Journal of Computer Applications
计算机应用,2014,34(1):58 -63
ISSN 1001-9081
CODEN JYIIDU
2014-01-10
http://www.joca.cn

文章编号:1001-9081(2014)01-0058-06
doi:10.11772/j.issn.1001-9081.2014.01.0058

支持位置追溯的射频识别移动对象索引机制

廖国琼[1,2*],叶小玉[1],蒋　剑[1],狄国强[1],刘德喜[1,2]
(1.江西财经大学 信息管理学院,南昌 330013;　2.江西省高校数据与知识工程重点实验室,南昌 330013)
(*通信作者电子邮箱 guoqiong.liao@gmail.com)

摘　要:随着射频通信技术的不断成熟及硬件制造成本的不断降低,射频识别(RFID)技术已开始应用于物品实时监控、跟踪与追溯领域。在供应链应用中,RFID对象数量繁多而且位置经常发生变化,如何从海量数据中查询标签对象的位置及其变化历史已成为供应链追溯亟须解决的问题。针对RFID移动对象特征及追溯查询需求,提出了一种有效的时空索引机制CR-L,并详细讨论了CR-L的结构及维护算法,包括插入、删除、二分裂及惰性分裂算法等。针对对象查询,CR-L利用读写器、时间及对象等三维信息设计了新的最小外界矩形(MBR)值计算原则,将相同读写器在相近时间内探测到的轨迹尽可能聚集于相同或相邻节点。对于轨迹查询,采用单链表将相同对象的轨迹链接起来。实验结果表明,所提索引机制具有较好的查询效率和较低的空间占用率。

关键词:射频识别;移动对象;时空索引;位置追溯;最小外界矩形
中图分类号:TP311.13　　**文献标志码**:A

Index mechanism supporting location tracing for radio frequency identification mobile objects

LIAO Guoqiong[1,2*], YE Xiaoyu[1], JIANG Jian[1], DI Guoqiang[1], LIU Dexi[1,2]
(1. *School of Information Technology, Jiangxi University of Finance and Economics, Nanchang Jiangxi* 330013, *China*;
2. *Jiangxi Key Laboratory of Data and Knowledge Engineering, Nanchang Jiangxi* 330013, *China*)

Abstract: As the radio frequency communication technology gets more mature and the hardware manufacturing cost decreases, Radio Frequency IDentification (RFID) technology has been applied in the domains of real-time object monitoring, tracing and tracking. In supply chain applications, there are usually a great number of RFID objects to be monitored and traced, and objects' locations are changed essentially, so how to query the locations and the histories of location change of the RFID objects, from the huge volume of RFID data, is an urgent problem to be addressed. Concerning the characteristics of mobile RFID objects and the tracing query requirements in supply chain applications, an effective spatio-temporal index, called as CR-L, was put forward, and its structure and maintenance algorithms, including insertion, deletion, bi-splitting, and lazy splitting, were discussed in detail. In order to support object queries effectively, a new calculation principle of Minimum Bounding Rectangle (MBR), considering the three dimensional information including readers, time and objects, was presented to cluster the trajectories by the same reader at close time into the same node or the neighboring nodes. As to trajectory queries, a linked list was designed to link all trajectories belonging to the same object. The experimental results verify that CR-L has better query efficiency and lower space utilization rate than the existing method.

Key words: Radio Frequency IDentification (RFID); moving object; spatio-temporal index; location tracing; Minimum Bounding Rectangle (MBR)

0　引言

随着射频通信技术的不断成熟及硬件制造成本的不断降低,射频识别(Radio Frequency IDentification, RFID)技术已广泛应用于供应链及物流环境中,对粘贴有电子标签的物品从原材料采购、生产加工、包装、运输、仓储及销售等环节进行全程自动监控[1-3],从而实现对流通物品位置的实时跟踪和追溯。

大体上,RFID对象位置追溯查询可分为轨迹查询和对象查询两类[4]:

1)轨迹查询(Trajectory Query,TQ),即给定对象的标签ID,查询该对象所经过的位置,主要包括:

TQ_look(tid, t_1, t_2):查询标签对象(tid)在时间[t_1, t_2]内所经过的位置;

TQ_history(tid, ALL):查询标签对象(tid)全部位置变化历史;

收稿日期:2013-07-31;**修回日期**:2013-10-19。　**基金项目**:国家自然科学基金资助项目(61262009);江西省自然科学基金资助项目(20122BAB201032);江西省优势科技创新团队建设计划项目(20113BCB24008);江西省教育厅重点科技项目(GJJ10694, GJJ12259)。

作者简介:廖国琼(1969-),男,湖北大冶人,教授,博士,CCF高级会员,主要研究方向:数据库、数据挖掘,物联网数据管理;　叶小玉(1989-),女,江西九江人,主要研究方向:数据库;　蒋剑(1974-),男,江西南昌人,讲师,硕士,主要研究方向:物联网数据管理;　狄国强(1964-),男,江西景德镇人,教授,主要研究方向:移动计算;　刘德喜(1975-),男,湖北襄樊人,副教授,博士,CCF高级会员,主要研究方向:数据挖掘、信息检索。

