Репликация
Сценарии использования

Авторские права
© Postgres Professional, 2018–2022
Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Использование материалов курса
Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Отказ от ответственности
Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Темы

Использование физической репликации

Использование логической репликации

Физическая репликация

Горячий резерв для высокой доступности

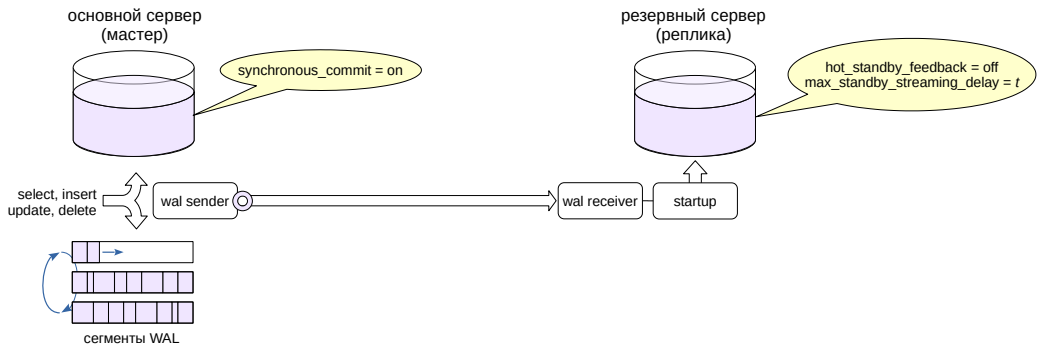
Балансировка OLTP-нагрузки

Реплика для отчетов

Несколько реплик и каскадная репликация

Отложенная репликация

Горячий резерв для HA



синхронная репликация: реплика максимально соответствует мастеру

запросы к реплике возможны, но не приоритетны:
они будут прерваны при любом конфликте

Реплика может создаваться для решения разных задач, и настройка зависит от ее предназначения.

Один из возможных вариантов — обеспечение горячего резерва для целей высокой доступности. Это означает, что при сбое основного сервера требуется как можно быстрее перейти на реплику, не потеряв при этом данные.

Надежность обеспечивает синхронная репликация в режиме *synchronous_commit* = on (значение write не гарантирует надежность; значение apply будет перебором).

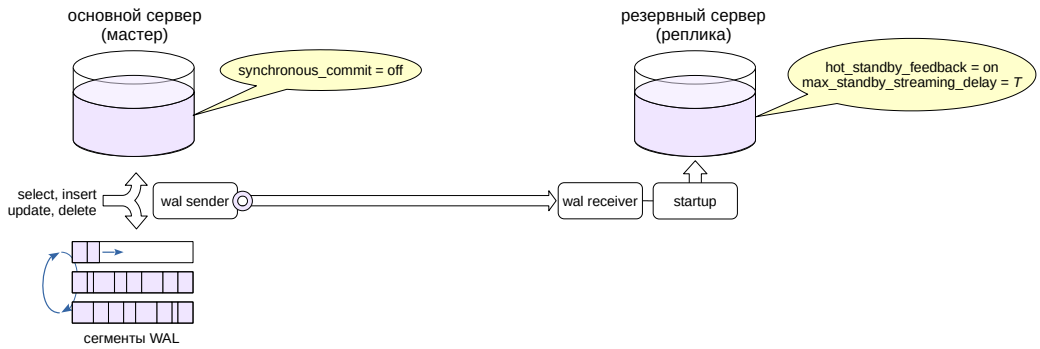
Чтобы реплика как можно меньше отставала от основного сервера, нужно применять журнальные записи сразу же. Для этого выставляем задержку *max_standby_streaming_delay* в небольшое значение.

Чтобы запросы к реплике не могли негативно повлиять на основной сервер, выключаем обратную связь (*hot_standby_feedback* = off).

С такими настройками запросы к реплике возможны, но в случае конфликтов они будут прерваны. Если запросы к реплике не предполагаются, можно создать ее в режиме «теплого резерва» (*hot_standby* = off).

Заметим, что репликация — базовый механизм для обеспечения высокой доступности, но далеко не все, что нужно. Более подробнее это обсуждается в модуле «Кластерные технологии».

Балансировка OLTP



запросы на реплике должны отрабатывать

долгие запросы повлияют на мастер, но в OLTP-нагрузке их не должно быть

согласованность данных должна обеспечиваться отдельно

Другой вариант — реплика используется для балансировки OLTP-нагрузки на чтение.

