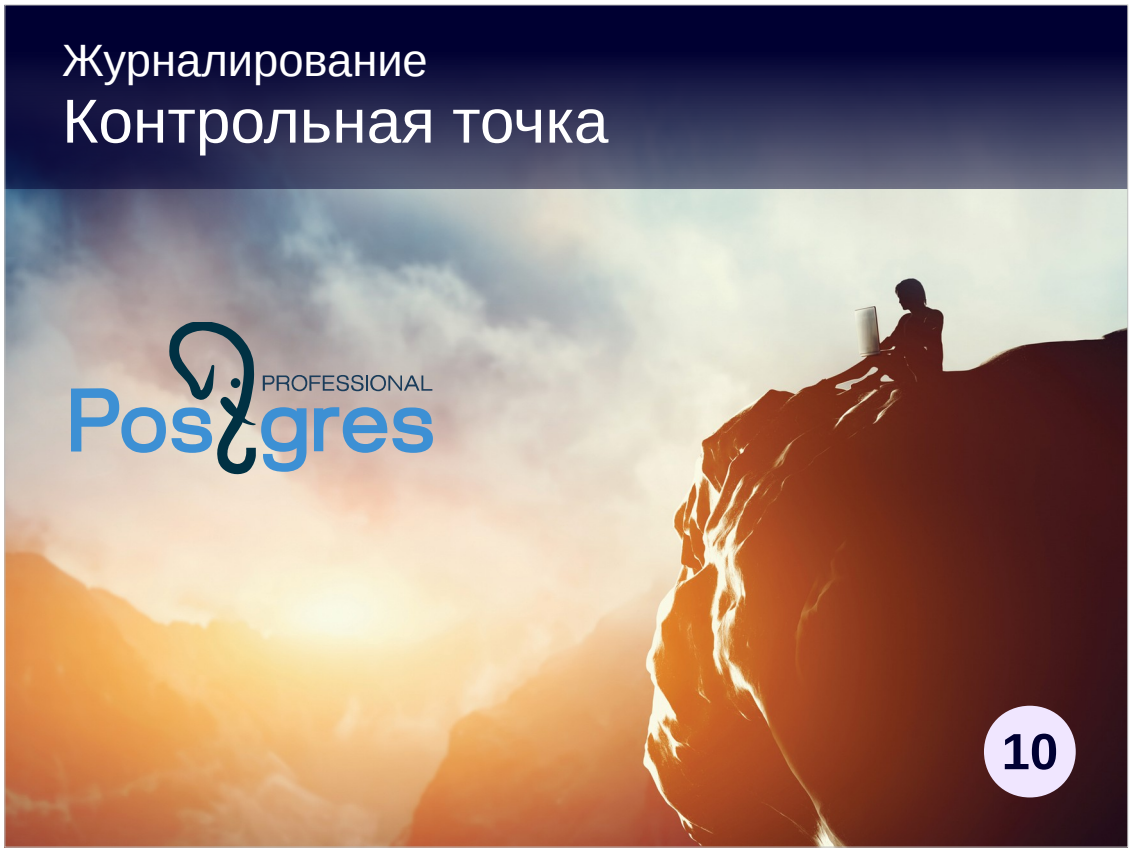


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Процесс контрольной точки

Процесс фоновой записи

Мониторинг

Контрольная точка

Необходимость контрольной точки

Процесс выполнения контрольной точки

Алгоритм восстановления после сбоя

Настройки

Необходимость

Размер хранимых журнальных записей

- начиная с какого сегмента надо применять журнальные записи при восстановлении?
- активно использующаяся страница может не вытесняться из кэша

Время восстановления

- сколько времени займет восстановление после сбоя?

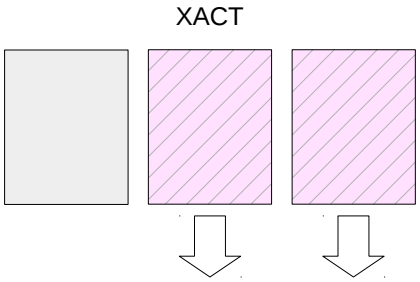
Если не предпринять специальных мер, то активно использующаяся страница, попав в буферный кэш, может никогда не вытесняться. Это означает, что при восстановлении после сбоя нам придется просматривать *все* журнальные записи, созданные с момента запуска сервера.

