H-Tree：一种面向大数据流在线监测的层次索引

臧文羽[1),2)] 李 军[1),3)] 方滨兴[1),2),3)] 谭建龙[2)]

[1)](中国科学院计算技术研究所信息安全研究中心 北京 100190)

[2)](中国科学院信息工程研究所信息内容安全技术国家工程实验室 北京 100093)

[3)](北京邮电大学计算机学院 北京 100876)

摘 要 随着计算机网络的迅猛发展和大数据时代的到来，数据越来越频繁地呈现出多属性异构的特点.这种包含多种不同类型属性的大数据流称为异构大数据流(Heterogeneous Big Data Streams).在面向大规模数据在线监测分析的应用中，通常需要在异构大数据流上注册大规模监测规则.因此，对于每一个数据流元组，必须用最小的计算开销满足所有的规则.同时，由于大数据流上监测规则集异常庞大，提高规则监测的性能是大规模数据流在线监测的关键.基于此，该文提出一种层次化的索引结构 *H*-Tree 及其在线规则匹配算法.具体的，*H*-Tree 将大数据流上的属性集划分为离散型属性和连续型属性.基于不同的属性集，构建两层索引结构：在第1层，通过改进的红黑树对离散型谓词构建触发索引；在第2层，通过量化连续型谓词构建多维索引结构.*H*-Tree 的在线规则匹配算法利用关联关系表对两层索引的监测结果进行融合过滤.实验分析表明，与经典的 R^+ 方法相比较，*H*-Tree 通过层次化的索引结构，在不降低准确度的前提下，显著提升了大数据流的监测效率.

关键词 异构大数据流；监测规则；索引

中图法分类号 TP309 **DOI 号** 10.3724/SP.J.1016.2015.00035

H-Tree: Hierarchy Index for Online Monitoring of Big Data Streams

ZANG Wen-Yu[1),2)] LI Jun[1),3)] FANG Bin-Xing[1),2),3)] TAN Jian-Long[2)]

[1)](*Research Center of Information Security, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*National Engineering Laboratory for Information Content Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093)

[3)](*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing* 100876)

Abstract With the evolution of computer network and the coming of big data era, data stream presents new characters of multiple attributes and heterogeneous. We name the streams that contain multiple attributes Heterogeneous Big Data Streams. In the big date stream oriented online monitoring systems, there usually exist many filtering queries that specify the filtering objectives. Thus, for every incoming tuple, it should satisfied all queries with the least computational cost. Meanwhile due to the huge amount of filtering queries on big data streams, a key problem in such a filtering scenario is how to index the query set to make the detection of heterogeneous big data streams more efficiency. Based on this problem, we propose a hierarchal index framework (*H*-Tree) and its online matching algorithms. Specifically, *H*-Tree clusters the attributes into discrete attributes and continuous attributes, which executes a two-stage indexing strategy. In the first stage, *H*-Tree builds indexes according to the discrete predicates. In the second

收稿日期：2013-06-02；最终修改稿收到日期：2014-08-26.本课题得到国家自然科学基金(61370025)、国家“八六三”高技术研究发展计划项目基金(2011AA010703，2012AA012502)、国家“九七三”重点基础研究发展规划项目基金(2013CB329606)及中国科学院战略性先导科技专项课题(XDA06030200)资助.**臧文羽**，女，1988年生，博士研究生，主要研究方向为数据挖掘、流计算、网络信息安全.E-mail：zangwenyu@software.ict.ac.cn. **李 军**，男，1983年生，博士，工程师，主要研究方向为网络信息安全、数据库、数据挖掘. **方滨兴**，男，1960年生，博士，教授，博士生导师，中国工程院院士，主要研究领域为网络安全、信息内容安全. **谭建龙**，男，1974年生，博士，研究员，博士生导师，主要研究领域为信息安全、相似性计算、文本检索.

stage, H-Tree build multi-dimension index based on the continuous predicates. Experiments demonstrate that H-Tree greatly improve efficiency on big data streams monitoring without losing accuracy compared with R^+ solution.

Keywords heterogeneous big data streams; filtering rules; index

1 引 言

伴随着大数据时代的到来,企业信息化水平不断提升,企业信息运营系统数据流的种类和规模也持续增长.利用大数据技术实现大型集团企业的运营状态在线监测具有广泛的应用前景.对于大数据环境下,包含不同媒体格式的数据流,通过传统的数据流监测[1-2]方法(对监测规则进行线性计算)已经不能满足在线监测的要求.面对运营大数据在线监测需求,通过对监测规则构建索引,可以提升规则计算的判别速度,大大提高异构大数据流在线监测的效率.

将这种包含不同类型属性且具有高维度的大数据流称为异构数据流.异构数据流包含文本、图片、音视频等不同属性的元信息,属性间存在较大的差异.近来,随着基于内容的大规模多媒体大数据流监测需求的不断增加,大数据流实时监测已经成为研究热点.然而在实际的监测过程中,由于监测规则集规模大、关联维度高,使得大规模数据流上的监测计算量大,效率低下.因此,如何高效地匹配数据流元组与大规模监测规则成为提高异构大数据流监测效率的关键.对监测规则构建索引是解决问题的有效途径,相关研究成果包括:针对单一属性的数据流,CEI-Index[3]使用区域划分的思路,以空间换时间提高谓词匹配效率;针对离散型属性简单且数量较少的发布订阅系统中的数据流,一般通过哈希表以及Cache技术[4]提升离散型属性集上的匹配效率;针对单纯的连续型属性,R-Tree[5]将多个连续型属性映射到高维向量空间并进行合理的划分,为每个划分建立覆盖区域.R^*-Tree[6]通过优化 R 树的空间划分策略来减少节点的访问次数,从而提高了监测效率.

上述研究大多是针对单一类型的数据流进行优化索引,而且规则集的数量也相对较少,因而很难直接应用到异构大数据流监测中.具体来说,面对异构大数据流的在线监测应用,传统[5-7]的索引技术都是假定数据流的所有维度都具有相同或者相似的性质;传统的索引方法没有充分利用异构数据流中不同属性之间的关系,需要做很多属性维度的转换,直接影响了监测性能.然而,异构数据流包含的属性之间尽管存在较大的差别,但不同属性之间也存在共享度较高的关联性.如果能够合理的将不同属性的谓词集结合起来,可以有效提升监测的性能.