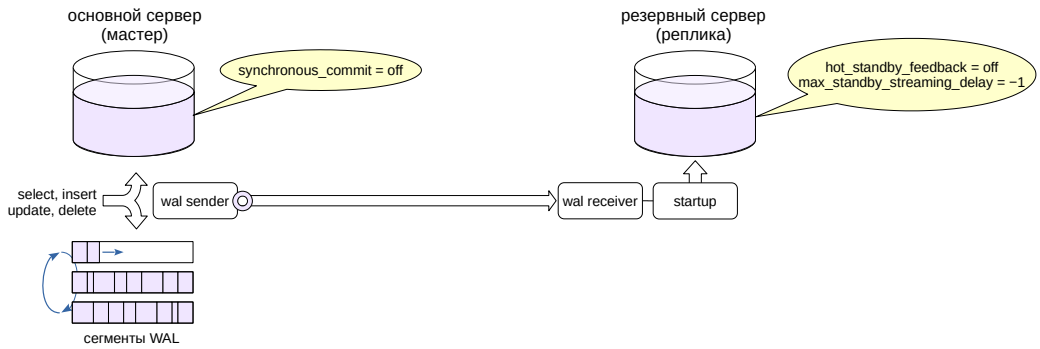
OLTP-нагрузка характеризуется небольшими по длительности запросами. Это позволяет достаточно тесно связать реплику с основным сервером, чтобы запросы гарантированно не прерывались из-за конфликтов, и в то же время не беспокоиться о негативном влиянии на основной сервер.

Включаем обратную связь (*hot_standby_feedback* = on) и выставляем достаточно большую задержку применения конфликтных записей (*max_standby_streaming_delay*).

Синхронная репликация в такой конфигурации, скорее всего, не требуется. Она все равно не обеспечивает согласованность данных: зафиксированное изменение на основном сервере не гарантирует, что запрос к реплике увидит эти изменения.

Для обеспечения согласованности нужно, чтобы изменения появлялись на обоих серверах *одновременно*. Режим *synchronous_commit* = apply позволяет дождаться применения журнальной записи на реплике (ценой сильного падения производительности основного сервера), но остается возможность увидеть данные на реплике раньше, чем на мастере. Надежное и прозрачное для приложений решение можно получить, используя специальные протоколы распределенных транзакций (см. модуль «Кластерные технологии»).

Реплика для отчетов



запросы на реплике должны отрабатывать, в том числе и долгие (но не требуются самые актуальные данные)

мастер не должен испытывать негативного влияния обратной связи

чтобы совсем развязать серверы — убрать слот и использовать архив WAL

Еще один возможный вариант — разделение по серверам нагрузки разного типа. Основной сервер может брать на себя OLTP-нагрузку, а длительные читающие запросы (отчеты) можно вынести на реплику.

Кроме распределения нагрузки между серверами, такое решение позволит избежать проблемы с тем, что длительные запросы на основном сервере могут задерживать выполнение очистки и приводить к разрастанию таблиц.

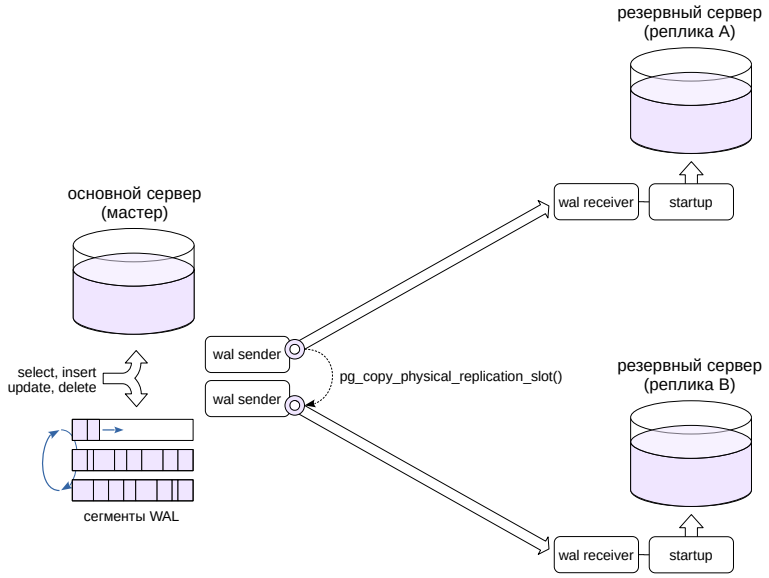
Из этих соображений «отчетная» реплика должна быть максимально отделена от основного сервера. Должна быть отключена обратная связь (*hot_standby_feedback* = off), но время откладывания конфликтующих журнальных записей нужно сильно увеличить, вплоть до бесконечности (*max_standby_streaming_delay* = −1).

Поскольку для длительных отчетов не требуются сиюминутные данные, можно, при наличии архива, отказаться от слота репликации (не забыв выставить параметр *max_standby_archive_delay* аналогично *max_standby_streaming_delay*). Тогда, если мастер успеет удалить необходимый реплике файл, реплика возьмет его из архива.

Такую реплику можно также использовать для выполнения резервного копирования.

Разумеется, на практике возможны и другие сочетания параметров. Главное — четко понимать, какую задачу требуется решить, и какое влияние оказывают те или иные параметры.