TQ_current(*tid*):查询标签对象(*tid*)当前所在位置。

2)对象查询(Object Query, OQ),即给定读写器 ID,查询该读写器探测到的标签对象,主要包括:

OQ_look(*rid*, t_1, t_2):查询读写器(*rid*)在时间[t_1, t_2]内检测到的标签对象;

OQ_history(*rid*, *ALL*):查询读写器(*rid*)探测到的全部标签对象;

OQ_current(*rid*):查询读写器(*rid*)当前所探测到的标签对象。

然而,在供应链环境中,RFID 对象数量繁多且位置经常发生变化,如何从海量数据中追溯 RFID 标签对象的位置及其变化历史已成为供应链追溯亟须解决的问题之一[5]。

本质上,供应链环境中的 RFID 对象是移动对象。但是,不同于一般移动对象,RFID 对象还具有如下特征:

1)标签对象的位置是由读写器决定,即探测到标签对象的读写器所在位置即为该对象的位置;

2)读写器的监控区域是离散的,故 RFID 对象的运动轨迹也是离散的。

迄今为止,空间及移动对象索引技术已得到较多研究。R 树[6]是一棵高度平衡的空间树,它利用空间对象运动轨迹的最小外界矩形(Minimum Bounding Rectangle, MBR)来近似表达空间对象之间的包含关系,但其查询效率会因重叠区域的增大而降低。在最坏情况下,其时间复杂度会由对数搜索退化成线性搜索。R$^+$树[7]采用对象分割技术,避免了兄弟节点的重叠。它要求跨越子空间的对象必须分割成两个或多个 MBR,即一个特定对象可能包含于多个节点之中。R$^+$树克服了 R 树中存在的多路查询问题,但其会引起节点重新划分,且删除运算会导致索引重建等。R*树[8]的目标是使 MBR 重叠面积最小化,它是通过增加构造时间开销换取较高的查找效率,但仍存在多路径查找问题。R^{H}树[19]通过将数据项存放在离根节点较近的节点,以减小低层节点的 MBR 面积。虽然该方法可以提高空间利用率和减少 I/O 访问次数,但索引构造时间较长,且不能解决多路径查找问题。QR 树[10]是一棵特殊的四叉树,它是将整个索引空间划分成多级子索引空间,然后对每级的子索引空间均采用 R 树进行索引,但其空间开销较大。

然而,上述索引方法均不能有效支持 RFID 移动对象的位置离散特征,无法处理 RFID 标签对象的“只进未出”问题[4],即“标签对象进入读写器探测范围但尚未离开”,因此文献[4]在 R 树基础上提出了一种时间参数化间隔模型索引(Time Parameterized Interval R-Tree, TPIR),但其未考虑不同类型追溯的查询需求,查询效率较低(下节做详细分析)。基于 TPIR 索引,文献[11]提出了一种 TPIR-Reorder 索引,其目标是通过减少磁盘访问时间来提高查询效率。它通过对位置标识(Location IDentifiers, LID)进行逻辑排序,使得查询目标结果尽量处于相邻近的磁盘上。但是,当查询过于频繁时,LID 逻辑排序的频率也会增加,导致后续查询等待时间较长,从而降低查询效率。而且,该索引未考虑不同类型追溯需求,难以建立满足多种类型追溯查询的统一索引。

因此,本文拟研究支持 TQ 和 OQ 查询的 RFID 移动对象的索引机制,以提高 RFID 对象的位置追溯查询效率。

1　问题描述及索引框架

对一个索引而言,查询效率、构造时间以及空间占用率都是需要考虑的重要性能指标。但当选择具体索引方法时,应分析哪些因素应优先考虑。在基于 RFID 的供应链追溯系统中,通常查询时间是首先要考虑的性能指标。此外,由于标签数量多且数据量大,需要大量空间存储位置信息,因此也应考虑如何降低空间占用率。

1.1　问题描述

图 1 是 RFID 对象 O_1、O_2 和 O_3 在读写器 R_1、R_2、R_3、R_4 和 R_5 之间的位置移动实例。O_1 依次经过 R_1、R_2、R_3、R_4 和 R_5,然后又回到 R_1;O_2 依次经过 R_1、R_3 和 R_5;O_3 只经过 R_2。

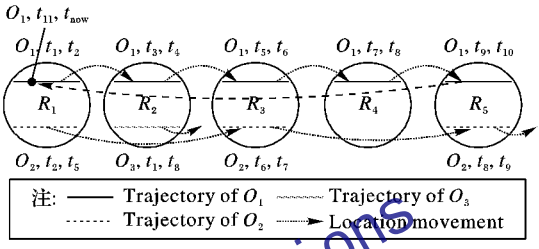

图 1　RFID 对象位置移动实例

定义 1　设 $tr = \langle O_i, t_{is}, t_{ie}, R_j \rangle$ 为标签对象 O_i 的一条运动轨迹(trajectory),表示标签对象 O_i 从 t_{is}(进入时间)至 t_{ie}(离开时间)位于读写器 R_j 的探测区域内。

