Elektronski fakultet niš
Univerzitet u nišu

Replikacija kod PostgreSQL baze podataka
Sistemi za upravljanje bazama podataka

Mentor:
Doc. dr. Aleksandar Stanimirovic

Student:
Todor Majstorovic 1088

Elektronski fakultet Niš
Univerzitet u Nišu

Sadržaj

Uvod

Baze podataka mogu da rade u paru kako bi dozvolile da sekundarni server preuzme ukoliko dođe do pada primarnog(visoka dostupnost - high availability), ili da dozvole većem broju

servera da opslužuju iste podatke(balansiranje opterećenja - load balancing): Idealno, serveri baza podataka treba da rade bez zastoja. Web serveri koji opslužuju statične web stranice se mogu kombinovati tako što se web zahtevi pravilno rasporede na veći broj servera. Baze koje služe samo za čitanje se mogu kombinovati na vrlo jednostavan način. Nažalost većina servera baza podataka imaju i upite za čitanje i za pisanje, i zato je njih puno teže kombinovati. Ovo je zbog toga što podaci koji se čitaju moraju da se postave na svaki server, dok naredbe za upis moraju da se propagiraju na svaki server da bi nadolazeći zahtevi za čitanje vraćali validne rezultate.

Ovaj problem sinhronizacije je osnovni problem za saradnju servera. Pošto ne postoji jedinstveno rešenje koje bi eliminisalo problem sinhronizacije, potrebno je pristupiti ovom problemu iz više uglova. Svako rešenje se odnosi na ovaj problem drugačije, i ima svoje prednosti i mane.

Neka rešenja se nose sa sinhronizacijom tako što dozvole samo jednom serveru da menja podatke. Serveri koji mogu da menjaju podatke se nazivaju read/write serveri, odnosno master ili primarni serveri. Serveri koji prate promene u masteru se nazivaju standby (pomoćni) odnosno slave(rob) serveri. Pomoćni serveri na koje se ne može povezatii se nazivaju warm standby serveri, a oni koji dozvoljavaju konekciju se nazivaju hot standby serveri

Neka rešenja su sinhrona, to jest svaka transakcija koja menja podatke se ne smatra komitovanom sve dok svi je svi serveri ne komituju. Ovo garantuje da failover(oporavak) ne gubi podatke i da svi serveri vraćaju dosledne rezultate bez obzira na to nad kojim serverom je pozvan upit. Suprotno, asinhrona solucija dozvoljava kašnjenje između komitova i njegove propagacije na ostale servere, tako da može doći do gubitka nekih transakcija, gde bi takvi serveri vraćali nevalidne rezultate. Asinhrona komunikacija je puno brža od sinhrone i ona se koristi kada je sinhrona prespora.

Sa druge strane, rešenja se mogu kategorisati po njihovoj granularnosti. Neka rešenja obuhvataju kompletan server baze podataka, dok se druga bave na nivou tabele odnosno na nivou jedne baze.

Performanse su jako bitna stavka i moraju se razmatrati u bilo kom rešenju. Uglavnom postoji kompromis između funkcionalnosti i performansi. Na primer sinhrono rešenje na sporoj mreži može da prepolovi performanse, dok asinhroni ima minimalne gubitke.

1. Poredjenje različitih rešenja

`Oporavak deljenog diska`

Oporavak deljenog diska zaobilazi problem sinhronizacionog čekanja tako što ima samo jednu kopiju baze podataka. Koristi jedinstveni disk kome pristupaju više servera. Ukoliko dođe do pada glavne baze, pomoćni server se kači i pokreće bazu kao da je došlo do pucanja same baze. Ovo omogućava brz oporavak bez gubljenja podataka.

Ova funkcionalnost je dosta zastupljena u uređajima koji koriste skladišta koja se nalaze na mreži. Korišćenje mrežnog fajl sistema je još jedna mogućnost, medjutim mora da se obrati pažnja na to da se fajl sistem ponaša po POSIX pravilima. Najveća mana ove metode je to da ukoliko dođe do pada deljenih diskova,ili se ošteti, primarni i pomoćni serveri postaju nefunkcionalni. Još jedan problem je taj što pomoćni server ne bi trebalo da pristupa deljenom skladištu dok je primarni server aktivan.

`Replikacija fajl sistema`

Modifikovana verzija prethodnog rešenja je replikacija fajl sistema, gde se sve promene fajla preslikavaju na fajl sistem koji se nalazi na drugom kompjuteru. Jedina restrikcija je ta što se preslikavanje mora uraditi tako da pomoćni server uvek ima doslednu kopiju fajl sistema, odnosno da svi upisi na pomoćni sistem mora da se urade u istom redosledu kao i oni na masteru. DRBD je popularna replikacija fajl sistema za Linux.

`Hot i Warm pomočni serveri koji koriste PITR(Point-In-Time-Recovery)`

Ovakvi serveri se mogu usaglasiti sa primarnim serverom tako što čitaju tok write-ahead log (WAL) zapisa. Ukoliko dođe do pada glavnog servera, pomoćni server čuva većinu podataka iz njega, i može jednostavno da se napravi novim masterom. Ovo je asinhrono i moguće je da se izvršava za celokupnu bazu podataka.

PITR pomoćni server može da se implementira korišćenjem file-based log shipinga, odnosno korišćenjem replikacije toka, ili kombinacijom oba.

`Replikacija Master/Standby servera bazirana na triggerima`

Ovakva replikacija šalje sve upite koji menjaju podatke na master. Master asinhrono šalje promene na pomoćni server. Pomoćni server odgovara na upite za čitanje dok je master aktivan. Pomoćni server je idealan za upite nad skladištima podataka.

Slony-I je primer ovakve replikacije. On ima granularnost na nivou tabele, i podržava veći broj pomoćnih servera. Zbog asinhronog ažuriranja pomoćnog servera, može doći do gubitka podataka kada dođe do oporavka.

Replikacioni middleware koji je zasnovan na izrazima

Sa ovakvim middleware-om, program presreće svaki SQL upit i šalje ga na jedan ili više servera. Svaki server radi nezavisno od drugih. Upiti za čitanje i pisanje se moraju poslati na sve servere, tako da svaki server dobije promene. Dok upiti koji su samo za čitanje mogu da se pošalju i samo jednom serveru, gde će se nakon toga posao raspodeliti i na ostale servere.

Ako se upiti prosleđuju svim serverima, funkcije kao random(), CURRENT_TIMESTAMP mogu da imaju različite vrednosti na različitim serverima. To je zbog toga što svaki server funkcioniše nezavisno, i zbog toga što se SQL upiti prosleđuju svim serverima(a ne rešenja upita). Ukoliko je ovo neprihvatljivo, middleware ili aplikacija mora da vrši upite nad vrednostima iz jednog servera i onda da vrši upite za upis sa tim vrednostima. Druga opcija je da se iskoristi ova replikacija sa tradicionalnim master/standby postavkom, na primer upiti koji vrše promene nad podacima se jedino šalju na master i propagiraju se na pomoćne servere pomoću master/standby replikacije, za razliku od replikacije middleware-a. Mora se obratiti pažnja da se sve transakcije ili komituju ili prekidaju na svim serverima, kao na primer korišćenjem komita iz dve faze(PREPARE TRANSACTION and COMMIT PREPARED. Pgpool-II and Continuent Tungsten su primeri ovakve replikacije.

