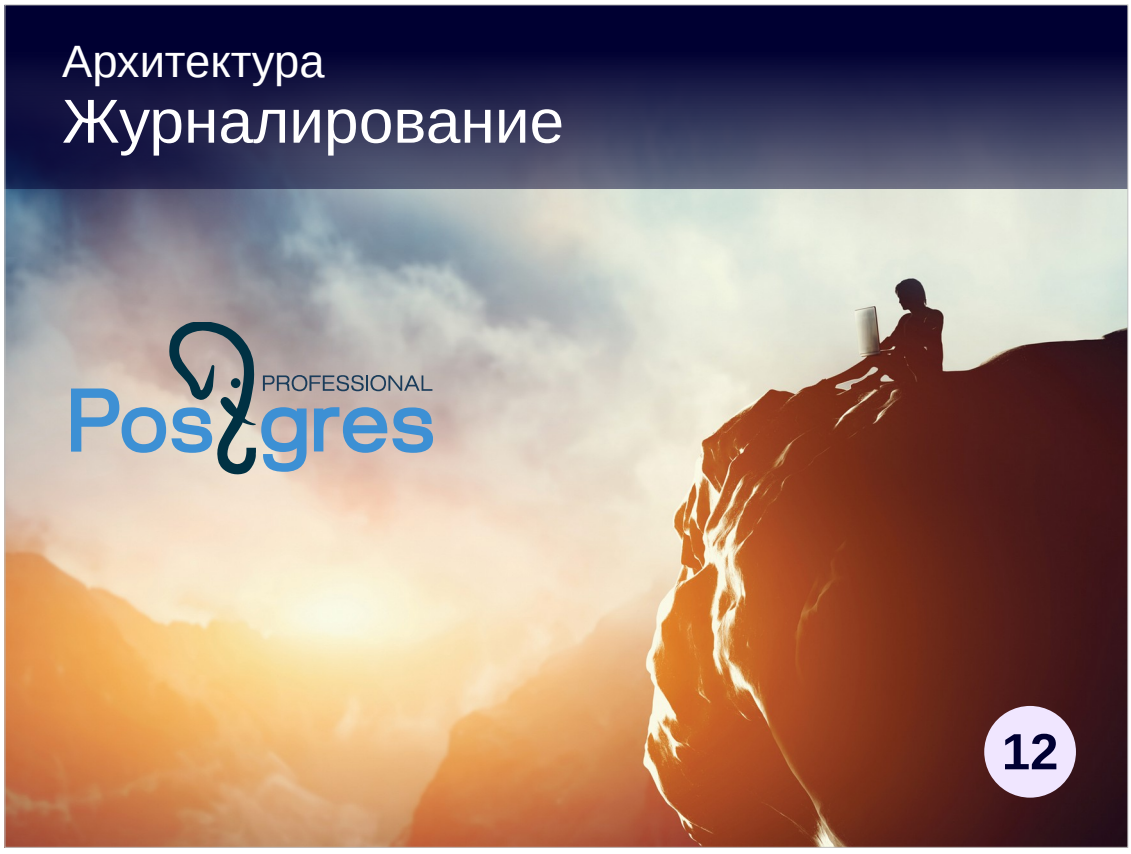


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Буферный кеш

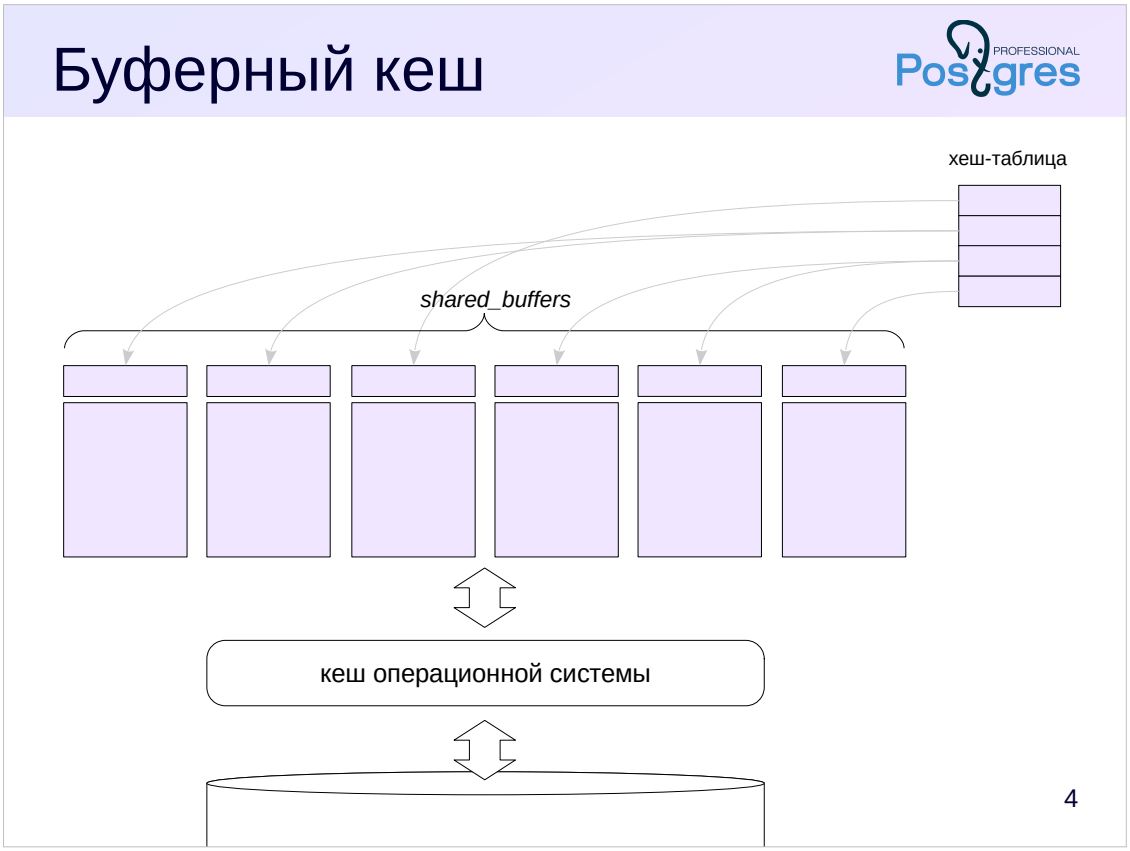
Журнал предзаписи

Буферный кеш

Устройство и использование

Фоновая запись

Локальный кеш для временных таблиц



Задача буферного кеша — сглаживать разницу в производительности двух типов памяти: оперативной (быстрая, но мало) и дисковой (медленная, но много). Чтобы работать с данными — читать или изменять, — процессы читают страницы в буферный кеш и, пока страница находится в кеше, мы экономим на обращениях к диску.

Буферный кеш располагается в общей памяти сервера и представляет собой массив буферов (его размер задается конфигурационным параметром *shared_buffers*). Каждый буфер состоит из места под одну страницу данных и заголовка. Заголовок содержит расположение страницы на диске (файл и номер страницы) и другую информацию.

Чтобы нужную страницу можно было быстро найти в кеше, используется хеш-таблица. Ключом хеширования в ней служит файл и номер страницы внутри файла. Чтобы найти страницу, надо проверить все буферы, указанные в соответствующей корзине хеш-таблицы: либо страница найдется, либо мы убедимся в том, что ее нет в кеше.

В последнем случае страницу придется прочитать в буферный кеш с диска. Однако напомним, что PostgreSQL читает и записывает данные не в обход кеша операционной системы, а через него. Это, очевидно, приводит к дублированию данных и двойному кешированию, но такова текущая реализация. Поэтому «промах» мимо буферного кеша не обязательно означает, что данные будут физически читаться с диска.

