replikacja w postgreSQL

Jakub Biernacki
Bartek Lubikowski
Paweł Lesiecki

Log-Shipping Standby Servers

Log-Shipping Standby Servers to metoda replikacji, w której serwery zapasowe (*standby or slave servers*) polagają na przetwarzaniu plików będącymi logami generowanymi przez serwer główny (*primary or master server*).

Ciągła archiwizacja może być używana do tworzenia klastrów wysokiej niezawodności składających się z jednego (lub więcej) serwera zapasowego, który przejmie rolę serwera głównego w przypadku jego awarii. Serwer główny oraz serwery zapasowe pracują wspólnie w celu zapewnienia wysokiej niezawodności (*high availabilty*). Główny serwer działa w ciągłym trybie archiwizacji a każdy zapasowy serwer odtwarza stan głównego serwera z plików WAL (*write-ahead log*). Rozwiązanie to nie wymaga żadnych zmian w tabelach bazy danych, z tego powodu jest ono łatwe i szybkie do wdrożenia w porównaniu do innych rozwiązań, a także posiada bardzo mały wpływ na spadek wydajności głównego serwera. Przesyłanie pliku WAL jest łatwe i szybkie pomiędzy serwerami niezależnie od odległości, która je dzieli. Przepustowość łącza potrzeba do przesyłania plików zależy od obciążenia serwera głównego.

Przesyłanie plików WAL jest asynchroniczne, to znaczy, pliki przesyłane są po zaakceptowaniu transakcji (*commit*). Z tego powodu istnieje prawdopodobieństwo utraty danych transakcji nie przesłanych przed awarią. Zmniejszenie ilości danych możliwych do utracenia można zminimalizować ustawiając parametr **`archive_timeout`** parameter na niską wartość, bądź stosując replikację korzystającą z przesyłania strumieniowego.

Czas, który mija pomiędzy aktywacją serwera zapasowego do jego pełnego działania jest bardzo krótki, w porównaniu do odzyskiwania serwera z kopii zapasowej. Z tego powodu *warm standby* oferuję bardzo wysoką dostępność, nie osiągalną przy zwykłych kopiach zapasowych bazy danych. Serwer zapasowy może być także używany do odczytywania danych. Takie rozwiązanie nosi nazwę *hot standby*.

Planowanie replikacji

Serwer główny oraz serwery zapasowe powinny być możliwie podobnie skonfigurowane. Ścieżki do miejsca gdzie przechowywane są tabele na dysku są przekazywane niezmodyfikowane pomiędzy serwerami i muszą być takie same. Gdy polecenie **`CREATE TABLESPACE`** jest wykonywane musi zostać utworzony punkt montowania na masterze oraz na

każdym slavie przed wykonaniem tego polecenia. Sprzęt zastosowany w serwerze nie musi być taki sam lecz, z praktyki wynika, że łatwiej skonfigurować dwa takie same serwery. Serwery muszą być identyczne jeśli chodzi o architekturę. Dane przesłane z serwera 32-bit nie będą działać na serwerze 64-bitowym i vice versa.

Przesyłanie pliku ze zmianami pomiędzy różnymi wydaniami głównymi PostgreSQLa nie jest możliwe, natomiast z powodu niezmieniania formatu zapisu danych pomiędzy różnymi wydaniami tej samej wersji jest to możliwe lecz nie jest w żaden sposób wspierane ani gwarantowane. W przypadku aktualizacji wersji w pierwszej kolejności zaktualizowane powinny być serwery zapasowe a następnie serwer główny. Wynika to z faktu, że jest większe prawdopodobieństwo, że nowsza wersja przeczyta plik WAL ze starszej niż na odwrót.

Praca serwera zapasowego

Serwer zapasowy bez przerwy wprowadza zmiany otrzymane w WAL z serwera głównego. Serwer zapasowy może odczytywać WAL bezpośrednio z archiwum lub bezpośrednio z serwera głównego poprzez połączenie TCP (jeśli zastosowana jest replikacja strumieniowa). W przypadku restartu serwera wprowadza zmiany znalezione we wszystkich WAL w katalogu **`pg_xlog`**.

Przy starcie serwera zapasowego w pierwszej kolejności przywracane są dostępne WAL z archiwum. Następnie wykonywane jest polecienie zdefiniowane parametrem **`restore_command`**, jeśli wywołanie polecenia się nie powiedzie przywracane są WAL znalezione w folderze **`pg_xlog`**. Jeśli to też się nie powiedzie i włączona jest replikacja strumieniowa serwer zapasowy stara się połączyć z serwerem głównym i zaczyna wysyłać WAL od ostatniego poprawnego znalezionego w folderze **`pg_xlog`**. Jeśli wyłączona jest replikacja strumieniowa lub się nie powiedzie, procedura zaczyna się od nowa.

