第 36 卷 第 4 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.4 Computer Engineering February 2010
·软件技术与数据库· 文章编号：1000—3428(2010)04—0050—03 文献标识码：A 中图分类号：TP311.13

改进的移动对象索引机制

杨 旭，余建桥，任廷艳
(西南大学计算机与信息科学学院，重庆 400715)

摘 要：针对 R 树结构在索引移动对象时存在的内部节点重叠问题，结合四叉树结构，提出一种 R+树与四叉树(Quad 树)混合的改进索引结构，采用懒惰更新算法以及扩充与收缩 MBR 方法，可同时索引移动对象历史及当前位置信息。仿真实验结果表明，该索引结构在查询数量较大时能够获得较高的查询效率。

关键词：时空数据库；移动对象；索引

Improved Index Mechanism for Moving Object

YANG Xu, YU Jian-qiao, REN Ting-yan
(College of Computer and Information Science, Southwest University, Chongqing 400715)

【Abstract】 Aiming at the problem of node overlap in R-tree in moving object index, a novel indexing structure is proposed, which combines R+-tree and Quad-tree. It uses the lazy update algorithm and extended and contracted MBR method. It can index both current and historical positions. Simulation experimental result shows this structure has highly performance when the number of searches is larger.

【Key words】 space-time database; moving object; index

1 概述

许多应用领域都需要处理大量的移动对象及其空间查询与检索，更好地管理海量移动对象及进行高效的数据库检索已成为时空数据库的关键问题。

采用懒惰更新算法的 LUR 树(Lazy Update R-tree)是基于 R 树的移动对象索引结构。R 树采用对象的最小外包矩形(记为 MBR)表示空间对象，用所有子节点的最小外包矩形 MBR 表示它们的父节点。LUR 树在索引移动对象时，仅当对象移动到所在节点 MBR 外时，才更新索引结构。如果对象移动后新的位置还在原节点 MBR 内，只在原节点中更新对象位置即可。尽管此方法可快速更新移动对象位置并减少更新代价，但由于采用基于 R 树结构，当移动对象数量很大且密度较高时，因此 R 树固有的内部节点重叠问题将变得十分严重，使得查询性能迅速下降。

针对以上问题，本文将 R+树结构用于懒惰更新的移动对象索引，可解决索引节点间的重叠问题，同时参照高维空间映射中过滤假活动子空间技术[1]，在 R+树更新过程中扩充与收缩 MBR，使索引节点 MBR 大小和位置随移动对象位置变化同步更新。

2 R+树和四叉树

2.1 R+树

R+树可以看作 R 树结构的一个变体。假设有叶节点(oid, p_{new},t)(oid 为移动对象 id，p_{new} 为移动对象位置，t 为当前时刻)，中间节点(N, child_pointer)(N 为中间节点最小外包矩形 MBR，child_pointer 为指向子节点的指针)，R+树结构应满足以下几点：

(1)若(*N*, child_pointer)包含节点 R，那么它必须完全包含 R，除非 R 为叶节点；

(2)2 个兄弟叶节点不能重叠；

(3)根节点必须包含至少 2 个子节点，除非它为叶子；

(4)所有叶节点都在同一层。

兄弟叶节点间没有重叠的好处在于减少了无效查询，提高了空间索引效率。图 1 为一个 R 树结构，W 为一个搜索区，此时，由于 W 在节点 A 和 B 的重叠区域内，则 A 和 B 的子树都要被搜索到，结果只有 B 返回一个合格的查询对象。在 R+树中，如图 2 所示，允许叶节点分布在 2 个最小外包矩形中，兄弟节点间无重叠区域，则查询不会出现上述情况。