Буферный кеш

```
=> CREATE DATABASE arch_wal;
```

```
CREATE DATABASE
```

```
=> \c arch_wal
```

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

Создадим таблицу.

```
=> CREATE TABLE t(
    id integer,
    s char(100) DEFAULT ''
) WITH (autovacuum_enabled = off);
```

```
CREATE TABLE
```

```
=> INSERT INTO t(id) SELECT g.id FROM generate_series(1,1000) g(id);
```

```
INSERT 0 1000
```

В состав PostgreSQL входит расширение, которое позволяет посмотреть, что происходит в буферном кеше.

```
=> CREATE EXTENSION pg_buffercache;
```

```
CREATE EXTENSION
```

Размер кеша определяется конфигурационным параметром:

```
=> SHOW shared_buffers;
```

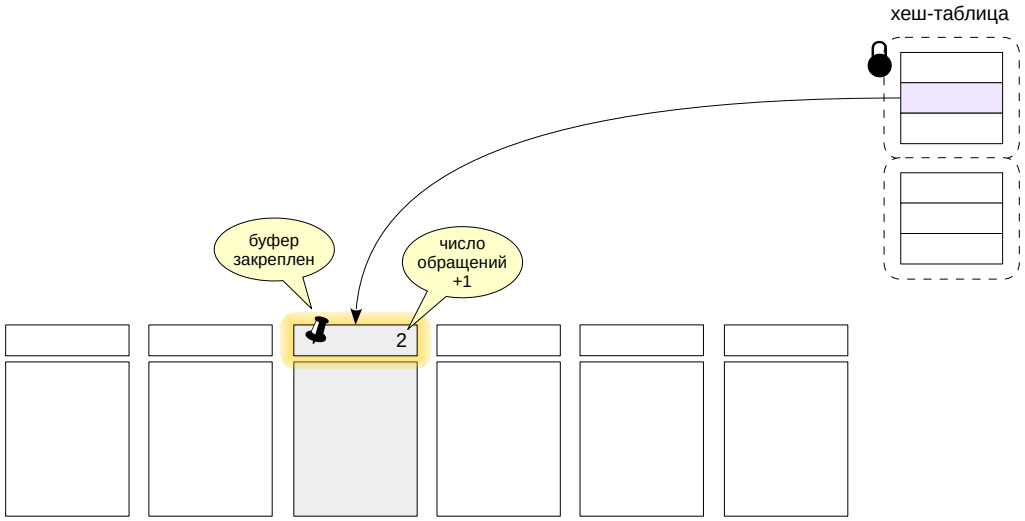
```
 shared_buffers
----------------
 128MB
(1 row)
```

Расширение показывает каждый буфер как отдельную строку одноименного представления:

```
=> SELECT count(*), pg_size_pretty(count(*) * 8192) FROM pg_buffercache;
```

```
 count | pg_size_pretty
-------+----------------
 16384 | 128 MB
(1 row)
```

Страница в кеше



Рассмотрим случай, когда необходимая страница находится в кеше.

Буферный кеш и хеш-таблица находятся в общей памяти, и доступ к ним есть у всех процессов сервера. Поэтому одновременный, конкурентный доступ к этим структурам необходимо упорядочивать.