Asinhrona replikacija sa više master servera

Za servere koji nisu direktno povezani, kao laptopovi ili udaljeni serveri, održavanje podataka predstavlja pravi izazov. Korišćenjem ove replikacije, svaki server radi nezavisno, i periodično komunicira sa drugim serverima da identifikuje upite koji istovremeno hoće da upisuju u isti fajl. Ovi konflikti se mogu razrešiti ili od strane korisnika ili korišćenjem pravila koja regulišu konflikte. Bucardo je jedan primer ovakve replikacije.

Sinhrona replikacija sa više master servera

U sinhronoj replikaciji sa više master servera, svaki server prihvata zahteve za upisivanje, i promenjeni podaci se transmituju sa glavnog na svak idrugi server u mreži pre nego što se transakcija komituje. Učestalim upisom može doći do zaključavanja, čime se dolazi do loših performansi. U tim slučajevima, performanse upisa su čak i gore nego kada postoji samo jedan server. Zahtevi za čitanje se mogu slati na bilo koji server. Neke implementacije koriste deljeni disk da bi smanjili komunikaciona čekanja. Ova replikacija je uglavnom dobra za čitanje read upita, mada je glavna prednost to što svi serveri primaju zahteve za upis- nema potrebe za deljenjem poslova između master

i pomoćnih servera, i zbog toga što se promene nad podacima šalju sa jednog servera na drugi, nema problema ni kod nedeterminističkih funkcija kao što je random().

PostgreSQL ne pruža uslove za ovakav tip replikacije, mada je moguće iskoristiti commit kroz 2 faze (PREPARE TRANSACTION and COMMIT PREPARED) da se implementira u aplikacionom kodu ili middleware-u

Tabela 1 rezimira atribute gorepomenutih rešenja

Feature	Oporavak deljenog diska	Replikacija fajl sistema	Hot i Warm pomočni serveri koji koriste PITR(Point-In-Time-Recovery)	Replikacija Master/Standby servera bazirana na triggerima	Replikacioni middleware koji je zasnovan na izrazima	Asinhrona replikacija sa više master servera	Sinhrona replikacija sa više master servera
Najzastupljenija implementacija	**NAS**	**DRBD**	**PITR**	**Siony**	**pgpool-II**	**Bucardo**	
Metod komunikacije	**deljeni disk**	**blokovi diska**	**WAL**	**Vrste tabela**	**SQL**	**Vrste tabela**	**Vrste tabela i ključevi**
Ne zahteva dodatni hardver		•	•	•	•	•	•
Dozvoljava veči broj master servera					•	•	•
Nema čekanja između mastera i pomočnog servera	•		•		•		
Nema čekanja na veći broj servera	•		•	•		•	
Padom mastera ne dolazi do gubitka podataka	•	•			•		•
Pomoćni server prihvata upite koji su samo za čitanje			**Samo Hot**	•	•	•	•

Granularnost na nivou tabela				•		•	•
Nije potrebna rezolucija konflikta	•	•	•	•			•

Postoji par slučajeva koji ne spadaju u gorepomenute.

`Particinisanje podataka`

Particinisanje podataka deli tabele u setove podataka. Svaki set se može modifikovati od strane jednog jedinog servera. Na primer podaci se mogu particinisati od strane kancelarija, recimo u Londonu i Parizu, sa po serverom u svakoj od kancelarija. Ukoliko je potrebno da upit kombinuje i podatke iz Londona i podatke iz Pariza, aplikacija može da vrši upit nad oba servera, ili se može iskoristiti Master/standby replikacija da se sačuva kopija koja je samo za čitanje u suprotnoj kancelariji.

`Paralelno izvršavanje upita na više serveraM`

Veliki broj gorepomenutih rešenja dozvoljava većem broju servera da barata većim brojem upita, ali nijedan ne dozvoljava upitu da koristi više servera da bi se brže izvršio. Ovo rešenje dozvoljava većem broju servera da rade istovremeno na istom upitu. Ovo se uglavnom izvršava tako što se podaci dele između servera i svaki server izvršava svoj deo upita i vraća rešenje u centralni server gde se kombinuju i vraćaju korisniku.Pgpool-II ima ovu sposobnost. Takodje, ovo može da se sprovede korišćenjem PL/Proxy setom alata.

2. Dostavljanje logova na standby server

Kontinuirano arhiviranje se može iskoristiti da bi se napravila HA (high availability) konfiguracija klastera sa jednim ili više pomoćnih servera koji su spremni da preuzmu ukoliko dođe do pada glavnog servera. Ovo se naziva warm standby ili log shipping.

Primarni i pomoćni serveri rade u paru da bi omogućili ovu sposobnost, iako su serveri slabo povezani. Primarni server radi u režimu neprekidnog arhiviranja, dok svaki pomoćni server radi u režimu neprekidnog oporavka, čitajući WAL fajlove iz primarnog. Nije potrebna nikakva promena nad tabelama u bazi podataka da bi se omogućila ova sposobnost, tako da ona pruža kratko vreme kašnjenja u odnosu na druga replikaciona rešenja. Ova konfiguracija takođe relativno malo utiče na primarni server.

Direktno premeštanje WAL zapisa sa jedne u drugu bazu podataka se naziva log-shipping. PostgreSQL izvršava file-based log shipping-e tako što premešta WAL zapise jednog po jednog fajla (WAL segment). WAL fajlovi (16MB) mogu da se šalju lako i jeftino na bilo kojoj udaljenosti, bez obzira da li je to susedni sistem, sistem na istom sajtu, ili sistem na drugoj strani planete. Protok koji je potreban za ovakvu tehniku varira u odnosu na stopu transakcije primarnog servera. Log shipovanje bazirano na zapisima je granularnije i WAL tokovi se menjaju postepeno preko mrežne veze.

Treba pomenuti da je log shipovanje asinhrono, to jest WAL zapisi se šalju nakon komitovanja transakcija. Kao posledica toga, postoji period u kome može doći do gubitka podataka ukoliko dođe do pada primarnog servera, i transakcije koje nisu poslate će se izgubiti. Veći broj podataka se može sačuvati korišćenjem archive_timeout parametra, koji može da se setuje i na par sekundi. Medjutim tako mali broj će znatno povećati protok koji je potreban za slanje fajlova. Stream replikacija ima znatno manji period u kome može doći do gubitka podataka.

Performanse oporavka su dovoljne dobre tako da bi pomoćni server uglavnom bio samo nekoliko sekundi od potpune dostupnosti nakon aktiviranja. Zbog toga, ovo se naziva Warm standby konfiguracijom.koja omogućava veliku pristupačnost. Oporavak servera iz arhiviranog backup-a i rollforward će trajati znatno duže, tako da ta tehnika je dobra samo za oporavak u slučaju pada master servera. Pomoćni server takođe može da se koristi za upite koji su samo za čitanje, u kom slučaju se oni nazivaju Hot standby serveri.

2.1. Planiranje

Poželjno je da se primarni i pomoćni serveri kreiraju tako da su što sličniji mogući, makar iz perspektive servera baze podataka. Na primer, putanja će se proslediti bez ikakve promene, tako da će i primarni i pomoćni server imati iste putanje za tablespace-ove ukoliko se iskoristi ta odlika. Imajte na umu da ukoliko se CREATE TABLESPACE iskoristi na primarnom serveru, svaka sledeća postavka mora da se kreira na primarnom serveru i na svim pomoćnim serverima pre nego što se izvrši. Hardware može da se razlikuje, ali u praksi

je puno lakše da se održavaju 2 slična servera nego 2 koja su potpuno različita. Arhitektura hardvera mora biti ista, znači shipovanje sa 32-bitnog na 64-bitni sistem ne bi bilo moguće.

