第32卷 第14期 计 算 机 工 程 2006年7月
Vol.32 № 14 Computer Engineering July 2006
·软件技术与数据库· 文章编号：1000—3428(2006)14—0066—03 文献标识码：A 中图分类号：TP311

基于内存的时空索引TPR*树并发控制方案

周 星[1]，刘兆宏[1]，夏 英[1]，葛君伟[1]，裴海英[2]
(1. 重庆邮电学院中韩GIS研究所，重庆 400065；2. 韩国仁荷大学计算机科学与工程部，韩国)

摘 要：随着定位技术的发展，基于定位的服务对数据库技术的要求越来越高，需要用它记录和管理大量持续运动物体的位置。TPR*树是支持移动物体现在和将来位置查询的索引结构，面临多个事务同时访问数据的问题。该文提出了基于内存的TPR*树的并发控制方案，能满足事务的一致性要求。通过结合加锁技术和时间戳技术，使得冲突减少，提高了并发效率和处理速度。

关键词：并发控制；索引结点；时间戳

TPR*-Tree Concurrency Control Scheme for Spatio-temporal Index in Main Memory Database

ZHOU Xing[1], LIU Zhaohong[1], XIA Ying[1], GE Junwei[1], BAE Hae-young[2]
(1. Sino-Korea Chongqing GIS Research Center, Chongqing University of Posts and Telecommunications, Chongqing 400065;
2. Dept. of Computer Science & Engineering, INHA University Younghyun-dong, Nam-ku, Inchon, Korea)

【Abstract】With the development of location technology, advanced location-based services will increasingly require database technologies capable of tracking the positions of large amounts of continuously moving objects in the following years. TPR*-tree is a kind of index structure which supports to query the current and future positions of moving objects. TPR*-tree concurrency control scheme in main memory is presented, which can satisfy to correctly query positions of moving objects. Two most important techniques for implementing scheduler are combined: locking and timestamping technologies which drop conflict and execution time. The scheme improves concurrent efficiency.

【Key words】Concurrency control; Index node; Timestamp

近年来，随着许多新应用的出现(无线定位系统，地理信息系统(GIS)，无线通信技术)，时空数据库在内存中要处理海量数据，同一时刻并行运行的事务数可达数百个。若对并发操作不加控制就可能存取和存储不正确的数据，破坏数据库的一致性。TPR*树[2]作为一种索引结构，虽然已经实现索引操作，但是在查找、插入算法中还没有考虑并发控制，所以和实际应用还有一定的距离。为了确保TPR*树在数据库管理系统中能正确、有效地实现，本文讨论了基于内存的TPR*树的并发控制方案。

因为内存价格的不断下降、容量迅速增大，把整个数据库永久地存储在内存中已切实可行。在并发控制方面，基于主存的数据库不仅只读事务，而且更新事务都能实现良好的性能的次序。但是，不是把数据库直接放在主存中就可以自动实现这个具有开创性意义的结果，而是要求最优化的技术支持主存数据库实现这些优势。在结点的存储单元和封锁粒度方面，一些基于主存的设计在文中被提出，有效提高了并发性能。

1 TPR*树的原索引结构

TPR*树是分级的、高度平衡的索引结构，是对R树的扩展。在TPR*树中，一个移动物体 o 表示为：(1)MBR, oR={(x1,y1)，(x2,y2)}，也就是在参考时间(Ref)点的一个区域；(2) 速度矢量 VBR,ov={ov1-,ov1+,ov2-,ov2+}，ovi+(ovi-)表示物体的速度在沿着第 i 维方向的上界(下界)，(1≤i≤2)。在任一维上，随着时间的推移，位置表示为

$$x_i(t)=x_i(t_{ref})+v_i(t-t_{ref}) \quad (1)$$

TPR*树用MBR作为关键字。TPR*树索引是由内部结点(internal node)和叶结点(leaf node)构成的。叶结点指向移动物体的记录，也就是移动物体的位置。MBR和VBR都是时间参数化的，整棵树有一个当前时间t_o。对于每个移动物体记录，有一个参考时间t_{ref}，当物体由于速度的变化更新时，参考时间t_{ref}就由更新时间t_{upd}取代。

2 实现并发控制的TPR*树的索引结构

为了增加cache命中率，加速内存和cache的访问时间，内存被分成固定大小的块。结点的大小建议是cache块(cache block size)的倍数，这些是能被进程读、写的仅有单元。每个结点最多能包含M个子结点(entry)。存储在树中的子结点1，2，…，M实际上就是子结点的块号。在每个结点存储块的头信息中包含这个结点是否是叶子结点、有多少个结点、MBR。两个锁(latch)标志位[5]。