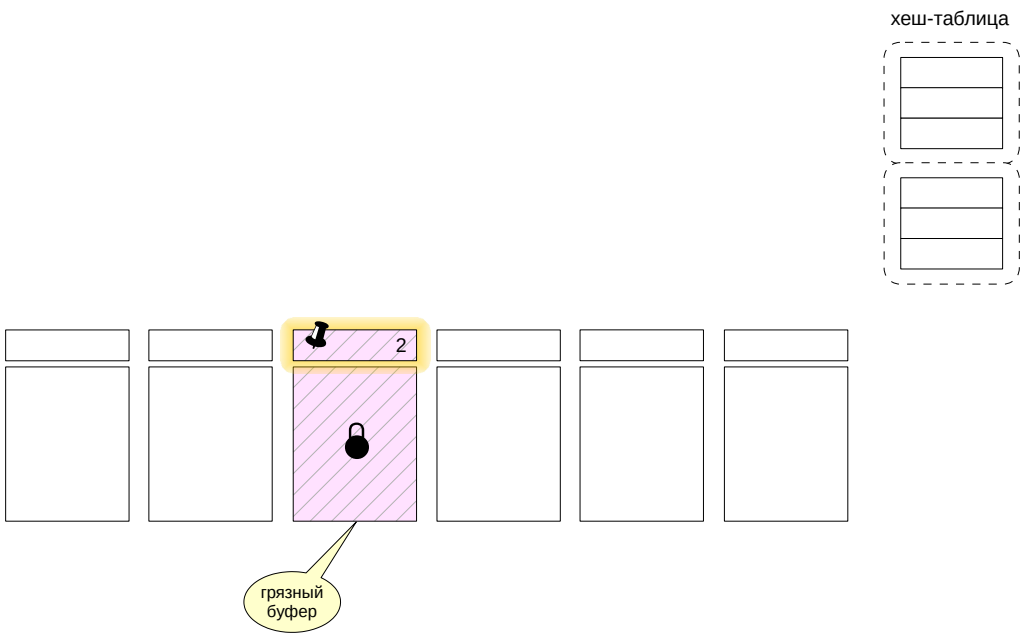
Вычислив хеш-код, процесс блокирует необходимый фрагмент хеш-таблицы в разделяемом режиме (только для чтения) с помощью так называемой «легкой блокировки», и уже затем находит буфер, содержащий нужную страницу.

Сервер управляет блокировками данных в разделяемой памяти полностью автоматически. Разработчики PostgreSQL прилагают много усилий к тому, чтобы эти блокировки работали эффективно. Например, чтобы не блокировать всю хеш-таблицу сразу, она разделена на несколько (128) фрагментов, каждый из которых может работать независимо от других. Тем не менее надо понимать, что, хотя доступ к кешу быстрее, чем к диску, он все же не бесплатен.

Дальше процесс «закрепляет» буфер, увеличивая счетчик pin count в заголовке страницы. Пока буфер закреплен (значение pin count больше нуля), считается, что буфер используется и его содержимое не должно «радикально» измениться. Например, в странице может появиться новая версия строки — обычно это не мешает благодаря многоверсионности и правилам видимости. Но в закрепленный буфер не может быть прочитана другая страница.

Процесс также увеличивает счетчик обращений к странице (usage count). Счетчик используется для того, чтобы понимать, какие страницы используются часто, а какие нет.

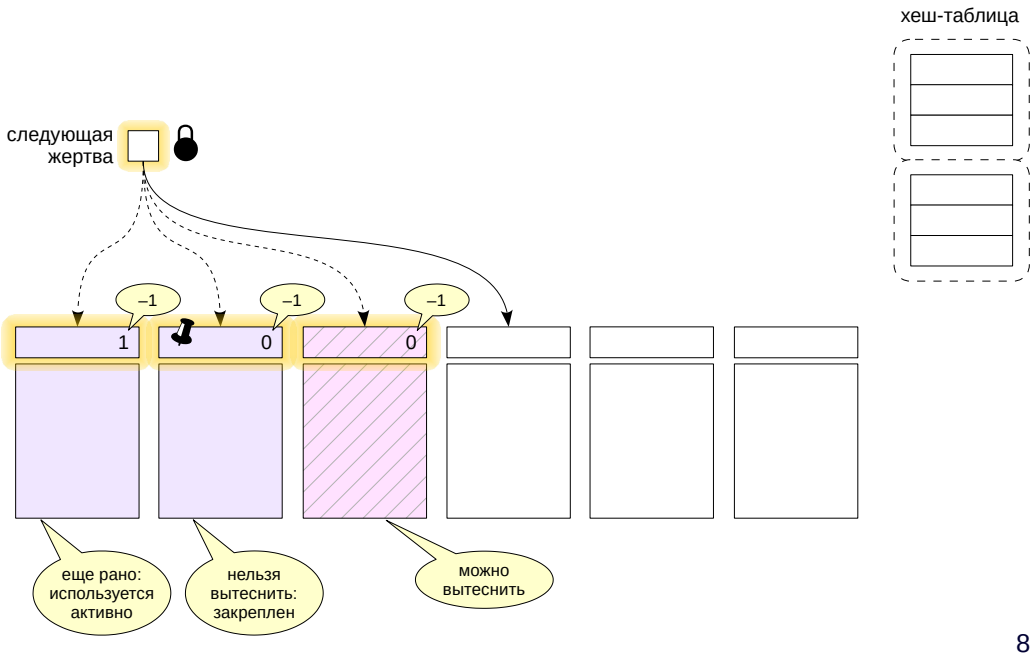
Изменение в странице



Если процессу необходимо изменить содержимое страницы в буфере, он должен получить монопольную блокировку для этой страницы (чтобы не менять данные в тот момент, когда их читают другие процессы).

Измененная страница, не записанная еще на диск, называется «грязной». В заголовке буфера делается отметка об этом. (На рисунках грязные буферы выделены цветом и штриховкой.)

Чтение с вытеснением



8

Если нужной страницы не нашлось в кеше, ее необходимо прочитать с диска, а для этого надо освободить какой-то из буферов, *вытеснив* находящуюся там другую страницу. Идея состоит в том, чтобы вытеснять страницы, к которым обращались редко, с тем, чтобы активно используемые данные как можно дольше находились в кеше.

Механизм вытеснения использует указатель на следующую «жертву». Этот указатель пробегает по кругу все буферы, уменьшая на единицу их счетчики обращений (usage count). Выбирается первый же буфер, который одновременно:

- имеет значение счетчика обращений, равное 0;

- не закреплен.

По-английски этот алгоритм называется clock-sweep.

