一种基于 Trie 树和扩展 B 树的 RFID 标签编码过滤方法研究

赵 文[1,2],刘学洋[1,2],刘殿兴[3],王立福[1,2]
(1.北京大学软件工程国家工程研究中心,北京 100871;
2.北京大学信息科学技术学院软件研究所高可信软件技术教育部重点实验室,北京 100871;
3.北京银行信息技术总部,北京 100011)

摘 要: 面向 EPC 模式(EPC Pattern)的标签编码过滤是 RFID 中间件的主要功能之一.为提高过滤效率,本文给出了一种基于 trie 树和扩展 B 树相结合的标签编码过滤方法.通过分析标签编码和 EPC 模式的结构特征,将系统中大量的 EPC 模式构造成一个层次查找结构,对于 EPC 模式中的常规编码段采用 trie 树表达,对于区间形式采用扩展 B 树表达.查找过程按照编码段由高至低依次进行,并采用了基于位向量集合的优化方法.实验表明标签编码过滤效率受 EPC 模式数量变化的影响较小,能够有效降低向上层应用传输数据的延迟.

关键词: RFID; 中间件; EPC 模式; 过滤; trie 树; B 树
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2011) 3A-126-08

Research on RFID Tag Code Filtering Method Based on Trie Tree and Extended B Tree

ZHAO Wen[1,2], LIU Xue-yang[1,2], Liu Dian-xing[3], Wang Li-fu[1,2]
(1. *National Engineering Research Center for Software Engineering, Peking University, Beijing* 100871, *China*;
2. *Key laboratory of High Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
3. *Information Technology Department, Bank of Beijing, Beijing* 100011, *China*)

Abstract: Tag code filtering by EPC Patterns is one of the main functions of RFID middleware. In order to improve filtering efficiency, a trie tree and extended B tree based tag code filtering method is proposed. By analyzing the structural features of tag code and EPC Pattern, a layered searching structure is constructed with the EPC Patterns in the system, trie tree is used for expressing normal code sections of the EPC Patterns, extended B tree is used for expressing range forms. The search in layered structure is conducted with the descending order of code segments, and searching process is also optimized based on bit vector section set. The simulation tests show that the efficiency of our method to filter tag code suffers little effect by the change of EPC Pattern quantity and can effectively reduce the data transmission delay to the upper application.

Key words: RFID; middleware; EPC Pattern; filtering; trie tree; B tree

1 引言

RFID 中间件的作用是从各种输入设备(读写器、传感器等)获取数据,进行必要的冗余处理、数据过滤和数据分组,生成应用层事件,然后将该事件发送到各种企业应用和数据库进行存储.RFID 中间件的作用主要包括两个方面:一是事件的管理,即将采集到标签数据按照过滤规则和分组规则形成应用层事件,并将形成的事件上传给相关的上层应用;二是读写器设备的管理,即操纵控制 RFID 读写设备按照预定的方式工作,实现物理读写器与逻辑读写器之间的映射,保证不同读写设备之间协同工作.

由于 RFID 中间件在 RFID 应用中处于承上启下的关键位置,因而其执行效率对 RFID 应用有很大的影响,低下的执行效率会产生向上层传输的延迟,成为数据传输的瓶颈.目前对 RFID 中间件执行效率的研究大多集中在标签编码的冗余过滤上,还很少考虑基于 EPC 模式的过滤优化,这使得 RFID 中间件的过滤效率会随着 EPC 模式数量的增多而显著降低,特别是在 RFID 中间件管理多个读写器的情况下,单位时间内上传的标签编

收稿日期:2010-08-31;修回日期:2010-11-15
基金项目:国家 863 高技术研究发展计划(No.2006AA04A119,No.2006AA04A121);国家 973 重点基础研究发展计划(No.2009CB320706);国家自然科学基金(No.60803014)

码数量可能很大,这种累计的延迟会导致缓存溢出从而使中间件崩溃.

本文研究 RFID 中间件中面向 EPC 模式的标签编码过滤技术.通过分析标签编码和 EPC 模式的结构特征,将系统中维护的大量 EPC 模式构成一个查找结构,给出了基于 trie 树和扩展 B 树相结合的标签编码过滤方法.采用该方法过滤标签编码的效率受 EPC 模式数量变化的影响较小,使 RFID 中间件能够适应海量标签数据的采集,有效降低向上层应用传输数据的延迟.

2　相关研究介绍

(1)**标准规范**　2005 年 9 月,EPCglobal 组织提出了 RFID 中间件相关标准 The Application Level Events (ALE) Specification Version 1.0[1],该标准指出标签数据的采集、过滤和分组,从而使 RFID 中间件上升到标准化阶段.2008 年和 2009 年 EPCglobal 组织又从实现的角度对该标准进行了补充,分别推出了两个版本的 ALE 规约[2,3].依据上述和其他相关标准和规范,目前 RFID 中间件的研究主要集中在提高 RFID 中间件性能、RFID 中间件系统架构,以及支持复杂事件处理三个主要方面,本文的研究可以归为第一类.

