维普资讯 http://www.cqvip.com

高效 FP-TREE 创建算法*)

邱 勇 兰永杰
(山东工商学院信息与电子工程学院 烟台264005)

摘 要 如何从大型数据库中挖掘关联规则是数据挖掘的一个重要的问题。FP- growth 是一个著名的不产生候选集的高效频繁模式挖掘算法，它使用专门的数据结构 FP-tree。为了进一步提高 FP-grown 算法效率，提出一个新的并行算法 PFPTC，可以并发地创建子 FP- tree，以及一个 FP-tree 合并算法称作 FP- merge，可以将两个 FP- tree 合并为一个。

关键词 数据挖掘，关联规则，并行算法，FP- tree

High Efficiency FP-tree Creating Algorithm*)

QIU Yong LAN Yong-Jie
(School of Information & Electronic Engineering, Shandong Institute of Business and Technology, Yantai 264005)

Abstract Mining association rules from large databases is an important problem in data mining. FP-growth is a famous algorithm to mine frequent patterns and it is non-candidate generation algorithm using a special structure FP-tree. In order to enhance the efficiency of FP-grown algorithm, propose a novel parallel algorithm PFPTC to create sub FP-trees concurrently and a FP-tree merging algorithm called FP-merge which can merge two FP-trees into one FP-tree.

Keywords Data mining, Association rules, Parallel algorithm, FP-tree

1 引言

关联规则是数据挖掘中一个重要的研究内容，它表达了在大量事务数据中数据项之间的关联或相关联系，而产生频繁项目集则是产生关联规则的关键步骤[1,2,5,6]。Apriori[1]，PARTITION[2]，ML_T2L1[3]，SETM[4]等是关于频繁项目集发现算法，其中 Apriori 算法[1]最为经典，其它的算法大多是根据 Apriori 算法进行改进，或衍生变化而来。Apriori 算法通过自底向上搜索发现频繁项目集，它要产生长度为1至最大频繁项目集的候选集。它用一种称作逐层搜索的迭代方法，k-项集用于发现(k+1)-项集。首先找出频繁1-项集的集合。然后利用前者找出2-项集的集合，如此迭代，直到不能找出新的频繁 k-项集为止。最后由频繁项目集产生关联规则。在每次迭代扫描中，它要通过计算出现率决定哪个项是频繁的。当频繁的模式长度较长时它的性能会很差，其中候选集的产生是最耗时的工作，占据了整个计算量的大部分[5]。

FP-growth[6]是近年来公布的效率较高的频繁集挖掘算法之一。它是一种不产生候选的挖掘频繁项目集的方法。它通过构造一个高度压缩的数据结构(FP-tree)，压缩原来的事务数据库，避免了高代价的候选产生，获得了更好的效率。

然而基于 FP- tree 的挖掘要创建复杂的数据结构，要对数据库进行两遍扫描，当数据库很大时 FP- tree 的创建是一个瓶颈问题。在本文中，我们提出了一种新的创建 FP- tree 并行算法和一种 FP- tree 合并算法 FP- merge，它可以将两个 FP- tree 合并成为一个。

2 FP-growth 算法

Han 等人提出了一种称为频繁模式树或 FP- tree 的数据结构及相应的频繁项目集生成算法 FP- growth[6]。FP-growth 从 FP- tree 中挖掘频繁项目集而不用产生候选集。FP- tree 是一个高度压缩的数据结构，它用较少的空间存储了频繁项目集挖掘所需要的全部信息。FP- tree 是一种扩展的前缀树，只有频繁数据项会成为树中的结点，每个结点包含数据项的标记和从根的子结点到该结点组成的项目集的支持数，从根到叶子结点的路径按照数据项的支持数排列，父结点的支持数大于等于它的子结点支持数之和。文[6]对 FP-tree 的定义如下：

1)它有一个标记为"null"的根节点，它的子节点为一个项前缀子树(item prefix subtree)的集合，还有一个由所有频繁项(frequent item)构成的头表(header table)。

2)每个项前缀子树的节点有三个域：item-name，count，node_link. item-name 记录了该节点所代表的项的名字。count 记录了所在路径代表的交易(ransaction)中达到此节点的交易个数。node_link 指向下一个具有同样的 item-name 域的节点，要是没有这样一个节点，就为 null。

3)频繁项头表(frequent item header table)的每个表项(entry)由两个域组成：(1)item-name；(2)node_link。node_link 指向 FP-tree 中具有与该表项相同 item-name 域的第一个节点。

构造 FP- tree 需要对数据库进行两遍扫描。第一次扫描发现每个数据项的支持数，即得到频繁1-项集。第二次扫描，

*)This paper is supported by Shandong Physical Science Foundation(Y2002G08).邱 勇 教授，主要研究方向为知识工程、数据库等；兰永杰 副教授，主要研究方向为软件工程与数据库。

