维普资讯 http://www.cqvip.com

一种混合的时空数据库索引机制*)

王平根[1,2] 周脚根[2]
(江西井冈山学院信息科学与传媒学院 江西 343009)[1]
(武汉大学空间信息与数字工程研究中心 武汉 430079)[2]

摘 要 近年来,时空数据库的应用得到迅速发展,在动态时空环境里,维持持续移动对象的位置信息是一个挑战。本文提出了一种新的索引机制,采用两种不同类型的索引结构,一种索引移动对象历史轨迹,一种能够有效地索引移动对象现在、最近、将来的位置信息,而且随着时间的推移,能够将两种索引结构内容快速过渡。在实现移动对象历史轨迹的完整或局部的有效查询的同时,又保证了移动轨迹的空间紧密性。

关键词 时空数据库,时空索引,四叉树

A Hybrid Indexing Mechanism for Spatial-Temporal Databases

WANG Ping-Gen[1] ZHOU Jiao-Gen[2]
(School of Information and Media, Jinggangshan College, Jiangxi 343009)[1]
(Spatial Info. & Digital Eng. Rcscarch Center, Wuhan University, Wuhan 430079)[2]

Abstract Recently the applications of spatial-temporal databasce become more and more popalar, but in dynamic mobile environments how to efficiently maintain location information of continuously moving objects is still a challenging technology. This paper proposes a new indexing mechanism that employs two kinds of indexing structurcs: one is for indexing historical trajectories, the other for indexing current and future locations of moving objects, and it is efficient in shifting between the two indexing structures. With this indexing approach, not only the historical trajectories can be efficiently retrieved partially of completely but also the spacc compactness of moving trajectories can be guaranteed.

Keywords Spatial-temporal databases, Spatial-temporal indexing, Quadtree

1 引言

随着无线通信技术和定位技术的迅速发展,移动对象的概念越来越重要,人们对时空数据库(Spatial-Temporal Database,简记为 STDB)的研究日益关注。STDB 是指对移动对象(如车辆、飞机、移动用户等)的位置及时态进行管理的数据库。随着移动对象位置的不断变化,数据库需要频繁更新。因此,管理随空间和时间变化的移动对象是一种挑战。而索引作为数据库的关键技术,可以极大地改善数据库的整体性能,尤其是对于动态数据库。

在实际应用中,时空索引的空间有效索引能力是必不可少的,同时移动对象的完整(或局部)轨迹的查询也有很大的需求。TB 树[3](一种基于 R 树[12]的索引结构)是近年提出的一种可以有效地实现有关轨迹的查询,但是局限于小数量的移动对象情况。当移动对象达到一定数量时,R 树所固有的内部结点重叠问题将变得十分严重,查询性能会迅速下降。同时,这种索引结构的优点——一个叶子结点只能保存同一轨迹的线段(空间临近的不同对象的轨迹,可能保存在不同的叶子结点中),也使这种索引放弃了空间区分能力。此外,TB 树索引结构只能够索引移动对象的历史轨迹,在实际应用中我们往往需要的是完整时间段的索引结构。为了解决这些问题,我们提出了一种新的混合索引技术。该结构有如下三个方面特色:

(1)将链式结构引入四叉树,实现了位置信息的空间紧密行和轨迹连续性的有机结合。

(2)将两种不同的索引结构通过指针链接,实现了最新更新信息向历史信息的快速过渡;同时,有效回答涉及两种索引结构内容的查询,不需要两次遍历,只要找到一个索引树中的信息,就可以获得整个时段信息。

(3)按照时间戳,建立了四叉树森林,能够根据"距离现在越远的历史信息越不重要,要求的精度越不高"的原则,将距离现在越远的四叉树设计得越简单。这样设计也有助于提高时间段查找的性能。当然,这是以牺牲一定空间为代价的。

2 相关工作

目前,时空数据库索引技术引起了人们极大的关注,我们按照时间,把现有的时空数据索引技术划分为两类:一类是索引移动对象的历史(直到最近位置采样)信息的,另一类是索引移动对象现在、最近、将来的位置信息的。这两类索引各有各的优缺点。