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

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

Процесс контрольной точки

записываются грязные буферы XACT



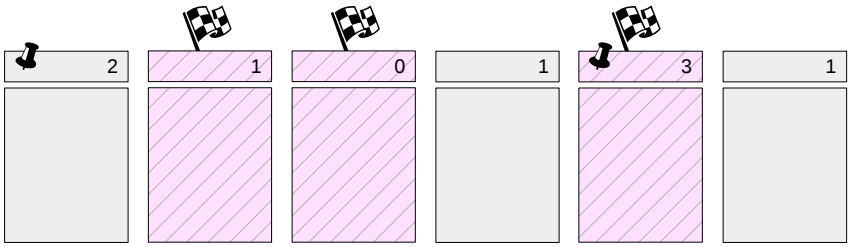
Рассмотрим подробнее, что происходит при выполнении контрольной точки.

Во-первых, помимо буферного кэша, в оперативной памяти располагаются и другие структуры, содержимое которых нужно сохранять на диске.

В частности, контрольная точка сбрасывает на диск буферы статуса транзакций (XACT). Поскольку количество таких буферов невелико (их 128), они записываются сразу же.

Процесс контрольной точки

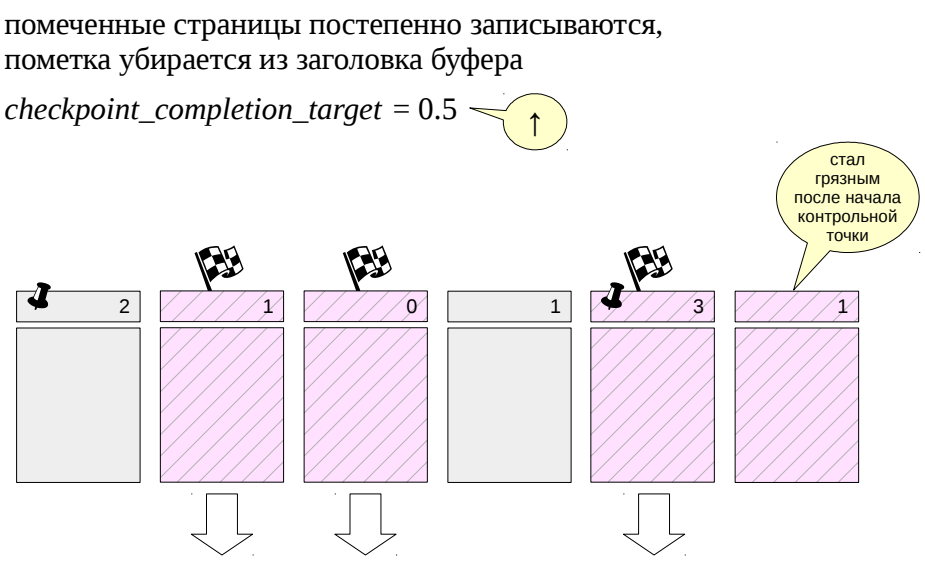
помечаются измененные страницы в буферном кэше



Во-вторых, основная работа контрольной точки — сбросить на диск все страницы из буферного кэша, которые были изменены на момент начала контрольной точки.

Поскольку размер буферного кэша может быть очень велик, сбрасывать сразу все страницы плохо — это создаст большую нагрузку на дисковую подсистему. Поэтому сначала все измененные на текущий момент страницы помечаются в заголовке специальным флагом...

Процесс контрольной точки



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

Помеченные буферы могут также быть записаны и серверными процессами — смотря кто доберется до буфера первым. В любом случае при записи снимается установленный ранее флаг, так что буфер будет записан только один раз.

