文章编号: 1001-0920(2011)08-1214-05

快速挖掘分布式数据库全局最大频繁项集

何 波

(重庆理工大学 计算机科学与工程学院，重庆 400054)

摘 要: 提出一种快速挖掘分布式数据库全局最大频繁项集算法(FMMFI). FMMFI 算法首先设置了中心节点, 并以各个节点构建局部 FP-tree, 采用挖掘最大频繁项目集算法(DMFIA)快速挖掘局部最大频繁项集; 然后与中心节点交互以实现数据汇总; 最终获得全局最大频繁项集. FMMFI 算法采用自上而下的剪枝策略, 能大幅减少候选项集, 降低通信量. 理论分析和实验结果表明, FMMFI 算法是有效的.

关键词: 数据挖掘；频繁模式树；全局最大频繁项集；分布式数据库

中图分类号: TP311 **文献标识码:** A

Fast mining of global maximum frequent itemsets in distributed database

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 an algorithm for fast mining global maximum frequent itemsets(FMMFI) in distributed database, which sets center node. FMMFI algorithm makes computer nodes compute local maximum frequent itemsets independently with discover maximum frequent itemsets algorithm(DMFIA) algorithm and local FP-tree. Then the center node collects data with other computer nodes and combines data. Finally, global maximum frequent itemsets are gained. FMMFI require far less candidate itemsets and communication traffic by using the strategy of top-down. Theoretical analysis and experimental results show the effectiveness of the FMMFI algorithm.

Key words: data mining；FP-tree；global maximum frequent itemsets；distributed database

1 引 言

数据挖掘[1]是从大量的、不完全的、有噪声的、模糊的、随机的数据集中, 提取有效的、新颖的、潜在有用的以及最终可理解的、模式的非平凡过程. 数据挖掘的一个重要研究课题是关联规则. 关联规则[1]是用来描述在给定的事务集中, 频繁出现的项集的规则. 关联规则挖掘的关键是获取频繁项集[2], 最典型的算法是 Apriori[1]. 许多数据挖掘应用只需要挖掘最大频繁项集, 而不用挖掘所有的频繁项集.

目前已提出了一些单机环境下挖掘最大频繁项集的算法, 如 Max-Miner[3], 快速开采最大频繁项目集算法(DMFI)[4]和挖掘最大频繁项目集算法(DMFIA)[5]. 然而, 实际中需要挖掘的往往是分布式数据库, 采用这些算法并不合适. 现有挖掘全局频繁项集的算法有高效并行关联规则挖掘算法(PDM)[6], 并行挖掘关联规则算法(CD)[7]和快速挖掘关联规则的分布式算法(FDM)[8]等, 也不适合用来挖掘全局最大频繁项集[9]. 目前挖掘全局最大频繁项集的算法很少, 鉴于此, 本文提出一种快速挖掘分布式数据库全局最大频繁项集算法(FMMFI). FMMFI 算法采用自上而下的剪枝策略, 能大幅减少候选项集并降低通信量. FMMFI 算法采用 FP-tree 结构, 与类 Apriori 算法相比, 能大大降低数据库扫描次数和运行时间.

2 相关描述和结论

2.1 挖掘全局最大频繁项集的问题描述

设全局事务数据库为 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节点(简称节点), 即它们之间除了通过网络传递信息外, 其他资源(如硬盘、内存等)全都是独立的. $\mathrm{DB}_i(i = 1, 2, \cdots, n)$ 是 DB 存储于节点 P_i 上的局部事

收稿日期: 2010-04-28；**修回日期:** 2010-07-15.

基金项目: 教育部科学研究基金项目((09yjc870032).

作者简介: 何波(1978−), 男, 副教授, 从事数据挖掘、智能信息服务等研究.

务数据库, 其中的事务有 D_i 条, 则

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

挖掘全局最大频繁项集的问题即是如何通过 n 台节点同时工作, 各台节点间通过网络传递有限的信息, 最终在整个事务数据库 DB 中挖掘出最大频繁项集.

2.2 相关定义

定义 1[2]　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[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[2]　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[2]　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　对于全局频繁项集 X, 如果 X 的所有超集都不是全局频繁项集, 则称 X 是全局最大频繁项集 FM.