由此,结合面临的挑战以及异构大数据流的自身特点,本文从异构大数据流监测的应用背景出发,针对传统高维索引对异构大数据流处理的性能瓶颈问题,提出了一种层次化的索引结构 H-Tree.本文的主要贡献包括:

(1)在高维索引的基础上,引入了层次化索引的思想.考虑异构大数据流本身的特点,将数据流包含的属性划分为离散型属性和连续型属性.考虑到离散型属性上的谓词共享度高的特点,提出了一种两层的索引结构 H-Tree,给出了索引构建和监测匹配算法.

(2)对提出的 H-Tree 索引构建和监测匹配算法进行详细的算法复杂度分析.

(3)在大量公开数据集上,对 H-Tree 进行了大量的测试实验,结果表明:H-Tree 结构的匹配算法在不降低监测准确率的情况下,较 R-Tree 的性能有了大幅度的提升.

本文第 2 节形式化定义异构大数据流监测问题并提出简单的解决算法;第 3 节详细介绍 H-Tree 的索引框架和核心算法;第 4 节给出 H-Tree 的评估模型;第 5 节是实验结果的对比与分析;第 6 节是对论文工作的一个总结.

2 问题描述

对异构数据流进行内容监测时,首先在数据流上注册大量的监测规则.然后将数据流中的每一个实时到达的元组与注册的监测规则逐一匹配.这里,监测规则和数据流元组的定义如下:

定义 1. 监测规则.令 Q 表示注册的监测规则集合.S 代表一个异构数据流.每个监测规则 $q\in Q$ 是多个谓词的"与".每个谓词关联着相应的属性,这些属性主要分为两类:连续型属性和离散型属性,相

应地，监测规则 q 包含的谓词分为：连续型谓词（$v_i<p<v_j$）和离散型谓词（$p==v_k$）.

若任意谓词的结果是布尔型变量. 则 q 可表示为

$$q=\prod_{p_i\in q}p_i\begin{cases}p_i=1, & 真\\ p_i=0, & 假\end{cases} \tag{1}$$

定义 2.　数据流元组. 令 t 表示当前的数据流项，元组 t 是由不同属性上的键值对组成的集合. $t=(k_i,v_i)$，其中 k_i 对应着第 i 个属性. 若连续型属性 i 对应的值域为 $[s_i,t_i]$，离散型属性 j 对应的值域为 $S=\bigcup v_j$，则元组 t 对应监测规则 q 的计算可表示为

$$q(t)=\bigcup_i(s_i,t_t)=\prod_i\begin{cases}st_p\le v_i\le ed_p, & 连续\\ v_i\in S_p, & 离散\end{cases} \tag{2}$$

因此，在判断监测规则的过程中，如果谓词 p_i 对于当前数据流项 t 的结果为真，就认为数据流项 t 满足属性谓词 p_i；如果数据流项 t 满足一条监测规则 q 中的所有谓词，就说数据流项 t 满足监测规则 q. 对于异构数据流项 t，监测计算具有如下 2 个性质：

（1）如果 t 满足监测规则 q，那么监测规则 q 中所有的谓词都会出现在已经计算为“真”的序列中. 也就是说，q 中所有的属性谓词都必须经过计算且返回值为真.

（2）如果 t 不满足监测规则 q，那么 q 中至少存在一个属性谓词在已经完成的计算序列中，且计算结果为假.

由以上 2 个性质可以看出，对数据流元组监测的过程就是元组在属性域包含的谓词集上的检索过程. 因此，如何提升检索效率是监测算法的关键. 如表 1 所示，数据流 S 上注册的监测规则集 Q 包括 5 个监测规则，每个监测规则的含义如表所示. 其中每个监测规则 $q\in Q$ 都是不同谓词的“与”运算. 将连续型属性 A_{time}，A_{len} 映射为两维空间，那么 q_1，q_2，q_3，q_4，q_5 的关系如图 1 所示. 不同的监测规则之间的共享，对应着图 1 中两维空间上的矩形块之间的交叠.

表 1　异构数据流 S 上注册的监测规则

监测规则 ID	规则表达式
q_1	(20101220<*t.time*<20101225) ∧ (512<*t.len*<1024) ∧ (*t.ip*=“59. 64. 135. 205”)
q_2	(20101220<*t.time*<20101225) ∧ (0<*t.len*<256) ∧ (*t.ip*=“121. 76. 53. 84”)
q_3	(20110101<*t.time*<20110201) ∧ (512<*t.len*<1024) ∧ (*t.ip*=“155. 134. 121. 78”)
q_4	(20110101<*t.time*<20110225) ∧ (512<*t.len*<1024) ∧ (*t.ip*=“78. 65. 202. 123”)
q_5	(20110201<*t.time*<20110229) ∧ (512<*t.len*<1024) ∧ (*t.ip*=“202. 104. 123. 144”)

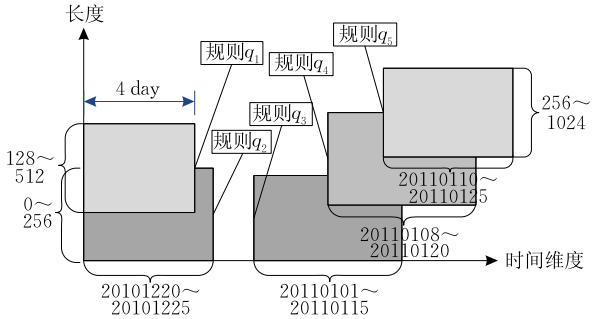


图 1　表 1 中 5 个监测规则在对应的连续型高维空间中的关系

2.1　经典的解决方法

针对异构数据流下的内容监测问题，本文首先来介绍两个简单的解决办法.

（1）监测规则顺序检查算法（Query Sequence Inspect）：这种算法[8-10]广泛应用于面向数据流的挖掘中，其核心思想是针对监测规则不建立索引，根据训练好的分类器，对于每一个数据流元组 t，依次取得其相应属性的键值对 (k,v)，然后将 (k,v) 键值对依次与分类器中的规则进行比较，将所有谓词结果都为“真”的监测规则作为结果输出. 这种算法效率低，性能差：基于集成分类器[8]提出的数据流监测算法在规则少于 1000 的情况下，元组的监测时间接近 1s，这远远达不到我们的大规模内容监测的应用要求.