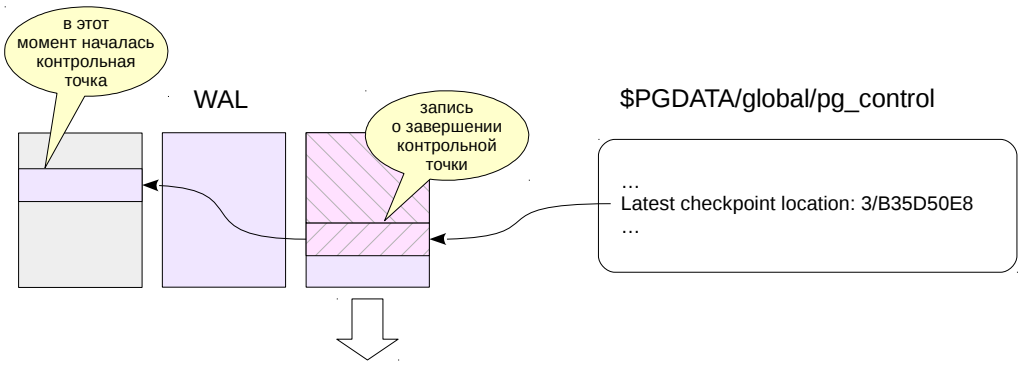
В процессе работы контрольной точки страницы продолжают изменяться в буферном кэше. Но новые грязные буферы уже не рассматриваются процессом контрольной точки, так как на момент начала работы они не были грязными.

Активность записи грязных буферов определяется значением параметра *checkpoint_completion_target*. Он показывает, какую часть времени между двумя соседними контрольными точками следует использовать для записи страниц. Значение по умолчанию равно 0.5, то есть запись занимает примерно половину времени между контрольными точками. Обычно значение увеличивают, например до 0.9, чтобы нагрузка была более равномерной. Значения выше 0.9 использовать не рекомендуется, поскольку фактически процесс может занять несколько больше времени, чем указано в параметре.

Процесс контрольной точки

в журнале создается запись о завершении контрольной точки с указанием момента ее начала

в файл `pg_control` записывается LSN контрольной точки



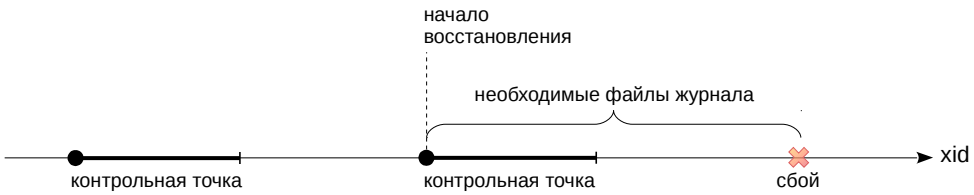
В конце работы процесс создает журнальную запись об окончании контрольной точки. В этой записи содержится LSN момента начала работы контрольной точки. Поскольку контрольная точка ничего не записывает в журнал в начале своей работы, по этому LSN может находиться любая журнальная запись.

Кроме того, в специальный файл $PGDATA/global/pg_control записывается указание на созданную журнальную запись. Таким образом можно быстро выяснить последнюю пройденную контрольную точку.

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. перезаписать нежурналируемые таблицы init-файлами
4. выполнить контрольную точку



Если в работе сервера произошел сбой, то при последующем запуске процесс startup обнаруживает это, посмотрев в файл pg_control и увидев статус, отличный от «shut down». Тогда будет выполнено автоматическое восстановление.

Сначала процесс прочитает из того же pg_control LSN записи о последней контрольной точке. (Для полноты картины заметим, что, если присутствует файл backup_label, то запись о контрольной точке читается из него — это нужно для восстановления из резервных копий.) Из этой записи процесс узнает позицию LSN начала контрольной точки.

Далее процесс startup будет читать журнал от найденной позиции, последовательно применяя записи к страницам, если в этом есть необходимость (что можно проверить, сравнив LSN страницы на диске с LSN журнальной записи). Изменение страниц происходит в буферном кэше — для этого postmaster запускает необходимые фоновые процессы.

Аналогично записи применяются и к файлам: например, если запись говорит, что файл должен быть создан, а его нет — файл создается.

В конце процесса все нежурналируемые таблицы перезаписываются с помощью образов в init-файлах.