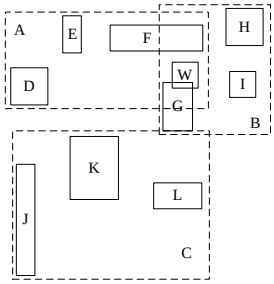

图 1 R 树矩形结构

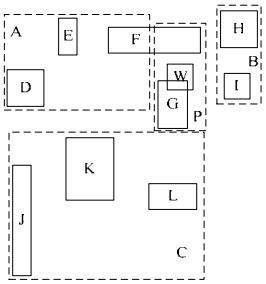

图 2 R+树矩形结构

2.2 四叉树

四叉树是空间索引结构的一个重要分支。它是一种基于空间划分组织索引结构的树结构。以二维空间为例，四叉树定义如下：

(1)根节点表示整个二维空间；

(2)每个非叶节点都有 4 个子节点，每个子节点代表父节点按水平和垂直方向进行划分得到的 4 个子空间；

(3)每个叶子节点都满足一定的要求，例如，有的要求叶子节点空间中对象个数不得大于某个值。

根据是否保存中间节点存储的信息，可将四叉树分为线

作者简介：杨 旭(1984－)，女，硕士研究生，主研方向：时空数据库，索引技术；余建桥，教授；任廷艳，硕士研究生

收稿日期：2009-12-10 **E-mail：**yangxuyx@swu.edu.cn

性四叉树和一般四叉树。线性四叉树索引结构中，只保存它的所有叶节点数据，中间各层节点数据都可以由叶节点计算得出。因此，四叉树占用空间较小，查询效率较高，并且插入和删除操作简单方便。但是基于空间划分来组织索引结构使得在四叉树建立索引之前必须预先知道空间对象所分布的范围，可调节性较差[2]。

2.3 LUR 树索引

LUR 树基于 R 树构造索引结构，采用懒惰更新算法，即只有对象离开其原来所处的叶节点时，才选择“删除-重插回根节点”、“删除-重插回父节点”或“扩充叶节点 MBR”实现该对象的位置更新。同时，LUR 树在 R 树基础上引入了一个 Direct-Link 结构，此结构以对象 id 为关键字，以实现标识符为条件的对象查找。

3 QLUR+树

QLUR+树基于 R+树和四叉树构成索引树结构，R+树索引移动对象当前时刻信息，四叉树存储并索引移动对象历史信息。算法结合了 LUR 树的懒惰更新，提出了扩充与收缩 MBR。本节首先给出 QLUR+树基本思想，然后给出此索引树结构以及树的查询和更新算法。

3.1 QLUR+树基本思想

为保证 QLUR+树的更新性能，提出了扩充与收缩 MBR 思想。扩充 MBR 即如果对象移动到所在节点 MBR 以外且位于父节点 MBR 的空白区域，则对此节点 MBR 进行扩充，使移动对象仍在此 MBR 内；收缩 MBR 即如果对象在移动后新的位置仍在原来的 MBR 内，则检查此节点的外包矩形是否仍然为最小外包矩形，若不是则根据目前对象位置调整 MBR。考虑图 2 的情况，若对象 K 移动到了一个新的位置，如图 3 所示，则根据收缩 MBR 思想，叶节点 C 的 MBR 大小和位置在变化后如图 4 所示。

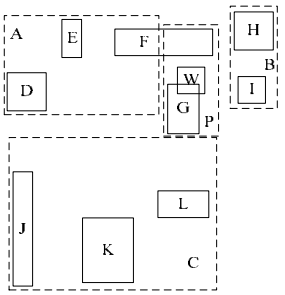

图 3 对象 K 移动到新位置

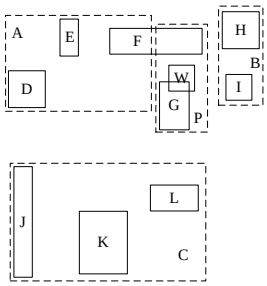

图 4 收缩 MBR

收缩 MBR 使得索引结构的空间可利用率得到了提高。当节点 C 进行收缩 MBR 调整后，其他节点才能够更大限度地扩充 MBR，从而降低了插入和删除频率，在保证查询性能的前提下获得较好的更新效率。

3.2 QLUR+树结构

QLUR+树是一种混合的索引结构。采用 R+树来索引移动对象当前信息，四叉树存储并索引移动对象历史信息。R+树采用懒惰更新和扩充与收缩 MBR 的算法思想，在树结构中引入了一个附加索引结构 F-L，R+树和四叉树通过索引结构 F-L 相关联。F-L 索引关键字为对象 id，每条索引记录包括 2 个指针 f 和 c，f 指向四叉树中对象最近的历史存储信息，c 指向 R+树中相应的移动对象，这样就建立了 R+树和四叉树的连接。

引入 F-L 索引结构后，当查询某一对象的当前信息或历史信息时，首先通过对象 id 在 F-L 中找到此对象，即可通过 c 指针和 f 指针找到该对象所在叶节点，然后在叶节点中查询该对象记录。查询当前信息则从 c 指针进入 R+树，查询历史信息则从 f 指针进入四叉树。这种基于对象 id 的查询方法避免了从根节点进入遍历树的过程。