维普资讯 http://www.cqvip.com

各事务内的属于1-项集的数据项按数据项的支持数的递减排序，并加到FP-tree中。如果两个事务共有一个公共的前缀，公共部分被合并成共享结点，相应地增加该结点的支持数。有同样标记的结点用数据项链连接起来。该数据项链用于简化频繁模式的挖掘。另外，要建立一个包含全部频繁项的头表，头表中有指针指向相应的数据项链。算法1实现了FP-tree的创建[6]。

算法1(FP-tree构造算法)

输入：事务数据库DB；最小支持度阈值ξ
输出：FP-tree
方法：1. 第一遍扫描事务数据库DB，收集频繁项的集合F和它们的支持度，使F中的项目的支持度大于等于最小支持度阈值ξ。
2. 对F按支持度降序排序，结果为频繁项表L。
3. 创建FP-tree的根结点，以"null"标记它。用L创建频繁项头表，指针为null。
4. 第二遍扫描事务数据库DB，对于DB中的每个事务执行以下操作：对事务中的频繁项按照L中的顺序进行排序，排序后的频繁项表记为[p|P]，其中p是第一个元素，而P是剩余元素的表。调用insert_tree([p|P], T)。

过程insert_tree([p|P], T)的执行如下：

如果T有一个子结点N，其中N.item-name=p.item-name，则将N的count域值增加1；否则，创建一个新节点N，使它的count为1，使它的父节点为T，并且使它的node_link和那些具有相同item_name域串起来。如果P非空，则递归调用insert_tree(P, N)。

FP-growth根据前缀分拆FP-tree，它递归地处理FP-tree的路径，产生频繁项目集。模式片段被连接起来以保证产生全部的频繁项目集。这样，FP-growth避免了高代价的对候选项集的生成及测试操作。FP-growth描述如下[6]：

算法2(FP-growth：在FP-tree中通过频繁模式增长挖掘频繁模式)

```
输入：一棵用算法1建立的树FP-tree
输出：所有的频繁项目集
方法：调用FP-growth(Tree,null).
、Procedure FP-growth(Tree,α)
{if Tree 只有一条路径P then
对P中的节点的每一个组合(记为β)做：
    产生频繁集β∪α，并且把它的支持度指定为β中节点的最小支持度.
else 对Tree的头表从表尾到表头的每一个表项(记为a)做：
    {产生频繁集β=a∪α，并且支持度为a的支持度
    建立β的条件模式库(conditional pattern base)和β的条件树(conditional FP-tree)Tree2
    if Tree2!=∅
    then 调用FP-growth(Tree2,β)}}
```

3 并行FP-tree构造算法

为了从数据库中有效地挖掘有用信息，我们需要解决挖掘的效率问题，当数据量很大时，挖掘算法的效率成为挖掘的关键问题。如前所述，显然构造FP-tree是FP-growth算法的关键步骤，FP-tree包含频繁项目集的压缩信息，可以容易地从FP-tree中挖掘频繁项目集。为了构造FP-tree，我们必须扫描数据库两次，一次用于创建1-项目集，另一次用于建造FP-tree。为了提高基于FP-tree挖掘算法的效率，开发并行算法是一种有效策略。为此我们给出一个构造FP-tree的并行算法PFPTC(Parallel Frequency Pattern Tree Construction)及一个相应的FP-tree合并算法。

首先，初始的数据库扫描发现频繁1-项目集。为了有效地得到频繁项，我们在可利用的处理器当中划分数据集。每个处理器得到相等数量的事务进行处理。结果，数据集被划分为n等份。每个处理器并行地计数本地事务中出现的数据项。在本地计数完成之后，要用全局统计发现频繁的数据项。最后，支持数小于最小支持度阈值的数据项被清除，将剩余的频繁的数据项按频数的降序排序。

要构造局部FP-tree需要对数据集进行二次完全的扫描，每个处理器读如同第一次同样的事务数据。使用这些事务数据，每个处理器建造它自己的频繁模式树，这些频繁模式树从null根开始。构造局部FP-tree过程如下：对排序的事务中第一个数据项，检测是否是根的一个子结点。如果它存在则增加这结点支持数。否则，为此项增加一个新的结点，为根节点的子结点，支持数为1。然后，将当前项结点当做新临时根结点并对其它数据项重复同样的步骤。我们不必为局部FP-tree创建头表和同名节点链，因为它仅仅对应局部数据库，我们可以在所有局部FP-tree合并成一个全局FP-tree后才创建头表和同名节点链。以下是具体算法描述。

