Практические работы по дисциплине «Технологии обработки транзакций клиент-серверных приложений» направления подготовки бакалавриата 09.03.04 «Программная инженерия»

Практическая работа №6

Транзакции. Журнал транзакций.

Теория для понимания практики:

Журнал в общем понимании СУБД

Журнализация изменений БД

Одним из основных требований к развитым СУБД является надежность хранения баз данных. Это требование предполагает, в частности, возможность восстановления согласованного состояния базы данных после любого рода аппаратных и программных сбоев. Очевидно, что для выполнения восстановлений необходима некоторая дополнительная информация. В подавляющем большинстве современных реляционных СУБД такая избыточная дополнительная информация поддерживается в виде журнала изменений базы данных.

Итак, общей целью журнализации изменений баз данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя. Поскольку основой поддержания целостного состояния базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции. Общими принципами восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Это, собственно, и означает, что восстанавливается последнее по времени согласованное состояние базы данных.

Возможны следующие ситуации, при которых требуется производить восстановление состояния базы данных:

Индивидуальный откат транзакции. Тривиальной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK. Возможны также ситуации, когда откат транзакции инициируется системой. Примерами могут быть возникновение исключительной ситуации в прикладной программе (например, деление на ноль) или выбор транзакции в качестве жертвы при обнаружении синхронизационного тупика. Для восстановления согласованного состояния базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть при аварийном выключении электрического питания, при возникновении неустранимого сбоя процессора (например, срабатывании контроля оперативной памяти) и т.д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных устройств внешней памяти может возникать сравнительно редко, но тем не менее, СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и журнал изменений базы данных.

Во всех трех случаях основой восстановления является избыточное хранение данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

Возможны два основных варианта ведения журнальной информации. В первом варианте для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакцией. Эти локальные журналы используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти. Кроме того, поддерживается общий журнал изменений базы данных,

используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант - поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов. Далее мы рассматриваем именно этот вариант.

Журнализация и буферизация

Журнализация изменений тесно связана не только с управлением транзакциями, но и с буферизацией страниц базы данных в оперативной памяти. По причинам объективно существующей разницы в скорости работы процессоров и оперативной памяти и устройств внешней памяти (эта разница в скорости существовала, существует и будет существовать всегда) буферизация страниц базы данных в оперативной памяти - единственный реальный способ достижения удовлетворительной эффективности СУБД.

Если бы запись об изменении базы данных, которая должна поступить в журнал при выполнении любой операции модификации базы данных, реально немедленно записывалась бы во внешнюю память, это привело бы к существенному замедлению работы системы. Поэтому записи в журнал тоже буферизуются: при нормальной работе очередная страница выталкивается во внешнюю память журнала только при полном заполнении записями.

Но реальная ситуация является более сложной. Имеются два вида буферов - буфер журнала и буфер страниц оперативной памяти, которые содержат связанную информацию. И те, и другие буфера могут выталкиваться во внешнюю память. Проблема состоит в выработке некоторой общей политики выталкивания, которая обеспечивала бы возможности восстановления состояния базы данных после сбоев.

Проблема не возникает при индивидуальных откатах транзакций, поскольку в этих случаях содержимое оперативной памяти не утрачено и

можно пользоваться содержимым как буфера журнала, так и буферов страниц базы данных. Но если произошел мягкий сбой, и содержимое буферов утрачено, для проведения восстановления базы данных необходимо иметь некоторое согласованное состояние журнала и базы данных во внешней памяти.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) - "пиши сначала в журнал", и состоит в том, что если требуется вытолкнуть во внешнюю память измененный объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи о его изменении.

Другими словами, если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Обратное неверно, т.е. если во внешней памяти журнале содержится запись о некоторой операции изменения объекта базы данных, то сам измененный объект может отсутствовать во внешней памяти базы данных.

Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна быть в состоянии восстановить состояние базы данных, содержащее результаты всех зафиксированных к моменту сбоя транзакций.

Простым решением было бы выталкивание буфера журнала, за которым следует массовое выталкивание буферов страниц базы данных, изменявшихся данной транзакцией. Довольно часто так и делают, но это вызывает существенные накладные расходы при выполнении операции фиксации

транзакции.

Оказывается, что минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце транзакции.

Рассмотрим теперь, как можно выполнять операции восстановления базы данных в различных ситуациях, если в системе поддерживается общий для всех транзакций журнал с общей буферизацией записей, поддерживаемый в соответствии с протоколом WAL.

Индивидуальный откат транзакции

Для того, чтобы можно было выполнить по общему журналу индивидуальный откат транзакции, все записи в журнале от данной транзакции связываются в обратный список. Началом списка для незакончившихся транзакций является запись о последнем изменении базы данных, произведенном данной транзакцией. Для закончившихся транзакций (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит первая запись об изменении базы данных, произведенном данной транзакцией. Обычно в каждой записи проставляется уникальный идентификатор транзакции, чтобы можно было восстановить прямой список записей об изменениях базы данных данной транзакцией.

Итак, индивидуальный откат транзакции (еще раз подчеркнем, что это возможно только для незакончившихся транзакций) выполняется следующим образом:

Выбирается очередная запись из списка данной транзакции.

Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции

DELETE выполняется INSERT, и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.

