

Replikacija podataka u PostgreSQL-u

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Predmet: Sistemi za upravljanje bazama podataka

|  |  |
| --- | --- |
| Student: | Profesor: |
|  |  |
| Jovan Vukadinović,  broj indeksa 1654 | Prof. dr Aleksandar Stanimirović |

Niš, jun 2024. godina

Sadržaj

[Uvod 4](#_Toc170641991)

[Vrste replikacija u PostgreSQL-u 5](#_Toc170641992)

[Razlika izmedju asinhrone i sinhrone replikacije podataka 5](#_Toc170641993)

[Problema sa performansama 6](#_Toc170641994)

[Razumevanje replikacije i gubitka podataka 6](#_Toc170641995)

[Replikacija sa jednim glavnim serverom naspram replikacije sa više glavnih servera 7](#_Toc170641996)

[Razlika između logičke i fizičke replikacije 8](#_Toc170641997)

[Kada koristiti fizičke replikacije 9](#_Toc170641998)

[Kada koristiti logičku replikaciju 9](#_Toc170641999)

[Korišćenje asinhrone replikacije u PostgreSQL-u 10](#_Toc170642000)

[Postavljanje strimovanja replikacije 10](#_Toc170642001)

[Podešavanje konfiguracionih fajlova na glavnom serveru 11](#_Toc170642002)

[Upravljanje *pb\_basebackup* i *recovery.conf* 11](#_Toc170642003)

[Omogućavanje čitanja na podređenom serveru 12](#_Toc170642004)

[Osnovni protokol 13](#_Toc170642005)

[Konfiguracija kaskadne replikacije 14](#_Toc170642006)

[Prebacivanje podređenih servera u glavni server 15](#_Toc170642007)

[Mešanje strimovanja i replikacije zasnovane na fajlovima 16](#_Toc170642008)

[Konfiguracija glavnog servera 17](#_Toc170642009)

[Kofiguracija podređenog servera 17](#_Toc170642010)

[Korišćenje sinhrone replikacije u PostgreSQL-u 18](#_Toc170642011)

[Postavljanje sinhrone replikacije 18](#_Toc170642012)

[Razumevanje nedostataka sinhrone replikacije 19](#_Toc170642013)

[Razumevanje parametra *application\_name* 19](#_Toc170642014)

[Implementacija sinhrone replikacije 20](#_Toc170642015)

[Provera replikacije 21](#_Toc170642016)

[Razumevanje problema performansi 22](#_Toc170642017)

[Postavljanje synchronous\_commit na “on” 22](#_Toc170642018)

[Postavljanje synchronous\_commit na “remote\_write” 22](#_Toc170642019)

[Postavljanje synchronous\_commit na “off” 22](#_Toc170642020)

[Postavljanje synchronous\_commit na “local” 23](#_Toc170642021)

[Promena podešavanja trajnosti u hodu 23](#_Toc170642022)

[Razumevanje praktičnih implikacija i performansi 24](#_Toc170642023)

[Redundantnost i zaustavljanje replikacije 25](#_Toc170642024)

[Konekciono upravljanje, balansiranje opterećenja i preuzimanje rada sa pgpool-II 26](#_Toc170642025)

[Konfiguracija pgpool-II 26](#_Toc170642026)

[Testiranje upisa i čitanja 30](#_Toc170642027)

[Zaključak 31](#_Toc170642028)

[Literatura 32](#_Toc170642029)

# Uvod

U današnje vreme, kada podaci predstavljaju ključni resurs za donošenje odluka, analizu i unapređenje poslovnih procesa, pouzdano i efikasno upravljanje bazama podataka postaje sve bitnije. Među različitim tehnikama koje se koriste za postizanje visokog nivoa pouzdanosti i dostupnosti podataka, replikacija podataka zauzima centralno mesto. Replikacija omogućava da podaci budu prisutni na više lokacija, čime se povećava otpornost na kvarove, unapređuje performanse sistema i omogućava kontinuitet poslovanja čak i u slučaju neočekivanih događaja.

Biće opisana replikacija podataka u PostgreSQL bazi podataka, jednoj od najpopularnijih i najmoćnijih otvorenih platformi za upravljanje bazama podataka. Fokus će biti na razumevanje osnovnih principa replikacije, razlikovanje različitih vrsta replikacije i analizu njihovih prednosti i mana. Detaljno će biti istraženo kako asinhrona i sinhrona replikacija funkcionišu u PostgreSQL-u, kako bi bilo jasnije koji pristup je najprikladniji za specifične potrebe.

Sinhrona replikacija obezbeđuje visoku konzistentnost podataka, osiguravajući da su svi podaci identični na glavnom i podređenom serveru u realnom vremenu. Međutim, ova metoda može dovesti do povećane latencije i problema sa performansama, posebno u sistemima sa visokim opterećenjem. S druge strane, asinhrona replikacija omogućava bržu obradu transakcija na glavnom serveru, ali uz određeni rizik od gubitka podataka u slučaju kvara pre nego što su podaci preneti na podređeni server.

Pored tehničkih aspekata replikacije, biće predstavljeni i praktični izazovi koji se javljaju prilikom implementacije različitih replikacionih strategija. Problemi sa performansama, kao što su kašnjenja u prenosu podataka i dodatni troškovi povezani sa sinhronom replikacijom, detaljno su analizirani kako bi se identifikovala najbolja rešenja za njihove prevazilaženje. Takođe, istražena su različita scenarija upotrebe logičke i fizičke replikacije, objašnjavajući kada i zašto je jedna metoda pogodnija od druge.

Deo rada posvećen je postavljanju strimovanja replikacije u PostgreSQL-u, uključujući konfiguraciju glavnog i podređenih servera, kao i upravljanje ključnim konfiguracionim fajlovima. Praktični primeri i uputstva omogućiće sticanje konkretnih veština potrebnih za implementaciju replikacije u stvarnim okruženjima.

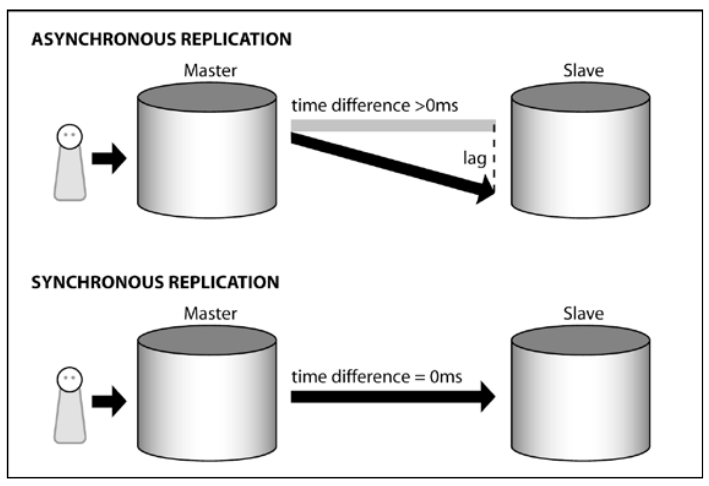
Na kraju, će biti prikazano jedan vrlo popularan alat za replikaciju u PostgreSQL-u koji se zove Pgpool-II. Kroz sveobuhvatan pregled teorijskih i praktičnih aspekata replikacije podataka, steći ćemo solidnu osnovu za implementaciju replikacionih strategija u različitim poslovnim okruženjima.

# Vrste replikacija u PostgreSQL-u

Dva glavna tipa replikacija koja postoje u PostgreSQL-u su sinhrona i asinhrona replikacija podataka. Važno je znati njene razlike kao i prednosti i mani, kako bi se mogao izvršiti pravilan izbog alata koji će se koristiti u bazi.

## Razlika izmedju asinhrone i sinhrone replikacije podataka

Prva razlika koja se može napraviti jeste da li će se replikacija vršiti sinhrono ili asinhrono. Zamislimo sledeći scenario, pretpostavimo da imamo dva servera i želimo da replikujemo podatke sa jednog servera (glavnog) na drugi server (podređeni). Sledeći dijagram ilustruje koncept sinhrone i asinhrone replikacije:



Slika 1. Koncept sinhrone i asinhrone replikacije podataka

Možemo koristiti jednostavnu transakciju kao što je prikazano u sledećem primeru:

BEGIN;

INSERT INTO foo VALUES ('bar');

COMMIT;

U slučaju asinhrone replikacije, podaci se mogu replikovati nakon što je transakcija potvrđena na glavnom serveru. Drugim rečima, podređeni server nikada nije ispred glavnog, i u slučaju pisanja, obično je malo iza glavnog servera. Ovo kašnjenje se naziva “lag”, odnosno vreme koje portekne između momenta upisa na glavnom i podređenom serveru.

