第35卷 第7期 计 算 机 工 程 2009年4月
Vol.35 No.7 Computer Engineering April 2009

·软件技术与数据库· 文章编号：1000—3428(2009)07—0042—04 文献标识码：A 中图分类号：TP311

基于Quadtree和Hash表的移动对象全时态索引

李 东，彭宇辉，殷江龙
(华南理工大学计算机科学与工程学院，广州 510640)

摘 要： 为解决大量移动对象位置频繁更新所带来的性能下降问题，提出一种基于改进的Quadtree和Hash表的QH全时态索引结构。这种新的索引结构可以支持移动对象全时态索引，在Hash表中通过存储移动对象指针来支持移动对象标识查询，并对Quadtree的叶子节点采用适时合并的方法来防范分支太深而造成的查询效率低下。实验证明，QH索引与TPR-tree相比，移动对象的更新效率更高、对象标识查询较优、范围查询性能相近。

关键词： 移动对象；索引结构；范围查询

Past, Current and Future Positions Index of Moving Object Based on Quadtree and Hash Table

LI Dong, PENG Yu-hui, YIN Jiang-long
(Institute of Computer Science and Engineering, South China University of Technology, Guangzhou 510640)

【**Abstract**】Traditional index structures do not work well on moving objects because of the need of frequently updating the index which leads to the poor performance. This paper presents a novel indexing structure based on Quadtree and Hash table, namely the QH-index which can index the past, present and future positions of moving object and can support moving object's range queries and point queries which include the object identifier based query. Merging timely the corresponding nodes to degrade the depth of the tree can guarantee the query efficiency. Experiments show that the QH-index gains much better performance in updating and in querying by the object identifier than those of the TPR-tree, and the efficiency of range query is no less than that of TPR-tree.

【**Key words**】moving object; index structure; range query

1 概述

由于移动对象不断地产生大量的时空信息，因此如何有效地管理这些动态信息一直是被广泛关注的研究课题。时空存取方法主要分为2类，一类是对移动对象历史位置的存储和访问;另一类是对移动对象当前和未来位置的存储和访问。索引技术的发展也因此侧重在不同方面。例如，STR-tree结构[1]和TB-tree结构[2]是针对移动对象过去位置信息的索引，TPR-tree 结构[3]是对移动对象当前和未来位置的索引。其中移动对象未来趋势预测技术和全时态索引技术由于更适合在智能交通调度、基于位置服务等领域的应用而得到了广泛的关注，是一个充满挑战性的研究方向。特别是对移动对象全时态索引的研究，目前学术界的成果较少。文献[4]在B^x-tree结构的基础上进行扩展，建立一种索引移动对象从过去到未来的完整信息的全时态索引结构BB^x-Index。文献[5]提出了一种对R-tree进行改进的全时态索引R^{PPF}-tree结构。

但是，在当前已提出的移动对象索引方法中，基本上都不支持对移动对象标识的查询，由此导致索引结构的更新和查询存在一定的缺陷。本文在Quadtree和Hash表的基础上提出了一种新的移动对象全时态索引——QH索引结构，贡献主要体现在以下方面：

(1)提出了支持历史信息轨迹查询、当前和将来信息查询的全时态索引结构QH。在QH索引中，移动对象历史轨迹信息经过轨迹切分后利用Hash表进行索引，在轨迹切分过程中为每一个移动对象建立一个链表以保存被切分的历史轨迹线段，每个线段为链表中的一个节点。提出用改进的Quadtree支持移动对象当前以及预测性未来信息管理。在QH索引结构的基础上给出了支持移动对象的全时态查询算法。

(2)在QH索引的基础上，加入了移动对象的标识，从而使QH索引支持移动对象的标识查询，提高了移动对象查询和更新的性能。

(3)给出了2种移动对象更新策略：一定的时间间隔进行更新和根据速度矢量的改变进行更新。

(4)适时对Quadtree的叶子节点进行合并来降低分支的深度，从而提高查询效率。

2 QH索引结构

在移动对象数据库中，通常管理着非常庞大的移动对象。在查询处理时如果扫描所有的移动对象将会极大地影响系统性能。为了减小搜索空间，就必须对移动对象进行索引。移动对象的索引方法通常借鉴于空间数据索引技术，这些索引方法对移动对象的索引具有很好的借鉴意义，但是它们并不能直接用于移动对象的索引。这是因为上述方法更多地考虑查询效率，而没有着重考虑索引的更新代价。在移动对象数据库中，移动对象的位置更新会引起索引结构频繁的动态变化，因此，需要重点考虑索引的更新性能。为了提高查询效率，引入了标识查询；为了提高更新性能，引入了改进的