（2）统一高维索引（Uniform high-Dimension Index）：这种方法[5]广泛应用于空间数据库监测规则相关的应用系统中，核心思想是不对谓词关联的属性本身的性质进行区分，将所有的属性依次对应为空间对象的维度，每一个监测规则对应于属性空间中的一个区域块，然后根据所有的监测规则构建高维索引，该算法可以把每一个数据流元组看作高维空间中的一个点，因此，数据流元组的监测过程就变成了高维索引中的点监测规则. 这种算法与 *H*-Tree 最大的不同之处在于它不对属性集合进行分类，而是统一的映射到多维度空间构建高维索引，当异构数据流的属性数量在 80 以上，监测规则大于 2000 时，性能下降明显[11].

数据流监测的目标是：针对每一个数据流元组，用最小的开销计算出所有满足的监测规则. 本文的索引结构针对异构数据流实时监测的特点，充分利用属性间的关联性，将规则集按照属性分类构建层次化的索引结构，最大限度地利用不同属性索引的特点，大大的提升了实时监测效率.

3　*H*-Tree 的结构和算法

本节重点介绍 *H*-Tree 的索引结构及核心算

法. *H*-Tree 是一种面向异构数据流上大规模监测规则计算的动态索引,支持监测规则的实时更新,其主体流程包括索引构建和实时匹配计算. 如图 2 所示,构建索引时,首先对异构数据流的属性集进行分类:连续型属性和离散型属性. 然后,根据属性类型将输入的监测规则集划分为不同的谓词集,基于不同的谓词集构建层次化的索引结构:根据离散型属性上的谓词构建 Treap 树作为第 1 层索引;在第 2 层,将所有连续型属性映射为多维空间,根据连续性属性相关的谓词构建高维索引. 由于离散型属性上的谓词都是离散值,所以构建的第 1 层索引可以快速定位到监测规则上,而且空间开销也比较小. 在 *H*-Tree 的第 2 层,本文将性质相同的属性谓词按照维度划分构建索引,尽可能的提升监测规则处理速度. 对实时到达的数据流元组 t 进行监测计算时,首先对 t 进行向量抽取和裁剪计算;量化后的向量经过谓词属性分类处理后得到不同的谓词向量;运用 *H*-Tree 的监测规则算法通过两层索引结构过滤得到满足条件的监测规则集. 在本节余下的部分中,首先在 3.1 节给出 *H*-Tree 索引框架的基本结构;在 3.2 节,给出 *H*-Tree 的动态构建算法;在 3.3 节,描述 *H*-Tree 的实时监测流程.

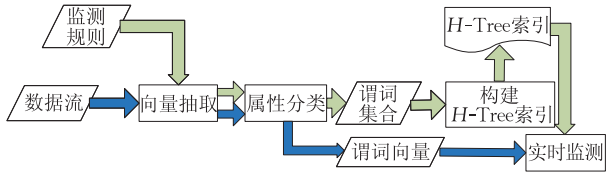


图 2 数据流内容监测的总体框图示意

3.1 *H*-Tree 的索引结构

从结构上来讲,*H*-Tree 包括 3 个重要组成部分以及 3 个重要操作. 其中,3 个组成部分分别是:(1)第 1 层的 Treap 索引结构;(2)第 2 层的高维向量索引结构;(3)监测规则和谓词的关联表. 基于 *H*-Tree 的 3 个主要操作分别是:(1)搜索;(2)插入;(3)删除.

H-Tree 总体上是 1 个两层的索引结构. 图 3 给出了基于表 1 中的监测规则构建的 *H*-Tree. 如图 3 所示,其第 1 层是由离散型谓词构建的 Treap 树索引;第 2 层是根据连续型谓词对应的多维向量构建的高维空间树;另外 1 个很重要的组成部分是监测规则与谓词的关联表,用来完成两层索引监测规则结果的快速融合.

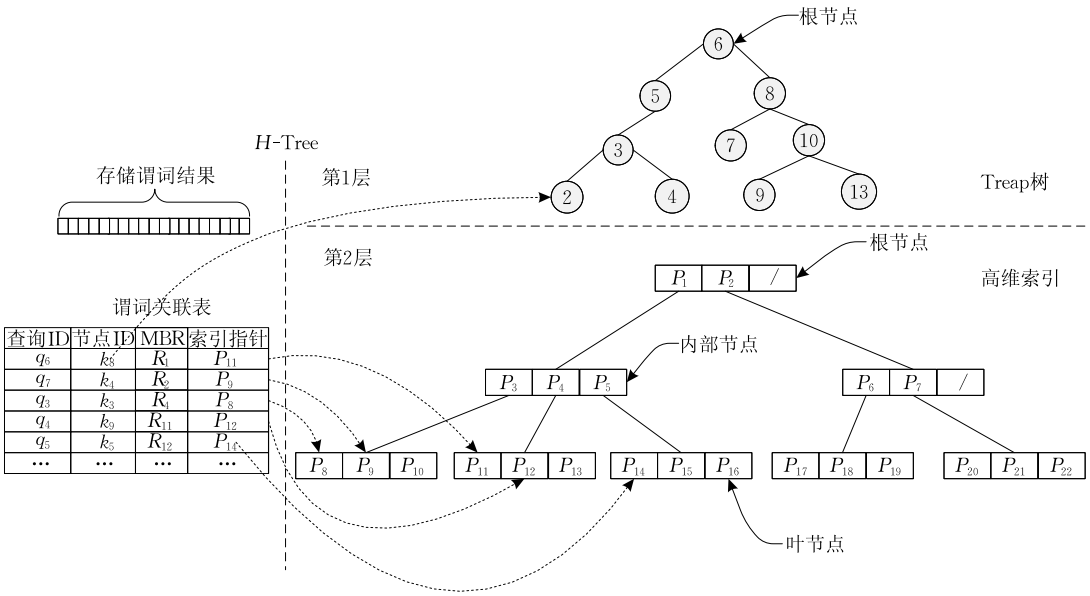


图 3 *H*-Tree 树结构(主要有 3 个部分组成. 左边是 *H*-Tree 中存储监测规则与谓词的关联表. 右边是 *H*-Tree 的两层索引框架. 第 1 层是 Treap 树,是根据离散型属性上的谓词集构建. 第 2 层是高维索引树,是根据连续型属性的谓词构建)