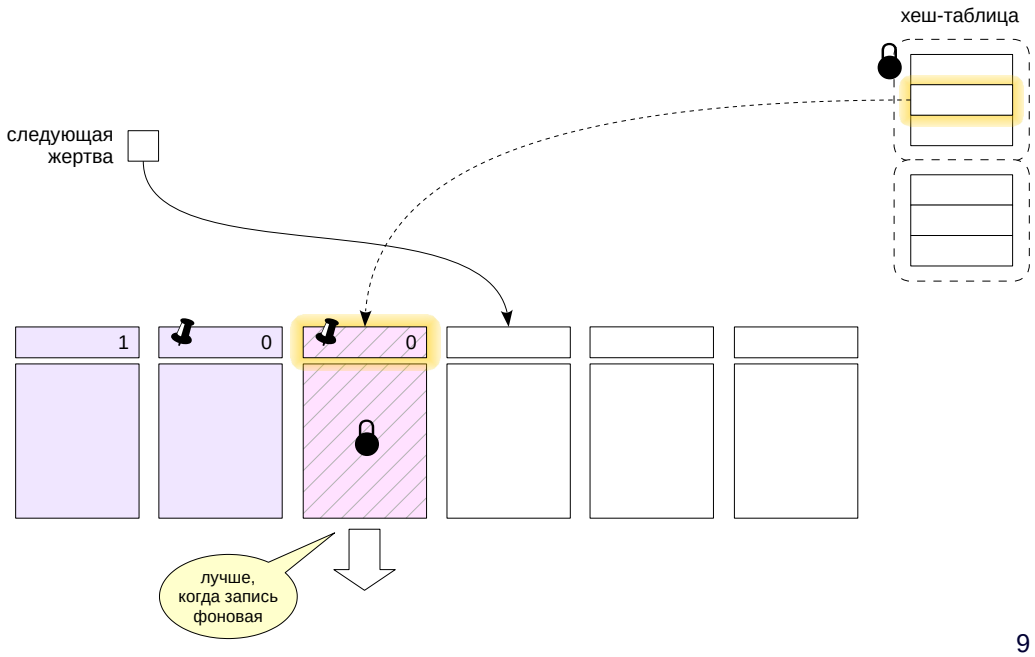
Чем больше значение счетчика у буфера (то есть чем чаще он используется), тем больше у него шансов задержаться в кеше. Чтобы избежать «наматывания кругов» при вытеснении, максимальное значение счетчика обращений ограничено числом 5,

В нашем примере процесс обращается к первому буферу по указателю. Счетчик обращений этого буфера еще не равен 0, так что он уменьшается на единицу и указатель сдвигается к следующему буферу.

Следующий буфер закреплен (то есть используется каким-то процессом), и поэтому страница не может быть вытеснена из него. Значение его счетчика также уменьшается на единицу.

Наконец, мы приходим к незакрепленному буферу с нулевым счетчиком обращений. Страница из этого буфера и будет вытеснена.

Чтение с вытеснением



9

Однако в нашем примере найденный буфер оказался грязным — он содержит измененные данные. Поэтому сначала страницу требуется сохранить на диск. Для этого буфер закрепляется и устанавливается блокировка содержимого страницы, после чего страница записывается на диск.

Значительно эффективнее, когда запись происходит асинхронно и процесс обнаруживает вытесняемую страницу чистой. Чтобы это было возможным, существует процесс фоновой записи (background writer).

Процесс фоновой записи использует тот же самый алгоритм поиска буферов для вытеснения, что и обслуживающие процессы, только использует свой указатель. Он может опережать указатель на «жертву», но никогда не отстает от него.

Записываются буферы, которые одновременно:

- содержат измененные (грязные) страницы,

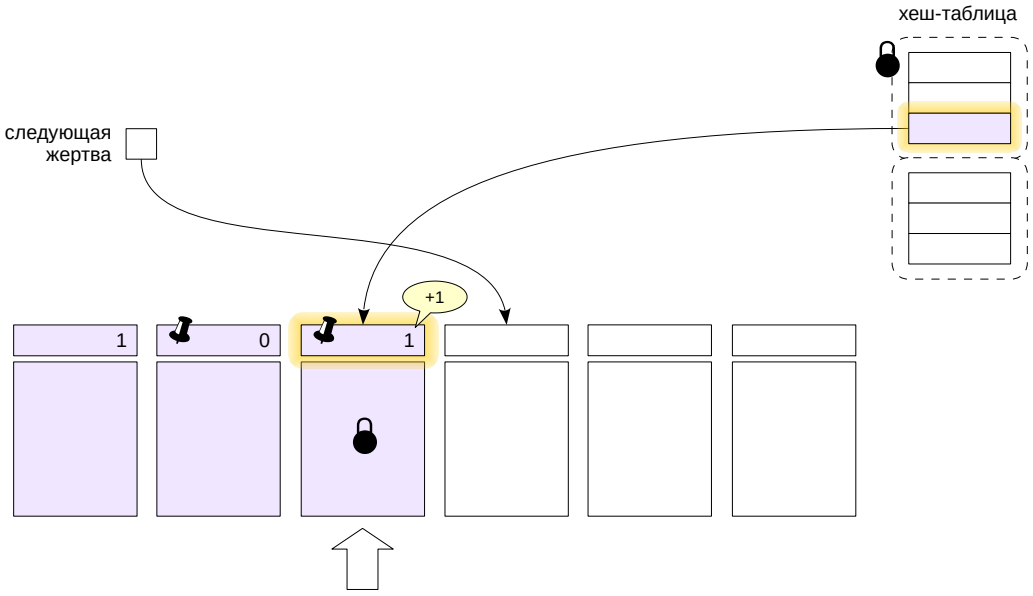
- не закреплены (pin count = 0),

- имеют нулевое число обращений (usage count = 0).

Таким образом фоновый процесс записи находит те буферы, которые с большой вероятностью вскоре потребуется вытеснить.

