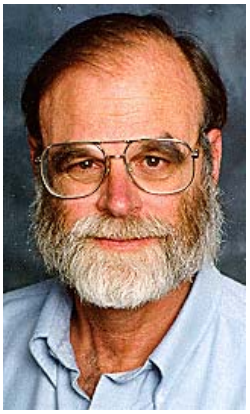


个人简介
专业打杂程序员
联系方式
新浪微博 腾讯微博

IT新闻:
苹果新Retina MacBook Pro（2014年中）开箱图+SSD简单测试 7分钟前
网吧里玩出的世界冠军 打场游戏赚了400万 9分钟前
Twitter收购深度学习创业公司Madbits 34分钟前

昵称：YY哥
园龄：7年2个月
粉丝：342
关注：2
+加关注

< 2009年3月 >

日	一	二	三	四	五	六
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

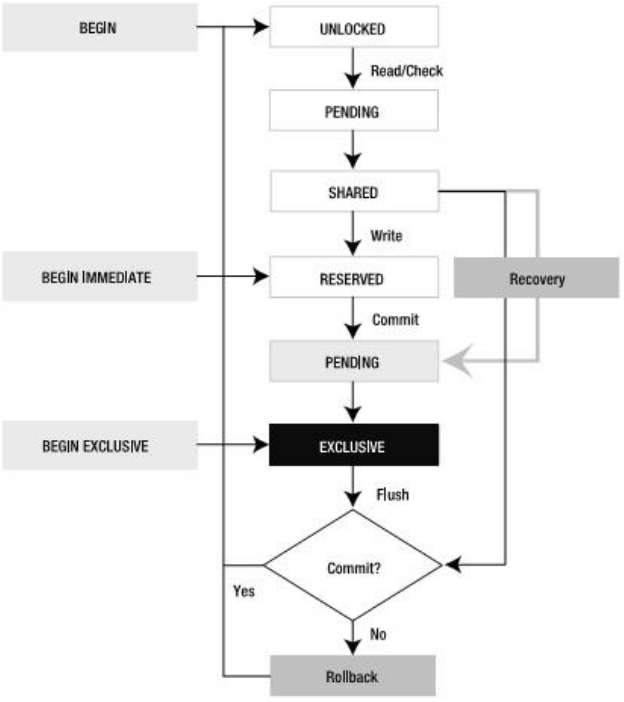
随笔分类
c/c++(9)
Linux相关(24)
MySQL(11)
Others(2)
Web技术(12)
数据结构与算法(15)
数据库技术(30)
系统相关(3)
云计算与虚拟化(3)

随笔档案
2014年7月 (4)

SQLite入门与分析(五)---Page Cache之并发控制

写在前面:本节主要谈谈SQLite的锁机制，SQLite是基于锁来实现并发控制的，所以本节的内容实际上是属于事务处理的，但是SQLite的锁机制实现非常的简单而巧妙，所以在这里单独讨论一下。如果真正理解了它，对整个事务的实现也就理解了。而要真正理解SQLite的锁机制，最好方法就是阅读SQLite的源码，所以在阅读本文时，最好能结合源码。SQLite的锁机制很巧妙，尽管在本节中的源码中，我写了很多注释，也是我个人在研究时的一点心得，但是我发现仅仅用言语，似乎不能把问题说清楚，只有通过体会，才能真正理解SQLite的锁机制。好了，下面进入正题。

SQLite的并发控制机制是采用加锁的方式，实现非常简单，但也非常的巧妙，本节将对其进行一个详细的解剖。请仔细阅读下图，它可以帮助更好的理解下面的内容。



1、RESERVED LOCK
RESERVED锁意味着进程将要对数据库进行写操作。某一时刻只能有一个RESERVED Lock，但是RESERVED锁和SHARED锁可以共存，而且可以对数据库加新的SHARED锁。
为什么要用RESERVED锁？
主要是出于并发性的考虑。由于SQLite只有库级排斥锁（EXCLUSIVE LOCK），如果写事务一开始就上EXCLUSIVE锁，然后再进行实际的数据更新，写磁盘操作，这会使得并发性大大降低。而SQLite一旦得到数据库的RESERVED锁，就可以对缓存中的数据进行修改，而与此同时，其它进程可以继续进行读操作。直到真正需要写磁盘时才对数据库加EXCLUSIVE锁。
2、PENDING LOCK
PENDING LOCK意味着进程已经完成缓存中的数据修改，并想立即将更新写入磁盘。它将等待此时已经存在的读锁事务完成，但是不允许对数据库加新的SHARED LOCK(这与RESERVED LOCK相区别)。
为什么要有PENDING LOCK?
主要是为了防止出现写饿死的情况。由于写事务先要获取RESERVED LOCK，所以可能一直产生新的SHARED LOCK，使得写事务发生饿死的情况。

