

**毕业设计（论文）文献翻译**

|  |  |
| --- | --- |
| 文献名称 | 基于收缩树的高效加权Web日志挖掘规则 |
| 专业名称 | 软件工程 |
| 班级学号 | 15205231 |
| 姓名 | 张建赟 |
| 指导教师 | 胡正 |

**2019 年 3 月 10 日**

**基于收缩树的高效加权Web日志挖掘规则**

P. Sampatht, C. Ramesht, T. Kalaiyarasit, S. Sumaiya Banut, G. Arul Selvan

Department of Computer Science & Engineering, Bannari Amman Institute of Technology Sathyamangalam, Tamil Nadu, India Department of Computer Science & Engineering, EGS Pillay Engineering College, Nagapattinam, Tamil Nadu, India

*摘要—* **Web挖掘技术被用来分析Web信息资源。利用Web内容挖掘和结构挖掘方法对网页内容进行分析。用户访问信息保存在Web日志下。使用挖掘技术来分析Web日志。Web日志在Web服务器环境下维护。从Web日志中提取用户访问模式。使用访问模式改进了Web内容管理和链接连通性。关联规则挖掘技术用于提取项集关系。在规则挖掘过程中使用项集频率。网页权重用于表示网页的重要性。加权规则挖掘技术用于获取具有权重值的频繁访问的网页。**

**频繁模式MMMG算法设计用于在数据库中查找常见数据集合。在频繁模式挖掘算法中，内存容量和运行时间要求非常高。收缩树结构是一种用于频繁模式挖掘操作的可重构结构。高吞吐量和更快的执行速度是基于收缩树的可重构体系结构的亮点。收缩树机制用于Web访问日志的频繁模式提取过程。针对加权规则挖掘过程，增强了基于收缩树的规则挖掘方案。系统采用自动权重估计方案。动态网页权重分配方案使用页面请求计数和跨度时间值。该系统对权值估计过程进行了改进，包括时间跨度、请求计数和访问序列细节。基于用户兴趣的页面权重用于提取频繁项集。**

***关键词：网络日志，结构挖掘，网络内容挖掘，收缩树。***

**Ⅰ. 简介**

频繁模式挖掘的目标是确定数据库事务中的哪些项通常一起出现。考虑到典型的现代数据库的规模，全面搜索通常是不可行的，这使得这成为一个具有挑战性的计算问题。FP-growth算法[1]使用两次扫描将数据库中的所有事务存储为树。FP-growth算法最初是从软件开发人员的角度设计的，它使用递归来遍历树和挖掘模式。直接在硬件中实现递归处理很麻烦，因为动态内存分配通常需要一些软件管理。因此，软件中广泛使用的动态数据结构在硬件实现中很少采用，很难将FP-growth算法直接转换为硬件实现。

在[2]中，我们之前介绍了用于频繁模式挖掘的可重构收缩树结构，并描述了使用现场可编程门阵列（FPGA）平台的原型。收缩树在扫描数据库时被配置为以流水线方式存储候选模式的支持计数，并由一个简单的软件模块控制，该模块读取支持计数并做出修剪决策。本文重点通过消除计数节点，对[2]中引入的原方案进行了改进，提出了一种新的计数模式算法[11]。提出并实现了一种新的适合于收缩树挖掘的数据库投影方法。基于所提出的可重构平台，研究了FPGA和软件/硬件的挖掘时间及其相关性。我们在几个实验平台上进行了基准测试。实验结果表明，基于FPGA的方法能比软件实现的FP-growth算法快几倍。