Serwer zapasowy zmienia tryb na pracę jako normalny serwer natychmiast gdy **`pg_ctl promote`** jest uruchomione lub znaleziony zostaje plik skonfigurowany parametrem **`trigger_file`**. Przed uruchomieniem wszystkie WAL dostępne dla serwera zapasowego są przywracane.

Przygotowywanie serwera głównego

Konfiguracja ciągłej archiwizacji na serwerze głównym powinna zapisywać do lokalizacji dostępnej dla serwera zapasowego nawet po wyłączeniu serwera głównego czyli na serwerze zapasowym lub innym zaufanym serwerze pośredniczącym.

Jeśli włączona zostaje replikacja strumieniowa uwierzytelnianie na serwerze głównym powinno pozwolić serwerom zapasowym na połączenie. Należy utworzyć rolę oraz odpowiednie wpisy w **`pg_hba.conf`**. Należy się upewnić także, że **`max_wal_senders`** jest ustawiony na odpowiednio dużą wartość. Należy utworzyć początkową kopię do pierwszej konfiguracji serwer zapasowego używając polecenia: **`SELECT pg_start_backup('label');`** Zrzut bazy należy umieścić w folderze wskazywanym w komendzie przechowywanej w parametrze **`restore_command`**.

Przygotowywanie serwera zapasowego

Konfiguracje serwerów zapasowych do przywrócenia danych z pierwszej kopii pobranej z serwera głównego zaczyna się od stworzenia w pliku **`recovery.conf`** komendy zapisanej pod parametrem **`recovery_command`** oraz ustawienia parametru **`standby_mode`** na **`on`**. Komendę **`restore_command`** ustawiamy jako zwykłe kopiowanie plików z archiwum WAL. W przypadku posiadania kilku serwerów zapasowych dla wysokiej niezawodności należy ustawić parametr **`recovery_target_timeline`**.

W przypadku konfigurowania replikacji strumieniowej, należy podać parametry połączenia w parametrze **`primary_conninfo`** zawierające nazwę hosta (lub adres IP) oraz port, użytkownika i hasło.

Jeśli serwer zapasowy jest konfigurowany do zapewnienia wysokiej niezawodności należy ustawić archiwizajcę WAL, połączenia oraz uwierzytelnianie tak jak na serwerze głównym ponieważ serwer zapasowy zacznie działać jak serwer główny po awarii.

Jeśli używane jest archiwum WAL jego rozmiar może być zmniejszony poprzez konfigurację parametru **`archive_cleanup_command`** aby usuwane były pliki nie potrzebne więcej przez serwer zapasowy. Do tego celu stworzone jest narzędzie **`pg_archivecleanup`** w przypadku konfiguracji z jednym serwerem zapasowym.

Przykładowy plik **`recovery.conf`**:

```
standby_mode = on
restore_command = 'cp /path/to/archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /path/to/archive %r'
```

Ilość serwerów zapasowych jest nieograniczona lecz należy ustawić parametr **max_wal_senders** na odpowiednią wartość aby zapewnić im możliwość równoczesnego połączenia.

Replikacja strumieniowa

Replikacja strumieniowa pozwala serwerom zapasowym na bycie bardziej aktualnym niż w przypadku rozwiązania opartego na przesyłaniu całych plików WAL. Serwer główny wysyła rekordy WAL do podłączonych serwerów zapasowych bez oczekiwania na zapełnienie pliku WAL.

Istnieje pewne opóźnienie zanim zmiany na serwerze głównym staną się widoczne na serwerach zapasowych lecz to opóźnienie jest możliwie małe w porównaniu do rozwiązania opartego na przesyłaniu całych plików. Opóźnienie wynika z asynchroniczności replikacji strumieniowej. Opóźnienie to wynosi zazwyczaj mniej niż sekundę.