Sinhrona replikacija nameće stroža pravila konzistentnosti. Ako odluka padne za sinhronu replikaciju, sistem mora da osigura da podaci upisani transakcijom budu prisutni na najmanje dva servera u trenutku kada se transakcija potvrđuje. Ovo podrazumeva da podređeni server ne zaostaje za glavnim i da podaci koje vide krajnji korisnici budu identični na oba servera.

### Problema sa performansama

Slanje nepotrebnih poruka preko mreže može biti skupo i vremenski zahtevno. Ako se transakcija replikuje na sinhron način, PostgreSQL mora da osigura da podaci stignu do drugog čvora, što može dovesti do problema sa latencijom. Sinhrona replikacija može biti skuplja od asinhrone replikacije na mnoge načine, pa bi bilo poželjno dva puta razmisliti da li je ovaj dodatni trošak zaista potreban i opravdan. U slučaju sinhrone replikacije, potrebne su potvrde sa udaljenog servera, što naravno izaziva dodatni trošak. Mnogo je urađeno u PostgreSQL-u da se ovaj trošak što više smanji, ali on i dalje postoji.

### Razumevanje replikacije i gubitka podataka

Kada se transakcija replikuje sa glavnog na podređeni server, mnogo toga treba uzeti u obzir, posebno kada je u pitanju gubitak podataka. Pretpostavimo da replikujemo podatke asinhrono na sledeći način:

1. Transakcija se šalje glavnom serveru.
2. Ona se potvrđuje na glavnom serveru.
3. Glavni server se ugasi pre nego što potvrda stigne do podređenog servera.
4. Podređeni server nikada neće primiti ovu transakciju.

U slučaju asinhrone replikacije, postoji vremenski period, takozvani lag, tokom kojeg podaci mogu biti izgubljeni. Veličina ovog perioda može varirati u zavisnosti od vrste podešavanja. Njegova veličina može biti vrlo kratka (možda svega nekoliko milisekundi) ili duga (minute, sate ili dane). Važna činjenica je da podaci mogu biti izgubljeni. Mali zaostatak samo smanjuje verovatnoću gubitka podataka, ali bilo koji zaostatak veći od nula podložan je gubitku podataka. Ako podaci mogu biti izgubljeni, žrtvujemo deo konzistentnosti u CAP teoremu, tojest ako dva servera nemaju iste podatke, oni nisu sinhronizovani.

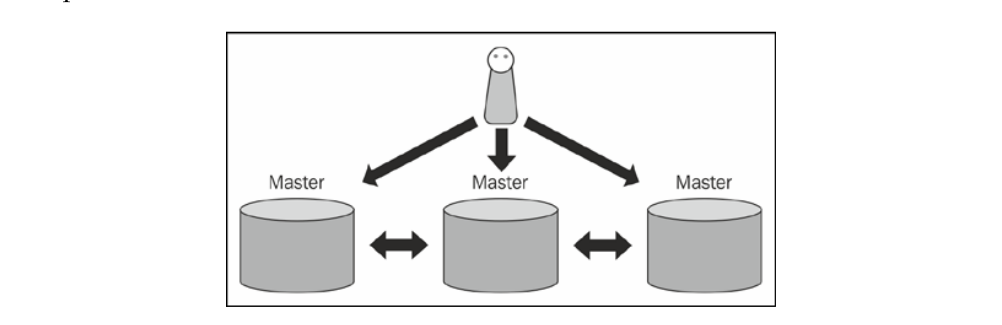
Ako želite da budete sigurni da podaci nikada neće biti izgubljeni, morate preći na sinhronu replikaciju. Kao što je i pomenuto, transakcija je sinhrona zato što će biti validna samo ako se potvrdi na najmanje dva servera.

## Replikacija sa jednim glavnim serverom naspram replikacije sa više glavnih servera

Drugi način za klasifikaciju različitih replikacionih podešavanja jeste razlikovanje između replikacije sa jednim glavnim serverom, odnosno single-master i replikacije sa više glavnih servera, tojest multi-master.

“Single-master” znači da se upisi mogu vršiti samo na jednom serveru, koji zatim distribuira podatke podređenim serverima unutar sistema. Podređeni serveri mogu primati samo čitanja, ali ne i upise.

Suprotno od replikacije sa jednim glavnim serverom, “multi-master” replikacija omogućava upise na sve servere unutar klastera. Sledeći dijagram pokazuje kako stvari funkcionišu na konceptualnom nivou:



Slika 2. Multi-master dijagram

Mogućnost upisa na bilo koji čvor unutar klastera zvuči kao prednost, ali to nije nužno tako. Razlog za to je što multi-master replikacija dodaje puno složenosti sistemu. U slučaju kada postoji samo jedan glavni server, potpuno je jasno koji podaci su tačni i u kom pravcu će podaci teći, te retko dolazi do sukoba tokom replikacije. Multi-master replikacija je sasvim drugačija, jer se upisi mogu vršiti na mnogim čvorovima istovremeno, i klaster mora biti savršeno svestan sukoba i rukovati njima na odgovarajući način. Alternativa bi bila korišćenje zaključavanja, locks mehanizma, za rešavanje problema, ali ovaj pristup takođe ima svoje izazove.

## Razlika između logičke i fizičke replikacije

Još jedan način klasifikacije replikacije jeste razlikovanje između logičke i fizičke replikacije. Razlika je suptilna, ali veoma važna:

* Fizička replikacija znači da će sistem preneti podatke u izvornom obliku na udaljeni uređaj. Dakle, ako se nešto upiše, udaljeni uređaj će dobiti podatke u binarnom formatu, a ne preko SQL-a.
* Logička replikacija znači da će se promene, koje su ekvivalentne dobijenim podacima, replikovati.

Primer za bolje razumevanje razlike:

test=# CREATE TABLE t\_test (t date);

CREATE TABLE

test=# INSERT INTO t\_test VALUES (now())

RETURNING \*;

t

------------

2024-06-16

(1 row)

INSERT 0 1

Vidimo dve transakcije. Prva transakcija kreira tabelu. Kada je to završeno, druga transakcija dodaje jednostavan datum u tabelu i potvrđuje.

U slučaju logičke replikacije, promena će biti poslata u neki red u logičkom formatu, tako da sistem ne šalje običan SQL, već nešto poput ovoga:

test=# INSERT INTO t\_test VALUES ('2024-06-16');

INSERT 0 1

Treba primetiti da je poziv funkcije zamenjen stvarnom vrednošću. Loš primer bi bio kada bi podređeni server ponovo izračunao now(), jer datum na udaljenom uređaju može biti potpuno drugačiji.

Fizička replikacija funkcioniše na potpuno drugačiji način. Umesto slanja nekog SQL-a (ili nečeg sličnog) koji je logički ekvivalentan promenama, sistem će poslati binarne promene koje PostgreSQL interno pravi.

Ovo su neke od binarnih promena koje su ove dve transakcije mogle izazvati:

1. Dodavanje 8K bloka u *pg\_class* i stavljanje novog zapisa tamo (da označi da tabela postoji).
2. Dodavanje redova u *pg\_attribute* za čuvanje imena kolona.
3. Izvršenje raznih promena unutar indeksa na tim tabelama.
4. Zabeležavanje statusa potvrde, i tako dalje.

Cilj fizičke replikacije je stvaranje kopije datog sistema koja je, uglavnom, identična na fizičkom nivou. To znači da će isti podaci biti na istom mestu unutar kreiranih tabela na svim uređajima. U slučaju logičke replikacije, sadržaj bi trebalo da bude identičan, ali nije važno da li je na istom mestu ili ne.

### Kada koristiti fizičke replikacije

Fizička replikacija je veoma pogodna za korišćenje i posebno jednostavna za postavljanje. Široko se koristi kada je cilj imati identične kopije određenog sistema, kao rezervnu kopiju ili za jednostavno skaliranje sistema.

U mnogim podešavanjima, fizička replikacija je standardna metoda koja korisniku nudi najniži mogući nivo složenosti. A ona je i idealna je za skaliranje podataka.

### Kada koristiti logičku replikaciju

Logička replikacija obično je malo teža za postavljanje, ali nudi veću fleksibilnost. Posebno je važna kada je u pitanju nadogradnja postojeće baze podataka.

Fizička replikacija nije u potpunosti pogodna za skokove između verzija, jer se ne možete jednostavno osloniti na činjenicu da svaka verzija PostgreSQL-a ima isti format na disku. Format skladištenja se može promeniti tokom vremena, pa stoga binarna kopija nije izvodljiva za skok sa jedne verzije na drugu.