**Ⅱ. 相关工作**

与传统计算平台相比，FPGA的一个优势是能够在操作级粒度上执行并行算法，而不是在模块级或更高级别。最近有几项关于基于硬件的数据挖掘算法加速器的研究，在[3]中，作者描述了决策树分类（DTC）算法的硬件结构，并表明优化基尼分数计算可以显著提高整体性能。Apriori算法在FPGA上的并行实现首先在[4]中完成。由于多次读取事务数据库所需的处理时间，硬件实现速度仅比最快的软件实现速度快4%。[5]提出的HAPPI体系结构采用流水线方法解决了基于Apriori的硬件方案的瓶颈问题。当不同项目的数量和最小支持值增加时，HAPPI架构在[4]中优于[5]中的架构。几个独立评估得出的结论表明，FP-growth是最有效的关联规则挖掘算法之一[6，1]，虽然目前存在软件解决方案，但我们尚未得知FP-growth算法目前具备基于硬件的实现。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Items | ID | Items |
| 1 | B,C,D | 5 | A,B,C |
| 2 | B,C | 6 | A,B,C |
| 3 | A,C,D | 7 | A,B,D |
| 4 | A,C,D |  | |

图1. 一个简单的数据库事务

**Ⅲ. 设计收缩树**

1. *频繁模式挖掘*

给定一个数据库事务D，其中每一行代表一个事务，如图1所示，让I = 是数据库D和 是所有事务的集合。对于任何事务。作为模式（项集）P的支持计数，其中P是一组项。等于包含P的事务数。如果不小于预定义的最小支持阈值D，则称模式（项集）P很频繁。频繁模式挖掘的目标是在D中查找满足它的模式集。在图1中，并让4。，因此并不频繁；，因此频繁。

1. *收缩树*

在超大规模集成电路术语中，收缩树是以多维树模式排列的流水线处理元素（PE）。我们的架构的目标是模拟FP增长算法的内存布局，同时实现更高的吞吐量。然后，在FPGA硬件中映射的收缩树的作用类似于软件中使用的FP树。对于事务数据库，收缩树中元素的相对位置应与FP树中元素的相对位置相同。为了实现这一目标，本研究中的收缩树可根据以下观察结果进行设计：

1.对应于FP树中根节点的控制PE充当收缩树的输入和输出接口。

2.收缩树构造算法在接收到新事务时从根节点开始。对于在事务中存储任意两个项目的两个节点，一个节点必须是另一个节点的祖先。如果两个事务共享一个公共前缀，则使用一个前缀路径合并共享的部分。节点的度数等于共享公共前缀的事务数。在最坏的情况下，节点的等级等于频繁项目的数量n。除非使用动态部分重新配置，否则硬件中的PES不能像软件中那样动态创建和删除。如果每个PE连接到n个子PES，那么每个PE中的硬件结构和操作将非常复杂。为了避免这种阻碍，收缩树中的每一个PE都与其最左边的孩子相连。

3.在FP-growth算法中，每个事务中的第一个项存储在根的子项中，而其他项存储在其子项中。然而，收缩树中的所有PE都是并行运行的，并且在收缩树硬件实现中没有“指针”概念。由于树的固有特性，事务的操作通常从根开始。因此,控制PE充当收缩树的输入输出接口。每个时钟周期，一个项目被传输到控制PE，后者可以将该项目转发给其子项。

每个PE都有一个关联的级别。控制PE处于O级。一般PE的级别等于其与控制PE的距离。每一个PE的孩子有相同的水平。

特点 1. 每个普通PE只有一个直接连接到其最左边的子级的父级。其他孩子通过他们的左兄弟间接地与父母联系。

特点2. 一棵收缩树的W值和K值分别为。

PE有三种操作模式：写入模式、扫描模式和计数模式，将在下面的章节中讨论。收缩树是在写模式下构建的（算法1）。输入项按照定义的写入模式算法设置的方向从根节点流式传输。在扫描模式和计数模式下提取候选项集的支持计数。我们将此过程称为候选项集匹配。

**Input:** Input item

Match 0; InPath 1

If PE is empty **then** //Step (1)

Store the item ;count l;match 1; stop forwarding

**else if**( is in PE) and (InPath = 1 )then //Step(2)

match 1; count++; stop forwarding

**else if** match = 0 then //Step(3)

Inpath O;forward to the sibling

Else //Step( 4)