(2)**冗余过滤**　目前对提高 RFID 中间件性能的研究主要集中在提高冗余过滤的效率上,但到目前为止尚无成熟的标准规范可以遵循.文献[4]提出了一种基于滑动窗口的平滑过滤方法,该方法通过在一个滑动窗口的时隙中赋予每个标签多次的读取机会来降低标签编码的漏读率;文献[5]提出了一种叫做 SMURF 的冗余过滤方法,它也是基于平滑过滤的思想,不过可以通过用户指定的 CQL[6]语言来声明过滤规则,并且有自适应调节的功能;文献[7]提出了一种面向 UHF 频段的冗余过滤方法,叫做带通过滤,该方法与硬件设备和空中接口密切相关;国内中科院自动化所也做了这方面的相关研究,提出了基于标签编码哈希表的过滤方法,给出了面向三种特定应用的冗余过滤器的实现[8].

(3)**基于 EPC 模式的过滤**　提高 RFID 中间件效率的另外一个方面就是如何按照上层定制的用来描述事件周期和产生相关报告的 ECSpec[3]来快速生成 ALE,其核心就是面向 EPC 模式的标签编码过滤.到目前为止,这方面的研究大多集中在通过并行计算[9]和负载均衡[10,11]的策略来提高其效率.

基于上述标准规范和研究工作,本文以 RFID 中间件中面向 EPC 模式的标签编码过滤技术为主要研究内容,通过分析标签编码和 EPC 模式的结构,对系统中维护的大量 EPC 模式进行预处理,来提高面向 EPC 模式的标签编码过滤效率,减小 EPC 模式数量的变化对过滤效率的影响,降低向上层应用传输数据的延迟.

3　标签编码与 EPC 模式的结构

3.1　标签编码结构

EPC 是 EPCglobal 提出的一种单品级别的编码结构,它定义了一种 GID-96 编码[12],同时兼容了目前广泛应用的商业标准 EAN/UCC(包括 GTIN、SSCC、GLN、GRAI、GIAI),以及 DoD Identity Type.GID 编码的格式如下:

urn:epc:id:gid:GeneralManagerNumber.ObjectClass.SerialNumber

其结构可以分为四个组成部分(编码段):头部,组织编码,物品类别码和单品序列码.

- 头部(Head):标识了 EPC 的类型,长度,版本,组成结构等相关信息;
- 组织编码(GeneralManagerNumber):标识一个组织;
- 物品类别码(ObjectClass):标识特定组织下的一个物品类别;
- 单品序列码(SerialNumber):标识特定物品类别下的一个单品.

例如 urn:epc:id:gid:123789.302414.169740 就是一个 EPC,其中 urn:epc:id:gid 是头部,表明采用的 GID-96 编码结构,123789 是组织编码,302414 是物品类别码,169740 是单品序列码.

3.2　EPC 模式的结构

EPC 模式是对标签编码集合的规约,通常用于对采集的标签编码进行过滤和分组,不同的上层应用通过在 ECSpec 中指定不同的 EPC 模式来表达它们对哪些标签感兴趣.EPC 模式的结构和标签编码结构一样,也包括四个部分,通过在不同的编码段中设定特殊标识来对编码集合进行规约.EPC 模式中三种特殊标识如下:

- "*"是一个通配符,规约了所在编码段的全部值,例如:如果 EPC 模式中的 ObjectClass 部分为"*",则表示所有物品类别;
- "[low-high]"是一个编码范围,规约了所在编码段的取值范围,例如:如果 EPC 模式中的 SerialNumber 部分为"[100-500]",则表示单品序列码在 100 至 500 范围内的单品;
- "X"是一个分组标识,表示按照"X"所在编码段的不同取值来分组.

在 EPC 模式中,我们称不含上述三种特殊编码以外的编码段为常规编码段.严格来说,通过 EPC 模式中的常规编码形式就足以规约所有可能的标签编码集合.这是因为不同种类的标签编码都是有固定长度的,从而决定标签编码具有最大值,极端情况下可以通过一一列举来表达所需要的标签编码.EPC 模式中提供的三种特殊标识只是为了更方便实际应用对标签编码集

合规约的表达,因而 EPC 模式足以表达实际应用对标签编码过滤的需求.

4 面向 EPC 模式的过滤方法

基于 EPC 模式的过滤实质上是一种基于规则的过滤,与其他基于规则的过滤技术相比较,其特殊性主要体现在标签编码结构和 EPC 模式结构的不同.通过对标签编码结构和 EPC 模式结构的分析,给出了面向 EPC 模式的标签编码过滤方法,总体思路如图 1 所示.

过滤方法的核心是将系统中的多个 EPC 模式按照编码段进行划分,在每一个编码段,将常规编码构造成一棵 trie 树,将"[low-high]"形式的编码区间规约构造成一棵扩展 B 树,两者的叶子节点保存了从树根到树叶经过的路径所匹配的 EPC 模式集合,"*"为单独一个节点.当一个标签编码到来时,按照编码段从高到低依次查找每一层次的 trie 树、扩展 B 树以及"*"节点,从而得到每一层所匹配的 EPC 模式集合,通过取每一层次查找结果的交集来得到标签编码最终满足的 EPC 模式集合.

