Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE33Colic

UPOREDNA ANALIZA PROCESA REPLIKACIJE KOD RELACIONIH I BAZA PODATAKA NOVE GENERACIJE

COMPARATIVE ANALYSIS OF THE REPLICATION PROCESS IN RELATIONAL DATABASES AND DATABASES OF THE NEW GENERATION

Stefan Colić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad opisuje problem procesa replikacije u softverskim sistemima sa analizom pojedinih replikacionih postavki. Fokus je stavljen na ispitivanje osobina proizvoda nove generacije baza podataka koja nastoji da pruži horizontalnu skalabilnost nerelacionih baza podataka sa visokim stepenom konzistentnosti koje pružaju relacione baze.*

Ključne reči: *PostgreSQL, CockroachDB, replikacija, konzistentnost, mikroservisi, CAP teorema.*

Abstract – *This paper describes the problem of replicating data analyzing several replication setups. Focus is placed on examing a product of the next generation of database systems that tends to provide horizontal scalability of non-relational databases and high level of consistency of relational databases.*

Keywords: *PostgreSQL, CockroachDB, replication, consistency, microservices, CAP theorem*

1. UVOD

Visoka dostupnost je važna karakteristika mnogih softverskih sistema. Nedostupnost, spor odziv ili pogrešno funkcionisanje, mogu imati negativan uticaj na pojedince i organizacije i zato je često bitno da svaka komponenta softvera bude otporna na različite vrste otkaza. Replikacija podataka je jedan od načina kako se mogu povećati dostupnost, pouzdanost i skalabilnost sistema. Jedan od razloga za repliciranje informacija jeste redundantnost, odnosno sprečavanje gubitka istih usled otkaza sistema za skladištenje podataka ili drugih hardverskih komponenti. Takođe benefit postojanja više kopija leži u poboljšavanju performansi usled većeg opterećenja. Korisnici mogu istovremeno pristupati različitim kopijama koje su im geografski bliže što smanjuje odziv i pruža bolju pokrivenost. Nedostatak koji dolazi sa procesom replikacije je zaostajanje prilikom propagacije promena između replika što dodatno uvodi problem konzistentnosti. Postoji opasnost čitanja zastarelih informacija što može dovesti do neželjenih aktivnosti. Ovaj rad analizira osobine replikacionih postavki *PostgreSQL* relacione baze i *CockroachDB* baze koja se klasifikuje pod nazivom *NewSQL*.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Branko Milosavljević, red. prof.

Glavni razlog posmatranja ove dve tehnologije jeste visok stepen konzistentnosti koje pružaju pri replikaciji i pristupu informacijama [1].

Postavke se posmatraju u kontekstu sistema koji pruža različite oblike plaćanja pri čemu se simulira korišćenje od strane određenog broja korisnika. U tom procesu, praćeno je ponašanje baza podataka sa ciljem dostizanja zaključka o upotrebljivosti oba sistema u sličnim okolnostima.

2. TIPOVI REPLIKACIJE

Svaki čvor u replikacionom procesu može imati ulogu vođe (eng. *Leader*), pratioca (*eng. Follower*) ili ulogu koja predstavlja varijaciju ova dve [2]. Lider je čvor koji prihvata upise i radi širenje promena ka ostalim čvorovima pri čemu uglavnom omogućuje i operacije čitanja, dok pratioci pružaju mogućnosti samo čitanja i prihvataju promene poslate od strane lidera [3]. Postoji više tipova replikacije u zavisnosti od prisutnih uloga i odnosa između čvorova pri čemu razlikujemo [2]:

1. *Single-Leader* replikaciju – Postoji samo jedan lider i može postojati veći broj pratilaca.
2. *Multi-Leader* replikaciju – Za razliku od prethodnog tipa, može postojati više lidera, gde svaki lider može predstavljati pratioca drugog lidera.
3. *Leaderless* replikaciju – Fundamentalno se razlikuje od prethodna dva tipa, jer svi čvorovi imaju istu ulogu, odnosno ne postoji eksplicitni čvor koji je deklarisan kao vođa.

