Базы данных

Лекция 07 – Berkeley DB. Configuration.

Преподаватель: Поденок Леонид Петрович, 505a-5

+375 17 293 8039 (505a-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2024.11.11

Оглавление

Настройка базы данных

Есть некоторые проблемы настройки базы данных и кэша, которые необходимо учитывать при создании БД. В большинстве случаев для управления базами данных нужно сделать очень мало. Однако есть проблемы, которые необходимо принимать во внимание, и они в значительной степени зависят от метода доступа, который выбран для БД.

Примеры и описания ниже в основном касаются метода доступа Btree. Это связано с тем, что большинство приложений БД используют BTree.

Установка размера страницы[1]

Внутри БД хранит записи базы данных на страницах. Размеры страниц важны, поскольку они могут влиять на производительность приложения.

Размер страниц БД может быть выбран от 512 байт до 64 Кбайт.

Выбранный размер должен быть степенью числа 2.

Размер страницы базы данных устарнавливается с помощью `DB->set_pagesize( )`.

Размер страницы базы данных можно выбрать только во время создания базы данных.

При выборе размера страницы следует учитывать следующее:

- страницы переполнения;
- блокировки;
- дисковый ввод-вывод.

1) Setting the Page Size

Страницы переполнения[2]

Страницы переполнения используются для хранения ключа или элемента данных, которые не могут поместиться на одной странице.

Чтобы создать страницы переполнения, делать ничего не нужно – при сохранении данных, которые слишком велики для размера страницы БД, они будут созданы автоматически.

Единственный способ предотвратить создание страниц переполнения — убедиться, что выбран размер страницы, достаточно большой для хранения записей БД.

Поскольку страницы переполнения существуют за пределами обычной структуры базы данных, их использование дорого с точки зрения производительности. Если будет выбран слишком маленький размер страницы, база данных будет вынуждена использовать чрезмерное количество страниц переполнения. Это существенно повредит производительности приложения.

По этой причине нужно выбирать размер страницы, который по крайней мере будет достаточно большим для хранения нескольких записей, учитывая ожидаемый их средний размер.

В случае BTree для достижения наилучших результатов следует выберать такой размер страницы, который может содержать не менее 4 таких записей.

Узнать, сколько страниц переполнения использует база данных, можно с помощью метода `DB->stat()` или проверить базу данных с помощью утилиты командной строки db_stat.

---

2) Overflow Pages

Блокировка

Блокировка и многопоточный доступ к базам данных БД встроены в библиотеку.

Однако для включения подсистемы блокировки и для обеспечения эффективного совместного использования кэша между базами данных необходимо использовать окружение.

Если приложение многопоточное или если к базам данных обращаются более одного процесса одновременно, то размер страницы может влиять на производительность приложения. Причина в том, что для большинства методов доступа (исключением является очередь) БД реализует блокировку на уровне страницы. Это означает, что самая тонкая гранулярность блокировки находится на странице, а не на записи.

В большинстве случаев страницы базы данных содержат несколько записей базы данных. При этом, когда записи на этих страницах читаются или записываются, БД, чтобы обеспечить безопасный доступ нескольким потокам или процессам, выполняет блокировку страниц.

По мере того, как размер страницы увеличивается относительно размера записей базы данных, количество записей, которые хранятся на любой данной странице, также увеличивается. В результате увеличивается вероятность того, что два или более читателей и/или писателей захотят получить доступ к записям на одной странице.

Когда два или более потоков и/или процессов хотят управлять данными на странице, возникает конфликт блокировок. Конфликт блокировок разрешается ожиданием потока (или процесса), пока другой поток откажется от своей блокировки. Именно это ожидание и вредит производительности приложения.

Можно выбрать размер страницы, который настолько велик, что приложение будет тратить чрезмерное и заметное количество времени на разрешение конфликта блокировок.