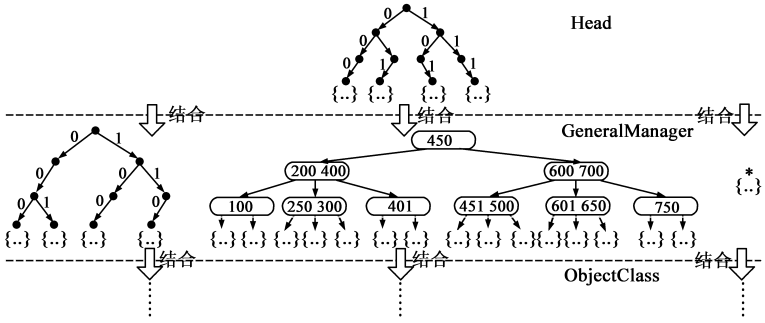


图1 面向EPC模式的标签编码过滤方法

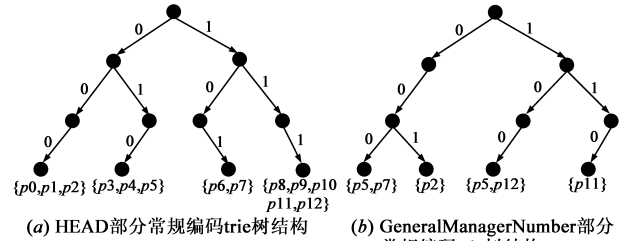
4.1 常规编码段 trie 树

与标签编码相似,EPC 模式也可以按照编码段进行划分,表 1 给出了简化的示例,其中 Head 部分只包括由二进制形式表示的常规编码,而 GeneralManagerNumber 部分既有常规编码还有区间"[low-high]"的形式.对于每一个编码层次中的常规编码部分,按照其二进制数值由高位到低位构建 trie 树,并根据二进制值确定分支方向,0 为左分支,1 为右分支,在叶节点存储 EPC 模式的集合.对于表中某个 EPC 模式的某一个编码部分,若其二进制表达式与 trie 结构的某条分支匹配,则该分支的叶节点记录该 EPC 模式.

表 1 EPC 模式表

PatternName	Head(binary)	GeneralManagerNumber	ObjectClass
p0	000	[100 – 200]	*
*p*1	000	[300 – 400]	*
*p*2	000	001	001
*p*3	010	[250 – 450]	*
*p*4	010	[401 – 500]	*
*p*5	010	000	100
*p*6	101	[451 – 600]	*
*p*7	101	000	110
*p*8	111	[601 – 700]	*
*p*9	111	[650 – 750]	*
*p*10	111	100	000
*p*11	111	110	010
*p*12	111	100	001

图 2 中的(*a*),(*b*)分别给出了表 1 中 Head 和 GeneralManagerNumber 编码段中常规编码对应的 trie 树结构(为了简明的说明问题,常规编码部分只给出了三个二进制位).



(*a*) HEAD部分常规编码trie树结构 (*b*) GeneralManagerNumber部分常规编码trie树结构

图2 常规编码段trie树结构示例

4.2 基于区间的扩展 B 树

在 EPC 模式中,有很大一部分是区间"[low-high]"的形式,如表 1 中的 GeneralManagerNumber 部分.这些区间可能存在很多的重叠部分(例如同样的一批货物可能有不同的上层应用来关注),如何快速的找到标签编码所匹配的区间是进行高效过滤的关键,在本节给出了基于区间的扩展 B 树的查找方法.

为了适应区间查找,需要对区间进行划分.区间的

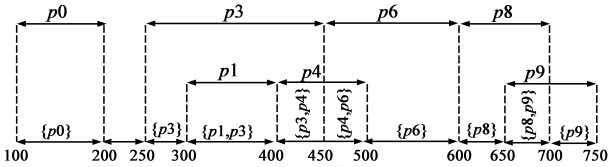


图3 区间的划分示例

划分相对简单,即利用所有区间的端点来分隔整个编码空间,相应地也就可以得到划分后的每个区间所对应的 EPC 模式.图 3 给出了表 1 的 GeneralManagerNumber 中多个区间的划分示例.

下一步工作就是将区间划分后的端点值组织成一棵 B 树.考虑采用 B 树而不采用线性表的方式主要是为了节省存储空间,此外 B 树的平衡性保证了在最坏情况下的查找性能.由于 B 树执行的是精确查找(即在查找过程中,如果找到与查找关键字相等的值则返回,否则找不到相应的节点),为了适应面向 EPC 模式的过滤,对 B 树进行适当扩展如下:

(1)B 树的叶节点扩展出 k 个虚分支($2 \leqslant k \leqslant m$, m 是 B 树的阶),每个虚分支指向一个扩展叶节点,扩展叶节点存储了与所在区间相对应的 EPC 模式.由于每个叶节点都有虚分支,因而扩展后的 B 树仍然是平衡的;

(2)对 B 树的查找过程也做适当修改,即当编码数值小于等于某个区间端点时走该端点的左子树,否则走该端点的右子树.这样就可以保证一定能够查找到 B 树的扩展叶节点.

图 4 是按照图 3 中的区间划分结果构造成的 3 阶 B树.以查找编码值 350 为例,由于 350 小于 450,所以走左子树,由于 350 大于 200 小于 400,因而走 200 的右子树也就是 400 的左子树,以此向下查找,最后找到编码值 350 匹配了 $p1$ 和 $p3$.

