文章编号: 1001-0920 (2012) 04-0618-05

基于频繁模式树的分布式关联规则挖掘算法

何 波

(重庆理工大学 计算机科学与工程学院，重庆 400054)

摘 要: 提出一种基于频繁模式树的分布式关联规则挖掘算法 (DMARF). DMARF 算法设置了中心结点, 利用局部频繁模式树让各计算机结点快速获取局部频繁项集, 然后与中心结点交互实现数据汇总, 最终获得全局频繁项集. DMARF 算法采用顶部和底部策略, 能大幅减少候选项集, 降低通信量. 理论分析和实验结果均表明了 DMARF 算法是快速而有效的.

关键词: 数据挖掘；频繁模式树；全局频繁项集；关联规则

中图分类号: TP311 **文献标识码:** A

Distributed algorithm for mining association rules based on FP-tree

HE Bo

(School of Computer Science and Engineering，Chongqing University of Technology，Chongqing 400054，China. E-mail：hebo@cqut.edu.cn)

Abstract: The paper proposes a distributed algorithm for mining association rules based on frequent pattern tree(FP-tree), named distributed algorithm for mining association rules based on FP-tree(DMARF) algorithm, which sets center node. DMARF algorithm makes computer nodes compute local frequent itemsets independently from the local FP-tree. Then the center node exchanges data with other computer nodes. Finally, global frequent itemsets are gained. DMARF can decrease candidate itemsets and communication traffic by using the strategies of top and bottom. Theoretical analysis and experimental results show that DMARF algorithm is fast and effective.

Key words: data mining；frequent pattern tree；global frequent itemsets；association rules

1 引 言

数据挖掘是从数据集中发现新颖的、预先未知的、隐藏和有趣的知识. 数据挖掘中的关联规则是一个重要的研究课题, 有着广泛的应用前景. 关联规则[1]描述的是在给定的事务集中频繁出现项集的规则, 关联规则挖掘的关键是获取频繁项集.

挖掘频繁项集的算法有 Apriori[1], FP-growth[2]和 PARTITION 等, 但这些算法并不适合分布式数据库. 目前, 分布式关联规则挖掘算法主要有CD[3]和FDM[4]等. CD 算法是基于 Apriori算法的简单并行化, 每次扫描后需要同步, 结点间通信量非常大. FDM算法对 CD 算法进行了改进, 在各个结点利用类 Apriori 算法挖掘出局部频繁项集, 各结点交换项集的支持度计数. 这些分布式算法大多采用类 Apriori 算法, 存在候选项集多、通信量大、同步次数多和扫描数据库次数多等问题. 针对这些问题, 本文提出一种基于频繁模式树的分布式关联规则挖掘算法 (DMARF). DMARF 算法设置了中心结点, 利用局部频繁模式树让各计算机结点快速获取局部频繁项集, 然后与中心结点交互实现数据汇总, 最终获得全局频繁项集.

2 相关描述

2.1 挖掘全局频繁项集的问题描述

全局事务数据库为 DB, 总的事务条数为D. 设P_1, $P_2, \cdots, P_n$ 为 n 台基于无共享体系结构的计算机结点 (简称结点), 即它们之间除通过网络传递信息外, 其他资源 (如硬盘、内存等) 全部是独立的, DB_i $(i = 1, 2, \cdots, n)$ 是 DB 存储于结点 P_i 上的局部事务数据库, 其中的事务有 D_i 条, 则

$$\mathrm{DB} = \bigcup_{i=1}^{n} \mathrm{DB}_i,\ D = \sum_{i=1}^{n} D_i.$$

收稿日期: 2010-12-16；**修回日期:** 2011-03-18.

基金项目: 教育部人文社会科学研究项目(09yjc870032).

作者简介: 何波(1978–), 男, 副教授, 从事数据挖掘、智能信息推荐等研究.

