Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

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

Темы

Уровни журнала и решаемые задачи

Надежность записи

Производительность

Уровни журнала

minimal, replica, logical

Уровни журнала

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

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

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/D3B73310
(1 row)
```

```
=> SELECT pg_walfile_name('0/D3B73310');
```

```
     pg_walfile_name
--------------------------
 00000001000000000000D3
(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/D3B98090
(1 row)
```

Объем журнала:

```
=> SELECT '0/D3B98090'::pg_lsn - '0/D3B73310'::pg_lsn;
```

```
 ?column?
----------
   150912
(1 row)
```

Помимо изменений системного каталога, в журнал попадают записи:

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

```
postgres$ /usr/lib/postgresql/13/bin/pg_waldump -p /var/lib/postgresql/13/main/pg_wal -s 0/D3B73310 -e 0/D3B98090 00000001000000000000D3 | grep 'CREATE\|INSERT+INIT\|COMMIT'
```

```
rmgr: Storage     len (rec/tot):     42/    42, tx:          0, lsn: 0/D3B73310, prev 0/D3B72AC8, desc: CREATE base/76940/76941
rmgr: Heap        len (rec/tot):     59/    59, tx:     696126, lsn: 0/D3B88428, prev 0/D3B883F8, desc: INSERT+INIT off 1 flags 0x00, blkref #0: rel 1663/76940/76941 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:     696126, lsn: 0/D3B8BCC0, prev 0/D3B8BC80, desc: INSERT+INIT off 1 flags 0x00, blkref #0: rel 1663/76940/76941 blk 1
rmgr: Heap        len (rec/tot):     59/    59, tx:     696126, lsn: 0/D3B8F570, prev 0/D3B8F530, desc: INSERT+INIT off 1 flags 0x00, blkref #0: rel 1663/76940/76941 blk 2
rmgr: Heap        len (rec/tot):     59/    59, tx:     696126, lsn: 0/D3B92E20, prev 0/D3B92DE0, desc: INSERT+INIT off 1 flags 0x00, blkref #0: rel 1663/76940/76941 blk 3
rmgr: Heap        len (rec/tot):     59/    59, tx:     696126, lsn: 0/D3B966D0, prev 0/D3B96690, desc: INSERT+INIT off 1 flags 0x00, blkref #0: rel 1663/76940/76941 blk 4
rmgr: Transaction len (rec/tot):    421/   421, tx:     696126, lsn: 0/D3B97ED0, prev 0/D3B97E90, desc: COMMIT 2022-12-26 19:09:06.901916 MSK; inval msgs: catcache 75 catcache 74 catca
```

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 13 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/D3B9E5D0
(1 row)
```

```
=> SELECT pg_walfile_name('0/D3B9E5D0');
```

```
     pg_walfile_name
--------------------------
 00000001000000000000D3
(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/D3BB6980
(1 row)
```

Объем журнала уменьшился:

```
=> SELECT '0/D3BB6980'::pg_lsn - '0/D3B9E5D0'::pg_lsn;
```

```
 ?column?
----------
    99248
(1 row)
```

В журнале нет записей INSERT+INIT:

```
postgres$ /usr/lib/postgresql/13/bin/pg_waldump -p /var/lib/postgresql/13/main/pg_wal -s 0/D3B9E5D0 -e 0/D3BB6980 000000010000000000000D3 | grep 'CREATE\|INSERT+INIT\|COMMIT'

rmgr: Storage     len (rec/tot):     42/    42, tx:          0, lsn: 0/D3B9FCF0, prev 0/D3B9FCD0, desc: CREATE base/76940/76944
rmgr: Transaction len (rec/tot):     34/    34, tx:     696128, lsn: 0/D3BB6958, prev 0/D3BAC878, desc: COMMIT 2022-12-26 19:09:09.877108 MSK
```