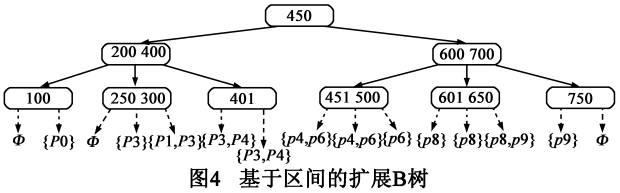


图4　基于区间的扩展B树

4.3　不同层次查找结构的结合

通常在一个 EPC 模式中,既含有常规编码段,又含有区间编码段,这就需要将两者结合起来.当一个标签编码到来时,一种简单的方式就是根据标签编码的不同编码段找到对应的查找结构,然后计算不同编码段对应的查找结果的交集即可.这种方法直观,但是这种方法在低层编码段的查找中没有充分利用高层编码段已经得到的查找结果,因而效率相对较低.为此,本文在不同编码段的查找结构相结合时进行了一定的优化,进一步减少了比较的次数.图 5 是表 1 中 Head 部分 trie 树和 GeneralManagerNumber 部分的区间扩展 B 树相结合的例子.在表 1 中,对于 GeneralManagerNumber 段,由于 $p0$ 对应的区间是[100 – 200],而 $p2$ 对应的区间是[300 – 400],因而 trie 中叶节点$\{p0, p1, p2\}$对应的区间范围就是[100 – 400],因而当一个标签编码的 Head 部分在 trie 树中查找完成后(例如找到节点$\{p0, p1, p2\}$),就可以利用 trie 树叶节点对应的区间范围来减少在下层扩展 B 树中的查找比较次数(例如可以直接从节点(200 400)发起查询,而不必从根节点 450 发起查询,从而减少了查询比较的次数).

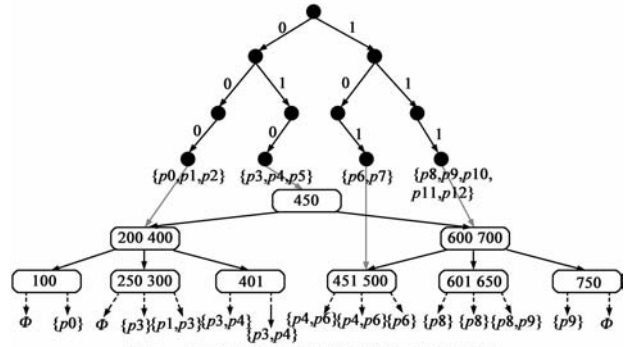


图5　与下层基于区间的扩展B树的结合

通过上述方式,在上层的查找结构与下层基于区间的扩展 B 树相结合时,可以对结合部分进行预处理,如图 5 中的层间连线所示.

上层的查找结构与下层的 trie 树结合可以仿照与扩展 B 树相结合的方式,不同的是根据上层查找结构中的 EPC 模式集合的最大共享前缀来得到在下层 trie 树中的起始查找位置.图 6 是一个示例,$\{p2, p5\}$在下层 trie 树中有共享前缀“00”,因而在下层 trie 树中发起查找时不需要从根节点开始,从而减少了比较的次数.

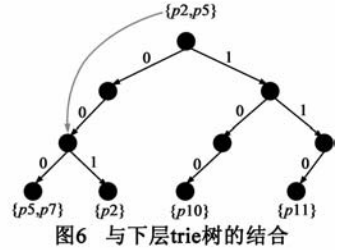


图6　与下层trie树的结合

4.4　EPC 模式集合的位向量段集合表示与查找优化

当一个标签编码到来时,按照编码段从高到低依次到每一层次的查找结构中查找,通过取每一层次查找结果的交集来得到编码最终满足的 EPC 模式集合.如何加快不同编码段查找结果交集的计算对算法性能有一定影响.基于 EPC 模式规约的编码空间呈现稀疏聚集的性质,我们将查找结构中叶节点对应 EPC 模式集合用位向量段集合来表示,在不同层次查找结果交集的计算中,只要计算同段位向量段的与操作就可以得出结果,从而提高算法效率.

采用位向量段集合来表示 EPC 模式集合是基于 EPC 模式规约的编码空间呈现稀疏聚集的性质.例如一个业务位置(business location)通常只跟一些相对固定的合作伙伴有业务往来,那么该业务位置所关注的物品

标签编码也跟这些合作企业有关,相应的 EPC 模式也就规约了这部分编码空间,也就是说这些 EPC 模式所规约的编码空间相对集中,但是每个相对集中的位置可能距离较远.如果把在 EPC 模式表中将编码空间相对较近的 EPC 模式放在一起,就可以将查找结构叶节点中的多个 EPC 模式用少量的位向量段来表示,这样不仅能节省空间,而且能够通过同段位向量段的与操作来加快交集的计算.

