Ай-ТИ Фреш

ТЕХНИЧЕСКИЙ РАЗБОР

PostgreSQL 18: remote_apply, когда SELECT не видит COMMIT

WAL, сброшенный на диск реплики, ещё не значит «строка видна SELECT»

Июль 2026

itfresh.ru · ИТ-аутсорсинг для юридических лиц

О ЧЁМ ЭТОТ РАЗБОР

Суть проблемы

Синхронная репликация включена ради защиты от потери транзакций при отказе primary. Приложение делает COMMIT и тут же читает данные с реплики — строки нет. Причина не в сети: synchronous_commit=on ждёт flush WAL на реплике, но не ждёт replay записей в таблицы, а видимость для SELECT даёт только replay. Точечный remote_apply закрывает разрыв, не удорожая поток коммитов.

Почему это важно бизнесу

- Кассовый или складской заказ на секунду «пропадает» на реплике — оператор создаёт дубликат вручную
- Отчёты на read-реплике расходятся с данными primary ровно в момент сверки
- Оплаты, отгрузки и медицинские записи требуют строгой согласованности чтения после записи
- Разбор инцидента затягивается: мониторинг показывает лаг 0 мс по flush, а проблема в replay
- Неверный уровень synchronous_commit либо режет производительность, либо не даёт нужной гарантии

ЧТО МЫ ЗАКЛАДЫВАЕМ

Ключевые параметры реализации

18.x

версия кластера, на которой проверяем поведение remote_apply перед внедрением

PostgreSQL 18, разд. 19.5

5 значений

synchronous_commit: off/local/remote_write/on/remote_apply, по умолчанию on

PostgreSQL 18, разд. 19.5

30 с

max_standby_streaming_delay по умолчанию — до отмены конфликтующего запроса на реплике

PostgreSQL 18, разд. 19.6

10 с

wal_receiver_status_interval — с этим шагом реплика шлёт write/flush/replay LSN

PostgreSQL 18, разд. 19.6

60 с

wal_receiver_timeout — порог обрыва соединения приёмника WAL при молчании primary

PostgreSQL 18, разд. 19.6

ANY 2

наш шаблон synchronous_standby_names для кворума из трёх реплик

PostgreSQL 18, разд. 26.2.8.2

РЕАЛИЗАЦИЯ 1 ИЗ 3

Настройка synchronous_commit под read-after-write

Что настраиваем

Кластер PostgreSQL 18: 1 primary + 2–3 synchronous standby, учётные и складские базы клиента

Как мы это делаем

1. Помечаем транзакции, которым нужна видимость: SET LOCAL synchronous_commit = remote_apply перед COMMIT оплаты или резерва склада
2. Фоновый трафик (логи, аналитика) оставляем на synchronous_commit = on — ожидание replay дороже ожидания flush
3. synchronous_standby_names = 'ANY 2 (standby1, standby2, standby3)' — кворум вместо приоритетного списка FIRST
4. Чтение сразу после записи шлём на primary либо на реплику, у которой pg_last_wal_replay_lsn() догнал LSN коммита
5. Разрыв LSN меряем под нагрузкой pgbench: pg_current_wal_lsn() против pg_last_wal_replay_lsn() до прод-релиза

РЕЗУЛЬТАТ

Связка «оплата — проверка статуса» перестаёт терять свежие строки, а фоновая аналитика не платит лишней задержкой коммита там, где строгая согласованность чтения не требуется

КЛЮЧЕВОЙ НЮАНС

remote_apply на весь кластер заставляет каждый COMMIT ждать самую медленную реплику по replay, а не по flush — это самый дорогой уровень, включаем его только на нужных транзакциях

РЕАЛИЗАЦИЯ 2 ИЗ 3

Мониторинг разрыва write/flush/replay LSN

Что настраиваем

Все синхронные и асинхронные реплики кластера, метрики снимаются с primary из pg_stat_replication