所谓挖掘全局频繁项集问题, 就是如何通过 n 台结点同时工作, 各台结点间通过网络传递有限的信息, 最终在整个事务数据库 DB 中挖掘出频繁项集.

2.2 相关定义

定义 1　对于某一项集 X, 在局部数据库 DB_i ($i=1,2,\cdots,n$) 中包含 X 的事务的条数称为 X 的局部频度, 用 X_{si} 表示.

定义 2　对于某一项集 X, 在全局事务数据库 DB 中包含 X 的事务的条数称为 X 的全局频度, 用 X_s 表示.

定义 3　对于项集 X, 若 $X_{si}\geqslant \mathrm{minsup}*D_i, i=1,2,\cdots,n$, 则称 X 是相对于 DB_i 的局部频繁项集 F_i, 其中 minsup 表示最小支持度阈值.

定义 4　对于项集 X, 若 $X_s\geqslant \mathrm{minsup}*D$, 则称 X 是全局频繁项集 F.

定义 5　如果项集 $X\subseteq Y$, 则 X 是 Y 的子集, Y 是 X 的超集.

定义 6　对于某一项目 x_i, 项集 $X=\{x_i\}$, 若 $X_s\geqslant \mathrm{minsup}*D$, 则称 x_i 是全局频繁项目. 所有全局频繁项目组成的集合 $E=\{x_1,x_2,\cdots,x_m\}, m=|E|$, 其中 $x_1,x_2,\cdots,x_m$ 均是全局频繁项目, 共有 m 个.

2.3 相关定理

定理 1　若项集 X 是 DB_i 的局部频繁项集, 则 X 的所有非空子集也是 DB_i 的局部频繁项集.

推论 1　若项集 X 不是局部频繁项集, 则 X 的超集一定不是局部频繁项集.

定理 2　若项集 X 是全局频繁项集, 则至少存在一个局部数据库 $\mathrm{DB}_i, i=1,2,\cdots,n$, 使 X 及 X 的所有非空子集均为 DB_i 的局部频繁项集.

定理 3　若项集 X 是全局频繁项集, 则 X 的所有非空子集也是全局频繁项集.

推论 2　若项集 X 不是全局频繁项集, 则 X 的所有超集必定不是全局频繁项集.

定理 4　若项目 x_i 不是全局频繁项目, 则所包含的 x_i 的项集一定不是全局频繁项集.

2.4 FP-tree[2]及FP-growth[2]算法

频繁模式树 (FP-tree) 是满足以下 3 个条件的树型结构:

1) 它由一个标为“null”的根结点和作为根结点的孩子的项目前缀子树集合以及频繁项目头表组成.

2) 项目前缀子树中的每一结点包含 3 个域, 即 item-name, count 和 node-link. 其中: item-name 记录项目名; count 记录能到达该结点路径所表示的事务的数目; node-link 为指向 FP-tree 中具有相同 item-name 值的下一结点, 当下一结点不存在时, node-link为 null.

3) 频繁项目头表的每一表项包含 2 个域, 即 item-name 和 head of node-link, 其中 head of node-link 为指向 FP-tree 中具有相同 item-name 值的首结点的指针.

对于给定的一事务数据库 db 及最小支持度阈值 minsup, 建立 FP-tree 的主要步骤如下:

1) 扫描 db 一遍, 得到各项目的频度; 根据最小支持度阈值 minsup 得到频繁项目; 对频繁项目按其频度由大到小排列 (次序记为 ξ), 形成头表.

2) 再次扫描 db, 对每一条事务中的所有频繁项目按次序 ξ 组成相应的项目集, 并插入到 FP-tree 中.

FP-growth 算法是基于 FP-tree 的频繁项目集分布增长挖掘算法. 算法的主要思想是: 如果 FP-tree 中只包含一条单一路径, 则直接测试这条路径上项目的所有组合, 它们的支持数是对应子路径中结点最小的支持数; 否则, 构造当前项目的条件模式基, 并在其对应的条件 FP-tree 进行挖掘, 获得频繁项目集.