当一个包含移动对象 id 和新信息的更新请求(oid, p_{new}, t)到来时，首先在 F-L 结构中查找是否包含对象 id 为 oid 的移动对象。若不包含，则判断此对象为一个新的移动对象，通过传统的 R+树插入算法将其插入到索引树中。若 F-L 结构中包含此对象 id，直接通过 F-L 结构的 c 指针在 R+树中找到此对象所在节点，检查懒惰更新算法是否可行，即叶节点 MBR 是否包含对象移动后的新位置。如果对象新位置仍在原 MBR 内，则仅在原节点记录中更新对象位置，然后检查是否可使用收缩 MBR，如可行则根据对象新位置重新定义节点 MBR 大小及位置；如果对象移动到 MBR 外，则检查新的位置是否为父节点空白区域，如果是父节点空白区域则使用扩充 MBR 使得对象仍位于此 MBR 内，否则按照传统 R+树更新方法进行对象的删除与重插入。更新前，要将上一次更新至当前更新间的轨迹信息通过 F-L 索引指针 f 插入到四叉树中。这样就快速实现了当前信息的更新及当前信息向历史信息的过渡。

四叉树在存储移动对象历史信息时，每个节点可包含多个对象，将每个对象记录用链表结构存储，每项包括(position, t1, t2, p)，其中，t1, t2 为轨迹的时间间隔，position 为此时间间隔内对象的位置，p 为指向下一个节点项的指针。对象记录结构如图 5 所示。

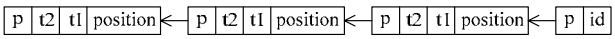

图 5 四叉树叶节点中对象记录结构

显然，连续的节点存储的为连续的时间段，方便轨迹查询和时间段查询。考虑存储空间问题，可设定一个时间限制 t0，在此时间之前的信息将从四叉树中删除，此值可根据具体需求和环境设定。

图 6 给出了完整的 QLUR+树的索引结构。

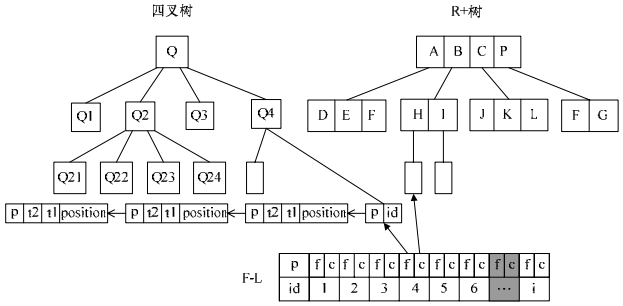

图 6 QLUR+树索引结构

3.3 算法

3.3.1 查询算法

根据所给查询条件，将查询分为基本位置查询和空间约束查询。基本位置查询即查询条件中给定了移动对象 id 及查询时间 T，此时可直接通过 F-L 索引结构进行搜索。空间约束查询根据给定时间范围或时间戳确定当前查询或历史查询后，选择不同的树从根节点进入。

下面给出查询算法：

Algorithm Search(E)

Begin:

1： IF E 为基本位置查询

2： 根据给定对象 id 搜索 F-L 找到此对象
3： IF T 为当前时刻
4： 沿 c 指针进入 R+树中对象所在叶节点，根据 id 找到记录项并查询当前位置信息
5： ELSE //T 为历史时刻
6： 沿 f 指针进入四叉树中对象所在叶节点，找到对象并根据时刻 T 依次查找每个时间段，得到对象位置
7： ELSE //E 为空间约束查询
//根据所给时间判断是当前时刻查询还是历史查询
8： IF T 为当前时刻
//采用 R+树的查找算法
9： 从 R+树根节点进入，往下找到对象所在叶节点进行查询
10： ELSE //历史查询
从四叉树根节点进入，往下找到对象所在叶节点进行查询
11：返回查询结果

3.3.2 更新算法

更新包括了新移动对象第 1 次位置信息的插入以及已有移动对象位置的改变。新移动对象直接插入到 R+树中，已有移动对象位置的改变则通过判断是否适用懒惰更新确定更新算法，对象更新后检查对象所在节点是否可进行主动紧缩。更新前将上一次更新至此次更新间轨迹信息插入到四叉树中，同时修改指针 f 使其指向链表第 1 个节点。