Теперь страница может быть выброшена, а из хеш-таблицы необходимо удалить ссылку на эту страницу. Для этого потребуется монопольно заблокировать нужный фрагмент хеш-таблицы.

Чтение с вытеснением



После того как страница из грязного буфера записана на диск, в освободившийся буфер читается новая страница, а в хеш-таблицу добавляется ссылка на нее.

Счетчик обращений увеличивается на единицу. Ссылка на следующую «жертву» уже указывает на следующий буфер, а у только что загруженного есть время нарастить счетчик обращений, пока указатель не обойдет по кругу весь буферный кеш и не вернется вновь.

Использование буферного кеша

Перезапустим сервер, чтобы очистить буферный кеш.

```
=> \q
```

```
student$ sudo pg_ctlcluster 12 main restart
```

```
student$ psql arch_wal
```

С помощью расширения можно убедиться, что в кеше нет буферов, относящихся к нашей таблице:

```
=> SELECT isdirty, usagecount, count(*)
FROM pg_buffercache b
WHERE b.relfilenode = pg_relation_filenode('t'::regclass)
GROUP BY isdirty, usagecount;
```

```
 isdirty | usagecount | count
---------+------------+-------
(0 rows)
```

Теперь выполним запрос, который прочитает все страницы таблицы. Команда EXPLAIN ANALYZE с параметром BUFFERS покажет, какое количество обращений к буферам потребовалось для выполнения.

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
SELECT * FROM t;
```

```
                QUERY PLAN
-----------------------------------------
 Seq Scan on t (actual rows=1000 loops=1)
   Buffers: shared read=18 dirtied=18
 Planning Time: 0.115 ms
 Execution Time: 0.164 ms
(4 rows)
```

В данном случае запрос прочитал 18 страниц, ни одной из которых не оказалось в кеше (read).

А почему эти страницы были изменены (dirtied)?

Как рассматривалось в обзоре внутреннего устройства многоверсионности, при первом чтении новых версий строк в заголовках этих версий проставляется статус транзакции (зафиксирована или оборвана). Это вызывает изменение страницы.

Что теперь мы обнаружим в кеше?

```
=> SELECT isdirty, usagecount, count(*)
FROM pg_buffercache b
WHERE b.relfilenode = pg_relation_filenode('t'::regclass)
GROUP BY isdirty, usagecount;
```

```
 isdirty | usagecount | count
---------+------------+-------
 t       |          1 |    18
(1 row)
```

Как раз 18 буферов заполнены данными таблицы.

А как изменится выполнение того же самого запроса?

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
SELECT * FROM t;
```

```
                QUERY PLAN
-----------------------------------------
 Seq Scan on t (actual rows=1000 loops=1)
   Buffers: shared hit=18
 Planning Time: 0.032 ms
 Execution Time: 0.071 ms
(4 rows)
```

Теперь все данные были взяты из кеша (hit), а сам запрос выполнился быстрее. Именно с наличием кеша и связан тот факт, что при повторном выполнении запросы обычно работают быстрее.

Временные таблицы

Данные временных таблиц

- видны только одному сеансу — нет смысла использовать общий кеш
- существуют в пределах сеанса — не жалко потерять при сбое

Используется локальный буферный кеш

- не требуются блокировки
- память выделяется по необходимости в пределах *temp_buffers*
- обычный алгоритм вытеснения

Исключение из общего правила представляют собой временные таблицы. Поскольку временные данные видны только одному процессу, им нечего делать в общем буферном кеше. Более того, временные данные существуют только в рамках одного сеанса, так что их не нужно защищать от сбоя.

Для временных данных используется облегченный локальный кеш.

Поскольку локальный кеш доступен только одному процессу, для него не требуются блокировки. Память выделяется по мере необходимости (в пределах, заданных параметром *temp_buffers*), ведь временные таблицы используются далеко не во всех сеансах. В локальном кеше используется обычный алгоритм вытеснения.

Журнал предзаписи

Устройство журнала

Контрольная точка

Восстановление после сбоя

Синхронный и асинхронный режимы

Журнал предзаписи

Необходимость

данные меняются в оперативной памяти (в частности, в буферном кеше) и попадают на диск асинхронно

при сбое остается только информация, записанная на диск

Механизм

действия над данными записываются в журнал:
изменение страниц в буферном кеше, фиксация и отмена транзакций

журнальная запись попадает на диск раньше измененных данных

после сбоя операции, записанные в журнал, но не попавшие на диск, можно выполнить повторно

Основная причина существования журнала — необходимость восстановления согласованности данных в случае сбоя, при котором теряется содержимое оперативной памяти, в частности, буферный кеш. Тем самым обеспечивается выполнение свойства долговечности (буква «D» из набора свойств транзакций ACID).

Одновременно с изменением данных создается запись в журнале, содержащая достаточную информацию для повторения этой операции. Журнальная запись в обязательном порядке попадает на диск (или другое энергонезависимое устройство) до того, как туда попадет измененная страница — отсюда и название: «журнал предзаписи», «write-ahead log».

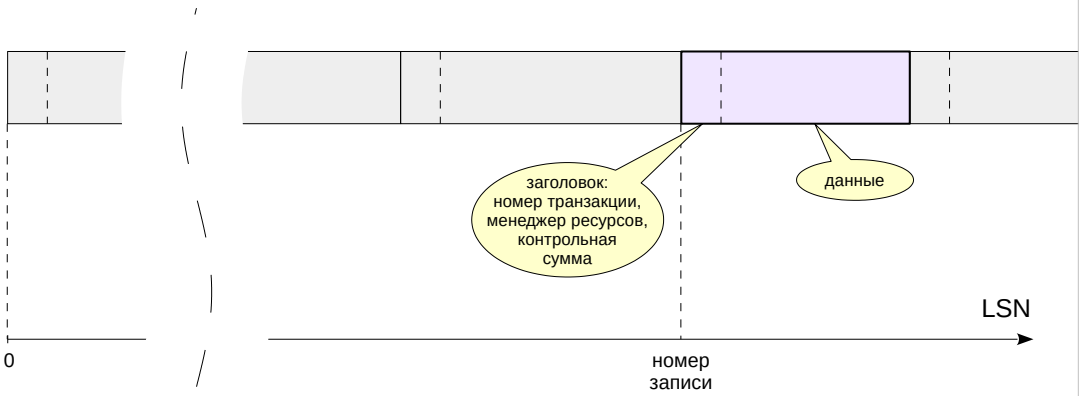
Журналировать нужно все операции, при выполнении которых возможна ситуация, что при сбое результат операции не дойдет до диска. В частности, в журнал записываются изменение страниц в буферном кеше и факт фиксации и отмены транзакций.