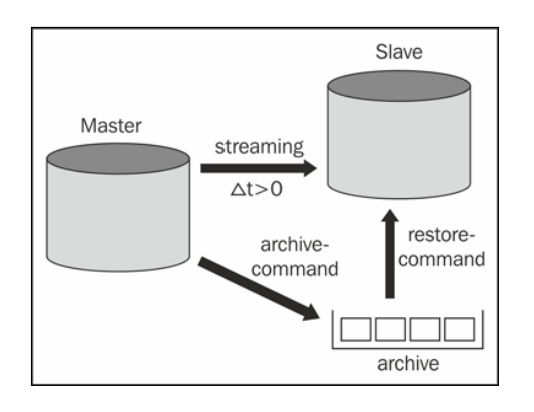
Logička replikacija omogućava razdvajanje načina na koji su podaci skladišteni od načina na koji se prenose i replikuju. Koristeći neutralni protokol koji nije vezan za određenu verziju PostgreSQL-a, lako je preći sa jedne verzije na narednu.

Od verzije PostgreSQL-a 9.4, postoji nešto što se zove logičko dekodiranje (Logical Decoding). To omogućava korisnicima da izvlače interne promene koje se šalju u XLOG i ponovo ih prikazuju kao SQL.

# Korišćenje asinhrone replikacije u PostgreSQL-u

## Postavljanje strimovanja replikacije

Sa replikacijom strimovanjem, kašnjenje će biti minimalno i postojaće dodatni nivo zaštite podataka. Pogledajmo opštu arhitekturu PostgreSQL striming infrastrukture. Sledeći dijagram ilustruje osnovni dizajn sistema:



Slika 3. Opšta arhitektura PostgreSQL striming infrastrukture

Treba primetiti strim konekciju. To je u osnovi normalna konkecija baze podataka, kao što bi se koristilo u bilo kojoj drugoj aplikaciji. Jedina razlika je što će u slučaju strim konekcije, konekcija biti u posebnom režimu kako bi mogla da prenosi XLOG.

## Podešavanje konfiguracionih fajlova na glavnom serveru

Kako bi se uspostavila striming konekcija na glavnom serveru sledeće stvari moraju biti podešene:

* Parametar *wal\_level* mora biti postavljen na *hot\_standby*
* Parametar *max\_wal\_senders* mora biti na dovoljno visokom nivou da podrži dovoljan broj podređenih servera.

Sada kada glavni server zna da treba da proizvodi dovoljno XLOG-ova i rukuje XLOG senderima, može se preći na sledeći korak.

Iz sigurnosnih razloga, mora se konfigurisati glavni server da omogućava striming replikacione konekcije. Ovo zahteva promenu *pg\_hba.conf* datoteke. Ovo je potrebno da bi se pokrenuo *pg\_basebackup* i kasnija striming konekcija. Čak i ako se koriste tradicionalne metode za uzimanje osnovne bekap kopije, i dalje se moraju omogućiti replikacione konekcije da bi se strimovao XLOG, tako da je ovaj korak obavezan.

Kada je glavni server uspešno konfigurisan, može se restartovati baza podataka (da bi *wal\_level* i *max\_wal\_senders* stupili na snagu) i nastaviti rad na podređenom serveru.

## Upravljanje *pb\_basebackup* i *recovery.conf*

Do sada je bilo prikazano da je proces replikacije apsolutno identičan izvođenju normalnog PITR-a. Jedina razlika do sada je *wal\_level*, koji mora biti drugačije konfigurisan za običan PITR. Osim toga, tehnika je ista.

Da bi se preuzela osnovna bekap kopija, može se koristiti *pg\_basebackup*:

iMac:dbhs$ pg\_basebackup -D /target\_directory \

-h sample.postgresql-support.de\

--xlog-method=stream

Sada kada je osnovna bekap kopija preuzeta, može se preći na konfiguraciju strimovanja. Da bi se to uradilo, mora se kreirati fajl pod nazivom *recovery.conf*. Evo jednostavnog primera:

standby\_mode = on

primary\_conninfo = ' host=sample.postgresql-support.de port=5432 '

Od PostgreSQL verzije 9.3 nadalje, postoji -R flag za *pg\_basebackup*, koja može automatski generisati *recovery.conf*. Drugim rečima, novi podređeni server se može generisati korišćenjem samo jedne komande.

Postoje dva nova podešavanja:

* *standby\_mode*: Ovo podešavanje će osigurati da PostgreSQL ne prestane sa radom kada ostane bez XLOG-a. Umesto toga, čekaće da stigne novi XLOG. Ovo podešavanje je neophodno kako bi drugi server bio u standby modu, koji stalno pušta XLOG.
* *primary\_conninfo*: Ovo podešavanje će podređenom serveru reći gde da pronađe glavni server. Ovde se mora staviti standardni PostgreSQL konekt string (kao u *libpq*). Promenljiva *primary\_conninfo* je centralna i govori PostgreSQL-u da strimuje XLOG.

Za osnovna podešavanja, ova dva podešavanja su potpuno dovoljna. Sve što sada treba učiniti je da se pokrene podređeni server, kao što se pokreće normalna instanca baze podataka:

iMac:slavehs$ pg\_ctl -D / start

server starting

LOG: database system was interrupted; last known up

at 2024-06-18 21:08:39 CET

LOG: creating missing WAL directory

"pg\_XLOG/archive\_status"

LOG: entering standby mode

LOG: streaming replication successfully connected

to primary

LOG: redo starts at 0/2000020

LOG: consistent recovery state reached at 0/3000000

Instanca baze podataka je uspešno startovana. Detektuje da su normalne operacije bile prekinute. Zatim ulazi u standby režim i počinje da strimuje XLOG sa primarnog sistema. PostgreSQL zatim dostiže konzistentno stanje i sistem je spreman za rad.

## Omogućavanje čitanja na podređenom serveru

Do sada je postavljeno samo strimovanje. Podređenmi server već dobija transakcioni log sa glavnog servera, ali na njemu još nije dozvoljeno čitanje. Ako se proba povezivanje sa instancom, desiće se sledeći scenario:

iMac:slavehs$ psql -l

FATAL: the database system is starting up

psql: FATAL: the database system is starting up

Ovo je podrazumevana konfiguracija. Instanca na podređenom serveru je konstantno u režimu bekapa i nastavlja da reprodukuje XLOG.

Ako je potrebno omogućiti čitanje na podređenom serveru, mora se prilagoditi *postgresql.conf* na sistemu tog servera, odnosno *hot\_standby* mora biti postavljen na “on”. Ovo se može postaviti odmah, ali takođe se ova promena može napraviti i kasnije i jednostavno restartovati instanca podređenog servera kada je potrebno omogućiti ovu funkciju.

iMac:slavehs$ pg\_ctl -D ./target\_directory restart

waiting for server to shut down....

LOG: received smart shutdown request

FATAL: terminating walreceiver process due to administrator command

LOG: shutting down

LOG: database system is shut down

done

server stopped

server starting

LOG: database system was shut down in recovery at 2024-06-18 21:56:12

CET

LOG: entering standby mode

LOG: consistent recovery state reached at 0/3000578

LOG: redo starts at 0/30004E0

LOG: record with zero length at 0/3000578

LOG: database system is ready to accept read only connections

LOG: streaming replication successfully connected to primary

Restart će isključiti server i ponovo ga pokrenuti. Ukoliko se pogleda log, može se videti da je proces pod nazivom *walreceiver* prekinut.

Kada se sistem ponovo pokrene, omogućeno je povezivanje sa serverom. Dozvoljene su samo operacije koje su u režimu čitanja:

test=# CREATE TABLE x (id int4);

ERROR: cannot execute CREATE TABLE in a read-only transaction

Server neće prihvatiti upise, što je i očekivano. Podređeni serveri su samo za čitanje.

## Osnovni protokol

Kada se koristi striming replikacija, treba obratiti pažnju na dva procesa:

* *wal\_sender*
* *wal\_receiver*

*wal\_sender* instance su procesi na instanci glavnog servera koji isporučuju XLOG svom ekvivalentu na podređenom serveru, zvanim *wal\_receiver*. Svaki podređeni server ima tačno jedan *wal\_receiver* proces, i ovaj proces je povezan sa tačno jednim *wal\_sender* procesom na izvoru podataka.