На уровне minimal операторы CREATE TABLE AS SELECT, TRUNCATE и некоторые другие выполняют синхронизацию, обеспечивая тем самым 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 13 main restart

student$ psql wal_tuning
```

Надежность записи

Кеширование

Повреждение данных

Неатомарность записи страниц

Кеширование

Синхронизация с диском

данные должны дойти до энергонезависимого хранилища через многочисленные кеши

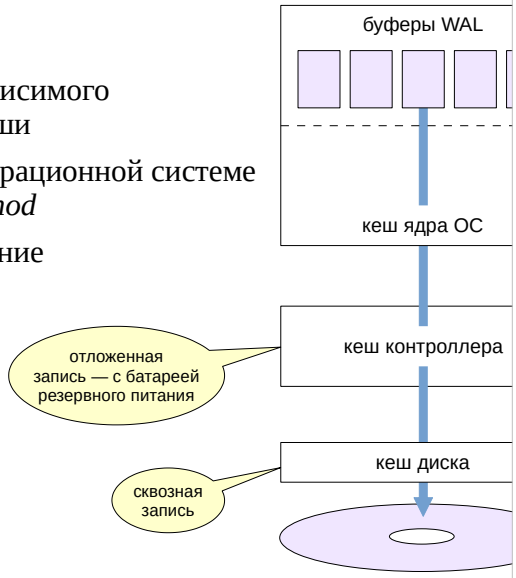
СУБД сообщает о синхронизации операционной системе способом, указанным в *wal_sync_method*

надо учитывать аппаратное кеширование

Настройки

fsync = on

wal_sync_method
(утилита pg_test_fsync)



Есть несколько моментов, которые влияют на надежность. Во-первых, всевозможные кеши, 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/13/wal-reliability

Повреждение данных

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

wal_log_hints = off (неявно on при контрольных суммах страниц)

wal_compression = 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 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.

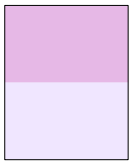
Проверить, включены ли контрольные суммы, можно с помощью параметра *data_checksums* (только для чтения). Параметр *ignore_checksum_failure* позволяет не прерывать транзакцию, прочитавшую сбойную страницу, но обычно его не следует включать.

При включенных контрольных суммах в журнал всегда попадает такая «несущественная» информация, как биты подсказок (рассмотрены в модуле «Многоверсионность»), поскольку изменение любого бита приводит и к изменению контрольной суммы. При выключенных контрольных суммах за запись битов подсказок в журнал отвечает параметр *wal_log_hints*.

Изменения битов подсказок всегда журналируется в виде полного образа страницы (FPI, full page image), что сильно увеличивает размер журнала. Для экономии можно включить сжатие полных образов с помощью параметра *wal_compression*.

Неатомарность записи

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 (такой же образ записывается и при изменении битов подсказок) — 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, мы безусловно записываем его на диск (к нему больше доверия, так как он, как и всякая журнальная запись, защищен контрольной суммой). И уже к нему применяем обычные журнальные записи.

Хотя PostgreSQL исключает из полного образа страницы незанятое место, все же объем журнальных записей увеличивается. Как уже говорилось, размер можно уменьшить за счет сжатия полных образов (параметр *wal_compression*).

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/76940/76947
(1 row)
```

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

```
student$ sudo pg_ctlcluster 13 main stop
```

```
postgres$ dd if=/dev/zero of=/var/lib/postgresql/13/main/base/76940/76947 oflag=dsync conv=notrunc bs=1 count=8
```

```
8+0 records in
8+0 records out
8 bytes copied, 0,00559028 s, 1,4 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 13 main start
```

```
student$ psql wal_tuning
```

Попробуем прочитать таблицу:

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

```
WARNING:  page verification failed, calculated checksum 29967 but expected 5965
ERROR:  invalid page in block 0 of relation base/76940/76947
```

Параметр ignore_checksum_failure позволяет попробовать прочитать таблицу 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 29967 but expected 5965
 id
----
  1
  2
  3
(3 rows)
```

Производительность

Характер нагрузки

Синхронная запись

Асинхронная запись

Характер нагрузки

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

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 символьную 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».

Синхронная запись

Алгоритм

при фиксации изменений сбрасывает накопившиеся записи, включая запись о фиксации

ждет *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

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 — расчет на то, что за время ожидания часть транзакций успеют 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.

Асинхронная запись

Алгоритм

циклы записи через *wal_writer_delay*

записывает только целиком заполненные страницы;
но если новых полных страниц нет, то записывает последнюю до конца

Свойства

гарантируется согласованность, но не долговечность

зафиксированные изменения могут пропасть (3 × *wal_writer_delay*)

Настройки

synchronous_commit

можно изменять на уровне транзакции

wal_writer_delay = 200ms

wal_writer_flush_after = 1MB

При асинхронной записи работает процесс wal writer, сбрасывая накопившиеся журнальные записи либо через *wal_writer_delay* единиц времени, либо по достижению 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 × *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/postgrespro/13/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 0.26 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.12 s, vacuum 0.07 s, primary keys 0.06 s).
```

Запускаем тест на 10 секунд.