Как мы это делаем

1. На primary опрашиваем pg_stat_replication: sent_lsn, write_lsn, flush_lsn, replay_lsn, write_lag, flush_lag, replay_lag, sync_state
2. Байтовый лаг применения считаем как pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn) — это и есть невидимый COMMIT в байтах
3. На самой реплике снимаем pg_last_wal_replay_lsn() и pg_last_xact_replay_timestamp() — отставание в секундах для отчёта
4. Алерты на flush_lag и replay_lag заводим раздельно: flush_lag около нуля при растущем replay_lag — прямой признак разрыва чтения
5. sync_state отслеживаем в значениях sync/quorum/potential/async: уход реплики в potential меняет состав кворума молча

РЕЗУЛЬТАТ

На дашборде видно две разные величины — задержку записи WAL и задержку видимости данных, вместо одного усреднённого «лага репликации», который маскирует ровно ту проблему, из-за которой открыт тикет

КЛЮЧЕВОЙ НЮАНС

flush_lag и replay_lag в pg_stat_replication — разные метрики; жалобу «COMMIT прошёл, SELECT не видит» разбираем строго по replay_lag и replay_lsn

РЕАЛИЗАЦИЯ 3 ИЗ 3

Отказоустойчивость кворума при деградации реплики

Что настраиваем

Продовые кластеры с 2–3 синхронными standby на разных площадках клиента

Как мы это делаем

1. synchronous_standby_names задаём как ANY N вместо одиночного имени — потеря одной реплики не останавливает запись
2. max_standby_streaming_delay и max_standby_archive_delay поднимаем выше дефолтных 30 с только на отчётных репликах
3. Отложенную копию держим отдельным async standby с recovery_min_apply_delay = 15min — защита от логической порчи данных
4. Гасим одну реплику на учениях: при ANY 2 из 3 коммиты продолжаются, пока живы любые две реплики
5. Прогоняем pg_promote() на кандидате и сверяем, что новый primary не потерял транзакции, подтверждённые remote_apply

РЕЗУЛЬТАТ

Кластер переживает деградацию одной реплики без остановки записи и без потери подтверждённых клиенту транзакций, а delayed standby закрывает сценарий UPDATE или DELETE без WHERE

КЛЮЧЕВОЙ НЮАНС

При synchronous_commit = remote_apply каждый COMMIT ждёт применения WAL, поэтому реплику с recovery_min_apply_delay держим асинхронной и вне synchronous_standby_names

ЧТО МЫ ВЫУЧИЛИ НА ПРАКТИКЕ

Подводные камни

- ✕ **on принимают за гарантию видимости**
  on ждёт flush WAL на реплике, но не replay: SELECT сразу после COMMIT может не увидеть строку. Для read-after-write нужен remote_apply
- ✕ **remote_apply включают на весь кластер**
  Коммит начинает ждать самую медленную реплику по replay, а не по flush. Включаем через SET LOCAL только в нужных транзакциях
- ✕ **FIRST там, где нужен ANY**
  FIRST требует живыми именно первые в списке реплики: падение одной блокирует запись, хотя кворум есть. Переводим на ANY N
- ✕ **hot_standby_feedback не включён**
  По умолчанию off, и долгие SELECT на реплике падают с conflict with recovery. Включаем на отчётных репликах, следя за bloat на primary
- ✕ **мониторят только sync_state**
  Значение sync ничего не говорит о величине отставания. Нужен отдельный алерт на replay_lag и pg_wal_lsn_diff по replay_lsn
- ✕ **delay ставят на синхронную реплику**
  С recovery_min_apply_delay и remote_apply каждый COMMIT ждёт окончания задержки применения. Такую реплику держим асинхронной
- ✕ **wal_level занижают до minimal**
  При minimal требуется max_wal_senders = 0, физическая репликация невозможна. Проверяем wal_level = replica (дефолт) или logical
- ✕ **путают flush_lsn и replay_lsn**
  Разница по flush_lsn показывает нулевую задержку записи, тогда как проблема в задержке replay. В разборе смотрим обе величины