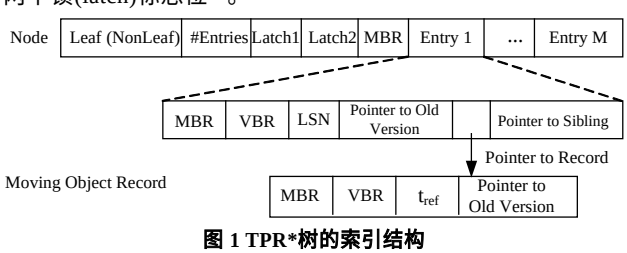

图1 TPR*树的索引结构

基金项目：韩国通信信息部和中韩GIS研究中心基金资助项目
作者简介：周 星(1977—)，女，硕士，主研方向：时空数据库索引；刘兆宏、夏 英，硕士；葛君伟，博士；裴海英，博导
收稿日期：2005-10-14 **E-mail：**zhouxingxing@sina.com.cn

子结点中VBR表示相应的MBR在t时刻的速度和方向。每一个子结点有一个唯一逻辑序列号(LSN)[3]。逻辑序列号在TPR*树的生命期中是单调增长的。每个结点有一个指向右结点的单向链把同一层中的所有结点连起来。

3 并发控制技术

实际应用中，以 TPR*树为索引的时空数据库，处理数据最大的特点就是更新频繁。主要包括两种更新：

(1) 对于每一个移动物体，由于物体持续无规则地运动，因此对于式(1)中的速度矢量是不断变化的。对这种物体，在确定了物体的更新时间t_{upd}之后，要对移动物体进行周期性的更新，确保精确定位。

(2)因为索引叶结点的速度是取叶结点中数据项的最大速度，父结点的速度取所有子结点的最大速度，所以移动物体速度改变，会导致索引结点的速度改变。这样，对索引结点也要及时更新，才能使时间参数化的 MBR 总是保持最小。

当前，实现并发控制可串行性，有两种主要的方式：封锁和时间戳。基于 TPR*树的树形结构和更新频繁的特点，对它采用将这两种方式相结合的方案。对于只读事务采用动态多版本时间戳执行；对于读/写事务采用封锁机制执行。

对 TPR*树结点操作并发控制模型作如下描述：

一个移动物体 MBR 可能与多个结点 MBR 相交。但它只能存在某个 MBR 中,所以在查询中，经常会遇到回溯的问题。这样，给一个索引结点加锁后，会造成很多更新操作无法进行。

在TPR*树中，每一层的索引结点都已经用一个指向右兄弟的指针连接成为链式。这种链指针的定义是一致的，因为TPR*树的所有叶子结点在同一层[1]。这样：1)在查找操作时，不需要任何的锁。即使这时有插入操作，可以从右链中得到所需结点，不会改变树的结构；2)在插入操作时，遍历过程中不用加锁，插入只加排他锁。删除操作最多只有一个结点加排他锁。

在用TPR*树查询时，频繁地要读取当前和原来的版本。如果其他操作把数据删除，事务就会中止读这些数据。为了使这样的读操作继续进行，需要保留数据以前的版本[4]。这样只是让缓冲区管理器在主存中存储了当前活跃的某个事务对应的一些块。

多版本的时间戳由计数器产生，当一个只读事务开始的时候，计数器就加 1，新的值就成了该事务的时间戳。晚开始事务比早开始事务时间戳要大，即如果事务 T(j)比事务 T(i)(i,j 标识不同的事务)要晚，那么版本时间戳 VTS 满足 *VTS(y)>VTS(x)* (x，y 分别标识 T(i)，T(j)产生的版本)。

(1) 查找操作

右链指针为查找结点提供了新的方式。在进行查找时，通过比较 MBR，从根结点向下找到相交或包含的子结点。如果父结点预期的 LSN 和子结点的 LSN 相等，就继续沿树先根遍历。如果父结点预期的 LSN 小于子结点的 LSN，说明子结点当前已经被分裂为两个结点，在查找时，这个信息还没有反映到父结点，查找必须沿着子结点的右链，修改这种错误，直至找到相等的结点。在查找过程中，新的结点和旧的结点在同一个 cache 块内，通过右链指针连在一起，基本上可以被看作同一个结点，所以查找效率降低很小。但是，即使现在有个分裂可能扩展到根结点，读事务仍然能正确进行。当查找到结点时，就读取适合的版本。

每一个索引结点和移动物体的记录指向一个版本列表(Version List)，是由一维数组组成的。第一个版本有一个指针指向这个版本的时间戳 Vts(x)。这个时间戳指向下一个版本的时间戳。如图 2 为结点 V3 的版本列表。如果这个索引结点或记录只有一个版本，事务 T(i)就读这个版本。如果有两个以上的版本，事务 T(i)读取距离当前时间最近的版本。事务 Ti 沿着第一个版本的时间戳的指针，满足 Vts(y)>TS(Ti)>Vts(x)的时间戳 Vts(x)，指向的版本就是事务 T(i)要读取的版本。