H-Tree 中的节点可以分为 3 类:首层节点 top_node,第 2 层的中间节点 mid_node 和叶子节点 leaf_node. 节点的主要定义和主要包含元素如图 4 所示. 其中,attr 为首层 Treap 树节点对应的离散型属性,value 为该 Treap 节点对应的离散值,weight 为该节点代表的谓词的优先级,left,right 为该节点的左

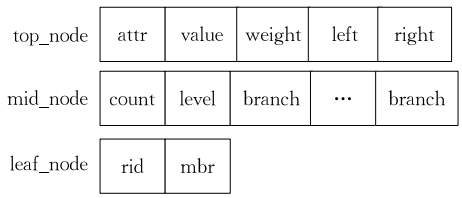


图 4 *H*-Tree 中 3 种节点类型

右孩子节点. branch 代表着第 2 层索引对应的高维树结构的中间节点指针. mbr 为第 2 层叶子节点对应的高维向量.

3.2 *H*-Tree 的动态构建

基于 *H*-Tree 的层次化索引结构，提出一种可行的索引构建算法，共分为 3 个步骤：规则预处理、谓词集合划分和层次化索引构建. 先对预处理后的规则集按照属性分类进行一定的划分，在此基础上再对划分后的数据集分层构建索引.

规则集划分时，对于预处理后的 n 条规则，按照属性类别和值域划分了离散型谓词集 $\boldsymbol{A}$ 和连续型谓词集 $\boldsymbol{B}$. 其中 $\sum \|A\|+\|B\|=\sum_{q\in Q} p(q)$. 假设将第 1 维上的谓词集分为了 s 个区间 $I_1, I_2, \cdots, I_s$，其中每个区间只有离散型谓词或者连续型谓词. 由于经过了预处理的维度转换，使得 $I_1, I_2, \cdots, I_s$ 的区间内含有的谓词性质基本相同或者相似，便于分层次索引的构建. 同时，异构数据流注册的规则集通常包含着优先级信息，规则的优先级越高，表示其被监测计算的需求越紧迫. 当一个新的监测规则注册到系统中，*H*-Tree 首先预处理模块将监测规则按照属性类型划分为离散型谓词 p_d 和连续型谓词 p_c. 然后将离散型谓词 p_d 插入到 *H*-Tree 的第 1 层索引中，也就是插入到离散型属性对应的 Treap 树索引中；最后将连续型谓词 p_c 插入到 *H*-Tree 的第 2 层索引中.

当将 p_d 插入到第 1 层的 Treap 树的时候，首先按照排序二叉树的标准插入法进行插入，但这时可能违反 Treap 树的堆性质，因此需要自底向上进行旋转，直到堆性质得到满足. 删除是相反的，先把优先级设置为最低，自上而下转移到叶子，然后删除. *H*-Tree 的第 2 层插入算法与 *R*-Tree 的插入过程类似：首先利用 $searchLeaf(p_c)$ 去定位到要插入的目标叶子节点，定位的过程本身是个递归过程. 第 2 层索引插入 p_c 的过程从第 2 层的根节点开始，顺次按照广度优先搜索，按照多维空间的包含关系进行搜索，当找到一个叶子节点 n 以后，检查 n 的分支数. 如果发现已经超过分支阈值 M，则直接进行节点分裂，产生新节点，并将 n 已有的节点和 P_c 的向量利用启发式策略平均分派到两个节点中，最后依次更新父亲节点信息. 如果 n 的分支数没有超过阈值 M，则直接调用 $update\ ParentNode(L)$ 来完成插入操作. *H*-Tree 插入和删除的具体细节如算法 1 和算法 2 所示.

算法 1.　*H*-Tree 的插入算法.

输入：监测规则 q，离散型谓词 p_d，连续型谓 p_c，*H*-Tree T，Insert node n

输出：NULL

```
1. IF n==NULL THEN
2.   DirectInsert(p_d);
3.   rt←GetRoot(T).
     L←searchLeaf(p_c;T)
4.   IF left branch of L less than M THEN
5.     Update ParentNode of L
6.   ELSE
7.     Split Node L;
8.     Adjust child tree of L;
9.   END IF
10. ELSE
11.   IF the value of p_d is less than n: value THEN
12.     Insert (n: leftChild, p);
13.   ELSE
14.     Insert (n: rightChild, p);
15.   END IF
16. END IF;
```

算法 2.　*H*-Tree 的实时监测算法.

输入：数据流元组 t，索引 H，检索结果 res

输出：NULL

```
1. FOR all 针对当前滑动窗口中的元组 t DO
2.   r←H_first: TreapSearch(t).
3.   IF r is not empty THEN
4.     H_second: Search(t; res);
5.   ELSE
6.     t←Get Next Tuple t;
7.   END IF
8.   IF res is not empty THEN
9.     res←Combine(res; r)
10.  ELSE
11.    t←Get Next Tuple t;
12.  END IF
13. END FOR;
```

需要注意的是：基于离散型谓词的特点，*H*-Tree 的第 1 层索引用 Treap 树构建，加快离散型谓词的快速监测规则计算. 我们假设空树的优先级为无穷大，则上面的添加和删除算法可以正确处理只有一个儿子的情形. 第 1 层索引的添加和删除操作的期望时间复杂度均为 $O(\log n)$. 从编程复杂度来说，Treap 是工程上最容易实现的数据结构，不仅没有任何特例需要考虑，而且在期望 $O(\log n)$ 时间内同时支持插入、删除、分离和合并.

插入过程最重要的问题是第 2 层索引中节点的分裂策略. 在这里本文采用类似 *R*-Tree 的启发式策略. 详细来讲，首先取出所有要分裂的块. 然后选择

交叠面积最小、同时覆盖两个块的面积最大的两个块.最后将剩余的块按照面积交叠差异依次划归到不同的节点中.由于删除操作与构建操作都比较简单,在此,本文不再赘述.