3 DMARF 算法

3.1 DMARF 算法思想

DMARF 算法设置一个结点 P_0 作为中心结点. 首先, 挖掘出所有全局频繁项目, 组成集合 E, 各个结点根据 E 构建局部 FP-tree $_i$; 其次, 各个结点根据 FP-tree $_i$ 计算局部频繁项集 F_i, 并向中心结点发送局部频繁项集 F_i, 同时汇总得到所有结点局部频繁项集的并集 $F'\left(F'=\bigcup_{i=1}^{n}F_i\right)$, F' 便是全局频繁候选项集; 再次, 采用顶部和底部策略对 F' 进行剪枝, 并向其他结点广播剪枝后的 F'; 最后, 从各个结点收集 F' 中所有局部频繁项集 d 的局部频度 d_{si}, 获得 d 的全局频度 d_s, 挖掘出全局频繁项集. 根据定理 2, 全局频繁项集必定是某一局部频繁项集, 各个结点局部频繁项集的并集必定包含全局频繁项集. 因此, DMARF 算法能挖掘出所有的全局频繁项集.

DMARF 算法首先需挖掘出所有的全局频繁项目. 各个结点 P_i 扫描 DB_i 一次, 并计算各个局部项目的局部频度; 中心结点 P_0 从各个结点收集所有项目 E_i 的全局频度, 并挖掘出所有全局频繁项目组成集合 E. 中心结点 P_0 将按支持度降序排序的顺序将全局频繁项目 E 发送给各个结点; 然后, 各个结点 P_i 根据 E 构建局部 FP-tree $_i$. 根据定理 4, 若项目 x_i 不是全局频繁项目, 则所包含的 x_i 的项集一定不是全局频繁项集. 因此, 所构建的局部 FP-tree 只包含全局频繁项目, 大幅减少了局部 FP-tree 中项目的个数, 提高了挖掘全局频繁项集的效率.

顶部策略描述如下:

1) 确认 F' 中所有项集的最大项数为 k, 转 2).

2) 从各个结点收集 F' 中所有 k-项集的全局频

度, 转 3).

3) 判断 F' 中的所有 k-项集, 如果 k-项集 Q 是全局频繁项集, 则根据定理 3, 将 Q 和 Q 的所有非空子集加入全局频繁项集 F, 将 Q 和 Q 的所有非空子集从 F' 中删除; 否则, 将 Q 从 F' 中删除.

采用顶部策略判断全局频繁候选项集 F' 中的 k-项集 (k 是 F' 中所有项集的最大项数), 直接将全局频繁 k-项集的所有非空子集加入全局频繁项集 F, 可大大减少需要判断的候选项集, 降低网络通信量.

底部策略描述如下:

1) 从各个结点收集 F' 中所有 2-项集的全局频度, 转 2).

2) 判断 F' 中的所有 2-项集, 如果 2-项集 B 是全局频繁项集, 则将 B 加入全局频繁项集 F, 并将 B 从 F' 中删除; 否则, 根据推论 2, 将 B 和 B 的所有超集从 F' 中删除.

采用底部策略判断全局频繁候选项集 F' 中的 2-项集, 直接删除非全局频繁 2-项集的所有超集, 可减少候选项集, 降低网络通信量.

大多数分布式挖掘算法需进行多次同步. 比如, 如果频繁项集的最大长度为 k, 则往往需进行 k 次同步, 这样便造成各节点间的通信量较大, 挖掘效率较低. 在 DMARF 算法中, 计算局部频繁项集的工作可以异步执行, 只在最后结束时才进行 1 次同步, 从而大大减少了同步次数.

DMARF 算法中各结点采用 FP-growth 算法计算局部频繁项集, 与类 Apriori 算法相比, FP-growth 算法采用 FP-tree 这种压缩的数据结构, 只需扫描数据库 2 次, 从而大大减少了数据库的扫描次数和计算时间, 具有明显的优势. 性能分析表明, FP-growth 算法的计算速度比类 Apriori 算法快一个数量级.