定义 7　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 个全局频繁项目.

2.3 相关定理与推论

定理 1[2]　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[2]　若项集 X 为全局频繁项集, 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[10]　若项集 X 为全局频繁项集, 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[2]　若项集 X 是全局最大频繁项集, 则 X 必定是全局频繁项集.

定理 5[10]　若项集 X 是全局最大频繁项集, 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.

推论 3　若项集 X 不是任意一个局部数据库的局部频繁项集, 则 X 一定不是全局最大频繁项集.

定理 6[10]　若项集 X 是全局最大频繁项集, 则至少存在一个局部数据库 $\mathrm{DB}_i(i=1,2,\cdots,n)$, 使 X 是 DB_i 上某一局部最大频繁项集的子集.

推论 4　若项集 X 不是任意一个局部数据库的局部最大频繁项集的子集, 则 X 一定不是全局最大频繁项集.

定理 7　若项目 x_i 不是全局频繁项目, 则包含的 x_i 的项集一定不是全局频繁项集, 同时也不是全局最大频繁项集.

证明　若项目 x_i 不是全局频繁项目, 则 $\{x_i\}$ 不是全局频繁项集. 假设 $x_i\in X$, 有 $\{x_i\}\subseteq X$. 根据推论 2, 项集 X 不是全局频繁项集, 同时也不是全局最大频繁项集. □

2.4 FP-tree[9]

频繁模式树 (FP-tree) 是指满足以下 3 个条件的树型结构: 1) 它由一个标为 "null" 的根节点, 根节点的子项目前缀子树集合, 以及频繁项目头表组成. 2) 项目前缀子树中的每个节点均包含 3 个域: item-name, count, node-link. 其中: item-name 记录项目名, count 记录能到达该节点路径所表示的事务的数目, node-link 为指向 FP-tree 中具有相同的 item-name 值的下一节点. 当下一个节点不存在时, node-link 即为 null. 3) 频繁项目头表的每一表项包含 2 个域: item-name 和 head of node-link, 其中 head of node-link 为指向 FP-tree 中具有相同的 item-name 值的首节点指针.

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 的主要步骤是: 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 (该次序记为 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 FP-tree 中.

2.5 DMFIA 算法[5]

DMFIA 算法是单机环境下高效挖掘最大频繁项集的算法. DMFIA 采用最大频繁候选项集集合来保存自顶向下的信息, 并分批计算其中项集的支持度. 对于里面的频繁项集, 将其加入已有最大频繁项集集合中; 对于不频繁的项集, 则通过每次删除该项集中的一个项来产生新的候选. DMFIA 算法不是基于原始数据库, 而是基于 FP-tree 树进行支持度计算的. 整个算法初始化时将原始数据库压缩到一棵 FP-tree 中, 通过访问该树中相应的条件模式基计算支持度. 基于该 FP-tree 树, DMFIA 避免了对硬盘上原始数据库的

多遍扫描.

3 快速挖掘分布式数据库全局最大频繁项集算法(FMMFI)思想

FMMFI算法设置了中心节点P_0. 首先, 挖掘全局频繁项目E, 各节点根据E构建局部FP-tree$_i$; 其次, 各节点采用DMFIA[5]算法独立地计算局部最大频繁项集FM$_i$; 再次, 各节点向中心节点P_0发送其局部最大频繁项集FM$_i$, P_0汇总得到所有节点局部最大频繁项集的并集FM′$\left(\text{FM}' = \bigcup_{i=1}^{n}\text{FM}_i\right)$; 最后, 采用自上而下剪枝策略对FM′剪枝, 从FM′中挖掘出所有的全局最大频繁项集FM. 根据定理6, 全局最大频繁项集必定是某一局部最大频繁项集的子集, 各节点局部最大频繁项集的并集必定是全局最大频繁项集的超集. 因此, FMMFI算法能挖掘出所有的全局最大频繁项集. 设置中心节点汇总数据可避免一个局部最大频繁项集同时出现在多个节点而造成重复计算.

