Sławomir Civic Białek
<civic@bialek.org>

Jak zatrudnić słonie do replikacji baz PostgreSQL?

Katowice 28 sierpnia 2007

Streszczenie

Slony to opensource'owe i darmowe rozwiązanie problemu replikacji baz danych dla PostgreSQL w architekturze Master-Slave. Zapewnia niezłe rezultaty przy minimalnym koszcie nawet przy dużych bazach danych i dużych instalacjach takich jak dziesiątki węzłów rozrzuconych po całym świecie. Pokażemy czym slony jest, a czym nie jest, jak działa, i jak go bezboleśnie i skutecznie wdrożyć do działania.

1. Wprowadzenie

System zarządzania bazami danych PostgreSQL staje się coraz bardziej zaawansowany co czyni z niego godnego konkurenta dla innych produktów. Nadaje się już do wielu zastosowań, także do tych bardziej poważnych i wymagających. Nie posiada jednak wbudowanych możliwości replikacji baz danych, a są one coraz częściej niezbędne lub przynajmniej bardzo przydatne we współczesnych systemach informatycznych. Obecnie istnieje kilka zewnętrznych pakietów oprogramowania, które umożliwiają replikację danych dla PostgreSQL. Jednym z nich jest Slony, który zostanie tutaj przybliżony.

Slony ujrzał światło dzienne w 2004 roku i od tego czasu stał się jednym z najbardziej dopracowanych rozwiązań tego typu. Nie jest jednak rozwiązaniem nadającym się do wszystkich zastosowań, przede wszystkim dlatego, że istnieją różne modele replikacji, każdy z nich ma swoje zalety, ale także wady i ograniczenia. Przed podjęciem decyzji o zastosowaniu tego lub innego pakietu należy najpierw dobrze określić swoje potrzeby i skonfrontować z możliwościami.

2. Główne cechy

Slony realizuje replikację w sposób asynchroniczny w modelu single master to multiple slaves. Asynchronicznie oznacza, że transakcje są zatwierdzane w źródłowej bazie danych od razu, bez oczekiwania na potwierdzenie, że zakończyła się ich replikacja. W odróżnieniu od replikacji synchronicznej (w której zakończenie transakcji oznacza również jej zreplikowanie) w przypadku poważnej awarii źródłowej bazy danych możemy nie mieć gwarancji, że wszystkie zatwierdzone transakcje zostały zreplikowane. Ma to jednak także swoje zalety - replikacja asynchroniczna jest mniej kłopotliwa w użytkowaniu, bardziej wydajna i skalowalna, a węzły docelowe mogą być dostępne przez wolniejsze i zawodne łącza.

W Slony całość połączonych ze sobą baz nazywamy klastrem. Klaster składa się z dwóch lub więcej węzłów, przy czym węzłem jest pojedyncza baza danych. Dwie bazy danych założone w jednej instancji PostgreSQL to już dwa węzły (jeżeli obie zostaną przyłączone do klastra). Oczywiście mogą one także istnieć na różnych maszynach, nawet oddalonych od siebie tysiące kilometrów. Model single master to multiple slaves oznacza, że źródłem replikowanych danych jest tylko jeden węzeł, z którego skutek wykonania transakcji jest przenoszony na pozostałe węzły klastra (jednego lub więcej). Skutkiem tego jest jedno z najważniejszych ograniczeń Slony — replikowane dane tylko na jednym z węzłów są dostępne w trybie do zapisu (update, delete), na pozostałych dostępne są wyłączenie w trybie odczytu (select). Ogranicza to pole zastosowań tego produktu, z drugiej strony zwalnia nas z takich problemów jak rozwiązywanie konfliktów występujących w systemach multimaster. Określenia te (master, slave) i ograniczenia z tego wynikające nie dotyczą jednak w Slony węzłów jako całości, a jedynie konkretnych, replikowanych obiektów. Dzięki temu jeden węzeł dla części tabel może pełnić rolę mastera, a dla innych slave'a. Dla tych drugich masterem może być inny węzeł, otrzymamy w ten sposób pewną namiastkę replikacji multimaster.

Istotną cechą Slony jest możliwość utworzenia replikacji kaskadowej, w której węzły typu slave przekazują dane dalej. Umożliwia to stworzenie drzewiastej struktury replikacji i odciążenie węzłów typu master w przypadku potrzeby replikacji do większej liczby baz, a także zaoszczędzenie pasma połączeń pomiędzy niektórymi węzłami.