2.1 对历史位置信息的索引

M. A. Nascimento 等人提出了 HR 树[1],他们将时间看成是一维,维持一个数组,数组中保存指向各个时间戳所对应树的指针,每一个更新时间创建一个新的 R 树。插入和删除只在最新的时间戳所指向的树上进行,但后一个时间戳树并不是完全新建,只是局部更新上一个时间戳树,与上一个时间戳树共用没有变化的结点。D. Pfoser 等人提出了两种基于 R 树的轨迹索引,TB 树和 STR 树[3],能够实现轨迹的拓扑查

*)本文工作得到国家自然科学基金资助(编号:60573183)。

维普资讯 http://www.cqvip.com

询。STR 树尽可能地实现移动对象空间紧密性和部分轨迹连续性(部分轨迹相连)之间的折衷。事实上,两种性能都不能够保证。对于 TB 树,只能够实现轨迹连续性,而放弃了空间紧密性这一重要性能。文[4]在 TB 树的基础上进行的改进版本 TB* 树,它消除了冗余数据,缩减了索引大小,增加了一个辅助存储空间,减少了更新的时间,但不能够改变 TB 树固有的缺点。Tao 等人提出了 MV3R-tree[17],主要思想是建立两个索引结构,一个 MVR 树索引时间戳查询,一个 3DR 树索引长时间间隔的查询。V. P. Chaka 等人[18]提出了 SETI,一种将空间和时间分离而形成两层索引结构。

2.2 对现在和最近将来位置信息的索引

TPR 树是基于 R* 树[2]的一种重要的时空索引结构,在此基础上,发展出了一系列新的时空索引结构。C. M. Procopiuc 等人提出的 STAR 树[9],这种索引结构适应于频繁更新的情况;为了能够及时删除索引内的不再有效结点,S. Saltenis 等人提出了 R^{EXP} 树[8];Y. Tao 提出了 TPR* 树[14],对 TPR 树进行了改进,但是索引算法变得复杂了。M. Pelanis 等人提出了 R^{ppf} 树[15],能够在一个索引上实现各时间状态的索引,但是整个索引结构过于复杂,难于实现。G. Kollios 等人使用双重转换技术[16],将多维移动对象映射到一维空间,将问题转化到二维空间,有效地改善了范围查询的性能。J. M. Patel 等人则提出将未来的在 d 维空间的轨迹映射成 $2d$ 空间中的点[10]。J. Tayeb 等人提出了使用 PMR 四叉树进行轨迹索引[6];D. Kwon 等人提出了 LUR 树来索引移动对象当前的位置信息[11],提高了查询效率,但延长了更新时间。

3 四叉树与 TPR 树

为了便于理解本文提出的混合索引结构,这里简单介绍四叉树和 TPR 树。

四叉树是一类常见的、简单易行的空间索引结构。它是属于基于空间划分组织索引结构的一类索引机制。将已知范围的空间划成四个相等的子空间。如果需要可以将每个或其中几个子空间继续划分下去。这样就形成了一个基于四叉树的空间划分。四叉树的优点是:

(1)可以用顺序存储的线性表来表示索引,占用空间小,使得查询速度得到提升。

(2)插入和删除操作简单方便,平均耗时远小于 R 树的插入和删除的花费。

由于四叉树是以空间划分来组织索引结构的索引机制,有着这类索引机制的共同问题就是:在建立索引之前必须预先知道空间对象所分布的范围,可调节性比较差。

TPR 树是基于 R* 树的索引结构,R* 是 R 树的变形。不同于四叉树,R 树属于基于空间数据划分的索引机制,让空间上靠近的移动对象尽可能分到同一个范围矩形,或者拥有尽可能近的共同祖先节点,从而提高查询效率。在组织 R 树的时候,尽可能地让空间对象的空间位置的远近体现在其最近共同祖先的远近上。

TPR 树可以索引移动在一维、二维、三维空间上的对象,为了使包含在范围矩形中的移动对象一直保持在其中,范围矩形将随着其中的对象一起移动,它的各边速度总是维持着某个方向的最大值或最小值。TPR 树是基于 R 树的,对 R 树而言,优点是:它按数据来组织索引结构,这使其具有很强的灵活性、可调节性。无须预知移动对象所在的整个空间范围就能建立空间索引。但是当数据量剧增或者维数超过一定值的时候,内部结点的覆盖也越来越严重,从而查询需遍历多条路径才能找到结果,大大地降低了查询的性能。