В журнал *не* записываются операции с нежурналируемыми (и временными) таблицами.

В случае сбоя можно прочитать журнал и при необходимости повторить те операции, которые уже были выполнены, но результат которых не успел попасть на диск.

https://postgrespro.ru/docs/postgresql/12/wal-intro

Устройство журнала



последовательность записей
номер записи — 64-битный LSN (log sequence number)
специальный тип pg_lsn

Логически журнал можно представить себе как последовательность записей различной длины. Каждая запись содержит данные о некоторой операции, предваренные заголовком. В заголовке, в числе прочего, указаны:

- Номер транзакции, к которой относится запись.

- Менеджер ресурсов — компонент системы, ответственный за данную запись, который понимает, как интерпретировать данные. Есть отдельные менеджеры для таблиц, для каждого типа индекса, для статуса транзакций и т. п.

- Контрольная сумма (CRC).

Для того, чтобы сослаться на определенную запись, используется тип данных pg_lsn (LSN = log sequence number) — 64-битное число, представляющее собой байтовое смещение до записи относительно начала журнала.

На диске журнал хранится в виде файлов фиксированного размера (обычно 16 Мбайт). Когда заканчивается место в текущем файле, система переключается на следующий.

В оперативной памяти работа с журналом ведется в специальных буферах, которые организованы наподобие буферного кеша.

Устройство журнала

Начнем транзакцию.

```
=> BEGIN;
```

```
BEGIN
```

Текущая позиция в журнале:

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

```
 pg_current_wal_insert_lsn
---------------------------
 0/420E3D8
(1 row)
```

LSN выводится как два 32-битных числа в шестнадцатеричной системе через косую черту.

Выполним какое-нибудь действие, например, изменим строку в таблице:

```
=> UPDATE t SET s = 'FOO' WHERE id = 1;
```

```
UPDATE 1
```

Изменится ли позиция в журнале?

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

```
 pg_current_wal_insert_lsn
---------------------------
 0/4210C98
(1 row)
```

Позиция увеличилась.

Завершим теперь транзакцию.

```
=> COMMIT;
```

```
COMMIT
```

Позиция в журнале снова изменилась:

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

```
 pg_current_wal_insert_lsn
---------------------------
 0/4210CC0
(1 row)
```

Размер журнальных записей (в байтах), соответствующих нашей транзакции, можно узнать простым вычитанием одной позиции из другой:

```
=> SELECT '0/4210CC0'::pg_lsn - '0/420E3D8'::pg_lsn;
```

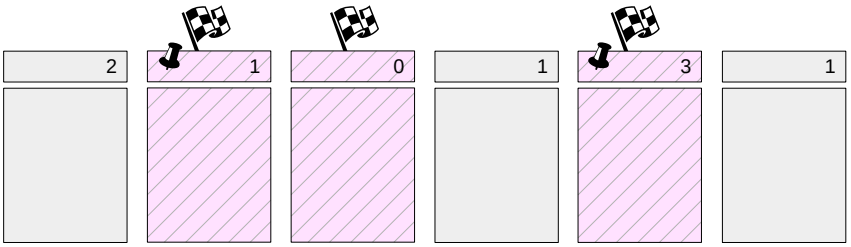
```
 ?column?
----------
    10472
(1 row)
```

Безусловно, в журнал попадает информация обо всех действиях во всем кластере, но в данном случае мы считаем, что в нашей системе больше ничего не происходит.

Контрольная точка

Процесс периодического сброса грязных страниц на диск

Ограничивает количество журнальных записей, необходимых для восстановления в случае сбоя



начало контрольной точки: помечаются грязные страницы в буферном кеше

Если не предпринять специальных мер, то активно использующаяся страница, попав в буферный кеш, может никогда не вытесняться. Это означает, что при восстановлении после сбоя придется просматривать *все* журнальные записи, созданные с момента запуска сервера.

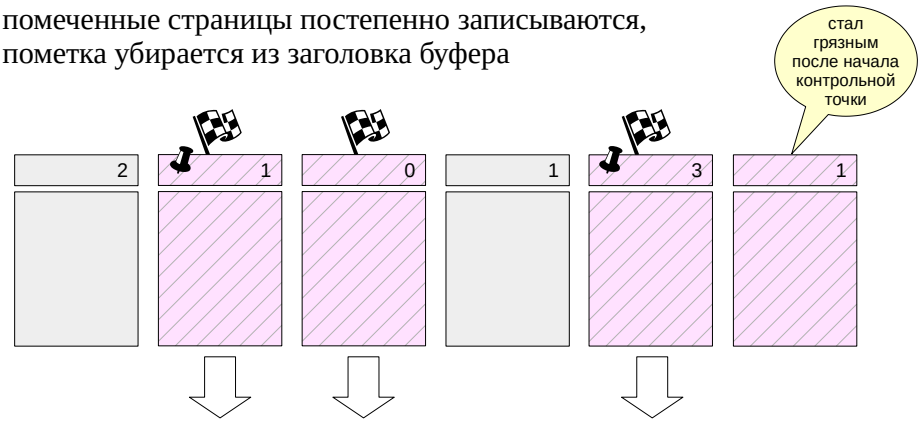
На практике это, конечно, недопустимо. Во-первых, файлы занимают много места — их все придется хранить на сервере. Во-вторых, время восстановления будет запредельно большим.