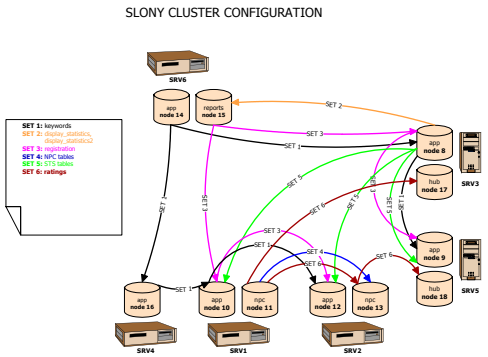
Istnieje także specjalny tryb zachowywania replikowanych zmian do plików (log shipping), które mogą być potem przesłane do docelowego węzła poprzez ftp czy email. Pliki te zawierają polecenia SQL, których sekwencyjne wykonanie doprowadzi bazę do takiego stanu jak baza źródłowa (do momentu zamknięcia pliku). W tym

trybie komunikacja jest oczywiście jednokierunkowa i węzły źródłowe nie otrzymują informacji na temat stanu synchronizacji takiego slave'a.

Podstawową jednostką podlegającą replikacji jest tzw. zbiór (ang. set), który może się składać się z wielu tabel i sekwencji występujących w bazie danych. Zbiór posiada swoje źródło — jeden węzeł typu master i jeden lub więcej typu slave, do których jest on w całości replikowany (bezpośrednio lub pośrednio w przypadku kaskady). Używane tutaj pojęcia master i slave odnoszą się w Slony do zbioru, a nie do węzła. W ramach pojedynczego klastra może istnieć więcej niż jeden zbiór i każdy z nich może mieć swoje źródło w innym węźle klastra. W dodatku elastyczność konfiguracji pozwala, aby każdy dowolny węzeł był jednocześnie masterem jak i slavem, oczywiście dla różnych replikowanych zbiorów. Odseparowanie struktur danych klastra w osobnym schemacie o nazwie wybieranej w czasie tworzenia klastra umożliwia także utworzenie całkowicie niezależnych klastrów korzystających z tych samych węzłów.

Ze względu na możliwość replikacji kaskadowej pojęcia master i slave mogą być trochę nieprecyzyjne, więc w konfiguracji Slony wprowadzono pojęcia źródła (ang. origin), dostawcy (ang. provider) oraz subskrybenta (ang. subscriber). Węzeł będący w środku kaskady jest jednocześnie subskrybentem (od dostawcy — źródła danych) i dostawcą dla kolejnego subskrybenta w łańcuchu replikacji.

Bardziej złożoną, sprawdzoną w rzeczywistych warunkach na połączeniach typu WAN, strukturę klastra przedstawia poniższy rysunek.



Instalacja i uruchomienie Slony mogą być przeprowadzone na działającym systemie bez zakłócania jego pracy, oprócz wprowadzenia dodatkowego obciążenia na wykonanie pierwszej synchronizacji (skopiowanie danych). Konfiguracja może być zmieniana w locie — usuwanie istniejących węzłów, dodawanie nowych można wykonywać niemal bez ograniczeń, dzięki czemu możemy elastycznie rozwijać strukturę klastra wraz z rozwojem i zmianami potrzeb systemu informatycznego. Ważną i również bezproblemową operacją, wykonywalną w czasie pojedynczych sekund, jest zamiana ról węzłów — źródło danych staje się subskrybentem, a subskrybent źródłem, co można wykorzystać np. do trwałego lub tymczasowego (na czas jakiejś operacji) przeniesienia głównego (umożliwiającego odczyt i zapis) serwera bazy danych dla wybranego zbioru. W przypadkach krytycznych i awarii źródła danych możemy w szybki sposób uczynić nowym źródłem dowolny z pozostałych węzłów.

Trzeba tutaj zwrócić uwagę, że Slony zajmuje się tylko replikacją danych. Samo wykrycie sytuacji awaryjnej jest poza zainteresowaniami twórców tego pakietu, należy o to zadbać na inne sposoby. Również o przełączanie aplikacji pomiędzy węzłami należy zadbać samemu. Poza rozwiązaniami ręcznymi i programowymi pomocne w tym mogą być inne pakiety oprogramowania takie jak pgpool.

Aktualnie Slony 1.2.x współpracuje z PostgreSQL w wersjach od 7.4 do najnowszych z serii 8.3. Posiadacze wersji 7.3 mogą skorzystać ze Slony 1.1.5 i na przykład wykorzystać go do przeprowadzenia aktualizacji PostgreSQL do nowszej wersji, po czym przejść na Slony z serii 1.2.x (które już 7.3 nie wspierają). Slony umożliwia replikację pomiędzy węzłami z różnymi wersjami PostgreSQL (o ile tylko mieszczą się w zakresie wspieranych wersji). Wersje pakietu Slony na wszystkich węzłach klastra powinny być takie same. Slony można uruchomić systemach unixowych (nie tylko Linux, ale także rodzina BSD, Solaris itp.) oraz ostatnio także na MS Windows.