Uwierzytelnianie

Z powodu przysyłania w strumieniu WAL ważnych informacji o przywilejach ważne jest aby dać dostęp do replikacji tylko bardzo zaufanym użytkownikom. Serwery zapasowe muszą uwierzytelniać się do serwera głównego z przywilejami **REPLICATION** oraz **LOGIN**. Uwierzytelnianie klienta do replikacji kontrolowane jest przez wpisy w **pg_hba.conf**. Przykładowo jeśli serwer zapasowy jest na adresie *192.168.1.100* a nazwa konta do replikacji nazywa się *kowalski*, administrator systemu może dodać taki wpis :

```
host    replication    kowalski    192.168.1.100/32     md5
```

Nazwa hosta, numer portu serwera głównego, nazwa użytkownika oraz hasło są ustalane w pliku **recovery.conf** na serwerze zapasowym. Hasło może być przechowywane także w pliku **~/.pgpass**. Przykładowo jeśli adres IP główny serwera to *192.168.1.50*, porcie *5432* nazwa konta do replikacji to *foo* natomisat hasło to *bar* w pliku

recovery.conf należy dodać następujący wpis:

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

Hot Standby

Hot Standby, czyli “gorąca gotowość” oznacza w Postgresie możliwość nawiązania połączenia z serwerem w trybie czytającym (*read-only*) podczas gdy serwer pracuje jako serwer zapasowy (lub jest w procesie przywracania systemu po upadku serwera głównego). Serwer operujący w tym trybie nie powoduje rozłączenia aktywnych użytkowników podczas procesu przywracania systemu. Odpytywanie serwera działającego w trybie “gorącej gotowości” jest podobne do tego działającego “normalnie”, są jednak pewne ograniczenia, które przedstawione są poniżej.

Punkt widzenia użytkownika

Kiedy parametr **hot_standby** ma wartość **TRUE** (**ON** lub **1**) na serwerze zapasowym, serwer ten będzie pozwalał na połączenia tylko wtedy gdy przywracanie systemu zapewni jego pełną spójność. Wszystkie połączenia z serwerem pracującym w trybie hot standby pozwalają wyłącznie na czytanie (*read-only connections*). Transakcje na serwerze opracującym w tym trybie mogą zawierać więc następujące komendy:

- dostępowe: **SELECT, COPY TO**
- związane z kursorem: **DECLARE, FETCH, CLOSE**
- związane z parametrami serwera: **SHOW, SET, RESET**
- związane z samymi transakcjami: **BEGIN, END, ABORT, START TRANSACTION, SAVEPOINT, ROLLBACK TO SAVEPOINT**, bloki **EXCEPTION** i inne...
- **LOCK TABLE** (w jednym z trybów **ACCESS SHARE**, **ROW SHARE**, **ROW EXCLUSIVE**)
- **PREPARE**, **EXECUTE**, **DEALLOCATE**, **DISCARD**
- związane z pluginami: **LOAD**

Transakcje na serwerze zapasowym pracującym w trybie *hot standby* nie mają przypisanego ID, nie mogą zatem tworzyć plików WAL. Dlatego następujące komendy spowodują błąd:

- komendy DML (**INSERT**, **UPDATE**, **DELETE**, **COPY FROM**, **TRUNCATE**)
- komendy DDL (**CREATE**, **DROP**, **ALTER**, **COMMENT**)

- **SELECT ... FOR SHARE | UPDATE**
- komendy **LOCK** która jawnie żąda trybu wyższego niż **ROW EXCLUSIVE MODE**.
- komendy **LOCK** bez jawnego żadania odpowiedniego trybu (gdyż domyślnie jest to **ACCESS EXCLUSIVE MODE**)
- kommend związanych z transakcjami które jawnie wprowadzją stan *non-read-only*:
    - **BEGIN READ WRITE**, **START TRANSACTION READ WRITE**
    - **SET TRANSACTION READ WRITE**, **SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE**
    - **SET transaction_read_only = off**
- dwófazowych komitów - PREPARE **TRANSACTION**, **COMMIT PREPARED**, **ROLLBACK PREPARED** ponieważ wymaga to tworzenia plików WAL, co jest w tym trybie niemożliwe.
- komend związanych ze zmianą stanu obiektów typu **SEQUENCE**: **nextval()**, **setval()**
- **LISTEN**, **UNLISTEN**, **NOTIFY** (“normalne” transkacje read-only mogą uaktualniać obiekty typu **SEQUENCE** i używać komend **UNLISTEN** i **NOTIFY**, w przypadku połączania z serwerem *hot standby* transakcje te są jeszcze bardziej restrykcyjne – możliwe, że ograniczenie to zniknie w przyszłych wersjach).