2.2. Rad standby servera

Dok je u standby-u, server neprekidno primenjuje WAL koji je dobio od mastera. Pomoćni server može da čita WAL iz WAL arhive ili direktno putem TCP konekcije. Pomoćni server takođe pokušava da povrati sve WAL-ove koji su nadjeni u clusterovom pg_xlog direktorijumu. To se uglavnom desi nakon restarta servera, kada pomoćni server ponavlja WAL-ove koje je već dobio od mastera pre restarta, međutim moguće je i ručno kopirati fajlove u pg_xlog u bilo kom trenutku kako bi ih ponovili.

Kada se pokrene sistem, pomoćni server pribavlja sve WAL-ove koji su dostupni u arhivi, pomoću restore_command. Kada dođe do kraja dostupnih WAL-ova u arhivi i restore_command ne uspeva, pokušava da povrati sve WAL-ove koji su dostupni u pg_xlog direktorijumu. Ako to ne uspe, i konfigurisana je streaming replikacija, pomoćni server pokušava da se konektuje na primarni server i počinje da prosleđuje WAL iz poslednjeg validnog zapisa iz arhive ili pg_xlog-a. Ako to ne uspe ili streaming replikacija nije konfigurisana, ili ako se konekcija prekinula, pomoćni server se vraća na prvi korak i pokušava da povrati fajl iz arhive. Ovo se ponavalja sve dok se server ne stopira ili se oporavak pokrene od strane triggera.

2.3. Podesavanje standby servera

Da bi se postavio pomoćni server, mora da se povrate backup fajlovi iz baze koji su uzeti sa primarnog servera. Napravite fajl standby.signal u klasteru pomoćnog serevra. Postavite restore_command da jednostavno kopira fajlove iz WAL arhive. Ukoliko planirate da imate veći broj pomoćnih servera, recovery_target_timeline mora biti postavljena na latest, da bi se obezbedilo da pomoćni server prati sve promene koje se dešavaju za vreme oporavka.

Ukoliko želite da koristite replikaciju streama, popunite primary_conninfo sa stringa za konekciju libpq, uključujući ime hosta (odnosno IP adresu) i sve potrebne podatke za konekciju na primarni server. Ukoliko je potrebna šifra za autentikaciju, i ona mora da se navede u primary_conninfo komandi.

Ukoliko kreirate pomoćni server i treba vam da on ima visoku dostupnost, postavite WAL arhiviranje, konekcije i autentikaciju kao da je to primarni server, nakon toga će pomoćni imati istu ulogu kao i primarni nakon oporavka.

Ukoliko se koristi WAL arhiva, može joj se smanjiti veličina korišćenjem archive_cleanup_command parametra, on uklanja fajlove koji nisu potrebni pomoćnom serveru. Takodje, može se iskoristiti pg_archivecleanup uslužni program koji je dizajniran konkretno za korišćenje uz archive_cleanup_command u konfiguracijama sa jednim pomoćnim programom. Medjutim mora se obratiti pažnja ukoliko se koristi arhiva za backup, morate da čuvate fajlove sa poslednjeg backupa, iako oni više nisu neophodni pomoćnom server.

Primer ovakve konfiguracije:

```
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo
password=foopass options=''-c wal_sender_timeout=5000'''

restore_command = 'cp /path/to/archive/%f %p'

archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

Možete imati koliko god hoćete pomoćnih servera, ali ukoliko koristite replikaciju streamova, morate postaviti max_wal_senders na dovoljno veliki broj u primarnom serveru da bi dozvolili da se svi oni povežu simultano.

2.4 Replikacija toka

Ova replikacija omogućava pomoćnom serveru da ostane u toku više nego sto je to moguće pomoću file-based log shippinga. Pomoćni server se konektuje na glavni, koji salje WAL zapise na pomoćni čim se generišu, bez čekanja na WAL fajl da se ispuni.

Replikacija je asinhrona po defaultu, u kom slučaju postoji mali zastoj izmedju komita transakcije na primarnom serveru i vremena dok se te promene pojave na pomoćnom. Medjutim taj zastoj je mnogo manji nego u file-based log shippingu, uglavnom manje od 1 sekunde ako je pomoćni server dovoljno jak da izdrži taj teret. Kod ove replikacije archive_timeout nije neophodan za smanjenje vremena u kojem može doći do gubitka podataka.

Ako koristitie ovu replikaciju bez file-based arhiviranja, server može reciklirati zastarele WAL segmente pre nego što ih pomoćni server uopšte primi. Ukoliko dođe do ovoga, pomoćni server se mora ponovo inicijalizovati iz backupa nove baze. Medjutim, ovo se može izbeći ukoliko postavite wal_keep_segments na dovoljno veliku vrednost tako da mu pomoćni server uvek može pristupati.

Da bi koristili ovu replikaciju, pomoćni server se mora postaviti kao file-based log-shipping pomoćni server. Da bi ga konvertovali u server replikacionog streama, morate postaviti primary_conninfo podešavanje da pokazuje na primarni server. Postavite listen_addresses i autentikacione opcije primarnog servera tako da se pomoćni server može konektovati na pseudo-bazu primarnog servera.

Ukoliko sistem podržava keepalive socket opciju, može se postaviti tcp_keepalives_idle, tcp_keepalives_interval i tcp_keepalives_count da bi se brže uočio eventualni pad konekcije.

Postavite maksimalni broj konkurentnih konekcija sa pomoćnih servera.

Kada se standby pokrene i primary_conninfo je ispravno postavljen, pomoćni server će se konektovati na primarni nakon izvršavanja svih WAL fajlova koji su u arhivi. Ukoliko je konekcija uspostavljena, videće se walreceiver proces u pomoćnom serveru, i odgovarajući walsender proces u primarnom serveru.

2.4.1. Autentikacija

Vrlo je bitno da se privilegije pristupa replikacije postave tako da samo pouzdani korisnici mogu da čitaju iz WAL toka, pošto je vrlo jednostavno da se izvuču privilegovane informacije. Pomoćni serveri moraju da se identifikuju kao super korisnici ili akaunti koji imaju REPLICATION privilegiju. Preporučuje se da se napravi namenski korisnički account koji će imati REPLICATION i LOGIN privilegije za replikaciju. Iako REPLICATION privilegija daje dosta visoke dozvole korišćenja, ne daje dozvolu korisniku da menja podatke na primarnom serveru, za razliku od SUPERUSER privilegije.

Provera identiteta klijenta se kontroliše od strane pg_hba.conf zapisa koji specificira REPLICATION u ***`database`*** polju. Na primer ukoliko pomoćni server radi na 192.168.1.100 IP adresi, i ime replikacije je foo, admin može dodati sledeće informacije u pg_hba.conf fajl primarnog servera:

