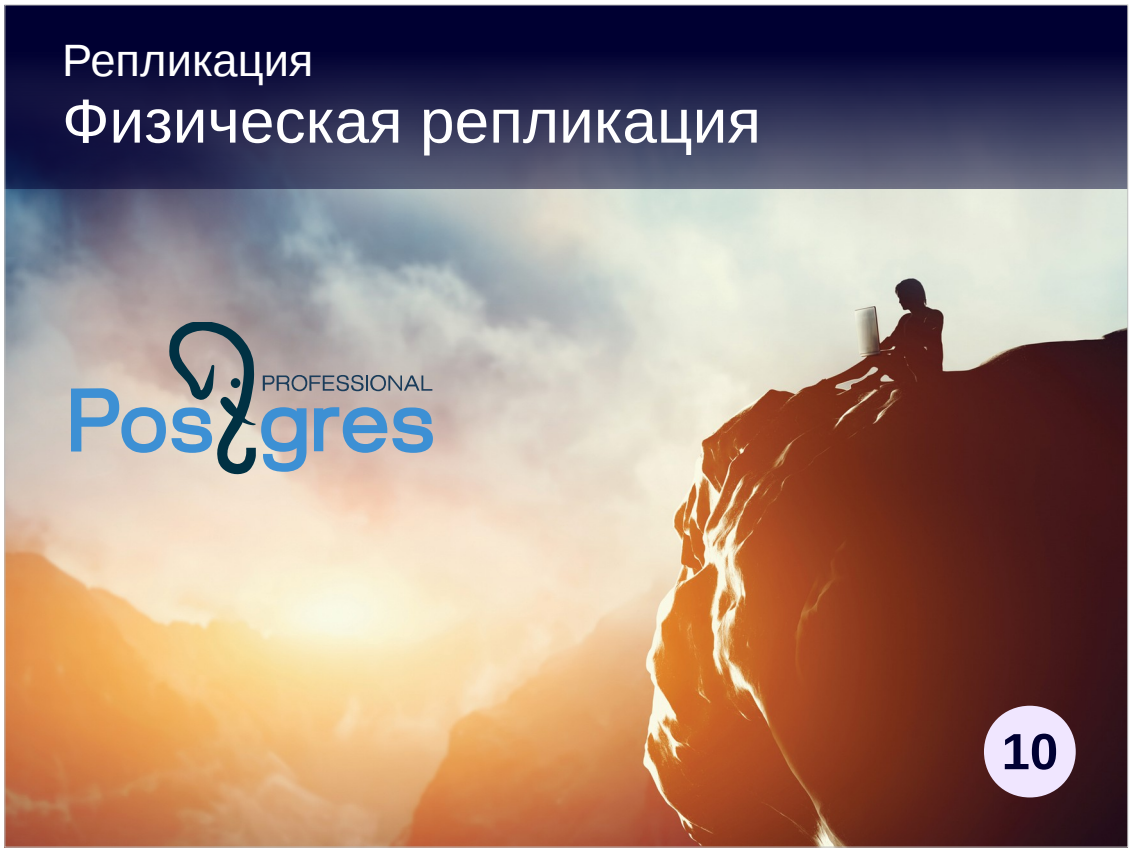


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Задачи репликации

Схема работы физической репликации

Способы доставки журнальных записей

Особенности и ограничения использования реплики

Синхронная и асинхронная репликация

Мониторинг репликации

Возможные проблемы и способы их решения

Задачи репликации

Репликация

процесс синхронизации нескольких копий кластера баз данных на разных серверах

Базовый механизм для решения ряда задач

отказоустойчивость	при возникновении сбоя система должна сохранить работоспособность
масштабируемость	распределение нагрузки между серверами

Одиночный сервер, управляющий базами данных, может не удовлетворять требованиям, предъявляемым к системе. Есть две основных задачи, для решения которых требуется наличие нескольких серверов.

Во-первых, отказоустойчивость: один физический сервер — это возможная точка отказа. Если сервер выходит из строя, система становится недоступной.

