

# Журналирование Настройка журнала



## Авторские права

© Postgres Professional, 2016–2025

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

## 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](mailto:edu@postgrespro.ru)

## Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, 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.

Уровни журнала и решаемые задачи

Надежность записи

Производительность

## Настройка

 `wal_level = replica`

чем выше уровень,  
тем больше информации  
в журнале

## Minimal

восстановление после сбоя

## Replica

восстановление из резервной копии, репликация

+ операции массовой обработки данных, блокировки,  
номера выполняющихся транзакций

## Logical

логическая репликация

+ информация для логического декодирования

3

Кроме основной задачи — восстановления согласованности после сбоя — журнал можно использовать и для других целей, если добавить в него дополнительную информацию. Для этого существуют несколько уровней журнала, устанавливаемых параметром `wal_level`.

Уровень **minimal** содержит только информацию, необходимую для восстановления после сбоя. В журнал не записываются операции массовой обработки данных (такие, как `CREATE TABLE AS SELECT`, `CREATE INDEX` и т. п.), их долговечность гарантируется немедленной записью данных на диск.

Уровень **replica** (используется по умолчанию) позволяет восстанавливать систему из горячих резервных копий, сделанных утилитой `pg_basebackup`. Для этого в журнал записываются все изменения данных, включая операции массовой обработки. Уровень также позволяет организовать репликацию — передавать поток журнальных записей на другой сервер (эти темы рассматриваются в курсе DBA3), для чего в журнал включается информация о ряде блокировок.

Уровень **logical** используется для логической репликации — он должен быть включен на сервере, публикующем изменения. Для правильной работы логического декодирования в журнал записывается дополнительная информация, позволяющая корректно расшифровать смысл операций по журнальным записям.

Чем выше уровень, тем больше информации попадает в журнал и, следовательно, тем больше его объем.

## Уровни журнала

Используем новую базу данных.

```
=> CREATE DATABASE wal_tuning;
```

```
CREATE DATABASE
```

```
=> \c wal_tuning
```

You are now connected to database "wal\_tuning" as user "student".

```
=> CREATE EXTENSION pg_walinspect;
```

```
CREATE EXTENSION
```

Уровень журнала по умолчанию — replica.

```
=> SHOW wal_level;
```

```
wal_level
-----
replica
(1 row)
```

Посмотрим, как записывается в журнал команда CREATE TABLE AS SELECT.

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/1FD7528
(1 row)
```

```
=> CREATE TABLE t_wal(n) AS SELECT 1 FROM generate_series(1,1000);
```

```
SELECT 1000
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/1FE81F8
(1 row)
```

Объем журнала:

```
=> SELECT '0/1FE81F8'::pg_lsn - '0/1FD7528'::pg_lsn;
```

```
?column?
-----
68816
(1 row)
```

Помимо изменений системного каталога, в журнал попадают записи:

- CREATE — создание файла отношения;
- INSERT+INIT — вставка строк в таблицу;
- COMMIT — фиксация транзакции.

```
=> SELECT resource_manager, record_length, xid, start_lsn, record_type
FROM pg_get_wal_records_info('0/1FD7528','0/1FE81F8')
WHERE record_type IN ('CREATE','INSERT+INIT','COMMIT')
ORDER BY start_lsn;
```

resource_manager	record_length	xid	start_lsn	record_type
Storage	42	0	0/1FD75A8	CREATE
Heap	59	745	0/1FD8590	INSERT+INIT
Heap	59	745	0/1FDBE28	INSERT+INIT
Heap	59	745	0/1FDF6D8	INSERT+INIT
Heap	59	745	0/1FE2F88	INSERT+INIT
Heap	59	745	0/1FE6838	INSERT+INIT
Transaction	421	745	0/1FE8050	COMMIT

(7 rows)

На уровне replica журнал содержит все изменения данных, что позволяет применить их к физической резервной копии или к реплике.

---

Установим для журнала уровень minimal (при этом придется также задать нулевое значение параметра max\_wal\_senders и перезапустить сервер).

```
=> ALTER SYSTEM SET wal_level = minimal;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET max_wal_senders = 0;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_tuning
```

Посмотрим, как теперь записывается в журнал команда CREATE TABLE AS SELECT.