```
# Allow the user "foo" from host 192.168.1.100 to connect to the
primary

# as a replication standby if the user's password is correctly
supplied.

#

# TYPE  DATABASE        USER            ADDRESS
METHOD
```

```
host    replication     foo             192.168.1.100/32
md5
```

Ime hosta,broj porta primarnog servera, ime konekcije kao i šifra su svi navedeni u primary_conninfo segmentu. Šifra može da se postavi i u ~/.pgpass fajlu na pomoćnom serveru(mora se navesti replication u database polju). Recimo da primarni server radi sa hosta čiji je ip 192.168.1.50, i čiji je port 5432, ime naloga za replikaciju je foo, i šifra je foopass, admin može da doda sledeće informacije u postgresql.conf fajl koji se nalazi na pomoćnom serveru:

```
# The standby connects to the primary that is running on host
192.168.1.50

# and port 5432 as the user "foo" whose password is "foopass".

primary_conninfo = 'host=192.168.1.50 port=5432 user=foo
password=foopass'
```

2.4.2. Nadzor

An important health indicator of streaming replication is the amount of WAL records generated in the primary, but not yet applied in the standby. Može se izračunati ovo kašnjenje tako što će te uporediti trenutnu lokaciju za upis WAL-a na primarnom serveru sa poslednjom lokacijom WAL-a za upis koju je pomoćni server primio. Ova lokacije se mogu preuzeti putem pg_current_wal_lsn na primarnom serveru i pg_last_wal_receive_lsn na pomoćnom. Lokacija poslednjeg primljenog WAL-a se takodje nalazi u process statusu WAL receiver procesu, koji se prikazuje korišćenjem ps komande.

Možete preuzeti više otpremljenih WAL procesa korišćenjem pg_stat_replication prikaza. Ukoliko su velike razlike između pg_current_wal_lsn i sent_lsn polja, to može značiti da master server prima velike količine podataka, dok razlike između njih na pomoćnom serveru mogu ukazati na mrežno kašnjenje, ili da i pomoćni server prima velike količine podataka.

Kod hot pomoćnih servera, status WAL prijemnika se može preuzeti putem pg_stat_wal_receiver prikaza. Velike razlike između pg_last_wal_replay_lsn i received_lsn-a mogu ukazivati na to da se WAL prima pre nego što se može ponovo pokrenuti.

2.6. Slotovi replikacije

Slot replikacije predstavlja automatizovani način da se osigura da master server ne uklanja WAL segmente dok ih ne prime svi pomoćni serveri, i da master ne uklanja redove usled cijeg uklananja može doći do konflikta povraćanja, čak i kad je pomoćni server isključen.

Umesto slotova replikacije, moguće je da se spreči uklanjanje starih WAL segmenta korišćenjem wal_keep_segments, ili tako što će se segmenti čuvati u arhivi putem archive_command komande. Međutim, ove metode uglavnom čuvaju više WAL segmenta nego što je potrebno, dok slotovi replikacije čuvaju samo neophodne. Prednost ovih metoda je da ograničavaju prostorne zahteve pg_wal-a, što nije moguće uraditi putem slotova replikacije.

Takođe, hot_standby_feedback i vacuum_defer_cleanup_age obezbeđuju da vakuum ne briše bitne redove, međutim prva metoda nema efekta dok je pomoćni server isključen, a druga mora da se postavi na jako veliku vrednost da bi pružala adekvatnu zaštitu. Korišćenjem slotova replikacije izbegavate ove probleme.

2.6.1. Upiti i manipulacija slotova replikacije

Svaki slot replikacije ima ime, koji mora biti sastavljen od malih slova, brojeva, i donjih crta.

Slotovi koji su vec napravljeni se mogu videti u pg_replication_slots prikazu.

Oni se mogu napraviti ili odbacivati putem replikacije streama ili putem SQL funkcija.

2.6.2. Primer konfiguracije

Na primer, ovako se može napraviti slot replikacije:

```
postgres=# SELECT * FROM
pg_create_physical_replication_slot('node_a_slot');

  slot_name  | lsn

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

 node_a_slot |
```

```
postgres=# SELECT slot_name, slot_type, active FROM
pg_replication_slots;
```

```
  slot_name  | slot_type | active
-------------+-----------+--------
 node_a_slot | physical  | f
(1 row)
```

Da bi pomoćni server mogo da koristi ovaj slot, primary_slot_name treba biti ime pomoćnog servera. Kao na primer:

```
primary_conninfo = 'host=192.168.1.50 port=5432 user=foo
password=foopass'

