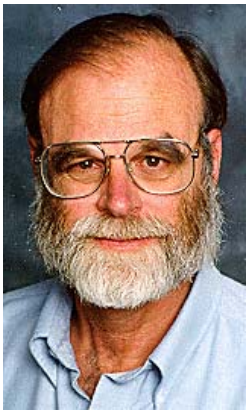


个人简介
专业打杂程序员
联系方式
新浪微博 腾讯微博

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2月 >

日	一	二	三	四	五	六
25	26	27	28	29	30	31
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
1	2	3	4	5	6	7

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)

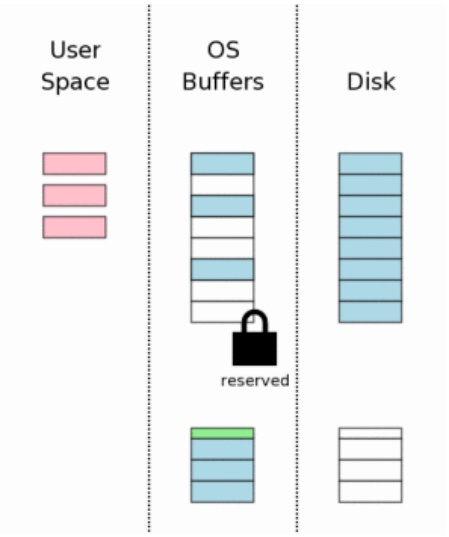
随笔档案
2014年7月 (4)

SQLite入门与分析(四)---Page Cache之事务处理(2)

写在前面：个人认为pager层是SQLite实现最为核心的模块，它具有四大功能：I/O，页面缓存，并发控制和日志恢复。而这些功能不仅是上层Btree的基础，而且对系统的性能和健壮性有关至关重要的影响。其中并发控制和日志恢复是事务处理实现的基础。SQLite并发控制的机制非常简单——封锁机制；别外，它的查询优化机制也非常简单——基于索引。这一切使得整个SQLite的实现变得简单，SQLite变得很小，运行速度也非常快，所以，特别适合嵌入式设备。好了，接下来讨论事务的剩余部分。

6、修改位于用户进程空间的页面(Changing Database Pages In User Space)

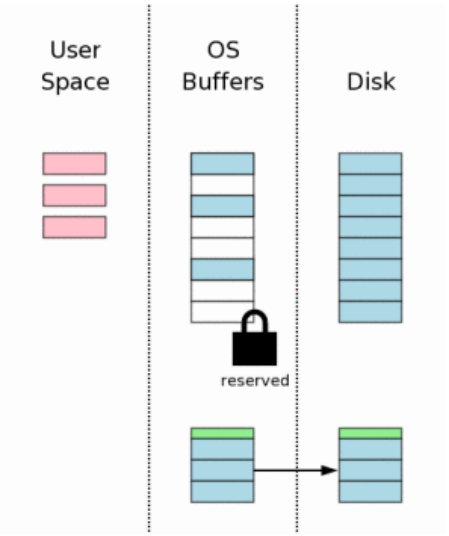
页面的原始数据写入日志之后，就可以修改页面了——位于用户进程空间。每个数据库连接都有自己私有的空间，所以页面的变化只对该连接可见，而对其它连接的数据仍然是磁盘缓存中的数据。从这里可以明白一件事：一个进程在修改页面数据的同时，其它进程可以继续进行读操作。图中的红色表示修改的页面。



7、日志文件刷入磁盘(Flushing The Rollback Journal File To Mass Storage)

接下来把日志文件的内容刷入磁盘，这对于数据库从意外中恢复来说是至关重要的一步。而且这通常也是一个耗时的操作，因为磁盘I/O速度很慢。

这个步骤不只把日志文件刷入磁盘那么简单，它的实现实际上分成两步：首先把日志文件的内容刷入磁盘（即页面数据）；然后把日志文件中页面的数目写入日志文件头，再把header刷入磁盘（这一过程在代码中清晰可见）。



