浅析 Postgresql 存储管理中共享缓冲区的管理

□ 张雅洁

(温州职业技术学院计算机系 325000)

无论对于什么数据库来说,存储部分都是整个数据库的基础,而其中的缓冲区管理作为沟通数据库中各部分的桥梁,可以看作是一个 DBMS 的活动中心,对数据的所有操作,都要作用驻留在缓冲区的信息之上。概括的说,缓冲区管理就是对系统缓冲区的调度和管理,包括缓冲区与外界交换数据,以及对其中的数据进行操作时,缓冲区各项属性的变化。

PostGresql 作为一个大型的数据库,它的缓冲区分成了两种类型:shared 缓冲区 和 local 缓冲区。其中 local 缓冲区 是用来处理临时表的,用到的比较少,处理过程相应的也比较简单,对其中的数据进行处理的过程中不需要持有锁,也不需要登记日志或设置检查点(checkpoint);数据库中大部分数据的处理经过的都是 shared 缓冲区,对它的处理相应要求比较复杂。我们一般所说的缓冲区 指的都是这种 shared 缓冲区。

在 PG 数据库中同时提供了对这两类缓冲区 的支持,因此我们也将对这两类缓冲区分。

别进行分析。PG 对 shared 缓冲区 和 local 缓冲区 在标记上予以区分,shared 缓冲区 的 id 是从 1 开始递增的正整数,local 缓冲区 的 id 是从-1 开始递减的负整数,如果 id 为 0 则该缓冲区是非法的。

缓冲区部分的代码主要是提供给数据库各部分的接口,数据库的各个部分将根据各自的策略调用这些接口函数。

下面介绍共享缓冲区管理中用到的一些原则策略和算法。

1.共享磁盘缓冲区有两种不同的存取控制机制

a、reference counts: 用来记录目前访问缓冲区 的进程的个数, 当值为 0 时表示缓冲区 空闲可以被插入 freelist。

b、缓冲区 content lock。

2. Pins

Pin 和 unpin 针对共享缓冲区,pin 表示访问缓冲区, 使其不能加入 freelist。Unpin 表示解除对缓冲区 的 pin 使其能够加入 freelist。进程在操作缓冲区 前必须获得该缓冲区 的一个 pin(refcount+1)。一个 unpin 的缓冲区可在任何时刻被重新声明重新使用来装载一个新页面。一个 pin 通过 Read 缓冲区()来获得,通过 Write 缓冲区(如果页面被修改)或 Realse 缓冲区(如果页面未修改)来释放。对一个单独的后端并发的对同一页面 pin 多次是允许的,并是十分常见的,缓冲区 manager 可以对这种情况有效处理。持续的持有某页面的 pin 也是允许的,例如:顺序扫描持有当前页的 pin,直到该页面上的所有元组均被处理过。才释放 pin。如果扫描是一个连接的外扫描,时间将会比较长。类似的,b 数索引扫描会持有当前索引页的 pin,这是允许的,因为通常的普通操作可以在 pin 的计数非零时执行。Pin 在事务边界将不能被保持。

3.缓冲区内容锁(缓冲区 content lock)

有两种缓冲锁,共享的和互斥的。多个后端可以持有同一缓冲区 的共享锁, 但互斥锁防止任何其他的进程持有共享或互斥锁 (也称 Read 或 Write 锁)。这些锁均为短时锁,不能被长久的持有。缓冲区 l ocks 通过 Lock 缓冲区 申请和释放。单个后端在同一缓冲区上持有多个锁是不允许的。一个进程在 lock 缓冲区 之前,必须先持有缓冲区 的 pin(增加 refcount)。

4.缓冲区的存取原则

(1)扫描页面上的元组必须首先 pin 该页面,并且获得共享或互斥锁,为了检查共享缓冲区上一个元组的提交状态(XID 和状态位),该进程必须同样的获得 pin 和共享(互斥)锁。

(2)一旦一个进程认为某元组是 interesting(对当前事务可见)的,该进程就可以丢弃内容锁,然后继续存取元组数据,直到放弃缓冲区 的 pin 为止。堆扫描的典型做法就是这样的。因为 heap_fetch 返回的元组包含一个指向共享缓冲区 中元组数据的指针。因此只要持有 pin,元组就不会丢失。元组状态可以改变,但是我们假设在初始可见的决定做出后对操作无影响。