3、加锁机制的具体实现

SQLite在pager层获取锁的函数如下：

```
//获取一个文件的锁,如果忙则重复该操作,
//直到busy 回调用函数返回flase,或者成功获得锁
static int pager_wait_on_lock(Pager *pPager, int locktype){
  int rc;
  assert( PAGER_SHARED==SHARED_LOCK );
  assert( PAGER_RESERVED==RESERVED_LOCK );
  assert( PAGER_EXCLUSIVE==EXCLUSIVE_LOCK );
  if( pPager->state>=locktype ){
    rc = SQLITE_OK;
  }else{
   //重复直到获得锁
```

2014年3月 (1)
2013年9月 (1)
2013年8月 (1)
2013年2月 (1)
2012年11月 (4)
2012年1月 (1)
2011年12月 (1)
2011年10月 (1)
2011年3月 (1)
2010年9月 (1)
2010年8月 (1)
2010年7月 (3)
2010年6月 (2)
2010年5月 (7)
2010年4月 (1)
2010年3月 (1)
2010年1月 (1)
2009年12月 (2)
2009年10月 (2)
2009年9月 (14)
2009年8月 (4)
2009年6月 (14)
2009年5月 (3)
2009年4月 (1)
2009年3月 (3)
2009年2月 (11)
2008年10月 (7)
2008年8月 (5)
2008年7月 (1)
2008年6月 (2)
2008年5月 (2)
2008年4月 (5)

kernel

kernel中文社区
LDN
The Linux Document Project
The Linux Kernel Archives

manual

cppreference
gcc manual
mysql manual

sites

Database Journal
Fedora镜象
highscalability
KFUPM ePrints
Linux docs
Linux Journal
NoSQL
SQLite

技术社区

apache
CSDN
IBM-developerworks
lucene中国
nutch中国
oldlinux
oracle's forum

最新评论

1. Re:理解MySQL——架构与概念
我试验了下.数据 5 9 10 13 18begin;select * from asf_execution where num> 5 and num 5 and INSTANCE_ID_<18 lock in share mode;会有 1.行锁 2.间隙所 [5 18)插

```
    do {
      rc = sqlite3OsLock(pPager->fd, locktype);
    }while( rc==SQLITE_BUSY && sqlite3InvokeBusyHandler(pPager->pBusyHandler) );

    if( rc==SQLITE_OK ){

        //设置pager的状态
      pPager->state = locktype;
    }
  }
  return rc;
}
```