3.3 实时监测算法

在这一节重点介绍数据流元组在 H-Tree 的监测过程如图 5 所示.针对元组 t,H-Tree 充分挖掘面向大数据流监测上规则集的特征,首先扫描第 1 层的 Treap 树索引来计算满足的离散谓词集.由于离散型谓词共享度很高,这就大大提升了扫描速度,加快了监测规则的整个处理过程.然后,根据第 1 层索引命中的有效节点,搜索谓词关联表,找到第 2 层索引的节点指针,进入第 2 层高维索引中继续查找,最后,融合第 1 层和第 2 层命中的结果,将最终命中的规则集进行快速融合计算.算法 2 给出了数据流元组的监测规则处理的具体细节.

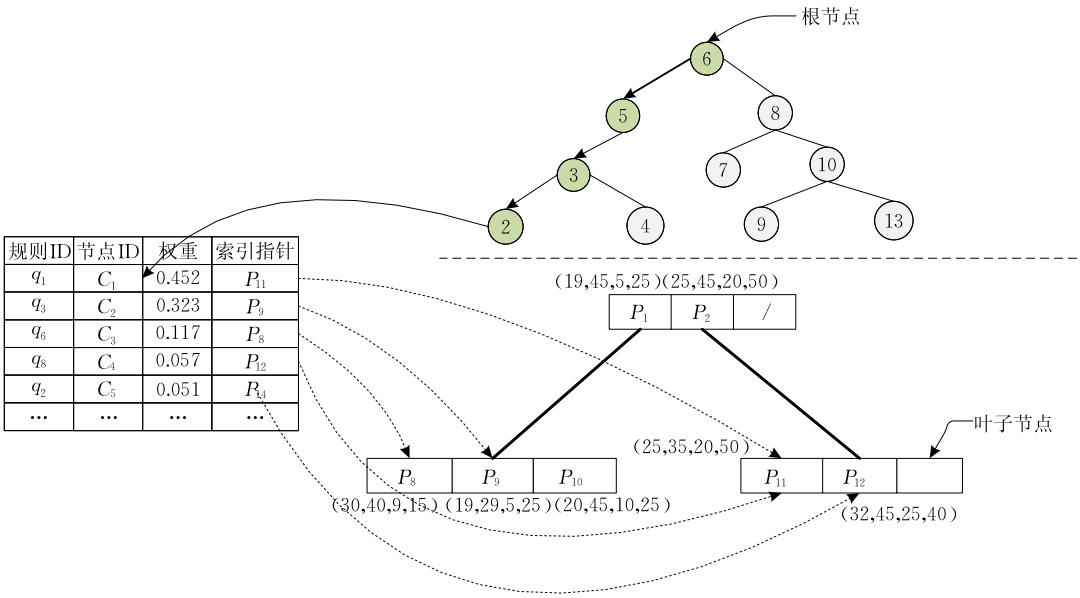


图 5 H-Tree 的实时监测过程

在算法 2 中,对于任意的数据流元组 t,首先把离散型属性值在多属性 Treap 树索引中扫描,如果命中,表示有离散型谓词命中,这种情况下才进行第 2 层索引的监测规则过程.否则直接返回.通过构建第 1 层触发索引,大大加快了 H-Tree 数据流元组的监测规则过程.如果命中了离散型谓词,则接下来进行第 2 层索引的并行监测规则过程.即对于第 1 层中的每一个高维索引依次进行属性值键值对的监测规则,并将监测规则结果存入 $vRes$ 中.本文根据获得的 $vRes$ 运行结果融合算法,最终获得命中的监测规则结果集 Q_r.

4 算法复杂度分析

本节针对提出的 H-Tree 中的实时监测算法和插入、删除算法的时间和空间开销进行分析.在这里,监测规则集合本文用 Q 来表示,监测规则和谓词的映射结构用 C 表示,H 用来表示多属性哈希索引.

如算法 2 所示,对于每个实时到来的数据流元组 t,监测规则处理的时间开销可以分为 4 个部分:(1)计算每个谓词结果的时间;(2)检查监测规则和谓词映射关系的时间;(3)检索哈希索引的时间;(4)检索第 2 层高维索引树的时间.为此,本文设计了评估方程,对监测规则处理过程的时间开销进行评估.因此,时间开销的评估方程如下:

$$idx\_searching(H)+sum=\sum_{p\in TP(C)} v(p)\times scanning(p,Q)+multi\_dim(t) \quad (3)$$

其中,$idx\_searching(H)$ 是多属性 Treap 树的搜索开销,$scanning(p,Q)$ 代表着对谓词 p 对应的关联表计算的开销,$multi\_dim(t)$ 代表着 t 在 H-Tree 的第 2 层连续高维索引的检索开销.在算法 2 中,本文做了以下假设:(1)多属性 Treap 树索引的搜索开销 $idx\_searching(H)$ 与离散型属性个数成正比;(2)$scanning(p,Q)$ 的大小与 Q 的数量成正比.基于以上假设,式(3)可以转换为式(4).

$$sum=|H|\times\log C_{idx}+\sum_{p\in TP(C)} v(p)\times\lambda_p\times\kappa_p\times n \quad (4)$$

C_{idx} 代表多属性 Treap 树的节点数量.λ_p 代表着谓词 P 在监测规则集 Q 的频率,κ_p 是谓词 p 的宽度.n 表示监测规则集 Q 的数量.同时,我们用式(5)对

H-Tree 的空间开销进行评估，空间开销主要由以下 3 个部分组成：(1) 第 1 层索引即多属性 Treap 索引的空间；(2) 存储谓词与监测规则关系的空间开销；(3) 连续型高维索引的开销.

$$sum=\sum_{h\in H}M_{\text{treap}}(h)+\sum_{q\in Q}M_{\text{bitmap}}(q)+\sum_{p\in TP(H)}M_{\text{mapping}}(Q)+M_{\text{dim}} \quad (5)$$

其中，M_{treap} 构建多属性 Treap 树索引所需要的空间，M_{bitmap} 是被用来对所有的监测规则结果进行融合计算的空间. M_{mapping} 是用来对所有的谓词与监测规则集 Q 的关系进行缓存的空间. M_{dim} 是 *H*-Tree 的第 2 层高维索引的内存开销.

5　实验结果

为了验证 *H*-Tree 的性能，本文基于公开数据集和人工数据集进行了属性数量变化、谓词数量变化、规则数量变化下的性能比较实验. 在实验中，既实现了提出的层次化索引框架 *H*-Tree，又实现了算法 QSI 和算法 UDI，实验结果表明，无论在何种参数设置情况下，由于采用了层次化的索引结构，*H*-Tree 的性能都远远优于 QSI 算法和 UDI 算法的. 在这一部分，本文首先描述对结果的评估办法，然后再具体阐述实验结果.