(3)增加一个元组或改变已存在元组的 Xmin/Xmax 值时,必须持有包含该元组的缓冲区的 pin 和互斥内容锁。这保证其他进程在做可视检查时不会看到一个元组半更新的状态。

(4)在只获得一个缓冲区 的共享锁和 pin 的条件下,更新元组的提交状态位是允许的。

因为另一个同时正在查看该元组的后端将和这个状态位做或运算。因此冲突更新几乎没有任何危害。更甚者,如果必有冲突,也很少意味着一个位更新将丢失,需要以后重做。这个标志位都只是提示(他们在 pg_clog 中缓存事务状态查找表的结果),因此如果由于冲突更新,他们的位被重置为 0,也不会有大的危害。

(5)为了物理上移走一个元组或紧凑页面上的空闲空间,进程必须持有 pin 和互斥锁,并且遵守当持有互斥锁时,缓冲区 的共享引用计数为 1,即没有其他后端持有 pin。

5.缓冲区置换策略

Freelist 中的缓冲区时替换的最初候选者。特别的, 完全空闲的缓冲区(不包含有效页)总在该链表中。我们也可以将某些包含短期内不会被使用页的缓冲区放入 *freelist*。该链表是单链表(使用缓冲区头的域)。该链表的表头和表尾指针被声明为全局变量。虽然链表的链接点是 bufferheader,但他们通过 BufreelistLock,而非 bufferheader spinlock 保护。当没有空闲缓冲区可用时,我们使用简单的时钟扫描算法来选者一个缓冲区替换出其页面来再次使用。时钟扫描算法避免在普通操作过程中获取系统级的锁。

他们的工作机制如下:

每一个缓冲区头包含一个使用计数,当 buffer unpin 的时候,使用计数加 1 (增至一个较小的上限)。该操作只需获得 bufferheader 的 spinlock, 获取 spinlock 将减少 buffer 的引用计数。时钟指针是一个缓冲区索引,NEXTVictimBuffer 在所有可用的缓冲区上循环移动。NEXTVictimBuffer 环 BufFreelistLock 保护。

Clock 算法如下:

(1)获取 BufFreelistLock

(2)如果 BufferFreelist 非空,移出表头的缓冲区。如果该 buffer 被 pin 或有一个非零的 usage count,则它不能被使用。忽略之,然后返回第二步。否则, pin 该 buffer,释放 BufFreelistLock,返回该 buffer。

(3)当 freelist 为空时,选择被 NextVictimBuffer 所指的 buffer。下次循环前移 NextVictimBuffer。

(4)如果选中的 buffer 被 pin 或有一非零的 usagecount,该 buffer 不能被使用。减少其 usage count

(5)然后返回第三步检查下一个 buffer。

pin 选中的 buffer ,释放 BufFreelistLock.返回该 buffer(如果选中的 buffer 被标记为脏也,在我们重新使用它之前要将其先写回磁盘。如果其他进程钉住了该 buffer,我们将放弃该 buffer,重新尝试另外的 buffer。这不是选者 victimbuffer 的关键。

有一个特殊规定, 当运行 VACUUM 时, 一个后端并不增加他所存取的 buffer 的 usagecount。实际上,如果 RealeaseBuffer()发现一个缓冲区的 pincount 和 usagecount 均为 0,那么他将该 buffer 追加到 freelist 的尾部。(这意味着 VACUUM 且仅 VACUUM 在 RealeaseBuffer 期间必须持有 BufFreelistLock。这不会造成一个了不起的竞争问题。)这项规定使 VACUUM 运行在一个数量相对较少的缓冲区之上, 而不是整个的缓冲区之上。因为一个只被 VACUUM touch 的页面在短期内将不会再被使用,因此这项规定是合理的。

既然 VACUUM 通常能很快地请求很多页面, 因此它也能在下次调用时收回他所填充的或更改的那些 buffer。并且能使用其在 cache 中找到的信息在共享内存缓冲区中完成其任务。,

浅析Postgresql存储管理中共享缓冲区的管理

作者：张雅洁
作者单位：温州职业技术学院计算机系,325000
刊名：科协论坛（下半月）
英文刊名：SCIENCE & TECHNOLOGY ASSOCIATION FORUM
年，卷(期)：2007，(7)
引用次数：0次