Несколько реплик



К основному серверу можно подключить несколько реплик. Никаких специальных настроек для этого не требуется, но надо учитывать, что каждой реплике будет соответствовать отдельный процесс wal sender и отдельный слот репликации. При развертывании можно растиражировать одну базовую копию и клонировать слот репликации функцией pg_copy_physical_replication_slot().

https://postgrespro.ru/docs/postgresql/13/functions-admin#FUNCTIONS-REPLICATION

Обычно такая схема требуется для распределения нагрузки, но собственно распределение должно решаться внешними средствами (см. модуль «Кластерные технологии»).

Мастер и две физические реплики

Настроим конфигурацию с двумя репликами.

Создадим автономную резервную копию первого сервера, одновременно создав слот.

```
student$ pg_basebackup --pgdata=/home/student/backup -R --create-slot --slot=beta
```

Поскольку третий сервер при старте начнет применять записи WAL с той же позиции, что и второй, можно дублировать слот:

```
α=> \c - postgres
```

```
You are now connected to database "student" as user "postgres".
```

```
α=> SELECT pg_copy_physical_replication_slot('beta','gamma');
```

```
 pg_copy_physical_replication_slot
-----------------------------------
 (gamma,)
(1 row)
```

```
α=> \c - student
```

```
You are now connected to database "student" as user "student".
```

Выложим автономную копию в каталоги PG_DATA второго и третьего серверов:

```
student$ sudo pg_ctlcluster 13 beta status
```

```
Error: /var/lib/postgresql/13/beta is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/beta
```

```
student$ sudo cp -r /home/student/backup /var/lib/postgresql/13/beta
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/beta
```

```
student$ sudo pg_ctlcluster 13 gamma status
```

```
Error: /var/lib/postgresql/13/gamma is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/gamma
```

```
student$ sudo cp -r /home/student/backup /var/lib/postgresql/13/gamma
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/gamma
```

В конфигурации третьего сервера укажем слот gamma:

```
student$ sudo sed 's/beta/gamma/g' -i /var/lib/postgresql/13/gamma/postgresql.auto.conf
```

```
student$ sudo tail -n 1 /var/lib/postgresql/13/gamma/postgresql.auto.conf
```

```
primary_slot_name = 'gamma'
```

Запускаем обе реплики:

```
student$ sudo pg_ctlcluster 13 beta start
```

```
student$ sudo pg_ctlcluster 13 gamma start
```

Слоты инициализировались:

```
α=> SELECT slot_name, active_pid, restart_lsn FROM pg_replication_slots;
```

```
 slot_name | active_pid | restart_lsn
-----------+------------+-------------
 beta      |      86956 | 0/5000060
 gamma     |      86998 | 0/5000060
(2 rows)
```

Проверяем.

```
α=> CREATE DATABASE replica_usecases;
```

```
CREATE DATABASE
```

```
α=> \c replica_usecases
```

```
You are now connected to database "replica_usecases" as user "student".
```

```
α=> CREATE TABLE revenue(city text, amount numeric);

CREATE TABLE
```

```
student$ psql -p 5433 -d replica_usecases
```

```
β=> \d revenue
```

```
               Table "public.revenue"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 city   | text    |           |          |
 amount | numeric |           |          |
```

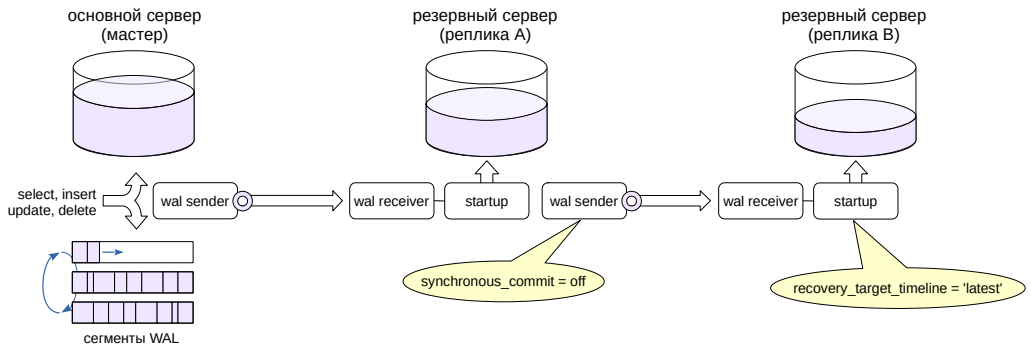
```
student$ psql -p 5434 -d replica_usecases
```

```
γ=> \d revenue
```

```
               Table "public.revenue"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 city   | text    |           |          |
 amount | numeric |           |          |
```

Мы настроили две реплики одного мастера на основе одной базовой копии.

Каскадная репликация



позволяет снизить нагрузку на мастер и перераспределить сетевой трафик

не поддерживается каскадная синхронная репликация

обратная связь поступает от всех реплик

9

Несколько реплик, подключенных к одному основному серверу, будут создавать на него определенную нагрузку. Кроме того, надо учитывать нагрузку на сеть для пересылки нескольких копий потока журнальных записей.