代码如下：

```
/*
**Sync日志文件,保证所有的脏页面写入磁盘日志文件
*/
static int syncJournal(Pager *pPager){
  PgHdr *pPg;
  int rc = SQLITE_OK;
```

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

```
  /* Sync the journal before modifying the main database
  ** (assuming there is a journal and it needs to be synced.)
  */
  if( pPager->needSync ){
    if( !pPager->tempFile ){
      assert( pPager->journalOpen );
      /* assert( !pPager->noSync ); // noSync might be set if synchronous
      ** was turned off after the transaction was started.  Ticket #615 */
#ifndef NDEBUG
      {
        /* Make sure the pPager->nRec counter we are keeping agrees
        ** with the nRec computed from the size of the journal file.
        */
        i64 jSz;
        rc = sqlite3OsFileSize(pPager->jfd, &jSz);
        if( rc!=0 ) return rc;
        assert( pPager->journalOff==jSz );
      }
#endif
      {
        /* Write the nRec value into the journal file header. If in
        ** full-synchronous mode, sync the journal first. This ensures that
        ** all data has really hit the disk before nRec is updated to mark
        ** it as a candidate for rollback.
        */
        if( pPager->fullSync ){
          TRACE2("SYNC journal of %d\n", PAGERID(pPager));
    //首先保证脏页面中所有的数据都已经写入日志文件
          rc = sqlite3OsSync(pPager->jfd, 0);
          if( rc!=0 ) return rc;
        }
        rc = sqlite3OsSeek(pPager->jfd,
                           pPager->journalHdr + sizeof(aJournalMagic));
        if( rc ) return rc;
    //页面的数目写入日志文件
        rc = write32bits(pPager->jfd, pPager->nRec);
        if( rc ) return rc;

        rc = sqlite3OsSeek(pPager->jfd, pPager->journalOff);
        if( rc ) return rc;
      }
      TRACE2("SYNC journal of %d\n", PAGERID(pPager));
      rc = sqlite3OsSync(pPager->jfd, pPager->full_fsync);
      if( rc!=0 ) return rc;
      pPager->journalStarted = 1;
    }
    pPager->needSync = 0;

    /* Erase the needSync flag from every page.
    */
    //清除needSync标志位
    for(pPg=pPager->pAll; pPg; pPg=pPg->pNextAll){
      pPg->needSync = 0;
    }
    pPager->pFirstSynced = pPager->pFirst;
  }

#ifndef NDEBUG
  /* If the Pager.needSync flag is clear then the PgHdr.needSync
  ** flag must also be clear for all pages.  Verify that this
  ** invariant is true.
  */
  else{
    for(pPg=pPager->pAll; pPg; pPg=pPg->pNextAll){
      assert( pPg->needSync==0 );
    }
    assert( pPager->pFirstSynced==pPager->pFirst );
  }
#endif

  return rc;
}
```