4 一种混合时空索引结构

4.1 新索引结构

我们所提出的索引结构是一种混合索引结构,能够完整地索引移动对象各时间段的位置信息。我们采用一种基于四叉树索引结构(简称为历史树)和基于 R 树索引结构(改进的 TPR 树,也简称为 TPR 树)的合并索引,历史树索引移动对象历史轨迹,TPR 树索引移动对象现在和最近将来的位置信息,同时,随着时间的推移,将 TPR 树的内容过渡到历史树中。我们将为移动对象的各段轨迹建立双向链接,当要查找某一移动对象的完整或者部分轨迹时,只要找到一段,就可以依据前向后向指针找到轨迹所有各段。这样既保证了历史轨迹的空间性,又实现了轨迹的连续性。

为了使 TPR 树中的信息能够快速插入到历史树中,我们在历史树与 TPR 树的叶子结点间通过指针建立链接,亦即将当前的位置信息与历史信息之间建立了链接。当 TPR 树中的新一段轨迹需要插入历史树中的时候,我们先找到它保存在历史树中上一段轨迹的结点,如果需要插入的项的空间位置包含于该结点,则插入该项,否则,自顶向下找到可以包含插入项的叶子结点,进行插入。同时,与前一段轨迹建立连接。

在两个索引树间建立连接的另一个好处是,当查询从现在开始到过去某一时间点的轨迹信息时,只需要找到 TPR 树中的当前位置信息,然后,通过链接就可以找到所有历史轨迹信息。很显然,我们更多时候关心的是移动对象的最近历史轨迹信息,如果把所有历史轨迹的信息放入到一棵索引树中,则有四个缺点:1)增加了树的高度;2)增加了不同叶子结点的内容重叠(不断地拆分,使跨两个子空间的轨迹增多);3)不利于时间段查询,当以时间为主的查询出现时,我们必须遍历所有满足空间条件的叶子结点;4)更新操作涉及范围广。为此,我们根据实际需求设计了历史树森林,按照事先预定的时间间隔划分每一棵历史树,只有离现在最近的历史树涉及插入和删除两种操作,其他的历史树仅涉及删除操作。事实上,其它的历史树已经变成了静态树,它的删除操作与移动对象的运动无关,只与数据库的删除操作有关。为了保证轨迹的连续性,我们将各历史树叶子结点之间用指针建立链接。这种设计方法在不影响查询性能的情况下,会极大地改善最近历史树的插入和更新性能。同时,为了避免过时的数据在数据库中占用空间,我们可以设定时间门限值,当数据保存时间超过该值的时候删除数据库中的数据和对应索引树,这样做既简单易行,同时又不影响其他时段历史树的使用。

我们设计的新索引结构如图 1 所示。移动对象最新的位置信息到来时,则直接进入 TPR 树进行插入,同时根据指针指引(或重新搜索)将最新轨迹段插入到最近历史树中(即图 1 中 T5 所指的 Qt5 树)。查询时,先判断是对历史还是对现在或最近将来的查询,从而选择进入不同的索引树。对历史树的查询可以分时间段进入不同的历史树。一般来讲,历史树的时段间隔取值应适当,取得太大了,起不到预期的效果,取得太小了,跨时段查询的情况就会增多,反而降低了查询效率。图 1 中,E1、E2、E4、E6、E7、E9、E11 和 E13 是同一条轨迹的各段,通过指针链接,为了使图示更加清晰,我们在图 1 中只画出单项指针。