Во-вторых, производительность. Один сервер может не справляться с нагрузкой. Зачастую желательна возможность не вертикального, а горизонтального масштабирования — распределения нагрузки на несколько серверов. Дело может быть как в стоимости аппаратуры, так и в необходимости различных настроек для разных типов нагрузки (например, для OLTP и отчетности).

В случае PostgreSQL распределенные системы строятся по принципу «shared nothing»: несколько серверов работают независимо друг от друга и не имеют ни общей оперативной памяти, ни общих дисков. Следовательно, если серверы должны работать с одними и теми же данными, то эти данные требуется синхронизировать между ними. Механизм синхронизации и называется репликацией.

Мастер → реплика

Записи WAL передаются на реплику и применяются

- поток данных только в одну сторону
- реплицируется кластер целиком, выборочная репликая невозможна

Реплика — точная копия мастера

- одна и та же основная версия сервера
- полностью совместимые архитектуры и платформы

Реплика доступна только для чтения

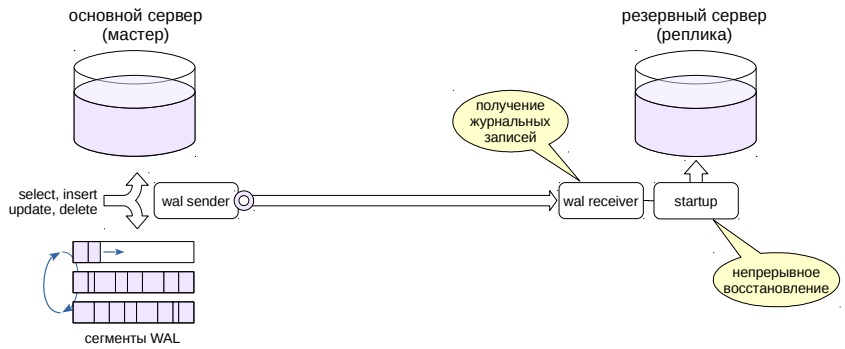
При физической репликации, рассматриваемой в этой теме, серверы имеют назначенные роли: один является ведущим («мастер») и один или несколько серверов являются ведомыми («реплики»).

Мастер передает на реплику журнальные записи, а реплика применяет эти записи к своим файлам данных. Применение происходит чисто механически, без «понимания смысла» изменений, и выполняется быстрее, чем работало бы повторное выполнение операторов SQL. Но при этом критична двоичная совместимость между серверами — они должны работать на одной и той же программно-аппаратной платформе и на них должны быть запущены одинаковые основные версии PostgreSQL.

Поскольку журнал является общим для всего кластера, то и реплицировать можно только кластер целиком, а не отдельные базы данных или таблицы. Возможность «отфильтровать» журнальные записи отсутствует.

Реплика не может генерировать собственных журнальных записей, она лишь применяет записи мастера. Поэтому при таком подходе реплика может быть доступна только для чтения — никакие операции, изменяющие данные, на реплике не допустимы.

Потоковая репликация



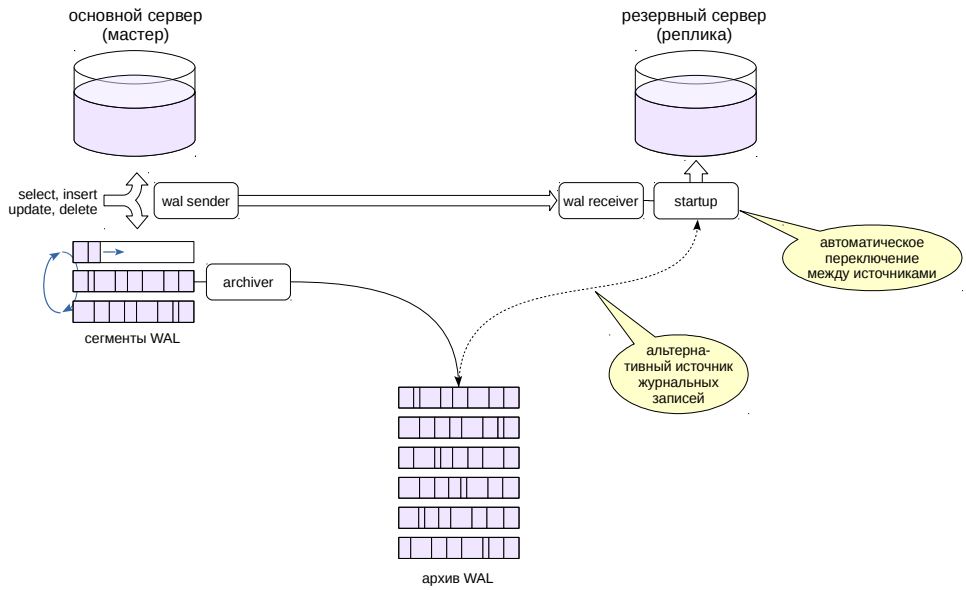
Есть два способа доставки журналов от мастера к реплике. Основной, который используется на практике — потоковая репликация.