Biorąc pod uwagę powyższe cechy i możliwości Slony dobrze nadaje się między innymi do:

— tworzenie grupy serwerów dla rozłożenia obciążenia na większą ilość maszyn (tylko dla operacji odczytu)
— bezpieczne udostępnienie zawsze aktualnej kopii danych w trybie tylko do odczytu
— tworzenie zapasowych serwerów, które w razie krytycznej sytuacji mogą niemal natychmiastowo przejąć rolę zabezpieczanych maszyn
— online'owy backup, także do odległych lokalizacji po łączach WAN
— uruchomienie pełnoprawnego zapasowego serwera bazy danych, który przejmie role głównego serwera na czas koniecznych prac przy nim (np. upgrade wersji PostgreSQL)
— przyrostowe aktualizacje kopii bazy oparte na przekazywaniu plików (tryb log shipping)

3. Zasada działania

Zanim przystąpimy do użytkowania Slony warto poznać przynajmniej ogólnie sposób w jaki realizuje swoje funkcje. Ułatwia to jego wykorzystanie, zrozumienie pewnych zdarzeń i ograniczeń oraz rozwiązywanie ewentualnych problemów.

Slony wykorzystuje standardowe możliwości oferowane przez PostgreSQL dzięki czemu nie trzeba dokonywać zmian w kodzie źródłowym i można go doinstalować do działającego serwera. Cały pakiet składa się z następujących elementów:

— zestawu procedur napisanych w językach C oraz pl/pgsql

— triggerów (wykorzystujących w/w procedury)
— struktur danych przechowujących dane o budowie klastra oraz informacje o procesie replikacji
— programu slonik — wykonującego polecenia konfiguracyjne
— programu slon — działającego jako daemon i wykonującego właściwą część replikacji danych

Podczas przyłączania węzła do klastra w jego bazie tworzony jest osobny schemat o nazwie zadeklarowanej przez użytkownika (poprzedzony znakiem podkreślenia) zawierający struktury danych związane z replikacją. Kilka tabel, widoków i sekwencji oraz rejestruje zestaw procedur. Struktury te są używane zarówno do przechowywania danych o konfiguracji klastra, jak i samych replikowanych danych (tymczasowo).

Właściwa replikacja oparta jest głównie o triggery, które zakładane w procesie konfiguracji na tabelach źródłowych, porównują stan wiersza przed i po operacji. Zapisują one różnice (tylko zmienione i nowe dane, rezultat wykonania transakcji, nie treść poleceń SQLa) do specjalnych tabel przechowujących zalogowane zmiany. Analogicznie na tabelach węzłów docelowych są także zakładane triggery, tylko że zabezpieczające je przed modyfikacjami (tylko środowisko Slony będzie mogło je modyfikować co zagwarantuje spójność).

Do każdego węzła musi być podłączony jeden proces slon. Sprawdza on regularnie czy nie zostały zalogowane nowe zmiany, a jeśli tak to generuje i zapisuje w odpowiedniej tabeli zdarzenie SYNC informujące inne węzły o danych do pobrania. slon posiada kilka opcji pozwalających sterować parametrami czasowymi generacji tych zdarzeń. Jednocześnie proces slon łącząc się z innymi węzłami (zgodnie z konfiguracją) pobiera informacje o tam wygenerowanych zdarzeniach i zajmuje się ich wykonaniem — czyli głównie pobraniem logów zmian i naniesieniem ich na lokalną bazę danych. Gdy to samo dzieje się na pozostałych węzłach otrzymujemy w efekcie działającą replikację.

Zdarzenie (ang. event) jest dość ważnym pojęciem w działaniu Slony. Cała praca klastra polega na propagacji i wykonywaniu zdarzeń. Zmiany konfiguracji klastra także powodują wygenerowanie odpowiednich zdarzeń, które dokonują odpowiednich modyfikacji w tabelach opisujących konfigurację klastra na wszystkich węzłach. Należy zadbać o ich odbieranie, przekazywanie i wykonywanie (czyli prawidłowa konfiguracja, połączenia i działające procesy slon), w przeciwnym wypadku niepotwierdzone będą wraz ze wszystkimi danymi przechowywane na węzłach, które je wygenerowały.

Wróćmy do triggerów. Oparcie na triggerach choć ma szereg zalet to powoduje kilka ograniczeń. Triggerami nie da się objąć zmian w strukturach danych (polecenia typu DDL — Data Definition Language), sekwencji, ani tzw. dużych obiektów binarnych. Replikacja zmian wykonywanych przy pomocy DDL mogła by też spowodować inne problemy. Jednak Slony zapewnia tutaj możliwość wykonania dowolnego polecenia, w tym DDL, na wszystkich węzłach klastra. Z pewnymi ograniczeniami i trudnościami pozwala to na zmienianie schematu bazy w klastrze. Należy zachować szczególną ostrożność i najpierw zapoznać się z rozdziałem dokumentacji opisującym to zagadnienie (Database Schema Changes).

