PostgreSQL 存储管理机制研究

林河水　程　伟　孙玉芳
(中科院软件所　北京100080)

摘　要　作为开放源码数据库的重要代表，PostgreSQL 数据库因其良好的性能得到越来越广泛的应用，日益受到人们的关注。本文着重介绍了 PostgreSQL 数据库存储管理的实现机制，分析了数据库表的存储和组织、存储管理对事务管理的支持以及并发访问控制的实现。对 PostgreSQL 存储管理的体系结构作了一个全景的描述。

关键词　存储管理，锁，并发访问控制，体系结构

Research on the Storge System of the PostgreSQL

LIN He-Shui　CHENG Wei　SUN Yu-Fang
(Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract　As the most important representation of open source database system, PostgreSQL has been applied in many database application domains, and been awarded more and more concerns because of its excellent performance. This paper describes the implementation mechanism of storage system of PostgreSQL. By analyzing the storage and organization of its database tables, the facilities that the storage system provides to transaction control and the way how to control storage access asynchronously, it gives an overview of the storage management architecture of PostgreSQL.

Keywords　Storge management, Lock, Access asynchronously control, Architecture

1　介绍

PostgreSQL 是一个对象关系型数据库管理系统(ORDBMS)，它前身是加利福尼亚大学伯克利分校(the University of California at Berkeley)的数据库管理系统 Postgres[2,6]；目前它已被成功地应用在包括医疗、电子商务[2]等在内的广阔领域中。PostgreSQL 符合 SQL92/99标准，具有以下一些鲜明特色：

·*支持复杂对象数据类型*　许多领域的数据，比如：工程、地理等，比起通常的商业数据，更加复杂、多变。它们通常都是许多维数据的集合体，如果用通常的 RDBMS 来管理，则每维的数据都必须有一个表来模拟；这样一来，为了查询一个对象的有关信息，必然涉及到访问若干个表从而影响到效率。一个直接的解决办法是将描述一类对象所有维的数据构造成一个对象类型，用这个对象作为属性的数据类型存储在表中，这就是数据库管理系统的对象支持技术。

·*支持用户自定义数据类型、操作符以及访问方法*　一般来说，这几个方面是紧密相连的：支持用户自定义数据类型要求提供相应操作符以便运算，同时也要求提供相应类型的访问方法。这里有几个难点：(1)如何做到用户定义的数据类型与原有的操作符方法运算原则相一致?(2)如何做到用户加入的操作符方法在满足自定义的数据类型的同时也能运算现有的数据类型且不冲突?PostgreSQL 的实现做到用户增加数据类型时，不需要有专业的知识和技能。

·*支持"活跃"数据库和规则*　一些与数据的性质联系强的应用使用触发器和报警器就可以满足其要求。然而，一些专家系统的知识比较适合用规则来描述，它的工作完全依赖于规则库的规则，这些规则有以下一些特征：规则之间可能彼此之间冲突；加入一个规则可能导致规则库的缩小而不是增加，因为有些冲突的规则必须被删除。这单靠触发器和报警器是解决不了问题的。所以 Postgres 设计时就考虑到要支持规则库和"活跃数据库"。

·*减少系统崩溃后的恢复代码和代价*[2]　这里，Postgres 使用了"多版本"控制技术，即删除元组的操作并不实际删除它而只是作无效标志，更新元组的操作取代为先删除后插入；数据库管理员可以用命令 VACUUM 定期地清理数据库中无效元组来紧缩存储空间。

PostgreSQL 还拥有其他一些现代数据库特征，甚至某些特色是其他商业系统所没有的[6]；但这些特色的实现都离不开存储管理的支持，存储管理是整个数据库管理系统(DBMS)的基础[4]，它与最终系统的性能、效率以及可靠性戚戚相关。现在让我们来考察一下 PostgreSQL 的存储管理是如何实现的。PostgreSQL 存储管理有以下两个重要特色[3]：(1)支持事务控制和并发数据访问；(2)结合 WAL 机制并采用"多版本"控制机制，定期保存历史数据并简化系统崩溃后的处理。