В этом случае реплика подключается к мастеру по протоколу репликации и читает поток записей WAL. За счет этого при потоковой репликации отставание реплики от мастера минимально (и при необходимости может быть сведено к нулю).

На реплике за прием журнальных записей отвечает процесс wal receiver. Запустившись, он подключается к мастеру по протоколу репликации, а мастер запускает процесс wal sender, который обслуживает это соединение. Полученные записи пишутся на диск в виде сегментов WAL так же, как это происходит на мастере. Далее они (сразу или с задержкой) применяются к файлам данных. За применение отвечает процесс startup — тот же самый, который обеспечивает восстановление после сбоев, только теперь работает постоянно.

Потоковая репликация может, если нужно, использовать слот репликации.

Репликация с архивом



Другой способ даставки журнальных записей состоит в том, чтобы предоставить реплике доступ к архиву журналов — точно так же, как при обычном восстановлении из архива (этот механизм рассмотрен в модуле «Резервное копирование»).

Если реплика по каким-то причинам не сможет получить очередную журнальную запись по протоколу репликации (например, из-за обрыва связи), она попробует прочитать ее из архива с помощью команды *restore_command* из файла recovery.conf. При восстановлении связи реплика снова автоматически переключится на использование потоковой репликации.

В принципе, репликация может работать и с одним только архивом, без потоковой репликации. Но в этом случае:

- реплика вынужденно отстает от мастера на время заполнения сегмента;

- мастер ничего не знает о существовании реплики, что в некоторых случаях может привести к проблемам.

Использование реплики

Допускаются

- запросы на чтение данных (select, copy to, курсоры)
- установка параметров сервера (set, reset)
- управление транзакциями (begin, commit, rollback...)
- создание резервной копии (pg_basebackup)

Не допускаются

- любые изменения (insert, update, delete, truncate, nextval...)
- блокировки, предполагающие изменение (select for update...)
- команды DDL (create, drop...), в том числе создание временных таблиц
- команды сопровождения (vacuum, analyze, reindex...)
- управление доступом (grant, revoke...)

Как мы уже говорили, на реплике не допускается любое изменение данных. Более точно, не допускаются:

- изменения таблиц, материализованных представлений, последовательностей;

- блокировки (так как для этого требуется изменение страниц данных);

- команды DDL, включая создание временных таблиц;

- такие команды, как vacuum и analyze;

- команды управления доступом.

При этом реплика может выполнять запросы на чтение данных, если установлен параметр *hot_standby* = on. Также будет работать установка параметров сервера и команды управления транзакциями — например, можно начать (читающую) транзакцию с нужным уровнем изоляции.

Кроме того, реплику можно использовать и для изготовления резервных копий.

Ограничения

Изоляция и многоверсионность

- большое число точек сохранения задерживает выполнение запросов
- не поддерживается уровень изоляции serializable

Триггеры и блокировки

- триггеры и пользовательские блокировки (advisory locks) не работают

Резервное копирование с реплики

- параметр *full_page_writes* должен быть заранее включен на мастере
- задержки из-за ожидания контрольной точки и переключения сегментов

Изменение табличных пространств

- каталоги на реплике нужно создавать заранее, так как прав пользователя ОС postgres может не хватить

У репликации есть ряд ограничений и особенностей.