Problem replikacji sekwencji został rozwiązany na razie dość siłowo. Stan wszystkich zasubskrybowanych sekwencji jest przekazywany przy każdym zdarzeniu typu SYNC. To niestety może powodować problemy wydajnościowe przy ich dużej liczbie. Konsekwencją tego podejścia jest także brak możliwości replikacji samej sekwencji, ponieważ bez zmian powodujących generowanie zdarzenia SYNC nie zostanie przekazany aktualny jej stan. Wdrażając klaster można się zastanowić czy w ogóle muszą one być replikowane — stan większości może zostać odtworzony na podstawie zawartości w tabelach (ale wymaga to dodatkowej czynności i czasu) i zazwyczaj są potrzebne dopiero przy wstawianiu nowych rekordów (co nie zachodzi na węzłach typu read-only). Zależne to będzie od charakteru klastra, jeśli subskrybenci mają być zdolni błyskawicznie przejąć rolę źródła danych to aktualna wartość sekwencji raczej będzie potrzebna, ale w innych przypadkach można przyjąć inną strategię.

Zmiany dużych obiektów binarnych czyli BLOBów (obsługiwanych przez funkcje takie jak lo_create, lo_import), nie mogą być obsłużone triggerami, co uniemożliwia replikowanie ich przez Slony. Możemy za to użyć dla danych binarnych typu bytea, który nie posiada takich ograniczeń (za to nie można na nim korzystać z rodziny funkcji lo_*).

Przyjęty model działania i komunikacji może także sprawiać problemy wydajnościowe przy zbyt dużej liczbie węzłów (rzędu kilkudziesięciu), ale to już zależy od wielu parametrów i konkretnego przypadku. W takim przypadku można spróbować ratować się podziałem klastra na kilka niezależnych, mniejszych.

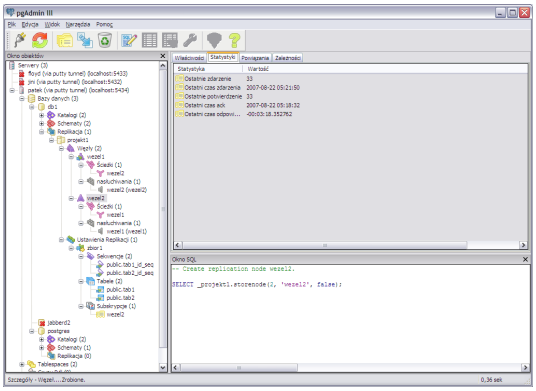
4. Narzędzia

Dla Slony podstawowym narzędziem służącym do konfiguracji i zarządzania klastrem jest slonik działającym na zasadzie procesora prostych tekstowych poleceń. Wbrew pozorom przy odpowiednim wykorzystaniu i organizacji jest to całkiem skuteczna metoda. Dzięki niemu możemy stworzyć zestaw skryptów dla różnych etapów tworzenia i modyfikacji klastra i obsługi szczególnych sytuacji i zdarzeń (np. awarii). W dalszych przykładach praktycznego kursu wykorzystamy właśnie slonika.

W pakiecie lub swoim systemie (po instalacji, jeśli wykorzystamy w konfiguracji pakietu opcję –with-perltools) znajdziemy także zestaw skryptów napisanych w języku PERL, które w oparciu o jeden plik konfiguracyjny opisujący klaster mogą wykonywać poszczególne czynności potrzebne przy konfiguracji i późniejszym wykorzystaniu Slony.

Od jakiegoś czasu także popularny pgAdmin posiada stale rozwijane wsparcie dla Slony. W najnowszych wer-

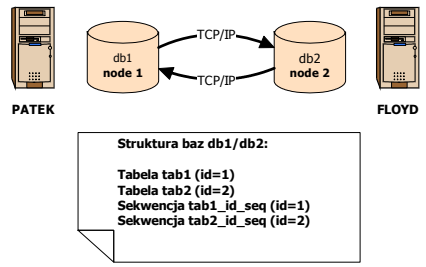
sjach (1.8.x) umożliwia już wykonywanie w charakterystyczny dla siebie sposób oparty o GUI niemal wszystkich standardowych czynności, a przy okazji prezentuje z wykorzystaniem możliwości GUI strukturę klastra i trochę podstawowych informacji na temat jego pracy.



5. Praktyka

Zobaczmy teraz jak wygląda użycie Slony w praktyce, na prostym przykładzie. Przećwiczenie tego pozwoli do końca zrozumieć jak on działa i większe instalacje nie powinny być problemem. W razie trudności można liczyć na pomoc innych użytkowników oraz developerów Slony na listach dyskusyjnych.

Załóżmy, że mamy do czynienia z sytuacją z poniższego rysunku. Dwa serwery, w bazie danych dwie tabele wraz z sekwencjami, które chcemy replikować.



