DBMS 中索引的并发控制

Concurrency Control of Index Structures in DBMS

冯玉才　张黎敏　曹忠升

（华中理工大学计算机科学与技术学院　武汉430074）

Abstract　Indices are used most frequently in DBMS. As a result, they will become point of great data contention, leading to a low degree of concurrency. This paper proposes a concurrency control method aiming at B^+-tree index, which is based on the combination of Multiversion and Optimistic Method (MO), and puts it into practice.

Keywords　Index, MO method, Multiversion control, Optimistic method, Transaction

一、引言

并发控制是 DBMS 中关键的一项技术，其所采用的算法直接影响着 DBMS 的效率。并发控制涉及到很多对象，包括数据字典、普通数据、存储过程以及索引等等，其中，索引的并发是随着并发粒度的下降而提出的。并发粒度是衡量 DBMS 并发效率的重要指标，它指的是 DBMS 中支持并发存取数据的最小单位，例如表级并发粒度意味着多个事务可以同时对不同的表更新而不需等待，但对同一张表的更新需等待。并发粒度越小，系统的效率越高。在八十年代，各种流行的数据库管理系统都采用表级并发粒度。在其控制下，多个用户会由于存取同一张表而阻塞。随着技术的进步、数据库应用需求的发展，表级并发控制的效率太低，已不能满足用户的需要，高效的并发度应此要求诞生。

元组级的并发提出了一个新课题：索引的并发。在 DBMS 中，索引是针对表创建的。当并发粒度在表级时，事务对表的更新是串行的，当然，对索引的更新也是串行的，因此，在这种情况下，不会发生两个事务对同一索引结构进行更新的情况，也不存在索引的并发问题。一旦并发粒度降到元组级，多个事务可以同时对同一张表的不同元组进行更新，对其索引结构的更新也会同时进行。索引一般都是按照 B 树系列的数据结构组织的，B 树的插入和删除在一定条件下会涉及很多节点，如果两个以上的事务同时对同一 B 树插入和删除而不进行任何控制，那么 B 树的正确性难以得到保证。我们以 B^+ 树索引为例说明并发存取索引引起的问题。

图1所示的是一个3阶 B^+ 树。假定有两个并发事务 T_1、T_2，它们分别在 B^+ 树上做如下操作：T_1：删除值20和 T_2：插入值15。假定事务 T_1 先执行。它搜索到20所在的结点 q 后，线程发生切换，另一事务 T_2 开始执行，它首先查找15，发现将要插入15的结点 q 已满，则申请一新结点，原来的结点 q 中包含13和15，新结点包含20。此时，线程发生切换，事务 T_1 将会错误地删除15。

上例只说明了分裂的情况，同样，合并也会导致索引结构的变化，从而在事务切换时引起不可预知的错误。

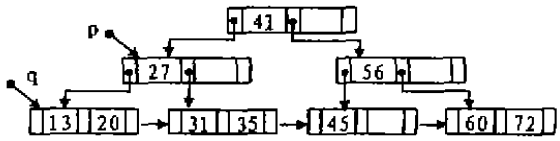

图1　B^+ 树示意图

因此，当并发控制粒度降到元组后，索引的并发控制势在必行。为此，我们结合多版本控制和乐观方法两种机制，提出一种新的索引并发控制策略 MO（即 Multiversion and Optimistc Method），它采用乐观方法的控制思想和多版本控制的实现技术，高效地解决了元组级并发控制中索引的并发问题。

二、相关概念和知识

本文所设计的算法是在 DM2[2] 基础上实现的。因此，本文将涉及到 DM2中的一些概念，下面分别做简要介绍：

1）数据块　用于存放基表中的一个个元组，一个数据块可存放多个元组，但一个元组不能跨块存放。数据块是系统存取数据的基本单位。

2）B^* 树　B^* 树是传统 B^+ 树的改良，它在叶结点上因插入一个索引项溢出或因删除一个索引项需合并时，在临近左右 n 个叶结点上重新分配索引项，从而减少了块分裂和合并的机会，提高了空间利用率。为了提