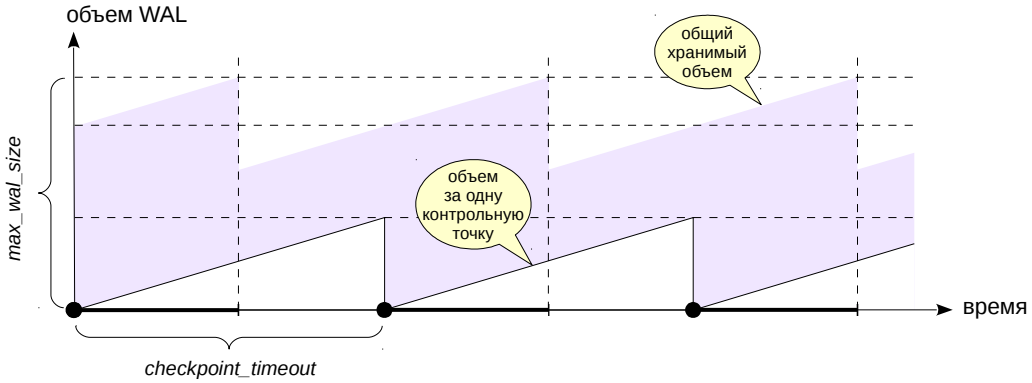
На этом процесс startup завершает работу, а процесс checkpointer выполняет контрольную точку, чтобы зафиксировать восстановленное состояние.

Частота контрольных точек

Настройки

checkpoint_timeout = 5min ↑

max_wal_size = 1GB



10

Обычно контрольная точка настраивается из следующих соображений.

Сначала надо определиться, какая частота срабатываний нас устраивает (исходя из допустимого времени восстановления и объема журнальных файлов за это время при стандартной нагрузке). Чем реже можно позволить себе контрольные точки, тем лучше — это сокращает накладные расходы.

Устраивающая частота записывается в параметр *checkpoint_timeout* (значение по умолчанию — 5 минут — слишком мало, обычно время увеличивают до получаса).

Однако возможна ситуация, когда нагрузка увеличится от расчетной, и за указанное время будет сгенерирован слишком большой объем журнальных записей. Для этого в параметре *max_wal_size* указывают общий допустимый объем журнальных записей.

Для восстановления после сбоя сервер должен хранить файлы с момента последней контрольной точки, плюс файлы, накопившиеся во время работы текущей контрольной точки. Но вплоть до версии 11 PostgeSQL дополнительно хранит файлы и за позапрошлую контрольную точку, так что общий объем можно оценить как

(2 + *checkpoint_completion_target*) * *объем-между-контр-точками.*

Таким образом, большая часть контрольных точек происходит по расписанию: раз в *checkpoint_timeout* единиц времени. Но при повышенной нагрузке контрольная точка вызывается чаще, при достижении объема *max_wal_size*.

Объем файлов журнала

Сервер хранит журнальные файлы

- необходимые для восстановления (обычно < *max_wal_size*)
- еще не прочитанные через слоты репликации
- еще не записанные в архив, если настроена непрерывная архивация
- не превышающие по объему минимальной отметки *min_wal_size*

Настройки

max_wal_size	= 1GB
min_wal_size	= 80MB
wal_keep_segments	= 0

Надо понимать, что объем, указанный в параметре *max_wal_size*, может быть превышен. Это не жесткое ограничение, а пожелание.

Кроме того, сервер не имеет права стереть журнальные файлы, еще не переданные через слоты репликации, и еще не записанные в архив при непрерывном архивировании. Это может привести к перерасходу места, так что если этот функционал используется, необходим постоянный мониторинг.

Также можно установить минимальный порог объема, при котором журнальные файлы не удаляются, а просто используются заново. Это позволяет сэкономить на постоянном создании и удалении файлов. Историческая настройка для этого — *wal_keep_segments* (количество файлов), а новая и рекомендуемая — *min_wal_size* (объем).

https://postgrespro.ru/docs/postgresql/10/wal-configuration

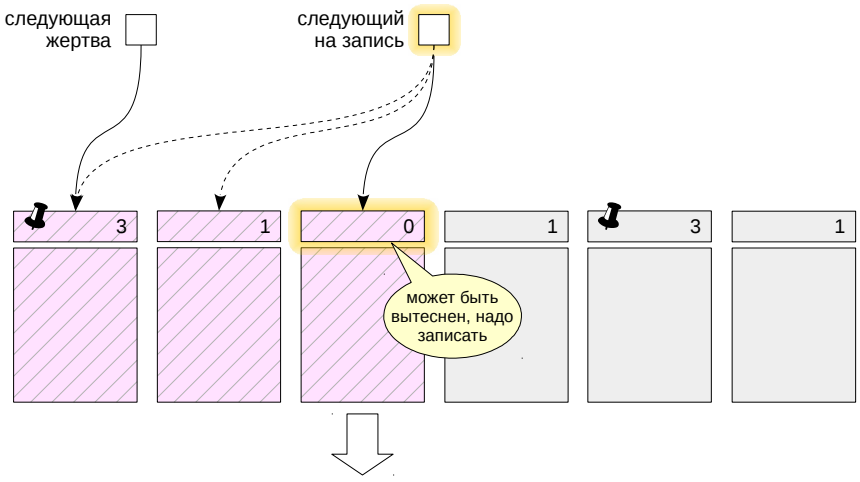
https://postgrespro.ru/docs/postgresql/10/runtime-config-wal#RUNTIME-CONFIG-WAL-CHECKPOINTS