Kako ovo funkcioniše interno. Kao što je već navedeno, konekcija sa podređenim serverima na glavni je u osnovi normalna konekcija na bazu podataka. Transakcioni log koristi više-manje isti metod kao komanda COPY. Unutar COPY moda, PostgreSQL koristi mali mikro jezik za prenos informacija sa jedne strane na drugu. Glavna prednost je što ovaj mali jezik ima svoj parser, pa je moguće brzo i relativno jednostavno dodati funkcionalnosti na jednostavan način. Od verzije PostgreSQL 9.4, sledeće komande su podržane:

* IDENTIFY\_SYSTEM: Ovo zahteva od servera da se identifikuje. Server odgovara sa četiri polja (*systemid, timeline, xlogpos, dbname*).
* TIMELINE\_HISTORY tli: Ovo zahteva od servera da pošalje timeline istorijski fajl za dati timeline. Odgovor se sastoji od imena fajla i sadržaja.
* CREATE\_REPLICATION\_SLOT slot\_name {PHYSICAL | LOGICAL output\_plugin}: Ovo kreira replikacioni slot (fizički ili logički). U slučaju logičkog replikacionog slota, potreban je izlazni plugin za formatiranje podataka koji se vraćaju putem replikacionog slota.
* START\_REPLICATION [SLOT slot\_name] [PHYSICAL] xxx/xxx [TIMELINE tli]: Ovo govori serveru da započne strimovanje WAL za dati replikacioni slot na određenoj poziciji za određeni timeline.
* START\_REPLICATION SLOT slot\_name LOGICAL XXX/XXX [ ( option\_name [option\_value] [, ... ] ) ]: Ovo započinje logičko strimovanje od određene pozicije nadalje.
* DROP\_REPLICATION\_SLOT slot\_name: Ovo uklanja replikacioni slot.
* BASE\_BACKUP [LABEL 'label'] [PROGRESS] [FAST] [WAL] [NOWAIT] [MAX\_RATE rate]: Ovo vrši osnovni bekap, s određenim opcionalnim parametrima.

Ono što vidite je da je nivo protokola prilično blizak onome što *pg\_basebackup* nudi kao komande sa linije komandnog interfejsa.

## Konfiguracija kaskadne replikacije

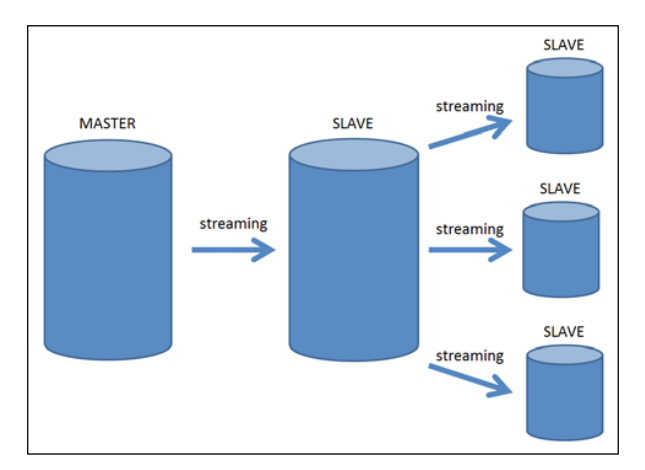
Pokazano je da je postavljanje striming replikacije vrlo jednostavno. Sve što je potrebno je postaviti nekoliko parametara, uzeti osnovni bekap, i replikacija je pokrenuta.

Međutim, u mnogim slučajevima situacija je malo drugačija. Pretpostavimo u ovom primeru da je potrebno koristiti glavni server za distribuciju podataka na desetine servera. Overhead replikacije je zapravo vrlo mali, uobičajeno je overhead podređenog servera oko 3 procenata ukupne performanse – međutim, ovo je samo gruba procena, ali ako radite nešto malo dovoljno često, to i dalje može predstavljati problem. Definitivno nije korisno za glavni server da ima, recimo, 100 podređenih servera.

Još jedan primer korišćenja je sledeći: imati glavni server na jednoj lokaciji i nekoliko podređenih servera na nekoj drugoj lokaciji. Nema smisla slati puno podataka na veliku udaljenost iznova i iznova. Mnogo je bolje poslati ih jednom i distribuirati na drugu stranu.

Da bi osigurali da ne moraju svi serveri dobijati transakcioni log sa jednog glavnog servera, može se koristiti kaskadna replikacija. Kaskadiranje znači da glavni server može strimovati svoj transakcioni log na podređeni server, koji će zatim služiti kao dispečer i strimovati transakcioni log na udaljene podređene servere.

Sledeći dijagram ilustruje osnovnu arhitekturu kaskadne replikacije:



Slika 4. Dijagram arhitekture kaskadne replikacije

Podređeni serveri sa desne strane dijagrama mogu opet služiti kao dispečeri. Ovim vrlo jednostavnim metodom može se osnovati sistem beskonačne veličine.

Procedura za postavljanje je u suštini ista kao i za postavljanje jednog podređenog servera. Lako se mogu uzeti osnovni bekapovi sa operativnog podređenog servera, *postgresql.conf* i *pg\_hba.conf* moraju biti konfigurisani kao u slučaju jednog glavnog servera.

## Prebacivanje podređenih servera u glavni server

Podređeni server može biti veoma koristan ako je potrebno povećati broj čitanja ili dobiti bekap podataka. Međutim, podređeni server ne mora uvek ostati podređeni server. U nekim trenucima, možda će biti potrebno prebaciti podređeni server u glavni. U klasičnim slučajevima, to se dešava kada originalni glavni server padne ili kada hardver mora biti promenjen.

PostgreSQL nudi jednostavne načine za ovo. Prvi način, i verovatno najprikladniji način, za prebacivanje podređenog servera u glavni je korišćenjem *pg\_ctl* komande:

iMac:slavehs$ pg\_ctl -D /target\_directory promote

server promoting

iMac:slavehs$ psql test

psql (9.2.4)

Type "help" for help.

test=# CREATE TABLE sample (id int4);

CREATE TABLE

Komanda promote će signalizirati postmasteru i prebaciti podređeni server u glavni. Kada je ovo završeno, možete se povezati i kreirati objekte.

Pored komande promote, postoji druga opcija za prebacivanje podređenog servera u glavni. Posebno kada pokušavate integrisati PostgreSQL sa softverom za visoku dostupnost po vašem izboru, može biti lakše kreirati jednostavan fajl nego pozvati *init* skriptu.

Da bi se koristila metoda zasnovana na fajlu, može se dodati komanda *trigger\_file* u *recovery.conf* fajl:

trigger\_file = '/some\_path/start\_me\_up.txt'

U ovom slučaju, PostgreSQL će čekati da fajl pod nazivom */neka\_putanja/start\_me\_up.txt* bude kreiran. Sadržaj ovog fajla je potpuno nebitan, PostgreSQL jednostavno proverava da li fajl postoji, i ako postoji, PostgreSQL prestaje sa oporavkom i prebacuje se u glavni server.

Kreiranje praznog fajla je prilično jednostavan zadatak:

iMac:slavehs$ touch /some\_path/start\_me\_up.txt

Sistem baze podataka će reagovati na novi fajl, start\_me\_up.txt:

FATAL: terminating walreceiver proced fire up:

LOG: trigger file found: /some\_path/start\_ss due to

administrator command

LOG: redo done at 0/50000E0

LOG: selected new timeline ID: 2

LOG: archive recovery complete

LOG: database system is ready to accept connections

LOG: autovacuum launcher started

PostgreSQL će proveravati za fajl koji je definisan u *recovery.conf* svakih 5 sekundi. Za većinu slučajeva, ovo je savršeno dovoljno i prihvatljivo brzo.

## Mešanje strimovanja i replikacije zasnovane na fajlovima

U nekim slučajevima, striming replikacija može biti savršena. U drugim slučajevima, replikacija zasnovana na fajlovima i PITR su dovoljno dobre. Ali postoje i mnogi slučajevi u kojima je potrebno ponešto od oba. Na primer, kada se prekine replikacija na duži period, možda će postojati potreba da se ponovo sinhronizuju podređeni serveri koristeći arhivu umesto izvođenja potpunog osnovnog bekapa. Takođe može biti korisno zadržati arhivu za neku kasniju istragu ili operaciju ponovnog puštanja.

Dobra vest je da PostgreSQL omogućava mešanje replikacije zasnovane na fajlovima i strimovanju. Ne morate odlučivati da li je striming replikacija ili replikacija zasnovana na fajlovima bolja. Jer postoji mogućnost kombinovanja najboljeg iz oba sveta u isto vreme.

Već je prikazana većina potrebnih komandi, samo ih je potrebno pravilno izkombinovati. Da bi bilo jasnije, u nastavku je kompletan primer ovog korišćenja.