冯玉才　教授、博士生导师，研究领域为数据库。张黎敏　硕士研究生，研究方向为数据库。

高查询速度、基表实例数据和在基表上创建的索引都按照性能优秀的 B^+ 树进行组织，DM2中存在三种索引：主关键字索引、聚集索引和普通索引[2]。

3）元组标识符 TID　元组一旦放入一数据块中，它就有了一个确定的编号，称为元组标识符 TID。每个元组都有自己的唯一的 TID，同时，TID 也标志了它在数据库中的物理位置，它在元组插入到数据块时由系统自动生成。

三、并发控制策略

索引的并发控制要解决两个问题：第一，如何并发操作索引；第二，如何保证索引的正确性。我们用乐观方法解决第一个问题：在事务对索引并发操作时，总认为它们间不存在索引冲突，不用锁机制等方法间断事务的运行，从而保证事务在元组级的并发性；对于第二个问题，我们用多版本控制的技术予以解决。

3.1　索引的并发

一般而言，数据的并发是基于对数据的控制来实现的。对于普通数据，广泛采用封锁算法进行控制。其基本思想是要求事务在对一数据项操作之前必须首先申请对该数据项的封锁，获准后方能进行操作。如果该数据项已经被其他事务封锁且操作方式冲突，那么该事务必须处于等待状态，直到该数据对象被释放为止。索引数据相对于普通数据而言，有诸多不同之处，以 B^+ 树[1]索引为例，其特殊性表现在三个方面：

1）多个索引值共处一个索引节点；

2）一个非唯一索引值对应多个元组；

3）插入、删除一个索引项可能影响多个索引节点。

性质1）决定了不能对索引节点封锁：如果封锁对象是索引节点，那么一个元组的操作会封锁整个其索引值所在的索引节点，造成同一索引节点上其它索引值对应的元组不可操作；

性质2）决定了不能对索引值封锁：如果封锁对象是索引值，那么一个元组的操作会封锁其索引值，这一封锁将造成有相同索引值的其它元组的不可操作，另外，封锁对象过小，会增加系统开销、降低系统效率；

以上说明了封锁对象无法确定，性质3）则说明了封锁对象的动态变化性，除非封锁整个索引树，否则需动态跟踪更新的索引节点，这无疑增加了控制的复杂度。

通过以上讨论，我们得出结论：索引的性质决定了封锁机制不适于控制索引的并发。因此，我们采用乐观方法控制索引的并发。乐观方法对事务运行时发生的存取冲突不是立即加以处理，而是让事务继续运行，待事务结束时再检验冲突、解决冲突。基于以上思想，我们控制索引存取的做法是：不封锁索引块，让事务顺利通过对索引块的访问。此做法适应了索引的三个特殊性质：事务对某个索引结点 B_i 更新时，不必关心其它事务是否正在操作 B_i 和将会对 B_i 做什么操作；当结点 B_i 需分裂、合并或与左右兄弟交换索引项而引起一个索引结点集合 S_{B_i} 的变动时，也不必关心其它事务对 S_{B_i} 的任一子集中的索引结点所做的操作。在这种控制之下，元组级的并发粒度得以实现。

3.2　索引的正确性维护

不封锁索引固然提高了并发度，但同时引起诸多问题，其中最主要的问题是多个事务同时操作索引导致索引结构的破坏，这更增加了索引正确性维护的难度。

我们采用多副本方法解决以上问题：在每个事务中设置私有缓冲区，通过私有缓冲区将事务各自更新后的索引数据隔离，并及时刷新无效索引块。具体做法如下：当事务需更新某个元组时，它先读取元组所在的数据块 B_i，做完更新操作后，并不把更新过的数据块（记为 B_i'）写入数据库，而是把 B_i' 链入事务的更新链中。当事务读取数据块时，首先从更新链中读取，若更新链无此数据块记录时，再从数据库中读取。当事务提交时，将更新链中的数据块写入数据库。私有缓冲区有两个优点：