ВАЖНО!!! — вышеприведенный сценарий особенно вероятен по мере увеличения количества параллелизма, встроенного в приложение.

С другой стороны, если выбрать слишком маленький размер страницы, то это только сделает дерево глубже, что также может привести к снижению производительности.

Таким образом, смысл заключается в том, чтобы выбрать разумный размер страницы (такой, который будет содержать значительное количество записей), а затем уменьшить размер страницы, если будет иметь место конфликт блокировок.

Можно проверить количество конфликтов блокировок и взаимоблокировок, возникающих в приложении, изучив статистику блокировок в окружении базы данных.

Можно использовать метод `DB_ENV->lock_stat()`, либо утилиту командной строки db_stat.

Количество недоступных блокировок, которых ожидало приложение, хранится в поле st_lock_wait статистики блокировок.

Эффективность ввода-вывода

Размер страницы может влиять на эффективность перемещения данных БД на диск и с диска. Для некоторых приложений, особенно тех, для которых кэш в памяти не может быть достаточно большим для хранения всего рабочего набора данных, эффективность ввода-вывода может существенно влиять на производительность приложения.

Большинство операционных систем для определения того, сколько данных перемещать на диск и с диска для одной операции ввода-вывода, используют внутренний размер блока.

Этот размер блока обычно равен размеру блока файловой системы.

Для оптимальной эффективности дискового ввода-вывода следует выбрать размер страницы базы данных, равный размеру блока ввода-вывода операционной системы.

БД выполняет передачу данных на основе размера страницы базы данных. То есть она перемещает данные на диск и с диска по одной странице за раз.

По этой причине, если размер страницы не соответствует размеру блока ввода-вывода, операционная система может внести неэффективность в то, как она реагирует на запросы ввода-вывода БД.

Например, предположим, что размер страницы меньше размера блока операционной системы. В этом случае, когда DB записывает страницу на диск, она будет записывать только часть логической страницы файловой системы.

Всякий раз, когда какое-либо приложение записывает только часть логической страницы файловой системы, операционная система все равно читает реальную страницу файловой системы, перезаписывает ту часть страницы, в которую приложение и не писало, после чего записывает страницу файловой системы обратно на диск.

В результате получаем значительно больше дискового ввода-вывода, чем если бы приложение выбрало размер страницы, равный размеру блока базовой файловой системы.

В качестве альтернативы, если будет выбран размер страницы, который больше, чем размер блока базовой файловой системы, то операционной системе, возможно, придется прочитать больше данных, чем необходимо для выполнения запроса на чтение.

Кроме того, в некоторых операционных системах запрос одной страницы базы данных может привести к тому, что операционная система прочтет больше блоков файловой системы, чтобы удовлетворить критериям опережающего чтения.

В этом случае операционная система будет читать значительно больше данных с диска, чем фактически требуется для выполнения запроса на чтение базы данных.

Примечание

Размер страницы, отличный от размера блока файловой системы, может повлиять на транзакционные гарантии. Причина в том, что размеры страниц, превышающие размер блока файловой системы, заставляют базу данных записывать страницы с шагом в размере блока. В результате в результате транзакционной фиксации возможна частичная запись страницы.

Советы по выбору размера страницы

В общем, и без учета других соображений, размер страницы, равный размеру блока файловой системы, является идеальной ситуацией.

Если данные спроектированы таким образом, что 4 записи базы данных не могут поместиться на одной странице (предполагая BTree), то следует увеличить размер страницы, чтобы их вместить.

После того, как пришлось отказаться от соответствия размеру блока файловой системы, общее правило заключается в том, что большие размеры страниц лучше.

Исключением из этого правила является ситуация, когда в приложении имеет место много параллелизма. В этом случае, чем ближе можно сопоставить размер страницы с идеальным размером, необходимым для данных приложения, тем лучше. Это позволит избежать ненужного конфликта за блокировки страниц.