3.2 DMARF 算法描述

DMARF 算法步骤如下:

1) 挖掘所有全局频繁项目, 组成集合 E, 各结点首先根据挖掘出的 E 构建其对应的 FP-tree $_i$, 然后采用 FP-growth 算法计算局部频繁项集 F_i.

2) 各结点向中心结点发送其局部频繁项集 F_i, 使得拥有所有结点局部频繁项集的并集 $F'\left(F' = \bigcup_{i=1}^{n} F_i\right)$ 是全局频繁项集 F 的超集; 然后采用顶部和底部策略对 F' 进行剪枝.

3) 向其他结点广播剪枝后的 F', 各结点利用局部 FP-tree 计算 F' 中所有局部频繁项集 d 的局部频度 d_{si}; 然后将其发送给中心结点, 汇总后最终获得 d 的全局频度 d_s.

4) 检查各局部频繁项集的全局频度 d_s 是否满足最小支持度阈值 minsup, 得到最终的全局频繁项集 F.

下面给出 DMARF 算法描述.

算法 1 DMARF 算法.

输入: 全局事务数据库为 DB, 共有 D 个元组, n 台基于无共享体系结构的计算机结点 P_i, $i=1,2,\cdots,n$, 设置 P_0 作为中心结点, 最小支持度阈值为 minsup.

输出: 全局最大频繁项集 F.

Step 1: /*挖掘全局频繁项目, 组成集合 E, 创建局部 FP-tree $_i$, 用 FP-growth 算法生成局部频繁项集*/

for $(i = 1; i <= n; i++)$ /*挖掘全局频繁项目 E */

{scanning DB_i once;

computing local frequency of local items E_i ;

P_i sends E_i and local frequency of E_i to P_0 ;

}

P_0 collects global frequent items E from E_i ;

E is sorted in the order of descending support count; /* 按支持度降序排序的全局频繁项目 E */

P_0 sends E to other nodes P_i;

for $(i = 1; i <= n; i++)$

{creating the FP-tree $_i$; /*创建各结点的 FP-tree$_i$*/

F_i =FP-growth (FP-tree $_i$, null);

/*采用 FP-growth 算法挖掘出局部频繁项集 F_i */

}

Step 2: /*各结点都获取所有结点局部频繁项集的并集, 利用顶部和底部策略略煎枝*/

for $(i = 1; i <= n; i++)$

P_i send F_i to P_0;

P_0 combines F_i equal F'; /*$F' = \bigcup_{i=1}^{n} F_i$*/

P_0 confirms the largest size k of itemsets in F';

for each itemsets $Q \in$ local frequent k-itemsets in F'

/* 根据顶部策略剪枝 F' */

{ P_0 broadcasts Q;

P_i sends $Q.si$ to P_0;

$Q.s = \sum_{i=1}^{n} Q.si$

if $(Q.s >= \mathrm{minsup} * D)$ /* Q是全局频繁项集*/

{$F = F \bigcup \{Q$ and any nonempty subset of $Q\}$;

P_0 deletes Q and any nonempty subset of Q from F' ;

}else

P_0 deletes Q from F';

}

for each itemsets $B \in$ local frequent 2-itemsets in F'

/* 根据底部策略剪枝F' */

{ P_0 broadcasts B;

P_i sends $B.si$ to P_0;

$B.s = \sum_{i=1}^{n} B.si$

if ($B.s >= \text{minsup} * D$)

{$F = F \bigcup \{B\}$;

P_0 deletes B from F'

}else

P_0 deletes B and superset of B from F';

/*从F'中删除B及B的超集*/

}

Step 3: /*计算剩余候选项集的全局频度*/

P_0 broadcasts the remain of F';

for each itemsets $d \in$ the remain of F'

{P_i sends $d.si$ to P_0;

$d.s = \sum_{i=1}^{n} d.si$;

}

Step 4: /*获得全局频繁项集 F*/

for each itemsets $d \in F'$

if ($d.s >= \text{minsup} * D$)

{$F = F \bigcup \{d\}$;

P_0 deletes d from F';

}else

P_0 deletes d from F'.