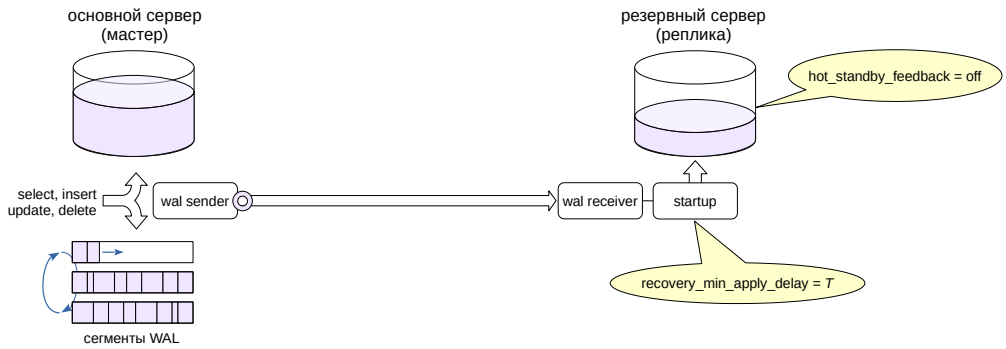
Для снижения нагрузки реплики можно соединять каскадом; при этом серверы передают журнальные записи друг другу по цепочке. Чем дальше от мастера, тем большее может накопиться запаздывание. Схема мониторинга усложняется: процесс надо контролировать на нескольких серверах.

Для настройки на промежуточных репликах требуется обеспечить достаточное значение параметров *max_wal_senders* и *max_replication_slots* и проверить настройки подключения по протоколу репликации в pg_hba.conf.

Если совершится переход на реплику (ближайшую к основному серверу), то на ней произойдет увеличение номера линии времени. Значение (по умолчанию с версии 12) *recovery_target_timeline* = 'latest' направит восстановление по этой новой линии.

Заметим, что каскадная синхронная репликация не поддерживается: основной сервер может быть синхронизирован только с непосредственно подключенной к нему репликой. А вот обратная связь поступает основному серверу от всех реплик.

Отложенная репликация



«машина времени» и возможность восстановиться на определенный момент в прошлом без архива

pg_wal_replay_pause()
pg_is_wal_replay_paused()
pg_wal_replay_resume()

Задача: иметь возможность просмотреть данные на некоторый момент в прошлом и, при необходимости, восстановить сервер на этот момент.

Обычный механизм восстановления из архива на момент времени (point-in-time recovery) позволяет решить задачу, но требует большой подготовительной работы и занимает много времени.

Другое решение — создать реплику, которая применяет журнальные записи не сразу, а через установленный интервал времени (*recovery_min_apply_delay*). Чтобы задержка работала правильно, необходима синхронизация часов между серверами.

Откладывается применение не всех записей, а только записей о фиксации изменений. Записи до фиксации могут быть применены «раньше времени», но это не представляет проблемы благодаря многоверсионности.

Обратную связь следует отключать, чтобы не вызвать разрастание таблиц на мастере.

Типичный сценарий: на основном сервере происходит какая-либо проблема (допустим, удалены критичные данные). На реплике данные еще есть, поскольку соответствующие записи еще не применены. Дальнейшее проигрывание записей приостанавливается с помощью функции pg_wal_replay_pause(), пока идет исследование проблемы.

Если принято решение вернуться на момент времени до удаления, нужно указать целевую точку восстановления одним из параметров *recovery_target_** и перезапустить реплику.

Логическая репликация

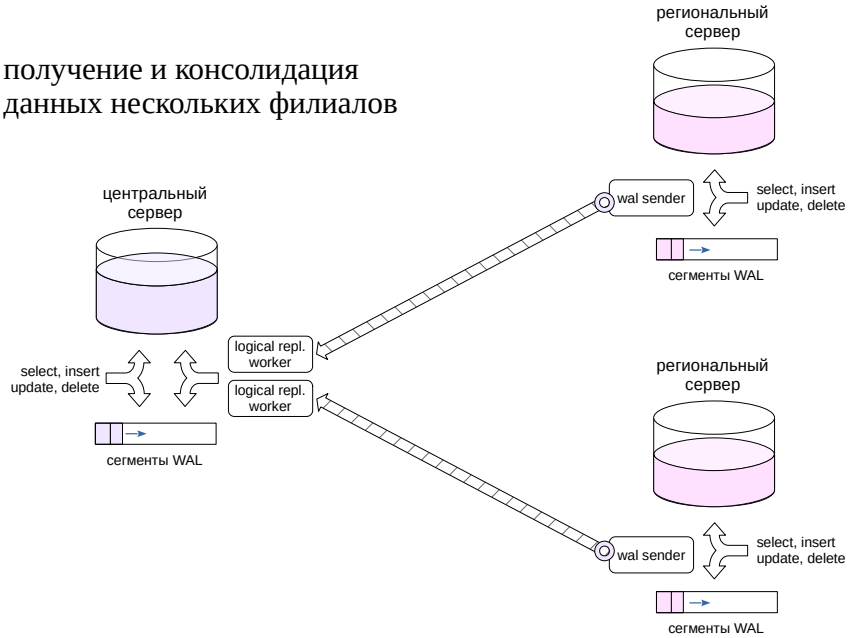
Консолидация и общие справочники

Обновление основной версии сервера

Мастер-мастер (в будущем)

Консолидация

получение и консолидация данных нескольких филиалов



Пусть имеются несколько региональных филиалов, каждый из которых работает на собственном сервере PostgreSQL. Задача состоит в консолидации части данных на центральном сервере.