每个位向量段是形如〈*SecID*, *Vec*〉的二元组,其中 *SecID* 为位向量段的段号,*Vec* 为位向量段的实际值.一个 EPC 模式对应的位向量段号 *SecID* 由它在 EPC 模式表中的位置决定,如果位向量段的二进制长度为 len(*Vec*),那么 EPC 模式表中从第 len(*Vec*) * (*SecID*-1) + 1 个到第 len(*Vec*) * *SecID* 个 EPC 模式对应的位向量段号就为 *SecID*,相应的 *Vec* 中的第 *i* 位就对应 EPC 模式表中的第(*SecID*-1) * len(*Vec*) + *i* 条 EPC 模式.由于在 EPC 模式表中将编码空间相对较近的 EPC 模式放在一起,所以查找结构中叶节点所包含的 EPC 模式在 EPC 模式表中也相距较近,这样就可以用尽可能少的位向量段来表示尽可能多的 EPC 模式.在不同层次查找结果取交集的计算中,只要计算少量同段位向量段的与操作便可得出结果.采用上述方法为图 3 中的每个区间分配的位向量段如表 2 所示.

表 2 位向量段分配举例

区间	位向量段
[100,200]	(1,0x01)
[250,299]	(1,0×08)
[300,400]	(2,0×0A)
[401,450]	(1,0×18)
[451,500]	(1,0×50)
[501,600]	(1,0×40)
[601,650]	(2,0×01)
[651,700]	(2,0×03)
[701,750]	(2,0×02)

采用同样的方式我们也可以将 trie 树叶节点中的 EPC 模式集合以及"*"节点中的 pattern 集合表示为位向量段的集合.将图 5 中的 EPC 模式集合表示为位向量段的集合后如图 7 所示.

编码过滤效率的提高在于预先将系统内部多个 EPC 模式构造成一个层次查找结构(trie 树 + B 树),从而保证每条编码的过滤都可以在相对固定的有限步中完成,而不受 EPC 模式数量的影响,而位向量段集合表示只是作为加快计算的一个补充.

4.5 更新操作

对查找结构的更新操作主要增加或删除一条 EPC 模式的操作.增加操作的过程如下:对于 EPC 模式中的常规编码段,首先要找到所对应的 trie 树,然后在 trie 树中查找,如果能找到叶节点,则在叶节点中增加该 Pattern,如果找不到叶节点,则按照其二进制编码增加一条从根节点到叶节点的路径,并在叶节点中加入该 Pattern;对于 EPC 模式中的区间形式,首先要找到所对应的扩展 B 树,然后用端点值在扩展 B 树中进行查找,找到端点应该插入的位置并插入,接着利用 B 树的更新算法来对 B 树进行更新,从而保证 B 树的平衡性,最后根据该区间所覆盖的区间划分,在扩展叶节点中添加该 Pattern.接下来就是要更新不同层次查找结构的结合部分,与 4.4 节中的过程一致.为了减小删除/新增 EPC 模式带来的位向量段调整的复杂性和时空消耗,在删除时并不调整 EPC 模式表中记录的顺序,而仅重新计算删除的 EPC 模式所在位向量段的所有位向量;在新增时,从 EPC 模式的表中的首行开始查找到第一个空白行然后插入,并重新计算新增的 EPC 模式所在位向量段的所有位向量;只有在 RFID 中间件重启时在按照稀疏聚集的性质重新计算生成新的 EPC 模式表,并重新计算所有的位向量.删除操作与增加操作基本过程相似,篇幅所限就不再赘述.

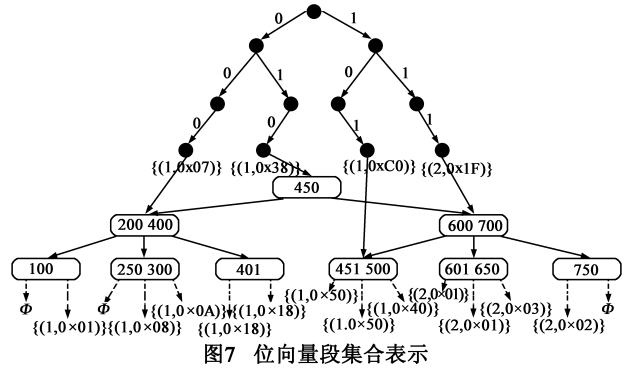


图7 位向量段集合表示

4.6 算法的详细描述

(1)数据结构

● 向量段格式为 *VectorSection* ≡〈*SecID*, *Vec*〉. *SecID* 为向量段的段号,*Vec* 为向量,*Vec* 中的第 *i* 位对应规则库中的第 *SecID* * len(*Vec*) + *i* 条规则.实现时可以采用 map 数据结构,而 *SecID* 和 *Vec* 均为整型.

● *Node* 是系统中节点的父类,为在 trie 树,扩展 B 树和"*"节点中的操作提供统一的表示.

● trie 树与扩展 B 树中的叶节点为 *LeafNode* ≡〈*VectorSections*, *Nodes*〉.其中 *VectorSections* 为 *VectorSection* 的集合,每一个 *VectorSection* 的 *Vec* 中必须有含"1"的位;*Nodes* 是 *Node* 的集合,第一个元素指向下一层 trie 树中的节点,第二个元素指向下一层扩展 B 树中的节点,第三个元素指向下一层的"*"节点.