Ponadto, podczas pracy w trybie *hot standby* parametr `transaction_read_only` jest zawsze równy `TRUE` i wartość ta nie może być zmieniona. Ogólnie, gdy tylko czytamy, baza zachwuje się dokładnie tak samo jak w przypadku standardowego połączenia. Jeśli nastąpi *failover* (czyli “padnie” serwer główny i serwer zapasowy będzie chciał przejąć rolę mastera) lub jawna zmiana z serwera zapasowego na serwer głów (np. w celu przetestowania czy nasza replikacja działa poprawnie) baza zacznie pracować w normalnym trybie (możliwe będzie wykonywanie transakcji, które modyfikują bazę – nawet te, rozpoczęte w trybie *hot standby*).

Punkt widzenia administratora

By serwer mógł pracować w trybie *hot standby* administrator systemu musi ustawić parametr **hot_standby** na **TRUE** oraz dostarczyć plik konfiguracyjny **recovery.conf**. Serwer pracujący w trybie *hot standby* będzie akceptował połączenia tylko wtedy, gdy znajdzie się w spójnym stanie - każda próba nawiązywania połączenia, gdy baza nie jest w spójna skutkować będzie błędem. By dowiedzieć się czy po przywracaniu systemu (*system recovery*) znajduje się on w spójnym stanie możemy zajrzeć do logów:

```
LOG:  entering standby mode
```

```
... then some time later ...

LOG:  consistent recovery state reached
LOG:  database system is ready to accept read only connections
```

Niektóre z parametrów (**max_connections**, **max_prepared_transactions**, **max_locks_per_transaction**) serwera zapasowego pracującego w trybie *hot standby*, muszą być przekonfigurowane zgodnie z konfiguracją serwera głównego. Wartości tych parametrów muszą być przynajmniej równe, w przeciwnym razie serwer zapasowy nie wystartuje (nie będzie spełniał swojej domyślnej roli). Kluczowe może okazać się odpowiednie ustawienie parametrów **max_standby_archive_delay** i **max_standby_streaming_delay**. Wartości dla tych parametrów zdeterminnowane są przez potrzeby biznesowe.

Następujące komendy administracyjna są nie dostępne podczas pracy w trybie *hot standby:* **CREATE INDEX** (i inne komendy z DDL), **GRANT**, **REVOKE**, **REASSIGN**, **ANALYZE**, **VACUUM**, **CLUSTER**, **REINDEX**. Oczywiście, jeżeli niektóre z tych komend są naprawdę potrzebne mogą być wykonane na serwerze głównym.

Narzędzie monitorujące **check_pgsql** będzie pracować normalnie w trybie “gorącym”, jednak skrypt monitorujący **check_postgres** może zwracać różne wyniki, np. **last vacuum time** nie jest obsługiwane ponieważ *vacuum* nie działa na serwerze pracyjącym w tym trybie.

Komendy związane z plikami WAL (**pg_start_backup**, **pg_switch_xlog**) nie będą działać podczas przywracania systemu. Dynamicznie ładowane moduły takie jak **pg_stat_statements** działają poprawnie.

Jak wyżej wspomniano, “gorący” tryb pracy serwera zapasowego umożliwia tylko operacje czytające (*read-only*), nie ma tu wyjątku nawet dla tworzenia tabel tymczasowych, dlatego skrypty, które na nich polegają nie będą działać poprawnie – ograniczenie to prawdopodobnie zniknie w najnowszych wersjach Postgresa.

Wywołanie komend takich jak **ALTER DATABASE ... SET TABLESPACE**, **DROP DATABASE** skutkuje wygenerowaniem logu WAL co automatycznie powoduje natychmiastowe (niezależne od parametru **max_standby_streaming_delay**) rozłączenie wszystkich użytkowników mających sesję na serwerze zapasowym. Sama zmiana nazwy bazy (**ALTER DATABASE ... RENAME**) nie powoduje rozłączenia użytkowników, może jednak powodować problemy w

aplikacjach polegających na nazwie bazy.

Podczas normalnej pracy serwera (tj. nie podczas przywracanie systemu) komendy **DROP USER** lub **DROP ROLE** które dotyczą akutalnie zalogowanego użytkownika wykonane na serwerze głównym nie skutkują zakończeniem sesji, jednak próba ponownego połączenia nie będzie już możliwa.