2　表存储和文件组织

数据库管理系统(DBMS)的基本职责就是要管理各种数据库，提供定义、操纵、控制数据库的功能模块[5]。在关系型数据库中，数据库由表(关系)构成，表(关系)是这类数据库管理系统的基本单位，数据库数据都存储在表中。概括起来，表中

林河水　硕士生，研究方向为数据库管理系统。程　伟　博士生，研究方向为数据库管理系统、系统软件。孙玉芳　教授，博导，研究方向为系统软件、中文信息处理。

存储的对象有下列几种：

·数据字典　数据字典主要是数据库管理系统的管理信息。管理信息有两类：一类是辅助或启动系统自身运行的系统信息；另一类是描述用户定义行为的信息。在 PostgreSQL 实现中，存储数据字典的表称为系统表(system catalog)[1]。

·常规数据　常规数据就是用户的数据。存储这类数据的表称为常规表。

·临时数据　这类数据是系统进行关系算术运算时所产生的中间结果，这些数据保存在临时表中，因下一个事务启动时并不需要它。此外，在跟踪系统行为时也可产生临时数据。

关系型数据库管理系统的这些特色在 PostgreSQL 中体现得淋漓尽致，PostgreSQL 系统中用到的一切数据都存储在表中。

目前，最基本的存储介质就是磁盘。在 PostgreSQL 中，大部分的表都以磁盘文件的形式存储在磁盘介质中。用文件系统来存储表会遇到一个问题：一个表可以很大，但文件的大小通常受支持平台的限制，如何在受限的文件系统上支持大表？PostgreSQL 采取的策略是：将表分割成若干部分后再存储，每个部分就成为表的一个物理段文件(relation segment)，所有这些段文件组成表整个逻辑文件，如：常规表磁盘文件、临时表文件(BufFile)。一般地，除了最后一段文件外，所有的段文件都有固定大小(40Mb)。表的读入和写出操作就在这些段文件之间进行。为了读/写表逻辑文件中某一个磁盘块，首先必须给它定位，即计算出它所在的物理段文件和段文件内的偏移量，然后打开特定段之前的所有段文件，最后才是执行读写操作。PostgreSQL 通过低级磁盘文件操作把表的数据写到磁盘上。

还有一类表，被保存在稳定或不易挥发的主存中。这种存储仅涉及主存访问而不涉及外存的访问。PostgreSQL 目前的实现是开辟出一片共享内存区来模拟稳定或不易挥发的内存来存储着这种类型的表。在这片共享内存中，存储了表的描述符和它的内容。表的内容可以通过索引哈稀表来实现迅速的访问，也可以被顺序访问。

3　存储文件系统接口

在 PostgreSQL 的实现机制中，一个表对应于一个逻辑文件，而一个逻辑文件又是由若干个物理文件构成的。就逻辑文件本身而言，它是没有对应的磁盘文件的；但从用户的角度来说，自然习惯于基于整个表的整体访问操作，而不管实际读写的是表的哪一个物理段文件。这就要求 PostgreSQL 提供某种程度的抽象，屏蔽掉物理段文件的操作细节，展现在用户面前的是整个表逻辑文件的读写接口，这些接口就构成了 PostgreSQL 的存储文件操作机制(Storage manager)。

另一方面，由于表的访问最终靠文件操作来完成，这样一个后端进程在进行数据操作时很可能因支持平台的每个进程打开文件数的限制而崩溃。为了避免这种情况的发生，PostgreSQL 有必要监视后端打开的文件数，这是 PostgreSQL 虚拟文件操作机制的功能。它把一个后端所有打开的物理文件描述符组成一个 LRU 链进行统一管理。存储文件操作必须通过这一接口，虚拟文件系统接收到存储文件操作的申请后判断它所在的后端进程打开的文件数是否已威胁到了平台每进程打开文件数的约束；如果是，则回收 LRU 尾部的文件描述符，最后才是通过平台提供的低级文件操作机制完成文件操作。这样就有效地解决了平台资源限制的问题。PostgreSQL 存储文件操作机制如图1所示。

