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

Практическая работа №5

Транзакции. Восстановление базы данных.

Теория для понимания практики:

Надежность баз данных

Восстановление после отказов

Одна из основных функций СУБД — обеспечение сохранности данных, при этом данные должны оставаться в корректном состоянии. Попробуем классифицировать все многообразие программных и аппаратных средств (приложений, операционных систем, вычислительных систем, информационных сетей, электропитания и т. д.) и причин, которые могут привести к потере данных (ошибки при разработке и эксплуатации, стихийные и техногенные бедствия, действия злоумышленников и пр.).

С точки зрения поддержки сохранности данных отказы систем можно разделить на следующие категории:

Отказы транзакций и приложений. По каким-либо причинам приложение не может завершить транзакцию, например пропала связь с сервером или невозможно выполнить требования протокола. В этом случае транзакция обрывается и выполняется ее откат с помощью операций обращения (w−1). Не менее важно обеспечить согласованность при наличии разного рода сбоев, приводящих к обрывам транзакций или отказу сервера базы данных.

Для того чтобы реализовать принцип атомарности при обрывах транзакций, т. е. отменить результаты выполнения операций оборванной транзакции, вводится операция обращения записи w−1(x). По определению эта операция восстанавливает значение элемента данных x в то состояние, в котором элемент был непосредственно перед выполнением прямой операции

wi(x).

Расписание, в котором операция записи непосредственно предшествует операции ее обращения, эквивалентно расписанию, из которого данная пара операций исключена. Для того чтобы корректно выполнить обрыв транзакции, необходимо обратить все операции записи, выполненные этой транзакцией, в обратном порядке, и затем зафиксировать эту транзакцию. Любые операции чтения при этом можно игнорировать. Такая реализация операции обрыва называется откатом (rollback). Зачастую операции обрыва и отката не различают. Так, в языке SQL обрыв транзакции выполняется оператором ROLLBACK.

Напомним, что операция w−1 является чисто логической, ее реализации в разных системах могут отличаться. В частности, в PostgreSQL выполнение отката не требует записи данных, потому что новые значения всегда записываются на новое место, а старые, следовательно, сохраняются.

Отказы сервера. Сервер базы данных или вся вычислительная система утратили возможность выполнять запросы приложений, но содержимое базы данных, размещенное на энергонезависимых носителях (вращающихся дисках, SSD и т. п.), сохранилось. В этом случае выполняется рестарт сервера базы данных, возможно, после рестарта операционной системы. Во время рестарта сервера необходимо привести базу данных в согласованное состояние, в котором все изменения, сделанные зафиксированными транзакциями, сохранятся, а незавершенные к моменту отказа сервера транзакции будут оборваны и для сделанных ими изменений будет выполнен откат.

Разрушение носителей. Для того чтобы предотвратить потерю данных при разрушении носителей, необходимо регулярно создавать резервные копии. Восстановление после разрушения носителя сводится к восстановлению базы данных с резервной копии, возможно, с последующим восстановлением согласованности, как при отказе системы.

Меры по обеспечению надежности, необходимые для каждой базы

данных, зависят от требования к прикладной системе. Чем выше уровень требований, тем более сложные (и дорогостоящие) необходимы решения. Любая конфигурация системы управления базами данных обеспечивает корректность откатов транзакций и восстановление после отказов сервера, однако выбор мер защиты от разрушения носителя может варьироваться в очень широком диапазоне.

Отказы сервера баз данных

Для того чтобы обеспечить обработку отказов транзакций и отказов сервера, система управления базами данных регистрирует все изменения, выполняемые транзакциями, в журнале транзакций.