Replikaciona postavka sa jednim liderom ima benefit lakšeg razumevanja i pruža podlogu za visok nivo konzistentnosti zahvaljujući *ACID* svojstvima. Nedostatak jeste u ograničenim mogućnostima jednog čvora koji prihvata upise. Te nedostatke otklanja uvođenje većeg broja lidera za opterećenje (eng. *load*) koje se sastoji iz prilibližnog ili većeg broja upisa u odnosu na operacije čitanja. Negativna strana takvog pristupa jeste kreiranje podloge za konflikte koje je neophodno razrešiti na aplikativnom nivou. Oba načina mogu širenje podataka ka čvorovima rešiti asinhrono, sinhrono ili hibridno. Jedan vid hibridnog pristupa jeste *Leaderless* replikacija gde svi čvorovi mogu prihvatati upise, pri čemu se upis ne smatra uspešnim dok se ne postigne kvorum (eng. *quorum*), odnosno dok određen broj (najčešće većina) ne potvrdi da je replikacija uspešno izvršena.

Varijacija pristupa sa jednim liderom jeste replikacija koja koristi konsenzus algoritam. Ona se oslanja na koncept kvoruma pri čemu otklanja mogućnost nastajanja konflikata i rešava se problem izbora novog vođe u slučaju otkaza prethodnog bez negativnog uticaja na konzistentnost podataka [4]. Najpoznatiji algoritmi ove prirode su *Paxos* i *Raft*.

U radu posmatrane postavke su potpuno sinhrona replikacija kod *PostgreSQL* relacione baze koja se oslanja na prenos sadržaja *Write-Ahead Log*-a putem otvorenog toka (eng. *stream*) ka svakom pratiocu i *CockroachDB* replikacije koja je hibridan pristup svih opisanih tipova. Oslanja se na *Raft* konsenzus protokol i nadograđuje ga konceptom kao što je *Multi-Raft*, odnosno postoje više pojedinačnih *Raft* grupa, što ovoj postavci omogućuje da prihvata upise na više čvorova [5]. Takođe uvode apstrakciju zakupa (eng. *lease*) pri čemu se pružaju konzistentna čitanja sa zaobilaženjem *Raft*-a [6].

3. ARHITEKTURA SISTEMA

Sistem koji se opisuje u ovom radu (koncentrator plaćanja) sastoji se iz većeg broja komponenti koje su pokretane na više identičnih računara u lokalnoj mreži. Arhitektura je prikazana na slici 1.

Slika 1. *Arhitektura sistema koji je korišćen za ispitivanje osobina replikacije PostgreSQL-a i CockroachDB-a*

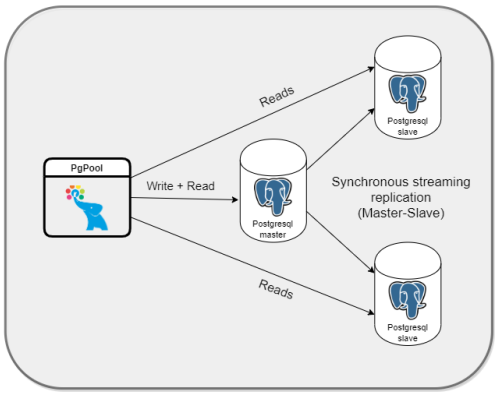
Ulaznu tačku u sistemu čine više instanci *Nginx*-a koji uzimaju ulogu *API Gateway*-a. Njihov zadatak je da *HTTP* zahteve proslede do instanci odgovarajućih mikroservisa na osnovu putanje zahteva. Pre nego što konkretan zahtev prosledi na adekvatno mesto, vrši se autentifikacija i potpomaže se u autorizaciji njegovim prosleđivanjem autentifikacionom servisu koji čita sadržaj *JSON Web Token*-a i pruža odgovor nazad instancama *Nginx*-a koje odlučuju da li zahtev može biti prosleđen dalje. Ovaj mikroservis je realizovan upotrebom *Spring Boot* radnog okvira i zbog svoje učestale upotrebe (pri slanju svakog zahteva), podatke čuva u *H2* bazi koja je smeštena u njegovoj memoriji.