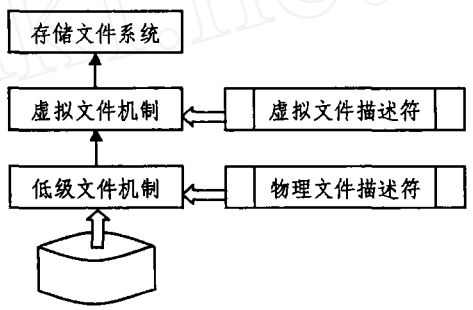

图1　Postgre SQL 存储文件操作机制

4　存储文件页存储格式

在 PostgreSQL 的实现中，表物理段文件的基本存储单位是文件页(磁盘块)。文件页的大小限制了表元组的大小并影响到磁盘操作效率，因此文件页不能太小，目前系统缺省采用的文件页是8192字节，最大可被设为2^15字节，这是由磁盘块索引 lp_off 和 lp_len 所决定的，它们都是15位宽。文件页也是内存和磁盘的交换单位。

一个文件页空间被逻辑地分割成三个部分：页描述区(PageHeader)、元组数据空间(Tuple Item space)以及特殊空间(Special space)。页描述区记载了页的使用情况，如页分布格式版本，元组数据空间和特殊空间的起始位置以及文件页相关的事务日志记载点等信息。元组空间是实际记录元组数据的地方。每个被记录的元组称为一项，每项由描述 id 和元组数据构成，项描述 id 描述了元组存储位置、大小以及一些状态标志。项描述 id 和项数据分别在元组数据空间的两头往中间靠拢存放。PostgreSQL 文件页分布示意如图2所示。

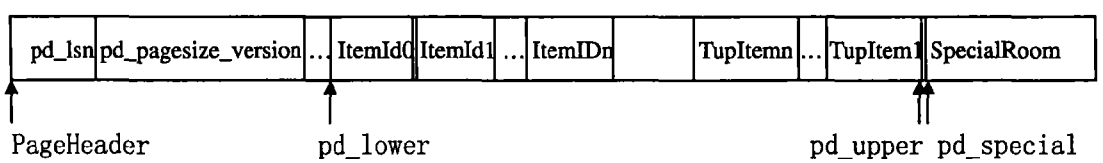

图2　PostgreSQL 文件页分布

写元组时，首先要把它写到文件页的内存缓冲区(Buffer)中，这按以下步骤进行：首先，从缓冲页的元组数据存储区中分配空间；然后，构造元组描述项 id 并把它写入缓冲页的元组数据空间的低端处；最后才是把元组实际数据写到缓冲页的元组数据空间的高端处并设置缓冲区的脏标记。写到缓冲区中的元组并不立即更新到磁盘中，物理写元组被推迟到它所在的缓冲区被替换(Replace)时进行。当替换缓冲区时，首先判断缓冲区是否脏，如果脏的话，就要实际启动磁盘 IO 进行写了；如果不脏，该缓冲区直接被回收再用。物理写文件页时，首先要更新该文件页相关的事务日志，事务日志

由页头部的页描述符指出；然后把文件缓冲页写到指定的磁盘块中。详细的缓冲区机制参见第6节。

5 存储文件并发访问控制

存储管理是数据库管理系统实现的一个核心部分。它往往需要支持多个后端并发访问，甚至要支持多CPU环境、分布式环境的并行访问，所以并发访问控制又成为数据库存储管理系统的关键核心。由于PostgreSQL对分布环境的支持有限，它的并发访问机制就要简单得许多。PostgreSQL实现存储文件并发访问控制的基本方法是通过使用锁来控制磁盘文件操作——临界区——的互斥访问。后端进程要对磁盘文件进行访问操作时，首先要获取锁：如果成功获得，则进入临界区去执行磁盘读写访问，访问完后退出临界区并释放锁；否则，进程睡眠直到被别的后端唤醒后重试。在PostgreSQL实现中，使用到三种类型的锁[1]：