定义 2　如果一轨迹 *tr* 既有 t_{is} 又有 t_{ie},即对象已离开了读写器,则称 *tr* 为线轨迹(line trajectory)。

定义 3　如果一轨迹 *tr* 只有 t_{is},而没有 t_{ie},即对象仍在读写器的探测范围内,则称 *tr* 为点轨迹(point trajectory)。

特殊地,对于点轨迹,其离开时间记为 t_{now},即 $t_{ie} = t_{now}$。

图 2 是图 1 中全部运动轨迹示意图,其中:O_1 有 6 条轨迹(tr_1, tr_3, tr_5, tr_7, tr_9, tr_{10}),O_2 有 3 条轨迹(tr_2, tr_6, tr_8),O_3 有 1 条轨迹(tr_4)。特殊地,tr_{10} 是一条“只进未出”的点轨迹,即 O_i 在时间 t_{11} 进入 R_1 检测范围,但未离开。

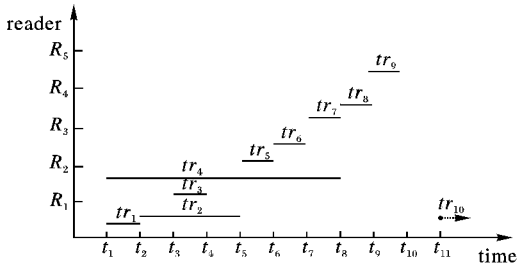

图 2　运动轨迹图形表示

TPIR 索引树是使用读写器标识表示对象的位置。与 R 树的区别是,TPIR 对每条轨迹增加了时间维,即标签对象进入和离开读写器的时间,并且使用用读写器、对象和时间三维信息(权重相同)确定 MBR 大小。假设每个 MBR 最大容纳轨迹数为 4,于是可建立如图 3 所示的 TPIR 索引树[5],其具体构建过程如下:

t_1 到 t_3 时刻:tr_1、tr_4、tr_2 和 tr_3 依次产生,置于根节点中。

t_5 时刻:tr_5 产生。此时根节点轨迹数已达最大值,需要分裂。按照最小包含原则,tr_1、tr_2 和 tr_4 置于节点 *A* 中,而 tr_3 和 tr_5 置于新节点 *B* 中,且 *A* 和 *B* 为根节点的孩子节点。

t_6 和 t_7 时刻:依次产生 tr_6 和 tr_7,置于节点 *B* 中。

t_8 时刻:tr_8 产生,应置于节点 B 中,但此时 B 中的轨迹数已达最大值,需要分裂,于是分裂为节点 B 和 C,其中 tr_3、tr_5 和 tr_6 置于 B 中,而 tr_7 和 tr_8 置于新节点 C 中。

t_9 和 t_{10} 时刻:tr_9 和 tr_{10} 依次产生,置于节点 C 中。

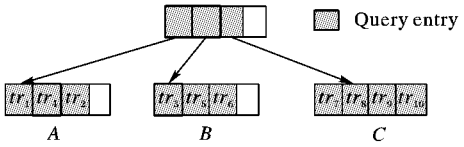

图 3 TPIR 树索引

假设现有一 OQ 查询:在 t_1 到 t_{10} 时间内,读写器 R_2 探测到了哪些标签对象?

通过 TPIR 索引,得到的查询结果为 tr_3 和 tr_4。可以看出,需访问的节点数为 3,包括 1 个根节点和 2 个叶子节点。

而实际上,tr_3 和 tr_4 是同一读写器 R_2 产生的两条轨迹。如果它们在索引中能位于同一节点中,则需访问节点数为 2,从而可提高查询效率。因此,本文拟针对 RFID 移动对象的 OQ 查询设计新的 MBR 计算原则和索引结构,以提高查询性能。

1.2 索引框架

在空间索引中,在选择子入口进行插入操作时,依据原则是 MBR 值尽可能最小。在已有的空间索引中,基本上是以矩形的表面积作为判断 MBR 值大小的依据。然而,RFID 对象的移动轨迹是由标签对象、读写器和时间三种因素共同决定。因此,本文拟采用三维信息,即标签对象维(o)、读写器维(r)和时间维(t)共同确定 MBR 值。

为保证相同读写器以及相近时间的轨迹尽可能聚集于相同节点或相近节点,本文对不同维度赋予不同权重。权重越大,表示 MBR 值对该维度的依赖程度越大。由于 RFID 移动对象的位置是由读写器确定,因此赋予读写器维较高权重。其次,考虑到位置的时态约束,故时间维的权重次之,而对象维的权重最低。