Любая из этих обратных операций также журнализуются. Собственно для индивидуального отката это не нужно, но при выполнении индивидуального отката транзакции может произойти мягкий сбой, при восстановлении после которого потребуется откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат.

При успешном завершении отката в журнал заносится запись о конце транзакции. С точки зрения журнала такая транзакция является зафиксированной.

Восстановление после мягкого сбоя

К числу основных проблем восстановление после мягкого сбоя относится то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, страницу данных и несколько страниц индексов. Страницы базы данных буферизуются в оперативной памяти и выталкиваются независимо. Несмотря на применение протокола WAL, после мягкого сбоя набор страниц внешней памяти базы данных может оказаться несогласованным, т.е. часть страниц внешней памяти соответствует объекту до изменения, часть - после изменения. К такому состоянию объекта не применимы операции логического уровня.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованы, т.е. соответствуют состоянию объекта либо после его изменения, либо до изменения.

Будем считать, что в журнале отмечаются точки физической согласованности базы данных - моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память.

Немного позже мы рассмотрим, как можно достичь физической согласованности. Назовем такие точки tpc (time of physical consistency).

Тогда к моменту мягкого сбоя возможны следующие состояния транзакций:

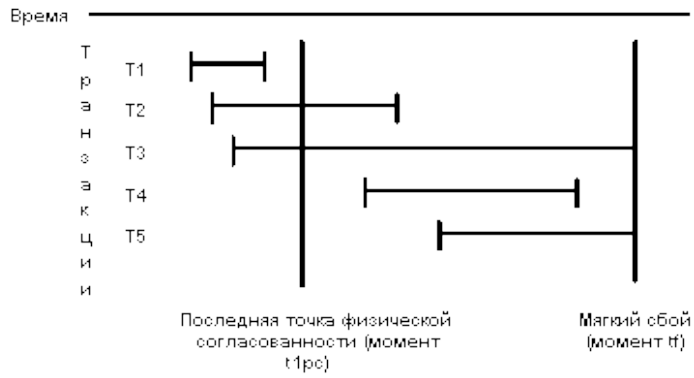

Рисунок 1 – Состояния транзакций

Предположим, что некоторым способом удалось восстановить внешнюю память базы данных к состоянию на момент времени tlpc (как это можно сделать - немного позже). Тогда:

Для транзакции T1 никаких действий производить не требуется. Она закончилась до момента tlpc, и все ее результаты отражены во внешней памяти базы данных.

Для транзакции T2 нужно повторно выполнить оставшуюся часть операций (redo). Действительно, во внешней памяти полностью отсутствуют следы операций, которые выполнялись в транзакции T2 после момента tlpc. Следовательно, повторная прямая интерпретация операций T2 корректна и приведет к логически согласованному состоянию базы данных (поскольку транзакция T2 успешно завершилась до момента мягкого сбоя, в журнале содержатся записи обо всех изменениях, произведенных этой транзакцией).

Для транзакции T3 нужно выполнить в обратном направлении первую часть операций (undo). Действительно, во внешней памяти базы данных полностью отсутствуют результаты операций T3, которые были выполнены после момента tlpc. С другой стороны, во внешней памяти гарантированно присутствуют результаты операций T3, которые были выполнены до момента tlpc. Следовательно, обратная интерпретация операций T3 корректна и

приведет к согласованному состоянию базы данных (поскольку транзакция Т3 не завершилась к моменту мягкого сбоя, при восстановлении необходимо устранить все последствия ее выполнения).

Для транзакции Т4, которая успела начаться после момента tlpc и закончиться до момента мягкого сбоя, нужно выполнить полную повторную прямую интерпретацию операций (redo).

Наконец, для начавшейся после момента tlpc и не успевшей завершиться к моменту мягкого сбоя транзакции Т5 никаких действий предпринимать не требуется. Результаты операций этой транзакции полностью отсутствуют во внешней памяти базы данных.

Физическая согласованность базы данных

Каким же образом можно обеспечить наличие точек физической согласованности базы данных, т.е. как восстановить состояние базы данных в момент tpc? Для этого используются два основных подхода: подход, основанный на использовании теневого механизма, и подход, в котором применяется журнализация постраничных изменений базы данных.

При открытии файла таблица отображения номеров его логических блоков в адреса физических блоков внешней памяти считывается в оперативную память. При модификации любого блока файла во внешней памяти выделяется новый блок. При этом текущая таблица отображения (в оперативной памяти) изменяется, а теневая - сохраняется неизменной. Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия. Для явного восстановления файла достаточно повторно считать в оперативную память теневую таблицу отображения.

Общая идея теневого механизма показана на следующем рисунке:

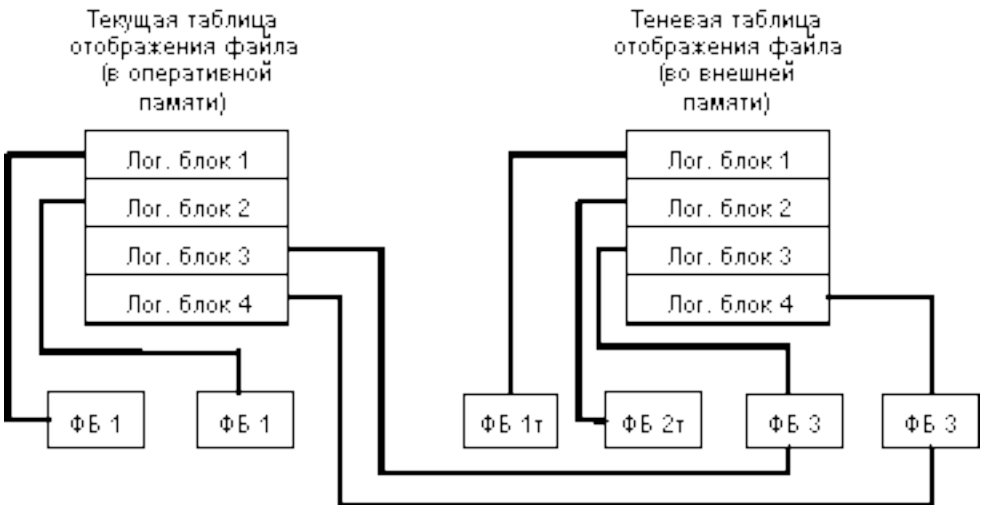

Рисунок 2 – Общая идея теневого механизма

В контексте базы данных теневой механизм используется следующим образом. Периодически выполняются операции установления точки физической согласованности базы данных (checkpoints в System R). Для этого все логические операции завершаются, все буфера оперативной памяти, содержимое которых не соответствует содержимому соответствующих страниц внешней памяти, выталкиваются. Теневая таблица отображения файлов базы данных заменяется на текущую (правильнее сказать, текущая таблица отображения записывается на место теневой).

Восстановление к tlpc происходит мгновенно: текущая таблица отображения заменяется на теневую (при восстановлении просто считывается теневая таблица отображения). Все проблемы восстановления решаются, но за счет слишком большого перерасхода внешней памяти. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных. Теневой механизм - это надежное, но слишком грубое средство. Обеспечивается согласованное состояние внешней памяти в один общий для всех объектов момент времени. На самом деле, достаточно иметь набор согласованных наборов страниц, каждому из которых может соответствовать свой набор времени.

Для достижения такого более слабого требования наряду с логической журнализацией операций изменения базы данных производится журнализация постраничных изменений. Первый этап восстановления после мягкого сбоя

состоит в постраничном откате незакончившихся логических операций. Подобно тому, как это делается с логическими записями по отношению к транзакциям, последней записью о постраничных изменениях от одной логической операции является запись о конце операции. Для того, чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании любой страницы из буфера оперативную память в нее помещается идентификатор последней записи о постраничном изменении этой страницы. Имеются и другие технические нюансы.

В этом подходе имеются два поднаправления. В первом поднаправлении поддерживается общий журнал логических и страничных операций. Естественно, наличие двух видов записей, интерпретируемых абсолютно по-разному, усложняет структуру журнала. Кроме того, записи о постраничных изменениях, актуальность которых носит локальный характер, существенно (и не очень осмысленно) увеличивают журнал.

Поэтому все более популярным становится поддержание отдельного (короткого) журнала постраничных изменений. Такая техника применяется, например, в известном продукте Informix Online.

Восстановление после жесткого сбоя

Понятно, что для восстановления последнего согласованного состояния базы данных после жесткого сбоя журнала изменений базы данных явно недостаточно. Основой восстановления в этом случае являются журнал и архивная копия базы данных.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем для всех закончившихся транзакций выполняется redo, т.е. операции повторно выполняются в прямом смысле.

Более точно, происходит следующее:

- по журналу в прямом направлении выполняются все операции;
- для транзакций, которые не закончились к моменту сбоя, выполняется откат.

На самом деле, поскольку жесткий сбой не сопровождается утратой буферов оперативной памяти, можно восстановить базу данных до такого

уровня, чтобы можно было продолжить даже выполнение незакончившихся транзакций. Но обычно это не делается, потому что восстановление после жесткого сбоя — это достаточно длительный процесс.

Хотя к ведению журнала предъявляются особые требования по части надежности, в принципе возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной копии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных, но это лучше, чем ничего.

Последний вопрос, который мы коротко рассмотрим, относится к производству архивных копий базы данных. Самый простой способ - архивировать базу данных при переполнении журнала. В журнале вводится так называемая "желтая зона", при достижении которой образование новых транзакций временно блокируется. Когда все транзакции закончатся, и следовательно, база данных придет в согласованное состояние, можно производить ее архивацию, после чего начинать заполнять журнал заново.

Можно выполнять архивацию базы данных реже, чем переполняется журнал. При переполнении журнала и окончании всех начатых транзакций можно архивировать сам журнал. Поскольку такой архивированный журнал, по сути дела, требуется только для воссоздания архивной копии базы данных, журнальная информация при архивации может быть существенно сжата.

Буферный кеш и журнал в PostgreSQL

Рассматриваемые темы:

- Устройство буферного кэша
- Алгоритм вытеснения
- Журнал предзаписи
- Контрольная точка
- Процессы, связанные с буферным кэшем и журналом

Буферный кэш

Массив буферов состоит из страницы данных (8 КБ) и доп. Информации.

Блокировки в памяти используются для совместного доступа.