Записи заносятся в журнал строго последовательно, т. е. новые записи всегда добавляются в конец журнала. Каждая запись снабжается уникальным идентификатором LSN (log sequence number), который однозначно ее идентифицирует. В системе PostgreSQL для этого используется тип pg_lsn, представляющий собой 8-байтовое число, указывающее смещение записи от начала журнала. В других системах может использоваться другой формат LSN, например старшие 4 байта могут указывать номер файла журнала, а младшие — смещение в этом файле. Важным требованием к LSN, которое необходимо для корректной работы алгоритма восстановления и поэтому соблюдается во всех системах, является строгая упорядоченность: записи журнала, созданные позже, должны иметь большие значения LSN, чтобы, сравнивая номера LSN двух записей, всегда можно было установить, какая из них появилась раньше.

Переключение на новый файл журнала происходит, когда размер журнала превышает определенный порог, при создании резервной копии базы данных или по другим причинам, в том числе по указанию администратора базы данных.

Как и для любых других типов файлов, несколько последних записей журнала могут находиться только в оперативной памяти, пока не произойдет их перенос на энергонезависимый носитель информации. Далее будем

условно называть такой носитель диском. Заметим, что при нормальной работе системы записи журнала не используются, поэтому нет необходимости сохранять в оперативной памяти записи, которые уже перенесены (вытолкнуты) на диск.

Ведение журнала транзакций подчиняется следующим правилам опережающей записи (write-ahead logging, WAL):

WAL1 – записи, регистрирующие любые изменения в базе данных, должны быть занесены в журнал и вытолкнуты на диск, до того, как сами эти изменения (в базе данных) попадут на диск;

WAL2 – информация о фиксации транзакций в журнале должна быть вытолкнута на диск раньше, чем завершится операция фиксации, и раньше, чем приложение получит информацию о том, что фиксация выполнена успешно.

В журнал могут заноситься записи нескольких разных типов, в том числе:

•BEGIN отмечает начало транзакции (в PostgreSQL такая запись не применяется);

• COMMIT регистрирует фиксацию транзакции;

• ROLLBACK отмечает откат транзакции (такая запись необходима в системе PostgreSQL, но может быть ненужной в других системах, в которых применяются явные операции отката w−1);

• UNDO содержит информацию о том, как выполнить откат операции модификации базы данных (не применяется в PostgreSQL);

• REDO содержит информацию, достаточную для повторного выполнения операции модификации данных;

• CHECKPOINT содержит дополнительную информацию для восстановления базы данных при рестарте.

Отказы сервера баз данных

Существует еще несколько типов записей журнала, например в некоторых СУБД в журнале отмечается создание резервных копий базы

данных (для обеспечения восстановления носителя данных). В системе PostgreSQL в журнале регистрируется создание файлов и каталогов, а также другие вспомогательные действия.

Записи о модификации данных могут быть логическими, связанными с выполняемыми операциями SQL (INSERT, UPDATE, DELETE), или физическими, отражающими состояние измененных страниц базы данных. В любом случае для каждой операции модификации могут быть созданы две записи: UNDO и REDO. Это дает возможность как устранить результаты выполнения операции, так и выполнить ее повторно.

Существует несколько стратегий записи в журнал, гарантирующих возможность восстановления согласованного состояния базы данных. Эти стратегии обеспечивают выполнение требований атомарности и долговечности транзакций: завершенные транзакции не могут быть потеряны, а оборванные (или не завершенные до отказа) транзакции не должны оставлять изменений в базе данных. Для выполнения этих требований, необходимо чтобы информация об изменениях, которые еще не записаны на диск базы данных, обязательно сохранялась в журнале и при этом попадала на энергонезависимый носитель данных.

Выбор стратегии ведения журнала определяется следующими свойствами:

FORCE/NOFORCE. Свойство FORCE означает, что все изменения, выполненные транзакцией, заносятся в базу данных и выталкиваются на диск, до того, как завершается выполнение операции COMMIT. В этом случае нет необходимости в записях REDO.

Свойство NO FORCE, наоборот, предполагает, что записи REDO выталкиваются из журнала на диск до завершения фиксации, но не требуют записи изменений в базу данных.

STEAL / NO STEAL. Свойство STEAL означает, что страницы (блоки) базы данных, содержащие изменения незафиксированных транзакций, могут выталкиваться на диск. В этом случае необходимы журнальные записи UNDO,

которые должны выталкиваться на диск раньше, чем страницы базы данных, содержащие эти изменения.