·旋转锁(Spinlock)　这种锁的作用时间应非常短。如果锁被持有的时间要超出许多指令甚至要跨内核调用(或甚至是调用非第三方的子例程)，则不要用旋转锁[1]。旋转锁主要是用作轻量级锁的基础设施。旋转锁的实现是通过使用硬件的原子测试/设置指令来实现的，等待的进程忙循环等待直到它们可以得到锁为止。它不提供死锁监测、错误原子释放。如果锁在大约1分钟的时间内不能得到，则超时处理——取消本事务。

·轻量级锁(LWLocks)　这些锁典型地被用于共享内存中的数据结构的互斥访问。轻量级锁支持排他和共享模式(相应用于共享对象的读写访问和只读访问)。轻量级锁不提供死锁检测，但轻量级锁管理器在elog恢复期间被自动释放，所以持有轻量级锁的期间发出错误是安全的。当没有锁的竞争时获得或释放轻量级锁是相当快的(几十的指令[1])。当一个进程不得不阻塞在一个轻量级锁上时，它阻塞在一个SysV信号量上，所以不会消耗CPU时间。等待的进程将会以先来后到的顺序被授予锁。这里没有超时概念。

·正则锁(regular lock)(重量级锁)　支持表驱动语义的多种类型锁模式，且它有完整的死锁检测和事务结束后的自动释放功能。正则锁应用于所有用户驱动的锁请求[1]。正则锁通过使用轻量级锁来实现。

PostgreSQL正则锁粒度可以是数据库的表、元组、页等，请求锁的既可以是后端事务，也可以是跨事务的会话。支持的锁模式有八种，按排他级别从低到高的顺序分别是：(1)访问共享锁，用于查询；(2)行共享锁，用于为更新查询；(3)行排他锁，用于插入/更新/删除；(4)共享更新排他锁，用于清理(非全部)；(5)共享锁，用户创建索引；(6)共享行排他锁，类似排他模式，但允许行共享；(7)排他锁，阻塞行共享/为更新查询；(8)访问排他锁，用户改变表/丢弃表/全清理/不受限锁表。

主后端启动时，分配一块共享内存区作为正则锁方法表(lockmethodtable)并适当地初始化正则锁方法表的各个域，此外还有初始化锁方法数组和其他一些共享变量。后端启动后通过锁方法来引用正则锁。目前，PostgreSQL用到系统锁方法(lockmethod = 0)和用户锁方法(lockmethod = 1)两种方法，它们都指向同一个正则锁方法表项。

正则锁获得(LockAcquire)方式：首先，要判断指定锁方法lockmethod要有效，在有效的前提下获得用到的正则锁方法表项lockmethodtable的同步信号量，然后寻找/创建锁的信息对(锁信息lock哈稀表项，锁持有者lockholder哈稀表项)，并更新锁信息lock中的锁模式请求值。然后判断持有者lockholder之前是否已获得过该锁模式或本进程跨事务持有过，如果持有过，则授予锁；否则，判断请求的锁模式是否同锁的等待者请求的锁模式冲突，如果冲突的话，则把该事务进程连入锁的等待者列；否则判断请求的锁模式是否同锁已授予的锁模式或别的等待者的已获得的锁模式是否有冲突，如果冲突则连入等待者列；否则，授予事务请求的锁并更新锁信息和锁持有lockholder的有关域。否则让进程可中断地睡眠直到被唤醒获得锁或被中断为止。所有不能得获得锁的情况下，都要返回申请失败信息。

正则锁的释放方式：首先，要判断指定锁方法要有效，在有效的前提下获得锁方法表的同步信号量，然后寻找/创建锁的信息对(锁信息哈稀表项，锁持有者哈稀表项)，并更新锁信息中的锁模式请求值和授予值以及锁持有信息的持有值。然后唤醒等待者中可以被唤醒的所有等待者。