Windows下具体的实现如下：

```
static int winLock(OsFile *id, int locktype){
  int rc = SQLITE_OK;    /* Return code from subroutines */
  int res = 1;           /* Result of a windows lock call */
  int newLocktype;       /* Set id->locktype to this value before exiting */
  int gotPendingLock = 0;/* True if we acquired a PENDING lock this time */
  winFile *pFile = (winFile*)id;

  assert( pFile!=0 );
  TRACE5("LOCK %d %d was %d(%d)\n",
          pFile->h, locktype, pFile->locktype, pFile->sharedLockByte);

  /* If there is already a lock of this type or more restrictive on the
  ** OsFile, do nothing. Don't use the end_lock: exit path, as
  ** sqlite3OsEnterMutex() hasn't been called yet.
  */
  //当前的锁>=locktype,则返回
  if( pFile->locktype>=locktype ){
    return SQLITE_OK;
  }

  /* Make sure the locking sequence is correct
  */
  assert( pFile->locktype!=NO_LOCK || locktype==SHARED_LOCK );
  assert( locktype!=PENDING_LOCK );
  assert( locktype!=RESERVED_LOCK || pFile->locktype==SHARED_LOCK );

  /* Lock the PENDING_LOCK byte if we need to acquire a PENDING lock or
  ** a SHARED lock.  If we are acquiring a SHARED lock, the acquisition of
  ** the PENDING_LOCK byte is temporary.
  */
  newLocktype = pFile->locktype;
  /*两种情况: (1)如果当前文件处于无锁状态(获取读锁---读事务
  **和写事务在最初阶段都要经历的阶段),
  **(2)处于RESERVED_LOCK, 且请求的锁为EXCLUSIVE_LOCK(写事务)
  **则对执行加PENDING_LOCK
  */
  ////////////////////(1)//////////////////
  if( pFile->locktype==NO_LOCK
   || (locktype==EXCLUSIVE_LOCK && pFile->locktype==RESERVED_LOCK)
  ){
    int cnt = 3;
    //加pending锁
    while( cnt-->0 && (res = LockFile(pFile->h, PENDING_BYTE, 0, 1, 0))==0 ){
      /* Try 3 times to get the pending lock.  The pending lock might be
      ** held by another reader process who will release it momentarily.
      */
      TRACE2("could not get a PENDING lock. cnt=%d\n", cnt);
      Sleep(1);
    }
    //设置为gotPendingLock为1,使和在后面要释放PENDING锁
    gotPendingLock = res;
  }

  /* Acquire a shared lock
  */
  /*获取shared lock
  **此时,事务应该持有PENDING锁,而PENDING锁作为事务从UNLOCKED到
  **SHARED_LOCKED的一个过渡,所以事务由PENDING->SHARED
  **此时,实际上锁处于两个状态:PENDING和SHARED,
```

入INSERT I.....
--麒麟飞
2. Re:理解MySQL——架构与概念
例1-5
insert into t(i) values(1);
这句话应该是可以插入的.
不会被阻塞
--麒麟飞
3. Re:理解MySQL——架构与概念
注：SELECT ... FOR UPDATE仅在自动提交关闭(即手动提交)时才会对元组加锁，而在自动提交时，符合条件的元组不会被加锁。

这个是错误的.自动提交的,也会尝试获取排它锁.
你可以试验下.
--麒麟飞
4. Re:浅谈mysql的两阶段提交协议
YY哥 偶像啊!细腻文笔 配有说服力的代码和图 我崇拜你 !!!
之前sqlite的深入分析帮了我大忙..现在做mysql相关 有来你的博客找东西 哈哈哈!!
--hark.perfe
5. Re:(i++)+(i++)与(++i)+(++i)
@arrowcat
这类语句本身没什么意义，但是楼主思考的角度让我豁然开朗。
--HJWAJ

阅读排行榜

1. 理解MySQL——索引与优化(77627)
2. SQLite入门与分析(一)---简介(48610)
3. 理解MySQL——复制(Replication)(26209)
4. libevent源码分析(19048)
5. SQLite入门与分析(二)---设计与概念(16977)

评论排行榜

1. (i++)+(i++)与(++i)+(++i)(40)
2. SQLite入门与分析(一)---简介(30)
3. 浅谈SQLite——实现与应用(20)
4. 一道算法题,求更好的解法(18)
5. 理解MySQL——索引与优化(16)