5.1　结果评估办法

在实验的监测规则部分，用表 2 所示的 3 种人工数据集分别进行实验. 所有的谓词与监测规则之间的关系服从 Zipf 分布，Zipf 分布广泛使用在基于内容的关键字检索中. 测试数据流是从网关上捕获的实时多媒体数据流，然后按照已经选定的属性和维度进行内容抽取，将抽取的内容以异构数据流的形式推送到监测算法中. 实验部署在处理器为 Duo 2.66GHz CPU，内存为 2GB，Linux AS5 的机器上. 其中，实验最关心的是监测规则计算的时间开销. 这个时间开销定义为：对于任意一个数据流元组，完成所有监测规则的计算所花费的时间均值. 本文将 *H*-Tree，QSI，UDI 这 3 种算法的表现做对比. 为了使比较更有意义、更公平，本文在每组实验中仅仅改变一个参数，其他参数均设置为固定值. 所有实验结果都是连续重复运行 20 次后求得平均值.

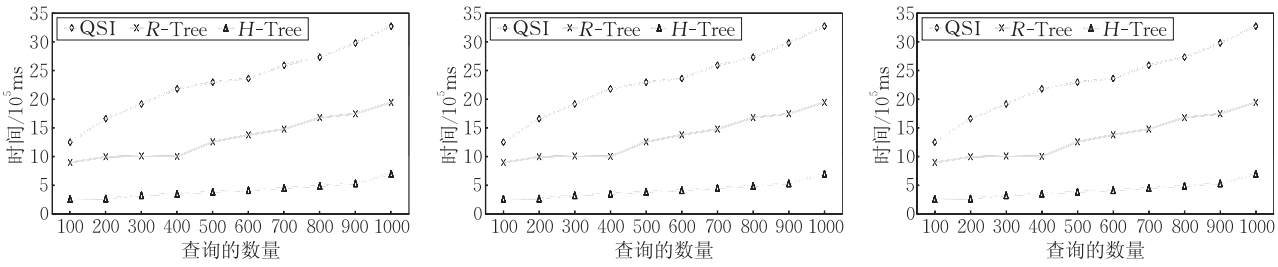
表 2　实验数据集

名称	连续型属性集	监测规则集	离散性属性集
stock data analysis	40	10000000	17
spam detection	70	10000000	21
Intrusion detection	80	10000000	20

5.2　实验结果

5.2.1　监测规则规模的影响

首先，我们最关心的是监测规则规模不同的情况下 3 个算法的表现. 其中，本文用监测规则的数量这个指标来衡量监测规则的规模. 依据该指标本文进行实验：将监测规则的数量从 100 增加到 1000. 图 6 显示了实验的结果. 从实验结果中可以看到无论在何种参数情况下，*H*-Tree 都远远优于 QSI 以及 UDI. 另外 *H*-Tree 比 QSI 优秀的一个重要原因是本文通过多属性触发索引的扫描淘汰了大量的元组，很多数据流元组假如经过多属性触发索引的扫描过程以后发现不命中，那么元组中剩余属性上的键值对就根本不需要进行扫描.



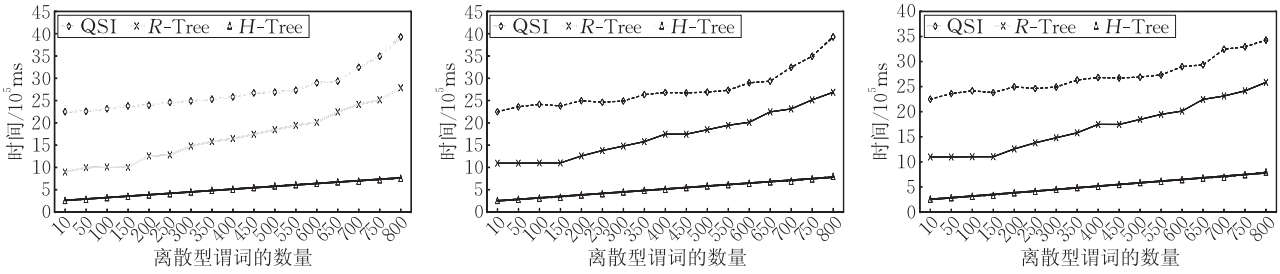
(a) Stock data analysis数据集　(b) Intrusion detection数据集　(c) Spam detection数据集

图 6　本文将监测规则数量从 100 增加到 1000 的过程中 3 种算法的表现(从图中可以明显看出 3 个算法的时间开销都明显增加，但是 *H*-Tree 的增加幅度明显比其他两个算法的要小得多)

5.2.2　离散型谓词规模的影响

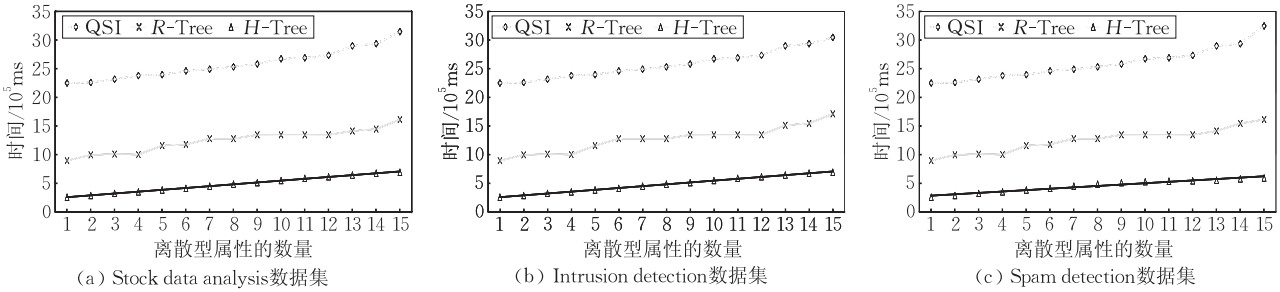
图 7 和图 8 显示在离散型谓词不同的情况下 3 个算法的表现. 本文用离散型属性的数量和离散型谓词的数量这两个指标来衡量离散型谓词对监测规则性能的影响. 依据这两个指标，本文在监测规则的数量固定在 10000 条，连续性谓词的数量固定在 120 个的情况下进行了两组实验，第 1 组是在将离散型属性的数量固定在 5 的情况下进行，本文将离散型谓词的数量从 10 渐增到 800. 第 2 组实验是在将离散型谓词的数量固定在 100 的情况下展开，本

文依次将离散型属性的数量从 1 增加到 10. 从图 7 和图 8 的两组结果中可以看到无论在何种参数情况下，*H*-Tree 都远远优于另外两种算法. 而且，在随着离散型属性数量的增加，*H*-Tree的表现更加优秀，主要是因为本文利用了并行性的处理算法，将监测规则更好地分布到不同的触发索引上. 另外，本文发现，在离散型属性不变的情况下，增加离散型谓词的数量对 *H*-Tree 几乎没有影响，是因为触发索引的检索速度基本不受谓词数量的影响，这与本文最初的设想基本相同.