Forward to the children

**End if**

算法1: 在每个PE中的写模式算法

1. *收缩树创建*

写模式算法的设计原则是，在给定相同事务数据库的情况下，所构建的收缩树应具有与FP树相似的布局。事务中的第i个项映射到收缩树中的第i个级别。对于同一事务路径中的任意两个PE，第i级的PE是或更高级别的PE的祖先。PE与它的同级事务的路径绝不相同。假设事务数据库有N个频繁项。第一级PE的数目最多为N。收缩树的深度最多为N。假设图1中的所有项目都是频繁的。所有交易中的第一个项目只包括项目A和B。因此，我们只需要收缩树的第一级中的两个PE。但是，计算每个级别中的项数可能会损害总体模式挖掘性能。因此，K和W的值通常设置为等于N。如果事务中的项目数小于N，则事务的最后一个项目可能不会放入收缩树的叶中。如果两个事务共享相同的前缀，则它们将共享收缩树中的路径。总之，写模式算法背后的设计思路是，项目通过FP树的路径与它在收缩树中的路径相同。

在L算法中提出了写模式算法，在每个时钟周期内，收缩树的每个PE中都运行相同的算法。也就是说，每个PE的内部硬件结构是相同的。最初，所有的PE都是空的。每个时钟周期将一个项目加载到控制PE中，从而将每个项目传输到通用PE中。在将事务中的所有项发送到收缩树之后，将向控制PE发送一个控制信号，该信号说明旧事务的终止和新事务的开始。该信号将广播给所有的PE，这些PE将为下一个事务重新初始化。初始化包括重置算法L的第一行中的匹配和输入标志。该算法的输入是一个项目IT。匹配标志在PE中的项与之匹配时设置。当PE不包含当前事务中的任何项时，不会设置inpath标志。

**Ⅳ. 收缩树模式挖掘**

为了挖掘收缩树中的频繁模式，需要一个协作的硬件/软件平台。软件将候选模式发送到收缩树。在一定时钟周期之后，收缩树将候选模式的支持计数发送回软件。软件将支持计数与支持阈值进行比较，并确定候选模式是否频繁。在软件中用支持阈值检查了所有候选模式之后，就完成了模式挖掘。获取候选模式支持计数的方法称为候选项集匹配。

1. *候选项集匹配*

匹配的主要原则是，包含查询的候选项集的任何路径都将报告给控制PE。请注意，这样的路径可能包含比查询项集更多的项。在引入项目集匹配之前，我们研究了收缩树的一些有用特性，这些特性将有助于频繁的模式挖掘。如前几节所述，每个频繁项目都被分配一个序列号。在写入模式算法和扫描模式算法中，每个事务中的项目都按递增顺序进入收缩树。

获取候选项集支持计数的第一步是找到包含最后一个项的PE。这些被称为报告PE的PE负责报告存储的计数。在第二步中，将存储在报告PE中的支持计数报告给软件。由于每个PE都有独立的硬件组件和存储的数据，因此在报告PE中设置了一个标志IsLeaf。在第二步中，每个PE检查自己的IsLeaf标志。如果设置了，PE将报告其计数。匹配算法应在建立收缩树后运行。指示扫描模式的信号首先从控制PE广播到所有PE。扫描模式算法的设计原理与写模式算法相似。稍后到达候选项集中的项遵循前一项的路径。然而，写模式算法的目标是寻找与新事务共享相同前缀的路径。相反，扫描模式算法的目标是定位包含指定项集的所有路径。在扫描模式算法中，最后一个项目所在的PE是报告PE。由于可能存在多个报告PE，因此在PE转发时，候选项集中的某些项必须重复。如前几节所述，每个PE有两个输出接口。扫描算法的设计基于以下假设：