算法3(PFPTC：并行FP-tree构造)

```
输入：事务数据库DB；最小支持度阈值ξ
输出：频繁模式树FP-tree
方法：FP-tree按下列步骤建立
设有n个处理器p1, p2, …, pn;
划分事务数据库DB为n部分：DB1, …, DBn
处理器pj扫描DBj对项目计数，然后建立1-项目集F，按支持度降序排序F，得到L.
for ( j = 1; j≤n; j++)concurrently do
{ 处理器pj扫描DBj，构造FP-treej，(不建立头表和同名结点链；}
  while(n>1)concurrently do
  for ( j = 1; j≤n; j=j+2)concurrently
    {处理器pj调用FP-merge(FP-treej, FP-treej+1, int(j/2)+1 );
    n=n/2;}
  为FP-tree1建立头表和同名结点链；
返回FP-tree1;
End;
```

算法4(FP-merge：FP-tree合并算法)

```
输入：两个局部FP-tree
输出：合并的FP-tree，名为FP-tree(result)
方法：call FP-merge(FP-tree1, FP-tree2, result)
Procedure FP-merge(FP-tree1, FP-tree2)
{ while FP-tree2 !=NULL, do
  { retrieve one item train in FP-tree2 from leaf to root;
    n=leaf.count;
Let the item train be [p|P], where p is the last element and P is the frontal list.
Call merge-insert ([p|P], FP-tree1, n).
  Remove the item train from FP-tree2;}
  rename FP-tree1 as FP-tree(result);}
```

算法5(merge-insert：插入项目串到FP-tree)

```
输入：项目串(item trains), FP-tree结点(node), 串支持度(support)
输出：插入后的FP-tree
方法：call merge-insert (items train, node, support)
Procedure merge-insert ([p|P], T, num)
{If T has a child N such that N.item-name = p.item-name
N.count= N.count+num;
Else do
{Create a new node N;
N.count = num;
Let N's parent link be linked to T;}
If P is not empty, Call merge-insert (P, N, num);}
```

让我们用表1的事务数据库举例说明算法。设处理器数量是2，最小支持度阈值为3。图1显示了两个局部FP-tree(FP-tree1 and FP-tree2)，图2至图4显示了FP-tree1和FP-tree2的合并过程。

表1 示例事务数据库DB

标识	项目	排序频繁项目	数据划分
T001	f, a, c, d, g, i, m, p	f, c, a, m, p	DB1
T002	a, b, c, f, l, m, o	f, c, a, b, m	DB1
T003	b, f, h, j, o, p	f, b, p	DB1
T004	b, c, k, s, p	c, b, p	DB1
T005	a, f, c, e, l, m, n	f, c, a, m	DB2
T006	p, b, c	c, b, p	DB2
T007	c, f	f, c	DB2
T008	f, c	f, c	DB2