Если на мастере выполняется транзакция с числом вложенных транзакций > 64 (то есть активно используются точки сохранения), создание снимка на реплике будет приостановлено — это означает невозможность некоторое время выполнять запросы.

Не поддерживается уровень изоляции serializable на реплике.

Триггеры не будут срабатывать на реплике (они выполнились на мастере и реплицируется уже результат их работы). Не будут работать и рекомендательные (advisory) блокировки.

При выполнении резервного копирования с реплики нет технической возможности полноценно управлять мастером, в частности, вызывать контрольные точки и включать параметр *full_page_writes*. Поэтому параметр нужно включить на мастере заранее, а контрольную точку приходится ждать (что может существенно задержать процесс).

Создание табличных пространств реплицируется, но, если у пользователя ОС, под которым работает СУБД, недостаточно прав для создания каталога, произойдет ошибка. Лучше всего сначала создать каталоги на мастере и на всех репликах, а уже затем выполнять команду CREATE TABLESPACE.

До версии 9.6 включительно хеш-индексы не журналировались и соответственно не реплицировались. Начиная с версии 10 это исправлено.

Процессы

Мастер	Реплика
wal sender	wal receiver
	startup
archiver	archiver (*archive_mode* = always)
checkpointer (контрольные точки)	checkpointer (точки рестарта)
writer	writer
stats collector	stats collector
wal writer	
autovacuum	

На мастере и на реплике выполняется разный набор процессов. Кроме уже расмотренных wal sender (на мастере) и wal reciever и startup (на реплике), работают также другие процессы.

Общие процессы:

- **background writer** — записывает грязные страницы из буферного кэша на диск (на реплике, для того, чтобы применить журнальную запись, страницы точно так же читаются в буферный кэш и затем записываются в фоновом режиме или при вытеснении).

- **stats collector** — статистика не передается в журнале, а собирается на реплике отдельно.

- **checkpointer** — на реплике при получении журнальной записи о контрольной точке выполняется похожая процедура — точка рестарта. Если в процессе восстановления случится сбой, реплика сможет продолжить с последней точки рестарта, а не с самого начала.

Процессы только на мастере:

- **wal writer** — реплика не создает журналы упреждающей записи, а только получает их с мастера.

- **autovacuum launcher/worker** — реплика не выполняет очистку, ее выполнение на мастере передается реплике через журнал.

Есть также процесс, наличие которого зависит от настройки:

- **archiver** — при значении *archive_mode* = on выпоняется только на мастере, а при *archive_mode* = always также и на реплике (подробнее рассматривается в теме «Переключение на реплику»).

Режимы репликации

Асинхронный

synchronous_commit = off
local

Синхронный

фиксация с ожиданием подтверждения от синхронной реплики

обеспечивает надежность, но не согласованность

synchronous_commit = remote_write
on
remote_apply

synchronous_standby_names = FIRST *N* (*список реплик*)
ANY *N* (*список реплик*)

Как известно, фиксация транзакций может выполняться в двух режимах. В более быстром асинхронном режиме есть шанс потерять уже зафиксированные изменения в случае сбоя. В синхронном режиме команда COMMIT не завершается, пока запись не дойдет до диска. При наличии нескольких серверов надежность можно еще более увеличить, дожидаясь подтверждения от реплики о получении записи, тогда данные не пропадут даже при выходе из строя носителя.