维普资讯 http://www.cqvip.com

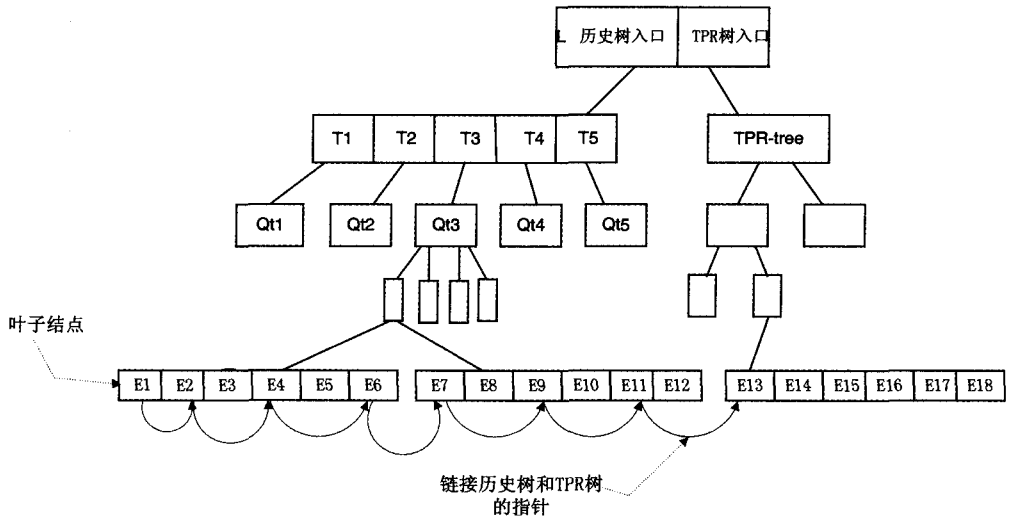

图 1　混合索引的结构框架

4.2　结点结构

索引树中包含两种结点，叶子结点对应一个硬盘页，保存移动对象的实际位置（点或线段）。下面，简单介绍索引中的结点结构。

（1）历史树

叶子结点的项（Trajetory #，t1，t2，$Pointer_{pre}$，$Pointer_{next}$），Trajetory # 是指轨迹的位置信息，t1，t2 是指轨迹的时间间隔，$Pointer_{pre}$，是指向前一段轨迹的指针，$Pointer_{next}$ 是指向后一段轨迹的指针。中间结点的项（Region，$Pointer_{chi}$），Region 是指孩子结点的区域大小，$Pointer_{chi}$ 指向孩子结点的指针。

（2）修改的 TPR 树

中间结点的每一项有如下形式：（TPBR，$Pointer_{chi}$），TPBR 表示孩子结点的区域大小，是时间的函数，$Pointer_{chi}$ 是指向孩子结点的指针，对于叶子结点：（Position1，t1，Pointer），Position1 是指最近更新位置信息，t1 是 Position1 的插入时间，Pointer 是指向前一段轨迹的指针。当最新的更新位置 Position2，即当移动对象的速度或者方向与 Position1 相比发生变化而更新的信息到来时，则用（Position2，now，pointer）替换（Position1，t1，pointer），同时将轨迹（Position1，Position2，t1，now）插入到历史树中。关于 TPR 的详细的说明见文[7]。

4.3　算法

这里，我们介绍所引结构中结点插入和删除算法。

（1）插入算法

当 TPR 树收到最新位置信息时，自顶向下遍历找到要插入的结点（查找算法同原 TPR 树），如果当前信息是移动对象第一次发出的位置信息，则将新项插入（结点溢出的分裂算法也同原有的 TPR 树），否则，将上一次更新的位置（POS1）同要插入的位置信息（POS2）组成的轨迹段插入到历史树中，同时在 TPR 树中删去 POS1 的信息，并将最新位置信息 POS2 插入，一次遍历将实现所有操作。因为 TPR 树与历史树之间建立了链接，所以很容易找到要插入轨迹段（Trjectory2）的上一段轨迹（Trajectory1）所在结点，如果 Trjectory2 包含在 Trajectory1 所在的结点，则插入，否则自顶向下遍历，直到找到可以包含 Trjectory2 的结点，将 Trjectory2 插入，同时，Trajectory1，Trjectory2 之间建立链接。算法如下：