Свойство NO STEAL запрещает выталкивание на диск страниц, содержащих изменения незавершенных транзакций. В этому случае записи UNDO не требуются, потому что на диск попадают только изменения зафиксированных транзакций.

Комбинация этих свойств дает четыре возможных стратегии ведения журнала.

FORCE + NO STEAL. Эта стратегия требует, чтобы все изменения записывались в базу данных в момент фиксации. Для ее реализации используется метод теневых страниц, состоящий в том, что измененные транзакцией страницы записываются на новое место, а при фиксации происходит обновление только одной страницы, содержащей указатели на актуальные версии данных. Для восстановления не требуются записи ни REDO, ни UNDO, поэтому при восстановлении после системных отказов журнал не нужен. Эта стратегия оказывается менее эффективной при нормальной работе (т. е. создает большую дополнительную нагрузку, чем остальные) и в высокопроизводительных системах не применяется.

FORCE + STEAL. Поскольку все изменения должны быть записаны на диск базы данных до фиксации, могут возникать задержки при фиксации транзакций, выполнивших относительно большое количество изменений. Для этой стратегии не требуются записи REDO.

NO FORCE + NO STEAL. Изменения записываются на диск базы данных только после фиксации транзакций. При этом некоторые страницы могут слишком долго оставаться в оперативной памяти (например, страницы, часто обновляемые различными транзакциями). В этом случае записи UNDO не требуются.

NO FORCE + STEAL. Эта стратегия дает возможности для полностью асинхронной записи изменений базы данных на диск, никак не связанной с фиксацией транзакций. Обычно именно эта стратегия используется

высокопроизводительными системами, в том числе PostgreSQL.

Занесение в журнал двух записей на каждое изменение может создавать значительную дополнительную нагрузку на сервер, однако обычно эта нагрузка не ведет к снижению производительности, потому что последовательная запись в журнал почти для всех носителей данных выполняется быстрее, чем изменение в произвольном порядке (для вращающихся дисков она может быть быстрее на два порядка). Кроме этого, при наличии журнала запись изменений на диски базы данных выполняется отдельным фоновым процессом, т. е. не замедляет выполнение транзакций. Благодаря опережающей записи в журнал информация об изменениях не будет потеряна, даже если возникнет необходимость в рестарте сервера баз данных.

Отказы сервера баз данных

Рестарт сервера

После рестарта сервера база данных может оказаться в несогласованном состоянии. Во-первых, активные транзакции, которые не успели зафиксироваться, до того, как произошел отказ системы, должны быть оборваны, и поэтому необходимо выполнить откат тех изменений, которые уже были занесены в базу данных. Во-вторых, изменения транзакций, которые были зафиксированы, могли не попасть в базу данных (остаться только в оперативной памяти и в журнале). Для того чтобы привести базу данных в согласованное состояние, при рестарте запускается алгоритм восстановления.

Известно несколько различных алгоритмов восстановления. Кратко опишем алгоритм «Redo history», выполнение которого включает две фазы.

Анализ и повторное выполнение. На фазе анализа выполняется анализ журнала и повторно выполняются изменения операций всех транзакций, которые еще не были занесены в базу данных. Для этого выполняется просмотр журнала в прямом направлении (от начала к концу) и выполняются следующие действия:

• При обнаружении записи BEGIN транзакция записывается в список активных транзакций. Если записи BEGIN не используются, то началом

транзакции является первая операция, помеченная идентификатором транзакции, который не встречался ранее.

• Появление записи COMMIT приводит к исключению транзакции из списка активных.

• Записи REDO используются для повторения операций, если соответствующие изменения еще не внесены в базу данных. Для того чтобы определить, какие изменения занесены на страницу, на каждой странице имеется поле PSN (page sequence number), содержащее LSN последней записи журнала, изменения которой уже есть на странице. Если LSN > PSN, то изменения заносятся на страницу и изменяется ее PSN.