● 扩展 B 树的非叶节点格式为 *BTNode* ≡〈*keyNum*, *parent*, *keys*, *Nodes*〉.其中 *keyNum* 是 *keys* 的数量,*keys* 是区间端点值的集合,满足⌈*m*/2⌉ - 1 ≤ | *keys* | ≤ *m*(*m* 为 B 树的阶),在实现时可以采用整型数组;*Nodes* 是一个集合,满足 | *keys* | = | *Nodes* | + 1,是 *BTNode* 指向下一级的所有节点;在实现时可以采用数组,*Nodes*[*i*]对应 *keys*[*i*]的左分支,*Nodes*[*i* + 1]对应 *keys*[*i*]的右分支.

● trie 树非叶节点的格式为 *TrieNode* ≡〈*left*, *right*, *patterns*, *LeafNode*〉.其中 *left* 和 *right* 都是 *Node*,分别指

向左子树和右子树,*patterns* 存储匹配的 EPC 模式集合,*LeafNode* 存储 *patterns* 位向量段形式.

(2)算法操作

在给出算法之前,首先给出几个 B 树中的基本函数,以及算法中用到的几个简单函数,由于其技术相对简单和成熟,因而就不再给出详细算法描述:

- *Search*(*bt*, *key*):在 B 树 *bt* 上查找关键字 *key*,返回结果为(*node*, *i*, *tag*).若查找成功则特征值 *tag* = 1,节点 *node* 中第 *i* 个关键字等于 *key*;否则特征值 *tag* = 0,等于 *key* 的关键字应插入在节点 *node* 中第 *i* 和第 *i* + 1 个关键字之间.
- *FindLeaf*(*root*, *key*):用于在 trie 树或扩展 B 树中找到 *key* 所对应的叶节点.对于 trie 树,按照 *key* 的二进制值来查找;对于扩展 B 树按照"小于等于"走左分支,"大于"走右分支来查找.该操作是两种查找的抽象表示,在实际中可以分别实现.
- *GetSection*(*epc*, *i*):返回 *epc* 中的第 *i* 段.
- *RemoveSection*(*epc*, *i*):删除 *epc* 中的第 *i* 段.
- *FindCommonPrefix*(*patterns*, *sec*):找到一组 *patterns* 的某个编码段 *sec* 的二进制公共前缀.
- *FindMax*(*patterns*, *sec*):找到一组 *patterns* 的某个编码段 *sec* 的最大值.
- *FindMin*(*patterns*, *sec*):找到一组 *patterns* 的某个编码段 *sec* 的最小值.
- *SearchTrie*(*tr*, *s*):根据二进制串 *s* 搜索 trie 树 *tr*,返回搜索到的 trie 树节点.
- *FindCommonAncestor*(*bt*, *node*1, *node*2):找到 B 树中节点 *node*1, *node*2 的最近公共祖先.

接下来我们给出本文中的相关算法:

- *RangePartition*(set〈*Range*〉*ranges*, *LinkList head*):用于计算多个区间按照端点值的划分.输入参数是多个区间的集合,每个区间是一个三元组〈*start*, *end*, *pattern*〉分别是区间的起始点、终点以及该区间对应的 *pattern*;返回值是一个链表,表头是 *head*,链表中的每个元素是一个四元组〈*start*, *end*, *patterns*, *next*〉,*patterns* 是区间[*start-end*]中的 *pattern* 集合,*next* 是指向下个元素的指针.
- *ComputeVectorSections*(set〈*Patterns*〉*patterns*):根据系统规则表来计算 *patterns* 所对应的向量段集合.
- *CreateExtendBTree*(*BTree bt*, *LinkList l*):构造基于区间的扩展 B 树.输入参数是利用区间端点构造的 B 树 *bt*,以及通过 *RangePartition* 函数生成的链表 *l*.
- *CreateTrie*(*Trie tr*, set〈*Pattern*〉 *patterns*, int *sec*):根据 *patterns* 的 *sec* 段的常规编码构造 trie 树.
- *CombineWithTrie*(*Trie tr*, set〈*Patterns*〉*patterns*, int *sec*):与下层常规编码段 trie 树相结合时调用.返回 *patterns* 在 *sec* 层 *trie* 树中公共前缀所对应的节点.
- *CombineWithExtendBTree*(*BTree bt*, set〈*Patterns*〉*patterns*, int *sec*):与下层扩展 *B* 树相结合时调用.计算 *patterns* 在 *sec* 段区间的最大和最小值层,并依此在 *sec* 层的扩展 B 树中找到对应的节点.
- *FindPatterns*(*Node root*, *String epc*, set〈*VectorSection*〉*result*):找到标签编码 *epc* 所满足的 *pattern* 集合.*root* 为查找结构的根节点,*result* 为查找结果,是 *pattern* 集合对应的位向量段集合.