### Konfiguracija glavnog servera

Na glavnom serveru, se može koristiti sledeća konfiguraciju u *postgresql.conf*:

wal\_level = hot\_standby

archive\_mode = on

# allows archiving to be done

# (change requires restart)

archive\_command = 'cp %p /archive/%f'

# command to use to archive a logfile segment

# placeholders: %p = path of file to archive

# %f = file name only

max\_wal\_senders = 5

# we used five here to have some spare capacity

Pored ovoga, moraju se dodati neke linije konfiguracije u *pg\_hba.conf* da bi se omogućio striming. Evo primera:

# Allow replication connections from localhost, by a user with the

# replication privilege.

local replication hs trust

host replication hs 127.0.0.1/32 trust

host replication hs ::1/128 trust

host replication all 192.168.0.0/16 md5

U ovom slučaju, jednostavno smo otvorili celu mrežu da bi se omogućila replikaciju, kako bi primer bio jednostavan. Kada napravimo ove promene, može se restartovati glavni server i uzeti osnovni bekap.

### Kofiguracija podređenog servera

Kada konfigurišemo naš glavni server i uzmemo osnovni bekap, možemo početi sa konfigurisanjem našeg podređenog servera. Pretpostavimo radi jednostavnosti da koristimo samo jedan podređeni server, nećemo kaskadirati replikaciju na druge sisteme.

Samo treba da promenimo jednu liniju u *postgresql.conf* na podređenom serveru:

hot\_standby = on # to make the slave readable

U sledećem koraku, treba napisati jednostavan *recovery.conf* fajl i staviti ga u glavni data direktorijum:

restore\_command = 'cp /archive/%f %p'

standby\_mode = on

primary\_conninfo = ' host=sample.postgresql-support.de port=5432 '

trigger\_file = '/some\_path/start\_me\_up.txt'

Kada se pokrene podređeni server, desiće se sledeće:

1. PostgreSQL će pozvati *restore\_command* da preuzme transakcioni log iz arhive.
2. Radiće to dok ne nestane fajlova u arhivi.
3. PostgreSQL će pokušati da uspostavi striming konekciju.
4. Strimovaće podatke ako postoje.

Ako postoji potreba da se pretvori podređeni server u glavni, može se ponovo koristiti *pg\_ctl* *promote* ili fajl *trigger\_file* definisan u *recovery.conf*.

# Korišćenje sinhrone replikacije u PostgreSQL-u

Prikazana je replikacijom zasnovanom na fajlovima (ili prenosima logova, kako se često naziva) i jednostavno striming asinhrono podešavanje. U oba slučaja, podaci se šalju i primaju od strane podređenog servera, ili više njih, nakon što je transakcija potvrđena na glavnom serveru. Tokom vremena između potvrde na glavnom i trenutka kada podređeni server zapravo potpuno primi podatke, oni mogu biti izgubljeni.

Sinhrona replikacija može biti ključna u replikacionom sistemu, pružajući sistem koji osigurava nulti gubitak podataka.

## Postavljanje sinhrone replikacije

Kao što je pomenuto, sinhrona replikacija je napravljena da zaštiti podatke po svaku cenu. Osnovna ideja sinhrone replikacije je da transakcija mora biti na najmanje dva servera pre nego što glavni server vrati uspeh klijentu. Obezbeđivanje da su podaci na najmanje dva čvora ključni je zahtev kako bi se osiguralo da nema gubitka podataka u slučaju pada sistema. Podešavanje sinhrone replikacije funkcioniše isto kao i podešavanje asinhrone replikacije. Samo nekoliko parametara, koji su već pomenuto, mora biti promenjeno da se omogućili prednosti sinhrone replikacije. Međutim, ukoliko planirate kreiranje sistema zasnovanog na sinhronoj replikaciji, preporučuje se da se započnete sa asinhronim podešavanjem i postepeno proširuje konfiguracija, pretvarajući je u sinhronu replikaciju. Ovo će omogućiti lakše otklanjanje grešaka i izbegavanje problema u budućnosti.

## Razumevanje nedostataka sinhrone replikacije

Najvažnija stvar koja mora biti pomenuta o sinhronoj replikaciji je da je ona jednostavno skupa. Neophodno je razumeti fizička ograničenja sinhrone replikacije, jer bi inače mogli nastati veliki problemi.

Kada se postavlja sinhrona replikacija, mora se obratiti pažnja na sledeće stvari:

* Minimizacija kašnjenja
* Uverenje da postoje redundantne konekcije
* Sinhrona replikacija je skuplja od asinhrone replikacije
* Uvek dvaput razmisliti da li zaista postoji potreba za sinhronom replikacijom

U mnogim slučajevima je sasvim prihvatljivo izgubiti nekoliko redova u slučaju pada sistema. Sinhrona replikacija može se bezbedno preskočiti u ovom slučaju. Međutim, ako postoji nulta tolerancija na gubitak podataka, sinhrona replikacija je alat koji treba koristiti.

## Razumevanje parametra *application\_name*

Da bi sinhrona postavka bila jasnija, konfiguraciona promenljiva nazvana *application\_name* je od suštinskog značaja i igra važnu ulogu u sinhronoj postavci. U tipičnoj aplikaciji, koristi se parametar *application\_name* za potrebe otklanjanja grešaka, jer omogućava korisnicima da dodele ime svojoj bazi podataka. To može pomoći u praćenju grešaka, identifikovanju šta aplikacija radi, i slično:

test=# SHOW application\_name;

application\_name

------------------

psql

(1 row)

test=# SET application\_name TO 'whatever';

SET

test=# SHOW application\_name;

application\_name

------------------

whatever

(1 row)

Može se videti da je dozvoljeno menjanje parametra *application\_name*. Podešavanje je validno za sesiju u kojoj se nalazi i biće uklonjeno čim se diskonektuje. Postavlja se pitanje koja je veza između *application\_name* i sinhrone replikacije.

Ako je vrednost *application\_name* deo *synchronous\_standby\_names*, podređeni server će biti sinhroni. Pored toga, da bi bio sinhroni podređeni server, mora biti:

* Povezan
* Strimovati podatke u realnom vremenu, to jest, ne preuzimati stare WAL zapise

Kada podređeni server postane sinhronizovan, ostaje u toj poziciji dok se ne diskonektuje. Sa svim ovim informacijama, može se nastaviti konfiguracija sinhrone replikacije.

## Implementacija sinhrone replikacije

Kako bi videli kako sinhrona replikacija funkcioniše, najbolje je videti na celovitom, funkcionalnom primer koji obuhvata sve relevantne konfiguracione parametre. Nekoliko promena mora biti izvršeno na glavnom serveru. Sledeća podešavanja će biti potrebna u *postgresql.conf* na glavnom serveru:

wal\_level = hot\_standby

max\_wal\_senders = 5 # or any number

synchronous\_standby\_names = 'book\_sample'

hot\_standby = on

# on the slave to make it readable

Zatim se mora prilagoditi *pg\_hba.conf*. Nakon toga, server može biti restartovan i glavni server je spreman za rad.

U sledećem koraku može se izvršiti osnovna bekap kopija baš kao što je to rađeno ranije. Mora se pozvati *pg\_basebackup* na podređenom serveru. Idealno bi bilo uključiti transakcioni log prilikom kreiranja osnovne bekap kopije. Parametar *--xlog-method=stream* nam omogućava da brzo pokrenemo stvari bez većih rizika.

Već je preporučeno da se postavi *hot\_standby* na glavnom severu. Konfiguraciona datoteka će ionako biti replikovana. Ovo pristup nije idealan, ali je jednostavan pristup za primer.

Kada je osnovna bekap kopija izvršena, može se preći na pisanje jednostavne *recovery.conf* datoteke pogodnu za sinhronu replikaciju:

iMac:slavehs$ cat recovery.conf

primary\_conninfo = 'host=localhost

application\_name=book\_sample

port=5432'

standby\_mode = on

Konfiguraciona datoteka izgleda isto kao pre. Jedina razlika je što je dodat *application\_name* u. Treba imati na umu da parametar *application\_name* mora biti identičan *synchronous\_standby\_names* podešavanju na glavnom serveru.

Kada završimo pisanje *recovery.conf*, može se pokrenuti podređeni server.

U primeru, podređeni server je na istom serveru kao i glavni. U tom slučaju, mora se osigurati da te dve instance koriste različite TCP portove, inače instanca koja se druga pokrene neće moći da se startuje. Port se lako može promeniti u *postgresql.conf*.