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

```
starting vacuum...end.
progress: 1.0 s, 732.9 tps, lat 1.359 ms stddev 0.209
progress: 2.0 s, 779.0 tps, lat 1.282 ms stddev 0.206
progress: 3.0 s, 755.1 tps, lat 1.324 ms stddev 0.253
progress: 4.0 s, 708.0 tps, lat 1.413 ms stddev 0.568
progress: 5.0 s, 805.0 tps, lat 1.242 ms stddev 0.502
progress: 6.0 s, 942.8 tps, lat 1.059 ms stddev 0.190
progress: 7.0 s, 872.1 tps, lat 1.146 ms stddev 0.235
progress: 8.0 s, 770.0 tps, lat 1.299 ms stddev 0.461
progress: 9.0 s, 802.0 tps, lat 1.246 ms stddev 0.188
progress: 10.0 s, 783.1 tps, lat 1.277 ms stddev 0.181
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 10 s
number of transactions actually processed: 7951
latency average = 1.257 ms
latency stddev = 0.341 ms
tps = 794.980077 (including connections establishing)
tps = 795.195652 (excluding connections establishing)
```

В результатах нас интересует число транзакций или скорость (tps).

Теперь установим асинхронный режим.

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

```
ALTER SYSTEM
```

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

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

Снова запускаем тест.

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

```
starting vacuum...end.
progress: 1.0 s, 2117.8 tps, lat 0.470 ms stddev 0.199
progress: 2.0 s, 2225.1 tps, lat 0.449 ms stddev 0.157
progress: 3.0 s, 2194.0 tps, lat 0.455 ms stddev 0.165
progress: 4.0 s, 2212.8 tps, lat 0.451 ms stddev 0.187
progress: 5.0 s, 2187.1 tps, lat 0.457 ms stddev 0.279
progress: 6.0 s, 2211.0 tps, lat 0.452 ms stddev 0.166
progress: 7.0 s, 2247.9 tps, lat 0.445 ms stddev 0.147
progress: 8.0 s, 2227.7 tps, lat 0.449 ms stddev 0.162
progress: 9.0 s, 2234.5 tps, lat 0.447 ms stddev 0.169
```

```
progress: 10.0 s, 2227.0 tps, lat 0.449 ms stddev 0.157
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 10 s
number of transactions actually processed: 22101
latency average = 0.452 ms
latency stddev = 0.182 ms
tps = 2208.715404 (including connections establishing)
tps = 2209.409605 (excluding connections establishing)
```

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

Итоги

Журнал позволяет восстановить согласованность после сбоев, 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

Практика

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*?

1. Обратите внимание на значение параметра *data_checksums*.

Для детального анализа может пригодиться утилита pg_waldump с ключом --stats, показывающим статистическую информацию о составе журнальных записей.

1a. Full page writes = on

```
=> CREATE DATABASE wal_tuning;

CREATE DATABASE
```

```
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 0.26 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.12 s, vacuum 0.07 s, primary keys 0.06 s).
```

```
=> SHOW full_page_writes;

 full_page_writes
------------------
 on
(1 row)


=> CHECKPOINT;

CHECKPOINT
```

Запускаем тест на 30 секунд.

```
=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/3C7EE80
(1 row)


student$ pgbench -T 30 wal_tuning

starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 26016
latency average = 1.153 ms
tps = 867.171145 (including connections establishing)
tps = 867.270182 (excluding connections establishing)

=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/55E0260
(1 row)
```

Размер журнальных записей:

```
=> SELECT pg_size_pretty('1/55E0260'::pg_lsn - '1/3C7EE80'::pg_lsn);

 pg_size_pretty
----------------
 25 MB
(1 row)
```