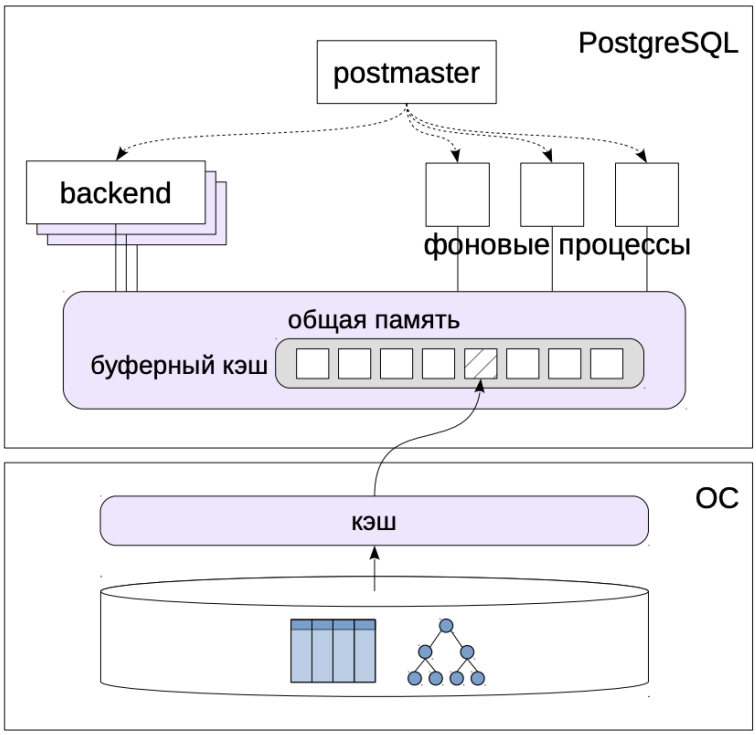

Рисунок 3 – Общая архитектура

Буферный кэш используется для сглаживания скорости работы оперативной памяти и дисков. Он состоит из массива буферов, которые содержат страницы данных и дополнительную информацию (например,имя файла и положение страницы внутри этого файла).

Размер страницы обычно составляет 8 КБ; размер можно изменить только при сборке PostgreSQL.

Любая работа со страницами данных проходит через буферный кэш.

Если какой-либо процесс собирается работать со страницей, он в первую очередь пытается найти ее в кэше. Если ее нет, он обращается к операционной системе с просьбой прочитать эту страницу и помещает ее в буферный кэш. (Обратите внимание, что ОС может прочитать страницу с диска, а может обнаружить ее в собственном кэше.)

После того, как странице записана в буферный кэш, к ней можно обращаться многократно без накладных расходов на вызовы ОС.

Однако буферный кэш, как и другие структуры общей памяти, защищен блокировками для управления одновременным доступа. Хотя блокировки и реализованы эффективно, доступ к буферному кэшу далеко не так быстр, как

простое обращение к оперативной памяти.

Поэтому в общем случае чем меньше данных читает и изменяет запрос, тем быстрее он будет работать.

Алгоритм вытеснения

Вытеснение редко используемых страниц:

1. «грязный» буфер записывается на диск
2. на освободившееся место читается другая страница

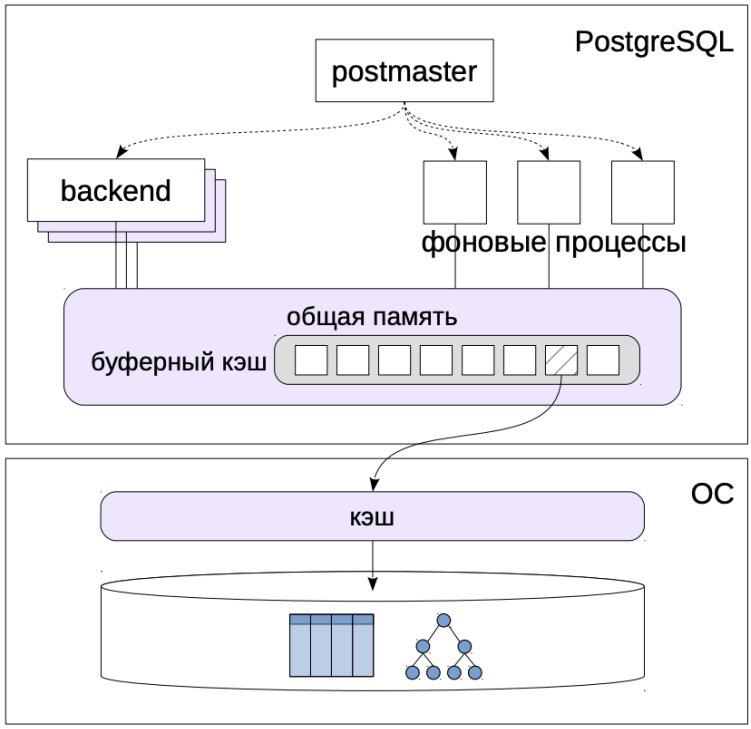

Рисунок 4 – Вытеснение

Размер буферного кэша обычно не так велик, чтобы база данных помещалась в него целиком. Его ограничивают и доступная оперативная память, и возрастающие при его увеличении накладные расходы. Поэтому при чтении очередной страницы рано или поздно окажется, что место в буферном кэше закончилось. В этом случае применяется *вытеснение* страниц.

Алгоритм вытеснения выбирает в кэше страницу, которая в последнее время использовалась реже других, и заменяет ее новой. Если выбранная страница изменялась, то ее предварительно надо записать на диск, чтобы не потерять изменения (буфер, содержащий измененную страницу, называется «грязным»).