1）通过一定的协议，可防止丢失修改。

2）防止"脏读"。由于事务的私有缓冲区是彼此隔离的，事务 T_i 对数据库所作的更新只在自己的缓冲区中体现，只对自己可见、对其它事务是不可见的。这种思想很好地防止了"脏读"现象。

当 $B_i \cap B_j <> \Phi$ 时（其中 B_i 为事务 T_i 的更新链中的私有缓冲块集合），即事务 T_i 和 T_j 有索引块冲突时，T_i 提交，则 T_j 中的更新链中的某些块将无效。为保证索引的正确性，当 T_j 做查询和提交时，需对更新链中的索引块的有效性做验证，若索引块无效，则刷新无效块，即重新从数据库中读出相应的新块，再重做相关的更新操作。

四、并发控制实现

4.1　总体设计

由于我们采用乐观方法对索引不封锁，因此索引并发控制的主要工作是维护索引的正确性。维护工作大致上分为两部分：控制信息的设计和控制器的设计。

控制信息存放于事物结构中的一个数组，该数组中每一个元素对应于更新链中的一个数据块（包括普通数据块和索引数据块），用于记录在该块上所做的更新操作以及该块是否有效等信息。

控制器是维护工作的核心，它通过各事务的控制信息，及时检测块冲突并刷新无效块来保证数据的一致性。图2给出了该方案的示意图。

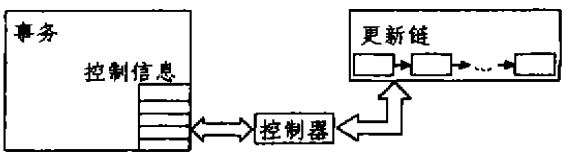

图2　总体设计图

4.2 具体实现

1. 控制信息的设计　控制信息的数据一方面方便控制器进行控制，另一方面保证块冲突检测等控制能正确实施。基于以上两方面的考虑，控制信息包括以下四方面：

①块的唯一性标志：通过该数据和更新链上的块对应。

②块属性：包括块类型和块对应的表号。更新链上的数据块有三类：普通数据块、索引块和字典块，我们只记录普通数据块和索引块，而且对于普通数据块和索引块的处理也不一样，因此，需要块类型区分。块所对应的表号有两个作用：首先，我们在刷新无效块时需将无效块按表分类，表号为分类提供数据；其次，更新链的字典块有可能随数据更新语句而变更，表号可将相应无效的字典块选出刷新。

③块的更新记录：它是刷新无效块时主要的数据依据，在数据块上所做的更新操作都记录于此。DM2中，更新分解为先删除后插入，因此，记录分为插入数据链和删除数据链，插入数据详细记录插入的数据，删除数据只记录删除元组的唯一性标志 TID。

④块的有效标志：该信息最简单，但最重要，它标志块的有效性，检测块冲突改变它，刷新块也依据它来判断是否刷新。

形式化表示如下：

```
typedef struct
{
   long bid; //块唯一性标志：块号
   int  tabid; //块对应的表号
   short btype; //块类型
   INSERTBUFFER   *insert_data; //插入数据链
   DELETEBUFFER   *delete_data; //删除数据链
   short_t flag; //块有效标志
}UPD_LNK_BLK_INF;
```

2. 控制器的设计　控制器的功能是利用以上的控制信息，保证数据和索引的正确性，它分四个子模块：控制信息初始化、更新数据记录、块冲突检测和无效块刷新，具体分述如下：

①控制信息初始化。本模块负责控制信息空间的申请以及控制信息各值的初始化。具体初始化包括：块唯一性标志、块属性的写入；块的表号初值置为－1；块有效标志置为有效。

②更新数据记录。当更新普通数据时，在控制信息中记录更新的数据。在记录插入数据时，只需简单地将数据记录到插入数据链中；当记录删除数据时，需判断删除的数据是否是本事务曾经插入的，若是，则从插入数据链中取下该项，相当于未曾插入过该元组，否则，将删除元组的 TID 记录到删除数据链中。