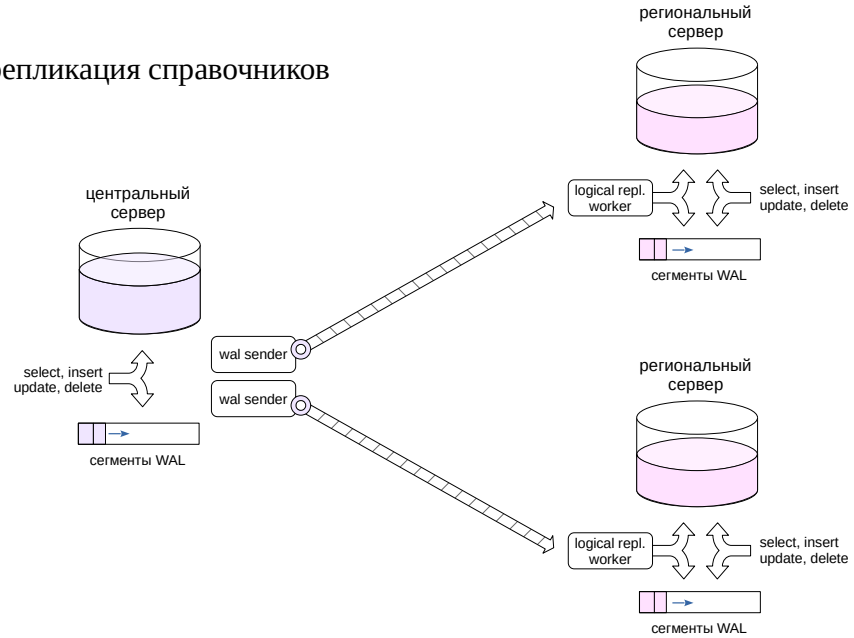
Для решения на региональных серверах создаются публикации необходимых данных. Центральный сервер подписывается на эти публикации. Полученные данные можно обрабатывать (например, приводить к единому виду) с помощью триггеров на стороне центрального сервера.

Поскольку репликация основана на передаче данных через слот, между серверами необходимо более или менее постоянное соединение, так как во время разрыва соединения региональные серверы будут вынуждены сохранять файлы журнала.

Есть множество особенностей такого процесса и с точки зрения бизнес-логики, требующих всестороннего изучения. В ряде случаев может оказаться проще передавать данные пакетно раз в определенный интервал времени.

Общие справочники

репликация справочников



Другая задача: на центральном сервере поддерживаются справочники, актуальные версии которых должны быть доступны на региональных серверах.

В этом случае схему надо развернуть наоборот: центральный сервер публикует изменения, а региональные серверы подписываются на эти обновления.

Консолидация с помощью логической репликации

Выведем обе настроенные ранее реплики из режима восстановления и настроим консолидацию данных.

```
β=> \c - postgres

You are now connected to database "replica_usecases" as user "postgres".

β=> SELECT pg_promote(), pg_is_in_recovery();

 pg_promote | pg_is_in_recovery
------------+-------------------
 t          | f
(1 row)
```

```
γ=> \c - postgres

You are now connected to database "replica_usecases" as user "postgres".

γ=> SELECT pg_promote(), pg_is_in_recovery();

 pg_promote | pg_is_in_recovery
------------+-------------------
 t          | f
(1 row)
```

Для логической репликации нужно повысить уровень WAL (потребуется рестарт).

```
β=> ALTER SYSTEM SET wal_level = 'logical';

ALTER SYSTEM
```

```
γ=> ALTER SYSTEM SET wal_level = 'logical';

ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 13 beta restart

student$ sudo pg_ctlcluster 13 gamma restart
```

Публикуем таблицу на втором и третьем серверах:

```
student$ psql -p 5433 -d replica_usecases

β=> CREATE PUBLICATION revenue FOR TABLE revenue;

CREATE PUBLICATION
```

```
student$ psql -p 5434 -d replica_usecases

γ=> CREATE PUBLICATION revenue FOR TABLE revenue;

CREATE PUBLICATION
```

Первый сервер подписывается на обе публикации:

```
α=> \c - postgres

You are now connected to database "replica_usecases" as user "postgres".

α=> CREATE SUBSCRIPTION msk CONNECTION 'port=5433 dbname=replica_usecases' PUBLICATION revenue;

NOTICE:  created replication slot "msk" on publisher
CREATE SUBSCRIPTION

α=> CREATE SUBSCRIPTION spb CONNECTION 'port=5434 dbname=replica_usecases' PUBLICATION revenue;

NOTICE:  created replication slot "spb" on publisher
CREATE SUBSCRIPTION

α=> \c - student

You are now connected to database "replica_usecases" as user "student".
```