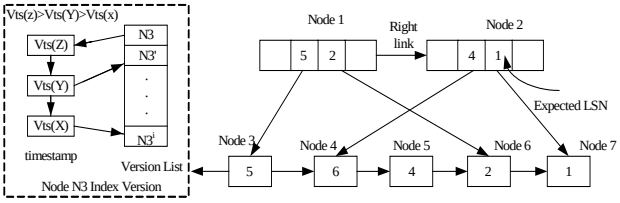

图 2 索引结点操作模型

(2)插入、删除操作

插入一个数据项(Entry 或 Record)，从根结点向下遍历到应该插入的叶子结点。在插入时，可能引起结点的分裂。新的右兄弟保留原来的逻辑序列号(LSN)，原来的结点接受一个新的逻辑序列号(LSN)，这两个结点用右指针相连。分裂总是向右移，原来的结点可以遍历右链得到，当它是安全时，就不再需要加锁这个结点,这是一个减少结点加锁的有实质性意义的改进。这时回溯这棵树结点在父结点中插入这个新结点。父结点也可能发生分裂,继续向上回溯，直至到达一个不需要分裂的结点。在修改一个结点前,必须对它加锁。一旦对一个结点加锁，它不可能再在这个结点的子结点加锁，也不能在这个结点的左边加锁，加锁过程保持了串行的次序。因此整个过程是没有死锁的。

在内存中,可以优化封锁机制的执行。通过选择小的封锁粒度，即锁标识位，来减少冲突。

1)封锁请求的处理：如果结点的锁标志位 Latch1 被设置,表示此结点处于被锁状态,此时拒绝其它事务的插入操作。如果锁标志位 Latch1 和 Latch2 都被设置,表示由一个或多个事务正在等待加锁此数据项。这些等待的事务被存储在传统的哈希表中。

如果一个事务希望锁一个结点,事务先检查它的标志位。如果没有设置，它先设置 Latch1，然后复制此结点,对其进行更新。如果第二个事务正在等待加锁这个结点,它设置 Latch2 并且把它自己加到锁表的等待队列中。

2)解锁的处理：当事务释放 Latch1,它检查 Latch2 是否被设置。如果 Latch2 被设置，它唤醒正等待加锁的事务。事务自己唤醒等待事务避免扫描哈希表，这样需要最小的机器指令(见图 3)。

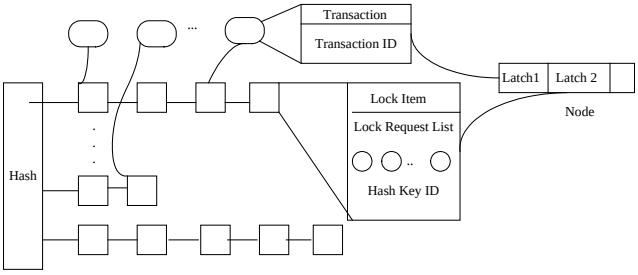

图 3 锁表和结点

当更新时，更新在复制结点上进行。用原子写操作将新

的版本替换需要修改的版本，更新完成后，排他锁立即被释放。

在进行删除操作时，设定叶结点可以包含少于 k 个记录。删除仅仅从叶结点中移除记录，不会对内部结点造成影响。先通过查找操作找到包含删除记录的叶结点，再加锁，把记录从结点中删除，最后解锁。

4 算法

在索引结点的查找、插入、删除算法中，这里主要给出和并发相关的一些主要步骤。

4.1 查找算法

窗口查询(WindowQuery):找出查询窗口 Q 覆盖的所有索引记录。

(1) 根结点 N 入空栈 S

(2) If 栈 S 为空，终止

(3) ELSE 【调整 MBR (Q)到 TPR*树的当前时间 t】

(4) If (N≠叶子结点)

比较 N.LSN 和孩子结点 E.LSN

If (E.LSN>N.LSN) 遍历右链，N.LSN=E.LSN

If entry E 超过一个版本，找到适合的版本 Vts(x),满足 Vts(y)>TS(Ti)>Vts(x)，

If ((E. MBR intersect Q) 入栈 S

(5) If(N＝叶子结点)

If 数据项 D 超过一个版本，找到合适的版本 Vts(x),满足 Vts(y)>TS(Ti)>Vts(x)