Nakon uspešne autentifikacije, zahtev se prosleđuje nekoj od dostupnih instanci konkretno traženog mikroservisa po kružnom (eng. *round-robin*) principu. Svaki mikroservis se samostalno registruje u registar instanci, *Consul*, koji čuva podatke o svakom i replicira ih i duž klastera koji čine tri instance, upotrebom *Raft* konsenzus protkola. Na osnovu tih podataka, proces pod nazivom *Consul-Template*, osmatra potencijalne promene i vrši ažuriranje konfiguracionog fajla *Nginx*-a pri svakoj novoj registraciji ili uklanjanju servisa iz registra.

Osnovna funkcija ovog sistema je da posreduje između klijenata i različitih servisa za plaćanje. Spram toga, podržano je plaćanje platnom karticom, putem *PayPal*-a i putem *Bitcoin* kriptovalute. Svaki oblik plaćanja je realizovan kao zaseban mikroservis (pokrenut u dve instance) koji prihvata zahteve od korisnika, obrađuje ih, šalje eksternim serivisma i čuva podatke u lokalnoj bazi. U testovima koji su sprovođeni, kreiran je veliki broj zahteva, zato su pozivi eksternim servisima mokovani predefinisanim odgovorima.

Mikroservisi se obraćaju sloju za skladištenje podataka, pri čemu se instance različitih servisa mogu obraćati istoj fizičkoj mašini. S obzirom da se osmatraju osobine baza podataka pod velikim opterećenjem, nije neophodno napraviti separaciju, koja bi u produkciji bila poželjna.

Jedan od razloga upotrebe prethodno opisanih baza podataka jeste u korišćenju istog *PostgreSQL wire protocol*-a, što svodi promenu upotrebljivane baze na promenu konekcionog parametra u konfiguraciji pojedinačnih mikroservisa. Replikaciona postavka u kontekstu *PostgreSQL*-a (prikazan na slici 2) zahteva upotrebu međusloja koji će vršiti rutiranje upita u zavisnosti od toga da li se upisuju ili čitaju podaci sa diska. Upisi su isključivo usmereni ka vođi, dok se čitanja mogu usmeravati na sve tri prikazane replike.

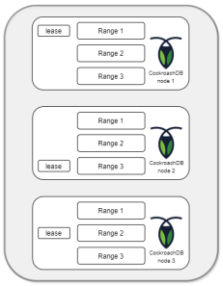


Slika 2. *PostgreSQL streaming replikacija sa upotrebom međusloja u vidu PgPool-II-a*

CockroachDB se klasifikuje u kategoriju *NewSQL* baza koje površinski izgledaju kao relacione baze, zbog svog *SQL* sloja. Na najnižem nivou, koristi se *Key-Value* skladište pri čemu je sadržaj particionisan na opsege (eng. *Range*) koji su raspoređeni po različitim replikama u zavisnosti od replikacionog faktora (primer na slici 3). Svaki opseg predstavlja prethodno opisanu *Raft* grupu. Takođe, svaki čvor u klasteru ujedno igra i ulogu *gateway*-a, odnosno za slučaj da ne sadrži *lease* za određen opseg, zahtev se prosleđuje adekvatnoj replici. Ovakav pristup ne zahteva upotrebu dodatne komponente za balansiranje zahteva s obzirom na postojanje parametra u *JDBC Api*-u koji omogućuje *HikariCP*-u da otvara i održava ravnomeran broj konekcija ka svim čvorovima.

Zahtevi su generisani distribuirano upotrebom *locust* alata, pri čemu su simulirani korisnici koji koriste različite metode plaćanja. U slučaju plaćanja platnom karticom i kriptovalutom, proces obuhvata: proveru podržanosti platne metode, generisanje tokena za plaćanje, plaćanje, nasumična promena statusa transakcije, prikaz detalja i prikaz poslednjih dvadeset izvršenih transakcija.