primary_slot_name = 'node_a_slot'
```

2.7. Kaskadna replikacija

Ovakva replikacija omogućava pomoćnom serveru da prima replikacione veze i da šalje WAL zapise drugim pomoćnim serverima. Ovo se može iskoristiti da bi se smanjio broj direktnih veza na master i da se smanji čekanje na propusnost.

Pomoćni server koji je i posiljalac i primalac se naziva kaskadnim. Pomoćni serveri koji su direktno povezani na master se nazivaju upstream serveri, a oni koji su konektovani na master preko nekog drugog pomoćnog servera se nazivaju downstream serveri, Kaskadnareplikacija ne ograničava ni broj ni uredjenje downstream servera, i svaki pomoćni server je povezan na samo jedan upstream server koji je potom vezan na jedan master/primarni server.

Kaskadni pomoćni server pored WAL zapise koje prima sa mastera, šalje i WAL zapise koji su povraćeni iz arhiva. Tako da i kad replikaciona veza pukne, replikacioni stream nastavlja dalje dokle god postoje novi WAL zapisi.

Ukoliko se pomoćni upstream server unapredi u master, downstream serveri nastavljaju da šalju podatke sa novog mastera ukoliko je recovery_target_timeline postavljeno na latest (što je postavljeno po defaultu) an upstream standby server is promoted to become new master, downstream servers will continue to stream from the new master if `recovery_target_timeline` is set to `'latest'` (the default).

Da biste koristili ovu replikaciju, dozvolite pomoćnom kaskadnom serveru da prima replikacione veze (to jest postavite max_wal_senders i hot_standby i konfigurišite host-based authentication) Takođe se mora postaviti primary_conninfo tako da pokazuje na kaskadni server.

2.8. Sinhrona replikacija

PostgreSQL replikacija streama je asinhrona po defaultu. Ukoliko dođe do pada primarnog servera, može se desiti da neke komitovane transakcije završe na pomoćnom serveru, što dovodi do gubitka podataka. Količina gubitaka je proporcionalna replikacionom zastoju u trenutku oporavka.

Sinhrona replikacija potvrđuje da su se sve promene koje ja transakcija izvršila prenele na jedan ili više sinhrona pomoćna servera. Ovime se povećava normalno trajanje komita transakcije. Ovakav nivo replikacije se naziva 2-safe replikacija u teoriji nauke o kompjuterima, i group-1-safe (group-safe i 1-safe) kada se synchronous_commit postavi na remote_write

Kada se potraži sinhrona replikacija, svaka transakcija za upis čeka da se potvrdi da je komit poslat na WAL i primarnog i pomoćnog servera. Jedini način da dođe do gubitka podataka je ukoliko i primarni i pomoćni server puknu u isto vreme. Zbog toga ova replikacija ima jednu od najdurabilnijih sistema ukoliko sysadmin pažljivo postavi i upravlja serverima. To čekanje na potvrdu povećava samopouzdanje korisnika da neće doći do gubitka promena ukoliko dođe do pucanja, međutim ovime se povećava i vreme za odgovor zahtevane transakcije. Minimalno vreme čekanja je povratno vreme između primarnog i pomoćnog servera.

Transakcije koje su samo za čitanje i njeni rollbackovi ne čekaju na odgovor pomoćnog servera. Subtransakcioni komitovi ne čekaju na odgovor pomočnih servera, samo komitovi najvišeg nivoa. Radnje koje traju dugo kao učitavanje podataka ili pravljenje indexa ne čekaju na poslednju poruku komita. Svi komitovi koji su u 2 faze zahtevaju čekanje, čak i sami prepare i commit.

Sinhroni pomoćni server može da bude fizička replikacija pomoćnog ili pretplanika logičke replikacije. Takođe može biti bilo koji drugi fizički ili logički WAL potrošač koji ume da šalje odgovarajuće povratne poruke. Osim ugrađenog fizičkog i logičkog replikacionog sistema, ovime se pokrivaju i specialni programi kao pg_receivewal i pg_recvlogical kao i poneki third-party replikacioni sistemi i prilagođeni programi.

2.8.1. Planiranje za performanse

Da bi se aplikacije ponašale kako je predviđeno, potrebno je da se pažljivo izplaniraju i postave pomoćni serveri. Čekanje ne koristi sistemske resurse, ali transkacija se stopira sve dok se ne potvrdi izvršenje transfera. Ukoliko se sinhrona replikacija koristi nesmotreno, ona će pogoršati performanse aplikacija zbog povećanog vremena odziva.

PostgreSQL omogućava developerima da specificiraju nivo trajnosti pomoću replikacije. Ovo se može specificirati i za ceo sistem, kao i za posebne korisnike ili konekcije, čak i za posebne transakcije.

Na primer, 10% promena aplikacije su neki jako bitne korisničke informacije, dok su ostalih 90% manje bitni podaci koje ne bih napravile problem čak i kad bi došlo do njihovog gubljenja, kao na primer poruke između klijenata.

Pomoću sinhrone replikacije, možemo podesiti da se sinhrono repliciraju najbitnije promene, bez usporavanja celokupnog rada aplikacije. Ovakve opcije su jako bitne da bi se sinhrono replicirale aplikacije visokih performansi.

2.8.4. Planiranje za visoku dostupnost

`synchronous_standby_names` specifies the number and names of synchronous standbys that transaction commits made when `synchronous_commit` is set to `on`, `remote_apply` or `remote_write` will wait for responses from. Takvi komitovi se nikad neće izvršiti ukoliko dođe do pucanja makar jednog pomoćnog servera.

Kod se zahteva high availability baza, mora se imati onoliki broj sinhronih pomoćnih servera koliko je zahtevano.Ovo se može uraditi tako što ćete davati imena sinhronim serverima pomoću `synchronous_standby_names`.

U sinhronoj replikaciji koja je bazirana na prioritetu, pomoćni serveri koji se nalaze ranije u listi se koriste kao sinhroni pomoćni serveri. Pomoćni serveri koji su kasnije u listi preuzimaju njihovu ulogu ukoliko dođe do pada prvobitnih pomoćnih servera.

U sinhronoj replikaciji koja je bazirana na quorumu, svi pomoćni serveri u listi će se koristiti kao kandidati za sinhrone pomoćne servere. Čak iako dođe do padanja jednog od njih, drugi nastavljaju da vrše ulogu kanditata.

Kada se prvobitno poveže na primarni server, standby se ne može prikladno sinhronizovati. Ovo se naziva `catchup` stanje. Kada se lag između primarnog i pomoćnog servera totalno izgubi(dođe do 0), po prvi put se prelazi ne real-time `streaming` stanje. Vreme za koje će se catch-up obaviti je prilično veliko u trentku kada se napravi novi pomoćni server za sinhronu replikaciju. Ukoliko se pomoćni sinhroni server ugasi, catchup period se povećava u zavisnosti od vremena koje je proteklo dok je pomoćni server bio ugašen. Pomoćni server postaje sinhron kada dođe u `streaming stanje.` Ovo stanje se može videti korišćenjem `pg_stat_replication pogleda.`