注意：正则锁的授予和释放都遵循先来后到的原则。请求正则锁时，不得与锁之前的等待者的请求锁模式或已授予的锁模式冲突，否则进程要睡眠等待。释放正则锁时被唤醒的等待者不得与之前仍须等待者的请求锁模式或已授予锁的模式相冲突。

旋转锁或轻量级锁的申请失败后将导致查询被撤销和死去。然而，所有这些限制在正则锁中不再存在[1]。也要注意在等待正则锁时我们可以接受查询撤销和死去中断，但在等待旋转锁或轻量级锁时我们不接受它们。因此，当等待时间超出几秒时使用轻量级锁是不好的。PostgreSQL锁机制的实现如图3所示。

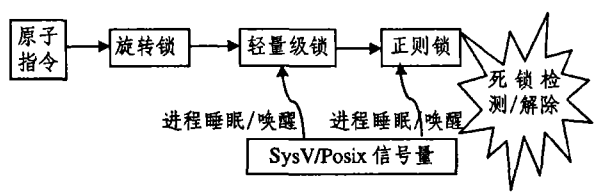

图3　PostgreSQL锁机制

6 表文件缓冲区管理

PostgreSQL有两种类型的缓冲区：全局缓冲区和本地缓冲区。全局缓冲区主要用作普通可共享表访问空间；本地缓冲区为仅本地可见的临时表的提供操作场所。下面对全局缓冲区的实现作详细介绍。

为了实现方便，PostgreSQL对缓冲区的管理基本上采取了静态方式：它在系统配置时规定好了缓冲区的总数，以后在每次系统启动时，由主后端或某一独立的后端从共享内存中分配一片空间用作缓冲区，全部(全局)缓冲区构成一个缓冲池。缓冲池由缓冲区管理跟踪BmTrace区(可选)、缓冲区描述符BufferDestriptors区、缓冲区以及缓冲区索引哈稀表构成，它们的构成关系如图4所示。

缓冲区管理跟踪区主要用于记录缓冲区被使用的情况，它共有BMT_LIMIT项。缓冲区是实际存储数据的地方，它共有NBUFFERS个。所有的缓冲区都有一个描述它的缓冲区描述符，它有NBUFFERS+1个，第NBUFFERS个(从0开始)缓冲区描述符不实际描述缓冲区，它仅用于作为空闲缓冲区链头，通过它所有空闲的缓冲区被连到一个双链表中。要注意每个缓冲区描述符都是按顺序描述一个对应缓冲区的，在

它们的描述结构中对应缓冲区号和缓冲区共享内存偏移量都是在缓冲池初始化期间被设置好，以后不再更改。此外，为了加速从被缓冲的表到对应缓冲区的搜索，在缓冲池建立的期间还建立了一个缓冲区哈稀表，以后每个被申请用来缓存表的磁盘块的缓冲区都要在该哈稀表创建一项，这样只要给出表的文件节点(RelFileNode)和磁盘块就可以迅速地搜索到缓冲指定文件页的缓冲区，而不要顺序搜索所有的缓冲区。

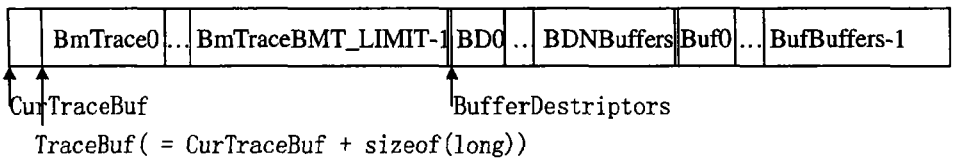

图4 全局缓冲池构成

由于这部分的缓冲区是共享的，因此必须支持缓冲区的互斥访问。PostgreSQL 实现缓冲区互斥访问的基本机制是使用轻量级锁，缓冲区的读写访问大致要经过以下两步：

第一步：进程要访问缓冲区必须通过访问它的描述符，这是通过缓冲区管理轻量级锁(BufMgrLock)来实现对缓冲区描述符的互斥访问；

第二步：如果缓冲区要读入/写出缓冲区，必须获得 IO 轻量级锁(io_in progress lock)；如果要写出表页内容，则必须获得文件页上下文刷新锁(cntx_lock)。