If(D. MBR intersect Q) 返回指向这个数据项的指针重复步骤(7)直到 N 的所有的数据项被比较

(6) 从栈中弹出栈顶指针

(7) 先序遍历，重复步骤(5)直到 N 所有子结点被比较

(8) 重复步骤(2)，直到栈为空

4.2 插入算法

对一个结点的更新算法：

(1) 找出由于这个结点更新而受影响的所有结点，让不再向上回溯的那个结点作为根结点 N。

(2) 复制所有这些结点，根为 N’。

(3) 当对这些结点更新后，加排他锁，树中指向 N 的指针，现在指向新的根结点 N’。

(4) 对这些版本加时间戳，以前的版本写入版本列表，释放锁。

5 结束语

本文通过建立结点操作的并发模型和优化封锁机制，阐明在内存中实现 TPR*树并发控制的方案，讨论了 TPR*树在并发控制实现中更新快的特点，以及结点分裂、加锁对并发的影响，为提高并发效率提供了理论基础，并给出在查找和插入中和并发控制相关的算法，有一定的实际指导意义。

参考文献

1 Lehman P L, Yao S B. Efficient Locking for Concurrent Operations on B-Trees[J]. ACM Transactions on Database Systems, 1981, 6(4): 650-670.

2 Tao Y. The TPR*-Tree: An Optimized Spatio-temporal Access Method for Predictive Queries[C]. Proc. of the Intl.Conf. on Very Large Data Bases, 2003-09.

3 Banks D, Kornacker M, Stonebraker M. High-concurrency Locking in R-Trees*[R]. University of California at Berkeley, 1994-09.

4 Ying Xia, Sung-Hee. Dynamic Versioning Concurrency Control for Index-based Data Access in Main Memory Database Systems[C]. Proceedings of the Tenth International Conference on Information and Knowledge Management, 2001-10.

(上接第 26 页)

3 讨论

根据天网用户的查询记录，我们发现了中文搜索引擎用户多任务Web查询的一些基本特性：38.6%的用户进行多主题Web查询，明显多于Excite搜索引擎的 11.4%，与AlltheWeb搜索引擎的 31.8%比较相近[4]。多主题会话中用户所提交的查询串占整个样本所含查询串的 72%，大大高于AlltheWeb搜索引擎的 44.3%[5]。这显示天网用户进行多任务Web查询的比例是比较高的，用户使用的查询串一般会在多任务用户会话中出现。

超过 1/2 的多任务会话包含 2~7 个不同主题，这与AlltheWeb多任务会话中含有 4~22 个主题，存在着较大的差异[5]。平均每个多任务会话有 2 个主题的变化，即进行 3 次不同主题的查询；会话的持续时间是通常会话持续时间的 2.2 倍；这一结果与文献[4]的结果类似。会话持续时间的差异通常是由用户提交了较多的查询所造成的。

天网系统具有较高比例的计算机/网络、教育、科学等主题的查询信息，这从一个方面揭示了天网用户群的特征，即教育与科研领域的用户较多。23.8%的多任务会话中包含“ 不确定信息 ”的查找，这表明仍有许多用户不能较好地表达自己的信息需求。

查询主题间的关联关系反映用户查询需求的一种共性，它为用户查询行为作预测、个性化搜索引擎服务提供了一种参考指标。

4 结论

通过本文的分析，发现了多任务中文 Web 查询的一些特征，并与 Excite 和 AlltheWeb 的多任务 Web 查询进行了比较。结果表明：多任务 Web 查询是人们查询信息时的一种常见模式，它揭示一些用户在有几个信息需求时才进入 Web 搜索引擎系统进行信息查询。

多任务 Web 查询具有较复杂的查询模式，需要更有效的检索技术为如此复杂的查询结构提供服务。目前的信息检索系统主要服务于用户的单任务查询，如何使 Web 搜索引擎系统支持用户的多任务查询是一个需要研究的问题。

参考文献

1 李晓明, 闫宏飞, 王继民. 搜索引擎——原理技术与系统[M]. 北京: 科学出版社, 2005.

2 王继民, 陈 翀, 彭 波. 大规模中文搜索引擎的用户日志分析[J]. 华南理工大学学报, 2004, 32(增刊): 1-5.

3 Spink A, Ozmutlu H C, Ozmutlu S. Multitasking Information Seeking and Searching Processes[J]. Journal of the American Society for Information Sciences and Technology, 2002, 53(8): 639-652.

4 Ozmutlu H C, Ozmutlu S, Spink A. Multitasking Web Searching and Implications for Design[C]. Proc. of Annual Meeting of the American Society for Information Science and Technology, 2003.

5 Ozmutlu H C, Ozmutlu S, Spink A. A Study of Multitasking Web Searching[C]. Proc. of International Conference on Information Technology: Coding and Computing, 2003.