Ukoliko se primarni server restartuje dok se čeka na izvršenje komitova, te transakcije će biti komitovane kada se baza podataka oporavi. Ne može se odrediti da su svi pomoćni serveri dobili sve WAL podatke koji nisu izvršeni u trenutku pada primarnog servera. Neke transakcije se mogu prikazati kao da nisu komitovane na pomoćnom, iako su komitovane na primarnom. Aplikacija neće potvrditi komit transakcije dok svi sinhroni pomoćni serveri ne prime te WAL podatke.

Ukoliko ne može da se obezbedi broj sinhronih pomoćnih servera koji je potreban, trebalo bi da se smanji broj sinhronih pomoćnih servera na čije odgovore transakcija mora da čeka u `synchronous_standby_names`(ili ih jednostavno isključite) i ponovo učitajte konfiguracioni fajl na primarnom serveru

Ukoliko je primarni server izolovan od ostalih pomoćnih servera, povratak bi trebao da se uradi na najboljeg kandidata preostalih pomoćnih servera.

Ako je potrebno da se ponovo napravi pomoćni server dok je transakcija u stanju čekanja, kada se pokreću pg_start_backup() i pg_stop_backup() mora se obezbediti da je `synchronous_commit` = `off`, inače će se čekati beskonačno dugo na pojavljivanje pomoćnog servera.

2.9. Continuous Archiving in Standby

Kada se koristi neprestano WAL arhiviranje, postoje 2 scenarija, WAL arhiva može da se deli između primarnog i pomoćnog servera, ili pomoćni može da ima sopstvenu WAL arhivu. Ukoliko je ima, `archive_mode` se postavlja na `always`, i pomoćni server će pozivati arhiviranu komandu za svaki WAL segment koji primi, bilo on primljen povraćanjem ili replikacijom streama. Deljena arhiva se može obrađivati na sličan način, ali `archive_command` komanda mora da testira dali postoji fajl koji se arhivira, i da li ima isti sadržaj. Na ovo mora da seobrati pažnja kada se koristi `archive_command`, i da se pazi da se ne overwriteuje fajl ukoliko već postoji, i da vrati potvrdnu poruku ukoliko se isti fajl arhivira dva puta.

Ukoliko se postavi `archive_mode` na on, arhiver se ne aktivira tokom povratka ili stanja čekanja. Ukoliko se pomoćni server unapredi u primarni, arhiviranje će početi nakon unapređenjha, ali neće da arhivira WAL sadržaj koji on sam nije generisao. Da bi dobili kompletni sadržaj WAL fajlora u arhivi, mora se obezbediti arhiviranje WAL-a pre nego što on dođe do pomoćnog servera. Ovo je sigurno tačno kod shippinga koje ja bazirano na fajlovima.(file-based log shipping), zato što pomoćni server može da povrati jedino one fajlove koji su u arhivi, dok to nije moguće ukoliko je aktivna streaming replikacija. Kada je server u režimu oporavka, ne postoji razlika između `on` i `always` modula.

3. Oporavak

Ako otkaže primarni server, pomoćni server treba da pokrene proceduru oporavka.

Ako pomoćni server otkaže, nije potrebno vršiti oporavka. Ukoliko pomoćni server može biti ponovo pokrenut, čak i nakon odredjenog vremena, proces ponovnog pokretanja može biti pokrenut blagovremeno, koristeći sistem za oporavak nakon restartovanja. Ako nije moguće restartovati pomoćni server, kreira se potpuno nova instanca pomoćnog servera.

Kada pomoćni server zameni primarni, a nakon toga se prethodni primarni restartuje, mora se obezbediti mehanizam obaveštavanja da taj server nije više primarni. Ovaj sistem se naziva STONITH(Shoot The Other Node In The Head), koji je neophodan za izbegavanje situacije gde oba sistema misle da su primarni, što može dovosti do nevalidnih podataka.

Mnogi sistemi sa oporavkom koriste samo dva sistema replikacije, primarni i pomoćni, povezani mehanizmom otkucaja koji potvrdjuje konekciju između njih i dostupnost primarnog. Moguće je koristiti dodatni sistem (sistem svedok) da spreči neodgovarajući oporavak, ali dodatna kompleksnost ne mora biti potrebna ukoliko nije adekvatno podešena i istestirana.

PostgreSQL ne poseduje sistem za identifikovanje otkaza na primarnom serveru i obaveštavanja pomoćnog. Mnogi alati se mogu iskoristiti za tu svrhu, poput IP address migration.

Kada se izvrši oporavak, postoji samo jedan server u klasteru. Ovo se naziva degenerativno stanje. Prvobitni pomoćni server postaje primarni, ali je prvobitni primarni server nedostupan i moguće je da se ne pokrene. Kako bi se vratili u normalno stanje, pomoćni server mora biti ponovno kreiran. **pg_rewind** se može iskoristiti da se ubrza ovaj proces na većim klasterima.

Proces zamene primarnog servera može biti izuzetno brzo, ali ponovno pripremanje pomoćnog servera je vremenski zahtevno. Preporučuje se povremeno menjanje primarnog i pomoćnog servera, šta može služiti kao test mehanizma oporavka kada se otkaz zaista desi.

Kako bi pokrenuli taj sistem, postoji više načina:

1. pg_ctl promote
2. pg_promote
3. trigger file

Ako podešavamo servere za reportove koji se koriste samo kao read-only, a ne za visoku dostupnost, nema potrebe izvršavati promote.

4. Alternativni metodi za dostavu logova

Alternativa za ugradjen pomoćni server je korišćenje **restore_command** koja polluje arhiviranu lokaciju. Ovo je bila jedina opcija u ispod verzije 8.4.

U ovom modu, server primenjuje WAL nad jednim po jednim fajlom, pa ako se koristi pomoćni server za upite, postoji određeno kašnjenje kada akcija u masteru (primarnom) bude vidljiva u pomoćnom, odgovarajući vremenu za upis u WAL. **archive_timeout** odredjuje taj interval.

Operacije koje se izvršavaju nad svim serverima su kontinualna arhiviranja i zadaci oporavka. Jedino mesto kontakta izmedju različitih servera je arhiva WAL fajla koju oni dele: primarni upisuje u arhivu, pomoćni čita iz nje. Mora se osigurati da WAL arhive na različitim primarnim serverima ne budu pomešane. Arhiva ne mora biti velika ako se samo koristi za pomoćne operacije.

Povezani rad slabo povezanih servera je **restore_command,** koja kada se izvrši pomoćnom serveru, kada traži naredni WAL fajl, čeka da on bude dostupan na glavnom. Normalan proces oporavka bi tražio fajl iz WAL arhive, gde bi nastala greška ukoliko fajl nije dostupan. U ovom slučaju očekivano je da je WAL fajl nedostupan, pa pomoćni server čeka dok on ne postane dostupan. Fajlovi koji imaju ekstenziju .history ne moraju da se čekaju, a mora se vratiti non-zero kod. Čekajuća **restore_command** može biti napisana kao skripta koja se ponovno izvršava nakon pribavljanja dostupnosti WAL fajla. Mora postojati način za pokretanje oporavka, koji će prekinuti **restore_command**, stopiratu petlju i vratiti file_not_found grešku. Ovo okončava oporavak i pomoćni server postaje primarni.

Primer restore_command