维普资讯 http://www.cqvip.com

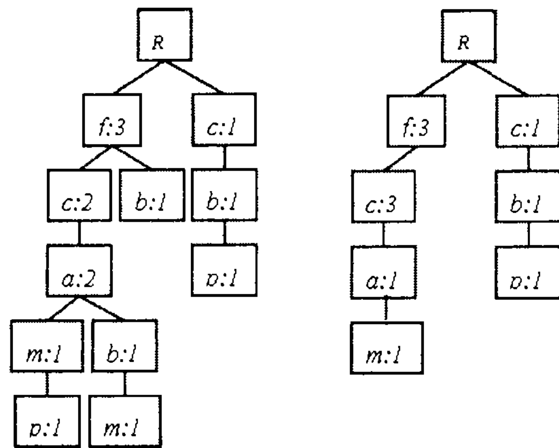

图1 两个局部 FP-tree：FP-tree1 和 FP-tree2

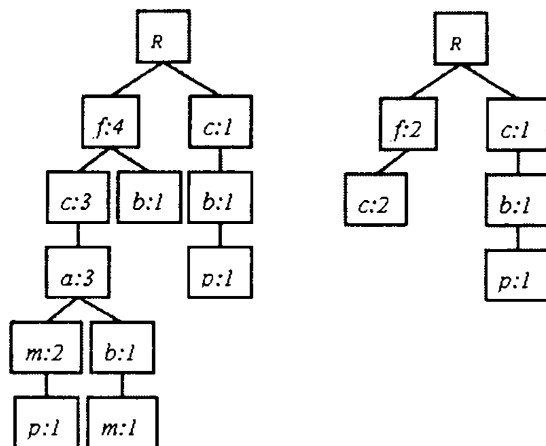

图2 从 FP-tree2移动（macf）到 FP-tree1

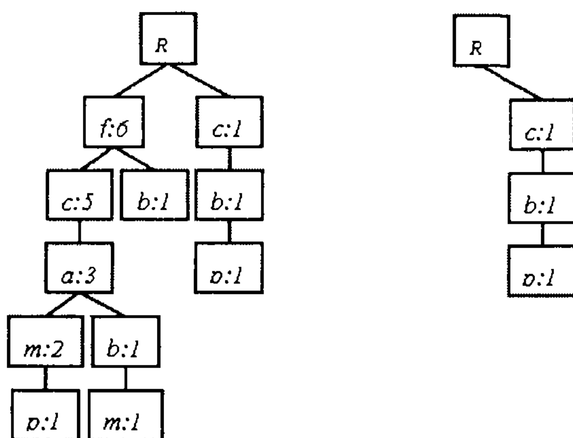

图3 从 FP-tree2移动（cf）2 到 FP-tree1

4 运行性能实验

实验的目标在于测试并行算法在不同尺寸数据集与非并行算法相比的性能。数据集采用 IBM QUEST 数据生成程序生成。所有的程序采用同样的编译环境，用同样的数据集和同样的参数并产生同样的模式。实验环境是奔腾41.8G PC，512Mb RAM，Window XP。试验结果表明 PFPTC 是有竞争力的，对于从大数据库或数据仓库挖掘频繁的数据项具有较好的效率和可伸缩性。实验结果如图5所示。

结论 本文中我们介绍了一种高效的创建 FP-trees 并行算法和一种 FP-tree 合并算法 FP- merge，实验表明并行 FP-tree 构造算法有较理想的性能。该算法可用作对大型数据库进行有效的频繁模式的挖掘。我们下一步的研究工作是把 FP-growth 算法与数据库更紧密地集成起来，将 FP-tree 的构造及挖掘任务在数据库服务器完成，建立基于 SQL 的高效可行的挖掘，使数据挖掘更实用。

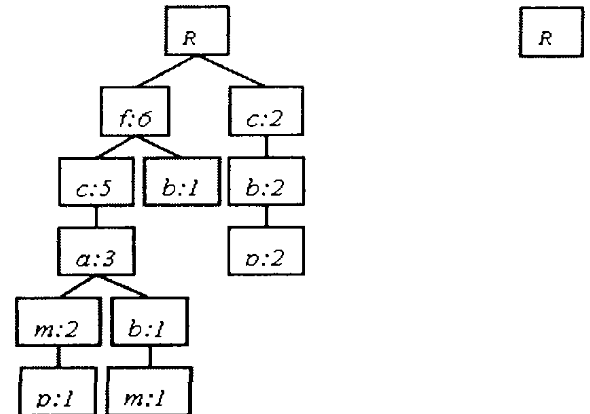

图4 从 FP-tree2移动（pbc）1 到 FP-tree1，合并完成

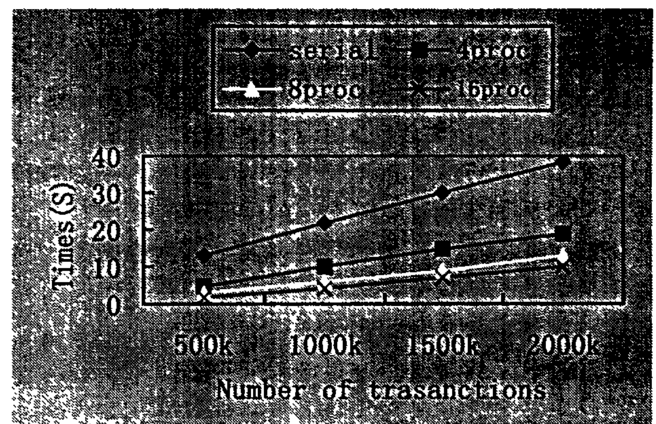

图5 实验结果

参 考 文 献

1 Agrawal R，Srikant R. Fast algorithms for mining association rules. In VLDB'94，487～499

2 Savasere A，Omiecinski E，Navathe S M. An efficient algorithm for mining association rules. In：Proc. of the 21st Intl. Conf on VLDB. Zurich，1995. 432～444

3 Han J，Fu Y. Discovery of multiple-level association rules from large databases. In：Proc. of the 21st Intl. Conf. on VLDB. Zurich，1995. 420～431

4 Houtsma M，Swami A. Set-Oriented mining for association rules in relational databases. In：Yu PS，ed. Proc. of the Intl. Conf. on Data Engineering. Los Alamitos：IEEE Computer Society Press，1995. 25～33

5 Zaki M J，Hsiao C J. CHARM：An efficient algorithm for closed itemset mining. In：Proc. SIAM Int. Conf. Data Mining，Arlington，2002. 457～473

6 Han J，Pei J，Yin Y. Mining Frequent Patterns without Candidate Generation. Kluwer Academic Publishers，Data Mining and Knowledge Discovery，2004. 53～87