5.1. Kompilacja i instalacja

Slony powinien być skompilowany dla konkretnej wersji PostgreSQL z którą przyjdzie mu współpracować. Najprościej skompilować go na każdym z serwerów klastra. Kompilacja i instalacja odbywa się w prosty i klasyczny dla projektów Open Source sposób:

```
./configure --with-perltools --with-docs
LANG=C gmake
gmake install
```

Ze względu na konflikt z polskimi locale przy generowaniu dokumentacji dodajemy ich zmianę na "C". W systemach opartych o RPM możemy zbudować RPMa:

```
./configure
gmake rpm
```

Niektóre dystrybucje zawierają DocBook DTD ze zbyt dużym ograniczeniem dla stałej NAMELEN, co uniemożliwia zbudowanie dokumentacji (https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=159382).

5.2. Przygotowania

Oprócz samego oprogramowania musimy wykonać kilka czynności przygotowawczych w samym PostgreSQL:

- utworzyć użytkownika dedykowanego do replikacji o uprawnieniach superusera (możemy co prawda skorzystać np. z użytkownika postgres lub innego, ale lepiej mieć wydzielonego użytkownika do tych celów) — CREATE USER slony SUPERUSER;
- udostępnić możliwość połączenia pomiędzy węzłami (odblokowanie portów na firewallu, nasłuch na porcie TCP/IP — postgresql.conf, dostęp i uwierzytelnianie użytkowników — pg_hba.conf), tak aby węzły pomiędzy którymi zachodzi replikacja mogły się wzajemnie do siebie łączyć na użytkownika z prawami superusera; dodatkowo potrzebujemy na czas konfiguracji możliwość połączenia do wszystkich węzłów ze stacji na której uruchamiamy program zarządzający konfiguracją slonik
- należy zarejestrować język Pl/PGSQL — CREATE LANGUAGE PLPGSQL;
- należy utworzyć schematy (bez danych) replikowanych tabel i sekwencji na węzłach docelowych (można je przenieść np. wykorzystując polecenie: pg_dump -s)

5.3. Utworzenie klastra

Do utworzenia klastra i pozostałych czynności konfiguracyjnych wykorzystamy program slonik, który przetworzy nasze polecenia i wykona odpowiednie zapisy oraz czynności na węzłach klastra.

W tym celu potrzebna jest możliwość połączenia ze stacji na której będziemy uruchamiać slonika do PostgreSQL na wszystkich węzłach klastra. Możemy to przetestować łącząc się z każdym z nich przy pomocy polecenia psql.

Ponieważ w wielu miejscach i sytuacjach będziemy potrzebowali danych takich jak nazwa klastra, identyfikatory węzłów, dane o połączeniach między węzłami proponuję utworzyć plik, który będzie zawierał definicję tego typu informacji. slonik umożliwia definiowanie stałych i wstawianie ich z dodatkowego pliku. Tworzymy plik projekt1.slonik o następującej zawartości:

```
# nazwa klastra
```

```
  cluster name = projekt1;
  # nazwy symboliczne - identyfikatory węzłów
  define patek_db1 1;
  define floyd_db2 2;
  # połączenia
  define patek_db1_conn
      'dbname=db1 host=patek user=slony';
  define floyd_db2_conn
      'dbname=db2 host=floyd user=slony';
  node @patek_db1 admin conninfo =
           @patek_db1_conn;
  node @floyd_db2 admin conninfo =
           @floyd_db2_conn;
```

Definicję klastra rozpoczynamy od nadania mu nazwy. Każdy węzeł musi posiadać swój unikalny, liczbowy identyfikator, więc powyżej zdefiniowaliśmy wygodniejsze w dalszym użyciu nazwy symboliczne (patek_db1, floyd_db1). Także ciągi opisujące sposób połączenia (nazwa bazy, host, użytkownik) zostały opatrzone swoimi nazwami (patek_db1_conn, floyd_db1_conn). Następnie używamy ich w poleceniu node określającym dla slonika (i tylko dla niego) dane do połączeń z poszczególnymi węzłami.

Wszystkie polecenia konfiguracyjne również wygodnie jest zapisać w formie skryptów. Skrypt tworzący klaster będzie miał następującą postać:

```
slonik <<_EOF_
  include <projekt1.slonik>
  init cluster ( id = @patek_db1,
      comment = 'wezel1' );
  store node   ( id = @floyd_db2,
      comment = 'wezel2' );

  store path (client = @patek_db1, server =
      @floyd_db2, conninfo = @floyd_db2_conn);
  store path (client = @floyd_db2, server =
      @patek_db1, conninfo = @patek_db1_conn);
_EOF_
```

Uruchamiamy powyższy skrypt na dowolnej maszynie, która może się połączyć poprzez TCP/IP z PostgreSQL na każdym z węzłów (nie musi to być nawet żaden z węzłów). Najpierw budujemy klaster na pierwszym węźle (init cluster), po czym dodajemy kolejne węzły (store node). Następnie określamy sposób połączenia pomiędzy węzłami dla daemona slon. Wystarczy określić sposób połączenia dla węzłów, pomiędzy którymi rzeczywiście będzie wykonywana replikacja, choć nie zaszkodzi jeśli zostaną zdefiniowane nadmiarowe połączenia (nie będą wykorzystywane dopóki nie będą potrzebne). W niektórych opracowaniach na temat Slony pojawia się jeszcze informacja o konieczności definiowania tzw. listen paths przy pomocy polecenia store listen — od wersji Slony 1.1 są wyznaczane automatycznie i nie ma już potrzeby ręcznego ich definiowania (nawet jeśli zdefiniujemy je ręcznie to zostaną skasowane i utworzone automatycznie na podstawie pozostałych elementów konfiguracji).