Поэтому существует специальный фоновый процесс контрольной точки (checkpointer), который периодически сбрасывает все грязные страницы на диск (но не вытесняет их из кеша). После того, как контрольная точка завершена, журналы вплоть до начала контрольной точки больше не нужны для восстановления.

Грязных страниц может оказаться много, и сбрасывать их одномоментно на диск — плохая идея. Поэтому в начале выполнения контрольной точки в заголовках грязных буферов ставится специальный флаг, который говорит о том, что в ходе выполнения контрольной точки данная страница должна быть сброшена.

Контрольная точка

помеченные страницы постепенно записываются,
пометка убирается из заголовка буфера



После завершения контрольной точки журнальные записи, предшествующие ее началу, могут быть удалены

Затем процесс контрольной точки *постепенно* проходит по всем буферам и сбрасывает помеченные страницы на диск. Отметим, что страницы не вытесняются из кеша, а только записываются. Поэтому контрольная точка не обращает внимания на число обращений к буферу и признак закрепленности.

Конечно, в процессе работы процесса контрольной точки страницы продолжают изменяться в буферном кеше. Но новые грязные буферы уже не рассматриваются процессом контрольной точки, так как на момент начала работы они не были грязными и не отмечены флагом.

В конце работы процесс создает журнальную запись об окончании контрольной точки и сохраняет LSN начала контрольной точки, чтобы при необходимости можно было быстро выяснить последнюю завершенную контрольную точку.

После этого сервер имеет право удалить все журнальные записи, предшествующие началу только что завершенной контрольной точки.

Контрольная точка

Сейчас в буферном кеше находится несколько грязных буферов:

```
=> SELECT count(*)
FROM pg_buffercache b
WHERE isdirty;
```

```
 count
-------
    19
(1 row)
```

Контрольная точка выполняется сервером периодически, но ее можно вызвать и вручную.

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

Поскольку по умолчанию буферный кеш имеет небольшой размер, контрольная точка выполняется очень быстро. Но в реальных системах, использующих большой буферный кеш, это может занимать существенное время.

Сколько грязных буферов останется в буферном кеше?

```
=> SELECT count(*)
FROM pg_buffercache b
WHERE isdirty;
```

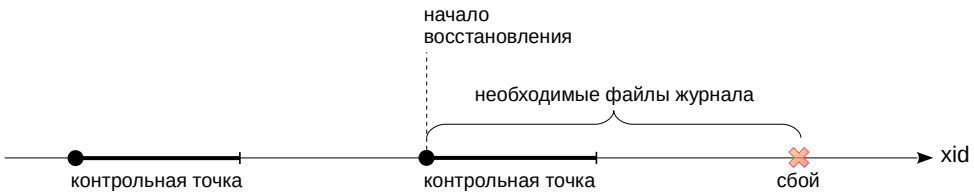
```
 count
-------
     0
(1 row)
```

Ни одного. Хотя, если бы в нашей системе была какая-то активность, грязные буферы — появившиеся уже после начала контрольной точки — могли бы остаться.

Восстановление

При старте сервера после сбоя

1. найти LSN_0 начала последней завершенной контрольной точки
2. применить каждую запись журнала, начиная с LSN_0, если LSN записи больше, чем LSN страницы
3. оборвать начатые, но не зафиксированные транзакции
4. очистить нежурналируемые таблицы
5. выполнить контрольную точку



Если в работе сервера произошел сбой, то при последующем запуске процесс startup обнаруживает это и выполняет автоматическое восстановление.

Сначала процесс startup определяет LSN_0 начала последней завершенной контрольной точки. Далее процесс читает журнальные записи, начиная с найденной позиции, последовательно применяя записи к страницам, если в этом есть необходимость.

Чтобы понять, надо ли применять журнальную запись к странице, надо сравнить LSN последнего изменения страницы (который при любых изменениях записывается в заголовок страницы) с LSN журнальной записи. Если LSN страницы оказывается меньше, то операцию необходимо повторить.

Таким образом восстанавливается согласованность данных во всех страницах. Все транзакции, которые были начаты, но не зафиксированы, помечаются как оборванные.

В конце процесса все нежурналируемые таблицы очищаются, поскольку их содержимое восстановить невозможно.

На этом процесс startup завершает работу, а процесс checkpointer выполняет контрольную точку, чтобы зафиксировать восстановленное состояние.

Режимы журналирования

Синхронный

- сброс журнальных записей на диск при каждой фиксации
- гарантируется долговечность
- увеличивается время отклика

Асинхронный

- сброс журнальных записей происходит периодически
- гарантируется согласованность, но не долговечность
- недавно зафиксированные изменения могут пропасть

Настройка

synchronous_commit = on / off — можно изменять на уровне транзакции

Запись журнала происходит в одном из двух режимов.

В **синхронном** режиме при фиксации транзакции продолжение работы невозможно до тех пор, пока все журнальные записи, относящиеся к этой транзакции, не окажутся на диске (энергонезависимом носителе).

При синхронной записи гарантируется долговечность — если транзакция зафиксирована, то все ее журнальные записи уже 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 поэтому уменьшает производительность системы.

В **асинхронном** режиме журнал записывается частями в фоновом режиме.

Асинхронная запись эффективнее синхронной. Во-первых, фиксация изменений не должна ничего ждать. Во-вторых, при каждой записи на диск обрабатываются все накопившиеся журнальные записи и, таким образом, уменьшается число избыточных обращений к диску.

Однако надежность уменьшается: зафиксированные данные могут пропасть в случае сбоя, если между фиксацией и сбоем прошло не очень много времени.

