第35卷 第21期 计 算 机 工 程 2009年11月
Vol.35 No.21 Computer Engineering November 2009
·软件技术与数据库· 文章编号：1000—3428(2009)21—0082—03 文献标识码：A 中图分类号：TP311

基于KDB树的RFID事件聚合过滤算法

张丰贵，程良伦
(广东工业大学自动化学院，广州 510006)

摘 要：分析RFID中间件查询数据的特点，提出一种对查询数据聚合转换的方法，减少查询索引的存储空间和数据插入时间。分析和比较已有多维查询索引的各方面性能，将多维索引KDB-tree应用到RFID中间件中。实验结果表明，KDB树索引在存储空间成本、数据插入成本和查询时间成本3个方面的综合性能最佳，在点查询上，KDB-tree只须单路径遍历索引树，数据查询时间少于其他方法。

关键词：RFID中间件；聚合转换；KDB树；事件过滤

Algorithm for RFID Event Aggregation and Filtering Based on KDB-tree

ZHANG Feng-gui, CHENG Liang-lun
(College of Automation, Guangdong University of Technology, Guangzhou 510006)

【Abstract】The characteristics of query data for RFID middleware are analyzed. Based on the characteristics, a method named aggregate transformation is proposed to save storage cost of index and reduce insertion time. The performance of several existing multidimensional indexes is analyzed and compared, KDB-tree index is supplied into RFID middleware. Theoretic analysis and experimental results demonstrate that KDB-tree index outperforms others in synthesized consideration of storage cost, insertion time cost and query time cost. In particular the query time cost of KDB-tree is distinctly lower than others because it provides single-path traverse in the query processing.

【Key words】RFID middleware; aggregation transformation; KDB-tree; event filtering

1 概述

RFID中间件是位于读写器硬件与后端应用系统(如ERP)之间的媒介层，它是一套包括驱动程序管理、事件过滤与汇集、事件管理、安全管理以及网络管理的完整功能体系。RFID中间件通过对RFID设备的控制，可以实现对标签数据的实时采集与分析，并把预定义的应用逻辑与后台应用系统无缝整合，简化了RFID应用系统的集成与部署。为了对巨量的标签数据进行快速准确处理，RFID中间件必须具备很好的事件过滤性能，但目前RFID中间件的研究集中在中间件框架设计上。EPCglobal组织提出了与事件过滤有关的应用层事件(ALE)标准，该标准已经得到业界的广泛支持。本文研究RFID中间件的事件过滤功能，介绍一种基于聚合转换和KDB-tree的RFID中间件事件过滤算法，该算法符合EPCglobal的ALE标准。

2 RFID中间件事件过滤

EPCglobal提出了一种事件查询接口，分别是ECSpec和ECReportSpec。ECSpec用于向RFID中间件发出请求，ECReportSpec用于RFID中间件向应用程序发回报告。ECSpec描述了在哪个时间段对哪些RFID事件进行过滤，ECReportSpec指定如何报告过滤结果。RFID中间件会在ECSpec, ECReportSpec指定的“事件周期”内启动过滤服务和返回查询报告[1]。

ECSpec是RFID应用程序的标准接口，可以设置多个参数以建立多种查询。这些参数包含了RFID逻辑阅读器和标签的过滤条件，如一个“ReaderID=1-3, EPCpattern=<10.[1-2].[3001-4000]>”的ECSpec，第1部分是逻辑阅读器的范围，第2部分是标签的EPC(Electronic Product Code)模式，分别代表厂商、产品系列、产品序列号，通过ECPSpec可以定制RFID事件的查询。

3 已有的多维查询索引介绍

RFID中间件收集和过滤阅读器收集来的数据，这个工作量非常大。标签的持续运动导致标签数据会持续产生，多个EPCSpec定制了多个RFID标签事件范围的查询，这些查询使得RFID中间件需要根据多个定制的条件持续过滤标签数据。因此，RFID中间件要快速准确地从这么多标签数据中过滤出定制的事件，需要有很好的过滤算法。

对持续数据流的范围查询可以建立多种多维索引。如CQI(Cell-based Query Index)，这是一种基于内存的索引方案，它把查询域分割成多个单元，每个单元有覆盖查询条件或与查询条件交集的查询列表，利用这些列表去找到结果。VCR(Virtual Construct Rectangle index)[2]也是一种基于内存的低成本的搜索方案，它把查询域分割成段，这些段用VCs(Virtual Constructs)封装，并在VCs中插入查询ID，遍历这些VCs找到结果。但这些方法都把持续查询看成一个域，而基于ECSpec的查询不是一个域而是许多的段，如果把这些方法应用到RFID查询索引中，则需要太多的存储空间和数据插入次数，因此，这些方法很难应用到RFID查询中。