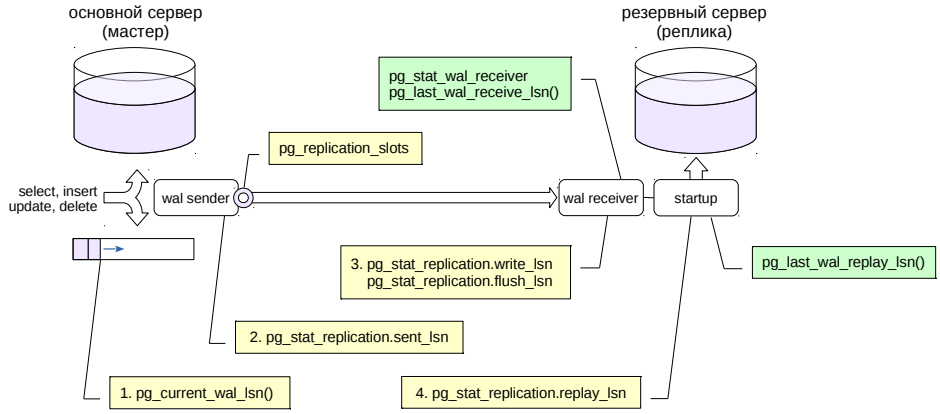
Синхронная репликация включается тем же параметром *synchronous_commit,* что и синхронная фиксация. Разные значения этого параметра позволяют управлять уровнем надежности. При значении **remote_write** мастер ожидает подтверждения о получении записи (остается шанс потерять изменения, если на реплике случится сбой и она не успеет записать данные на диск).

При значении **on** мастер ожидает подтверждения о попадании журнальной записи на диск реплики (это надежный режим; однако приложение, обратившись к реплике, может не увидеть изменений).

Наконец, значение **remote_apply** заставляет мастер дождаться применения записи на реплике. Каждый следующий режим вызывает все большие задержки.

Мастер может синхронизироваться как с одной, так и с несколькими (начиная с 9.6) репликами. Можно указать как список реплик в порядке приоритета, так и организовать синхронизацию на основе *кворума* (начиная с 10), при которой мастер дожидается подтверждения от любых *N* реплик из числа доступных.

Мониторинг



1 – 2 = задержка записи в поток на мастере
2 – 3 = задержка получения данных репликой
3 – 4 = задержка применения данных на реплике

В ходе репликации возможны проблемы, о которых должен предупредить заранее настроенный мониторинг. Основной информацией являются позиции в журнале предзаписи. Можно выделить четыре важные точки:

1. появление журнальной записи на мастере;

2. трансляция записи процессом wal sender;

3. получение записи процессом wal reciver;

4. применение полученной записи процессом startup.

Все эти точки обычно отслеживаются на мастере. Первую точку дает функция pg_current_wal_lsn(), остальные — представление pg_stat_replication. Реплика передает мастеру статус репликации при каждой записи на диск, но как минимум раз в *wal_receiver_status_interval* секунд (по умолчанию — 10 секунд).

Если используется слот репликации, то информацию о нем можно получить из представления pg_replication_slots.

Данные о ходе репликации можно получить и на реплике в представлении pg_stat_wal_receiver и с помощью функций pg_last_wal_receive_lsn() и pg_last_wal_replay_lsn().

Возможные проблемы

Мастер может удалить файл журнала, нужный реплике

если реплика не успеет его получить
(например, будет остановлена на некоторое время)

1. Слот репликации

вводит зависимость мастера от реплики, требует мониторинга

2. Архив журнала предзаписи

позволяет обойтись без слота

При репликации возможен ряд сложностей, которые можно решать разными способами.

Сервер PostgreSQL периодически удаляет файлы журнала, которые не требуются для восстановления. При этом мастер может удалить файл, который содержит данные, еще не переданные реплике.

Это не проблема, если используется архив журналов предзаписи, поскольку реплика, получив отказ по протоколу репликации, прочитает необходимые ей данные из архива, а затем, «догнав» мастер, снова переключится на получение данных из потока.

Но если архива нет, реплика не сможет продолжать восстановление и ее придется пересоздавать из новой резервной копии. Чтобы этого избежать, можно использовать слот репликации. Однако надо понимать, что создание слота ставит мастер в зависимость от реплики: если реплика не будет получать журнальные записи, они будут накапливаться на мастере и свободное пространство рано или поздно закончится. Поэтому слот — еще одна точка, которую необходимо включать в мониторинг.