```
=> DROP TABLE t_wal;
```

```
DROP TABLE
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/1FED178
(1 row)
```

```
=> CREATE TABLE t_wal(n) AS SELECT 1 FROM generate_series(1,1000);
```

```
SELECT 1000
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/200F640
(1 row)
```

Объем журнала уменьшился:

```
=> SELECT '0/200F640'::pg_lsn - '0/1FED178'::pg_lsn;
```

```
?column?
-----
140488
(1 row)
```

В журнале нет записей INSERT+INIT:

```
=> SELECT resource_manager, record_length, xid, start_lsn, record_type
FROM pg_get_wal_records_info('0/1FED178', '0/200F640')
WHERE record_type IN ('CREATE', 'INSERT+INIT', 'COMMIT')
ORDER BY start_lsn;
```

resource_manager	record_length	xid	start_lsn	record_type
Storage	42	0	0/1FEFD18	CREATE
Transaction	34	747	0/200F618	COMMIT

(2 rows)

На уровне minimal изменения, выполненные операторами CREATE TABLE AS SELECT, CREATE INDEX, TRUNCATE и некоторыми другими, не журналируются. Эти операторы всегда сами выполняют синхронизацию, обеспечивая долговечность. А журнал содержит только записи, необходимые для восстановления после сбоя.

---

Вернем уровень по умолчанию (replica).