FMMFI算法首先要挖掘全局频繁项目. 各节点P_i扫描DB$_i$一次, 计算各局部项目的局部频度, 中心节点P_0从各个节点收集所有项目E_i的全局频度, 挖掘出全局频繁项目E. 中心节点P_0将按照支持度降序排序的全局频繁项目集合E发送给各个节点. 然后, 各节点P_i根据E构建局部FP-tree$_i$. 根据定理7, 若项目x_i不是全局频繁项目, 则包含的x_i的项集一定不是全局最大频繁项集. 构建的局部FP-tree不包含非全局频繁项目, 大幅减少了局部FP-tree中项目的数量, 提高了挖掘全局最大频繁项集的效率.

在FMMFI算法中, 每个节点采用DMFIA[5]算法独立地计算局部最大频繁项集. DMFIA算法采用FP-tree结构, 与类Apriori算法相比, 能大幅降低数据库扫描次数, 减少候选项集和运行时间. DMFIA算法能快速挖掘出局部最大频繁项集, 是单机环境下的高效算法.

FMMFI算法采用自上而下的剪枝策略, 可描述如下:

1) 确认局部最大频繁项集的并集FM′中所有项集的最大项数为k.

2) 从各个节点收集FM′中所有k-项集的全局频度.

3) 判断FM′中的所有k-项集, 如果k-项集Q是全局频繁项集, 则将Q加入全局最大频繁项集FM, 并将Q和Q的所有非空子集从FM′中删除; 否则, 将Q从FM′中删除, 并将Q的所有$k-1$项子集加入FM′. 再转1).

上述剪枝策略能有效减少候选项集, 降低节点间的通信量. 该自上而下的剪枝策略的详细描述见第4节FMMFI算法描述Step 4.

大多数分布式挖掘算法需要多次同步. 例如, 当频繁项集的最大长度为k时, 往往需要k次同步, 这样会造成各节点间的通信量较大, 挖掘效率降低. FMMFI算法中计算局部频繁项集的工作可以异步执行, 因此只需要同步两次即可, 大大减少了同步次数.

4 FMMFI描述

FMMFI算法步骤如下:

1) 挖掘全局频繁项目E, 各节点根据E构建局部FP-tree$_i$;

2) 各个节点利用局部FP-tree$_i$, 采用DMFIA[5]算法独立计算局部最大频繁项集FM$_i$;

3) 各节点向中心节点P_0发送其局部最大频繁项集FM$_i$, P_0汇总得到所有节点局部最大频繁项集的并集FM′$\left(\text{FM}' = \bigcup_{i=1}^{n}\text{FM}_i\right)$.

4) 采用自上而下剪枝策略对FM′剪枝, 从FM′中挖掘出所有全局最大频繁项集FM.

FMMFI算法描述如下:

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节点$P_i(i = 1,2,\cdots,n)$, DB$_i(i = 1,2,\cdots,n)$是DB存储于节点P_i上的局部事务数据库, 其中的事务有D_i条, 则有DB$=\bigcup_{i=1}^{n}\text{DB}_i$, $D=\sum_{i=1}^{n}D_i$. 设P_0为中心节点, 最小支持度阈值为minsup.

输出: 全局最大频繁项集FM.

Step 1 各个节点构建局部FP-tree$_i$.

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;
/* 按照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$_i$; /* 创建各节点的FP-tree$_i$ */

Step 2 各节点采用DMFIA算法独立计算局部最大频繁项集FM$_i$.

for ($i=1; i<=n; i$++)
FM$_i$ = DMFIA (FP-tree$_i$, minsup);

Step 3 P_0汇总得到所有节点局部最大频繁项

集的并集 FM′.

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

P_i sends FM_i to P_0;

P_0 combines FM_i and produces FM′;

Step 4　采用自上而下剪枝策略对 FM′ 剪枝, 挖掘出所有的全局最大频繁项集 FM.

FM= ∅; /* FM 存放全局最大频繁项集 */

while (FM′ ≠ ∅)

{P_0 confirms the largest size k of itemsets in FM′; /* 确认局 FM′ 中项集最大长度为 k */

for all itemsets $Q \in k$-itemsets in FM′

if (Q are not the subset of any itemsets in FM) /* Q 不是 FM 中任何项集的子集 */

{ P_0 broadcasts Q;

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

{P_i sends $Q.si$ to P_0; /* P_i 利用 FP-tree$_i$ 计算 Q 的局部频度 */

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

}

if $Q.s$ >=minsup$*D$ /* Q 是全局最大频繁项集 */

{ FM = FM$\bigcup${Q};

P_0 deletes Q and any nonempty subset of Q from FM′;

}

else /* Q 不是全局最大频繁项集 */

{P_0 deletes Q from FM′;

for all item $x \in Q$

if ($Q - \{x\}$ are not the subset of any itemsets in FM)

FM′ = FM′ $\bigcup$ {$Q - \{x\}$};

}

}

}.