设读写器维单位间隔权重为 W_r,时间维单位间隔权重为 W_t,对象维单位间隔权重为 W_o,且 $W_r \gg W_t \gg W_o$。于是,RFID 对象索引的 MBR 值计算原则定义如下:

定义 4 设 tr_i 和 tr_j 为属于不同对象 O_i 及 O_j 的两条轨迹,且 $tr_i = \langle O_i, t_{is}, t_{ie}, R_x \rangle$ 和 $tr_j = \langle O_j, t_{js}, t_{je}, R_y \rangle$,其中 x 和 y 分别为两条轨迹的阅读器序号,则包含它们的 MBR 值 M 计算如下:

$$M = |j - i| \times W_o + |t_{je} - t_{is}| \times W_t + |y - x| \times W_r \quad (1)$$

由于 $W_r \gg W_t \gg W_o$,所以 M 值对 $|y - x|$ 即读写器维的依赖远大于其他维。若 $|y - x|$ 越小,即 M 值越小,tr_i 和 tr_j 处在同一个 MBR 范围内的可能性就越大。因此,定义 4 中 M 值计算方法可将相近时间内经过同一读写器的 RFID 标签轨迹尽可能聚集到相同或相邻 MBR 中,从而实现了轨迹优先按读写器“聚类”。

依定义 4,可生成如图 4 所示的时空索引,其中权重 W_r、W_t 和 W_o 分别设为 1000,100 和 1,读写器之间距离是由读写器序号大小决定,具体生成过程如下:

t_1 到 t_3 时刻:依次产生 tr_1、tr_4、tr_2 和 tr_3,存于根节点中。

t_5 时刻:tr_5 产生。由于此时根节点所包含轨迹数已达最大值,需进行分裂。按照定义 4 中的 $W_r \gg W_t \gg W_o$ 原则,MBR 的大小较大程度依赖阅读器维度。因此,属于相同阅读器(R_1)的轨迹 tr_1 和 tr_2 置于 A 节点中,而 tr_3、tr_4 和 tr_5 置于 B 节点中,且 A 和 B 为根节点的孩子节点。

t_6 时刻:tr_6 产生,置于 B 节点中。

t_7 时刻:tr_7 产生,应置于节点 B 中。但此时 B 中节点数已达最大值,需要分裂。同理,按照最小包含原则,tr_3 和 tr_4 置于节点 B 中,而 tr_5、tr_6 和 tr_7 置于新节点 C 中。

t_8 时刻:tr_8 产生,置于 C 节点中。

t_9 时刻:tr_9 产生,应置于节点 C 中,但此时 C 中节点数已达最大值,需要分裂。此时,tr_5、tr_6 和 tr_7 置于节点 C 中,而 tr_8 和 tr_9 置于新节点 D 中。

t_{11} 时刻:tr_{10} 产生。按最小包含原则,tr_{10} 应置于节点 A 中(都为 R_1 探测到的轨迹)。

因此,可以发现,同属于 R_2 的轨迹 tr_3 和 tr_4 已位于相同节点内。

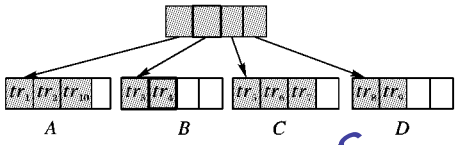

图 4 新 MBR 规则得到的索引

然而,新的 MBR 值计算原则“不利于”对象维,可能导致属于同一对象的轨迹分布到不同的节点中。因此,对于 TQ 查询而言,会增加访问节点数量,从而降低了查询效率。

本文的解决方案是将属于同一个对象所有运动轨迹通过单链表链接起来。这样,TQ 查询时只需找到该对象的轨迹链头,然后依次查询,即可得到该对象经过的全部位置。

在具体实现上,可增加一个哈希表(Hash Table),并在每个叶子节点增加一个指针指向同一对象的下一条轨迹,如图 5 所示。哈希表存放每个对象单链表的表头,当处理 TQ 查询时,先通过对象的 ID,找到其单链表所对应的链头,然后依次获得该对象的全部轨迹信息。

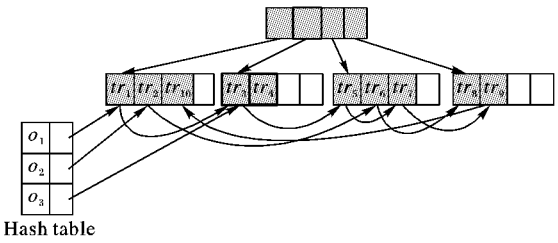

图 5 基本索引结构

本文将这种结合了聚类思想和单链表结构的索引,称为 CR-L(Clustered-based R & List)索引。

2 CR-L 结构及维护

2.1 CR-L 结构

与其他空间索引相同,CR-L 是用于存储 RFID 移动对象运动轨迹之间的时空包含关系,它由叶子节点、非叶子节点和哈希表三部分组成。

1)叶子节点。

叶子节点由多个入口 LEAF_BRANCH 组成,其形式为 $\langle {}^{*}branch[MAXCARD], level, count \rangle$,其中:$branch[MAXCARD]$ 是该节点中所有入口的地址,$level$ 是节点的层次(缺省为 0),$count$ 是节点中已包含入口的数目。$branch$ 的形式为 $\langle tr, {}^{*}next\text{-}object \rangle$,其中:$tr$ 是一条线轨迹或点轨迹,其形式为 $\langle tid,$

rid, t_e, t_l/t_{now}, *state*〉。*state* 是指对象的状态:当 *state* = 0 时,表示对象已进入读写器范围但未离开;当*state* = 1 时,表示对象已离开了读写器范围。