作者简介： 李 东(1970－)，男，副研究员，主研方向：现代数据库理论与实现技术，移动计算技术，XML技术，多媒体技术；彭宇辉、殷江龙，硕士研究生

收稿日期： 2008-07-24 **E-mail：** scut35@163.com

Quadtree。因为 Quadtree 结构简单，更新时不需要对树进行平衡调整，所以可以提高更新效率。

2.1 Quadtree的改进

Quadtree 是基于空间划分的空间索引，主要适用于二维空间(分支数为 4)，也可以推广至更高维 *d*(分支数扩大到 2 的 *d* 次幂)。如图 1 所示，它基于递归分解的原则，不断将空间四等分成东北、西北、西南、东南 4 个象限，直至每个子空间中的对象数目不大于某个阈值。四叉树结构简单，但不平衡，对应于高密度区域的子树要比对应于底密度区域的子树深；进行空间搜索时，性能较高；可以索引多种对象，如点、曲线等。

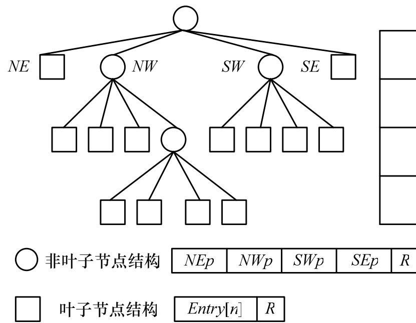

图 1　改进的 Quadtree 结构

为适合移动对象索引的需要，对 Quadtree 四叉树进行改进，加入时间和速度变量。改进的 Quadtree 四叉树中的非叶子节点为 *Not_LeafNode*(*NEp*, *NWp*, *SWp*, *SEp*, *R*)。其中，*NEp*, *NWp*, *SWp*, *SEp* 是分别指向其子节点的指针；*R* 的数据结构为(*xlt*, *ylt*, *xbr*, *ybr*)。其中，$<xlt, ylt>$表示该节点空间范围左上角的坐标；$<xrb, yrb>$表示该非叶子节点空间范围右下角的坐标。改进的 Quadtree 四叉树的叶子节点 *LeafNode*(*Entry*[*n*], *R*)中 *Entry*[*n*]为存储移动对象点的一个数组；*n* 为每个叶子节点中允许存在移动对象点的个数；*R* 同上面的定义。每个移动对象的结构为(*OID*, *X*, *V*, *t*)，其中，*OID* 为移动对象标识；$X<x, y>$为移动对象当前的位置；$V<Vx, Vy>$为移动对象当前的速度；*t* 为当前时间。插入移动对象时，从根节点开始搜索，检查移动对象的位置坐标$<x, y>$是否包含在节点的 *R* 中。如果包含，则进一步判断对象属于该节点下的哪个子节点，然后沿此子节点向下继续搜索；如果不包含，就不必沿此节点的子节点向下继续搜索。当搜索到叶节点时，插入到存储移动对象点的数组 *Entry*[*n*]中。当移动对象插入、更新或者删除时，改进的 Quadtree 的叶子节点可能会发生分裂或者合并。例如：每个叶子节点允许存在的移动对象点的个数为 *N*，现在叶子节点 *L*1 中的移动对象数已经达到 *N*，如果又有一个移动对象进入 *L*1，则 *L*1 要分裂成更小的叶子节点 *NEL*1, *NWL*1, *SWL*1, *SEL*1，*L*1 成为非叶子节点。另外，当一个非叶子节点 *L*2 其下的 4 个叶子节点中的移动对象的总数小于等于 *N*，则发生合并，4 个叶子节点的移动对象合并到 *L*2 中，然后删除 4 个叶子节点，同时 *L*2 变成叶子节点。通过这个过程，不但可以消除高密度区域的子树过深的情况，而且改善了当前区域范围查询的性能。查询改进的 Quadtree 的当前某个范围的移动对象时，设给定矩形 *Rq*=(*x*1, *y*1, *x*2, *y*2)，其中，$<x1, y1>$, $<x2, y2>$分别是 *Rq* 的左上角和右下角坐标，要查询其中所包含的移动对象。在查询时，从根节点开始搜索，检查节点中的 *R* 是否与 *Rq* 相交。如果相交，则沿此节点的子节点向下继续搜索；如果节点中的 *R* 不与 *Rq* 相交，则其中的对象不可能包含在 *Rq* 中，就不必沿此节点的子节点向下继续搜索。当搜索到叶节点时，逐个检查其中的对象是否包含在 *Rq* 中。