(a) Stock data analysis数据集 (b) Intrusion detection数据集 (c) Spam detection数据集

图 7 本文将离散型谓词从 10 增加到 800 的过程中 3 种算法的表现(从图中可以明显看出由于 *H*-Tree 采用了层次化的索引，利用第 1 层的触发索引可以保证基本不受离散型属性数量的影响)

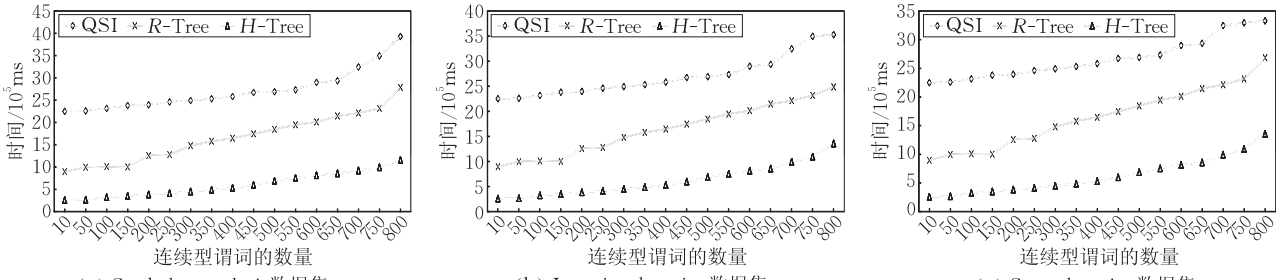


(a) Stock data analysis数据集 (b) Intrusion detection数据集 (c) Spam detection数据集

图 8 本文将离散型属性从 1 增加到 10 的过程中 3 种算法的表现(从图中可以明显看出离散型属性的增加几乎对 *H*-Tree 没有影响，因为多属性触发索引本身的扫描速度基本不会增加)

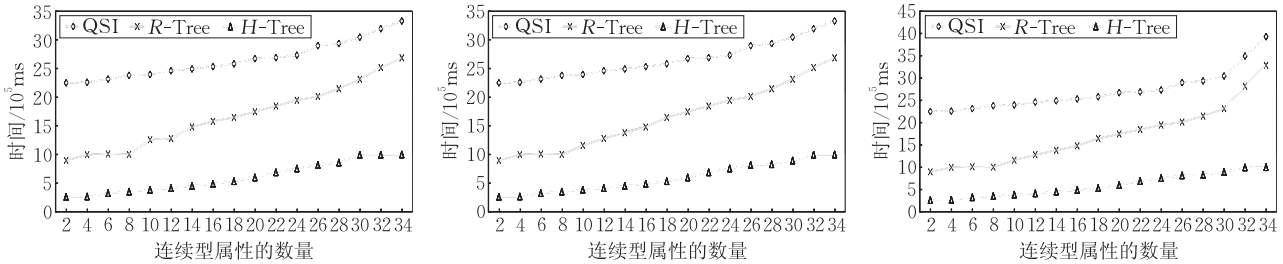
5.2.3 连续型谓词规模的影响

本文比较关心连续性谓词规模不同的情况下 3 种算法的表现. 图 9 和图 10 显示了两组实验的结果. 其中分别用连续型属性的数量和连续型谓词的数量这两个指标来衡量连续性谓词对监测规则的影响. 依据这两个指标本文进行了相应的两组实验. 其中第 1 组是将监测规则数量固定在 2000，连续性属性数量固定在 3 维的情况下，将连续性谓词的数量从 10 渐增到 800. 第 2 组实验是在将连续型谓词的数量固定在 200，监测规则固定在 2000 的情况下，依次将连续型属性的数量从 2 增加到 34. 从两组结果中本文可以看到无论在何种参数情况下，*H*-Tree 都远远优于另外两种算法. 另外 *H*-Tree 和 UDI 都比 QSI 优秀的一个重要原因是他们都对监测规则构建了索引，这样监测规则起来的速度要比 QSI 快得多.



(a) Stock data analysis数据集 (b) Intrusion detection数据集 (c) Spam detection数据集

图 9 本文将连续型谓词数量从 10 增加到 800 的过程中 3 种算法的表现(从图中可以明显看出 3 种算法的开销都明显增加，只是由于 *H*-Tree 对每个元组都进行命中扫描，所以开销增加幅度不如其他 2 种算法大)



(a) Stock data analysis数据集　(b) Intrusion detection数据集　(c) Spam detection数据集

图 10　本文将连续型属性从 2 增加到 34 的过程中 3 种算法的表现(从图中可以明显看出由于 *H*-Tree 采用了层次化的索引，虽然高维索引的维度增加会带来开销的增加，但对 *H*-Tree 的开销影响不大)

5.2.4　公开数据集上的实验结果

最后，本文通过公开的数据集(表 3 所示)来对 *H*-Tree 和 *R*-Tree 的性能进行比较. 实验方法如下：将公开的数据集的规则作为监测规则注册到系统中，然后区分出所有的离散型属性和连续型属性. 然后构造 10000 条数据流元组推送到系统中，比较两个算法的时间. 表 3 显示了实验结果. 通过表 3 本文可以观察到，在所有公开的数据集上，本文的 *H*-Tree 性能也比 *R*-Tree 的优秀，尽管内存开销比 *R*-Tree 要高一些. 综上所述，由于采用了层次化的索引框架以及高效的监测规则算法，*H*-Tree 性能较传统的解决算法有了大幅度的提升，非常适合大数据背景下的运营数据流在线监测系统.