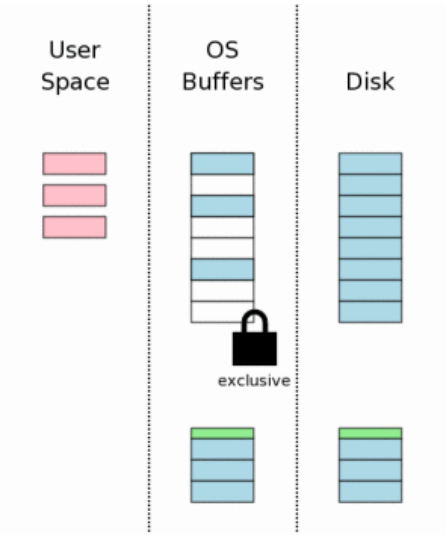
8、获取排斥锁（Obtaining An Exclusive Lock）

在对数据库文件进行修改之前(注：这里不是内存中的页面),我们必须得到数据库文件的排斥锁(Exclusive Lock)。得到排斥锁的过

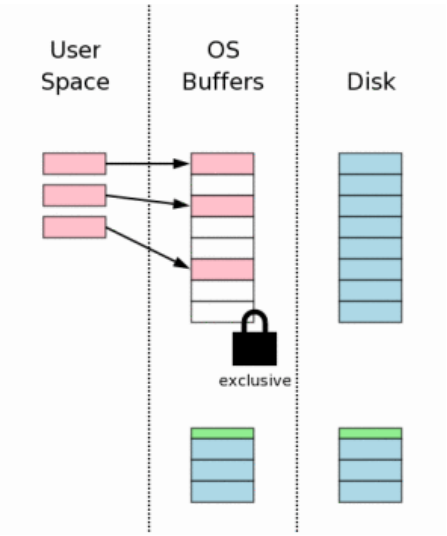
入INSERT I.....
--麒麟飞
2. Re:理解MySQL——架构与概念
例1-5
insert into t(i) values(1);
这句话应该是可以插入的.
不会被阻塞
--麒麟飞
3. Re:理解MySQL——架构与概念
注：SELECT ... FOR UPDATE仅在自动提交关闭(即手动提交)时才会对元组加锁，而在自动提交时，符合条件的元组不会被加锁。

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

程可分为两步：首先得到Pending lock；然后Pending lock升级到exclusive lock。
Pending lock允许其它已经存在的Shared lock继续读数据库文件，但是不允许产生新的shared lock，这样做目的是为了防止写操作发生饿死情况。一旦所有的shared lock完成操作，则pending lock升级到exclusive lock。



9、修改的页面写入文件(Writing Changes To The Database File)
一旦得到exclusive lock，其它的进程就不能进行读操作，此时就可以把修改的页面写回数据库文件，但是通常OS都把结果暂时保存到磁盘缓存中，直到某个时刻才会真正把结果写入磁盘。



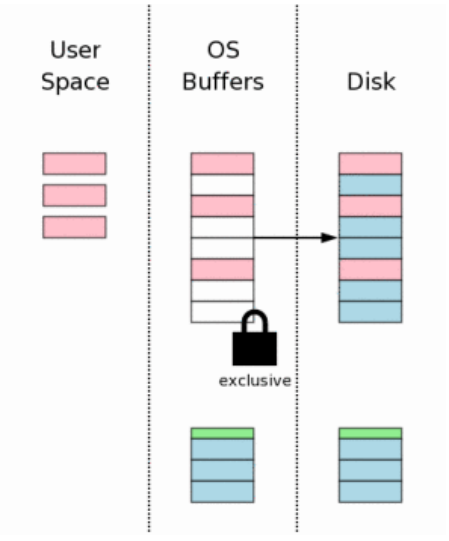
以上两步的实现代码：

```
//把所有的脏页面写入数据库
//到这里开始获取EXCLUSIVEQ锁，并将页面写回操作系统文件
static int pager_write_pagelist(PgHdr *pList){
  Pager *pPager;
  int rc;

  if( pList==0 ) return SQLITE_OK;
  pPager = pList->pPager;

  /* At this point there may be either a RESERVED or EXCLUSIVE lock on the
  ** database file. If there is already an EXCLUSIVE lock, the following
  ** calls to sqlite3OsLock() are no-ops.
  **
  ** Moving the lock from RESERVED to EXCLUSIVE actually involves going
  ** through an intermediate state PENDING.   A PENDING lock prevents new
  ** readers from attaching to the database but is unsufficient for us to
  ** write.  The idea of a PENDING lock is to prevent new readers from
  ** coming in while we wait for existing readers to clear.
  **
  ** While the pager is in the RESERVED state, the original database file
  ** is unchanged and we can rollback without having to playback the
  ** journal into the original database file.  Once we transition to
```

```
  ** EXCLUSIVE, it means the database file has been changed and any rollback
  ** will require a journal playback.
  */
  //加EXCLUSIVE_LOCK锁
  rc = pager_wait_on_lock(pPager, EXCLUSIVE_LOCK);
  if( rc!=SQLITE_OK ){
    return rc;
  }

  while( pList ){
    assert( pList->dirty );
    rc = sqlite3OsSeek(pPager->fd, (pList->pgno-1)*(i64)pPager->pageSize);
    if( rc ) return rc;
    /* If there are dirty pages in the page cache with page numbers greater
    ** than Pager.dbSize, this means sqlite3pager_truncate() was called to
    ** make the file smaller (presumably by auto-vacuum code). Do not write
    ** any such pages to the file.
    */
    if( pList->pgno<=pPager->dbSize ){
      char *pData = CODEC2(pPager, PGHDR_TO_DATA(pList), pList->pgno, 6);
      TRACE3("STORE %d page %d\n", PAGERID(pPager), pList->pgno);
      //写入文件
      rc = sqlite3OsWrite(pPager->fd, pData, pPager->pageSize);
      TEST_INCR(pPager->nWrite);
    }
#ifndef NDEBUG
    else{
      TRACE3("NOSTORE %d page %d\n", PAGERID(pPager), pList->pgno);
    }
#endif
    if( rc ) return rc;
    //设置dirty
    pList->dirty = 0;
#ifdef SQLITE_CHECK_PAGES
    pList->pageHash = pager_pagehash(pList);
#endif
//指向下一个脏页面
    pList = pList->pDirty;
  }
  return SQLITE_OK;
}
```