Фоновая запись

Процесс фоновой записи

Настройка

Процесс фоновой записи



Когда обслуживающий процесс собирается вытеснить страницу из буфера, он может обнаружить, что буфер грязный, и ему придется записать страницу на диск. Чтобы этого не происходило, в дополнение к процессу контрольной точки (checkpointer) существует также процесс фоновой записи (background writer, bgwriter или просто 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 данные (грязные),

- 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. За счет этого обслуживающий процесс скорее всего обнаружит, что выбранный им буфер не является грязным.

Процесс фоновой записи

Алгоритм

уснуть на *bgwriter_delay*

если в среднем за цикл запрашивается *N* буферов, то записать
N * *bgwriter_lru_multiplier* ≤ *bgwriter_lru_maxpages*
грязных буферов

Настройки

bgwriter_delay	= 200ms
bgwriter_lru_maxpages	= 100
bgwriter_lru_multiplier	= 2.0

Процесс фоновой записи работает циклами максимум по *bgwriter_lru_maxpages* страниц, засыпая между циклами на *bgwriter_delay*.

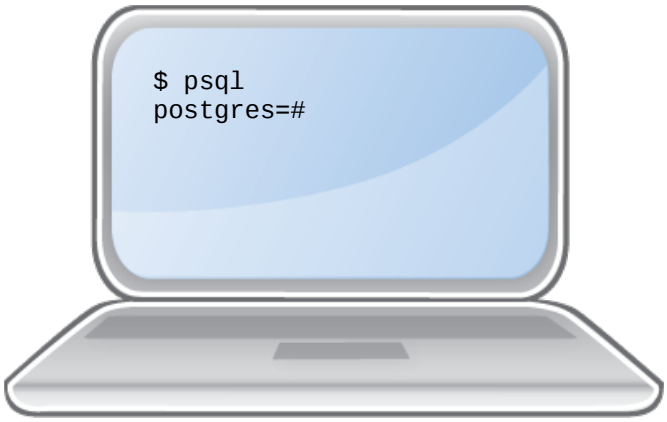
(Таким образом, если установить параметр *bgwriter_lru_maxpages* в ноль, процесс фактически не будет работать.)

Точное значение буферов для записи определяется по среднему количеству буферов, которые запрашивались обслуживающими процессами с прошлого запуска (используется скользящее среднее, чтобы сгладить неравномерность между запусками, но при этом не зависеть от давней истории). Вычисленное количество буферов умножается на коэффициент *bgwriter_lru_multiplier*.

Если процесс совсем не обнаружил грязных буферов (то есть в системе ничего не происходит), он «впадает в спячку», из которой его выводит обращение серверного процесса за буфером. После этого процесс просыпается и опять работает обычным образом.

Процесс фоновой записи имеет смысл настраивать после того, как настроены контрольные точки. Совместно с контрольной точкой эти процессы должны успевать записывать грязные буферы до того, как они потребуются обслуживающим процессам.

Демонстрация



Итоги

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

Контрольная точка и фоновая запись сбрасывают на диск грязные буферы

Обслуживающие процессы могут сбрасывать грязные буферы, но не должны этим заниматься

Практика

1. Настройте выполнение контрольной точки раз в 30 секунд. Установите параметры *min_wal_size* и *max_wal_size* в 16 МБ.
2. Несколько минут с помощью утилиты pgbench подавайте нагрузку 100 транзакций/сек.
3. Измерьте, какой объем журнальных файлов был сгенерирован за это время. Оцените, какой объем приходится в среднем на одну контрольную точку.
4. Проверьте данные статистики: все ли контрольные точки выполнялись по расписанию? Как можно объяснить полученный результат?
5. Сбросьте настройки в значения по умолчанию.