Nakon ovih koraka, instanca baze podataka može biti pokrenuta. Podređeni server će proveriti svoje informacije o konekciji i povezati se sa glavnim. Kada prelista sve relevantne transakcione logove, biće u sinhronizovanom stanju. Glavni i podređeni server će od tada držati potpuno iste podatke.

### Provera replikacije

Sada kada je pokrenuta instanca baze podataka, može se izvršiti povezivanje na sistem i videti da li sve funkcioniše ispravno.

Da bi se proverila replikacija, potrebno je povezati se na glavni server i pogledati *pg\_stat\_replication*. Za ovu proveru, potrebno je povezati se na bilo koju bazu podataka unutar instance glavnog servera, na sledeći način:

postgres=# \x

Expanded display is on.

postgres=# SELECT \* FROM pg\_stat\_replication;

-[ RECORD 1 ]----+------------------------------

pid | 62871

usesysid | 10

usename | hs

application\_name | book\_sample

client\_addr | ::1

client\_hostname |

client\_port | 59235

backend\_start | 2024-06-19 14:53:52.352741+01

state | streaming

sent\_location | 0/30001E8

write\_location | 0/30001E8

flush\_location | 0/30001E8

replay\_location | 0/30001E8

sync\_priority | 1

sync\_state | sync

Ovaj sistemski pregled prikazuje tačno jedan red po podređenom serveru koji je povezan sa sistemom glavnog servera. Može se videti da je parametar *application\_name* preuzet iz konekcijskog stringa koji je podređeni server prosledio glavnom, a u ovom primeru je to *book\_sample*. Pošto se parametar *application\_name* poklapa sa podešavanjem *synchronous\_standby\_names* na glavnom serveru, uverili smo se da se sistem replikuje sinhrono. Nijedna transakcija se ne može izgubiti jer će svaka transakcija završiti na dva servera trenutno.

Podešavanje *sync\_state* će tačno reći kako se podaci kreću sa glavnog na podređeni server.

### Razumevanje problema performansi

Pomenuto je više puta da je sinhrona replikacija skup proces. Treba imati na umu da se čeka odgovor od udaljenog servera, a ne samo lokalnog sistema. Mreža između ta dva čvora definitivno nije nešto što će ubrzati stvari. Pisanje na više od jednog čvora uvek je skuplje nego pisanje samo na jedan čvor. Stoga se mora pažljivo pratiti brzina, inače može doći do ogromnog kašnjenja.

Ponovo je potrebno razmisliti o tome, da li zaista ima potrebe za replikaciju svih transakcija sinhrono. U mnogim slučajevima, nije poželjno. Da bismo ovo pokazali, zamislimo tipičan scenario, banka želi da čuva podatke vezane za računovodstvo kao i neke podatke o logovanju. Nije poželjno izgubiti nekoliko miliona dolara samo zato što je pao čvor baze podataka. Ovaj tip podataka može biti vredan truda sinhronog replikovanja. Međutim, podaci o logovanju su sasvim drugačiji. Može biti previše skupo nositi se sa opterećenjem sinhronog replikovanja. Dakle, potrebno je da replikujemo ove podatke na asinhron način kako bismo obezbedili maksimalni protok.

Kako možemo konfigurisati sistem da ispravno upravlja važnim i manje važnim transakcijama. Odgovor je korišćenjem *synchronous\_commit*.

### Postavljanje synchronous\_commit na “on”

U podrazumevanoj PostgreSQL konfiguraciji, *synchronous\_commit* je podešen na “on”. U ovom slučaju, commit-ovi će čekati odgovor od trenutnog sinhronog standby servera koji pokazuje da je primio zapis o commitu transakcije i da ga je flush-ovao na disk. Drugim rečima, oba servera moraju potvrditi da su podaci bezbedno upisani. Jedino ako oba servera otkažu u isto vreme, što je veoma neočekivano, vaši podaci će preživeti moguće probleme.

### Postavljanje synchronous\_commit na “remote\_write”

Flush-ovanje na oba diska može biti veoma skupo. U mnogim slučajevima dovoljno je znati da je udaljeni server prihvatio XLOG i prosledio ga operativnom sistemu bez flush-ovanja na disk na podređenim serverima. S obzirom da možemo biti prilično sigurni da nećemo izgubiti oba servera istovremeno, ovo je razumna kompromis između performansi i doslednosti u zaštiti podataka.

### Postavljanje synchronous\_commit na “off”

Ideja je odložiti upisivanje WAL zapisa kako bi se smanjilo flush-ovanje diska. Ovo se može koristiti ako je performansa važnija od trajnosti. U slučaju replikacije, to znači da ne replikujemo na potpuno sinhron način.

Treba imati na umu da ovo može imati ozbiljan uticaj na aplikaciju. Zamislite transakciju koja se commituje na glavnom serveru i trenutno želite da koristite te podatke na jednom od podređenih servera. Postoji mali prozor tokom kojeg se mogu dobiti zastareli podaci.

### Postavljanje synchronous\_commit na “local”

Vrednost local znači da se flush-uje lokalno, ali se ne čeka odgovor od replike. Drugim rečima, transakcija postaje asinhrona.

Postavljanje *synchronous\_commit* na local takođe može uzrokovati mali prozor vremena tokom kojeg podređeni server može vratiti nešto starije podatke. Ovaj fenomen mora biti uzet u obzir kada odlučujete da izvršite čitanja na podređenom serveru.

Ako želite da replikujete sinhrono, bitno je uveriti se da je *synchronous\_commit* podešen na “on” ili “*remote\_write*”.

### Promena podešavanja trajnosti u hodu

Promena načina na koji se podaci replikuju u hodu je jednostavna i veoma važna za mnoge aplikacije, jer omogućava korisniku da kontroliše trajnost podataka u realnom vremenu. Svi podaci nisu stvoreni jednako, stoga je važno da se važniji podaci čuvaju na sigurniji način u odnosu na manje važne podatke, kao što su log fajlovi. Već je postavljena potpunu infrastruktura za sinhronu replikaciju prilagođavanjem parametra *synchronous\_standby\_names* (glavni server) zajedno sa parametrom *application\_name* (podređeni server). Dobra stvar kod PostgreSQL-a je što se može promeniti zahtevi za trajnost podataka u hodu:

test=# BEGIN;

BEGIN

test=# CREATE TABLE t\_test (id int4);

CREATE TABLE

test=# SET synchronous\_commit TO local;

SET

test=# \x

Expanded display is on.

test=# SELECT \* FROM pg\_stat\_replication;

-[ RECORD 1 ]-------+------------------------------

pid | 62871

usesysid | 10

usename | hs

application\_name | book\_sample

client\_addr | ::1

client\_hostname |

client\_port | 59235

backend\_start | 2024-06-19 14:53:52.352741+01

state | streaming

sent\_location | 0/3026258

write\_location | 0/3026258

flush\_location | 0/3026258

replay\_location | 0/3026258

sync\_priority | 1

sync\_state | sync

test=# COMMIT;

COMMIT

U ovom primeru, promenjeni su zahtevi za trajnost podataka u hodu. Ovo će se pobrinuti da ovaj veoma specifičan transakcioni zahtev ne čeka da se podaci upišu na disk na podređenom serveru. Može se videti, da se *sync\_state* nije promenio. PostgreSQL je potpuno sposoban da rukuje sa svakom transakcijom zasebno, i ovo je jedinstvena karakteristika ove open-source baze podataka. Dozvoljava kontrolu i omogućava odlučivanje prilikom zahteva za trajnost.

## Razumevanje praktičnih implikacija i performansi

Pogledajmo jednostavan test i vidimo kako se replikacija ponaša. Ovakvi testovi nam služe da se pokaže da različiti nivoi trajnosti nisu samo sporedna tema, već da su ključni za performanse.

U sledećem scenariju, povezane su dve podjednako snažne mašine (3 GHz, 8 GB RAM) preko 1 Gbit mreže. Mašine su jedna pored druge. Da bismo pokazali uticaj sinhronog replikovanja, ostavljen je *shared\_buffers* i svi drugi parametri memorije kao podrazumevane, i samo je isključen *fsync* kako bi se smanjio efekat čekanja na disku gotovo na nulu.

Test je jednostavan: koristimo tabelu sa jednom kolonom koja sadrži samo jedno polje tipa integer i izvodimo 10,000 pojedinačnih transakcija, svaka sastavljena od samo jedne INSERT naredbe:

INSERT INTO t\_test VALUES (1);

Možemo da probamo ovo sa potpunim, sinhronim replikovanjem (*synchronous\_commit = on*):

real 0m6.043s

user 0m0.131s

sys 0m0.169s

Kao što možete videti, testu je trebalo oko 6 sekundi da se završi.