表 3　公开数据集上的数据流监测规则效率比较

数据流	*H*-Tree			*R*-Tree		
	时间/ms	内存/MB	Drop/%	时间/ms	内存/MB	Error/%
syn-5	281712	15.67	0.389	1236000	12.29	0.376
syn-10	211668	12.63	0.291	1037125	10.25	0.054
Spam	333226	31.15	0.013	1417406	26.29	0.055
Intrusion	281953	20.93	0.134	8982137	18.04	0.169
Adult	306523	26.37	0.135	1225031	21.08	0.172
Magic	293145	23.07	0.032	1036250	22.45	0.097
Winered	293657	23.57	0.057	1131250	20.15	0.121
Winewhite	283031	22.95	0.133	1338120	18.15	0.182
Census	279782	20.76	0.964	1383600	16.17	0.817

6　结束语和下一步工作

在以实现大型集团企业的运营状态在线监测为背景的大数据流环境中，尤其是多媒体数据流环境下，数据流包含的属性越来越多，不同属性之间存在差异的异构大数据流也日益壮大，由于监测规则规模异常庞大，传统的高维索引方法对监测规则集合构建的索引性能无法满足实时性的要求. 为了解决这个问题，从异构大数据流本身的特点出发，结合不同属性上谓词的特点，本文提出了一种层次化的索引结构(*H*-Tree). 在 *H*-Tree 中，首先根据离散型谓词构建 Treap 树作为第 1 层索引；在第 2 层，将每一个连续型谓词映射为多维空间中一维，进而构建多维索引. 当数据流元组到来时，首先扫描 *H*-Tree 的第 1 层触发索引，完成离散型谓词的监测规则；当命中触发索引以后再搜索第 2 层索引. 文中方法既充分利用了第 1 层离散型触发索引的快速扫描能力，又借助第 2 层连续型谓词的高维区块划分，将连续型谓词映射为不同的区间块，进一步加快检索速度. 在公开数据集和人工数据集上的实验结果表明，在不降低准确率的情况下，*H*-Tree 的性能优于已有的解决方法.

参　考　文　献

[1] Zhang P, Zhu X, Shi Y. Categorizing and mining concept drifting data streams//Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data mining. Las Vegas, USA, 2008: 812-820

[2] Zhang P, Zhu X, Tan J, et al. Classifier and cluster ensembles for mining concept drifting data streams//Proceedings of the 10th IEEE International Conference on Data Mining. Sydney, Australia, 2010: 1175-1180

[3] Wu K L, Chen S K, Yu P S. Interval query indexing for efficient stream processing//Proceedings of the 13th ACM International Conference on Information and Knowledge Management. Washington, USA, 2004: 88-97

[4] Campailla A, Chaki S, Clarke E, et al. Efficient filtering in publish-subscribe systems using binary decision diagrams//Proceedings of the 23rd IEEE International Conference on Software Engineering. Toronto, Canada, 2001: 443-452

[5] Guttman A. R-trees: A dynamic index structure for spatial searching//Proceedings of the ACM International Conference on Management of Data. New York, USA, 1984: 47-57

[6] Schneider R, Seeger B, Beckmann N, et al. The R^*-tree: An efficient and robust access method for points and rectangles//Proceedings of the International Conference on Management of Data. NJ, USA, 1990: 322-331

[7] Sellis T K, Roussopoulos N, Faloutsos C. The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects//Proceedings of the 13th International Conference on Very Large Data Bases. Brighton, UK, 1987: 507-518

[8] Zhang P, Zhu X, Tan J, et al. Classifier and cluster ensembles for mining concept drifting data streams//Proceedings of the 10th IEEE International Conference on Data Mining. Sydney, Australia, 2010: 1175-1180

[9] Zhang P, Zhu X, Guo L. Mining data streams with labeled and unlabeled training examples//Proceedings of the 9th IEEE International Conference on Data Mining. Florida, USA, 2009: 627-636

[10] Zhang P, Zhu X, Shi Y, et al. Robust ensemble learning for mining noisy data streams. Decision Support Systems, 2011, 50(2): 469-479

[11] Zhang P, Li J, Wang P, et al. Enabling fast prediction for ensemble models on data streams//Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining. CA, USA, 2011: 177-185

[12] Aragon C R, Seidel R G. Randomized search trees//Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer Science. Gdansk, Poland, 1989: 540-545

ZANG Wen-Yu, born in 1988, Ph.D. candidate. Her current research interests include data mining, stream computing and network & information security.

LI Jun, born in 1983, Ph. D., engineer. His current research interests include network & information security, database and data mining.

FANG Bin-Xing, born in 1960, Ph. D., professor, Ph. D. supervisor, member of Chinese Academy of Engineering. His current research interests include network security and information content security.

TAN Jian-Long, born in 1974, Ph. D., professor, Ph. D. supervisor. His current research interests include information security, similarity calculations and text retrieval.

Background

With the development of database technology and real-time filtering algorithms, the processing, management of the complex data streams has become more and more popular in everyday life. Due to scalable online queries and multiple different attributes, Heterogeneous Streams processing can benefit real-time businesses applications such as stock analysis systems, net traffic monitoring systems, packet route selection. Thus, it may also bring serious threats to data stream querying. These heterogeneous streams can be sensitive since scalable queries contained complex predicates are difficult to index. Driven by the processing requirements of stream applications, a hierarchical-based index strategy is adopted to implement the real-time query processing in Heterogeneous Streams. Compared to existing works, in our solution, two-hierarchical index framework adopted to speed up the stream processing procedure. Specially, in the first floor, we build trigger index based on hash to improve the scanning speed; in the second floor, spatial indexing strategy is used which not only support quick insertion and deletion of queries but also efficiently search the target tuple. Experiments show that our method could satisfy the scalable real-time querying requirements.

This work is supported by the National Natural Science Foundation of China (Grant No. 61370025), National High Technology Research and Development Program (863 Program) of China (Grant Nos. 2011AA010703, 2012AA012502), National Basic Research Program (973 Program) of China (Grant No. 2013CB329606) and the "Strategic Priority Research Program" of the Chinese Academy of Sciences (Grant No. XDA06030200).