• Если изменение относится к активной транзакции, то изменяемый объект заносится в список объектов, измененных этой транзакцией. Этот список объектов не обязателен, но его наличие позволяет ускорить выполнение фазы отката (например, загрузить все необходимые для отката страницы в память перед выполнением этой фазы).

После завершения просмотра журнала все изменения, выполненные до отказа системы и зарегистрированные в журнале, будут занесены в базу данных.

Откат. На фазе отката выполняется просмотр журнала в обратном направлении (от конца к началу), и для всех операций из списка активных транзакций, полученного на первой фазе, выполняется операция отката с помощью записи UNDO. Операция отката регистрируется в журнале, т. е. для нее заносятся записи REDO или UNDO (если такие записи необходимы для восстановления в соответствии со стратегией ведения журнала, принятой в системе).

Несмотря на то что в системе PostgreSQL не используются записи UNDO, этот шаг все равно выполним, поскольку необходимые значения имеются в базе данных.

При обнаружении записи BEGIN для активной транзакции эта транзакция исключается из списка активных и выполняется ее фиксация, т. е.

в журнал заносится запись COMMIT, и журнал выталкивается на диск.

Работа алгоритма заканчивается, когда список активных транзакций становится пустым.

Конечно, фаза отката не нужна в системах, использующих стратегию NO STEAL, потому что в таких системах результаты выполнения транзакций не могут попасть в устойчивую память до фиксации транзакции.

Одним из требований к алгоритмам восстановления после рестарта является возможность многократного выполнения (в случае отказа системы во время процедуры восстановления). Для того чтобы выполнить это требование, необходимо гарантировать, что повторное выполнение операции REDO не разрушает содержимое страницы.

Существует два вида записей REDO:

• Запись содержит полный образ страницы.

В этом случае повторное выполнение операции записи дает такой же результат, как однократное.

• Запись содержит только информацию об изменениях.

Однократность применения таких записей обеспечивается проверкой значения поля PSN на странице.

Отказы сервера баз данных

Основным критерием качества алгоритмов восстановления является время, необходимое для возобновления нормальной работы СУБД.

Для того чтобы сократить время недоступности, можно использовать список страниц, обновленных незавершенными активными транзакциями, построенный на первой фазе работы алгоритма восстановления. После окончания первой фазы доступ к страницам из этого списка (или к модифицированным элементам данных) блокируется. Это дает возможность начать обработку новых транзакций сразу после окончания первой фазы. По мере выполнения отката незавершенных транзакций блокировки снимаются, открывая доступ к восстановленным состояниям элементов данных.

Контрольные точки

Механизм контрольных точек предназначен для сокращения объема журнала, который просматривается при восстановлении. Журнальная запись контрольной точки CHECKPOINT содержит список активных транзакций и список страниц, состояние которых в оперативной памяти отличается от состояния на постоянном носителе. После выталкивания записи о контрольной точки фоновый процесс записи копирует все изменения из этого списка на диск. При этом нормальная работа системы продолжается. Если при этом страницы, включенные в список, будут изменены новыми транзакциями, эти изменения попадут на диск. Изменения на страницах, не включенных в список, будут учтены в следующей контрольной точке.

Включение списка активных транзакций, вообще говоря, не обязательно, но позволяет упростить работу алгоритма восстановления на фазе анализа: список, содержащийся в записи журнала, используется в качестве начального значения для списка активных транзакций.

Когда копирование страниц, включенных в контрольную точку, заканчивается, в журнал заносится запись о завершении контрольной точки. После этого (сразу или через некоторое время) может быть создана новая контрольная точка и работа системы продолжается. Наличие записей контрольной точки позволяет при рестарте сервера на фазе повторного выполнения начать просмотр журнала не с самого начала, а с предпоследней контрольной точки. При этом начальное состояние списков, которые строятся на первой фазе, считывается из записи о контрольной точке.