Možemo ponoviti test sa *synchronous\_commit = local* (što efektivno znači asinhrono replikovanje):

real 0m0.909s

user 0m0.101s

sys 0m0.142s

U ovom jednostavnom testu, može se videti da se brzina povećala za skoro šest puta.

Naravno, ovo je grub primer koji ne odražava potpuno stvarnost. Ono što je važno razumeti, jeste da sinhrono protiv asinhronog replikovanja podataka nije pitanje nekoliko procenata. Ovo dodatno naglašava poentu, da se sinhrona replikaciaj vrši samo ako je zaista potrebno, i ako zaista morate koristiti sinhronu replikaciju, obavezno ograničite broj sinhronih transakcija na apsolutni minimum.

Takođe, treba voditi računa da mreža može da izdrži opterećenje. Sinhrono replikovanje podataka preko mrežnih veza sa visokom latencijom može značajno umanjiti performanse sistema. Treba imati na umu da bacanje skupih hardvera na problem neće rešiti problem. Dupliranje brzine vaših servera praktično neće ništa promeniti jer će prava ograničenja uvek dolaziti od mrežne latencije.

Kako se osigurati da prilikom korišćenja sinhrone replikacije performanse ne pate previše. U osnovi, postoje nekoliko važnih sugestija koje su se pokazale korisnim:

* Korišćenje dužih transakcija: Zapamtite da sistem mora da obezbedi da su podaci dostupni na oba servera pri commit-ovanju. Ne zanima nas šta se dešava u sredini transakcije, jer niko van naše transakcije ionako ne može videti podatke. Duža transakcija će dramatično smanjiti komunikaciju preko mreže.
* Pokretanje stvari istovremeno: Ako imate više od jedne transakcije koje se odvijaju istovremeno, biće korisno za performanse. Razlog za to je što će udaljeni server vratiti poziciju unutar XLOG-a koja se smatra sigurno obrađenom (flushed ili prihvaćenom). Ovaj metod osigurava da mnoge transakcije mogu biti potvrđene istovremeno.

## Redundantnost i zaustavljanje replikacije

Kada govorimo o sinhronom replikovanju, postoji jedan fenomen koji se ne sme zanemariti. Uzmimo primer da postoji klaster sa dva čvora koji se sinhrono replikuje. Šta se dešava ako jedan od čvorova prestane da radi? Odgovor je da glavni server ne može lako razlikovati između sporo radećeg i ugašenog čvora, pa će početi da čeka da se čvor vrati.

Na prvi pogled, ovo može izgledati nelogično, ali ako se dublje razmisli, shvatiće se da je sinhrona replikacija zapravo jedino ispravno rešenje. Ako neko odluči da koristi sinhronu replikaciju, podaci u sistemu moraju nešto vredeti, pa ne smeju biti ugroženi. Bolje je odbiti unos podataka i obavestiti krajnjeg korisnika nego rizikovati podatke i tiho ignorisati zahteve za visokom pouzdanošću.

Ako se odlučite za korišćenje sinhronog replikovanja, mora se razmotriti upotrebu najmanje tri čvora u klasteru. Inače će biti veoma rizično, i biće onemogućeno priuštiti gubitak jednog čvora bez značajnog vremena neprekidnog rada ili rizika od gubitka podataka.

# Konekciono upravljanje, balansiranje opterećenja i preuzimanje rada sa pgpool-II

Postoji mnogo rešenja za replikaciju, konekciono upravljanje, balansiranje opterećenja i preuzimanje rada u slučaju otkaza. Pogledajmo rešenje zvano pgpool-II, koji nudi sledeće stvari:

* **Balansiranje opterećenja:** pgpool-II middleware može distribuirati SELECT upite preko nekoliko servera koji su konfigurisani u *pgpool.conf*, u zavisnosti od težine dodeljene serverima. On ne šalje isti upit na više od jednog servera kada se koristi na način koji će biti prikazan u nastavku. Potrebno je biti pažljiv sa funkcijama koje menjaju podatke i izvršavaju se putem SELECT.
* **Konekciono upravljanje:** Dobijanje konekcije uvek ima manji trošak. Ponovno korišćenje konekcija je ključno za performanse. pgpool-II middleware pruža konekciono upravljanje sa parametrima za određivanje broja pool-ova kao i broja konekcija u pool-u. On ponovo koristi konekciju kada dođe zahtev za novom konekcijom sa istim svojstvima, kao što su korisnik, baza podataka i protokol.
* **Preuzimanje rada u slučaju otkaza za klaster:** Ovo se postiže putem skripte koju se mora napisati. U slučaju da pgpool-II detektuje otkaz primarnog čvora, on može izvršiti skriptu. U skripti možemo pokrenuti prebacivanje rezervnog čvora u glavni.
* **Visoka dostupnost za pgpool-II:** Ovo se radi kako pgpool-II ne bi postao jedinstvena tačka otkaza.

Jedan nedostatak korišćenja PostgreSQL-ove funkcije striminga sa pgpool-II je taj što na kraju replikujemo sve baze podataka i tabele u klasteru. Nije moguće selektivno replikovati određene baze podataka ili tabele. Replikaciono rešenje kao što je Slony, može se koristiti ako postoji potreba replikovanja samo dela promena sa primarnog čvora. Ako tražimo rešenje koje pruža samo konekciono upravljanje, trebalo bi da pogledamo pgbouncer.

## Konfiguracija pgpool-II

Ukoliko se koristi linux, pgpool-II se može preuzeti sledećom komandom, a potom je potrebno raspakovati ga:

wget -O pgpool-II-3.3.3.tar.gz \

http://www.pgpool.net/download.php?

f=pgpool-II-3.3.3.tar.gz && \

tar -xzvf pgpool-II-3.3.3.tar.gz

Potom, koristeći cd je potrebno prebaciti se u direktorijum, gde je potrebno da se konfiguriše i instalira:

cd pgpool-II-3.3.3 && ./configure &&

make && sudo make install

Zatim se treba prebaciti u direktorijum *sql/pgpool-recovery*, i uraditi *make* i *make install*. Ovi koraci će kreirati ekstenziju (sql fajl) u */usr/local/pgsql/share/extension/*. Potrebno je povezati se na bazu podataka template1 (psql) na primarnom čvoru i izvršiti sledeću komandu:

CREATE EXTENSION pgpool\_recovery;

Nakon toga sledi postavljanje konfiguracije. Može se pronaći nekoliko primera konfiguracionih fajlova u */usr/local/etc*:

pgpool.conf.samplepgpool.conf.sample-

master-slave pgpool.conf.sample-

replication pgpool.conf.sample-stream

pool\_hba.conf.sample

Koristiće se *pgpool.conf.sample-stream* jer je postavljena striming replikaciju. Takođe treba kreirati korisnika koji će se koristiti za health provere i provere striming replikacije. U psql sesiji na primarnom čvoru, treba kreirati sledećeg korisnika, i on će se koristiti za online oporavak:

CREATE USER pgpooluser WITH PASSWORD

'pgpool';

Izmenu pgpool konfiguracionih fajlova i pokretanje pgpool-a može obaviti root korisnik. Slede izmene koje je potrebno primeniti u *pgpool.conf*:

cd /usr/local/etc

cp pgpool.conf.sample-stream pgpool.conf

cp pool\_hba.conf.sample pool\_hba.conf

Izmene koje treba napraviti u *pgpool.conf*:

listen\_addresses = '127.0.0.1'

Ovo je slično konfiguraciji *postgresql.conf*. Ovde postavljamo pgpool-II na koje interfejse da sluša na backend-u 1:

backend\_hostname0 = '127.0.0.1'

backend\_port0 = 2345

Backend sa brojem 0 se tretira kao glavni server. Ali kako se koristi streaming replikacija, ovo nije od prevelike važnosti. Može postojati samo jedan glavni. Podaci o čvoru koji će biti kasnije u log datotekama će se odnositi na ovaj broj:

backend\_weight0 = 0

Može se odrediti odnos balansiranja opterećenja ovim podešavanjem. Vrednost ovde je postavljena na 0, a vrednost za rezervni je postavljena na 1 kako bi se osiguralo da SELECT upiti uvek idu na rezervni čvor. Težine treba prilagoditi uzimajući u obzir broj čvorova, odnos pisanja/čitanja i slične faktore. Na primer, ako postoje 3 čvora i težine su postavljene na 0, 1 i 2 za čvor 0, čvor 1 i čvor 2 redom, 0 SELECT upita će ići na čvor 0, oko jedne trećine SELECT upita će ići na čvor 1, a dve trećine SELECT upita će ići na čvor 2:

backend\_data\_directory0 = '/pgdata/9.3'

backend\_hostname1 = '127.0.0.1'

backend\_port1 = 5432

backend\_weight1 = 1

backend\_data\_directory1 = '/pgdata/

standby'

backend\_flag1 = 'DISALLOW\_TO\_FAILOVER'