Algorithm-1　Insert(newEntry)

```
let newEntry be a new inserted segment;
  FindPreNode(newEntry); // 找到欲插入项的上一段轨迹所在的结点
    IF leaf node N is found // 新项有历史轨迹;
      IF newEntry IN N // 新轨迹和上一段轨迹在同一叶子结点
        IF leaf node already full// 新轨迹和上一段轨迹在同一叶子结点,但该叶子结点已满
          SplitNode(N); // 调用分裂函数将子四叉索引空间拆分
        ELSE insert newEntry; 并将插入结点与 TPR 树建立链接
      ELSE{ FindPos(newEntry);
        Insert newEntry;} // 自顶向下找到新项所在的空间位置,并将插入结点与 TPR 树建立链接
ELSE {FindPos(newEntry);
    Insert newEntry; } //新项没有历史轨迹,则自顶向下找到新项所在的空间位置,并将插入项与 TPR 树建立链接
```

算法中所用的基本函数的实现：因为改进 TPR 树和历史树建立了链接，直接可以实现 FindPreNode(newEntry)；对于 SplitNode(N)、FindPos(newEntry) 和 InsertnewEntry 可以直接利用四叉树的基本算法。

（2）删除算法

由于，历史树保存的轨迹随着时间的推移，重要性逐渐减弱，直到最后时间段整个索引树将被删除。若要单独删除移动对象某一段轨迹或整个轨迹，先自顶向下找到这个轨迹段所在叶子结点，进行删除，若要删除的只是一段或几段，则在删除时将剩余轨迹继续链接。当删除造成某一叶子结点容量太小时，如果它的父结点的四个孩子结点的总容量小于某给定值 C 时，将四个孩子结点合并。删除某一段轨迹的算法如下：

Algorithm-2　Delete(E)

```
FindNode(N); // 自顶向下找到 E 所在的结点
Delete E; // 删除项 E
IF Capacity(Parent)<C // 如果父结点的容量小于某一规定值时
    Merge(N1,N2,N2,N4); // 将该父结点的四个孩子合并
Link(Pre,Post); // 将被删除的轨迹前后段重新链接
```

5　查询分类、性能比较及样例分析

5.1　查询分类与性能比较

我们首先将查询分为两大类：一类是基于坐标的查询，例如，在三维空间（包括时间维）的点、范围和最近邻居查询；另一类是基于轨迹的查询，查询某一或某些移动对象的轨迹拓扑。具体分类见表 1。表 1 中给出了 TB 树、TPR 树和本文方法对各种查询的实现能力的比较：

维普资讯 http://www.cqvip.com

表1 查询分类与各种索引方法的性能比较

查询分类	TB树	TPR树	本文方法
对过去某时间点某空间范围内的移动对象的查询	费用很高	×	√
对过去某时间段某空间范围的查询	费用很高	×	√
对某个或某些移动对象一段或整个轨迹查询	√	×	√
对过去某时间点的某移动对象最近邻居查询	×	×	√
对移动对象当前位置的查询	√	√	√
对当前某空间范围的内的移动对象的查询	√	√	√
对移动对象最近将来的位置的查询	×	√	√
对最近将来某空间范围内移动对象的查询	×	√	√

5.2 查询样例分析

前面我们将TB树、TPR树与本文方法在查询处理能力进行了定性的比较，这里我们将举实例说明它们应用能力上的差别。

实例1 查询某押运车A的在最近20分钟的轨迹并预测未来的5分钟的轨迹(具有现实意义的轨迹查询)。

这是一种典型的移动对象轨迹拓扑查询，使用TB和我们的混和索引技术，都容易得到A在最近20分钟的轨迹，但是TB索引技术不能够得到未来轨迹的预测。利用我们的方法可以简单实现，我们先通过A的最新位置信息在TPR树中找到它的插入位置，并根据线性插值法预测它最近5分钟的轨迹，并根据指针链接直接进入历史树叶子结点找到A先前的轨迹信息，仅需一次遍历。

实例2 查询某段街道在最近10分钟的所有移动对象(时空范围查询)

这种查询TB树是没有能力实现的，使用本方法可以简单解决，我们现在历史树中自顶向下对该段街道进行比较，找到可以覆盖它的空间，并逐层到达叶子结点，在叶子结点中筛选满足条件的项，最后返回结果。