Такой алгоритм вытеснения называется LRU — Least Recently Used. Он сохраняет в кэше данные, с которыми происходит активная работа.

Таких «горячих» данных обычно не так много, и при достаточном объеме буферного кэша получается существенно сократить количество обращений к ОС (и дисковых операций).

Журнал предзаписи (WAL)

Проблема: при сбое теряются данные из оперативной памяти, не записанные на диск.

Журнал – поток информации о выполняемых действиях, позволяющей повторно выполнить потерянные при сбое операции, запись попадает на диск раньше, чем измененные данные.

Журнал защищает страницы таблиц, индексов и других объектов статус транзакций (xact).

Журнал не защищает временные и нежурналируемые таблицы.

Наличие буферного кэша (и других буферов в оперативной памяти) увеличивает производительность, но уменьшает надежность. В случае сбоя в СУБД содержимое буферного кэша потеряется. Если сбой произойдет в операционной системе или на аппаратном уровне, то пропадет содержимое и буферов ОС (но с этим справляется сама операционная система).

Для обеспечения надежности PostgreSQL использует журналирование. При выполнении любой операции формируется запись, содержащая минимально необходимую информацию для того, чтобы операцию можно было выполнить повторно. Такая запись должна попасть на диск(или другой энергонезависимый накопитель) раньше, чем будут записаны изменяемые операцией данные (поэтому он и называется *журналом предварительной записи*).

Файлы журнала традиционно располагались в каталоге PGDATA/pg_xlog; начиная с версии 10 каталог переименован в pg_wal.

Журнал защищает все объекты, работа с которыми ведется в оперативной памяти: таблицы, индексы и другие объекты, статустранзакций.

В журнал не попадают данные о *временных таблицах* (такие таблицы доступны только создавшему из пользователю и только на время сеанса или транзакции) и о нежурналируемых таблицах (такие таблицы ничем не отличаются от обычных, кроме того, что не защищены журналом). В случае сбоя такие таблицы просто очищаются. Смысл их существования в том, что работа с ними происходит быстрее.

Производительность

Синхронный режим

- запись при фиксации
- обслуживающий процесс

Асинхронный режим

- фоновая запись
- walwriter

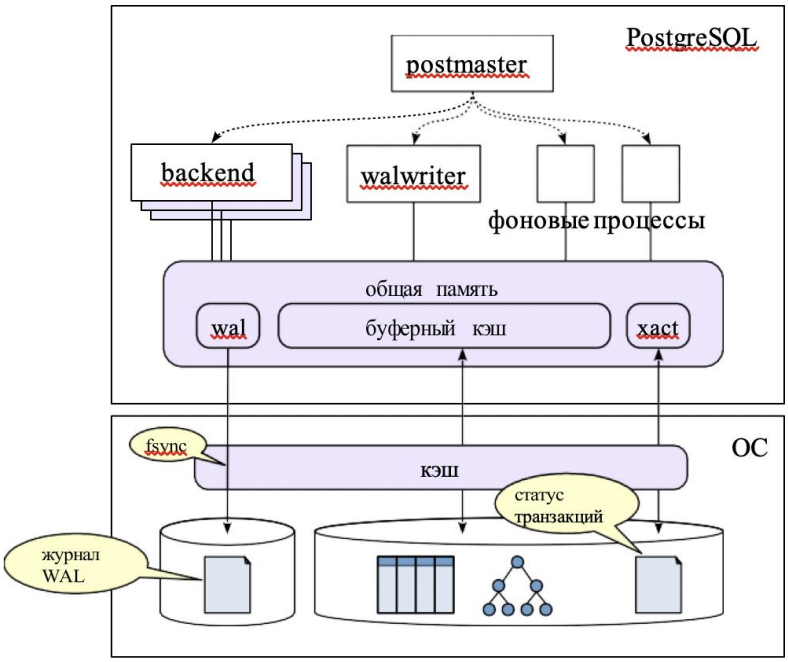

Рисунок 5 – Детализация архитектуры

Механизм журналирования более эффективен, чем работа напрямую с диском без буферного кэша. Во-первых, размер журнальных записей меньше, чем размер целой страницы данных; во-вторых, журнал записывается строго последовательно (и не читается, пока не случится сбой), с чем вполне

справляются простые HDD-диски.

На эффективность можно также влиять настройкой. Если запись происходит сразу (синхронно), то гарантируется, что зафиксированная транзакция не пропадет. Но запись — довольно дорогая операция,

в течение которой обслуживающий процесс, выполняющий фиксацию, вынужден ждать. Чтобы журнальная запись не «застряла» в кэше операционной системы, выполняется вызов fsync: PostgreSQL полагается на то, что это гарантирует попадание данных на энергонезависимый носитель.

Поэтому есть и режим отложенной (асинхронной) записи. В этом случае записи пишутся фоновым процессом walwriter постепенно, с небольшой задержкой. Надежность уменьшается, зато производительность увеличивается. Но и в этом случае после сбоя гарантируется восстановление согласованности.

Контрольная точка

Периодический сброс всех грязных буферов на диск гарантирует попадание на диск всех изменений до контрольной точки ограничивает размер журнала, необходимого для восстановления

Восстановление при сбое

- начинается с последней контрольной точки
- последовательно проигрываются записи, если изменений нет на диске

Рисунок 6 – Восстановление с контрольной точки

При запуске PostgreSQL после сбоя сервер входит в режим восстановления. На диске в это время находится несогласованная информация: одни страницы были записаны в одно время, другие ——в другое.