2) 非叶子节点。

非叶子节点由多个入口 BRANCH 组成,其形式为〈*MBR*, **child*〉,其中**child* 是指向孩子节点的指针,MBR 是一个三维最小化边界框。MBR 的形式为〈[*LB*(*o*), *UB*(*o*)], [*LB*(*r*), *UB*(*r*)], [*LB*(*t*), *UB*(*t*)], *state*〉,其中 *LB*( ) 指 MBR 相应维度的下边界,*UB*( ) 指 MBR 相应维度的上边界。

3) 哈希表。

哈希表用于存储所有对象单链表的表头和表尾信息,项的形式为〈*tid*, **head*, **tail*〉, 其中:**head* 是指向指定对象轨迹的第一个入口,**tail* 指向指定对象轨迹的最后入口。**head* 指针用于 TQ_look 和 TQ_history 查询,而**tail* 用于 TQ_current 查询。

2.2 CR-L 树维护

2.2.1 轨迹插入

与 R 树类似,CR-L 树的插入要求保证索引树的平衡,使所有叶子节点都处在树的同一层上。而区别是,CR-L 树在选择节点进行插入操作时是依据新的 MBR 原则计算其 *M* 值并使其最小,且将新插入的轨迹置于对象单链表的尾部。具体插入过程如算法 1 所示。

算法 1　Insert(*tr*)。

输入　*O*— 对象标识; *tr*— 待插入的轨迹。

输出　1— 成功;0— 失败。

```
BEGIN
1)  lnode = entry_query(tr);          /* 确定待插入的叶子节点 */
2)  if lnode. count < MAXCARD                    /* 节点未满 */
3)    br = entry_insert(lnode, tr)               /* 轨迹插入 */
4)  else                                         /* 节点已满 */
5)    br = split(lnode, tr)                      /* 节点分裂 */
6)  endif
7)  if O. p≠NULL                                 /* 插入对象已存在轨迹 */
8)    list_insert(O, br)                         /* 新轨迹插入到单链表中 */
9)  else                                         /* 对象还没有轨迹 */
10)   hashhead(O, br)   /* 哈希表中对象头指针指向该轨迹 */
11) endif
END
```

2.2.2 轨迹删除

从 CR-L 树中删除某轨迹时,首先需找到存放该轨迹的叶子节点,再删除该轨迹所对应的入口项,并依次调整父节点索引项的 MBR 直至根节点。

如果轨迹删除后,出现叶子节点下溢现象,则需要进行节点合并,并对父节点进行相应调整。如果父节点也出现下溢,还需对父节点进行合并,依此类推。此外,在删除轨迹的同时,还需去除该轨迹与单链表的关联。具体删除算法如算法 2 所示。

算法 2　Delete(*tr*)。

输入　*O*— 对象标识; *tr*— 待删除的轨迹。

输出　1— 成功;0— 失败。

```
BEGIN
1)  lnode = entry_query(tr);                     /* 查询叶子节点 */
2)  entry_delete(tr);                            /* 轨迹删除 */
3)  lnode. count = lnode. count - 1;             /* 节点入口项数减 1 */
4)  if (lnode is in the underflow state)         /* 节点下溢 */
5)    node-merge(lnode)                          /* 节点合并 */
6)  end if
7)  list_delete(O, br)                           /* 从单链表中删除轨迹 */
END
```

2.2.3 二分裂法

如果需往一个满的节点中插入一个新的入口项时,须将该节点分裂为两个节点。二分裂法的分裂原则是尽可能使新产生两个节点 MBR 的 *M* 值之和最小。在具体分裂时,先选取两个种子,然后计算全部 branch 分别到这两个种子的 MBR 值,选择最小的组加入,如算法 3 所示。

算法 3　Bi-split(*lnode*, *tr*)。

输入　*lnode*— 待分裂的节点;*tr*— 待插入的轨迹。

输出　1— 成功;0— 失败。

```
BEGIN
1)  brs = get_branch(lnode);    /* 获得待分裂的全部 branch */
2)  brs = brs∪tr;
3)  pick_2seed(brs);                             /* 选取 2 个种子 */
4)  branch_classify(brs, p);    /* 对每个 branch 分类:0 或 1 */
5)  nnode = create_node()                        /* 创建新节点 */
6)  nnode. level = lnode. level;
7)  add_branch(nnode);          /* "0"标记 branch 放入新节点 */
8)  if (the parent node needs to split)          /* 父节点需分裂 */
9)    lnode = parent_node;
10)   goto 1)
11) end if
END
```