Основной объем журнала составляют полные образы страниц (FPI):

```
=> SELECT pg_walfile_name('1/3C7EE80'), pg_walfile_name('1/55E0260');

     pg_walfile_name      |     pg_walfile_name
--------------------------+--------------------------
 000000010000000100000003 | 000000010000000100000005
(1 row)
```

```
postgres$ /usr/lib/postgresql/13/bin/pg_waldump --stats=record -p /var/lib/postgresql/13/main/pg_wal -s 1/3C7EE80 -e 1/55E0260 000000010000000100000003 000000010000000100000005

Type                                           N      (%)          Record size      (%)             FPI size      (%)        Combined size      (%)
----                                           -      ---          -----------      ---             --------      ---        -------------      ---
XLOG/FPI_FOR_HINT                             17 (  0,01)                833 (  0,01)              127592 (  0,83)              128425 (  0,49)
Transaction/COMMIT                         26021 ( 16,08)             885322 (  8,26)                   0 (  0,00)              885322 (  3,39)
Storage/CREATE                                 1 (  0,00)                 42 (  0,00)                   0 (  0,00)                  42 (  0,00)
CLOG/ZEROPAGE                                  1 (  0,00)                 30 (  0,00)                   0 (  0,00)                  30 (  0,00)
Standby/LOCK                                   1 (  0,00)                 42 (  0,00)                   0 (  0,00)                  42 (  0,00)
Standby/RUNNING_XACTS                          2 (  0,00)                108 (  0,00)                   0 (  0,00)                 108 (  0,00)
Standby/INVALIDATIONS                          2 (  0,00)                180 (  0,00)                   0 (  0,00)                 180 (  0,00)
Heap2/CLEAN                                26306 ( 16,26)            1633606 ( 15,25)                   0 (  0,00)             1633606 (  6,25)
Heap2/CLEANUP_INFO                             1 (  0,00)                 42 (  0,00)                   0 (  0,00)                  42 (  0,00)
Heap2/VISIBLE                               1706 (  1,05)             100664 (  0,94)               16384 (  0,11)              117048 (  0,45)
Heap/INSERT                                25856 ( 15,98)            2046662 ( 19,10)                   0 (  0,00)             2046662 (  7,83)
Heap/UPDATE                                 1674 (  1,03)             286705 (  2,68)                2928 (  0,02)              289633 (  1,11)
Heap/HOT_UPDATE                            76358 ( 47,18)            5518221 ( 51,50)                 604 (  0,00)             5518825 ( 21,12)
Heap/LOCK                                   1700 (  1,05)              99995 (  0,93)            13236564 ( 85,87)            13336559 ( 51,04)
Heap/INPLACE                                   6 (  0,00)               1292 (  0,01)                   0 (  0,00)                1292 (  0,00)
Heap/INSERT+INIT                             166 (  0,10)              13114 (  0,12)                   0 (  0,00)               13114 (  0,05)
Heap/UPDATE+INIT                              28 (  0,02)               5231 (  0,05)                   0 (  0,00)                5231 (  0,02)
Btree/INSERT_LEAF                           1710 (  1,06)             106382 (  0,99)               2030828 ( 13,17)             2137210 (  8,18)
Btree/VACUUM                                 274 (  0,17)              17032 (  0,16)                   0 (  0,00)               17032 (  0,07)
                                          --------                  --------                        --------                    --------
Total                                       161830                 10715503 [41,01%]              15414900 [58,99%]            26130403 [100%]
```

Обратите внимание на общий размер (строка Total) полных образов (столбец FPI size).

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
```

Запускаем тест на 30 секунд.

```
=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/55E0330
(1 row)
```

```
student$ pgbench -T 30 wal_tuning

starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 28467
latency average = 1.054 ms
tps = 948.882266 (including connections establishing)
tps = 948.979210 (excluding connections establishing)

=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/6E275F0
(1 row)
```

Размер журнальных записей:

```
=> SELECT pg_size_pretty('1/6E275F0'::pg_lsn - '1/55E0330'::pg_lsn);

 pg_size_pretty
----------------
 24 MB
(1 row)
```

Размер уменьшился, но не так существенно, как можно было бы ожидать.

Причина в том, что кластер инициализирован с контрольными суммами в страницах данных:

```
=> SHOW data_checksums;

 data_checksums
----------------
 on
(1 row)
```

При этом, несмотря на то что full_page_writes выключен, в журнал все равно записываются полные образы страниц при изменении битов подсказок. Эти данные и составляют большую часть всего объема:

```
=> SELECT pg_walfile_name('1/55E0330'), pg_walfile_name('1/6E275F0');

     pg_walfile_name      |     pg_walfile_name
--------------------------+--------------------------
 000000010000000100000005 | 000000010000000100000006
(1 row)
```

```
postgres$ /usr/lib/postgresql/13/bin/pg_waldump --stats=record -p /var/lib/postgresql/13/main/pg_wal -s '1/55E0330' -e '1/6E275F0' 000000010000000100000005 000000010000000100000006

Type                                           N      (%)          Record size      (%)             FPI size      (%)         Combined size      (%)
----                                           -      ---          -----------      ---             --------      ---         -------------      ---
XLOG/FPI_FOR_HINT                           1679 (  0,97)               82271 (  0,73)             13631708 (100,00)              13713979 ( 54,97)
Transaction/COMMIT                         28468 ( 16,44)              967992 (  8,55)                    0 (  0,00)                967992 (  3,88)
Storage/CREATE                                 1 (  0,00)                  42 (  0,00)                    0 (  0,00)                    42 (  0,00)
CLOG/ZEROPAGE                                  1 (  0,00)                  30 (  0,00)                    0 (  0,00)                    30 (  0,00)
Standby/LOCK                                   1 (  0,00)                  42 (  0,00)                    0 (  0,00)                    42 (  0,00)
Standby/RUNNING_XACTS                          2 (  0,00)                 108 (  0,00)                    0 (  0,00)                   108 (  0,00)
Standby/INVALIDATIONS                          2 (  0,00)                 180 (  0,00)                    0 (  0,00)                   180 (  0,00)
Heap2/CLEAN                                28785 ( 16,62)             1807070 ( 15,97)                    0 (  0,00)               1807070 (  7,24)
Heap2/VISIBLE                                  2 (  0,00)                 118 (  0,00)                    0 (  0,00)                   118 (  0,00)
Heap/INSERT                                28285 ( 16,33)             2234515 ( 19,75)                    0 (  0,00)               2234515 (  8,96)
Heap/UPDATE                                  170 (  0,10)                29023 (  0,26)                    0 (  0,00)                 29023 (  0,12)
Heap/HOT_UPDATE                            85229 ( 49,22)             6157734 ( 54,42)                    0 (  0,00)               6157734 ( 24,68)
Heap/LOCK                                    172 (  0,10)                 9288 (  0,08)                    0 (  0,00)                  9288 (  0,04)
Heap/INPLACE                                   2 (  0,00)                  458 (  0,00)                    0 (  0,00)                   458 (  0,00)
Heap/INSERT+INIT                             182 (  0,11)                14378 (  0,13)                    0 (  0,00)                14378 (  0,06)
Heap/UPDATE+INIT                               3 (  0,00)                  513 (  0,00)                    0 (  0,00)                   513 (  0,00)
Btree/INSERT_LEAF                            175 (  0,10)                11216 (  0,10)                    0 (  0,00)                11216 (  0,04)
                                          --------                    --------                        --------                    --------
Total                                      173159                    11314978 [45,36%]             13631708 [54,64%]            24946686 [100%]
```

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
```

Запускаем тест на 30 секунд.

```
=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/6E276C0
(1 row)
```

```
student$ pgbench -T 30 wal_tuning

starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
```

```
number of threads: 1
duration: 30 s
number of transactions actually processed: 25249
latency average = 1.188 ms
tps = 841.608753 (including connections establishing)
tps = 841.690066 (excluding connections establishing)
```

```
=> SELECT pg_current_wal_insert_lsn();

 pg_current_wal_insert_lsn
---------------------------
 1/7B14EF8
(1 row)
```

Размер журнальных записей:

```
=> SELECT pg_size_pretty('1/7B14EF8'::pg_lsn - '1/6E276C0'::pg_lsn);

 pg_size_pretty
----------------
 13 MB
(1 row)
```

В данном случае — при наличии большого числа полных образов страниц — размер журнальных записей уменьшился примерно в три раза. Хотя включение сжатия и нагружает процессор, практически наверняка им стоит воспользоваться при включенных контрольных суммах.