当更新索引数据时，有时会涉及多个索引块，我们的做法是：只记录其中第一个更新过的索引块所对应的表号，不记录更新的数据（更新数据已在普通数据块中做过记录）。这样做的理由在刷新无效块模块中解释。

③块冲突检测。该模块告知其它事务，其更新链上的块已无效。算法描述如下：

```
void checkblockconflict()
{
for(其它每个事务 Trans[i])
 for(本事务的每个更新数据块 upd_blk_inf[j])
  for(事务 Trans[i]的每个更新数据块 upd_blk_inf[k])
   if(普通数据块)
   { if(两块的块号相等)
     则更改块的有效标志 upd_blk_inf[k].flag 为无效
   }
   else/ * 索引数据块 * /
   { if(两块的表号相等)
   则更改块的有效标志 upd_blk_inf[k].flag 为无效
   }
释放本事务的控制信息空间
}
```

④无效块刷新。本模块将无效更新块从更新链中取下，按照更新记录重做。具体算法描述如下：

1. 将事务中的无效块按表分类。

2. 若表上有索引块无效，则置表上所有数据块都无效。

3. 对每个无效块，按照更新记录，先做删除操作，再做插入操作。

4. 重置无效块为有效。

4.3 控制器的嵌入

我们按照一个事务的运行顺序来说明。

当事务做更新操作前，它首先将数据块链入更新链中，随后，就要调用控制信息初始化模块，申请控制信息的空间并初始化控制信息；事务真正更新数据后，调用记录更新数据模块，在控制信息中增加更新数据，若表上有索引，则在读出 B^+ 树的第一块时记录该块表号；事务运行过程中，有可能需做查询操作，在事务开始执行到查询这一段时间，其它事务可能更新了数据库，本事务要调用刷新模块，去掉无效块，重做相应的操作以保证数据的一致性；最后，事务提交，首先当然要盗用刷新模块，维护数据一致性，然后，调用块冲突检测模块，通知其它事务数据库已发生变化，并指出无效块。

五、性能分析

我们从并发度和系统行为两方面考虑，下面分别讨论：

5.1 并发度

由以上讨论可知，只要事务通过了表级锁和行级锁的，即事务不同时对同一元组操作，即使在索引块上有冲突，也不需等待，因此，该算法保证了事务间元组级的并发，提高了系统的并发度。

5.2 系统行为

本索引的并发控制策略应该归为乐观算法 OCC (Optimistic Concurrency Control)[6]。系统的效率和块冲突发生的几率，并发控制算法本身有关。据统计，在很多应用系统中，冲突发生的几率是很小的[7~9]。我们

做一简单分析。假一数据库大小为 M 块，读集合和写集合大小为 B 块，则两个事务发生数据冲突的最大可能性为[5]：

$$1-\left(\frac{M-2B+1}{M-B+1}\right)^{B}$$

给 B，M 赋几个样值，我们得到下表（如图3）：

B	M	冲突几率 P(C)
5	100	0.0576
10	500	0.0025
100	1,000	0.1130

图3 冲突几率表

以上的数据说明，乐观方法在大多数情况下是行之有效的。就算法本身而言，控制器的效率主要由冲突检测算法和刷新无效块算法的时间复杂度来决定。冲突检测的时间与事务数和更新链上的块数有关，假定事务数为 N，事务更新链上的块数最多为 L，则冲突检测的时间复杂度为 $O(N*L^2)$，刷新无效块的时间取决于无效块的块数，时间复杂度为 O(L)。

控制器的调用也影响着系统的效率。冲突检测模块的执行时间较长，但只在每个事务提交前调用一次；刷新模块则调用频繁一些，每当事务做查询操作都需调用一次，但其时间消耗不多。因此，整体上看，控制器提高了系统的效率。

结束语 本文在分析了索引并发控制难点的基础上，提出了一种实现策略，并在 DM2 上实现。本算法具有普适性，不仅适用于 B^n 树，而且也适用于其它索引结构如 B^+ 树等。