Mikroservis za podršku plaćanja putem *PayPal*-a dodatno pored navedenih opcija pruža identičan tok za opciju pretplate.



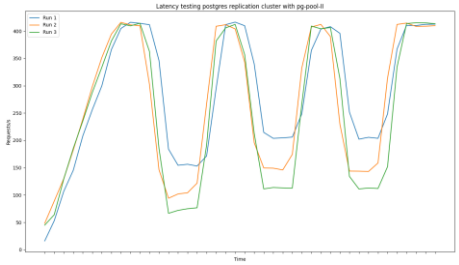
Slika 3. *CockroachDB čvorovi sa 3 opsega*

Sve navedene komponente su pokretane u *Docker* kontejnerima na sedamnaest računara identične konfiguracije u *Swarm Mode*-u. Na svim računarima dodatno su pokretani kontejneri (*CAdvisor, Node-Exporter, Grok-Exporter*) koji omogućuju prikupljanje metrika sa svakog pojedinačnog računara koje su čuvane u *Prometheus Time-Series* bazi. Dodatno je korišćena *Grafana* za elegantan prikaz prikupljenih podataka.

4. REZULTATI I TUMAČENJA

Svaki test pokretan je u istom okruženju sa istim rasporedom na računarima, pri čemu su pojedini testovi pokretani više puta radi preciznijeg prikupljanja informacija. Metrika koja je pritom najviše opažana je broj zahteva po sekundi (*Requests/s)*.

Test povećanog vremena kašnjenja sprovođen je upotrebom *traffic control* alata, pri čemu je simuliran problem sa mrežom od 200, 350 i 500 milisekundi. U *PostgreSQL* postavci sa međuslojem (slika 4), kašnjenja su simulirana na svakom čvoru, pri čemu su nešto lošije performanse dobijene kada je kašnjenje usmereno na lideru, što je očekivano s obzirom da taj čvor i dalje može prihvatati operacije čitanja, a izvršava jedini sve upise.



Slika 4. *Simulacija kašnjenja na PostgreSQL-u sa PgPool-om*

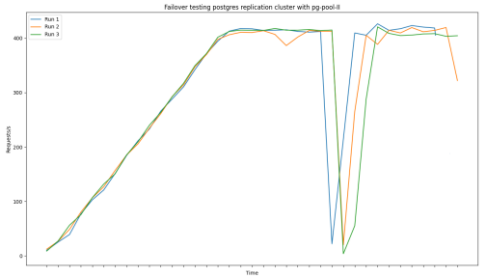
Testovi redom prikazuju 200, 350 i 500 milisekundi na opterećenju koje čini 200 korisnika. Razlog ovolikog pada u broju zahteva po sekundi ogleda se u tome da svaki zahtev izvršava dva do tri upita koji se izvršavaju unutar granica transakcije (*Begin* i *Commit*). Za svaki iskaz se agregira simulirano kašnjenje.

Kod *CockroachDB*-a, kašnjenja dodatno smanjuju broj zahteva po sekundi (slika 5), pri čemu se razlog za to može uzeti iz činjenice da je neophodan veći broj skokova između čvorova. S obzirom da ovakav sistem zbog svoje distribuiranosti u velikoj meri zavisi od optimizovanih upita, broj skokova zavisi od toga šta *cost-based optimizer* odluči da je neophodno uraditi na osnovu prikupljenih statistika tokom izvršavanja pri čemu direktno može uticati na povećano kašnjenje. Takođe, postojanje sekundarnih indeksa je od krucijalne važnosti za ovakav sistem, što dodatno može usporiti upise jer se isti mora održavati. Radi lakšeg prikaza, broj korisnika u ovim testovima je 800.