10、修改结果刷入存储设备(Flushing Changes To Mass Storage)

为了保证修改结果真正写入磁盘，这一步必不要少。对于数据库存的完整性，这一步也是关键的一步。由于要进行实际的I/O操作，所以和第7步一样，将花费较多的时间。



最后来看看这几步是如何实现的：

其实以上以上几步是在函数sqlite3BtreeSync()---btree.c中调用的(而关于该函数的调用后面再讲)。

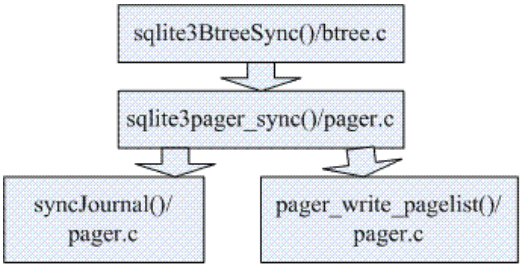
代码如下：

```
//同步btree对应的数据库文件
//该函数返回之后，只需要提交写事务，删除日志文件
```

```
int sqlite3BtreeSync(Btree *p, const char *zMaster){
  int rc = SQLITE_OK;
  if( p->inTrans==TRANS_WRITE ){
    BtShared *pBt = p->pBt;
    Pgno nTrunc = 0;
#ifndef SQLITE_OMIT_AUTOVACUUM
    if( pBt->autoVacuum ){
      rc = autoVacuumCommit(pBt, &nTrunc);
      if( rc!=SQLITE_OK ){
        return rc;
      }
    }
#endif

 //调用pager进行sync
    rc = sqlite3pager_sync(pBt->pPager, zMaster, nTrunc);
  }
  return rc;
}

//把pager所有脏页面写回文件
int sqlite3pager_sync(Pager *pPager, const char *zMaster, Pgno nTrunc){
  int rc = SQLITE_OK;

  TRACE4("DATABASE SYNC: File=%s zMaster=%s nTrunc=%d\n",
      pPager->zFilename, zMaster, nTrunc);

  /* If this is an in-memory db, or no pages have been written to, or this
  ** function has already been called, it is a no-op.
  */
  //pager不处于PAGER_SYNCED状态,dirtyCache为1,
  //则进行sync操作
  if( pPager->state!=PAGER_SYNCED && !MEMDB && pPager->dirtyCache ){
    PgHdr *pPg;
    assert( pPager->journalOpen );

    /* If a master journal file name has already been written to the
    ** journal file, then no sync is required. This happens when it is
    ** written, then the process fails to upgrade from a RESERVED to an
    ** EXCLUSIVE lock. The next time the process tries to commit the
    ** transaction the m-j name will have already been written.
    */
    if( !pPager->setMaster ){
        //pager修改计数
      rc = pager_incr_changecounter(pPager);
      if( rc!=SQLITE_OK ) goto sync_exit;
#ifndef SQLITE_OMIT_AUTOVACUUM
      if( nTrunc!=0 ){
        /* If this transaction has made the database smaller, then all pages
        ** being discarded by the truncation must be written to the journal
        ** file.
        */
        Pgno i;
        void *pPage;
        int iSkip = PAGER_MJ_PGNO(pPager);
        for( i=nTrunc+1; i<=pPager->origDbSize; i++ ){
          if( !(pPager->aInJournal[i/8] & (1<<(i&7))) && i!=iSkip ){
            rc = sqlite3pager_get(pPager, i, &pPage);
            if( rc!=SQLITE_OK ) goto sync_exit;
            rc = sqlite3pager_write(pPage);
            sqlite3pager_unref(pPage);
            if( rc!=SQLITE_OK ) goto sync_exit;
          }
        }
      }
#endif
      rc = writeMasterJournal(pPager, zMaster);
      if( rc!=SQLITE_OK ) goto sync_exit;

        //sync日志文件
      rc = syncJournal(pPager);
      if( rc!=SQLITE_OK ) goto sync_exit;
    }

#ifndef SQLITE_OMIT_AUTOVACUUM
    if( nTrunc!=0 ){
      rc = sqlite3pager_truncate(pPager, nTrunc);
```

```
      if( rc!=SQLITE_OK ) goto sync_exit;
    }
#endif

    /* Write all dirty pages to the database file */
    pPg = pager_get_all_dirty_pages(pPager);


   //把所有脏页面写回操作系统文件
    rc = pager_write_pagelist(pPg);
    if( rc!=SQLITE_OK ) goto sync_exit;

    /* Sync the database file. */
       //sync数据库文件
    if( !pPager->noSync ){
      rc = sqlite3OsSync(pPager->fd, 0);
    }

    pPager->state = PAGER_SYNCED;
  }else if( MEMDB && nTrunc!=0 ){
    rc = sqlite3pager_truncate(pPager, nTrunc);
  }

sync_exit:
  return rc;
}
```

下图可以进一步解释该过程：



分类: 数据库技术

YY哥
关注 - 2
粉丝 - 342
+加关注

« 上一篇：SQLite入门与分析(四)---Page Cache之事务处理(1)
» 下一篇：SQLite入门与分析(四)---Page Cache之事务处理(3)

posted @ 2009-02-26 14:16 YY哥 阅读(6732) 评论(4) 编辑 收藏

评论列表

#1楼 2009-02-27 09:34 ajax+u[未注册用户]

希望看到的是sqlite虚拟机的实现

#2楼 2009-02-27 16:35 Jeason

sqlite虚拟机是什么意思？
journal 这个文件是不是一个Session一个？还是多个Session共享同一个journal ？

支持(0) 反对(0)

#3楼[楼主] 2009-02-27 18:32 YY哥

VDBE(或者叫做VM)是SQLite中一的核心模块。
关于journal文件，得从SQLite的恢复机制说起，SQLite的恢复机制不同于通用DBMS，它采用的是影子分面技术。总的来说，每开始一个写事务，SQLite就会创建一个journal文件，以备恢复使用，事务成功提交，journal文件就会被删除，事务失败，则在下一个连接开始的时候(打开数据库文件时)会根据journal文件进行恢复操作，之后再将其删除。以后有时间会详细讨论这个问题的。

支持(0) 反对(0)

#4楼 2013-07-29 10:19 hark.perfe

写的太好了!!!非常有用

支持(0) 反对(0)