```
=> ALTER SYSTEM RESET wal_level;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM RESET max_wal_senders;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_tuning
```



Кеширование

Повреждение данных

Неатомарность записи страниц

## Синхронизация с диском

данные должны дойти до энергонезависимого хранилища через многочисленные кешы

СУБД сообщает о синхронизации операционной системе способом, указанным в `wal_sync_method`

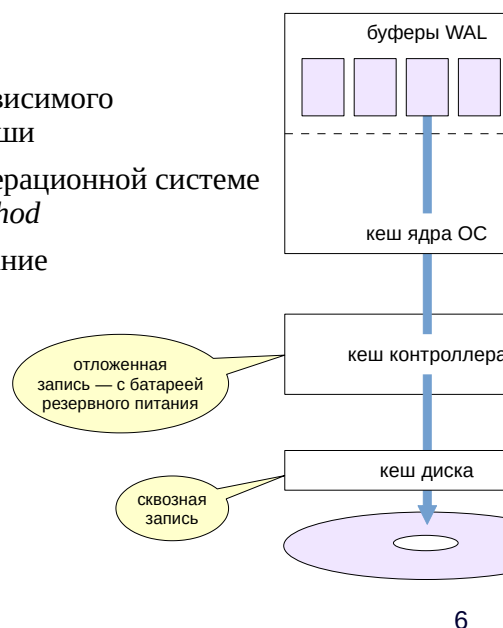
надо учитывать аппаратное кеширование

## Настройки

`fsync = on`

`wal_sync_method`

(утилита `pg_test_fsync`)



6

Есть несколько моментов, которые влияют на надежность. Во-первых, всевозможные кешы, находящиеся на пути к энергонезависимому хранилищу (такому как пластина жесткого диска).

Когда PostgreSQL требуется надежно записать данные, он пользуется способом, указанным в параметре `wal_sync_method`. Вариантов несколько, но основных — два: либо после записи операционной системе дается команда синхронизации (`fsync`, `fdatasync`), либо файл открывается или записывается со специальным флагом, который говорит о необходимости синхронизации, а по возможности — прямой записи, минуя кеш ОС. Утилита `pg_test_fsync` позволяет выбрать способ, наиболее подходящий для конкретной ОС и конкретной файловой системы. Но в любом случае это дорогостоящая операция.

Кроме этого, администратор должен убедиться в том, что данные действительно доходят до диска, а не задерживаются в кеше контроллера или самого диска. Если кеш контроллера откладывает запись, то контроллер в обязательном порядке должен иметь батарею резервного питания. Для дисков лучше устанавливать сквозную запись.

Вообще говоря, синхронизацию можно отменить (параметр `fsync`), но в этом случае про надежность хранения следует забыть. Единственный разумный вариант отключения этого параметра — временное увеличение производительности в случае, если данные можно легко восстановить (например, при начальной миграции).

<https://postgrespro.ru/docs/postgresql/16/wal-reliability>



## Контрольные суммы журнальных записей

включены всегда, CRC-32


## Контрольные суммы страниц

нужно включить при инициализации кластера

`initdb -k`

накладные  
расходы, увеличение  
размера журнала

## Настройки

 `data_checksums`

`ignore_checksum_failure = off`

Во-вторых, данные могут быть повреждены на носителе, при передаче данных по интерфейсным кабелям и т. п. Часть таких ошибок обрабатывается на аппаратном уровне, но часть — нет.

Чтобы вовремя обнаружить возникшую проблему, журнальные записи всегда снабжаются контрольными суммами.

Страницы данных также можно защитить контрольными суммами. Это лучше сделать сразу при инициализации кластера, но можно включить утилитой `pg_checksums` и потом, остановив сервер.

В производственной среде контрольные суммы должны быть включены обязательно, несмотря на накладные расходы на их вычисление и контроль. Иначе можно получить ситуацию, когда возникший сбой не будет вовремя обнаружен.

Проверить, включены ли контрольные суммы, можно с помощью параметра `data_checksums` (только для чтения). Параметр `ignore_checksum_failure` позволяет не прерывать транзакцию, прочитавшую сбойную страницу, но обычно его не следует включать.

## Образ страницы

при сбое страница может быть записана не полностью  
в журнал записывается образ страницы  
при первом ее изменении после контрольной точки  
при восстановлении журнальные записи  
применяются к записанному образу



## Настройки

`full_page_writes = on`  
`wal_compression = off`

увеличивает  
размер журнала

В-третьих, есть проблема атомарности записи.

Страница данных занимает 8 КБ (или больше: 16 КБ, 32 КБ), а на низком уровне запись происходит блоками, которые обычно имеют меньший размер (512 байт, 4 КБ, хотя бывают и другие размеры). Поэтому при сбое питания страница данных может записаться частично. Понятно, что при восстановлении бессмысленно применять к такой странице обычные журнальные записи.

Для защиты PostgreSQL позволяет записывать в журнал образ всей страницы при первом ее изменении после контрольной точки — этим управляет параметр `full_page_writes`. Отключать его имеет смысл, только если используемая файловая система и аппаратура сами по себе гарантируют атомарность записи.

Если при восстановлении мы встречаем в журнале образ страницы, то безусловно записываем его на диск (к нему больше доверия, так как он, как и всякая журнальная запись, защищен контрольной суммой). И далее к нему уже применяем обычные журнальные записи.

Хотя PostgreSQL исключает из полного образа страницы незанятое место, все же объем журнальных записей увеличивается. Если в кластере включены контрольные суммы страниц, при изменении битов-подсказок в журнале появляется дополнительная запись, отражающая изменение контрольной суммы.

Размер журнала можно уменьшить за счет сжатия полных образов, задав метод сжатия параметром `wal_compression`. Поддерживаются методы `pglz`, `lz4`, `zstd`. Значение `on` соответствует выбору `pglz`, `off` отключает сжатие.

## Контрольные суммы

Создадим еще одну таблицу:

```
=> CREATE TABLE t(id integer);
```

```
CREATE TABLE
```

```
=> INSERT INTO t VALUES (1),(2),(3);
```

```
INSERT 0 3
```

Вот файл, в котором находятся данные:

```
=> SELECT pg_relation_filepath('t');
```

```
pg_relation_filepath
-----
base/16390/16404
(1 row)
```

Остановим сервер и поменяем несколько байтов в странице (сотрем из заголовка LSN последней журнальной записи).

```
student$ sudo pg_ctlcluster 16 main stop
```

```
student$ sudo dd if=/dev/zero of=/var/lib/postgresql/16/main/base/16390/16404 oflag=dsync conv=notrunc bs=1 count=8
```

```
8+0 records in
8+0 records out
8 bytes copied, 0,0159579 s, 0,5 kB/s
```

Можно было бы и не останавливать сервер. Достаточно, чтобы:

- страница записалась на диск и была вытеснена из кеша;
- произошло повреждение;
- страница была прочитана с диска.

---

Теперь запускаем сервер.