5　FMMFI 实例

设有 3 个节点 P_1, P_2 和 P_3, 对应的局部数据库为 DB_1, DB_2 和 DB_3. 各节点的数据库如表 1 所示. 最小支持度阈值 minsup = 0.42.

根据表 1 和 minsup = 0.42, 可以得出全局频繁项目, 再按照支持度降序排序, 得到表 2.

全局频繁项目组成的集合 $E = \{c, b, f, q, a, m, k\}$. 节点 P_1, P_2 和 P_3 根据 E 构建的 FP-tree$_1$, FP-tree$_2$ 和 FP-tree$_3$ 分别如图 1～图 3 所示. 项目头表增加了全局频度域. 根据定理 7, 若项目 x_i 不是全局频繁项目, 则包含的 x_i 的项集一定不是全局最大频繁项集. 因此各个局部 FP-tree$_i$ 只包含全局频繁项目.

表 1　局部数据库 DB_1, DB_2 和 DB_3

局部数据库	编号	事务
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	f, h, j, q
	201	a, b, c, m, l, f, k
	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

表 2　全局频繁项目及支持度

全局频繁项目	支持数
c	7
b	6
f	5
q	5
a	4
m	4
k	4

节点 P_1 利用 FP-tree$_1$, 采用 DMFIA 算法独立计算 DB_1 的局部最大频繁项集 $\text{FM}_1 = \{\{c, b, a\}, \{c, b, m, k\}\}$.

节点 P_2 利用 FP-tree$_2$, 采用 DMFIA 算法独立计算 DB_2 的局部最大频繁项集 $\text{FM}_2 = \varnothing$.

节点 P_3 利用 FP-tree$_3$, 采用 DMFIA 算法独立计算 DB_3 的局部最大频繁项集 $\text{FM}_3 = \{\{c, b\}\}$.

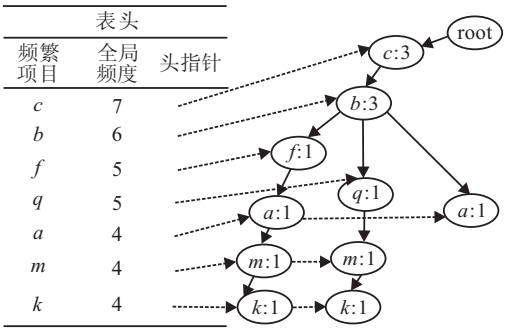

图 1　局部频繁模式树 FP-tree$_1$

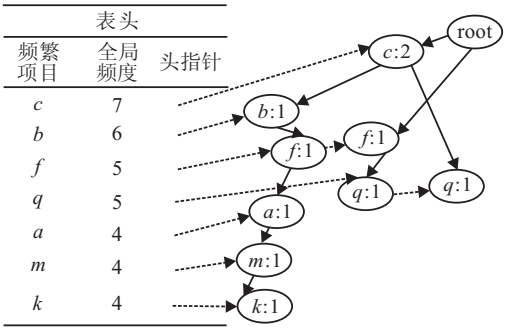

图 2　局部频繁模式树 FP-tree$_2$

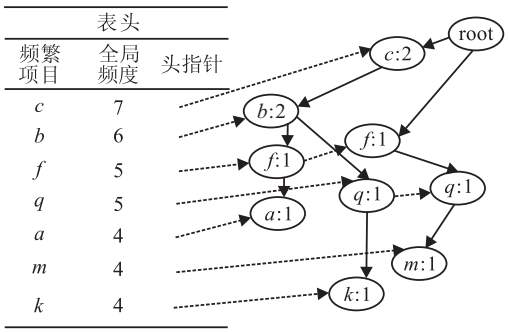

图 3 局部频繁模式树 FP-tree$_3$

中心节点 P_0 汇总得到所有节点局部最大频繁项集的并集 $\mathrm{FM}' = \{\{c,b,a\},\{c,b,m,k\},\{c,b\}\}$. 采用自上而下剪枝策略对 FM′ 剪枝, 最终挖掘出全局最大频繁项集 $\mathrm{FM} = \{\{c,b,a\},\{c,b,k\}\}$. 自上而下剪枝策略执行过程如表 3 所示. minsup = 0.42.

