ime_propertija>: 'hashed' } } )
```

Nakon izvršenja komanda omogućeno je šardovanje za kolekciju. Dodavanjem novih podataka vrši se upis u različite šardove. Status distribucije podataka po šardovimaa moguće je proveriti komandom:

```
use <ime_baze>;
db.<ime_kolekcije>.getShardDistribution();
```

Primer rezultati izvršenje komande za proveru stanja šardova dat je na slici 4.2.

```
Shard rs-shard-01 at rs-shard-01/shard01-a:27017,shard01-b:27017,shard01-c:27017
 data : 37.68MiB docs : 159706 chunks : 1
 estimated data per chunk : 37.68MiB
 estimated docs per chunk : 159706

Shard rs-shard-02 at rs-shard-02/shard02-a:27017,shard02-b:27017,shard02-c:27017
 data : 37.66MiB docs : 159798 chunks : 1
 estimated data per chunk : 37.66MiB
 estimated docs per chunk : 159798

Shard rs-shard-03 at rs-shard-03/shard03-a:27017,shard03-b:27017,shard03-c:27017
 data : 83.19MiB docs : 352815 chunks : 1
 estimated data per chunk : 83.19MiB
 estimated docs per chunk : 352815

Totals
 data : 158.54MiB docs : 672319 chunks : 3
 Shard rs-shard-01 contains 23.76% data, 23.75% docs in cluster, avg obj size on shard : 247B
 Shard rs-shard-02 contains 23.75% data, 23.76% docs in cluster, avg obj size on shard : 247B
 Shard rs-shard-03 contains 52.47% data, 52.47% docs in cluster, avg obj size on shard : 247B
```

Slika 4.2 Distribucija podataka po šardovima

Kao za upis podataka tako i za čitanje, klijent je potrebno da se konektuje na ruter. Ruter vrši rutiranje upita na određeni šard ukoliko upit sadrži ključ šardovanja. Kod jednostavnog klijenta za čitanje podataka je:

```
const mongoose = require('mongoose');
const { User } = require('./user');

function start() {
  mongoose.connect(`mongodb://router01:27017,router02:27017/testingDb`, {
useNewUrlParser: true });

  const db = mongoose.connection;

  db.on('error', console.error.bind(console, 'connection error:'));
  db.once('open', async () => {

    const users = await User.find({ name: { $regex: '.*Jarvis.*' } });
    users.forEach(user => {
      console.log(user.name);
    });

    process.exit();
  });
}

start();
```

Isto kao i klijent koji vrši upis podataka i ovaj NodeJS Mongo klijent se izvršava u Docker-u u okviru iste mreže gde su ostali čvorovi klastera.

5. Zaključak

Distribuirane baze podataka imaju svoje prednosti kao što su visoka dostupnost i mogućnost oporavka od greške. Problem kod ovih baza je način distribucije podataka. U zavisnosti od samih podataka koji se skladište u bazi zavise mogućnosti primene distribuirane baze podataka. Kao što je rečeno šarding podataka u distribuiranim bazama se radi na osnovu ključa koji se kasnije koristi za rutiranje zahteva na određeni čvor. Od izbora ključa zavise performanse baze. Pored samih podataka distribuirane baze podataka se izvršavaju na složenijoj infrastrukturi pa je samim tim i održavanje takve baze teže jer zahteva komunikaciju preko mreže. Distribuirane baze podataka podržavaju i izvršenje na jednom čvoru i distribuciju po potrebi. Veoma je bitno da se na osnovu analize zahteva i samih podataka odredi potreba za distribuiranom bazom podataka i načinom distribucije podataka.

LITERATURA

[1] *„Distributed database“* - https://en.wikipedia.org/wiki/Distributed_database, Poslednji pristup: 20.06.2020.

[2] *„CAP theorem“* - https://en.wikipedia.org/wiki/CAP_theorem, Poslednji pristup: 20.06.2020.

[3] *„Database internals: A deep-dive into how distributed data systems work“*, Alex Petrov, 2019

[4] *„MongoDB Manual“* - https://docs.mongodb.com/manual/, Poslednji pritup: 20.06.2020.