```
student$ sudo pg_ctlcluster 16 main start
```

```
student$ psql wal_tuning
```

Попробуем прочитать таблицу:

```
=> SELECT * FROM t;
```

```
WARNING: page verification failed, calculated checksum 40449 but expected 45426
ERROR:  invalid page in block 0 of relation base/16390/16404
```

Параметр `ignore_checksum_failure` позволяет попытаться все-таки прочитать таблицу, хоть и с риском получить искаженные данные (например, если нет резервной копии):

```
=> SET ignore_checksum_failure = on;
```

```
SET
```

```
=> SELECT * FROM t;
```

```
WARNING: page verification failed, calculated checksum 40449 but expected 45426
 id
----
  1
  2
  3
(3 rows)
```

Характер нагрузки

Синхронная запись

Асинхронная запись

## Постоянный поток записи

последовательная запись, отсутствие случайного доступа  
характер нагрузки отличается от остальной системы  
при высокой нагрузке — размещение на отдельных физических дисках  
(символьная ссылка из PGDATA/pg\_wal)

## Редкое чтение

при восстановлении  
при работе процессов walsender, если реплика не успевает быстро  
получать записи

11

При обычной работе происходит постоянная и последовательная запись журнальных файлов. Поскольку отсутствует случайный доступ, с такой нагрузкой отлично справляются и обычные диски HDD.

Но такой характер нагрузки существенно отличается от того, как происходит доступ к файлам данных. Поэтому обычно выгодно размещать журнал на отдельном физическом диске (дисках), примонтированных к ФС сервера; вместо каталога PGDATA/pg\_wal нужно создать символическую ссылку на соответствующий каталог.

Однако есть ситуация, при которой журнальные файлы необходимо читать (кроме понятного случая восстановления после сбоя). Она возникает, если используется потоковая репликация и реплика не успевает получать журнальные записи, пока они еще находятся в буферах оперативной памяти основного сервера. Тогда процессу walsender приходится читать нужные данные с диска. Взаимодействие с репликой подробно рассматривается в курсе DBA3 в модуле «Репликация».

Для мониторинга и оценки производительности работы WAL в PostgreSQL версии 14 было добавлено представление pg\_stat\_wal.

## Алгоритм

при фиксации изменений сбрасываются все накопившиеся записи, включая запись о фиксации

ожидание *commit\_delay*, если активно не менее *commit\_siblings* транзакций

## Свойства

гарантируется долговечность

увеличивается время отклика

## Настройки

*synchronous\_commit* = on

*commit\_delay* = 0

*commit\_siblings* = 5

включать  
при большом потоке  
коротких транзакций

12

Запись журнала происходит в одном из двух режимов: синхронном (когда при фиксации транзакции продолжение работы невозможно до тех пор, пока все журнальные записи о транзакции не окажутся на диске) и асинхронном (когда журнал записывается частями в фоне).

Синхронный режим включается параметром *synchronous\_commit*.

Поскольку синхронизация работает долго, выгодно выполнять ее как можно реже. Для этого процесс, записывающий журнал, делает паузу, определяемую параметром *commit\_delay*, но только в том случае, если имеется не менее *commit\_siblings* активных транзакций — в расчете на то, что за время ожидания часть транзакций успеет завершиться и можно будет синхронизировать их записи за один раз.

Затем процесс выполняет сброс журнала на диск до необходимого LSN (или несколько больше, если за время ожидания добавились новые записи).

При синхронной записи гарантируется долговечность — если транзакция зафиксирована, то все ее журнальные записи уже есть на диске и не будут потеряны. Обратная сторона состоит в том, что синхронная запись увеличивает время отклика (команда COMMIT не возвращает управление до окончания синхронизации) и снижает производительность системы.

## Алгоритм

циклы записи через `wal_writer_delay`  
записываются только целиком заполненные страницы;  
но если новых полных страниц нет, то записывается последняя до конца

## Свойства

гарантируется согласованность, но не долговечность  
зафиксированные изменения могут пропасть ( $3 \times wal\_writer\_delay$ )

## Настройки

`synchronous_commit`  
`wal_writer_delay = 200ms`  
`wal_writer_flush_after = 1MB`

можно  
изменять на уровне  
транзакции

13

При асинхронной записи работает процесс `wal writer`, сбрасывая накопившиеся журнальные записи либо через `wal_writer_delay` единиц времени, либо по достижении объема `wal_writer_flush_after`:

- Если с прошлого раза в буферах была целиком заполнена одна или несколько страниц, сбрасываются только такие, полностью заполненные, страницы (или часть таких страниц; вспомним, что журнальный кеш представляет собой кольцевой буфер — записывается только непрерывная последовательность страниц). При большом потоке изменений это позволяет не синхронизировать одну и ту же страницу несколько раз.
- Если же заполненные страницы не появились, записывается текущая (не до конца заполненная) страница журнала.

Асинхронная запись эффективнее синхронной — фиксация изменений не ждет записи. Однако надежность уменьшается: зафиксированные данные могут пропасть в случае сбоя, если между фиксацией и сбоем прошло менее  $3 \times wal\_writer\_delay$  единиц времени.

Параметр `synchronous_commit` можно устанавливать в рамках транзакций. Это позволяет увеличивать производительность, жертвуя надежностью только части транзакций.

<https://postgrespro.ru/docs/postgresql/16/wal-async-commit>

В реальности оба режима работают совместно. Журнальные записи долгой транзакции будут записываться асинхронно (чтобы освободить буферы WAL). А если при сбросе грязного буфера окажется, что соответствующая журнальная запись еще не на диске, она тут же будет сброшена в синхронном режиме.

## Влияние синхронной фиксации на производительность

Режим, включенный по умолчанию, — синхронная фиксация.