除了通过锁来实现缓冲区以及它对磁盘 IO 操作的互斥访问外，PostgreSQL 还要踪缓冲区被引用的次数以辅助缓冲区的申请和释放操作。每个事务要引用缓冲区时必须先取得对它的钉(引用 refcount)，访问已拥有的缓冲区时则只需增加私有引用(PrivateRefCount)，即，通过缓冲区的引用计数和本后端对缓冲区的私有引用共同来跟踪后端对缓冲区的引用。图5是缓冲区访问大致示意图。

图5 缓冲区访问示意

由于表文件缓冲区是后端进程访问表的直接入口，它处在 PostgreSQL 存储管理系统的最顶端，因此有必要对缓冲区的重要操作做出说明：

·缓冲区的申请(BufferAlloc)策略 该模块的基本功能是给定待缓冲的表文件某一页(表文件节点，块(页)号)申请一块缓冲区；如果缓冲区脏的话，还要刷新它。处理流程伪代码说明如下：

```
首先判断表中指定的磁盘块(表文件节点，块(页)号)是否已在缓冲区中，如果缓冲区中已有该磁盘块的缓冲{
  则钉住它然后判断该缓冲区是否在 IO 中，如果是{
    则等待该缓冲区读操作完成并返回缓冲区，此情况下如果读操作失败则要给缓冲区开始 IO；
  }
}
如果不存在{
  取得一个新的缓冲区；
  如果该缓冲区脏则{
    把缓冲区的内容写到磁盘文件中；
    判断在我们刷新磁盘内容的时候是否有别的进程读入我们所希望的磁盘块，如果是则{
      释放原申请的缓冲区，使用别的进程申请的缓冲区，以后同有磁盘块缓冲区情况一样处理；
    }
    如果不存在这样的缓冲区{
      判断申请到的缓冲区在我们写的过程中是否已被别人引用，如果是{
        同样释放它再次申请一个空闲缓冲区并重复前面相关操作；
      }
    }
  }
  如果到了这里，则我们申请到了一块空闲的缓冲区，我们先复位它，然后给它创建满足我们所需的哈稀表项后返回；
}
```

·读缓冲区(ReadBuffer)策略 该模块的基本功能是读入表文件中指定的页(表文件节点，块(页)号)内容到缓冲区中；缓冲区读采取即时读策略。处理流程伪代码说明如下：

```
首先分配一个缓冲区；
判断缓冲区内容是否是待缓冲的表文件页(表文件节点，块(页)号)，
  如果是{
    这最好，我们只需简单返回；
  }
判断希望的块是否是表不存在的待要扩展的块，如果要扩展{
清空我们分配到的磁盘块然后把表文件扩展一块；
  返回缓冲区描述符；
}
从磁盘中读入指定的表块；
返回缓冲区描述符；
```

·缓冲区写(WriteBuffer)策略 这里采取“懒惰写”策略，即在缓冲区被替换时才写出去。缓冲区写时，我们仅需要标记缓冲区脏和恰好脏标记，然后返回。

总之，共享缓冲区的访问遵循以下原则：(1)访问缓冲区要求首先钉缓冲区(增加引用计数)；(2)缓冲区描述符详细描述了缓冲区的状态，它的访问要求获得缓冲区管理轻量级锁；(3)缓冲区的 IO 操作要求记录缓冲区的 IO 状态并要获得相应的 IO 锁或文件页上下文刷新锁。