3.3 DMARF 算法实例

设有 3 个结点 P_1, P_2 和 P_3, 对应局部数据库 DB_1, DB_2和 DB_3. 各个结点的数据库如表 1 所示, 最小支持度阈值 $\text{minsup}=0.4$.

表 1　局部数据库 DB_1, DB_2 和 DB_3

Local database	ID	Transaction
DB_1	100	$a, b, c, k, m, f, e, l, p$
	101	c, k, b, m, o, q
	102	a, b, c, d, e
DB_2	200	b, f, h, j, q
	201	a, c, m, l, f
	202	c, r, s, t, q
DB_3	300	a, b, c, d, e, f
	301	b, c, d, k, q
	302	f, s, m, q, k

根据表 1 和 $\text{minsup}=0.4$, 可以得出全局频繁项目. 按支持度降序排序, 全局频繁项目组成的集合 $E=\{c, b, f, q, a, m, k\}$, 如表 2 所示.

结点 P_1, P_2和 P_3根据 E 构建 FP-tree$_1$, FP-tree$_2$和 FP-tree$_3$. 各个局部 FP-tree$_i$只包含全局频繁项目.

表 2　全局频繁项目及支持度

Global frequent items	Support count
c	7
b	6
f	5
q	5
a	4
m	4
k	4

结点 P_1利用 FP-tree$_1$, 采用 FP-growth 算法独立计算 DB_1的局部频繁项集

$$F_1 = \{\{c,b\},\{c,a\},\{b,a\},\{c,m\},\{b,m\},\{c,k\},\{b,k\},\{m,k\},\{c,b,a\},\{c,b,m\},\{c,b,k\},\{c,m,k\},\{c,b,m,k\}\}.$$

结点 P_2利用 FP-tree$_2$, 采用 FP-growth 算法独立计算 DB_2的局部频繁项集 $F_2=\varnothing$.

结点 P_3利用 FP-tree$_3$, 采用 FP-growth 算法独立计算 DB_3的局部频繁项集 $F_3=\{\{c,b\},\{q,k\}\}$.

中心结点 P_0 汇总得到所有结点局部频繁项集的并集

$$F' = \{\{c,b\},\{c,a\},\{b,a\},\{c,m\},\{b,m\},\{c,k\},\{b,k\},\{m,k\},\{q,k\},\{c,b,a\},\{c,b,m\},\{c,b,k\},\{c,m,k\},\{c,b,m,k\}\}.$$

采用顶部和底部策略对 F' 进行剪枝, 最终挖掘出所有的全局频繁项集 $F=\{\{c,b\},\{c,a\}\}$.

4　DMARF 算法比较实验

本文将 DMARF 算法与 CD 和 FDM 算法进行了比较, 利用 VC++ 6.0 实现了CD 和 FDM 算法. 测试环境为 100 M 局域网, 采用 5台联想 PC作为分布式结点, 各 PC机配置均为 P 4, 2.4 G, 内存 512 M, windows 2003 professional 操作系统. 1 台 DELL 工作站作为中心结点, 配置均为 P4,3.0 G, 内存 1GM, windows 2003 Server 操作系统. 实验数据来自某超市 2007 年 7 月的销售数据, 总数据量约为 25 000 条记录, 测试程序的编程语言为 VC++ 6.0, 消息传递库为标准 MPI. 采用固定结点数, 改变最小支持度方式进行测试. 将 DMARF 算法与 CD 和 FDM 算法在数据库扫描次数、通信量和执行时间等方面进行比较, 测试结果如图 1 ～图 3 所示.