Po wykonaniu skryptu, w bazie każdego węzła pojawi się schemat o nazwie naszego klastra (poprzedzony znakiem podkreślenia) ze wszystkimi elementami (tabele, widoki, sekwencje, procedury itd.) potrzebnymi do dalszych działań. Jeśli się wszystko powiedzie możemy przystąpić do uruchomienia procesów slon, które dokończą konfigurację w klastrze (po samym utworzeniu klastra węzły otrzymują tylko część konfiguracji, reszta zostanie zreplikowana). Uruchamiamy po jednym procesie slon na każdy węzeł. Teoretycznie mogą być także uruchomione z dowolnego miejsca, z jednej lub więcej stacji, także z poza klastra jeśli zapewnimy możliwość odpowiednich połączeń. Dość naturalne jednak jest, szczególnie w konfiguracjach bardziej rozproszonych geograficznie (oszczędność pasma) uruchomienie po jednym procesie slon na każdym z węzłów klastra. Na początek z argumentem -d1 (debug) aby móc dokładniej obserwować co się dzieje.

```
slon -d1 projekt1 \
    'host=patek dbname=db1 user=slony'
slon -d1 projekt1 \
    'host=floyd dbname=db2 user=slony'
```

Jeśli coś pójdzie nie tak na etapie eksperymentowania i nauki, szybką i skuteczną metodą zlikwidowania klastra jest wykonanie polecenia usuwającego schemat Slony na wszystkich węzłach klastra:

```
drop schema _project1 cascade;
```

5.4. Tworzenie zestawu

Nadszedł moment na określenie co chcemy replikować. W tym celu tworzymy zestaw składający się z wybranych tabel i sekwencji. Tradycyjnie wspomożemy się wygodnym skryptem i dodamy dwie tabele wraz z towarzyszącymi im sekwencjami:

```
slonik <<_EOF_
  include <projekt1.slonik>
  create set (id = 1, origin = @patek_db1,
      comment = 'zbior1');
  set add table (id = 1, set id = 1,
       origin = @patek_db1,
       fully qualified name = 'public.tab1',
       comment = '');
  set add table (id = 2, set id = 1,
       origin = @patek_db1,
       fully qualified name = 'public.tab2',
       comment = '');
  set add sequence (id = 1, set id = 1,
       origin = @patek_db1, fully qualified
       name = 'public.tab1_id_seq',
       comment = '');
  set add sequence (id = 2, set id = 1,
       origin = @patek_db1, fully qualified
       name = 'public.tab2_id_seq',
       comment = '');
```

```
_EOF_
```

Wymagane jest aby każdemu obiektowi dodawanemu do klastra, tak jak przy definiowaniu węzłów, nadawać unikalny, liczbowy identyfikator (zwykle kolejny numer, choć nie jest to wymagane, można wykorzystać związek z numeracją występującą poza klastrem). Każdy z rodzajów obiektów posiada swoją odrębną przestrzeń identyfikatorów, stąd w powyższym przykładzie zaczynamy od id = 1 przy tworzeniu zbioru, przy dodawaniu pierwszej tabeli oraz przy pierwszym poleceniu dodającym sekwencję.

Możemy przy tej operacji natknąć się na problem braku głównego klucza w tabeli, który jest wymagany. Jeżeli jest taki jawnie zadeklarowany w tabeli (CONSTRAINT PRIMARY KEY) to zostanie on automatycznie użyty. Możemy też przy pomocy dodatkowych opcji polecenia set add table wskazać inne pola lub poprosić Slony o założenie takiego klucza dla nas.

Po prawidłowym wykonaniu skryptu przy nadal działających procesach slon informacja o nowym zestawie powinna rozejść się w klastrze, co możemy sprawdzić bezpośrednio w tabelach na każdym z węzłów:

```
SELECT * from _projekt1.sl_set;
```

Dodanie tabel do zestawu spowoduje także utworzenie na wskazanym dla nich węźle źródłowym triggerów zapewniających logowanie wykonywanych operacji na tabelach. Warto sprawdzić czy to się udało:

```
db1=# \d tab1
(...)
Triggers:
    _projekt1_logtrigger_1 AFTER INSERT (...)
```

5.5. Subskrypcja

W końcu dochodzimy do ostatniego kroku (dla początkowej konfiguracji) — zasubskrybowania węzłów na utworzony zestaw tabel (i sekwencji).