由于本地缓冲区的可见范围仅在本后端中，因此对它的访问不涉及互斥问题，只须跟踪引用计数，其他读写思想同共享缓冲区。

7 PostgreSQL 存储管理体系结构

存储管理主要目标就是提供创建、存储以及访问数据库表文件的接口[4]。PostgreSQL 充分利用平台提供的文件系统机制来实现它的存储管理。它引进虚拟文件系统来解决操作平台对每个进程允许打开文件数的限制；通过共享内存机制来实现表文件的跨事务访问；通过缓冲表文件(BufFile)(主要针对临时表)、存储表文件(StorgeFile)以及主存贮存表(MainMemoryFile)系统来实现对表文件的操作，同时分段存储的思想突破了平台文件大小限制对表实现的约束。此外，为了辅助表文件的操作，它还建立了表文件页空闲空间映射和表文件页操作的机制，这都极大地方便了对表文件的访问。PostgreSQL 存储管理实现体系结构如图7所示。

在 PostgreSQL 整个存储管理体系结构中，表文件缓冲区管理处于最顶层，它提供外部访问表的接口。后端一般通过表文件缓冲区来访问表文件。缓冲区调用下面存储文件机制，而各类存储表操作又调用虚拟文件系统，最后由虚拟文件系统完成磁盘操作。另外，缓冲区的共享访问特性要求它有并发控制机制，这部分功能由锁机制提供。

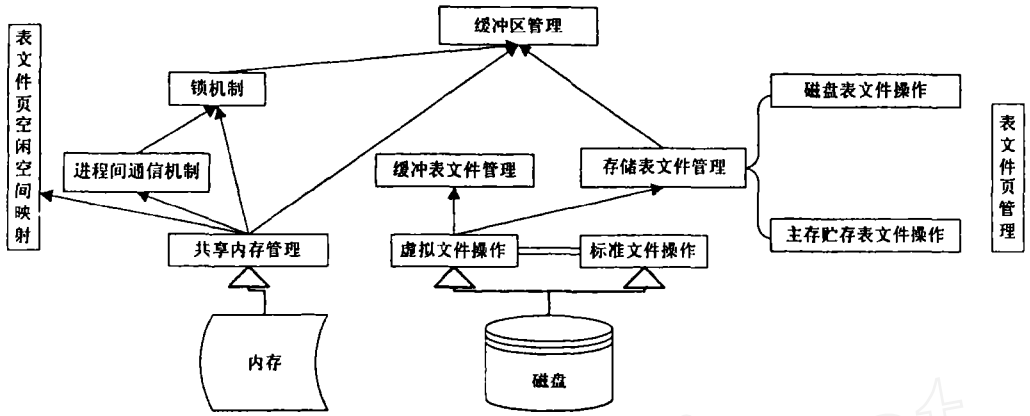

图7 PostgreSQL 磁盘管理体系结构

结论 通过对 PostgreSQL 存储管理实现机制的研究我们可以得出如下结论：

·PostgreSQL 存储文件系统的操作本身不涉及多个进程的并发访问，所以它的操作必须严格串行进行；

·后端可以并发访问的地方是共享文件缓冲区，它是存储表数据的内存映像，所以必须对它的访问操作进行精确的并发控制。PostgreSQL 采取两步走：(1)对于缓冲区本身的访问由于涉及的时间相对短，这部分的并发控制依靠轻量级锁来实现；(2)在替换缓冲区时要把缓冲区的内容刷新到磁盘文件中，由于 PostgreSQL 磁盘文件系统本身没有并发控制，因此也要在缓冲区层面来实现临界区的并发访问；再者，由于我们更新的是表文件的一个页面，花费的时间也不会太长，这部分的控制也依靠轻量级锁机制来实现；

·磁盘管理辅助事务管理的思想体现在：在写磁盘块时，总是先写事务日志后写数据；

·由于 PostgreSQL 基本运行环境是单机环境(允许多 CPU)，因此它的锁机制实现的基点是：运行环境有：(1)共享内存；(2)统一的时间定位机制。

PostgreSQL 的存储管理机制是建立在支持平台的文件系统上的，它的实现效率、可靠性等在很大程度上取决于支持的文件系统。如果把文件系统的某些功能直接由数据库存储管理来实现而不要通过调用平台的文件系统接口，那么存储系统的性能可望得到提高。