2.3 惰性分裂

惰性分裂技术是指节点溢出时不立即进行分裂,而是将待插入对象插入到邻近的兄弟未满节点中。该技术通过延迟节点分裂,以减少分裂次数并降低索引维护代价,同时可以节省存储空间。但是,惰性分裂会恶化节点的邻近性,因此当待插入节点及其邻近节点均已满时,需进行节点重组[12]。

为了防止惰性分裂引起索引结构混乱,本节所讨论的惰性分裂技术只适用于叶子节点。此外,为支持兄弟节点查找,对每个叶子节点增加指向父节点的指针*fathernode。

2.3.1 基于惰性分裂的插入算法

当待插入叶子节点中入口数目达到最大允许值 *MAXCARD* 时,并不立即分裂,而是先查看其兄弟节点是否有空余空间。若有,则将新轨迹插入到其兄弟节点中去;若所有兄弟节点都无空余空间,则进行惰性分裂,具体如算法 4 所示。

算法 4　lazy_insert(*tr*)。

输入　*O*— 对象标识; *tr*— 待插入的轨迹。

输出　1— 成功;0— 失败。

```
BEGIN
1)  lnode = entry_query(tr);
2)  if lnode. count < MAXCARD
3)    br = entry_insert(lnode, tr)
4)  else
5)    bn_set = brother_query(tr);
6)    if ∃bnode ∈ bn_set(bnode. count < MAXCARD)
7)      br = entry_insert(bnode, tr)
8)    else
9)      br = split_lazy(lnode, bn_set, tr)       /* 惰性分裂 */
```

```
10)    endif
11)  endif
12)  if O.p≠NULL
13)    list_insert(O, br)
14)  else
15)    hashhead(O, br)
16)  endif
END
```

2.3.2 惰性分裂

为了改善邻近性,因此当待插入节点及其邻近节点均已满时,需对节点进行重组,其依据仍然是分裂后的各个节点 MBR 值之和最小。惰性分裂大致步骤为:

1)获取待分裂节点及其兄弟节点所有入口项;

2)从全部入口项中选择 i 个种子;

3)计算所有入口到全部种子入口的 M 值,选择值最小的组加入;

4)创建新节点,将全部轨迹置于新节点、原节点及兄弟节点中。

具体算法不再赘述。

3 性能测试与分析

3.1 实验方案及参数

TPIR-Reorder[11] 是对 TPIR 方法的改进,它是通过对位置标识进行逻辑排序,使得查询目标结果尽量处于相邻近的磁盘上,因此该索引的创建依赖于目标查询结果。而本文主要研究支持不同类型 TQ 和 OQ 位置追溯的索引查询机制,因此目标查询结果不相同,难以建立适应所有查询类型的统一 TPIR-Reorder 索引。因此,本实验只比较 CR-L 和 TPIR 索引的性能,包括两种索引在不同类型查询下的查询时间开销、空间开销及索引构建时间开销等。

本实验通过程序随机产生大量 RFID 移动对象的运动轨迹。对于 OQ 查询,固定读写器数量(值为 500),变化标签对象数量(100,200,300,400,500);对于 TQ 查询,固定标签对象数量(500),变化阅读器数量(100, 200, 300, 400, 500)。设置每个标签对象都随机经过所有的读写器探测范围,产生的轨迹数目分别为 5 万、10 万、15 万、20 万。对于 OQ_look 和 TQ_look 查询,查询时间间隔为 10 min。实验统计的各类查询时间是随机产生 10000 个相关查询的运行时间。

3.2 实验结果及分析

图 6 是当读写器数量(500)不变时,改变对象数量时三种 OQ 查询的查询时间比较。从图 6 可看出,采用两种索引时,OQ 查询时间都随着对象数目的增加而线性增加。但是,采用 CR-L 时查询时间明显少于 TPIR,而且 CR-L 查询时间增长速度相对较慢,其主要原因是 CR-L 将相同读写器、相近时间内探测到的轨迹尽可能存放在相同或相邻节点中,大大减少了访问节点数目,从而节省了查询时间。

图 7 是当标签数量(500)不变时,改变读写器数量时三种 TQ 查询的时间比较。从图 7 可看出,两种索引的 TQ 查询时间都随着读写器数量的增加而增加。同样,基于 CR-L 的查询时间少于基于 TPIR 的查询时间。特别是 TQ_current 查询时间远远少于 TPIR。其原因是 CR-L 将相同对象的轨迹通过单链表链接起来,避免了对空间索引树的搜索和查找。而对于 TQ_current 查询,可通过哈希表的 $^*tail$ 指针直接得到。因此,单链表可大大提高 TQ 类型查询的查询效率。