В филиалах кипит работа:

```
β=> INSERT INTO revenue
  SELECT 'Москва', random()*1e6 FROM generate_series(1,70);

INSERT 0 70
```

```
γ=> INSERT INTO revenue
     SELECT 'Санкт-Петербург', random()*1e6 FROM generate_series(1,10);
```

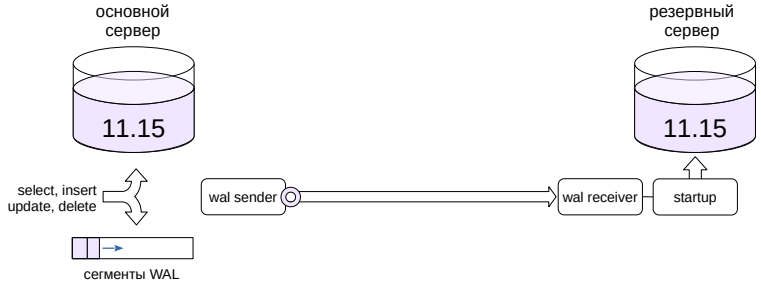
```
INSERT 0 10
```

А центральный офис видит работу всей компании (подождем несколько секунд, чтобы сработала репликация):

```
α=> SELECT city, sum(amount) FROM revenue GROUP BY city;
```

```
      city       |         sum
-----------------+----------------------
 Москва          | 36538213.52240911673
 Санкт-Петербург |    5984974.117391200
(2 rows)
```

Обновление серверов

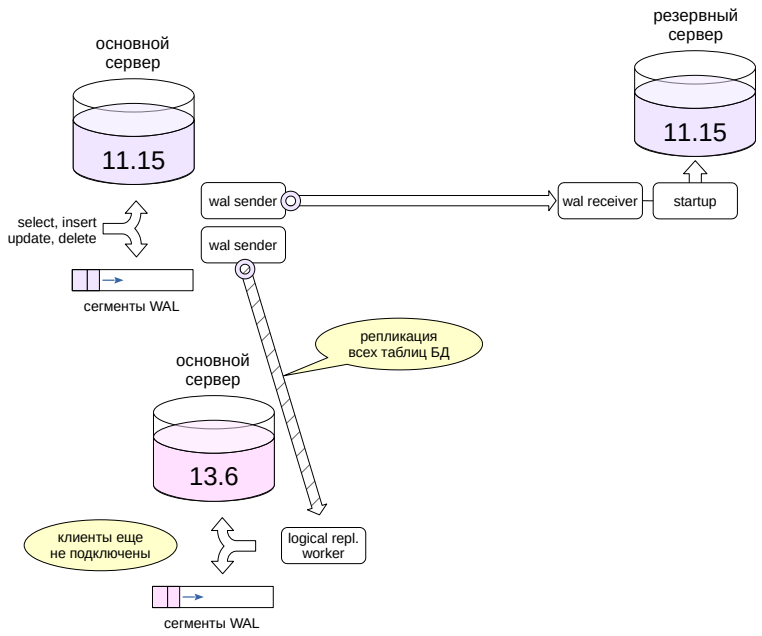


мастер и физическая реплика: требуется обновить основную версию

Логическая репликация может использоваться для обновления основной версии сервера без прерывания обслуживания (или с минимальным прерыванием).

Допустим, имеется основной сервер и физическая потоковая реплика.

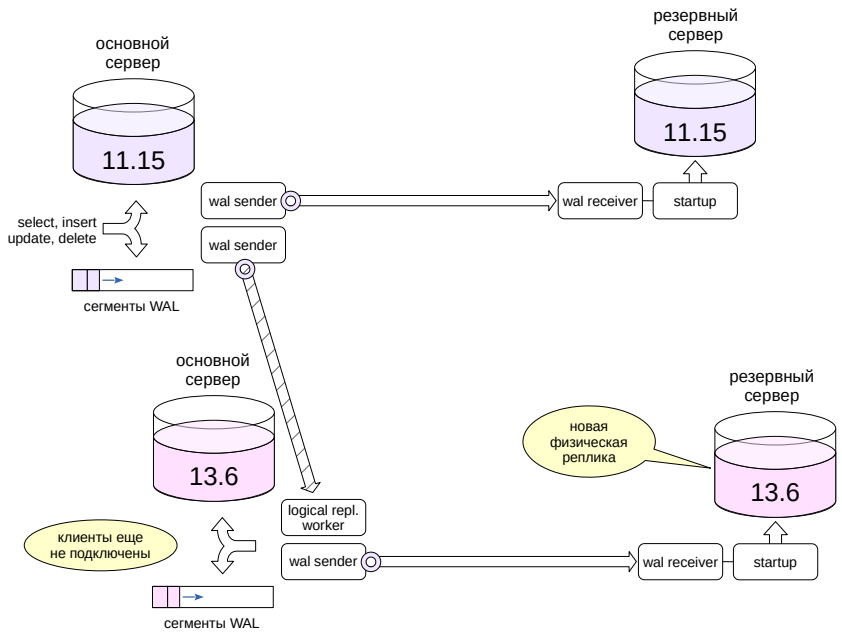
Обновление серверов



Создаем новый сервер с требуемой версией PostgreSQL и переносим на него структуру всех таблиц выбранной базы данных.