Algorithm Update(Entry)
Begin:
1: IF Entry 为新移动对象信息
2： 将 Entry 插入到 R+树中
3： 更新 F-L 结构
4：ELSE //Entry 为已有移动对象信息
5： 根据给定对象 id 搜索 F-L 找到此对象
6： 根据对象指针找到对象所在叶节点
7： 找到对象所在记录
8： IF Entry 仍在此叶节点 MBR 内
9： 上次更新到此次更新时间段内位置信息写入四叉树
10： 在叶节点内改写对象位置信息
11： IF MBR 可收缩
12： 根据对象新位置收缩 MBR
13： ELSE //Entry 移动到此叶节点 MBR 外
14： 上次更新到此次更新时间段内位置信息写入四叉树
15： IF 对象位于父节点空白区域
16： 使用扩充 MBR
17： ELSE 删除此对象信息
18： 将 Entry 信息插入到 R+树
19： 更新 F-L 结构

4 实验与分析

为验证所提出的 QLUR+树的性能，将其与 LUR 树进行比较实验，并对其查询性能和更新性能进行了分析。实验环境为：Windows 操作系统，Pentium 处理器 2.8 GHz，1.00 GB 内存，磁盘页面大小为 4 KB。实验数据采用 GSTD(General Spatio-Temporal Data)[3]生成的仿真数据集，由实验空间中 1 000 个~100 000 个数据组成。所有数据为随机分布，且在以后的时刻中位置随机变化。

4.1 查询代价

由于 QLUR+树和 LUR 树都采用了一个附加的索引结构，可直接根据对象 id 查找对象所在叶节点，所以它们在移动对象当前时刻的基本位置查询中具有同样的查询效率。不同的是 QLUR+树对当前时刻索引采用基于 R+树结构，因此，在空间约束查询中具有很高的查询效率，同时可支持历史信息查询。下面分别比较了 LUR 树和 QLUR+树在当前时刻查询中域查询与最近邻查询的性能。

(1)域查询：实验采用基于 R 树的域查询算法，比较了查询请求窗口大小变化时 LUR 树和 QLUR+树的性能。实验设定了 5 组查询请求集，每组包含 200 个请求窗口。查询请求窗口在每一维上的范围分别是：0.1%, 1%, 10%, 20%, 30%，那么，在空间中的总范围为：0.000 1%, 0.01%, 1%, 4%, 9%。图 7 显示了每次查询所需的平均时间。可以看出，随着窗口范围的增大，QLUR+树仍具有很高的性能。这是因为它没有叶节点重叠，当查询窗口增大时，被包含到查询窗口中的节点才会被首次访问，而且每个节点只需一次访问。

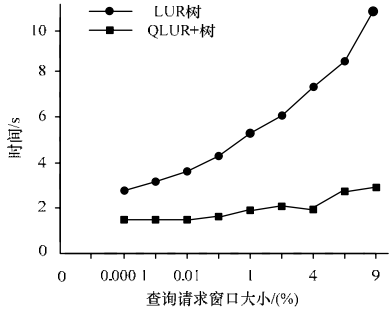

图 7 域查询的性能对比

(2)最近邻查询：最近邻查询中，k 值从 1 到 50，共产生了 50 个查询点，即每个 k 值进行了 50 次最近邻查询请求。图 8 显示了 2 种索引结构的查询性能对比，QLUR+树仍然具有良好的查询性能。原因是当 k 值增大即需要搜索的对象个数增多时，相应的搜索半径就会增大，当查询点所在区域与要搜索区域重叠时，要从根节点开始对此重叠区域进行搜索。而 R 树本身内部节点存在重叠，当搜索到重叠区域时，必定要从根节点进入重新搜索。

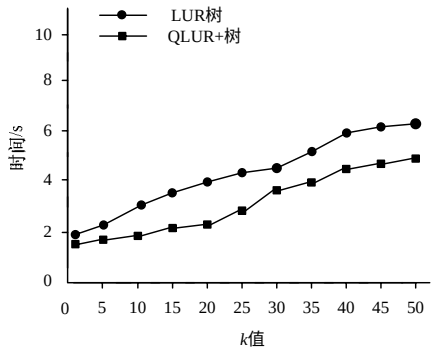

图 8 最近邻查询的性能对比

4.2 更新代价

在更新性能方面，由于 R+树不允许内部节点间有重叠，因此需要比 LUR 树稍大的更新代价。但是，由于采用了懒惰更新算法和扩充与收缩 MBR，也能够获得较好的更新性能，且随着对象数量和密度的增大对比变得不明显。实验比较了 LUR 树和 QLUR+树在每个对象 50 次更新操作条件下的执行时间，更新操作数量为数据集中对象数量的 20 倍。图 9 显示了 2 种索引树的更新性能。尽管此结构需要稍大的存储开销，但是，通过选择合适的四叉树深度和 R+树扇出，就能够得到很好的查询及更新性能。 (下转第 55 页)