参 考 文 献

1 冯玉才. 数据库系统基础(第二版). 华中理工大学出版社，1993
2 DM2概要设计. 华中理工大学计算机学院数据库与多媒体技术研究所，1998.6
3 Silberschatz A, Korth H F. Database System Concepts (Third Edition). McGraw-Hill Press
4 Yu P S, Dias D M. Performance Analysis of Concurrency Control Using Locking with Deferred Blocking. IEEE Trans. Software Eng., 1993, 19(10): 982～996
5 Bhargava B. Concurrency Control in Database System. IEEE Trans. Knowledge and Data Eng., 1999, 11(1): 3～16
6 Kung H T, Rbinson J T. On Optimistic Methods for Concurrency Control. Trans. Database System, 1981, 6(2): 213～226
7 Davidson S B. Optimism and Consistency in Partitioned Distributed Database Systems. Trans. Database System, 1984, 17(3): 456～481
8 Gray J N. The Transaction Concept: Virtues and Limitations. In: Proc. VLDB Conf., Cannes, France Sept. 1981
9 Gray J N, Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, Calif., 1993

（上接第28页）

术、系统结构和主要功能，并对它进行了基本的测试和性能分析，我们认为 HOOPE 并行支撑层已经达到了设计目标，它具有以下一些特点：

首先，HOOPE 并行支撑层提供的并行数组类库具有易用高效的特点，开发者可以用类似于 FORTRAN90的语法和编写串行程序的方式编写并行程序。

第二，并行数组类库的设计使用了虚拟共享网格和虚拟边界等技术，使得 HOOPE 并行数组类库具有较高的运行效率，这样就保证了上层用 HOOPE 数组类库直接进行开发的应用程序具有较高的运行效率。

第三，由于上层的应用组件和应用程序直接使用 HOOPE 并行数组类库进行开发，基本上不涉及底层的并行细节，只需根据不同的体系结构实现相应的并行支撑层，就可以将这些组件和应用程序移植到不同体系结构的机器上。

第四，HOOPE 并行数组类库用 C++ 实现，采用了模板等面向对象技术，根据实际需要，开发者可以很容易地对它进行扩充。

HOOPE 并行支撑层还有一些方面的不足，例如，目前仅支持 SPMD 计算模式和规则的网格结构。在实现三维叠前深度偏移应用的过程中，我们看到，一些串行应用程序中的算法使用并行数组类库可以很容易地转化为并行应用程序，这种方法不同于用并行编译器对串行程序进行并行编译的做法，HOOPE 并行数组类库仅需标准的 C++ 编译器即可；也不同于完全用特定的并行语言编写应用程序的做法，而是用类库的方式提供了类似于串行语言的使用方式，因而使用起来更为灵活，更有利于将一些串行程序移植为并行应用程序。这种思路对于应用领域中利用已有的串行算法进行并行应用开发具有一定的指导意义。

参 考 文 献

1 李英军，吕建，王宏琳. 面向对象应用框架在油气勘探领域的应用研究. 软件学报，1999，10(4)
2 李英军，吕建. 一个科学计算领域的面向对象并行应用框架. 计算机工程与科学，1998，20(3)
3 李英军，吕建，于大川，马晓星. 一个层次式面向对象并行计算框架的设计. 电子学报，已录用
4 张林波，迟学斌，汪道柳，等. 网络并行计算与分布式编程环境. 科学出版社，1996
5 Li Yingjun, Lu Jian. Framework-Based Software Reuse for Seismic Processing Applications. In: Proc. of Technology of Object-Oriented Languages and Systems, TOOLS 31, IEEE Computer Society, Los Alamitos, California, 1999. 22～25
6 Gropp W, Lusk E, Skjellum A. Using MPI, Portable Parallel Programming with the Message-Passing Interface, The MIT Press, 1994
7 Lemke M, Schüller A, Solchenback K, Trottenberg U. Parallel processing on distributed memory multiprocessors. In: Proc. GI-20. Annual meeting 1990, Informatik Eachberichte Nr. 257, Springer, 1990