推荐排行榜

1. SQLite入门与分析(一)---简介(12)
2. 理解MySQL——索引与优化(12)
3. 浅谈SQLite——查询处理及优化(10)
4. 乱谈服务器编程(9)
5. libevent源码分析(6)

```
**直到后面释放PENDING锁后,才真正处于SHARED状态
*/
///////////////(2)/////////////////////////////////
if( locktype==SHARED_LOCK && res ){
  assert( pFile->locktype==NO_LOCK );
  res = getReadLock(pFile);
  if( res ){
    newLocktype = SHARED_LOCK;
  }
}

/* Acquire a RESERVED lock
*/
/*获取RESERVED
**此时事务持有SHARED_LOCK,变化过程为SHARED->RESERVED。
**RESERVED锁的作用就是为了提高系统的并发性能
*/
////////////////////(3)///////////////////////////////
if( locktype==RESERVED_LOCK && res ){
  assert( pFile->locktype==SHARED_LOCK );
  //加RESERVED锁
  res = LockFile(pFile->h, RESERVED_BYTE, 0, 1, 0);
  if( res ){
    newLocktype = RESERVED_LOCK;
  }
}

/* Acquire a PENDING lock
*/
/*获取PENDING锁
**此时事务持有RESERVED_LOCK,且事务申请EXCLUSIVE_LOCK
**变化过程为:RESERVED->PENDING。
**PENDING状态只是唯一的作用就是防止写饿死.
**读事务不会执行该代码,但是写事务会执行该代码,
**执行该代码后gotPendingLock设为0，后面就不会释放PENDING锁。
*/
////////////////////////////(4)///////////////////////////////
if( locktype==EXCLUSIVE_LOCK && res ){
    //这里没有实际的加锁操作，只是把锁的状态改为PENDING状态
  newLocktype = PENDING_LOCK;
  //设置了gotPendingLock,后面就不会释放PENDING锁了,
  //相当于加了PENDING锁,实际上是在开始处加的PENDING锁
  gotPendingLock = 0;
}

/* Acquire an EXCLUSIVE lock
*/
/*获取EXCLUSIVE锁
**当一个事务执行该代码时,它应该满足以下条件:
**(1)锁的状态为:PENDING (2)是一个写事务
**变化过程:PENDING->EXCLUSIVE
*/
////////////////////(5)/////////////////////////////////////////
if( locktype==EXCLUSIVE_LOCK && res ){
  assert( pFile->locktype>=SHARED_LOCK );
  res = unlockReadLock(pFile);
  TRACE2("unreadlock = %d\n", res);
  res = LockFile(pFile->h, SHARED_FIRST, 0, SHARED_SIZE, 0);
  if( res ){
    newLocktype = EXCLUSIVE_LOCK;
  }else{
    TRACE2("error-code = %d\n", GetLastError());
  }
}

/* If we are holding a PENDING lock that ought to be released, then
** release it now.
*/
/*此时事务在第2步中获得PENDING锁,它将申请SHARED_LOCK(第3步,和图形相对照),
**而在之前它已经获取了PENDING锁,
**所以在这里它需要释放PENDING锁,此时锁的变化为:PENDING->SHARED
*/
////////////////////////(6)///////////////////////////////////
if( gotPendingLock && locktype==SHARED_LOCK ){
  UnlockFile(pFile->h, PENDING_BYTE, 0, 1, 0);
}
```

```
  /* Update the state of the lock has held in the file descriptor then
  ** return the appropriate result code.
  */
  if( res ){
    rc = SQLITE_OK;
  }else{
    TRACE4("LOCK FAILED %d trying for %d but got %d\n", pFile->h,
           locktype, newLocktype);
    rc = SQLITE_BUSY;
  }
  //在这里设置文件锁的状态
  pFile->locktype = newLocktype;
  return rc;
}
```

在几个关键的部位标记数字。

(I)对于一个读事务会的完整经过：
语句序列：（1）——>（2）——>（6）
相应的状态真正的变化过程为：UNLOCKED→PENDING(1)→PENDING、SHARED(2)→SHARED（6）→UNLOCKED
(II)对于一个写事务完整经过：
第一阶段：
语句序列：（1）——>（2）——>（6）
状态变化：UNLOCKED→PENDING(1)→PENDING、SHARED(2)→SHARED(6)。此时事务获得SHARED LOCK。
第二个阶段：
语句序列：（3）
此时事务获得RESERVED LOCK。
第三个阶段：
事务执行修改操作。
第四个阶段：
语句序列：（1）——>（4）——>（5）
状态变化为：
RESERVED→ RESERVED 、PENDING(1)→PENDING（4）→EXCLUSIVE（5）。此时事务获得排斥锁，就可以进行写磁盘操作了。

注：在上面的过程中，由于（1）的执行，使得某些时刻SQLite处于两种状态，但它持续的时间很短，从某种程度上来说可以忽略，但是为了把问题说清楚，在这里描述了这一微妙而巧妙的过程。