算法 1　面向 EPC 模式的 RFID 标签编码过滤算法

```
void RangePartition(set〈Range〉 ranges, LinkList head){
    p = head;
    for each Range r in ranges do{
        if (r.start〉= p.end) p = p.next;
        else if (r.end > p.end){
            p.patterns = p.patterns⋃r.pattern; p = p.next;
        }
        else{
            Linklist ln = new LinklistNode();
            ln.end = p.end; p.end = r.end; ln.start = p.end + 1;
            ln.next = p.next; p.next = ln.next;
        }
    }
}
Map〈int secID, int vec〉
ComputeVectorSections(set〈Pattern〉 patterns){
    for each pattern p in pattern list do {
        find position i of p in pattern list
        //m is the length of vector section
        secID = ⌊i/m⌋; offset = i%m;
        v = result.find(secID);
        if(v) v = v|pow(2, offset);
        else{
            v = pow(2, offset);
            // result is the type of returning
            result.add(secID, v);
        }
    }
    return result;
}
void CreateExtendBTree(BTree bt, LinkList l){
    for each node n of l from the head to the tail do{
        leaf = new LeafNode(ComputeVectorSections(n.patterns));
        (node1, i, tag) = Search(bt, n.start);
        (node2, j, tag) = Search(bt, n.end);
        if (node2 is ancestor of node1) node1.nodes[i] = leaf;
        else node2.nodes[j] = leaf;
    }
}
void CreateTrie(Trie tr, set〈Pattern〉 patterns, int sec){
    if (tr == null) tr = new TrieNode; TrieNode p = tr;
    else{
        for each pattern in patterns do {
          for each char c in pattern.get(sec) do {
          if (c == '0'){
              if (p.left == null) {n = new TrieNode; p.left = n;}
              p = p.left;}
          else{
              if(p.right == null){n = new TrieNode; p.right =
n;}
              p = p.right;}
```

```
            }
            p.patterns = p.patterns∪{pattern}; p = tr;
        }
    for each TrieNode n in Trie tr do
        if (n.patterns! = null)
            n.LeafNode = new LeafNode(ComputeVectorSections(n.
patterns));
        }
}
Node CombineWithTrie(Trie tr,set〈Pattern〉 patterns,int sec){
    String s = FindCommonPrifix(patterns,sec);
    Node node = SearchTrie(tr,s)
    return node;
}
Node CombineWithExtendBTree (Btree bt,set〈Pattern〉 patterns,int sec)
{
    int max = FindMax(patterns,sec);
    int min = FindMin(patterns,sec);
    (node1,i,tag) = Search(bt,max);
    (node2,j,tag) = Search(bt,min);
    BTNode node = FindCommonAncestor(bt,node1,node2);
    return node;
}
void FindPatterns(Node root,String epc,Map〈int SecID,int Vec〉result)
{
    LeafNode leaf = FindLeaf(root,GetSection(epc,1));
    epc = RemoveSection(epc,1);
    if (leaf) {
    V = {v| ∀ v1∈ leaf.VectorSections ∀ v2∈ result,v1.SecID = = v2.
SecID,
        v.SecID = v1.SecID,v.Vec = v1.Vec&v2.Vec};
        if (result) {
            FindPatterns(leaf.p1,epc,V);FindPatterns(leaf.p2,
epc,V);
            FindPatterns(leaf.p3,epc,V);
        }
    }
}
```

4.7 性能分析

对每一层次的查找结构的性能分析如下:在 trie 树中的查找性能只与对应编码段的二进制长度有关,因而我们可以将其看作一个常量,在扩展 B 树中的查找性能与树中 *key* 的数量有关,对于 *pattern* 的某个层次来说,假设其区间有 n 个,最坏的情况是这 n 个区间互相重叠,那么区间分割点共有 $4(n-1)+2$ 个,因而扩展 B 树的查找性能为 $\log[4(n-1)+2]$.

查找过程中另外一部分时间消耗是在不同层次查找结果的结合上,假设查找结构中每个叶节点的向量段集合均有 m 个向量段,那么上下层相结合时至多比较 m 次即可找到查找结果,m 的数量不可能确切得出,但可以肯定的是,由于在 EPC 模式表中将编码空间距离较近的 EPC 模式集中的放在一起,所以 m 的数量将会远小于规则的总数,因而这部分时间消耗也会很小.

4.8 实验

在 RFID 中间件系统 PKU Edge Server 中实现了面向 EPC 模式的标签编码过滤算法,通过自动生成一定数量的 EPC 模式和大量的标签编码来进行模拟实验,并和开源 RFID 中间件 Singularity 进行比较来说明本文给出算法的性能.测试所用软硬件版本和配置如下:

- 计算机是 Intel Pentium IV CPU(2.0GHz/2GB);
- 操作系统采用 Windows XP Professional;
- JDK 采用 1.6.0_03 版本;
- 数据库软件采用 SQL server 2005.

实验中生成的标签编码和 EPC 模式都是基于 GID-96,其中 *Head* 和 *GeneralManagerNumber* 部分为常规编码段,*ObjectClass* 为区间形式,*SerialNumber* 为"*".标签编码是不停生成的,不会有没有标签编码需要处理的情形.通过观察不同 EPC 模式数量下 PKU Edge Server 和 Singularity 每秒处理标签编码数量以及过滤标签编码需要的平均时间来比较两者的效率.

图 8 显示了在不同 EPC 模式数量情况下,PKU Edge Server 和 Singularity 每秒钟处理标签编码数量.通过比较发现,随着 EPC 模式数量的增多,singularity 每秒钟处理标签编码的数量会显著下降,而 PKU Edge Server 的下降则很缓慢,基本维持一个稳定的状态.