Заметим, что запись начала контрольной точки избыточна и нужна только для упрощения процесса анализа при восстановлении. В системе PostgreSQL начало контрольных точек в журнале не отмечается.

В системе PostgreSQL первая после контрольной точки запись REDO, относящаяся к любой странице, содержит полный образ этой страницы (при установленном параметре конфигурации full_page_writes). Это дает возможность корректно восстанавливать содержимое страниц, которые могли быть повреждены при отказе системы. Такие повреждения возможны, потому

что в реальности операция записи страницы на диск не атомарна.

Необходимо обратить внимание на то, что запись изменений, внесенных транзакциями, выполняется абсолютно асинхронно и никак не связана с фиксацией транзакций. Вследствие этого можно считать, что операции изменения записей, ранее прочитанных приложением и, скорее всего, оставшихся в кеше, не требуют времени больше, чем необходимо для выполнения этих изменений в оперативной памяти, в отличие от операций чтения, которые могут ожидать считывания необходимых данных с диска. Поведение, которое при этом наблюдают приложения, может показаться парадоксальным: операции записи выполняются на порядки быстрее, чем операции чтения. Конечно, операции записи требуют определенных вычислительных ресурсов и дают свой вклад в общую нагрузку системы, однако эти операции выполняются асинхронно и, как правило, не оказывают существенного влияния на обработку запросов.

Разрушение носителя

Если по каким-либо причинам файлы базы данных или журналы не могут быть прочитаны, то восстановление после рестарта невозможно. Такая ситуация рассматривается как разрушение носителя (не имеет значения, разрушен ли сам носитель или только логическая структура данных).

Для того чтобы предотвратить потерю данных в случае разрушения носителя, необходимо периодическое создание резервных копий. Существует большое разнообразие методов создания резервных копий, различающихся по сложности, полноте восстановления и по стоимости.

Простые методы создания копий обеспечивают восстановление базы данных в состояние, которое было на момент создания резервной копии. При этом все изменения, выполненные после создания копии до отказа, будут потеряны.

Более сложные методы предполагают (в дополнение к копированию самой базы данных) хранение журнала транзакций. Для приведения системы в рабочее состояние после разрушения носителя необходимо восстановить

базу данных с последней резервной копии и затем выполнить рестарт системы с использованием всех файлов журнала, записанных после создания этой резервной копии (а не с контрольной точки, как при отказе сервера).

Заметим, что для реализации этого процесса необходимы все файлы журнала вплоть до момента отказа с разрушением носителя, в том числе записанные после создания резервной копии. Для того чтобы такие файлы журнала были доступны, в системах, где необходима высокая надежность, при нормальной работе системы записывается несколько идентичных копий журнала на разные носители. При этом процесс восстановления может занимать значительное время.

Наиболее сложные и дорогостоящие методы предполагают поддержку дополнительных серверов (реплик) баз данных, готовых или почти готовых к работе. Такие схемы обеспечивают быстрое восстановление при любых отказах, но создают значительную дополнительную нагрузку во время нормальной работы системы.

Основными метриками, характеризующими качество процедур восстановления, являются следующие:

Время восстановления. Измеряется от начала процедуры восстановления до начала обработки новых транзакций. Время восстановления может варьироваться от нескольких часов для простых методов до долей секунды для сложных и непосредственно влияет на характеристику доступности системы.

Выживаемость. Обозначает количество разрушений носителя, которое приводит к потере данных. Эта характеристика не обязательно совпадает с количеством копий, т. к. процедура восстановления может включать создание новой резервной копии, что делает ее более дорогостоящей и более продолжительной, но обеспечивает более высокую выживаемость.

Задержка. Время, необходимое для распространения изменений по резервным копиям, обеспечивающим выживаемость. Для простых методов, не использующих журнал, задержка равна интервалу между созданиями

резервных копий. В высоконадежных системах задержка измеряется долями секунды.

Методы создания резервных копий и восстановления с них выбираются в зависимости от требований к системе. Важно учитывать, что сложность и стоимость реализации и сопровождения очень существенно зависит от этих требований.