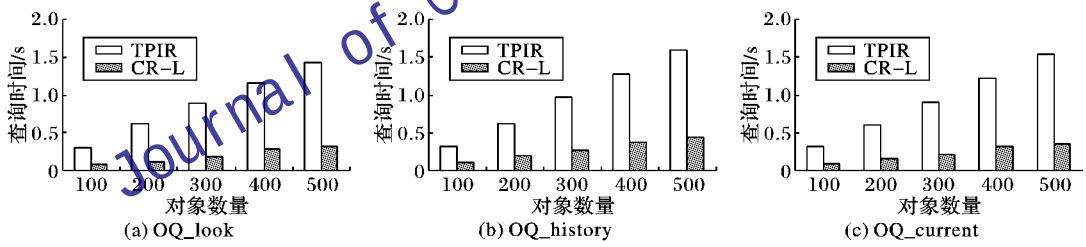

图 6 读写器数量(500)不变时不同对象数量的三种 OQ 查询时间比较

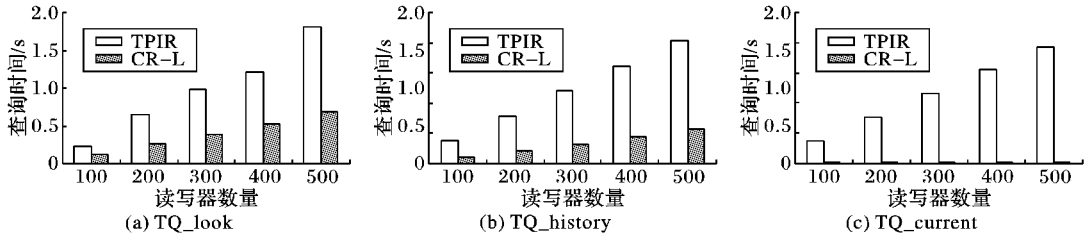

图 7 标签数量(500)不变时不同读写器数量的三种 TQ 查询时间比较

图 8 是 CR-L 采用不同分裂方法与 TPIR 空间开销比较结果。可以看出,采用惰性分裂的 CR-L(Lazy split)的存储空间最小,TPIR 次之,而采用二分裂法的 CR-L(Bi-split)最多。这是由于 CR-L 为提高 TQ 查询性能,增加了一个哈希表,而且对每个叶子节点增加了一个链接指针和指向父节点的指针,故采用二分裂法的存储空间比 TPIR 多。但是,由于惰性分裂允许将新对象插入到兄弟节点中,减少了新节点创建,从而节省了索引存储空间。

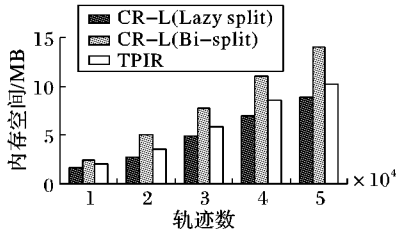

图 8 空间开销比较

图 9 是 CR-L 与 TPIR 索引构造时间比较结果。可以看

出,CR-L 构建时间略高于 TPIR 索引。这是因为 CR-L 索引除了要构建能反映轨迹包含关系的索引树之外,还需建立相同对象轨迹的单链表。

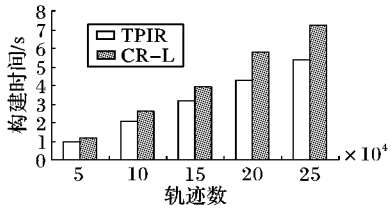

图 9　构建索引时间比较

4　结语

针对 RFID 移动对象的特征和位置追溯查询需求,本文设计了一种新颖的 RFID 移动对象索引——CR-L。为支持 OQ 查询,利用读写器、时间和标签对象三维信息设计了新的 MBR 值计算原则,将相同读写器在相近时间内探测到的轨迹尽可能聚类于相同或相邻节点,从而提高了 OQ 查询性能。同时,为提高 TQ 查询效率,采用单链表结构将相同对象的轨迹链接起来。实验结果表明,与已有 TPIR 索引相比,CR-L 索引除了构建时间略高于 TPIR 索引之外,在查询时间和存储空间两方面都具有较好的性能。

我们下一步工作是:一方面对所提出的索引进行优化,减少索引查询时间和构造时间;另一方面,对索引进行扩充以支持更多类型查询,如最近邻居查询等。

参考文献:

[1] TU W J, ZHOU W, PIRAMUTHU S. Identifying RFID-embedded objects in pervasive healthcare applications[J]. Decision Support Systems, 2009, 46(2): 586 – 593.

[2] WELBOURNE E, BATTLE L, COLE G, *et al*. Building the Internet of things using RFID[J]. IEEE Internet Computing, 2009, 13(3): 48 – 55.

[3] AGGARWAL C C. Managing and mining sensor data[M]. Berlin: Springer-Verlag, 2013.