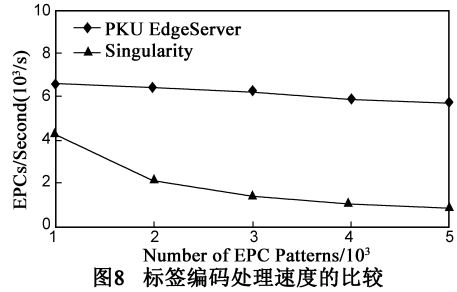


图8 标签编码处理速度的比较

图 9 显示了在不同 EPC 模式数量情况下,PKU Edge Server 和 Singularity 处理标签编码消耗的平均时间.通过比较发现随着 EPC 模式数量的增多,Singularity 处理标签编码消耗的平均时间增长较快,延时也会变得较长,而 PKU Edge Server 处理标签编码消耗的平均时间增长较缓,基本维持在一个稳定状态.

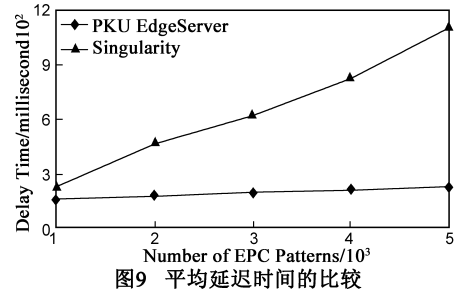


图9 平均延迟时间的比较

5 总结

本文主要研究了面向 EPC 模式的标签编码过滤方法,通过对标签编码和 EPC 模式结构的分析,给出了基于 trie 树和扩展 B 树的标签编码过滤方法,采用该方法

过滤标签编码的效率受 EPC 模式数量变化的影响较小,能够有效降低向上层应用传输数据的延迟,减少由于缓存溢出而导致的中间件崩溃现象.

RFID 中间件的发展正朝着嵌入式的方向转变,旨在通过与硬件绑定的一体化接入服务来为用户提供服务.结合本文提出的过滤方法,从嵌入式 RFID 中间件的角度来看,只要嵌入式系统能够提供存储 EPC 模式的数据库和数据库访问接口,那么本文给出的面向 EPC 模式的过滤方法就可以移植到嵌入式系统上.

参考文献

[1] EPCglobal. The Application Level Events (ALE) Specification, Version 1.0[S]. http://www.epcglobalinc.org/standards/ale/ale_1_0-standard-20050915.pdf,2005.

[2] EPCglobal. The Application Level Events (ALE) Specification, Version 1.1[S]. http://www.epcglobalinc.org/standards/ale/ale_1_1-standard-core-20080227.pdf,2008.

[3] EPCglobal. The Application Level Events (ALE) Specification,Version 1.1.1[S]. http://www.epcglobalinc.org/standards/ale/ale_1_1_1-standard-core-20090313.pdf,2009.

[4] Gupta A,Srivastava M. Developing auto-ID solutions using Sun Java system RFID software[EB/OL]. http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/sjsrfid/RFID.html,2005-11-21.

[5] Shawn R. Jeffery, Minos Garofalakis, Michael J. Franklin. Adaptive cleaning for RFID data streams[A]. Proceedings of the 32th VLDB Conference on very Large Data Bases[C]. Seoul Korea:Morgan Kaufmann Publishers,2006.163 – 174.

[6] Arvind Arasu, Shivnath Babu, Jennifer Widom. The CQL continuous query language:Semantic foundations and query execution[A]. Proceedings of the 32th VLDB Conference on very Large Data Bases[C]. Seoul Korea: Morgan Kaufmann Publishers,2006.121 – 142.

[7] Li Yang, Rushi Vyas, Manos M. Tentzeris. RFID tag and RF structures on a paper substrate using inkjet-printing technology [J]. IEEE Transactions on Microwave Theory and Tcehniques, 2007,55(12):2894 – 2901.

[8] Jiang Shao-gang, Tan Ji. Research of data processing and filtration in RFID middleware[J]. Computer Application, 2008, 28 (10):2613 – 2615.

[9] Fagui Liu, Yuzhu Jie, Wei Hu. Distributed ALE in RFID middleware[A]. Proceedings of 4th IEEE International Conference on Wireless Communications, Networking and Mobile Computing(WiCOM'08) [C]. Dailan China,2008.1 – 5.

[10] Q Dong, A Shukla, V Shrivastava, D Agrawal, S Banerjee, K Kar. Load Balancing in Large-scale Rfid Systems[R]. UW-CS Technical Report 1568 (www.cs.wisc.edu/techreports/2006/TR1568.pdf),2006.

[11] Feng Bo, Li Jin-tao, Zhang Ping, Guo Jun-bo, Ding Zhen-Hua. Study of RFID middleware for distributed large-scale systems[J]. Information and Communication Technologies, ICTTA,2006(2):24 – 28.

[12] EPCglobal. EPCglobal Tag Data Standards Version 1.4[S]. http://www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf,2008.

作者简介

赵　文　男,1967 年出生,博士,副研究员,主要研究领域为软件工程、工作流技术和 RFID 相关技术.

E-mail:zhaowen@pku.edu.cn

刘学洋　男,1978 年出生,博士,讲师,主要研究领域为软件工程、信息安全和 RFID 相关技术.

E-mail:liuxy@sei.pku.edu.cn