На основном сервере предыдущей версии публикуем изменения всех таблиц базы данных. Поскольку изменения схемы данных не реплицируются, на время обновления такие изменения должны быть запрещены.

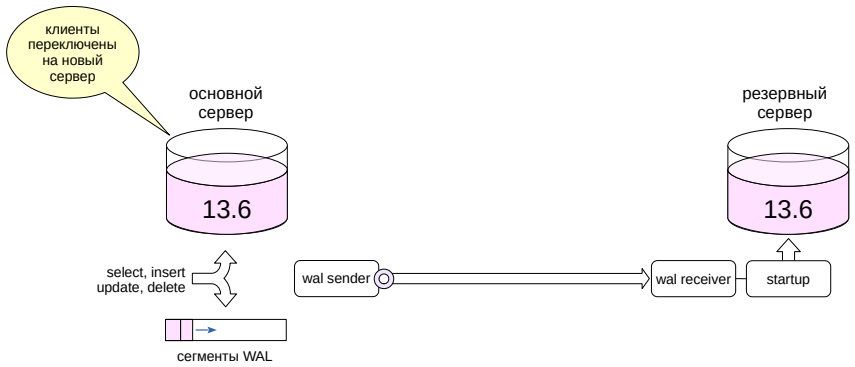
Обновление серверов



Создаем физическую реплику, аналогичную реплике сервера предыдущей версии.

В итоге получаем параллельную структуру серверов с новой версией PostgreSQL. Эта структура аналогична имеющейся для старой версии.

Обновление серверов

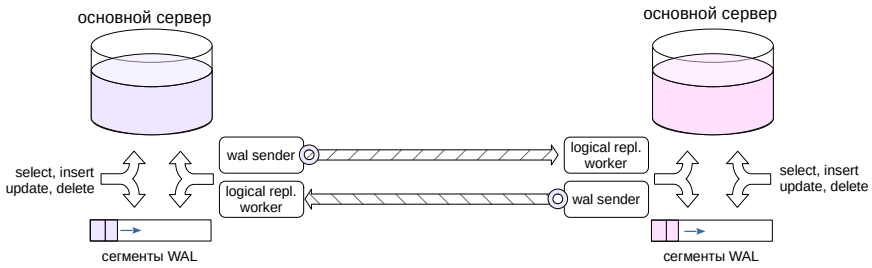


После этого переключаем клиентов на новые серверы, а старые останавливаем. Прерывание обслуживания будет определяться тем, насколько плавно можно выполнить это переключение.

Надо иметь в виду, что такой процесс нельзя считать универсальным: он накладывает достаточно много ограничений и существенно более сложен, чем использование утилиты pg_upgrade, которая позволяет выполнить обновление за небольшое время.

Мастер-мастер

кластер, в котором данные могут изменять несколько серверов
(дело будущего)



Задача: обеспечить надежное хранение данных на нескольких серверах с возможностью записи на любом сервере (что полезно, например, для георасределенной системы).

Для решения можно использовать двунаправленную репликацию, передавая изменения в одних и тех же таблицах от одного сервера к другому и обратно.

Сразу заметим, что в настоящее время средства, доступные в PostgreSQL 13, не позволяют этого реализовать, но со временем эта возможность должна будет появиться в ядре, см. расширения pg_logical (https://www.2ndquadrant.com/en/resources/pglogical/) и BDR (https://www.2ndquadrant.com/en/resources/bdr/).

Конечно, прикладная система должна быть построена таким образом, чтобы избегать конфликтов при изменении данных в одних и тех же таблицах. Например, использовать глобальные уникальные идентификаторы или гарантировать, что разные серверы работают с разными диапазонами ключей.

Надо учитывать, что система мастер-мастер, построенная на логической репликации, не обеспечивает сама по себе выполнение глобальных распределенных транзакций. При использовании синхронной репликации можно гарантировать надежность, но не согласованность данных между серверами. Кроме того, никаких средств для автоматизации обработки сбоев, подключения или удаления узлов из кластера и т. п. в PostgreSQL не предусмотрено — эти задачи должны решаться внешним средствами.

Итоги

Физическая репликация — базовый механизм для решения целого ряда задач

- обеспечение высокой доступности
- балансировка однотипной нагрузки
- разделение разных видов нагрузки
- и др.

Логическая репликация расширяет возможности

Практика

1. Настройте репликацию между серверами alpha и beta.
2. Настройте каскадную репликацию на сервер gamma с применение записей WAL с задержкой в десять секунд.
3. Проверьте работу репликации, убедитесь в том, что на сервере gamma данные появляются с установленной задержкой.
4. Остановите мастер и перейдите на сервер beta.
5. Проверьте, что сервер gamma продолжает работать в режиме реплики.