Slika 5. *Simulacija kašnjenja na CockroachDB-u*

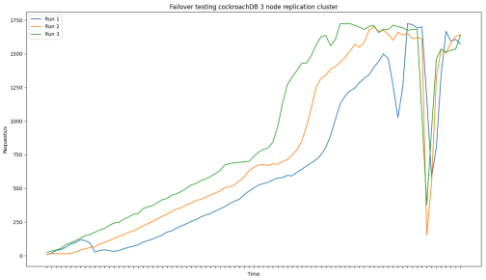
Ono što treba zaključiti je da neće svi zahtevi kasniti. Čitanja se vrše sa čvorova koji sadrže *lease*. Ako je problem van tog čvora, zahtev će vrlo verovatno biti netaknut. Za upise, neophodno je da 51+% potvrdi da je izvršilo replikaciju, što ne mora uzeti u obzir problematičan čvor. U većem klasteru je dodatno manja verovat-noća da se traženi podaci nalaze na problematičnom mestu.

Test otkaza lidera je dao prilično očekivane rezultate. Kod *PostgreSQL*-a, sistem je nedostupan onoliko vremena koliko je rečeno replikama da pokušavaju da kontaktiraju lidera, što je u slučaju testa bilo 12 sekundi (slika 6).



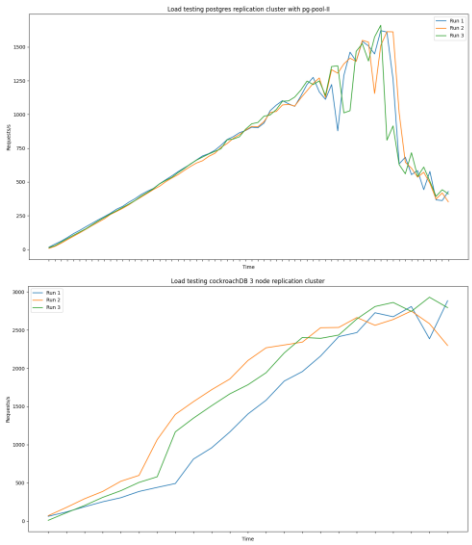
Slika 6. *Simulacija otkaza na PostgreSQL-u*

Sa druge strane, simulacija otkaza je vršena na svakom čvoru u klasteru (slika 7). Koliki je uticaj, zavisi od ukupnog broja *Raft* lidera na pogođenom čvoru, koji može biti varijabilan. Za razliku od *PostgreSQL*-a, ovaj sistem neće biti skroz nedostupan za vreme izbora novog lidera za pojedine opsege. Vreme neophodno za stabilizaciju je oko 9 sek. (vreme za izbor novog lidera i preuzimanja *lease*-a.



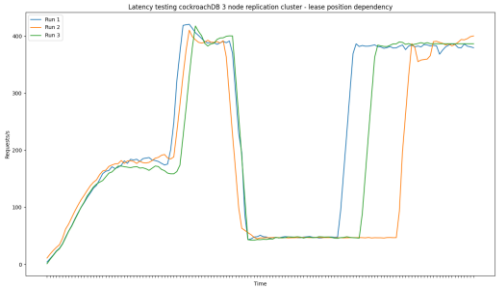
Slika 7. *Simulacija otkaza na CockroachDB-u*

Test velikog broja korisnika se odnosi na ispitivanje ponašanja u onim danima kada je veliko opterećenje sistema, pri čemu se bolje pokazao *CockroachDB* (slika 8) koji zbog distribuirane prirode manje opterećuje procesor i bolje koristi disk (običan HDD).



Slika 8. *Prikaz broja zahteva po sekundi pri većem broju korisnika*

Test praćenja pozicije se odnosi striktno na mogućnost *CockroachDB*-a da zakup (*lease*) prebaci u onu replikacionu zonu odakle dolazi najveći broj zahteva, ako će ta promena dovesti do poboljšanja performansi. S obzirom da čvor sa zakupom zaobilazi *Raft* konsenzus protokol, to će drastično poboljšati performanse operacija čitanja. Test je sprovođen sa jednostavnim servisom koji ima jednostavnu šemu baze radi lakše simulacije i provere navedene funkcionalnosti. Uočeno je da za transfer zakupa neophodno između 30 i 60 sekundi.