实例3 查询某移动对象B在过去某时间点T某空间点S的最近K个邻居

这种查询TB树也无法实现，使用本方法可以在短时间得到模糊值(范围偏大)，先是按照自顶向下的方法找到位置S所在的索引中的位置，并在该子空间内找到与时间T匹配的项。但若是再进一步缩小范围得到满足条件的K项，将需要浪费大量的索引时间。

总结 随着无线通信和位置技术的不断进步，促进了时空数据库的快速发展，移动对象频繁的更新位置，为移动信息的保存和使用带来了极大的困难，简单易行的索引技术，可以极大地改善数据库性能。在本文，我们提出了一种混合的索引技术，与以往的索引技术只能索引一种时间状态的信息不同，该索引能够索引移动对象过去、现在和最近将来的位置信息。将链式结构引入四叉树，保留了四叉树原有的优点，同时，又实现了R树家族的一些特点，例如，实现了轨迹的连续性。该混合索引直接应用了现有的一种完善的索引技术——TPR树，实现移动对象现在和将来位置的索引，稍作改进，实现和历史树的内容链接。由于四叉树和TPR树都是成熟的技术，因此该索引实现起来极为容易，对于可以使用顺序表存储的四叉树，若在其上建立链接，只需开辟出一个字段存放指针即可。因此，该混合索引具有极大的实用价值。

参考文献

1 Nascimento M A, Silva J R O. Towards historical R-trees. In: Proceedings ofACM-SAC' 98), 1998

2 The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In:Proceedings of SIGMOD'90, May 1990. 322～331

3 Pfoser D, Theodoridis Y, Jensen C S. Novel Approaches in Query Processing for Moving Object Trajectories. In: Proc. of VLDB'00, 2000. 395～406

4 Lee E J, Ryu K H, Nam K W. Indexing for Efficient Managing Current and Past Trajectory of Moving Object. In: Proceedings of APWeb'04, 2004. 782～787

5 Raptopoulou K, Vassilakopoulos M, Manolopoulos Y. Towards Quadtree-Based Moving Objects Databases. In: Proceedings of ADBIS'04, 2004. 230～245

6 Tayeb J, Ulusoy O, Wolfson O. A Quadtree Based Dynamic Attribute Indexing Method. The Computer Journal, 1998, 41(3): 185～200

7 Saltenis S, Jensen C S, Leutenegger S T, Lopez M A. Indexing the Positions of Continuously Moving Objects. In: Proc. of SIGMOD'00, 2000. 331～342

8 Saltenis S, Jensen C S. Indexing of Moving Objects for Location-Based Services. In:Proc. of ICDE'02, 2002. 463～472

9 Procopiuc C M, Agarwal P K, Har-Peled S. STAR-Tree: An Efficient Self-Adjusting Index for Moving Objects. In: Proc. of the Intl. Workshop on Algorithm Engineering and Experiments, 2002. 178～193

10 Patel J M, Chen Y, Chakka V P. STRIPES: An Efficient Index for Predicted Trajectories. In: Proc. of SIGMOD' 04, Paris, France, June 2004

11 Kwon D, Lee S, Lee S. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In: Proceedings of MDM' 02, 2002

12 Guttman A. R-trees: a Dynamic Index Structure for Spatial Searching. In: Proceedings SIGMOD'84, Boston, MA, 1984. 47～57

13 Finkel R, Bentley J L. Quad trees: A data structure for retrieval of composite keys. Acta Informatica, 1974

14 Tao Y, Papadias D, Sun J. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In: Proc. of VLDB'03, 2003. 790～801

15 Pelanis M, Saltenis S, Jensen C S. Indexing the Past, Present and Anticipated Future Positions of Moving Object: [A TIME-CENTER Technical Report]. July 2004

16 Kollios G, Gunopulos D, Tsotras V J. On Indexing Mobile Objects. In: Proc. of PODS'99, 1999. 261～272

17 Tao Y, Papadias D. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In: Proc. of VLDB' 01, 2001. 431～440

18 Chakka V P, Everspaugh A, Patel J M. Indexing Large Trajectory Data Sets With SETI. In: Proceedings of CIDR'03, 2003