Чтобы восстановить согласованность, PostgreSQL читает журнал предзаписи и последовательно проигрывает каждую журнальную запись, если

соответствующее изменение не попало на диск. Таким образом восстанавливаются все транзакции, кроме тех, запись о фиксации которых не успела попасть в журнал.

Однако объем журнала за время работы сервера может достигать гигантских размеров. Хранить его целиком и целиком просматривать при сбое совершенно не реально. Поэтому СУБД периодически выполняет *контрольную точку:* принудительно сбрасывает на диск всегрязные буферы (включая состояние транзакций). Это гарантирует, что изменения всех транзакций до момента контрольной точки находятся на диске.

Контрольная точка может занимать много времени, и это нормально. Собственно «точка», о которой мы говорим как о моменте времени ——это начало процесса. Но точка считается выполненной только после того, как записаны *все* грязные буферы, которые имелись на моментначала процесса.

Восстановление после сбоя начинается с ближайшей контрольнойточки, что позволяет хранить только файлы журнала, записанные с момента последней пройденной контрольной точки.

Основные процессы

- Запись журнала
- Контрольная точка
    - сброс всех грязных буферов
- Фоновая запись
    - сброс части грязных буферов
- Обслуживающие процессы
    - сброс вытесняемого грязного буфера

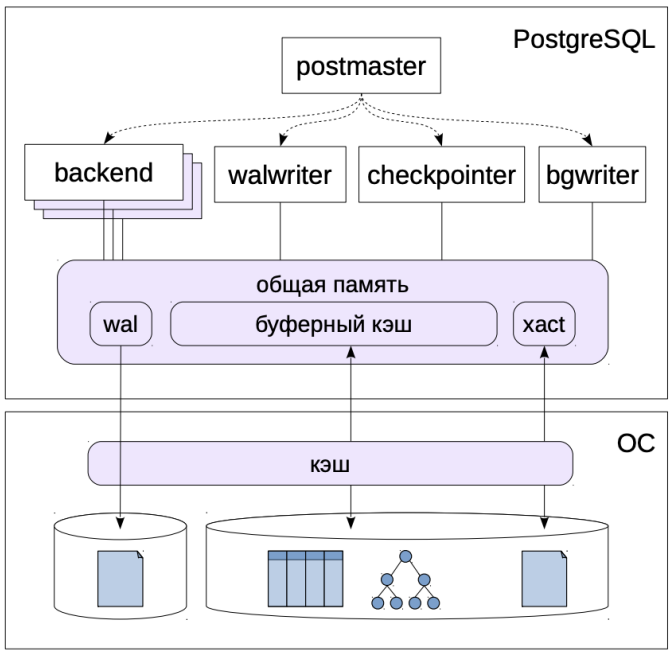

Рисунок 7 – Служебные процессы

Вернемся к иллюстрации и уточним оставшиеся служебные процессы, связанных с обслуживанием буферного кэша и журнала.

Во-первых, это процесс walwriter, занимающийся асинхронной записью журнала на диск. При синхронном режиме журнальные записи пишет тот процесс, который выполняет фиксацию транзакции.

Во-вторых, процесс контрольной точки checkpointer, периодически сбрасывающий все грязные буферы на диск.

В-третьих, процесс фоновой записи bgwriter (или просто writer). Этот процесс похож на процесс контрольной точки, но записывает только часть грязных буферов, причем те, которые с большой вероятностью будут вытеснены в ближайшее время. Таким образом, когда обслуживающему процессу понадобится буфер, он скорее всего найдетего не грязным и не будет терять время на сброс буфера на диск.

И в-четвертых, обслуживающие процессы, читающие данные в буферный кэш. Если, несмотря на работу процессов контрольной точки и фоновой записи, вытесняемый буфер окажется грязным, обслуживающий процесс самостоятельно запишет его на диск.

Уровни журнала

- Minimal
  - гарантия восстановления после сбоя
- Replica (по умолчанию)
  - резервное копирование
  - репликация: передача и проигрывание журнала на другом сервере
- Logical
  - логическая репликация: информация о добавлении, изменении и удалении табличных строк

Как уже говорилось, причиной появления журнала является необходимость защищать информацию от сбоев из-за потери содержимого оперативной памяти.

Однако журнал — механизм, который оказалось удобно применять и для других целей, если добавить в него дополнительную информацию.

Объем данных, который попадает в журнал, регулируется параметром wal_level.

На уровне minimal журнал обеспечивает только восстановление после сбоя. До версии 10 этот уровень устанавливался по умолчанию.

На уровне replica в журнал добавляется информация, позволяющая использовать его для создания резервных копий (модуль «Резервное копирование») и репликации (модуль «Репликация»). При репликации журнальные записи передаются на другой сервер и применяются там;таким образом создается и поддерживается точная копия (реплика) основного сервера.

До версии 9.6 существовали два отдельных уровня (archive и hot_standby), но они были объединены в один общий.

На уровне logical в журнал добавляется информация о добавлении, изменении и удалении табличных строк. Это позволяет организовать более гибкую логическую репликацию (также рассматривается в модуле

«Репликация»).