```
triggered = false;
while (!NextWALFileReady() && !triggered)
{
    sleep(100000L);         /* wait for ~0.1 sec */
    if (CheckForExternalTrigger())
        triggered = true;
}
if (!triggered)
        CopyWALFileForRecovery();
```

Metod za izvršenje oporavka je veoma bitan deo planiranja i dizajniranja baze. Jedna opcija je **restore_command.** Izvršava se jednom za svaki WAL fiajl, ali proces koji je izvršava se kreira i uništava za ssvaki fajl, pa ne postoji daemon ili serverski proces, tako da se signali ili

hendleri signala ne mogu koristiti. **restore_command** nije pogodna kao triger oporavka. Može se koristiti timeout,, posebno ako se iskoristi sa poznatim atributom **archive_timeout.** Jedan od nedostataka ovakvog pristupa je da zagušenje mreže može inicirati proces oporavka. Mehanizam obaveštavanja poput eksplicitnog kreiranja trigger fajla je idealan način, ako se može koristiti.

4.1. Implementacija

Procedura za konfigurisanje pomoćnog server je sledeća:

1. Konfigurisati pomoćni i primarni server što je moguće sličnije, uključujući i istu kopiju PostgreSQL.
2. Podesiti kontinualno arhiviranje iz primarnog u folder WAL arhive na pomoćnom serveru. **archive_mode**, **archive_command** i **archive_timeout** atributi treba da budu odgovarajuže podešeni na primarnom serveru.
3. Napraviti kopiju baze sa primarnog servera i učitajte podatke na pomoćni server.
4. Pokrenite proces oporavka na pomoćnom serveru iz lokalne WAL arhive, koristeći prethodno upisanu **restore_command**.

Oporavak tretira WAL arhivu kao read-only, tako da kada se WAL file iskopira na pomoćni server može biti višestruko iskopiran dok se čita od strane pomoćnog servera. Na ovaj način, pomoćni server se može iskoristiti za veliku dostupnost pošto se fajlovi dugotrajno čuvaju za svrhu oporavka.

4.2. Dostava logova baziranim na rekordima

Moguće je implementirati log shipping na osnovu recorda koristeći ovaj alternativni metod, ali je neophodan dodatni razvoj, a promene će biti vidljive tek kad se kompletan WAL fajl isporuči.

Moguće je pozvati pg_wallfile_name_offset() funkciju da nadjemo ime fajla i pomeraj u bajtovima u njemu sa trenutnog kraja WAL. Nakon toga se može direktno pristupiti WAL fajlu i iskopirati podatke sa zadnjeg poznatog kraja do trenutnog kraja na pomoćne servere. Na ovakav način, nevalidni podaci mogu nastati u vremenu pribavljanja kopiranog programa, koje može biti izuzetno malo, i nema nepotrebnog forsiranja dela fajlova da se arhiviraju. **restore_command** pomoćnog servera radi samo sa kompletnim WAL fajlovima, tako da se inkrementalno kopiranje može koristiti jedino u slučaju otkaza primarnog servera. Neophodno je pravilno komuniciranje **restore_command** sa sistemom za kopiranje podataka.

Počevši od verzije 9.0, moguće je koristiti replikaciju tokova (streaming replication) kako bi postigli ovu mogućnost na jednostavniji način

5. Hot Standby

Hot Standby je termin koji se koristi za sposobnost pokretanja read-only upita na serveru koji je u standby ili oporavak modu. Korisno je i za replikaciju i za obnavljanje podataka sa visokom preciznošću. Izraz Hot Standby se odnosi i na sposobnost servera da menja svoj mod iz obnove u normalan, a da je to potpuno transparetno na strani korisnika

Izvršenje upita u hot standby modu je nalik na normalne operacije, a razlike su opisane u nastavku ovog rada.

5.1. Korisnicki pregled

Kada je hot_standby atribut setovan na true na standby serveru, prihvatiće konekcije kada je sistem doveden u konzistentno stanje. Takve konekcije su isključivo za čitanje, čak se ne može upisivati ni u pomoćnim tabelama.

Podaci na standby serveru dolaze nakon nekog vremena sa primarnog pa može doći do kašnjenja. Izvršenje istog upita istovremeno na primarnom i standby serveru može vratiti različite vrednosti. Takvi podaci na standby su eventualno konzistentni sa primarnim. Kada se rezultat transakcije prosledi na standby, izmene te transakcije ce biti vidljive u svim narednim snapshotovima nad standby serverom.

Transakcije pokrenute tokom hot standby mogu izvršavati sledeće komande:

- Upit pristupa - `SELECT`, `COPY TO`
- Kursor komande - `DECLARE`, `FETCH`, `CLOSE`
- Parametri - `SHOW`, `SET`, `RESET`
- komande za upravljanje transakcijama
    - `BEGIN`, `END`, `ABORT`, `START TRANSACTION`
    - `SAVEPOINT`, `RELEASE`, `ROLLBACK TO SAVEPOINT`
    - `EXCEPTION` blocks and other internal subtransactions
- `LOCK TABLE`, samo u modu: `ACCESS SHARE`, `ROW SHARE` or `ROW EXCLUSIVE`.
- Planovi i resursi - `PREPARE`, `EXECUTE`, `DEALLOCATE`, `DISCARD`
- Plugini i ekstenzije- `LOAD`
- `UNLISTEN`

Transakciji pokrenutoj tokom hot standby nikad neće biti dodeljen ID i ne mogu upisivati u sistemski WAL.

Sledeće akcije vraćaju grešku:

- Data Manipulation Language (DML) - `INSERT`, `UPDATE`, `DELETE`, `COPY FROM`, `TRUNCATE`. Nisu dozvoljene akcije čiji je rezultat triger koji se izvršava tokom oporavka. Ovo ograničenje se odnosi i na pomoćne tabele, zato što se redovi tabela ne mogu čitati ili upisivati ako im nije dodeljen transakcijski ID, što je zabranjeno u Hot Standby okruženju.
- Data Definition Language (DDL) - `CREATE`, `DROP`, `ALTER`, `COMMENT`. Ovo ograničenje važi i za pomoćne tabele, zato što izvršenje ovih operacija zahteva ažuriranje sistemskog kataloga tabela.
- `SELECT ... FOR SHARE | UPDATE`, zato sto zaključani redovi ne mogu uzimate bez ažuriranje podataka.
- Pravila nad `SELECT` naredbom koja generišu DML komande
- `LOCK` koja zahteva mod iznad `ROW EXCLUSIVE MODE`.
- `LOCK` defaultna, pošto zahteva `ACCESS EXCLUSIVE MODE`.
- Komande za rad sa transakcijama koje eksplicitno podešavaju stanje koje nije isključivo za čitanje
    - `BEGIN READ WRITE`, `START TRANSACTION READ WRITE`
    - `SET TRANSACTION READ WRITE`, `SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE`
    - `SET transaction_read_only = off`
- 2-fazni komit - `PREPARE TRANSACTION`, `COMMIT PREPARED`, `ROLLBACK PREPARED` zato što i komande isključivo za čitanje upisuju u WAL u fazi pripreme
- Sekvencijala ažuriranja - `nextval()`, `setval()`
- `LISTEN`, `NOTIFY`

U normalnom režimu, read-only transakcije dozvoljavaju LISTEN i NOTIFY, tako da Hot Standby sesije rade u ograničenijem režimu u odnosu na normalne read-only sesije.

Tokom hot standby, parametar `transaction_read_only` i ne može biti modifikovan. Medjutim, dok god se ne izvršava naredba koja modifikuje bazu, konekcija se ponaša kao da je u normalnom režimu. Ako se desi oporavak ili promena režima, baza prelazi u normalan mod. Sesije ostaju konektovane dok server menja svoj mod. Kada se hot standby završi. moguće je izvršiti read-write transakcije čak i ako je sesija počela tokom hot standby-a.

Korisnik može videti u kom se modu nalazi sesija izvršenjem `SHOW transaction_read_only`. Moguće je pristupiti i informacijama za standby server, tako da je moguće napraviti programe koji su svesni stanja baze koji se mogu koristiti za nadgledanje i oporavak jer PostgreSQL nema interno napravljene mehanizme.

5.2. Obrada konflikta

Akcije na primarnom serveru se prenose na standby server. Kao rezultat, moguće su negativne interakcije ili konflikti izmedju njih. Najbolji primer su performanse, ako se unosi

velika količina podatak na primarnom, identičan stream WAL unosa se generiše na standby, tako da se standby upiti mogu boriti za sistemske resurse , poput I/O.

Postoji više tipova konflikta koji mogu nastupiti. Ovi konflikti su *hard* konflikti u smislu da mogu biti otkazani, čak i da se cela sesija okonča kako bi se razrešili. Ovi slučajevi uključuju:

- Pristup ekskluzivno zakljucanim resursima koje koristi primarni server, uključujući LOCK komande i razne DDL naredbe
- Odbacivanje tablespace na primarnim konfliktima sa standby upitima koji koriste taj tablespace.
- Odbacivanje baze kada se primarni server sukobljava sa sesijama koje su povezane sa bazom na standby serveru.
- Primena usisavanja rekorda sa WAL se sukobljava sa standby transakcijama čiji snapshotovi vide redove koji će biti obrisani.
- Primena usisavanja rekorda sa WAL koji se sukobljavaju sa upitima koji pristupaju odgovarajućoj stranici na standby, nebitno da li su podaci koji su obrisani vidljivi.

Na primarnom serveru, ovi slučajevu rezultiraju kao period čekanja gde korisnik bira da li će da otkaže neku od suprotstavljenih akcija. Medjutim, na standby-u nema izbora, WAL akcija se desila na primarnom i standby mora da je izvrši. To može da izazove da standby sve više i više zaostaje za primarnim, to se rešava mehanizmom koji nasilno prekida sve standby upite koji zahteva WAL rekorde koji ce biti primenjeni.

Primer je kada se na primarnom serveru pokrene DROP TABLE na tabeli nad kojom se vrši upit u standby serveru. Standby mora da delayuje primenu WAL rekorda i svega nakon njih, ili da otkaže problematične upite tako da se izvrši prvo DROP TABLE.

Kada je upit kratak, dozvoljava se da se izvrši kašnjenjem, ali dugo čekanje nije poželjno. Parametri max_standby_archive_delay i max_standby_streaming_delay odredjuju maksimalno dozvoljeno kašnjenje u WAL. Upiti čije je kašnjenje duže od definisanog će biti otkazani.

U slučaju kada standby server postoji samo za visoku dostupnost, najbolje je da su delay parametri što kraći, tako da standby server ne zaostaje za primarnim. Ako standby izvršava upite koji se dugo izvršavaju, poželjno je staviti što duže kašnjenje.

Kada se vrednosti `max_standby_archive_delay` ili `max_standby_streaming_delay` nadmaše, upiti će biti prekinuti. Rezultat je greška, ako je komanda DROP DATABASE otkazuje se cela sesija.

Otkazani upit može biti ponovo pokrenut čim se napravi nova transakcija. Zato što otkaz upita zavisi od WAL rekorda koji se ponovo pokreće, ponovljeni upit može biti izvršen kada se ponovo pokrene.

Najčešći razlog za konflikte je “early cleanup”. PostgreSQL dozvoljava brisanje zastarelih verzija redova kada nema transakcija koje moraju da ih vide prema MVCC pravilima. Ovo

važi jedino za primarni server, a kada se tamo izvrši, moguće je da su obrisani redovi potrebni na standby serveru.

Jedna od opcija je da se podesi parametar `hot_standby_feedback` koji sprečava `VACUUM` da briše zastarele redove tako da konflikti pri brisanju neće nastupiti. Ovo će odložiti zastarelo brisanje, pa je moguće postojanje prevelikih tabela.

Druga opcija je da se poveća vacuum_defer_cleanup_age, tako da se zastareli redovi brišu tek kada prodje duže vreme. Ovo daje više vremena upitima na standby serveru za izvršenje, bez podešavanja visokog `max_standby_streaming_delay.`

Broj otkazan upita i razlozi zašto su otkazani se mogu videti `pg_stat_database_conflicts` na standby serveru.

5.3. Hot Standby parametri

Na primarnom, parametri wal_level i vacuum_defer_cleanup_age se mogu koristiti. max_standby_archive_delay i max_standby_streaming_delay nemaju efekat kada su podešeni na primarnom.

U standby, parametri hot_standby, max_standby_archive_delay i max_standby_streaming_delay se mogu koristiti. vacuum_defer_cleanup_age nema uticaj dok god se server nalazi u modu standby, ali postaje bitan kada standby postane primaran.

5.4. Ograničenja

Postoji više ograničenja :

- Potpuno poznavanje transakcije u izvršenju je neophodno pre nego što se snapshot može napraviti. Transakcije koje sadrže veliki broj podtransakcije(više od 64 trenutno) odlažu početak read-only konekcija dok se ne završi najduža write transakcije. Poruka o ovome se šalje u server log
- Validne početne tačke za standby upite su generisanje na svakom checkpointu na masteru. Ako je standby ugašen dok je master ugašen, nije moguće ponovno ući u Hot Standby dok se ponovno ne pokrene primarnu, tako da generiše naredne checkpointe u WAL logu. Ova situacije nije problem u većini situacija gde može nastupiti.
- Na kraju oporavka, `AccessExclusiveLocks` koji je zadržan od strane pripremljenih transakcije će zahtevati duplo veći broj zaključavanja unosa u tabele. Ako izvršavate veliki broj konkurentnih koji koriste **AccessExclusiveLocks** ili jedna velika transakcija koja koristi više AccessExclusive Locks, povećajte duplo atribut `max_locks_per_transaction`. Preskočite ovaj korak ukoliko je `max_prepared_transactions` 0.
- Nije moguće setovati transakciju u nivo serijalizovane izolacije trenutno.

6. Primer Hot Standby podesavanja

Podesavanje postgresql.conf fajla da podrzi hot standby na primarnom