2.2 全时态索引结构

定义 1 线段是轨迹中最小的单元，记为 *S*(*OID*, *X*, ***V***, *T*)。其中，*OID* 为移动对象标识；*X* 为$<x, y>$，是移动对象在 *ts* 时刻的位置；***V*** 为$<Vx, Vy>$在时间段 *T* 内的速度矢量；*T* 为时间段$<ts, te>$，*ts* 为起始时间，*te* 为结束时间。

定义 2 时间段相邻是指 $T1<t1s, t1e>$和 $T2<t2s, t2e>$2 个时间段，当 *t*1*e*=*t*2*s* 时，称 *T*1 和 *T*2 2 个时间段相邻。

定义 3 轨迹是一组在时间上相连的线段的集合，记为 *T*={*s*1, *s*2,⋯, *si*-1, *si*, *si*+1,⋯, *sm*}，其中，线段 *si*-1 的 *te* 等于线段 *si* 的 *ts*。

定义 4 轨迹线段根据移动对象的速度矢量来划分，同一个速度矢量并且时间段相邻为一个线段 *S*(*OID*, *X*, ***V***, *T*)。例如，*S*1(*OID*1, *X*1, ***V***1, $T1<t1, t2>$)和 *S*2(*OID*1, *X*1, ***V***1, $T1<t2, t3>$)可以合并成一个线段 *S*(*OID*1, *X*1, ***V***1, $T1<t1, t3>$)。

如图 2 所示，整个 QH 索引结构用改进的 Quadtree 存储当前移动对象的位置信息，用 Hash 表和单向链表保存移动对象的历史轨迹。保存移动对象历史轨迹的方法是通过把轨迹划分成轨迹线段保存到单向链表的节点中来实现的。每个链表节点的数据结构为 *LinkedElement*(*S*, *NextLink*)。其中，*S*(*OID*, *X*, ***V***, *T*)见定义 1；*NextLink* 指向下一个链表结点。为了支持移动对象的标识查询，每个 Hash 表节点的数据结构为 *Element*(*OID*, *LeafNodeLink*, *LinkedList*)。其中，*OID* 为移动对象标识；*LeafNodeLink* 为指向在 Quadtree 中存储移动对象的叶子节点；*LinkedList* 指向保存移动对象历史轨迹的单向链表的头结点。在进行移动对象标识查询时，通过 *OID* 索引 Hash 表，然后通过 *LeafNodeLink* 就能定位存储移动对象的叶子节点，最后在叶子节点中找到目标移动对象。

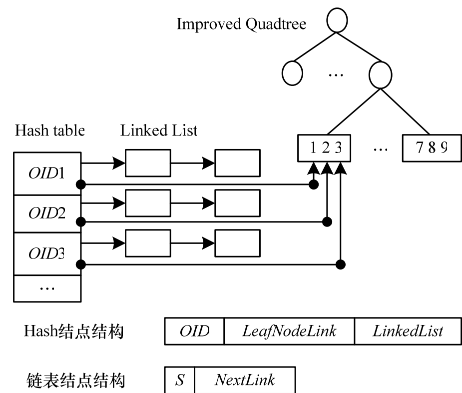

图 2　整个 QH 索引结构

3 更新策略及算法

QH 索引更新策略包括对 Quadtree 和 Hash 表进行更新。

对 Quadtree 来说，对应于高密度区域的子树要比对应于底密度区域的子树深。为了提高查询效率，必要时要对叶子节点进行合并。移动对象的更新过程如下：首先找到移动对象所在的叶子节点，比较当前移动对象的位置 *Xc* 是否超出了当前叶子节点 *L*1 区域范围。(1)*Xc* 超出 *L*1 的范围，则从叶子节点 *L*1 删除该移动对象，同时判断叶子节点 *L*1 的父节点 *P* 所有的孩子节点中的移动对象数是否小于等于阈值 *N*，如果是则进行合并节点 *P*1 的叶子节点，使节点 *P*1 变为叶子节点；如果没有超过阈值 *N*，则不用合并。然后找到移动对象所在区域的叶子节点 *L*2，重新插入移动对象到新的叶子节点 *L*2