基金项目：国家自然科学基金资助项目“RFID传感器网络的关键问题研究”(60673132)

作者简介：张丰贵(1983－)，男，硕士研究生，主研方向：RFID中间件；程良伦，教授、博士

收稿日期：2009-04-27 **E-mail：**zfg168@sina.com

为了避免太多的存储空间和太多的数据插入次数，可以使用多维索引 R-Tree，索引中存储四维数据：RID, manager, product 和 serial[3]，然而在高维数情况下，利用 R-Tree 索引的查询性能并不好。

鉴于以上各种多维索引的缺点，本文提出一种适合于 RFID 中间件的查询索引 KDB-tree，KDB-tree 是多维索引，它可以减少数据插入次数，使点查询只需单路径遍历索引树，即一个点查询对应单一的一条从根到叶子的路径，而且在动态插入时总保持树的平衡[4]，因此，利用这种索引会有很好的查询性能。

4 查询数据的聚合转换

为说明问题，给出以下 2 个定义：

定义 1 RFID 中间件每次从阅读器采集一个标签数据并对其进行过滤，相当于一次点查询，表示为 *PointQuery*(*RIDi*, *TIDi*)。

定义 2 查询数据代表一个基于 ECSpec 的连续查询，即阅读器和标签的过滤条件。用 *RID* 和 *TID* 分别表示阅读器的范围和标签的过滤条件。

查询数据包含二维空间(*RID, TID*)的单个或多个段。一个查询段是组成复杂查询数据的最小单位，本文把查询数据设为 *d*, *d*={*d*1, *d*2, …, *dn*}，其中，$1 \le n \le 2^{24}$。查询数据段表示为 *di*={(*minrid, mintid*), (*maxrid, maxtid*)}，其中，*di*∈*d*。

在二维空间(*RID, TID*)的查询数据由若干离散段组成。一个查询数据的例子如图 1 所示。

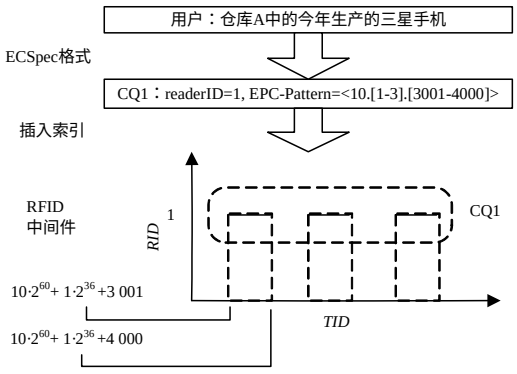


图 1 查询数据示例

用户搜索“仓库 A 中的今年生产的三星手机”之后，应用系统发出一个 ECspec，包含 CQ1:*readerID*=1, *EPC-Pattern*=<10.[1-3].[3 001-4 000]> [5]，假设安装在仓库 A 的阅读器 ID 是 1，CQ1 到达 RFID 中间件并插入到查询索引，那么查询数据有 3 个离散段：

(1){(1, $10\cdot2^{60}+1\cdot2^{36}+3\,001$), (1, $10\cdot2^{60}+1\cdot2^{36}+4\,000$)}；

(2){(1, $10\cdot2^{60}+2\cdot2^{36}+3\,001$), (1, $10\cdot2^{60}+2\cdot2^{36}+4\,000$)}；

(3){(1, $10\cdot2^{60}+3\cdot2^{36}+3\,001$), (1, $10\cdot2^{60}+3\cdot2^{36}+4\,000$)}。

由此可知，如果 EPC 模式的产品号 *P* 是一个范围，这时查询数据会包含多个段。如图 1 所示，段的大小取决于序列号的范围，段的数目等于序列号的范围。查询数据是一个复杂对象，最多包含 2^{24} 段，因为标签 ID 的产品系列号是 24 位的[3]。这样当保存一个 EPCspec 连续查询时，需要插入很多段到索引中，多次插入操作后索引会变得很庞大，这时点查询的速度会很慢。

为了避免多次插入，需要减少查询数据的段，本文提出一种聚合转换方法，把 RID 和 ECSpec 模式组成的二维数据(*RID, TID*)转化成三维数据(*RID, SID, PID*)。ECSpec 模式有 27 种模式，而只有 11 种模式是有意义的，如表 1 所示。