Итоги

Буферный кэш существенно ускоряет работу, уменьшая число дисковых операций

Надежность обеспечивается журналированием

Размер журнала ограничен благодаря контрольным точкам

Журнал удобен и используется во многих случаях

- для восстановления после сбоя
- при резервном копировании
- для репликации между серверами

Практика по журналу, желательна к выполнению

Практика 1

Логически журнал WAL можно представить в виде непрерывного потока записей. Каждая запись имеет номер, называемый LSN (Log Sequence Number). Это 64-разрядное число - смещение записи в байтах относительно начала журнала.

Текущую позицию показывает функция pg_current_wal_lsn (до версии 10 она называлась pg_current_xlog_location):

```
=> SELECT pg_current_wal_lsn();
 pg_current_wal_lsn
--------------------
 0/17B8CC8
(1 row)
```

Позиция записывается как два 32-разрядных числа через косую черту.

Запомним позицию в переменной psql:

```
=> SELECT pg_current_wal_lsn() AS pos1 \gset
```

Выполним теперь какие-нибудь операции и посмотрим, как изменилась позиция.

```
=> CREATE TABLE t(n integer);
CREATE TABLE
=> INSERT INTO t SELECT gen.id FROM generate_series(1,1000) AS
gen(id);
INSERT 0 1000
=> SELECT pg_current_wal_lsn();
 pg_current_wal_lsn
--------------------
 0/17E071C
```

```
(1 row)

=> SELECT pg_current_wal_lsn() AS pos2 \gset
```

Удобно смотреть не на абсолютные числа, а на разницу в байтах:

```
=> SELECT :'pos2'::pg_lsn - :'pos1'::pg_lsn;
 ?column?
----------
   162388
(1 row)
```

Физически журнал хранится в файлах по 16 МБ в отдельном каталоге. Начиная с версии 10 на них можно взглянуть не только в файловой системе (PGDATA/pg_wal), но и с помощью функции:

```
=> SELECT * FROM pg_ls_waldir() ORDER BY name;
           name           |   size   |      modification
--------------------------+----------+------------------------
 000000010000000000000001 | 16777216 | 2019-03-31 15:25:29+03
(1 row)
```

PostgreSQL удаляет файлы, не требующиеся для восстановления, по мере необходимости.

Практика 2

Процессы операционной системы

```
postgres$ ps -o pid,command --ppid `head -n 1
$PGDATA/postmaster.pid`
  PID COMMAND
 9241 postgres: checkpointer process
 9242 postgres: writer process
 9243 postgres: wal writer process
 9244 postgres: autovacuum launcher process
 9245 postgres: stats collector process
 9246 postgres: bgworker: logical replication launcher
```

К процессам, обслуживающим буферный кэш и журнал, можно отнести:

- checkpointer
- writer
- wal writer

Остановка в режиме fast

```
postgres$ rm /home/postgres/logfile
postgres$ pg_ctl -w -l /home/postgres/logfile -D
/usr/local/pgsql/data restart
waiting for server to shut down.... done
server stopped
waiting for server to start.... done
server started
```

Журнал сообщений сервера:

```
postgres$ cat /home/postgres/logfile
2019-03-31 15:24:09.505 MSK [9775] LOG:  listening on IPv4
address "0.0.0.0", port 5432
2019-03-31 15:24:09.505 MSK [9775] LOG:  listening on IPv6
address "::", port 5432
2019-03-31 15:24:09.507 MSK [9775] LOG:  listening on Unix
socket "/tmp/.s.PGSQL.5432"
2019-03-31 15:24:09.526 MSK [9776] LOG:  database system was
shut down at 2019-03-31 15:24:09 MSK
2019-03-31 15:24:09.529 MSK [9775] LOG:  database system is
ready to accept connections
```

Остановка в режиме immediate

```
postgres$ rm /home/postgres/logfile
postgres$ pg_ctl -w -D /usr/local/pgsql/data stop -m immediate
waiting for server to shut down.... done
server stopped
postgres$ pg_ctl -w -l /home/postgres/logfile -D
/usr/local/pgsql/data start
waiting for server to start.... done
server started
```

Журнал сообщений сервера:

```
postgres$ cat /home/postgres/logfile
2019-03-31 15:24:09.843 MSK [9834] LOG:  listening on IPv4
address "0.0.0.0", port 5432
2019-03-31 15:24:09.843 MSK [9834] LOG:  listening on IPv6
address "::", port 5432
```

```
2019-03-31 15:24:09.846 MSK [9834] LOG:  listening on Unix
socket "/tmp/.s.PGSQL.5432"
2019-03-31 15:24:09.865 MSK [9835] LOG:  database system was
interrupted; last known up at 2019-03-31 15:24:09 MSK
2019-03-31 15:24:10.118 MSK [9835] LOG:  database system was not
properly shut down; automatic recovery in progress
2019-03-31 15:24:10.120 MSK [9835] LOG:  invalid record length
at 0/17AD8BC: wanted 24, got 0
2019-03-31 15:24:10.120 MSK [9835] LOG:  redo is not required
2019-03-31 15:24:10.129 MSK [9834] LOG:  database system is
ready to accept connections
```

Перед тем, как начать принимать соединения, СУБД выполнила восстановление (automatic recovery in progress).