前，先判断叶子节点 $L2$ 的移动对象数是否大于阈值 N-1，如果是则把 $L2$ 分裂成 4 个叶子节点，然后重新插入叶子节点 $L2$ 中的移动对象到新的叶子节点中。(2)Xc 没有超出 $L1$ 的范围，则直接更新移动对象的信息。

对 Hash 表的更新策略主要有 2 种：(1)一定的时间间隔进行更新，到一定的时间间隔把 Quadtree 四叉树中的内容进行更新，把旧的数据放到 Hash 表中，根据时间相邻和速度矢量相等的原则看能不能把旧的数据和链表中的轨迹线段进行合并，如果没能找到时间相邻并且速度矢量相等的轨迹线段，则自己单独成为一个新的轨迹线段插到链表的尾部。这种方法简单容易实现，但是每隔一定时间对四叉树中的每个节点进行更新耗费大量资源。(2)根据速度矢量的改变进行更新，只有当移动对象的速度矢量发生改变时，才修改四叉树中的相应数据，同时把旧的数据放到 Hash 表中，这样就不会涉及到轨迹合并的问题。因为时间相邻的 2 个轨迹段速度矢量肯定不相同，否则，就不会发生更新。因此，利用速度矢量的改变来进行更新效率比较好。

基于速度变化 QH 更新算法：*Update*(*QH*, *OID*, ***Vp***, ***Vc***, *Xc*, *Tc*)

输入 移动对象标识 *OID*，过去的速度 ***Vp***，当前的速度 ***Vc***，当前位置 *Xc* 和当前时间 *Tc*

输出 null

```
Begin
  If Xc 超出了当前叶子节点 L1 区域范围  then
    Begin
      Delete(L1,OID); /*将该移动对象从叶子节点 L1 中删除*/
      If 叶子节点 L1 的父节点 P 所有的孩子节点中的移动对象数小于等于阈值 N then
        Begin
          Unite(P); /*合并 P 中的所有孩子节点，把所有移动对象放到 P 中，并使 P 变成叶子节点 */
        End
      Search(L2, Xc); /*找到移动对象所在区域的叶子节点 L2*/
      Insert(L2,OID); /* 将该移动对象插入叶子节点 L2 中*/
      If L2 中的移动对象数大于阈值 N  then
        Begin
          Divide(L2); /* 将 L2 分裂成 4 个孩子节点*/
        End
    End
  Else /*Xc 没有超出当前叶子节点 L1 区域范围*/
    Begin
      X=Xc;V=Vc;T=Tc;/*直接更新移动对象信息*/
    End
  If Vp≠Vc then
    Begin
      Insert(Hash,OID,Xp,Tp); /*把旧的移动对象信息插入到 Hash 表*/
    End
End
```

4 查询处理及算法

4.1 时刻查询

时刻查询是查询某一移动对象某一时刻的位置情况。例如：查询移动对象 a 在时刻 $t1$ 的位置 $X1$，如果是在全时态中查询，就要判断 $t1>t$, $t1<t$ 或者 $t1=t$(t 为当前时刻)。当 $t1=t$ 时，直接在 Quadtree 四叉树中搜索移动对象并返回当前位置 X；当 $t1>t$ 时，属于预测查询，先在 Quadtree 四叉树中找到当前位置 X，然后根据公式 $X1=X+\boldsymbol{V}*(t1-t)$得到 $t1$ 时刻的位置 $X1$。其中，$\boldsymbol{V}<\boldsymbol{V}x,\boldsymbol{V}y>$是当前移动对象的速度矢量。当 $t1<t$ 时，属于历史轨迹查询，直接在 Hash 表中查找对应的移动对象标识，到相应的链表中找到需要的轨迹段，然后根据公式 $X1=X+\boldsymbol{V}*(t1-ts)$得到 $t1$ 时刻的位置。