УРОВНИ РЕАЛИЗАЦИИ

Как правильно

МИНИМУМ

- synchronous_commit = on и хотя бы одна реплика в synchronous_standby_names
- Базовый съём pg_stat_replication: sync_state, state и состав активных реплик
- wal_level = replica и физический слот репликации на каждую реплику

НОРМАЛЬНО

- synchronous_standby_names в форме ANY N для кворума из 2–3 реплик
- Раздельные алерты на flush_lag и replay_lag вместо общего «лага репликации»
- hot_standby_feedback = on на репликах, которые держат отчётные SELECT
- max_standby_streaming_delay подобран под профиль запросов конкретной реплики

ХОРОШО

- SET LOCAL synchronous_commit = remote_apply для read-after-write транзакций
- Маршрутизация чтения по replay LSN вместо случайного round-robin
- Async standby с recovery_min_apply_delay как защита от логической порчи
- Учения pg_promote() с проверкой сохранности данных, подтверждённых remote_apply

ПРОВЕРЬТЕ У СЕБЯ

Чек-лист самопроверки

- [ ] Зафиксировано, какие транзакции требуют synchronous_commit = remote_apply, а какие обойдутся on?
- [ ] synchronous_standby_names задан как ANY-кворум, а не как единственная точка отказа FIRST 1?
- [ ] Мониторинг разделяет flush_lag и replay_lag, а не показывает один усреднённый лаг?
- [ ] hot_standby_feedback включён там, где реплика держит длинные отчётные запросы?
- [ ] Есть отдельный delayed standby с recovery_min_apply_delay вне синхронного кворума?
- [ ] Чтение сразу после записи маршрутизируется по replay LSN, а не случайным round-robin?
- [ ] wal_level, max_wal_senders и слоты репликации перепроверены после апгрейда до 18.x?
- [ ] Проводились учения с выключением синхронной реплики без остановки записи на primary?

Если хотя бы на два вопроса ответ «нет» или «не знаю» — тема требует внимания.

ЧТО МЫ ДЕЛАЕМ

Как поможет ITFresh

ITFresh — ИТ-аутсорсинг для юридических лиц до 50 рабочих мест в Москве и области. 15+ лет практики, собственная инфраструктура в дата-центре МТС (8 серверов Dell Xeon Platinum).

- Проектируем и настраиваем синхронную и асинхронную репликацию PostgreSQL под нагрузку клиента
- Разводим read-after-write трафик приложения между primary и репликами по replay LSN
- Строим мониторинг с раздельными метриками flush_lag и replay_lag и алертами до жалоб
- Настраиваем кворум ANY N и delayed standby как защиту от логических ошибок в данных
- Проводим учения pg_promote() и аудит конфигурации WAL и репликации при апгрейде кластера

15+
лет в ИТ-поддержке

50
рабочих мест — наш профиль

МТС
дата-центр, Москва

КОНТАКТЫ

Обсудить вашу задачу

Сайт	**itfresh.ru**
Телефон	**+7 903 729-62-41**
Telegram	**@ITfresh_Boss**

Бесплатно посмотрим вашу инфраструктуру по этому чек-листу и скажем, где тонко — без обязательств.

itfresh.ru

ДОКУМЕНТАЦИЯ И НАШ СТЕК

Техническая база

01 19.5 Write Ahead Log — synchronous_commit, wal_level (postgresql.org — 18)

02 19.6 Replication — hot_standby_feedback, standby delay (postgresql.org — 18)

03 26.2.8 Synchronous Replication (postgresql.org — 18)

04 26.2.8.2 Multiple Synchronous Standbys — FIRST и ANY (postgresql.org — 18)

05 26.2.6 Replication Slots и 26.2.7 Cascading Replication (postgresql.org — 18)

06 26.4.2 Handling Query Conflicts (postgresql.org — 18)

07 27.2.4 pg_stat_replication — write/flush/replay lag (postgresql.org — 18)

08 28.4 Asynchronous Commit (postgresql.org — 18)

Основано на официальной документации продуктов и нашей практике внедрения.