Slika 9. *Follow-the-workload funkcionalnost kod CockroachDB-a*

Test periodičnog gašenja i uključivanja *Docker* kontejnera je samo dodatno potvrdio neke opservacije iz prethodnih testova. Otkaz na bilo kom čvoru kod *PostgreSQL*-a potpuno obustavlja sve operacije upisa, dok kod *CockroachDB*-a će pojedini zahtevi ostati netaknuti.

5. ZAKLJUČAK

Tokom testiranja uočene su razne prednosti i jednog i drugog sistema, kao i određeni nedostaci. Sama priroda *Single-Leader* replikacije čini sistem lako razumljivim, ali implementacija kod *PostgreSQL*-a može dovesti do neželjenih okolnosti usled nedovoljnog razumevanja kako ona funkcioniše. Potpuno sinhrona postavka garantuje da kada je odgovor dobijen, sledeći zahtev će uočiti napravljene promene ili novije, nikad starije. Ali se ne rešava problem čitanja zastarelih podataka dok sam proces replikacije traje.

Razlog za to jeste izvršavanje *commit*-a na lideru nakon čega se čeka replikacija ka svim replikama. Pratilac koji prihvati podatke će ih moguće učiniti vidljivim pre nego što vrati odgovor vođi (zavisi od podešavanja), dok kod vođe podaci postaju vidljivi onog trenutka kada sve potvrde pristignu. Za striktno konzistentna čitanja, potrebno je da te operacije budu upućene ka lideru.

CockroachDB u rešava navedene probleme, što uz određena bolja ponašanja kroz testove čine sistem jako primamljivim. Pruža se visok nivo konzistentnosti koji gotovo da u potpunosti odgovara definiciji konzistentnosti iz *CAP* teoreme. Dostupnost u većim klasterima može biti na zavidnom nivou, za razliku od posmatrane relacione baze gde otkaz jednog pratioca negativno utiče na ceo sistem. Klaster je jednostavno namestiti i većina operacija se odvija automatski, bez preke potrebe za manuelnom intervencijom. Ono što je najveći nedostatak ovog sistema jesu perforomanse što zahteva visoku upotrebu indeksa koje mogu iste načiniti još lošijim, kao i visoku optimizaciju *SQL* upita. S obzirom da su podaci distribuirani, *join* operacije su jako neefikasne što čine ovaj sistem skoro neupotrebljivim u situacijama kada su te operacija jako učestale.

6. LITERATURA

[1] „Jepsen Analysis,“ [Na mreži]. Available: https://jepsen.io/analyses. [Poslednji pristup September 2020].

[2] M. Kleppman, „Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable and Maintainable Systems,“ 2017.

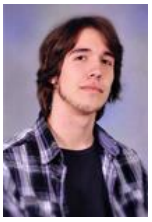
[3] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, „Understanding Replication in Databases and Distributed Systems, **“** *Proceedings - International Conference on Distributed Computing Systems*, 2000.

[4] D. Ongaro, J. Ousterhout, **„**In Search of an Understandable Consensus Algorithm,“ *USENIX Annual Technical Conference*, 2014.

[5] „Scalling Raft,“ [Na mreži]. Available: https://www.cockroachlabs.com/blog/scaling-raft/. [Poslednji pristup September 2020].

[6] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, P. Bardea, A. Ranade, B. Darnell, B. Gruneir, J. Jaffray, L. Zhang, P. Mattis, „CockroachDB: The Resilient Geo-Distributed SQL Database,“ *SIGMOD, the Association for Computing Machinery*, 2020.

Kratka biografija:



Stefan Colić rođen je u Beogradu 09.11.1995. godine. 2014. godine se upisuje na Fakultet Tehničkih nauka, smer softversko inženjerstvo i informacione tehnologije i diplomira 2018. godine sa prosekom 9.85. Iste godine upisuje master studije na istoimenom smeru.