参 考 文 献

1 PostgreSQL Development Group. PostgreSQL V- 7. 3. 4 source codes. PostgreSQL website http:www. Postgresql. org,2003

2 PostgreSQL Development Group. PostgreSQL V- 7. 3. 4 Documentation. PostgreSQL website http:ww-w. postgresql. org,2003

3 Stonebraker M. The design of the post- gres stroge system. EECS Department Univers-ity of California Berkely,Ca,94720,1987

4 Garcia-Molina H, Ullman J, Widom J. Database System Implementation. Prentice Hall,2001

5 Ullman J, Widom J. A First Course in Dat-abase sytems. Prentice-Hall,1997

6 何伟平. PostgreSQL 的昨天今天和明天——自由软件数据库 PostgreSQL 简介 v2. 0. PostgreSQL 中文网站. http://www.pgsqldb. org,2003

(上接第67页)

3 Liu Q,Yuan Y,Lin X. Multi-resolution Algorithms for Building Spatial Histograms. In:Proc. Fourteenth Australasian Database Conf. (ADC2003), Pages145～151

4 Wu Y, Agrawal D, El Abbadi A. Query Estimation by Adaptive Sampling. IEEE ICDE, 2002

5 吴胜利. 估算查询结果大小的直方图方法之研究. 软件学报, 1998,9(4):285～289

6 Aboulnaga A S,Naughton J F. Accurate estimation of the cost of spatial selections. In:ICDE'00, Proc. of the 16th Intl. Conf. on Data Engineering, March 2000. 123～134

7 Jin J,An N,Sivasubramaniam A. Analyzing range queries on spatial data. In:ICDE'00, Proc. of the 16th Intl. Conf. on Data Engineering, March 2000. 525～534

8 Aref W,Samet H. A Cost Model for Query Optimization Using R-Trees. In:Proc. of ACM GIS, Gaithersburg, Maryland, Nov. 1994. 60～67

9 An N,Yang Z Y, Sivasubramaniam A. Selectivity Estimation for Spatial Joins. In:ICDE'01, Proc. of the 17th Intl. Conf. on Data Engineering, April,2001. 368～375

10 梁中,孙小燕,谭勇桂. 空间索引技术-回顾与展望. 计算机工程与应用,2002,24:197～199

11 Kamel I,Faloutsos C. On Packing R-tree. In:Proc. of the CIKM, 1993. 490～499

12 Theodoridis Y,Sellis T. A Model for the Prediction of R-tree Performance. In: Proc. 15th ACM PODS Symposium, 1996. 161～171

13 Theodoridis Y, Stefanakis E, Sellis T. Cost Models for Join Queries in Spatial Databases. IEEE ICDE, 1998

14 Lo M L,Ravishankar C V. Spatial Joins Using Seeded Trees. In: Proc. of ACM SIGMOD Conf. 1994

15 Lo M L,Ravishankar C V. Spatial Hash-Joins. In:Proc. of ACM SIGMOD Conf. 1996

16 Faloutsos C, Sellis T,Roussopoulos N. Analysis of Object Oriented Spatial Access Methods. In:Proc. of the ACM SIGMOD/PODS Int. Conf. on Principle Of Data, 1987

17 Orenstein J A. Spatial Query Processing in An Object-Oriented Database System. In:Proc. ACM SIGMOD Int. Conf. on Management of Data, May,1986. 326～336

18 Wu Y L. Query Result Estimation in Database:[Doctoral Dissertation]. University of California Santa Barbara,2001

19 Belussi A, Faloutsos C. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. VLDB 1995. 299～310

20 Schroeder M. Fractals,Chaos,Power Laws:Minutes From an Infinite Paradise. W. H. Freeman and Company, New York, 1991

21 Faloutsos C, Seeger B, Traina A, Traina C Jr. Spatial join selectivity using power laws. In: Proc. of the 2000 ACM-SIGMOD Conf. Dallas,TX,May 2000. 177～188