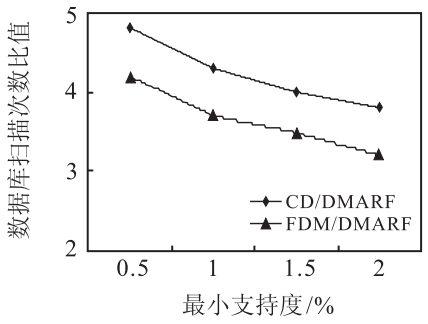

图 1 数据库扫描次数比较

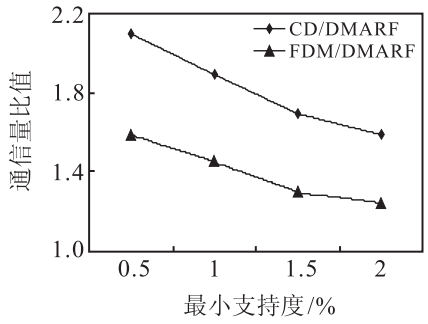

图 2 通信量比较

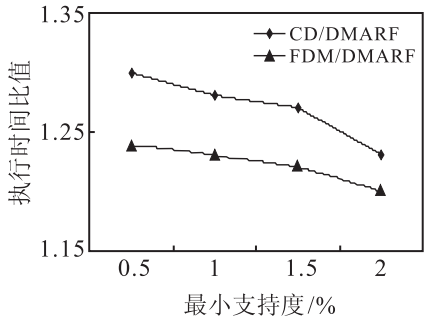

图 3 执行时间比较

由图1～图3可以看出, 在相同支持度下, 与CD和FDM等算法相比, DMARF算法的数据库扫描次数具有明显的优势, 通信量和执行时间都有所降低, 而且随着支持度的下降, 算法性能优势更加明显.

5 结 论

本文提出的DMARF算法采用顶部和底部策略, 有效地减少了候选项集, 降低了通信量. 实验结果表明, DMARF算法是有效可行的. 下一步的研究工作是在数据库动态改变的情况下如何实现全局频繁项集的快速更新.

参考文献(References)

[1] 陈志泊, 韩慧, 王建新. 数据仓库与数据挖掘[M]. 北京: 清华大学出版社, 2009: 3-9.
(Chen Z B, Han H, Wang J X. Data warehouse and data mining[M]. Beijing: Tsinghua University Press, 2009: 3-9.)

[2] Han J W, Pei J, Yin Y. Mining frequent patterns without Candidate Generation[C]. Proc of the 2000 ACM SIGMOD Int Conf on Management of Data. Now York: ACM Press, 2000: 1-12.

[3] Agrawal R, Shafer J C. Parallel mining of association rules[J]. IEEE Trans on Knowledge and Data Engineering, 1996, 8(6): 962-969.

[4] Cheung D W, Han J W, Ng W T, et al. A fast distributed algorithm for mining association rules[C]. Proc of IEEE 4th Int Conf on Management of Data. Miami Beach, 1996: 31-34.

[5] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]. Proc of the 20th Int Conf Very Large Database. Santiago, 1994: 487-499.

[6] Bayardo R J. Efficiently mining long patterns from databases[C]. Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM Press, 1998: 85-93.

[7] Schuster A, Wolff R. Communication-efficient distributed mining of association rules[C]. Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM Press, 2001: 473-584.

[8] Bo H, Peng T. Fast mining of maximum frequent itemsets in distributed multimedia database [C]. 2nd Workshop on Digital Media and Its Application in Museum and Heritage. Chongqing, 2007: 359-364.

[9] 何波, 王华秋, 刘贞, 等. 快速挖掘频繁项集的并行算法[J]. 计算机应用, 2006, 26(2): 391-392.
(He B, Wang H Q, Liu Z, et al. A fast and parallel algorithm for mining frequent itemsets[J]. J of Computer Applications, 2006, 26(2): 391-392.)