Prosty skrypt określający, który zestaw, od kogo, do kogo — i gotowe. Replikacja działa. Wykonujemy na węźle źródłowym testowe manipulacje na danych i obserwujemy jak ładnie przenoszą się na subskrybujący go węzeł.

```
slonik <<_EOF_
  include <projekt1.slonik>
  subscribe set (id = 1, provider =
     @patek_db1, receiver = @floyd_db2,
     forward = no);
  wait for event (origin = @floyd_db2,
     confirmed = @patek_db1);
  sync (id = @patek_db1);
  wait for event (origin = @patek_db1,
     confirmed = @floyd_db2);
_EOF_
```

Kilka uwag praktycznych związanych z uruchamianiem subskrypcji. Parametr forward określa czy dane będą replikowane dalej, na kolejne węzły (replikacja kaskadowa) — węzeł musi wiedzieć czy ma je zostawić dla kolejnych węzłów. Ostatnie trzy linie powyższego skryptu są opcjonalne — wymuszają one oczekiwanie na zakończenie operacji włączania subskrypcji włącznie z kopiowaniem danych do węzła docelowego (samo subscribe set tylko rozpoczyna proces i od razu kończy działanie). Informacja o zakończeniu całej tej operacji może się w niektórych przypadkach przydać. W momencie uruchamiania subskrypcji na tabelach docelowych wykonywany jest najpierw TRUNCATE (nie ma więc sensu zawracać sobie wcześniej głowy wypełnianiem tych tabel), a następnie przenoszone są dane z węzła źródłowego (przy pomocy COPY). W przypadku dużych zbiorów danych można na ten czas usunąć indeksy, co przyśpieszy tą operację, i przywrócić je po zakończeniu. O ile będą potrzebne, ponieważ kwestie takie jak indeksy czy uprawnienia do obiektów bazy funkcjonują już poza Slony i mogą się różnić na poszczególnych węzłach (co można celowo wykorzystać). Po tej operacji zobaczymy też, że zostały założone triggery, które zabraniają operacji INSERT/UPDATE/DELETE na replikowanych tabelach. Od tego momentu tabele będą dostępne tylko do odczytu.

```
db2=# \d tab1
(...)
Triggers:
    _projekt1_denyaccess_1 BEFORE INSERT (...)
```

5.6. Zamiana ról

Gdy mamy działającą replikację możemy przećwiczyć szczególne scenariusze jej wykorzystania. Załóżmy, że na głównym serwerze, który jest źródłem dla naszego zestawu potrzebuje wykonać jakieś prace wymagające jego wyłączenia, a jednocześnie musimy zapewnić działanie systemu informatycznego. Dzięki uprzednio skonfigurowanej i działającej replikacji mamy zapasowy serwer ze stale aktualną kopią danych. Wykonajmy więc zamianę ról — niech źródło danych stanie się subskrybentem, a subskrybent źródłem. Musimy tylko przekazać nasz zamiar slonikowi i gotowe:

```
slonik <<_EOF_
  include <projekt1.slonik>
  lock set (id = 1, origin = @patek_db1);
  wait for event (origin = @patek_db1,
     confirmed = @floyd_db2);
  move set (id = 1, old origin = @patek_db1,
     new origin = @floyd_db2);
  wait for event (origin = @patek_db1,
     confirmed = @floyd_db2);
_EOF_
```

W tym momencie musimy także zapewnić, już poza Slony, przełączenie użytkowników czy też aplikacji na nowy serwer. Ręcznie, na poziomie połączeń TCP/IP lub przy pomocy aplikacji proxujących połączenia do bazy. Możemy zająć się byłym (chwilowo) serwerem głównym.

Zaraz po tej operacji będzie zwykłym subskrybentem, replikacja będzie działać po prostu w drugą stronę. Jeśli taka jest potrzeba — może to pozostać stanem permanentnym i nie musimy wracać do poprzedniej konfiguracji. Jednak dla tego przykładu załóżmy, że jest to sytuacja tymczasowa. Gdy odłączymy serwer (wyłączymy proces slony, PostgreSQL, odłączymy go od sieci albo fizycznie wyłączymy) zmiany wykonane na tymczasowym serwerze będą logowane i będą oczekiwały na ponowne pojawienie się tego węzła. Uwaga — powinien wrócić z zawartością bazy taką jaka była przed odłączeniem, może to być już np. nowszy PostgreSQL ale struktury danych i dane muszą zostać zachowane, w przeciwnym wypadku takiego węzła nie da się z powrotem przyłączyć do klastra w opisany sposób. Można go oczywiście przyłączyć na zasadzie usunięcia z klastra i ponownego podłączenie z pełną synchronizacją replikowanych zestawów danych.