Экспорт и импорт

Наиболее простой способ создания резервных копий — экспорт логической структуры базы данных в какой-либо внешний формат. В системе PostgreSQL такой экспорт можно выполнить с помощью утилиты pg_dump. К достоинствам этого метода можно отнести простоту, возможность частичного восстановления базы данных, а также возможность восстановления в другой конфигурации сервера или даже в другой СУБД (в последних двух случаях решаются другие задачи, а не задача восстановления после разрушения носителя).

Размещение данных после восстановления экспортированной базы данных, скорее всего, не будет совпадать с размещением в исходной БД, поэтому восстановление актуального состояния по журналу в этом случае невозможно. Это означает, что база данных восстанавливается в то состояние, в котором она была в момент начала копирования, и, следовательно, задержка может достигать значений, равных интервалу времени между созданиями резервных копий.

Во многих случаях такие значения задержки допустимы — например, для баз данных, используемых только для разработки или тестирования приложений, когда система не находится в производственной эксплуатации или приложение является тиражируемым продуктом. В подобных случаях нагрузка на сервер базы данных относительно невелика, вероятность разрушения носителя достаточно мала, и ценность данных и особенно изменений, сделанных после создания копии, незначительна. Применение более сложных стратегий резервирования и восстановления имеет смысл

только для отработки и проверки самих процедур резервирования и восстановления.

Копирование с восстановлением по журналам

Если допустимые значения задержки не должны превышать долей секунды, то методы на основе экспорта и импорта оказываются непригодными. Традиционная стратегия восстановления состоит в том, что в качестве резервной копии создается точный образ базы данных, зачастую в формате, непригодном для непосредственного запуска сервера баз данных (возможно, в сжатом виде).

Кроме этого, необходимо сохранять все данные, записываемые при нормальной работе системы в журнал транзакций.

В системе PostgreSQL для записи журнала используется несколько файлов, называемых сегментами журнала. При переключении на новый сегмент самый старый из ранее заполненных сегментов уничтожается, как только будет записана контрольная точка, после которой данные из этого сегмента станут ненужными для восстановления после отказа системы.

Для того чтобы использовать резервные копии, необходимо выполнить процедуру архивирования сегментов журнала, до того как они будут удалены. В системе PostgreSQL это делается с помощью параметров конфигурации и задания команды операционной системы, которая будет выполнять архивирование (это может быть просто копирование на другой носитель).

Для того чтобы восстановление было возможно, необходимо периодически создавать резервную копию базы данных, а в промежутках между созданиями копии архивировать все сегменты журнала, порождаемые в результате нормальной работы системы.

Существует два метода создания резервной базы данных:

• Применение программ создания резервных копий, входящих в состав СУБД. Для создания такой копии не требуется останавливать нормальную работу сервера базы данных.

• Копирование файлов базы данных (включая журнал транзакций)

средствами операционной системы. Такое копирование может в некоторых системах работать быстрее, но для некоторых СУБД может требоваться остановка сервера баз данных, что, конечно, влияет на доступность системы.

В системе PostgreSQL такие копии кластера баз данных создаются с помощью утилиты pg_basebackup. Утилита работает как обычный клиент сервера баз данных (в некоторых случаях она устанавливает не одно, а два соединения с сервером). Поэтому во время создания копий нормальная работа сервера баз данных не останавливается, хотя копирование создает дополнительную нагрузку на сервер.

Вместо утилиты pg_basebackup можно использовать вызовы системных функций (например, через оператор SELECT). Это дает возможность более гибкого управления деталями создания резервной копии.

Если копия кластера баз данных записывается в архивный файл, то для использования такой копии необходимо восстановить ее на носителе и затем выполнить процедуру рестарта сервера. В результате база данных будет восстановлена в состояние, в котором она была во время создания резервной копии. Далее необходимо повторно выполнить изменения, зарегистрированные в сегментах журнала, накопленных после создания этой копии, как архивированных, так и текущих.