```
=> SHOW synchronous_commit;
```

```
synchronous_commit
-----
on
(1 row)
```

Запустим простой тест производительности с помощью утилиты pgbench. Для этого сначала инициализируем необходимые таблицы...

```
student$ pgbench -i wal_tuning
```

```
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.10 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.40 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.16 s,
vacuum 0.10 s, primary keys 0.13 s).
```

...а также сбросим статистику о работе журнала предзаписи:

```
=> SELECT pg_stat_reset_shared('wal');
```

```
pg_stat_reset_shared
-----
(1 row)
```

Запускаем тест на 10 секунд.

```
student$ pgbench -P 1 -T 10 wal_tuning
```

```
pgbench (16.3 (Ubuntu 16.3-1.pgdg22.04+1))
starting vacuum...end.
progress: 1.0 s, 216.0 tps, lat 4.611 ms stddev 7.600, 0 failed
progress: 2.0 s, 368.8 tps, lat 2.705 ms stddev 1.472, 0 failed
progress: 3.0 s, 391.2 tps, lat 2.559 ms stddev 1.061, 0 failed
progress: 4.0 s, 449.0 tps, lat 2.228 ms stddev 0.531, 0 failed
progress: 5.0 s, 441.0 tps, lat 2.267 ms stddev 0.831, 0 failed
progress: 6.0 s, 398.0 tps, lat 2.512 ms stddev 1.590, 0 failed
progress: 7.0 s, 505.0 tps, lat 1.976 ms stddev 0.277, 0 failed
progress: 8.0 s, 435.9 tps, lat 2.292 ms stddev 0.818, 0 failed
progress: 9.0 s, 291.9 tps, lat 3.419 ms stddev 1.893, 0 failed
progress: 10.0 s, 225.1 tps, lat 4.454 ms stddev 1.815, 0 failed
transaction type: <built-in: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 10 s
number of transactions actually processed: 3723
number of failed transactions: 0 (0.000%)
latency average = 2.685 ms
latency stddev = 2.285 ms
initial connection time = 2.878 ms
tps = 372.323978 (without initial connection time)
```

В результатах pgbench нас интересует число транзакций или скорость (tps), а в данных представления pg\_stat\_wal — количество операций записи и синхронизации журнала:

```
=> SELECT wal_records, wal_bytes, wal_write, wal_sync FROM pg_stat_wal;
```



wal_records	wal_bytes	wal_write	wal_sync
23890	1705950	3725	3725

(1 row)

Теперь установим асинхронный режим.