Po wykonaniu prac związanych z głównym serwerem chcemy przywrócić jego rolę. Wykonujemy więc operację odwrotną:

```
slonik <<_EOF_
  include <projekt1.slonik>
  lock set (id = 1, origin = @floyd_db2);
  wait for event (origin = @floyd_db2,
     confirmed = @patek_db1);
  move set (id = 1, old origin = @floyd_db2,
     new origin = @patek_db1);
  wait for event (origin = @floyd_db2,
     confirmed = @patek_db1);
_EOF_
```

I po chwili (tym razem być może dłuższej — w zależności od tego ile danych zebrało się do zreplikowania) znajdziemy się w początkowej sytuacji. Sprawne przełączenie użytkowników i aplikacji korzystających z naszych baz i udało nam się wymienić niepewne twarde dyski i zrobić aktualizację PostgreSQL z wersji 7.x na 8.x niemal bez utrudnień dla użytkowników (niestety, jednak sam moment przełączenia może zostać zauważony i być drobną uciążliwością).

5.7. Awaria

Scenariusz zamiany ról może być zastosowany tylko w sytuacji kiedy oba węzły w momencie zamiany są sprawne i działające. W przypadku awarii sytuacja wygląda trochę inaczej. Jeśli awarii ulega węzeł, który nie jest źródłem danych po prostu odłączamy go od klastra i gdy już będziemy pewni że nie da się go przyłączyć z powrotem pozostaje go usunąć z konfiguracji (nie należy o tym zapominać, inaczej będą kolejkowane dla niego zmiany danych). W przypadku awarii węzła będącego źródłem danych najprawdopodobniej chcemy zastąpić go innym węzłem. Tym razem będzie to operacja jednokierunkowa, przekazujemy status źródła danych dla naszego zestawu innemu węzłowi, a pierwotne źródło usuwamy z klastra. W tym przypadku dotychczasowe źródło nie musi być działające, i tak zostanie usunięte. Znowu instruujemy slonika:

```
slonik <<_EOF_
  include <projekt1.slonik>
  failover (id = 1,
     backup node = @floyd_db2);
  drop node (id = @patek_db1,
     event node = @floyd_db2);
_EOF_
```

W efekcie mamy nowe źródło dla naszego zestawu i pozbyliśmy się z klastra pechowego węzła. Jeśli uda się go naprawić będzie go można podłączyć jako nowy węzeł, zasubskrybować dane i przekazać mu status źródła wg procedury opisanej w poprzednim punkcie (zamiana ról). W momencie wykonywania failover przy większej ilości węzłów Slony sam ustali, który z nich ma najbardziej aktualną kopię danych z zestawu i najpierw wykona ich replikację do nowo wybranego źródła, dzięki czemu stracimy najmniej jak się da. Jednak trzeba liczyć się z tym, że w takim przypadku mogły zostać utracone jakieś transakcje, które ze źródła nie zdążyły się zreplikować na żaden z węzłów.

5.8. Inne operacje

To oczywiście nie wszystkie operacje jakie można wykonać na klastrze przy pomocy slonika. Polecam zapoznanie się z dokumentacją i przejrzenie opisów wszystkich poleceń. Szczególnie następujące mogą się dość szybko przydać:

— MERGE SET — połączenie dwóch zbiorów (połączenie dwóch zbiorów jest także metodą na dodanie tabeli i sekwencji do istniejącego zbioru)
— STORE TRIGGER — pozwala zablokować domyślne działanie wyłączające istniejące triggery w węzłach subskrybujących dane
— EXECUTE SCRIPT — wykonanie dowolnych poleceń SQL na wszystkich bądź wybranych węzłach, szczególnie przydatne dla poleceń typu DDL; należy koniecznie zapoznać się z osobnym rozdziałem dokumentacji zatytułowanym Database Schema Changes (DDL)
— cała rodzina poleceń typu DROP

6. Na zakończenie

Slony jest już niezłym, bogatym w możliwości i stabilnym produktem. Ponadto deweloperzy ciągle go rozwijają, uaktualniają, mają trochę nowych pomysłów na jego udoskonalenie i nowe możliwości. Choć nie jest receptą na wszystko, przydać się może w każdym poważniejszym projekcie informatycznym, w którym bierze udział PostgreSQL. Kiedyś pisano także o planach stworzenia Slony-II, który miał być oparty o inne założenia (synchroniczna replikacja multi-master), aczkolwiek ostatnio temat ten trochę przycichł. Artykuł ten nie wyczer-

puje całej tematyki związanej z obsługą Slony. Zagadnień, niuansów związanych z obsługą klastra i niektórymi sytuacjami jest trochę więcej — po wypróbowaniu tego oprogramowania zachęcam do zapoznania się z informacjami, które można znaleźć na stronie projektu (http://slony.info/), dokumentacji, listach dyskusyjnych i innych źródłach w Internecie.