Ako rezervni čvor ne uspe, ne treba pokušati prebacivanje na drugi čvor:

num\_init\_children = 8

Ovo je broj pool-ova. Ovaj broj pomnožen sa max\_pool, odnosno broj veza po grupi, treba da bude manji od maksimalnog broja veza postavljenog na PostgreSQL čvoru, umanjenog za *superuser\_reserved\_connections*, ako sve veze idu ka jednom PostgreSQL čvoru:

log\_statement = on

log\_per\_node\_statement = on

Ove opcije bi trebalo isključiti u produkcijskom okruženju. Ovde će biti trenutno uključene da bi se prikazalo balansiranje opterećenja i separacija čitanja/pisanja:

sr\_check\_user = 'pgpooluser'

sr\_check\_password = 'pgpool'

health\_check\_period = 10

Treba proveriti health svakih 10 sekundi:

health\_check\_user = 'pgpooluser'

health\_check\_password = 'pgpool'

failover\_command = '/usr/local/pgsql/

failover.sh %d /pgdata/standby/down.trg'

Komanda za prebacivanje može biti bilo koja skripta. U primeru je ona samo nekoliko redova. Kreirajte skriptu nazvanu failover.sh u */usr/local/pgsql* direktorijumu:

#! /bin/sh

# Arguments: $1: failed node id. $2:

trigger file

failed\_node=$1

trigger\_file=$2

# Do nothing if standby goes down.

if [ $failed\_node = 1 ];

then exit 0;

fi

# Create the trigger file.

/bin/touch $trigger\_file

exit 0;

Dodeljivanje execute dozvole za datoteku, korišćenjem sledeće komande:

chmod +x /usr/local/pgsql/failover.sh

Skripta proverava da li je došlo do kvara na rezervnom čvoru. Ako jeste, ne radi ništa. Ako nije, kreira datoteku trigger-a korišćenjem touch komande. Kada rezervni čvor uoči datoteku trigger-a, to će uzrokovati prekid striming replikacije i rezervni čvor će se otvoriti u režimu čitanja/pisanja. Napomena, da je */pgdata/standby/down.trg* je vrednost koja je postavljena za trigger\_file u *recovery.conf* za rezervni čvor:

mkdir -p /var/run/pgpool

Sada treba startovati pgpool:

pgpool -n > /tmp/pgpool.log 2>&1 &

Kreirajmo bazu podataka za testiranje pgpool-a. Na primarnom čvoru, povežite se sa psql-om. Osigurajte da se povezujemo preko pgpool-a navođenjem ispravnog porta (9999):

psql -p 9999

postgres=#CREATE DATABASE pgp;

postgres=# SHOW pool\_nodes;

node\_id | hostname | port | status | lb\_weight | role

-------------+------------+------------+------------+------------+------------

0 | 127.0.0.1 | 2345 | 2 | 0.000000 | primary

1 | 127.0.0.1 | 5432 | 2 | 1.000000 | standby

postgres=# SHOW pool\_processes;

pool\_pid| start\_time |database|username|reate\_time |pool\_counter

--------+---------------------+--------+--------+----------------+------------

13722 | 2024-06-20 15:52:47 | | | |

13723 | 2024-06-20 15:52:47 | pgp |postgres| 2024-06-20 15:52:51 | 1

13724 | 2024-06-20 15:52:47 | | | |

### Testiranje upisa i čitanja

U novoj psql sesiji treba ubaciti neki zapis, obrisati zapis, i izvršiti nekoliko SELECT naredbi, i na taj način videti kako pgpool-II upravlja svim tim:

psql -p 9999

postgres=# \c pgp

You are now connected to database "pgp"

as user "postgres"

pgp=# CREATE TABLE tbl(id INT);

pgp=# SELECT \* FROM tbl;

id

----

(0 rows)

pgp=# INSERT INTO tbl VALUES(1);

INSERT 0 1

pgp=# SELECT \* FROM tbl;

id

----

1

(1 row)

pgp=# DELETE FROM tbl;

DELETE 1

pgp=# SELECT \* FROM tbl;

id

----

(0 rows)

U ssh sesiji, otvoriti log file:

more /tmp/pgpool.log

Počinje sledećim redovima:

find\_primary\_node: primary node id is 0

2024-06-20 16:57:25 LOG: pid 15361: DB

node id: 1 backend pid: 15388 statement:

SELECT \* FROM tbl;

2024-06-20 16:57:37 LOG: pid 15361:

statement: INSERT INTO tbl VALUES(1);

2024-06-20 16:57:37 LOG: pid 15361: DB

node id: 0 backend pid: 15387 statement:

INSERT INTO tbl VALUES(1);

2024-06-20 16:57:40 LOG: pid 15361:

statement: SELECT \* FROM tbl;

2024-06-20 16:57:40 LOG: pid 15361: DB

node id: 1 backend pid: 15388 statement:

SELECT \* FROM tbl;

2024-06-20 16:57:45 LOG: pid 15361:

statement: DELETE FROM tbl;

2024-06-20 16:57:45 LOG: pid 15361: DB

node id: 0 backend pid: 15387 statement:

DELETE FROM tbl;

2024-06-20 16:57:50 LOG: pid 15361:

statement: SELECT \* FROM tbl;

2024-06-20 16:57:50 LOG: pid 15361: DB

node id: 1 backend pid: 15388 statement:

SELECT \* FROM tbl;

Može se primetiti da INSERT i DELETE naredbe idu u čvor 0, a SELECT naredbe u čvor 1

# Zaključak

Replikacija podataka u PostgreSQL-u predstavlja kritičan aspekt modernog upravljanja bazama podataka, omogućavajući visoku dostupnost, otpornost na kvarove i poboljšane performanse sistema. Prikazane su različite metode replikacije, uključujući sinhronu i asinhronu replikaciju, analizirajući njihove specifične prednosti i mane. Sinhrona replikacija, sa svojim fokusom na konzistentnost podataka, pruža visok nivo sigurnosti ali može negativno uticati na performanse sistema zbog povećane latencije. Nasuprot tome, asinhrona replikacija omogućava bržu obradu transakcija uz rizik od potencijalnog gubitka podataka u slučaju kvara glavnog servera pre nego što su podaci replikovani.

Praktična implementacija replikacije podataka zahteva detaljno razumevanje i pažljivo planiranje. Prikazano je kako pravilno konfigurisati striming replikaciju, uključujući postavljanje glavnih i podređenih servera, te upravljanje ključnim konfiguracionim fajlovima. Ovi koraci su ključni za osiguravanje da sistem funkcioniše efikasno i da se minimiziraju rizici od gubitka podataka.

Izazovi povezani sa replikacijom, kao što su problemi sa performansama i dodatni troškovi, mogu biti značajni, ali se mogu prevazići pažljivim planiranjem i upotrebom odgovarajućih alata. Analiziran je vrlo popularan alat za replikaciju u PostgreSQL-u Pgpool-II, pružajući uvid u njegove specifične karakteristike i prednosti. Ovaj alat nudi različita rešenja za optimizaciju procesa replikacije i može biti od velike pomoći u različitim scenarijima poslovanja.

Replikacija podataka nije samo tehnička funkcija već strateška komponenta koja može značajno uticati na poslovanje. Pravilno implementirana replikacija poboljšava otpornost sistema na kvarove, omogućava kontinuitet poslovanja i poboljšava korisničko iskustvo.

Važnost replikacije podataka će nastaviti da raste sa sve većim oslanjanjem na podatke u savremenim poslovnim i tehnološkim okruženjima. Kontinuirano istraživanje i unapređenje tehnika replikacije ključni su za osiguravanje da sistemi ostanu pouzdani, efikasni i spremni da odgovore na buduće izazove.

# Literatura

[1] PostgreSQL dokumentacija - <https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf>

[2] Ibrar Ahmed, Gregory Smith, Enrico Pirozzi - PostgreSQL 10 High Performance-Packt Publishing (2018)

[3] Jayadevan Maymala - PostgreSQL for Data Architects-Packt Publishing (2015)

[4] Joshua D. Drake, John C. Worsley - Practical PostgreSQL

[5] Baji Shaik, Avinash Vallarapu - Beginning PostgreSQL on the Cloud\_ Simplifying Database as a Service on Cloud Platforms-Apress (2018)