表 1 EPC 模式中 11 种有意义的模式

EPC 模式	单段多段	拆分段数
m.s.p	单	1
m.s.*	单	1
m.s.[p1-p2]	单	1
m.*.p	多	2^{24}
m.*.*	单	1
m.*.[p1-p2]	多	2^{24}
m.[s1-s2].p	多	s1-s2+1
m.[s1-s2].*	单	1
m.[s1-s2]. [p1-p2]	多	s1-s2+1
[m1-m2].*.*	单	1
..*	单	1

查询索引中的查询数据取决于 EPC 模式，EPC 模式中包含 Manager, Serial, Product 3 个部分，每部分都是常数或“[low- high]”或“*”之一，因此，ECP 模式有 $3^3=27$ 种。Manager 是给每个厂家分配的全球唯一标识，而 Serial 和 Product 由厂家自定义[3]，当 manager 定义为一个范围时，如果 serial 或 product 也定义为一个范围，查询数据就没有意义，除了“[m1-m2].*.*”和“*.*.*”。这时把“$m*2^{24}+s$”定义为 *S* 维度，RID 定义为 *R* 维度，p 定义为 *P* 维度，那么任何查询数据在三维空间(*R, S, P*)中只需用一个段 *d* 即可表示，而不需要拆分为多个段，*d* 表示为 *d*={(*minRID, minSID, minPID*), (*maxRID, maxSID, maxPID*)}。每次从应用程序传来一个查询数据，RFID 中间件只需把它转化成一个查询段，如图 2 所示，这样就避免了多次向索引进行插入查询段。

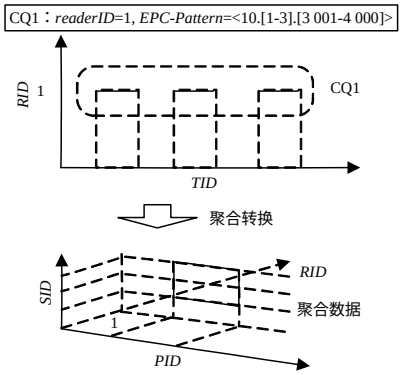


图 2 对图 1 的查询数据进行聚合转换的过程

5 基于 KDB-tree 索引的点查询

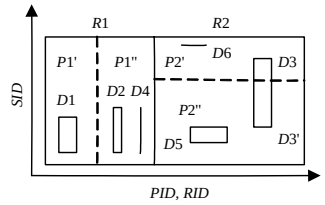
KDB-tree 是多维索引，比其他多维索引有更好的查询性能，特别在点查询上，它提供了单路径树遍历，即任意一个点查询的路径对应单一的一条从根到叶子的路径，而且在动态插入时总保持树的平衡。RFID 中间件从阅读器采集一个标签数据，把它转化成三维数据(*RID, SID, PID*)，并在 KDB-tree 索引中查找匹配的查询数据段，这是一个点查询的过程。其查询过程用 *PointQuery* (*root, d*)函数实现，其中，*d* 是一个查询数据段。点查询函数如下：

```
PointQuery(root,d)
    If root is NULL return FALSE
    for each entry e
        If Contain(e,p) is TRUE then
          If root is a leaf
              Return TRUE
          If root is not a leaf
 PointQuery(e.son,d)
```

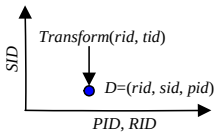
点查询的具体过程是从 KDB-tree 的根节点开始访问，如

果访问的节点为空则说明点查询未找到包含 *d* 的查询数据段，接着先序遍历各节点，对节点中的每个段 *e* 调用 *Contain* (*e*, *d*)，返回 TRUE 时判断该节点是否为叶节点，如果是说明已经找到包含 *d* 的查询数据段，否则递归调用点查询过程。

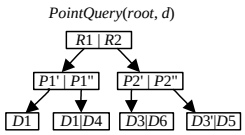
点查询的示例如图 3 所示。图 3(a)中用矩形表示各查询数据段，此时 *RID*=1，因此，在二维空间中可以表示出所有的查询数据段。图 3(b)是在进行点查询之前调用 *transform*() 把阅读器送来的(*RID*, *TID*)转换成 *d*=(*RID*, *SID*, *PID*)。图 3(c)是所有查询数据段在 KDB-tree 索引中的位置，以及 *PointQuery* (*root*, *d*)函数在 R-Tree 索引中的搜索过程。