Выбор размера кэша

Размер кэша важен для приложения, поскольку если он установлен на слишком маленькое значение, производительность приложения пострадает от слишком большого количества дисковых операций ввода-вывода.

С другой стороны, если кэш слишком большой, то приложение будет использовать больше памяти, чем ему фактически нужно.

Более того, если приложение использует слишком много памяти, то в большинстве операционных систем это может привести к тому, что ваше приложение будет выгружено из памяти, что приведет к крайне низкой производительности.

Размер кэша выбирается с помощью `DB->set_cachesize()` или `DB_ENV->set_cachesize()`, в зависимости от того, используется или нет окружение базы данных.

Размер кэша должен быть степенью 2, но в остальном он ограничен только доступной памятью и соображениями производительности.

Выбор размера кэша можно изменить в любое время, поэтому его можно легко настроить в соответствии с меняющимися требованиями к данным приложения.

Лучший способ определить, насколько большим должен быть ваш кэш, — это поместить приложение в рабочую среду и посмотреть, сколько дисковых операций ввода-вывода происходит. Если ваше приложение будет загружать на диск довольно много данных для извлечения записей, то следует размер кэша увеличить (при условии, что памяти для этого достаточно).

Чтобы оценить эффективность кэша можно использовать утилиту командной строки `db_stat` с опцией `-m`. В частности, она отображает количество страниц, найденных в кэше, вместе с процентным значением. Чем ближе к 100%, тем лучше.

Если это значение слишком низкое и возникают проблемы с производительностью, то следует рассмотреть возможность увеличения размера кэша, предполагая, что есть память для его поддержки.

Настройка BTree

- Разрешение дублирования записей.
- Установка обратных вызовов функций сравнения (компараторов).

Разрешение записей-дубликатов

Базы данных BTree могут содержать записи-дубликаты. Одна запись считается дубликатом другой, когда обе записи используют ключи, которые сравниваются как равные друг другу.

По умолчанию ключи сравниваются в лексикографическом порядке, при этом более короткие ключи сортируются выше, чем более длинные. Можно переопределить это поведение по умолчанию с помощью метода `DB->set_bt_compare()`.

По умолчанию базы данных DB не допускают дублирующих записей. В результате любая попытка записать запись, которая использует ключ, равный ранее существующей записи, приводит к тому, что *ранее существующая запись перезаписывается новой записью*.

Разрешение дублирующих записей полезно, если есть база данных, содержащая записи, ключом которых является часто встречающаяся часть информации.

Часто необходимо разрешить дублирующие записи для вторичных баз данных.

Например, предположим, что первичная база данных содержит записи, связанные с автомобилями. В этом случае бывает нужно найти все автомобили определенного цвета, поэтому цвет автомобиля индексируется. Однако для любого заданного цвета, вероятно, будет несколько автомобилей. Поскольку индекс является вторичным ключом, это означает, что несколько записей вторичной базы данных будут совместно использовать один и тот же ключ, и поэтому вторичная база данных должна поддерживать дублирующие записи.

Отсортированные дубликаты

Дубликаты записей могут храниться в отсортированном или несортированном порядке.

Можно заставить DB автоматически сортировать ваши дубликаты записей, указав во время создания базы данных флаг `DB_DUPSORT`.

Если поддерживаются отсортированные дубликаты, то для определения местоположения дубликата записи в его наборе дубликатов используется функция сортировки, указанная в `DB->set_dup_compare()`. Если такая функция не указана, то используется лексикографическое сравнение по умолчанию.

Неотсортированные дубликаты

Из соображений производительности BTrees всегда должны содержать отсортированные записи. (BTrees, содержащие неотсортированные записи, потенциально должны тратить гораздо больше времени на поиск записи, чем BTree, содержащий отсортированные записи).

Тем не менее, DB предоставляет поддержку для подавления автоматической сортировки дубликатов записей, поскольку может оказаться, что приложение вставляет записи, которые уже находятся в отсортированном порядке.