```
  GNU nano 2.0.6                  File: postgresql.conf                                Modified

postgresql.conf

 # - Connection Settings -
listen_addresses = '*'                          # what IP address(es) to listen on;

 # - Settings -
 # minimal, archive, hot_standby, or logical
wal_level = hot_standby
archive_mode = on               # enables archiving; off, on, or always

archive_command = 'cp %p /links/groups/itsc/postgres_wal_logs/todor/%f'

 # Podesiti na svim serverima koji prosledjuju podatke replikacije
max_wal_senders = 5             # max number of walsender processes

wal_keep_segments = 32          # in logfile segments, 16MB each; 0 disables

# Master server ignorise ovo podesavanje
hot_standby = on                        # "on" allows queries during recovery
```

Pravljenje role za replikaciju

```
Todors-MacBook-Pro:~ todor$ psql -d postgres
psql (12.3)
```

```
postgres-# CREATE ROLE replication WITH REPLICATION PASSWORD 'testpass' LOGIN
```

Host adresa za replikaciju mora biti eksplicitno setovana

```
                                        nano                                          ⌥⌘1
  GNU nano 2.0.6                  File: pg_hba.conf                                  Modified

# TYPE  DATABASE        USER            ADDRESS                 METHOD
# IPv4 local connections:
host    replication     replication     192.168.1.14/32         md5
```

Napravite backup sa primarnog na stadby server:

```
Todors-MacBook-Pro:~ todor$ STANDBY=test-bs
Todors-MacBook-Pro:~ todor$ PGDATA=/var/lib/pgsql/12/data
Todors-MacBook-Pro:~ todor$ sudo -u postgres psql -c "SELECT pg_start_backup('label',true)"
```

```
Todors-MacBook-Pro:~ todor$ rsync -ac --delete ${PGDATA}/ ${STANDBY}:${PGDATA} --exclude postmaster.pid --
exclude pg_hba.conf --exclude recovery.done
```

```
Todors-MacBook-Pro:~ todor$ sudo -u postgres psql -c "SELECT pg_stop_backup()"
```

postgresql.conf na standby je identican kao i master.
recovery.conf:

pg_hba.conf u slucaju da primarni otkaze pa standby postaje master:

Primer:

-na masteru:

```
psql -U postgres -c "SELECT pg_current_xlog_location()"
```

-na replici:

```
psql -U postgres -c "select pg_last_xlog_receive_location()"
```

Rezultat:

master:

```
ps -ef | grep postgres
Result: postgres 11866 11833  0 15:14 ?        00:00:00
postgres: wal receiver process   streaming 0/F7000140
```

replika:

```
ps -ef | grep postgres

Result: postgres 11866 11833  0 15:14 ?        00:00:00
postgres: wal receiver process   streaming 0/F7000140
```

7. Zakljucak

Replikacija baze podataka je jedan od osnovnih mehanizama za rešavanje problema prilikom otkaza servera baze podataka. PostgreSQL poseduje više tipova replikacije, a na korisniku je da odabere sistem koji će mu najbolje odgovarati, u zavisnosti od toga šta mu je cilj. Glavni problem prilikom replikacije je sinhronizacija podataka izmedju dva servera baze, kao i način obaveštavanja i oporavka u slučaju da jedan od tih servera otkaže. Pravilna konfiguracija primarnog i pomoćnih servera je ključna, jer nepravilno podešavanje može dovesti do netačnih podataka koji se pribavljaju iz baze. Replikacija može služiti i kao podrška za load balancing, kada se rasporedjuju upiti ka bazi na više servera, koji mogu prihvatati samo read-only upite.

U radu je prikazan teoretski pregled više tipova replikacije, i praktična implementacija Hot Standby režima u test okruženju sa dva servera.

8. Reference

1. https://www.postgresql.org/docs/12/high-availability.html
2. https://wiki.postgresql.org/wiki/Hot_Standby
3. https://linuxconfig.org/how-to-create-a-hot-standby-with-postgresql