时刻查询算法：*TimeQuery*(*QH*, *oid*1, *t*1)/*在 QH 索引结构里面查找*/

输入 移动对象的 *oid*1 和时刻 *t*1

输出 该移动对象在 *t*1 时刻的坐标位置

```
Begin
  If  t1<t then       /*t 为当前时间*/
    Begin
      X=Search(Hash,oid1,t1); /*在历史轨迹中查找包含 t1 时刻的链表节点*/
      X1=X+V*(t1-ts) /*根据公式得到 t1 时刻的位置*/
    End
  Else If t1=t then
    Begin
      X1=Search( Quadtree,oid1,t1); /*在四叉树中查找*/
    End
  Else
    Begin
      X= Search( Quadtree,oid1,t);  /*先找到当前时刻 t 时 oid1 的位置*/
      X1=X1+V(t1-t);   /*根据公式得到 t1 时刻的位置*/
    End
  Return X1;
End
```

4.2 范围查询

定义 5(移动对象的范围查询) 给定一个时间段[$t1$, $t2$]，一个查询窗口 q=([$x1$, $x2$], [$y1$, $y2$])(其中，[$x1$, $x2$]和[$y1$, $y2$]分别表示 q 在二维空间中每一维的取值范围)，找到所有在这个时间段内包含在查询窗口 q 内的移动对象。

范围查询算法：*Rangequery*(q, [$t1$, $t2$])

输入 查询范围 q=([$x1$, $x2$], [$y1$, $y2$])，查询时间段[$t1$, $t2$]

输出 范围查询的结果集 *resultset*

```
Begin
  resultset=null;
  If  t2<t  then  /*t 为当前时间，t2<t 则需要在历史轨迹 Hash 中查找 */
    Begin
      For 每个在 Hash 表中的移动对象 do
        Begin
          If  该移动对象的某个轨迹段的时间段 Ti∩[t1,t2]≠Φ 并且轨迹线段 Si∩q≠Φ then
            Begin
              把该移动对象放入 resultset
            End
        End
        Return  resultset;
    End
  If  t1≥t  then  /*t1>t 则是预测未来在[t1, t2]时间段通过范围 q 的移动对象*/
    Begin
      For 每个在 Quadtree 中的移动对象 do
        Begin
          计算出移动对象在将来时间段[t1, t2]中的轨迹线段 Si
          If Si∩q≠Φ then
```

```
                把该移动对象放入 resultset
            End
            Return    resultset;
        End
  If   t1<t   and   t2≥t   then   /*则[t1,t]时间段为历史轨迹查询，[t,t2]为预测查询*/
      Begin
        resultset=Rangquery(q,[t1,t])U Rangquery(q,[t,t2]);
        return resultset;
      End
  End
```

4.3 支持标识查询

QH 索引结构能有效支持标识查询，无论是基于标识的历史轨迹查询还是基于标识的当前位置查询和预测查询都很方便。现在只介绍基于标识的当前位置查询，例如，要查找标识为 *oid*1 的移动对象的当前位置，在不支持标识的索引结构中，必须搜索整个四叉树。但是在 QH 索引结构中，由于 Hash 表和四叉树中的移动对象以标识号建立连接(如图 2 所示)，因此只需要在 Hash 表中找到标识号为 *oid*1 的节点，并顺着 Hash 表连接到四叉树的指针找到要查询移动对象的当前位置。

5 实验分析

本节给出 QH 索引和 TPR-tree 的性能测试结果。性能测试以计算机处理时间为主要性能指标，从移动对象更新、查询移动对象和当前时刻的范围查询 3 个方面比较了 QH 索引和 TPR-tree。

实验中使用的机器配置是：主频为 1.67 GHz 的 AMD 处理器，512 MB 内存，Window 操作系统。另外利用 TPR-tree 的数据生成器来模拟移动对象的运动。TPR-tree 的数据生成器通过给定一些参数来模拟在二维空间的运动，生成均匀分布或非均匀分布的数据集，这些参数的设置/组合方法见文献[5]。本文利用此生成器模拟移动对象(如汽车)在范围 500 km× 500 km 交通路线网络中的运动。此路线网络目的地数目 10，每个移动对象的运动速度在 0~5 km/min 间均匀分布(加速、减速和最大最小速度)，运动方向随机，速度更新平均时间间隔 *UI*=60 min(更新时间间隔在 0~2*UI* 间均匀分布)，总时长为 600 min。