扫描模式算法如算法2所示。在步骤3中，即使小于存储的项，也应该发送给的兄弟。这是因为同级中的项可能小于或等于。如果输入项大于存储项，则的同级或子级中的存储项可能等于。因此，应将项t转发到所有开启的门，如步骤4所示。如果项等于并且底门打开，则可以匹配，如步骤2所示。如果兄弟的某些儿子具有与匹配的存储项，则仍需要将发送给的兄弟。总之，无论与的关系如何，都应将发送给的兄弟。

**Input:** Input item

Open the bottom door:IsLeaf 0

**If** PE is empty **then** //Step(1)

Stop forwarding

**else if** (it is in PE) and (bottom door is open)then //Step(2)

I sleaf l;forward to the sibling

**else if** < the item in PE then //step(3)

I sLeaf 0, close the bottom door; forward to the sibling;

**else if** > the item in PE then //step(4)

IsLeaf 0, forward to the sibling;

forward to the child if the bottom door is open

**end if**

算法2. 每个PE中的扫描模式算法



图2. 基于等效收缩树的加速硬件体系结构

1. *候选项集计数计算*

根据收缩树的属性1，从任何PE到控制PE只有一条路径，因为每个PE都有唯一的父级。一旦候选项目集中的所有项目发送到收缩树，表示计数模式的控制信号将广播到整个收缩树。第一个子级的输入接口总是连接到其父级，而其他子级则在写入和扫描模式下接受来自同级的输入。当候选频繁项集被传递到收缩树中后，PE将候选项集的支持计数报告给其唯一的父项。未直接连接到其父级的PE将其计数发送到左侧的同级。父PE收集子PE报告的支持计数，并将其发送到自己的父亲方向。算法3给出了每个PE中的计数模式算法，其中支持计数以流水线方式传输到每个PE的父级。算法的输入分别是PE的兄弟和子节点发送的两个计数值。变量是本地存储项的编号。该项控制PE将所有计数值相加，并将其作为输出信号发送。

Input: ,

Sent 0;

If(Sent=0)and(IsLeaf-1)then

Sent 1

forward()to its parent direction

else

forward to its parent direction

end if

算法3. 每个PE中的计数模式算法

**V. 按数据库投影的树缩放**

假设一个基于树的结构可以适应任何任意数据库的可用硬件资源（内存或逻辑空间），这是不太合理的。如果内存或逻辑空间不够大，就无法容纳整棵树，则必须将数据库划分为多个较小的数据库，其中的频繁项较少。如果没有数据库投影技术，当频繁项数量很多时，暴力式候选项目匹配将耗费掉不可容忍的时间。

1. *数据库投影模型*

为了使用收缩树挖掘频繁的项目集，将原始数据库投影到子数据库中。每个预测数据库的频繁项不超过，并保证可适用于FPGA。我们用一个例子来说明数据库投影。假设FPGA逻辑最多可以容纳一个包含两个频繁项的收缩树。图1中的数据库有四个频繁项，投影到子数据库中，每个子数据库最多有两个频繁项。频繁项通常按降频顺序排序，这就引入了密集的树结构。为了便于说明，我们按字母顺序排列经常项目，即A、B、C、D。从经常项目A开始，包含A的一组事务被收集为A计划数据库。由于A-projected数据库中有三个常见的B；C；D项，因此应该进一步将其投影到两个子数据库中。A-projected数据库和A-projected数据库。

1. *数据库投影频繁生成项目集*

对于每个频繁项目集，从的投影数据库（ 或 ）生成的频繁项目集与组合。此过程表示为。单前缀路径FP树的优化与[1]中提出的方法相同。但是，不需要为单个前缀路径p执行数据库投影，因为它已生成频繁模式集 。从包含单个前缀路径的树的投影数据库生成的频繁项集应与该单个前缀路径中的每个模式组合在一起，单独表示为。这是因为Q的顶部分支节点被空根替换。在树Q中生成的每个频繁模式与中的相结合。