Параметр *synchronous_commit*, управляющий режимом записи журнала, можно устанавливать не только глобально, но и в рамках отдельных транзакций. Это позволяет увеличивать производительность, жертвуя надежностью только части транзакций.

https://postgrespro.ru/docs/postgrespro/12/wal-async-commit

Итоги

Буферный кеш ускоряет доступ к часто используемым данным (но не решает всех проблем производительности)

Использование буферов в оперативной памяти приводит к необходимости журналирования

Процесс контрольной точки ограничивает размер хранимых журнальных файлов и сокращает время восстановления

Журнал не только позволяет восстановить согласованность после сбоев, но может использоваться и для других задач

Практика

1. Воспользуйтесь расширением pg_prewarm, чтобы автоматически восстанавливать содержимое буферного кеша после перезагрузки сервера.
   Проверьте и затем отключите расширение.
2. Создайте таблицу и вставьте в нее столько строк, чтобы ее объем составлял примерно половину от объема буферного кеша. Перезагрузите сервер, чтобы очистить кеш.
   Выполните запрос на чтение всех строк таблицы и после этого проверьте, сколько буферов в кеше содержат страницы этой таблицы. Объясните результат.
3. Сравните производительность системы при синхронном и асинхронном режимах фиксации, используя утилиту эталонного тестирования pgbench.

1. Для этого потребуется добавить расширение pg_prewarm в загружаемые библиотеки:

```
ALTER SYSTEM SET shared_preload_libraries = 'pg_prewarm';
```

и затем перезагрузить сервер.

https://postgrespro.ru/docs/postgresql/12/pgprewarm

2. Для анализа используйте расширение pg_buffercache, показанное в демонстрации.

3. Инициализируйте тестовые таблицы, вызвав pgbench с ключом -i.

Затем выполняйте эталонный тест в одном и другом режимах на протяжении некоторого времени.

https://postgrespro.ru/docs/postgresql/12/pgbench

1. Прогрев кеша с помощью pg_prewarm

Расширение необходимо добавить в загружаемые библиотеки и перезапустить сервер:

```
=> ALTER SYSTEM SET shared_preload_libraries = 'pg_prewarm';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 12 main restart
```

Создадим таблицу с данными:

```
student$ psql
```

```
=> CREATE DATABASE arch_wal;
```

```
CREATE DATABASE
```

```
=> \c arch_wal
```

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

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

```
CREATE TABLE
```

```
=> INSERT INTO t(n) SELECT id FROM generate_series(1,10000) id;
```

```
INSERT 0 10000
```

Проверим наличие страниц в буферном кеше:

```
=> CREATE EXTENSION pg_buffercache;
```

```
CREATE EXTENSION
```

```
=> SELECT count(*)
FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('t'::regclass);
```

```
 count
-------
    47
(1 row)
```

Перезапустим сервер.

```
student$ sudo pg_ctlcluster 12 main restart
```

Проверим буферный кеш:

```
student$ psql arch_wal
```

```
=> SELECT count(*)
FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('t'::regclass);
```

```
 count
-------
    47
(1 row)
```

Все страницы были автоматически загружены в буферный кеш при старте сервера.

Отключим расширение.

```
=> ALTER SYSTEM RESET shared_preload_libraries;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 12 main restart
```

2. Массовое вытеснение

```
student$ psql arch_wal
```

Размер буферного кеша:

```
=> SHOW shared_buffers;
```

```
 shared_buffers
----------------
 128MB
(1 row)
```

В созданной таблице 10000 строк занимают:

```
=> SELECT pg_size_pretty(pg_table_size('t'));
```

```
 pg_size_pretty
----------------
 384 kB
(1 row)
```

Добавим еще строк так, чтобы увеличить размер таблицы примерно до половины буферного кеша:

```
=> INSERT INTO t(n) SELECT id FROM generate_series(1,2000000) id;
```

```
INSERT 0 2000000
```

```
=> SELECT pg_size_pretty(pg_table_size('t')),
          pg_table_size('t')/8192 AS pages;
```

```
 pg_size_pretty | pages
----------------+-------
 70 MB          |  8899
(1 row)
```

Перезагрузим сервер.

```
student$ sudo pg_ctlcluster 12 main restart
```

```
student$ psql arch_wal
```

Выполним запрос, читающий все табличные страницы:

```
=> SELECT count(*) FROM t;
```

```
  count
---------
 2010000
(1 row)
```

В буферный кеш попало всего:

```
=> SELECT count(*)
FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('t'::regclass);
```

```
 count
-------
    96
(1 row)
```

Это работает механизм, защищающий буферный кеш от массового вытеснения «одноразовыми» данными, которые вряд ли понадобятся снова. Он действует при полном сканировании больших таблиц, очистке и т. п. Такие операции используют только небольшую часть кеша (так называемое буферное кольцо).

3. Зависимость производительности от режима фиксации

Инициализируем тестовые таблицы:

```
student$ pgbench -i arch_wal
```

```
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...
100000 of 100000 tuples (100%) done (elapsed 0.20 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

Синхронная фиксация включена:

```
=> SHOW synchronous_commit;
```

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

Запускаем эталонный тест TPC-B на 30 секунд:

```
student$ pgbench -T 30 arch_wal
```

```
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: 3200
latency average = 9.377 ms
tps = 106.639503 (including connections establishing)
tps = 106.653359 (excluding connections establishing)
```

Показатель — количество транзакций в секунду (tps).

Выключаем синхронную фиксацию:

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

```
ALTER SYSTEM
```

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

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

Повторяем тест:

```
student$ pgbench -T 30 arch_wal
```

```
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: 79422
latency average = 0.378 ms
tps = 2646.475686 (including connections establishing)
tps = 2646.804418 (excluding connections establishing)
```

На этом типе нагрузки и на данном оборудовании число транзакций в секунду примерно в два раза выше. В других случаях коэффициент, конечно, будет другим.

Восстановим значение параметра по умолчанию:

```
=> ALTER SYSTEM RESET synchronous_commit;
```

```
ALTER SYSTEM
```

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

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