Практика 3

Создадим небольшую таблицу:

```
=> CREATE DATABASE wal_log;
CREATE DATABASE
=> \c wal_log
You are now connected to database "wal_log" as user "postgres".
=> CREATE TABLE t(id integer);
CREATE TABLE
=> INSERT INTO t VALUES (1);
INSERT 0 1
```

Мы будем заглядывать в заголовок табличной страницы. Для этого понадобится расширение:

```
=> CREATE EXTENSION pageinspect;
CREATE EXTENSION
```

Начнем транзакцию.

```
=> BEGIN;
BEGIN
Текущая позиция в журнале:

=> SELECT pg_current_wal_insert_lsn();
 pg_current_wal_insert_lsn
```

```
-------------------------
 0/1C349E8
(1 row)
```

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

Эта позиция соответствует файлу:

```
=> SELECT pg_walfile_name('0/1C349E8');
     pg_walfile_name
-------------------------
 000000010000000000000001
(1 row)
```

Имя файла состоит из трех чисел. Первое - номер ветви времени (используется при восстановлении из архива), а два следующих - старшие разряды LSN.

Все журнальные файлы находятся в каталоге /usr/local/pgsql/data/pg_wal/, а в PostgreSQL 10 их также можно увидеть специальной функцией:

```
=> SELECT * FROM pg_ls_waldir() LIMIT 10;
           name           |   size   |      modification
--------------------------+----------+------------------------
 000000010000000000000001 | 16777216 | 2019-08-12 17:15:46+03
(1 row)
```

Изменим строку в таблице:

```
=> UPDATE t SET id = id + 1;
UPDATE 1
```

Позиция в журнале изменилась:

```
=> SELECT pg_current_wal_insert_lsn();
 pg_current_wal_insert_lsn
---------------------------
 0/1C34A30
(1 row)
```

Этот же номер LSN (или меньший, если в журнал попали дополнительные записи) мы найдем и в заголовке измененной страницы:

```
=> SELECT lsn FROM page_header(get_raw_page('t',0));
    lsn
-----------
 0/1C34A30
(1 row)
```

Завершим транзакцию.

```
=> COMMIT;
COMMIT
```

Позиция в журнале снова изменилась:

```
=> SELECT pg_current_wal_insert_lsn();
 pg_current_wal_insert_lsn
---------------------------
 0/1C34A54
(1 row)
```

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

```
=> SELECT '0/1C34A54'::pg_lsn - '0/1C349E8'::pg_lsn;
 ?column?
----------
      108
(1 row)
```

Безусловно, в журнал попадает информация обо всех действиях во всем кластере, но в данном случае мы рассчитываем на то, что в системе ничего не происходит.

Теперь воспользуемся утилитой pg_waldump, чтобы посмотреть содержимое журнала.

Список менеджеров ресурсов:

```
student$ pg_waldump -r list
XLOG
Transaction
Storage
CLOG
Database
Tablespace
```

```
MultiXact
RelMap
Standby
Heap2
Heap
Btree
Hash
Gin
Gist
Sequence
SPGist
BRIN
CommitTs
ReplicationOrigin
Generic
LogicalMessage
```

Утилита может работать и с диапазоном LSN (как в этом примере), и выбрать записи для указанной транзакции. Запускать ее следует от имени пользователя ОС postgres, так как ей требуется доступ к журнальным файлам на диске.

```
postgres$ pg_waldump -p /usr/local/pgsql/data/pg_wal -s
0/1C349E8 -e 0/1C34A54 000000010000000000000001
rmgr: Heap        len (rec/tot):     69/    69, tx:        670, lsn:
0/01C349E8, prev 0/01C344B0, desc: HOT_UPDATE off 1 xmax 670 ; new off 2 xmax
0, blkref #0: rel 1663/16625/16626 blk 0
rmgr: Transaction len (rec/tot):     34/    34, tx:        670, lsn:
0/01C34A30, prev 0/01C349E8, desc: COMMIT 2019-08-12 17:15:46.557989 MSK
```

Мы видим заголовки журнальных записей:

- операция HOT_UPDATE, относящаяся к странице, которую мы смотрели (rel+blk),
- операция COMMIT с указанием времени.

Задание на практическую работу:

1. Средствами операционной системы найдите процессы, отвечающие за работу буферного кэша и журнала WAL.

2. Остановите PostgreSQL в режиме fast; снова запустите его.

Просмотрите журнал сообщений сервера.

3. Теперь остановите в режиме immediate и снова запустите.

Просмотрите журнал сообщений сервера и сравните с предыдущим пунктом.

Примечание:

Если PostgreSQL собран из исходных кодов, остановка в режиме fast выполняется командой

```
pg_ctl stop
```

Если PostgreSQL установлен из пакета, используйте команду для остановки сервера базы данных

```
pg_ctlcluster 10 main stop
```

При этом сервер обрывает все открытые соединения и перед выключением выполняет контрольную точку, чтобы на диск записались согласованные данные. Таким образом, выключение может выполняться относительно долго, но при запуске сервер сразу же будет готов к работе.

Если PostgreSQL собран из исходных кодов, останов в режиме immediate выполняется командой

```
pg_ctl stop -m immediate
```

Если PostgreSQL установлен из пакета, используйте команду

```
pg_ctlcluster 10 main stop -m immediate --skip-systemctl-
redirect
```

При этом сервер также обрывает открытые соединения, но не выполняет контрольную точку. На диске остаются несогласованные данные, как после сбоя. Таким образом, выключение происходит быстро, но при запуске сервер должен будет восстановить согласованность данных с помощью журнала.