Parametry związane z Hot Standby

serwer główny

- **wal_level**
- **vacuum_defer_cleanup_age**

serwer zapasowy

- **hot_standby**
- **max_standby_archive_delay**
- **max_standby_streaming_delay**

Konfiguracja replikacji strumieniowej

Wymagania

Dwa serwery z zainstalowanym *Debianem Squeeze* oraz *Postgresql 9.1* (z gałęzi *squeeze-backports*) w poniższej konfiguracji będziemy pracować na dwóch hostach:

```
Master: 192.168.0.1
Slave:  192.168.0.2
```

Konfiguracja mastera

Pierwsza czynność, którą należy wykonać to edycja pliku **postgresql.conf**

```
root@master:~# vi -w /etc/postgresql/9.1/main/postgresql.conf
```

Należy odszukać podane niżej zakomentowane linie i edytować je w następujący sposób :

```
#listen address = 'localhost'           # what IP address(es) to listen on;
listen_addresses = '*'

#wal_level = minimal                    #  minimal, archive, or hot_standby
```

```
wal_sevel = hot_standby

#max_wal_senders = 0
max_wal_senders = 3
```

Następnie należy utworzyć nowego użytkownika na serwerze PostgresSQL za pomocą, którego będzie przebiegała replikacja :

```
root@master:~# psql -h localhost -U postgres -W -c “CREATE USER ruser WITH
REPICATION PASSWORD 'password';”
```

Teraz należy przydzielić nowemu użytkownikowi odpowiednie uprawnienia :

```
root@master:~# vi /etc/postgresql/9.1/main/pg_hba.conf
```

```
host        replication        ruser        192.168.0.2/24        md5
```

Na koniec należy zatrzymać usługę postgresql :

```
root@master:~# service postgresql stop
```

Konfiguracja slave'a

Na kolejnej maszynie również należy edytować plik konfiguracyjny *PostgreSQL*

```
root@slave:~# vi -w /etc/postgresql/9.1/main/postgresql.conf
```

```
#listen_addresses = 'localhost'
listen_addresses = '*'

#hot_standby = off
hot_standby = on
```

Następnie należy zatrzymać usługę oraz wyczyścić tzw. *data directory* (**Uwaga:** Wyczyszczenie data directory powoduje usunięcie wszystkich danych !!!)

```
root@slave:~# service postgresql stop
root@slave:~# cd /var/lib/postgresql/9.1/main
root@slave:~# rm -rf *
```

Kolejnym krokiem jest uwtorzenie pliku **`recovery.conf`** w *data directory*

```
root@slave:~# vi /var/lib/postgresql/9.1/main/recovery.conf
```

Następnie należy uzupełnić go następujące informacje :

```
primary_conninfo = 'host=192.168.0.1 port=5432 user=ruser password=password'
standby_mode = on
```

Uruchomienie replikacji

Należy skopiować wszystkie dane z mastera do slave'a. Można to zrobić poprzez następujące polecenie wykonując je na masterze:

```
root@master:~# rsync -av /var/lib/postgresql/9.1/main/*
192.168.0.2:/var/lib/postgresql/9.1/main/
```

Na masterze oraz na slave'a można teraz uruchomić usługę postgresql

```
root@master:~# service postgresql start
```

```
root@slave:~# service postgresql start
```

Teraz można sprawdzić status replikacji

```
root@master:~# psql -h localhost -U postgres -W -c “SELECT * FROM
pg_stat_replication;”
```

Test

Na masterze należy połączyć się do instniejącej już bazy *test*

```
root@master:~# su postgres
postgres@master:~# psql -d test
```

a następnie utworzyć testową relacje oraz dodać do niej kilka rekordów

```
CREATE TABLE t1(
  id INT PRIMARY KEY,
  name VARCHAR(25)
);

INSERT INTO t1 VALUES (1, 'foo');
```

```
INSERT INTO t1 VALUES (2, 'bar');
INSERT INTO t1 VALUES (3, 'baz');
INSERT INTO t1 VALUES (4, 'bin');
```

Na slave również należy połączyć się do bazy test

```
root@slave:~# su postgres
postgres@slave:~# psql -d test
```

oraz sprawdzić czy istnieje relacja utworzona na masterze

```
test=# \d
id | name
---+-----
1  | foo
2  | bar
3  | baz
4  | bin
```

Widać, że replikacja działa. Na koniec można spróbować ze slave'a dodać nowy rekord

```
test=# INSERT INTO t1 VALUES(4, 'asdf');
BŁĄD: nie można wykonać INSERT wewnątrz transakcji tylko do odczytu
```

Zadanie

Powyższa implementacja realizuje strumieniową replikację *hot standby.* Replikacje systemów bazodanowych często wykorzystywane są by zapewnić wysoką niezawodność (*high availability*). Zmodyfikuj powyższą implementację by po “padzie” serwera głównego serwer zapasowy przejął jego obowiązki stając się jednocześnie nowym serwerem głównym.