4、SQLite的死锁问题
SQLite的加锁机制会不会出现死锁?
这是一个很有意思的问题，对于任何采取加锁作为并发控制机制的DBMS都得考虑这个问题。有两种方式处理死锁问题：（1）死锁预防(deadlock prevention)（2）死锁检测(deadlock detection)与死锁恢复(deadlock recovery)。SQLite采取了第一种方式，如果一个事务不能获取锁，它会重试有限次（这个重试次数可以由应用程序运行预先设置，默认为1次）——这实际上是基本锁超时的机制。如果还是不能获取锁，SQLite返回SQLITE_BUSY错误给应用程序，应用程序此时应该中断，之后再重试；或者中止当前事务。虽然基于锁超时的机制简单，容易实现，但是它的缺点也是明显的——资源浪费。

5、事务类型(Transaction Types)
既然SQLite采取了这种机制，所以应用程序得处理SQLITE_BUSY 错误，先来看一个会产生SQLITE_BUSY错误的例子：

Session A	Session B
sqlite> BEGIN;	
	sqlite> BEGIN;
	sqlite> INSERT INTO foo VALUES ('x');
sqlite> SELECT * FROM foo;	
	sqlite> COMMIT;
	SQL error: database is locked
sqlite> INSERT INTO foo VALUES ('x');	
SQL error: database is locked	

所以应用程序应该尽量避免产生死锁，那么应用程序如何做可以避免死锁的产生呢？
答案就是为你的程序选择正确合适的事务类型。
SQLite有三种不同的事务类型，这不同于锁的状态。事务可以从DEFERRED，IMMEDIATE或者EXCLUSIVE，一个事务的类型在BEGIN命令中指定：
BEGIN [DEFERRED | IMMEDIATE | EXCLUSIVE] TRANSACTION;
一个deferred事务不获取任何锁，直到它需要锁的时候，而且BEGIN语句本身也不会做什么事情——它开始于UNLOCK状态；默认情况下是这样的。如果仅仅用BEGIN开始一个事务，那么事务就是DEFERRED的，同时它不会获取任何锁，当对数据库进行第一次读操作时，它会获取SHARED LOCK；同样，当进行第一次写操作时，它会获取RESERVED LOCK。
由BEGIN开始的Immediate事务会试着获取RESERVED LOCK。如果成功，BEGIN IMMEDIATE保证没有别的连接可以写数据库。但是，别的连接可以对数据库进行读操作，但是RESERVED LOCK会阻止其它的连接BEGIN IMMEDIATE或者BEGIN EXCLUSIVE命令，SQLite会返回SQLITE_BUSY错误。这时你就可以对数据库进行修改操作，但是你不能提交，当你COMMIT时，会返回SQLITE_BUSY错误，这意味着还有其它的读事务没有完成，得等它们执行完后才能提交事务。

Exclusive事务会试着获取对数据库的EXCLUSIVE锁。这与IMMEDIATE类似，但是一旦成功，EXCLUSIVE事务保证没有其它的连接，所以就可对数据库进行读写操作了。
上面那个例子的问题在于两个连接最终都想写数据库，但是他们都没有放弃各自原来的锁，最终，shared 锁导致了问题的出现。
如果两个连接都以BEGIN IMMEDIATE开始事务，那么死锁就不会发生。在这种情况下，在同一时刻只能有一个连接进入BEGIN IMMEDIATE，其它的连接就得等待。BEGIN IMMEDIATE和BEGIN EXCLUSIVE通常被写事务使用。就像同步机制一样，它防止了死锁的产生。
基本的准则是：如果你在使用的数据库没有其它的连接，用BEGIN就足够了。但是，如果你使用的数据库在其它的连接也要对数据库进行写操作，就得使用BEGIN IMMEDIATE或BEGIN EXCLUSIVE开始你的事务。

分类: 数据库技术

YY哥
关注 - 2
粉丝 - 342
+加关注

« 上一篇：SQLite Version3.3.6源代码文件结构
» 下一篇：SQLite入门与分析(六)---再谈SQLite的锁

posted @ 2009-03-01 12:32 YY哥 阅读(6011) 评论(2) 编辑 收藏

评论列表

#1楼 2009-03-02 10:44 ajax+u[未注册用户]

还是期待你的 分析 sqlite 中的 虚拟机工作机制

#2楼 2011-06-01 11:04 wz_dk_123

请教下，您文章中的那幅图使用什么工具画的.

支持(0) 反对(0)