(a)过程 1



(b)过程 2



(c)过程 3

图 3　点查询过程

6　实验结果与分析

本节比较KDB-tree, R-Tree, CQI和VCR索引的各方面性能，KDB-tree 索引建立在三维空间(*RID*, *PID*, *SID*)上，R-Tree 索引建立在四维空间(*RID*, *Manager*, *Product*, *Serial*)上，而 CQI 和 VCR 索引建立在二维空间(*RID*, *TID*)上。这些索引都保存在内存中，使用 Wall Clock Time 作为度量，程序运行环境是 PIV 2.6 GHz 处理器，1 GB 内存和 Windows XP 操作系统。各索引的实验使用相同的 ECSpec 查询数据集和相同的点查询的数据集。首先测试比较 KDB-tree, R-Tree, CQI 和 VCR 的存储空间成本，分别执行 5 000, 10 000, 50 000, 100 000 个查询数据插入，实验结果如图 4 所示。

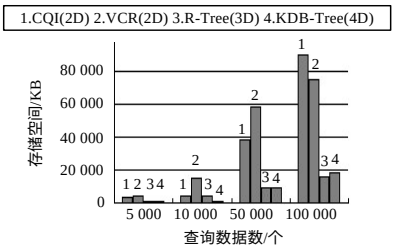


图 4　存储空间成本对比

CQI 和 VCR 索引需要最高的存储成本，因为它们需要把查询数据拆分成多个段，然后把所有的段插入的索引中。R-Tree 是的存储成本是最低的，它把一个查询数据转换成一个四维数据，直接向索引插入一次数据即可。R-Tree 的存储成本最少，而 KDB-tree 比 R-Tree 的存储成本稍微高一点，它和 R-Tree 类似，把一个查询数据转换成一个多维数据，但为避免索引节点数据域的重叠，有时需要允许分裂算法，这样在索引中会多出一些查询数据段，然而，也因为各节点数据域的非重叠，KDB-tree 索引搜索时只需单路径遍历，大大缩短了搜索时间。

测试比较 KDB-tree, R-Tree, CQI 和 VCR 的插入时间成本，分别执行 5 000, 10 000, 50 000, 100 000 个查询数据插入，实验结果如图 5 所示。

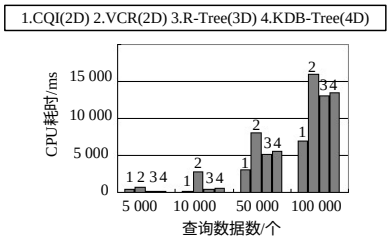


图 5　插入时间成本对比

CQI 索引的插入时间成本最低，因为其数据插入过程非常简单和直观，而 VCR 的插入时间成本最高，源于它需要把查询数据分成很多段，另外还需要比较索引中的节点数据再分别插入。而 KDB-tree 和 R-Tree 的插入时间成本相近，但是 CQI 和 VCR 索引在点查询过程中还需要一个精简处理过程。测试对比它们的搜索性能，改变点查询的次数，测试结果如图 6 所示。CQI 和 VCR 花费时间的最多，因为它们需要顺序比较的节点数据，而这些节点数据太多，所以搜索性能最差。KDB-tree 的搜索性能比 R-Tree 还要好，因为在 KDB-tree 索引中进行点查询，可以单路径遍历各个节点，而在 R-Tree 索引中需要多路径遍历，所以在 KDB-tree 索引中的查询性能是最好的。

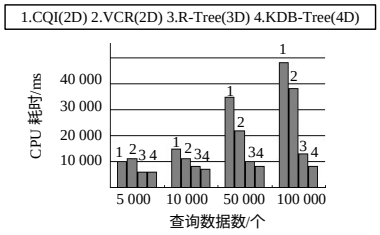


图 6　搜索性能对比

7　结束语

本文提出的基于聚合转换和 KDB-tree 的 RFID 中间件事件过滤算法可以应用到实际的 RFID 系统中，使 RFID 中间件更快速准确地过滤巨量的标签数据，提升 RFID 系统的性能。

参考文献

[1] EPCglobal. The Application Level Event(ALE) Specification Version 1.1[Z]. EPCglobal Standard Specification. 2008.

[2] Wu Kun-Lung, Chen Shyh-Kwei, Yu P S. Processing Continual Range Queries over Moving Objects Using VCR-based Query Indexes[C]//Proc. of International Conf. on Mobile and Ubiquitous Systems. Cambridge, MA, USA: [s. n.], 2004.

[3] EPCglobal. EPC Tag Data Standards Version 1.3[Z]. EPCglobal Standard Specification. 2005.

[4] Robinson J T. The KDB-tree: A Search Structure for Large Multidimensional Dynamic Indexes[C]//Proc. of ACM SIGMOD'81. New York, USA: ACM Press, 1981.

[5] Zarokostas N, Dimitropoulos P D, Soldatos J. RFID Middleware Design for Enhancing Traceability in the Supply Chain Management[C]//Proc. of PIMRC'07. Athens, Greece: [s. n.], 2007.

编辑　顾姣健