Периодичность создания резервной копии зависит от того, сколько времени допустимо потратить на восстановление. Более частое создание резервных копий сокращает количество сегментов журнала, которые необходимо применить, но увеличивает нагрузку на сервер при нормальной работе системы.

Для обеспечения большей выживаемости можно создавать несколько копий параллельно, однако более эффективным методом считается копирование резервной копии средствами операционной системы после того, как она создана, что может повысить эффективность использования оборудования.

Поскольку процедура восстановления предусматривает повторное

внесение изменений по журналу в том порядке, в котором эти изменения выполнялись, возможно восстановление не только последнего согласованного состояния баз данных до момента отказа, но и состояния на любой предшествующий момент времени (после завершения записи резервной копии). Такая возможность полезна в том случае, когда необходимость восстановления вызвана, например, некорректностью работы приложений или ошибками персонала.

В системе PostgreSQL предусмотрено завершение процесса восстановления по одному из следующих критериев:

• достигнуто согласованное состояние базы данных;

• достигнут явно указанный момент времени;

• выполнена транзакция с указанным идентификатором;

• достигнута заранее созданная именованная точка восстановления.

Создание резервной копии сопровождается выполнением контрольной точки и переключением файлов (сегментов) журнала. Для корректного восстановления базы данных из резервной копии необходима информация о двух позициях в журнале:

1) перед началом резервного копирования — с этой позиции журнал просматривается при восстановлении;

2) после окончания резервного копирования — достижение этой позиции гарантирует восстановление того состояния базы данных, в котором она была на момент завершения создания резервной копии.

В некоторых системах указанные позиции отмечаются специальными записями непосредственно в журнале. В системе PostgreSQL они записываются в отдельный файл, также включаемый в состав резервной копии.

Резервные серверы баз данных

Недостатком стратегии копирования и восстановления на основе копий в последовательных файлах является большое время восстановления (несколько часов для больших баз данных), а достоинством — относительно

небольшое количество ресурсов, необходимых для хранения резервных копий в сжатом виде. Очевидно, что большое время восстановления отрицательно влияет на характеристику доступности, которая считается крайне важной для некоторых классов приложений.

Для того чтобы исключить время копирования всей базы данных, резервные копии создаются в обычном формате базы данных и на этой копии запускается резервный сервер баз данных. При этом возможны различные варианты распространения на резервные серверы изменений, сделанных на основном сервере.

Применение резервных серверов позволяет сократить время восстановления до десятков или даже единиц секунд, что обеспечивает очень высокие значения характеристики доступности, однако, чем меньше требуемое время восстановления, тем большая дополнительная нагрузка ложится на систему при ее нормальной работе.

Для применения любой схемы защиты от разрушений носителя с запасным сервером требуется не менее чем вдвое большая конфигурация оборудования (как минимум два сервера, каждый из которых обладает производительностью, достаточной для выполнения запросов прикладной системы в нормальном режиме работы). Имеется возможность несколько уменьшить избыточность оборудования, используя предусмотренную в СУБД PostgreSQL и других высокопроизводительных системах возможность выполнения на запасном сервере запросов на чтение (но не на модификацию) данных. Запасные серверы часто используются для извлечения данных с целью загрузки в хранилище данных (data warehouse) или для получения аналитических отчетов непосредственно из базы данных. В том и другом случае обычно небольшое отставание состояния базы данных от актуального не имеет значения.

Итоги

Рассмотрены методы и алгоритмы обеспечения надежности хранения данных. Результативность этих методов выбирается в зависимости от

требований к информационной системе, использующей базу данных. Более сложные (и одновременно более дорогостоящие в эксплуатации) методы позволяют получить очень высокие значения характеристик надежности и доступности.

Задание на практическую работу:

Создайте резервную копию базы данных утилитой pg_dump, создайте новую базу данных и выполните восстановление резервной копии в данной базе данных. Выполните несколько обновляющих транзакций. Уничтожьте базу данных и восстановите ее содержимое, используя резервную копию. Объясните результаты.