**VI. 基于收缩树的加权规则挖掘**

该系统设计用于对Web日志进行加权规则挖掘。系统采用自动权重估计方案。每个网页都会根据请求计数和顺序分配一个权重值。收缩树是以多维树模式排列的流水线处理元素（PE）。事务项通过候选项匹配和计数更新操作更新到收缩树中。利用加权规则挖掘概念，对基于收缩树的算法进行了改进。在加权规则挖掘模型中，候选项和项集使用权重值进行更新。该系统设计用于在Web使用日志中查找频繁的模式。系统采用基于收缩树的模型。在系统中执行规则挖掘和加权规则挖掘操作。系统分为五个主要模块。它们是网络日志分析、重量估计过程、收缩树构造。

1. *网站日志分析*

网页请求详细信息保存在Web日志中。为每个页面请求更新页面URL、IP地址、请求时间和会话ID。每个页面请求都更新为一个单独的条目，其中包含用户的唯一会话ID。在清洗过程中去除噪声数据值。数据填充过程将日志数据传输到数据库中。页面请求条目根据会话ID分组，会话转换将为所有页面请求安排单个事务。

1. *权重估测过程*

权重估测模块用于为网页分配权重。访问日志用于权重估计过程。系统采用基于访问序列的权值估计模型，权重值用于加权规则挖掘过程。

1. *收缩树构造*

收缩树维护候选集及其频率值。内存映射为类似的子树值。网页请求序列在收缩树中更新，为每个页面请求更新频率值。

1. *规则挖掘过程*

规则挖掘过程是为了寻找频繁模式。页面请求序列模式从收缩树中提取。在挖掘过程中使用支持值，规则挖掘过程使用频率值。

1. *权重规则挖掘*

页面权重用于加权规则挖掘过程。页面请求频率和权重值在收缩树中更新。使用权重值提取频繁模式，加权支持估计作用于页面。

**VII. 结论**

关联规则挖掘技术用于提取频繁的模式。收缩树用于排列具有频率值的候选集。由于收缩树的大小有限，必须将事务性数据库投影到较小的数据库中，每个数据库都可以在硬件中有效地挖掘。提出并实现了一种充分利用FP-growth优势的高性能投影算法。它通过将树划分为密集和稀疏的部分，并将密集的树发送到硬件来减少挖掘时间。针对加权规则挖掘过程，增强了基于收缩树的规则挖掘方案。系统采用自动权重法去估计方案。

**参考文献**

[I] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," Data Mining and Knowledge Discovery, vol. 8, no. 1, pp. 53-87, Jan. 2004.

[2] S. Sun and J. Zambreno, "Mining Association Rules with Systolic Trees," Proc. In!'1 Conf Field-Programmable Logic and Applications (FPL '08), Sept 2008.

[3] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An FPGA Implementation of Decision Tree Classification," Proc. Conf Design, Automation, and Test in Europe (DATE), pp. 189-194, Apr. 2007.

[4] Z. Baker and V. Prasanna, "Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs," Proc. IEEE Symp. Field- Programmable Custom Computing Machines (FCCM), pp. 3-12, Apr. 2005.

[5] Y.-H. Wen, J.-W. Huang, and M.-S. Chen, "Hardware-Enhanced Association Rule Mining with Hashing and Pipelining," IEEE Trans. Knowledge and Data Eng., vol. 20, no. 6, pp. 784-795, June 2008.

[6] A. Ghoting, G. Buehrer, Y.-K. Chen, and P. Dubey, "Cache-Conscious Frequent Pattern Mining on a Modern Processor," Proc. Int'I Conf Very Large Data Bases (VLDB), pp. 577-588, 2005.

[7] R. Bayardo, B. Goethals, and M. Zaki, eds., Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI), Nov.2004.

[8] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets," Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI), 2004.

[9] C. Lucchese, S. Orlando, and R. Perego, "kDCI: On Using Direct Count Up to the Third Iteration," Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMl), 2004.