[4] BAN C H, HONG B H, KIM D H. Time parameterized interval R-tree for tracing tags in RFID systems[C]// Proceedings of the 16th International Conference of Database and Expert Systems Applications, LNCS 3588. Berlin: Springer-Verlag, 2005: 503 – 513.

[5] LEE D, PARK J. RFID enabled supply chain traceability: existing methods, applications and challenges[EB/OL]. [2012-10-10]. http://www.intechopen.com/books/sustainable-radio-frequency-identification-solutions/rfid-enabled-supply-chain-traceability-existing-methods-applications-and-challenges.

[6] HUANG W P, LIN P L, LIN H Y. Optimizing storage utilization in R-tree dynamic index structure for spatial databases[J]. Journal of Systems and Software, 2001, 55(3): 291 – 299.

[7] ELLIS T K, ROUSSOPOULOS N, FALOUTSOS C. The R⁺-tree: a dynamic index for multi-dimensional objects[C]// Proceedings of the 13th International Conference on Very Large Data Base. San Francisco: Morgan Kaufmann, 1987: 507 – 518.

[8] BECKMANN N, KRIEGEL H P. The R*-tree: an efficient and robust access method for points and rectangles[C]// Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1990: 322 – 331.

[9] XIA T, ZHANG D. Improving the R*-tree with outlier handling techniques[C]// Proceedings of the 13th Annual ACM Workshop on Geographic Information Systems. New York: ACM Press, 2005: 125 – 134.

[10] PAN Y C, HU Z Y, GUO W, *et al*. QR-tree: a hybrid spatial index structure[C]// Proceedings of the 2rd International Conference on Machine Learning and Cybernetics. Washington, DC: IEEE Computer Society, 2003: 459 – 463.

[11] AHN S, HONG B. Reordering for indexing in an RFID tag database[J]. Journal of Information of Science and Engineering, 2009, 25(6): 1671 – 1690.

[12] LEI X, XIE K, HAN L, *et al*. A novel implementation of dynamic R-tree based on lazy splitting and clustering [J]. Journal of Computer Science, 2007, 34(4): 102 – 104. (雷小锋, 谢昆青, 韩亮, 等. 基于惰性聚类分裂的动态 R 树实现方法[J]. 计算机科学, 2007, 34(4): 102 – 104.)

(上接第 57 页)

[6] HOU L, LI Z, HU N. Neighboring point data recovery for CDP based on data gap[J]. Computer Science, 2011, 38(5): 159 – 163. (候利曼, 李战怀, 胡娜. 基于数据差异的 CDP 邻近时间点恢复[J]. 计算机科学, 2011, 38(5): 159 – 163.)

[7] LIU Z, ZHANG H, WEN Z, *et al*. Massive data continuous data protection technology research and implementation[J]. Journal of Computer Research and Development, 2012, 49(S1): 37 – 41. (刘正伟, 张华忠, 文中领, 等. 海量数据持续保护技术研究及实现[J]. 计算机研究与发展, 2012, 49(S1): 37 – 41.)

[8] LI X, TAN Y, LI Y. Snapshot method for continuous data protection systems[J]. Journal of Software, 2011, 22(10): 2523 – 2537. (李琥, 谭毓安, 李元章. 一种连续数据保护系统的快照方法[J]. 软件学报, 2011, 22(10): 2523 – 2537.)

[9] WANG L, WU K, WANG D, *et al*. Distributed storage model for continuous data protection[J]. Journal of Chinese Computer Systems, 2011, 32(8): 1587 – 1592. (王丽娜, 武开智, 王德军, 等. 一种面向连续数据保护的分布式存储模型研究[J]. 小型微型计算机系统, 2011, 32(8): 1587 – 1592.)

[10] YANG J, CAO Q, LI X, *et al*. ST-CDP: snapshots in TRAP for continuous data protection[J]. IEEE Transactions on Computers, 2012, 61(6): 753 – 766.

[11] LU M, SIMHA D, CHIUEH T-C. Scalable index update for block-level continuous data protect[C]// Proceedings of the 2011 6th IEEE International Conference on Networking, Architecture and Storage. Piscataway, NJ: IEEE Press, 2011.

[12] MORREY C B, III, GRUNWALD D. Peabody: the time travelling disk[C]// MSS 2003: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies. Washington, DC: IEEE Computer Society, 2003: 241.

[13] ZHU M B, LI K, PATTERSON R H. Efficient data storage system: US, 6928526[P]. 2010-07-21.

[14] LI X, XIE Z. An improved block-level continuous data protection mechanism[J]. Journal of Computer Research and Development, 2009, 46(5): 762 – 769. (李旭, 谢长生. 一种改进的块级连续数据保护机制[J]. 计算机研究与发展, 2009, 46(5): 762 – 769).

[15] YANG Q, XIAO W, REN J. TRAP-Array: a disk array architecture providing timely recovery to any point-in-time [C]// Proceedings of the 33rd International Symposium on Computer Architecture. New York: ACM Press, 2006: 289 – 301.