```
=> ALTER SYSTEM SET synchronous_commit = off;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

pg_reload_conf
t

(1 row)

Сбросим накопленные данные.

```
=> SELECT pg_stat_reset_shared('wal');
```

pg_stat_reset_shared

(1 row)

И снова запускаем тест.

```
student$ pgbench -P 1 -T 10 wal_tuning
```

```
pgbench (16.3 (Ubuntu 16.3-1.pgdg22.04+1))
starting vacuum...end.
progress: 1.0 s, 1228.8 tps, lat 0.805 ms stddev 2.294, 0 failed
progress: 2.0 s, 236.0 tps, lat 4.233 ms stddev 9.715, 0 failed
progress: 3.0 s, 289.4 tps, lat 3.449 ms stddev 8.252, 0 failed
progress: 4.0 s, 728.5 tps, lat 1.373 ms stddev 1.128, 0 failed
progress: 5.0 s, 751.0 tps, lat 1.330 ms stddev 1.337, 0 failed
progress: 6.0 s, 1265.1 tps, lat 0.790 ms stddev 0.743, 0 failed
progress: 7.0 s, 1379.1 tps, lat 0.724 ms stddev 0.784, 0 failed
progress: 8.0 s, 1432.0 tps, lat 0.698 ms stddev 0.499, 0 failed
progress: 9.0 s, 1568.0 tps, lat 0.637 ms stddev 0.635, 0 failed
progress: 10.0 s, 2197.0 tps, lat 0.455 ms stddev 0.144, 0 failed
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 10 s
number of transactions actually processed: 11075
number of failed transactions: 0 (0.000%)
latency average = 0.901 ms
latency stddev = 2.304 ms
initial connection time = 9.425 ms
tps = 1108.479452 (without initial connection time)
```

```
=> SELECT wal_records, wal_bytes, wal_write, wal_sync FROM pg_stat_wal;
```

wal_records	wal_bytes	wal_write	wal_sync
66772	4390958	587	52

(1 row)

Кроме повышения tps, мы видим и саму причину повышения производительности: количество операций записи и синхронизации журнала значительно уменьшились.

Разумеется, на реальной системе соотношение может быть другим, но видно, что в асинхронном режиме производительность существенно выше.

Журнал позволяет восстановить согласованность после сбоя, но может использоваться и для других задач

Надежность записи требует настройки не только СУБД, но и на уровне ОС, ФС и аппаратуры

Настройка позволяет достичь баланса между долговечностью и производительностью

1. Изучите, как влияет на размер журнальных записей значение параметра *full\_page\_writes*.  
Для этого повторите простой тест `pgbench`, показанный в демонстрации, с разными настройками журнала. Перед запуском каждого теста выполняйте контрольную точку.  
Объясните полученный результат.
2. Во сколько раз уменьшается размер журнальных записей при включении параметра *wal\_compression*?

Для детального анализа может пригодиться функция `pg_get_wal_stats` из расширения `pg_walinspect`.

## 1a. Full page writes = on

```
=> CREATE DATABASE wal_tuning;
```

```
CREATE DATABASE
```

```
=> \c wal_tuning
```

You are now connected to database "wal\_tuning" as user "student".

```
=> CREATE EXTENSION pg_walinspect;
```

```
CREATE EXTENSION
```

```
student$ pgbench -i wal_tuning
```

```
dropping old tables...
```

```
NOTICE: table "pgbench_accounts" does not exist, skipping
```

```
NOTICE: table "pgbench_branches" does not exist, skipping
```

```
NOTICE: table "pgbench_history" does not exist, skipping
```

```
NOTICE: table "pgbench_tellers" does not exist, skipping
```

```
creating tables...
```

```
generating data (client-side)...
```

```
100000 of 100000 tuples (100%) done (elapsed 0.35 s, remaining 0.00 s)
```

```
vacuuming...
```

```
creating primary keys...
```

```
done in 0.59 s (drop tables 0.00 s, create tables 0.02 s, client-side generate 0.42 s,
```

```
vacuum 0.07 s, primary keys 0.09 s).
```

```
=> SHOW full_page_writes;
```

```
full_page_writes
```

```
-----
```

```
on
```

```
(1 row)
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
```

```
-----
```

```
0/2C477C0
```

```
(1 row)
```