Возможные проблемы

Записи WAL, несовместимые с запросами на реплике

- очистка удаляет версии строк, нужные запросам на реплике
- эксклюзивные блокировки (например, при удалении объектов)
- мониторинг: pg_stat_database_conflicts

1. Обратная связь по протоколу репликации
   - только для очистки: выполнение запроса на реплике предотвращает удаление версий строк так же, как и локальные запросы
   - *hot_standby_feedback* = on
2. Откладывание применения конфликтующих записей
   - *max_standby_streaming_delay*
   - *max_standby_archive_delay*

Вторая проблема возникает, если реплика используется для выполнения запросов. Мастер может выполнить действие, несовместимое с запросом на реплике:

- очистка может удалить версию строки, которая используется запросом;

- может прийти запись об эксклюзивной блокировке объекта (например, при удалении таблицы).

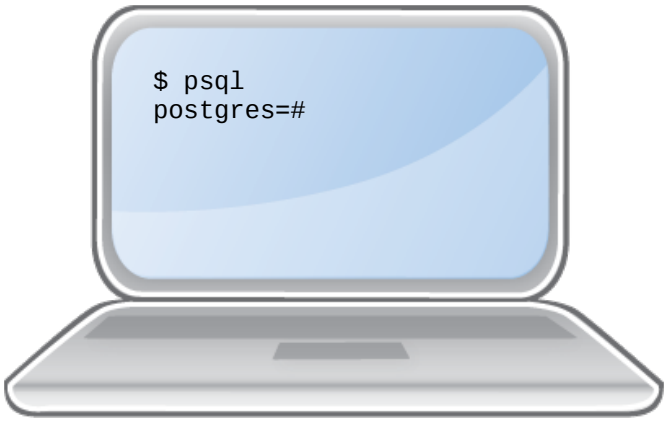
Возникновение таких конфликтов можно отслеживать с помощью представления pg_stat_database_conflicts.

Один из путей решения конфликта — откладывать несовместимые действия на мастере. Для этого включается *обратная связь,* с помощью которой мастер понимает, какие версии строк нужны реплике. В таком случае (с точки зрения очистки) запрос на реплике ничем не отличается от запроса на самом мастере.

Этот способ работает только для очистки (эксклюзивную блокировку отложить не получится) и только в случае потоковой репликации (если реплика переключается на получение записей из архива, она фактически перестает существовать для мастера).

Другой путь решения — откладывать применения полученных несовместимых журнальных записей на реплике. Этот способ работает как для потоковой репликации, так и для архива, и как для очистки, так и для эксклюзивных блокировок. В параметрах устанавливается максимальная задержка применения (от времени получения записи). Если за это время запрос не успеет завершиться, он будут прерван.

Демонстрация



Итоги

Механизм репликации основан на передаче журнальных записей на реплику и их применении

- основной режим: потоковая репликация
- может быть поддержана архивом журнальных записей

Реплика — точная копия мастера

- не генерирует собственные записи WAL, доступна только для чтения

Сложный механизм, требующий настройки и мониторинга

- настройка существенно зависит от решаемых задач

Практика

1. Настройте физическую потоковую репликацию между двумя серверами в синхронном режиме, без обратной связи.
2. Проверьте работу репликации. Убедитесь, что при остановленной реплике фиксация не завершается.
3. Воспроизведите ситуацию, при которой запрос на реплике прерывается из-за очистки версий строк на мастере.
4. Запретите откладывать применение конфликтующих изменений; проверьте, что запрос отменяется сразу.
5. Включите обратную связь и убедитесь, что очистка на мастере не прерывает выполнение запроса.

1. Для этого на мастере установите параметры:

- synchronous_commit = on,

- synchronous_standby_names = 'replica',

а на реплике в файле recovery.conf в параметр primary_conninfo добавьте «application_name=replica».

3. Для этого на реплике можно начать транзакцию с уровнем изоляции repeatable read и периодически выполнять запросы к таблице, а на мастере изменить эту табицу и выполнить очистку.

Учтите, что параметр max_standby_streaming_delay по умолчанию имеет значение 30 секунд.

4. Установите параметр max_standby_streaming_delay в ноль.

5. Установите на реплике параметр hot_standby_feedback = on. Максимальный интервал оповещений устанавливается в параметре wal_receiver_status_interval (по умолчанию 10 секунд).