QH 索引结构中阈值 *N* 决定了 QH 索引中的叶子节点动态分裂和合并的时机。对于一个 QH 索引来说，节点的阈值 *N* 取不同值导致移动对象的更新时间会不同。根据实验结果，分裂和合并的时机是节点里的对象数目达到 35 时，QH 索引会到达最好的更新效率，所以在以下的实验中阈值 *N* 都取 35。

(1)更新性能比较

初始条件是利用生成器先生成 100 000 个移动对象。然后分别随机产生移动对象数为 1 000, 2 000 到 10 000 的 10 组数据，在索引结构中进行更新，每组数据进行 10 次实验取平均值得到所需的时间，然后进行比较，如图 3 所示。

(2)移动对象查询性能比较

初始条件为分别利用生成器生成包含 10 000, 20 000 到 100 000 个移动对象的 10 组数据的索引结构。从中随机选择 100 个移动对象并按其 id 值进行查询，每组数据进行 10 次实验取平均值得到所需的时间，然后进行比较，如图 4 所示。

(3)当前时刻的范围查询性能比较

初始条件为分别利用生成器生成包含 10 000, 20 000 到 100 000 个移动对象的 10 组数据的索引结构。随机生成一个区域范围 q=([x1, x2], [y1, y2])，对当前时刻进行查询，每组数据进行 10 次实验取平均值得到所需的时间，然后进行比较，如图 5 所示。

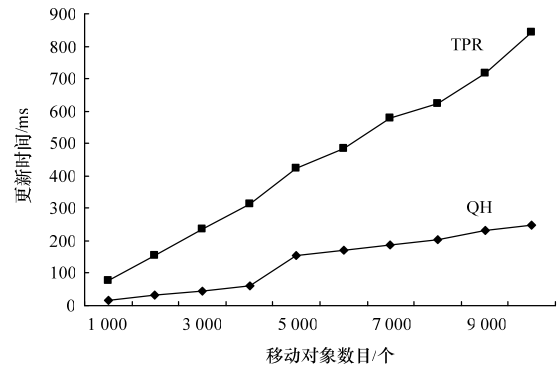

图 3 更新性能比较

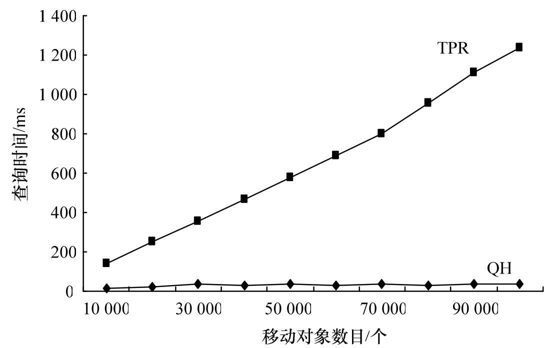

图 4 移动对象查询性能比较

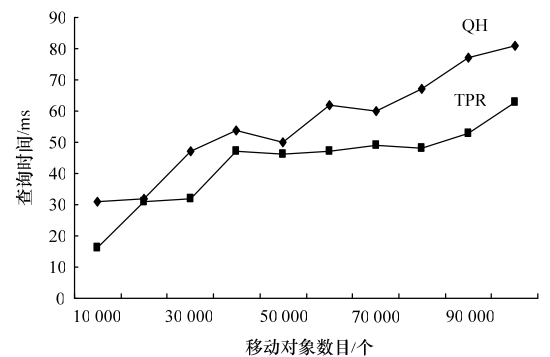

图 5 范围查询性能比较

从上面 3 个实验可以看出：由于 QH 索引对移动对象的插入和删除都较简单，没有像 TPR-tree 一样为了使树保持平衡而进行复杂耗时的分裂和重新插入操作，因此更新性能明显优于 TPR-tree。同时，虽然 QH 索引结构对应于高密度区域的子树要比对应于低密度区域的子树深，跟 TPR-tree 相比，其查询效率会相对差一点。但是，笔者引进了对象标识查询和适时地对 QH 索引的叶子节点进行合并等方法，使其查询效率有了很大的提高。总的来说，由于移动对象索引对移动对象更新性能要求比较高，因此在查询性能相当的情况下，QH 索引在性能上具有很大的优势。

6 结束语

在当前已提出的移动对象索引方法中，基本上都不支持对移动对象标识的查询，由此导致索引结构的更新和查询效率低。本文给出了一种有效地索引移动对象过去、当前以及未来位置的索引结构，它在已有的技术基础上进行了扩展，

(下转第 48 页)