То есть, если база данных настроена на поддержку несортированных дубликатов, то предполагается, что приложение будет выполнять сортировку «вручную». В этом случае следует ожидать значительного снижения производительности.

Каждый раз, когда записи помещаются в БД в порядке сортировки, неизвестном BDB, придется платить снижением производительности.

Тем не менее, BDB ведет себя следующим образом при вставке записей в базу данных, которая поддерживает несортированные дубликаты:

- Если приложение просто добавляет дублирующую запись с помощью DB->put(), то запись вставляется в конец ее отсортированного набора дубликатов.

- Если для помещения дублирующейся записи в базу данных используется курсор, то новая запись помещается в набор дубликатов в соответствии с флагами, которые предоставляются в методе DBC->put(). Соответствующие флаги:

DB_AFTER

Данные, предоставленные при вызове DBC->put(), помещаются в базу данных как дублирующая запись. Ключ, используемый для этой операции, — это ключ, используемый для записи, на которую в данный момент ссылается курсор. Поэтому любой ключ, предоставленный при вызове DBC->put(), игнорируется.

Дублирующая запись вставляется в базу данных сразу после текущей позиции курсора в базе данных.

Этот флаг игнорируется, если для базы данных поддерживаются отсортированные дубликаты.

DB_BEFORE

Ведет себя так же, как DB_AFTER, за исключением того, что новая запись вставляется непосредственно перед текущим местоположением курсора в базе данных.

DB_KEYFIRST

Если ключ, предоставленный при вызове DBC->put(), уже существует в базе данных, и база данных настроена на использование дубликатов без сортировки, то новая запись вставляется как первая запись в соответствующий список дубликатов.

DB_KEYLAST

Действует идентично DB_KEYFIRST, за исключением того, что новая дублирующая запись вставляется как последняя запись в списке дубликатов.

Настройка базы данных для поддержки дубликатов

Поддержка дубликатов может быть настроена только во время создания базы данных.

Это можно сделать, указав соответствующие флаги в `DB->set_flags()` перед первым открытием базы данных.

Флаги, которые можно использовать:

DB_DUP

База данных поддерживает несортированные дубликаты записей.

DB_DUPSORT

База данных поддерживает отсортированные дубликаты записей.

Этот флаг также устанавливает и флаг DB_DUP.

Следующий фрагмент кода иллюстрирует, как настроить базу данных для поддержки отсортированных дубликатов записей:

```
#include <db.h>
...
DB   *dbp;
FILE *error_file_pointer;
int   ret;
char *program_name = "my_prog";
char *file_name    = "mydb.db";

// Разного рода присвоения опущены для пущей ясности

// Инициализация дескриптора БД
ret = db_create(&dbp, NULL, 0);
if (ret != 0) {
    fprintf(error_file_pointer, "%s: %s\n", program_name, db_strerror(ret));
    return(ret);
}

// Установка обработчика ошибок для данной БД
dbp->set_errfile(dbp, error_file_pointer);
dbp->set_errpfx(dbp, program_name);
```

```
//
// Настройка базы данных для сортировки дубликатов
//
ret = dbp->set_flags(dbp, DB_DUPSORT);
if (ret != 0) {
    dbp->err(dbp, ret, "Попытка установить флаг DUPSORT не удалась");
    dbp->close(dbp, 0);
    return(ret);
}

// Теперь открываем БД
ret = dbp->open(dbp,       // Указательн а БД
                NULL,      // Указатальн транзакцию
                file_name, // Имя файла
                NULL,      // Логическое имя базы данных (опционально)
                DB_BTREE,  // Тип базы данных (используется btree)
                DB_CREATE, // Флаги открытия
                0);        // Режим доступа к файлу (значения по умолчанию)
if (ret != 0) {
    dbp->err(dbp, ret, "Database '%s' open failed.", file_name);
    dbp->close(dbp, 0);
    return(ret);
}
```

Настройка функций сравнения

По умолчанию DB использует функцию сравнения лексикографического порядка, при этом более короткие записи сортируются перед более длинными. В большинстве случаев это сравнение работает хорошо, и не нужно управлять им каким-либо образом.