1. Настройка репликации между первым и вторым серверами

Сначала создадим базу данных.

```
student$ psql
```

```
α=> CREATE DATABASE replica_usecases;
```

```
CREATE DATABASE
```

```
α=> \c replica_usecases
```

```
You are now connected to database "replica_usecases" as user "student".
```

Создаем слот и автономную резервную копию.

```
student$ pg_basebackup --pgdata=/home/student/backup -R --create-slot --slot=replica
```

Файлы postgresql.auto.conf и standby.signal подготовлены утилитой pg_basebackup.

```
student$ cat /home/student/backup/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=student passfile=''/home/student/.pgpass'' channel_binding=prefer host=''/var/run/postgresql'' port=5432 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
```

```
student$ ls -l /home/student/backup/standby.signal
```

```
-rw------- 1 student student 0 ноя  1 23:06 /home/student/backup/standby.signal
```

Выкладываем копию и запускаем реплику:

```
student$ sudo pg_ctlcluster 13 beta status
```

```
Error: /var/lib/postgresql/13/beta is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/beta
```

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/beta
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/beta
```

```
student$ sudo pg_ctlcluster 13 beta start
```

2. Настройка репликации между вторым и третьим серверами

```
student$ psql -p 5433 -d replica_usecases
```

Создадим автономную резервную копию со второго сервера, чтобы показать возможность выполнения резервного копирования с реплики. Слот создается утилитой.

```
student$ pg_basebackup -p 5433 --pgdata=/home/student/backup -R --create-slot --slot=replica
```

Файл postgresql.auto.conf подготовлен утилитой pg_basebackup, добавляем задержку воспроизведения:

```
student$ echo "recovery_min_apply_delay = '10s'" | tee -a /home/student/backup/postgresql.auto.conf
```

```
recovery_min_apply_delay = '10s'
```

Вот что получилось:

```
student$ cat /home/student/backup/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=student passfile=''/home/student/.pgpass'' channel_binding=prefer host=''/var/run/postgresql'' port=5432 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
primary_conninfo = 'user=student passfile=''/home/student/.pgpass'' channel_binding=prefer host=''/var/run/postgresql'' port=5433 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
recovery_min_apply_delay = '10s'
```

Копию записываем в каталог PGDATA третьего сервера.

```
student$ sudo pg_ctlcluster 13 gamma status
```

```
Error: /var/lib/postgresql/13/gamma is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/gamma
```

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/gamma
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/gamma
```

Запускаем сервер gamma.

```
student$ sudo pg_ctlcluster 13 gamma start
```

3. Проверка работы

На первом сервере создадим таблицу и проверим, что она появилась сначала на одной реплике, а через 10 секунд — и на другой.

```
α=> CREATE TABLE test(s text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO test VALUES ('Привет, мир!');
```

```
INSERT 0 1
```

Проверяем одну реплику:

```
β=> SELECT * FROM test;
```

```
       s
--------------
 Привет, мир!
(1 row)
```

Проверяем другую реплику:

```
student$ psql -p 5434 -d replica_usecases
```

```
γ=> SELECT * FROM test;
```

```
ERROR:  relation "test" does not exist
LINE 1: SELECT * FROM test;
                      ^
```

Таблицы пока нет. Подождем 10 секунд...

```
γ=> SELECT * FROM test;
```

```
       s
--------------
 Привет, мир!
(1 row)
```

Таблица появилась.

4. Переход на второй сервер

Текущая линия времени на третьем сервере.

```
γ=> SELECT received_tli FROM pg_stat_wal_receiver;
```

```
 received_tli
--------------
            1
(1 row)
```

При «повышении» второго сервера номер ветви увеличится на единицу, и процедура восстановления на третьем сервере пойдет по новой ветви благодаря значению по умолчанию:

```
γ=> SELECT setting, boot_val FROM pg_settings WHERE name = 'recovery_target_timeline';
```

```
 setting | boot_val
---------+----------
 latest  | latest
(1 row)
```

```
α=> \q
```

```
student$ sudo pg_ctlcluster 13 alpha stop
```

```
student$ sudo pg_ctlcluster 13 beta promote
```

Линия времени на третьем сервере сменилась.

```
γ=> SELECT received_tli FROM pg_stat_wal_receiver;
```

```
 received_tli
--------------
            2
(1 row)
```

5. Проверка работы

На втором сервере добавим в таблицу строки и проверим, что они появились на третьем сервере через 10 секунд.

```
β=> INSERT INTO test VALUES ('После перехода на второй сервер');
```

```
INSERT 0 1
```

Проверяем:

```
γ=> SELECT * FROM test;
```

```
      s
--------------
 Привет, мир!
(1 row)
```

Данных пока нет. Ждем 10 секунд...

```
γ=> SELECT * FROM test;
```

```
                  s
--------------------------------
 Привет, мир!
 После перехода на второй сервер
(2 rows)
```

Данные появились.