Запускаем тест на 10 секунд.

```
student$ pgbench -T 10 wal_tuning
```

```
pgbench (16.3 (Ubuntu 16.3-1.pgdg22.04+1))
```

```
starting vacuum...end.
```

```
transaction type: <builtin: TPC-B (sort of)>
```

```
scaling factor: 1
```

```
query mode: simple
```

```
number of clients: 1
```

```
number of threads: 1
```

```
maximum number of tries: 1
```

```
duration: 10 s
```

```
number of transactions actually processed: 4486
```

```
number of failed transactions: 0 (0.000%)
```

```
latency average = 2.229 ms
```

```
initial connection time = 3.880 ms
```

```
tps = 448.698085 (without initial connection time)
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
```

```
-----
```

```
0/3BF81D8
```

```
(1 row)
```

Смотрим статистику журнальных записей:

```
=> SELECT sum(count) count,
pg_size_pretty(sum(record_size)) record_size,
pg_size_pretty(sum(fpi_size)) fpi_size,
pg_size_pretty(sum(combined_size)) combined_size
FROM pg_get_wal_stats('0/2C477C0', '0/3BF81D8');

count | record_size | fpi_size | combined_size
-----+-----+-----+-----
28546 | 1965 kB     | 14 MB    | 16 MB
(1 row)
```

Значительную часть объема журнала составляют полные образы страниц (FPI).

## 1b. Full page writes = off

```
=> ALTER SYSTEM SET full_page_writes = off;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf
-----
t
(1 row)
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/3C04058
(1 row)
```

Запускаем тест на 10 секунд.

```
student$ pgbench -T 10 wal_tuning
```

```
pgbench (16.3 (Ubuntu 16.3-1.pgdg22.04+1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 10 s
number of transactions actually processed: 5212
number of failed transactions: 0 (0.000%)
latency average = 1.918 ms
initial connection time = 2.759 ms
tps = 521.257599 (without initial connection time)
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/3E243F8
(1 row)
```

Смотрим статистику журнальных записей:

```
=> SELECT sum(count) count,
pg_size_pretty(sum(record_size)) record_size,
pg_size_pretty(sum(fpi_size)) fpi_size,
pg_size_pretty(sum(combined_size)) combined_size
FROM pg_get_wal_stats('0/3C04058', '0/3E243F8');

count | record_size | fpi_size | combined_size
-----+-----+-----+-----
32965 | 2088 kB     | 0 bytes  | 2088 kB
(1 row)
```

Объем заметно сократился, полные образы страниц в журнал теперь не записываются.

## 2. Сжатие

```
=> ALTER SYSTEM SET full_page_writes = on;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET wal_compression = on;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf
-----
t
(1 row)
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/3E24578
(1 row)
```

Запускаем тест на 10 секунд.

```
student$ pgbench -T 10 wal_tuning
```

```
pgbench (16.3 (Ubuntu 16.3-1.pgdg22.04+1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 10 s
number of transactions actually processed: 5271
number of failed transactions: 0 (0.000%)
latency average = 1.897 ms
initial connection time = 2.533 ms
tps = 527.135213 (without initial connection time)
```

```
=> SELECT pg_current_wal_insert_lsn();
```

```
pg_current_wal_insert_lsn
-----
0/41E09D8
(1 row)
```

Смотрим статистику журнальных записей:

```
=> SELECT sum(count) count,
       pg_size_pretty(sum(record_size)) record_size,
       pg_size_pretty(sum(fpi_size)) fpi_size,
       pg_size_pretty(sum(combined_size)) combined_size
FROM pg_get_wal_stats('0/3E24578', '0/41E09D8');
```

```
count | record_size | fpi_size | combined_size
-----+-----+-----+-----
33359 | 2112 kB     | 1620 kB  | 3732 kB
(1 row)
```

В данном случае — при наличии большого числа полных образов страниц — размер журнальных записей уменьшился в несколько раз. Хотя сжатие и нагружает процессор, практически наверняка им стоит воспользоваться при включенных полных образах страниц.