Однако в некоторых ситуациях производительность приложения может выиграть от настройки пользовательской процедуры сравнения. Это можно сделать это либо для ключей базы данных, либо для данных, если база данных поддерживает отсортированные дубликаты записей.

Некоторые из причин, по которым может потребоваться предоставить пользовательскую функцию сортировки:

- База данных имеет ключи с использованием строк, и необходимо обеспечить некое упорядочение для этих данных, чувствительное к языку. Это может помочь увеличить локальность ссылок, что позволит базе данных работать более производительно.

- Используется система с порядком байтов Little Endian (например, x86) и в качестве ключей используются целые числа. Berkeley DB хранит ключи как строки байтов, а целые числа с прямым порядком байтов плохо сортируются при просмотре как строки байтов.

Существует несколько решений этой проблемы, одно из которых — предоставить пользовательскую функцию сравнения.

- По какой-либо причине не требуется, чтобы весь ключ участвовал в сравнении.

В этом случае следует предоставить пользовательскую функцию сравнения, проверяющую только соответствующие байты.

Создание функций сравнения

Функция сравнения ключей BTree устанавливается с помощью `DB->set_bt_compare()`.

Функция сравнения дубликатов данных BTree устанавливается с помощью метода `DB->set_dup_compare()`.

После открытия базы данных эти методы использовать нельзя.

Кроме того, если база данных при ее открытии уже существует, функция, предоставленная этим методам, должна быть той же, что и исторически использовавшаяся для создания базы данных, иначе может произойти повреждение.

Значение, которое предоставляется методу `set_bt_compare()`, является указателем на функцию, которая имеет следующую сигнатуру:

```
int (*function)(DB        *db,   //
                const DBT *key1, //
                const DBT *key2, //
                size_t    *locp) //
```

Эта функция должна возвращать целочисленное значение, меньшее, равное или большее 0.

Если key1 считается больше key2, то функция должна возвращать значение больше 0.

Если они равны, то функция должна возвращать 0.

Если key1 меньше key2, то функция должна возвращать отрицательное значение.

Функция, которая предоставляетсядля set_dup_compare(), работает точно так же, за исключением того, что параметры DBT содержат элементы данных записи вместо ключей.

Ниже пример процедуры, которая используется для сортировки целочисленных ключей в базе данных:

```
int compare_int(DB *dbp, const DBT *a, const DBT *b, size_t *locp) {
    int ai, bi;
    locp = NULL;
    /*
    * Returns:
    * < 0 if a < b
    * = 0 if a = b
    * > 0 if a > b
    */
    memcpy(&ai, a->data, sizeof(int));
    memcpy(&bi, b->data, sizeof(int));
    return (ai – bi);
}
```

ВАЖНО!!! — данные должны быть сначала скопированы в память, которая соответствующим образом выровнена, поскольку Berkeley DB не гарантирует никакого выравнивания данных, в том числе для процедур сравнения.

При написании процедур сравнения следует помнить, что базы данных, созданные на машинах с разной архитектурой, могут иметь разные порядки байтов целых чисел, которые может потребоваться компенсировать.

Чтобы заставить DB использовать вышеприведенную функцию сравнения, пишем:

```
#include <db.h>
#include <string.h>
...
DB *dbp;
int ret;

// Создаем БД
ret = db_create(&dbp, NULL, 0);
if (ret != 0) {
    fprintf(stderr, "%s: %s\n", "my_program",
    db_strerror(ret));
    return(-1);
}

// Настраиваем функцию сравнения BTree для этой БД
dbp->set_bt_compare(dbp, compare_int);

// После этого можем открывать БД
```