表 3 自上而下剪枝策略执行过程

FM′	k	判定Q
$\{\{c,b,m,k\},\{c,b,a\},\{c,b\}\}$	4	$\{c,b,m,k\}$
$\{\{c,b,a\},\{c,b,m\},\{c,b,k\},\{b,m,k\},\{c,b\}\}$	3	$\{c,b,a\}\surd$
$\{\{c,b,m\},\{c,b,k\},\{b,m,k\}\}$	3	$\{c,b,m\}$
$\{\{c,b,k\},\{b,m,k\},\{b,m\},\{c,m\}\}$	3	$\{c,b,k\}\surd$
$\{\{b,m,k\},\{b,m\},\{c,m\}\}$	3	$\{b,m,k\}$
$\{\{b,m\},\{c,m\},\{m,k\}\}$	2	$\{b,m\}$
$\{\{c,m\},\{m,k\}\}$	2	$\{c,m\}$
$\{\{m,k\}\}$	2	$\{m,k\}$

6 FMMFI 实验

为了测试算法的性能, 将 FMMFI 算法与分布式关联规则挖掘算法 CD 和 FDM 算法进行比较. CD 和 FDM 算法能够通过挖掘全局频繁项集找到全局最大频繁项集, 但主要是针对全局频繁项集长度相对较短的情况. CD 算法是基于 Apriori 算法的简单并行化, 每次扫描后需要同步, 节点间的通信量非常大. FDM 算法对 CD 算法进行了改进, 在各个节点上利用类 Apriori 算法挖掘出局部频繁项集, 各节点之间交换项集的支持度计数. 由于 FDM 算法频繁扫描数据库, 会产生较多的候选项集. FDM 算法没有设置中心节点, 使频繁项集同时出现在多个节点而造成重复计算. FMMFI 算法设置了中心节点, 采用 FP-tree 和自上而下的剪枝策略, 能快速挖掘出全局最大频繁项集.

测试环境为 100 M 局域网, 采用 5 台联想 PC 作为分布式节点, 各 PC 机配置均为 P4 2.4 GHz, 内存 512 M, Windows 2003 Professional 操作系统. 1 台 DELL 工作站作为中心节点, 配置均为 P4 3.0 GHz, 内存 1 GM, Windows 2003 Server 操作系统. 实验数据来自某超市 2007 年 7 月和 8 月的销售数据, 各个节点数据量约为 50 000 条记录, 测试程序的编程语言为 VC++6.0.

采用固定节点数、改变最小支持度的方式进行测试. FMMFI 算法与 CD 和 FDM 算法在通信量和执行时间等方面进行了比较, 如图 4, 图 5 所示.

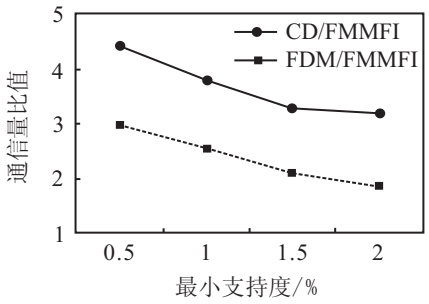

图 4 通信量比较

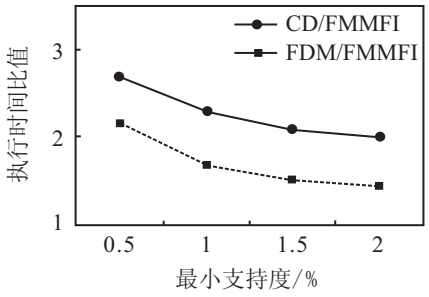

图 5 执行时间比较

实验结果表明, 在相同支持度下, 与 CD 和 FDM 等算法相比, FMMFI 算法在通信量上具有很大的优势; 在执行时间上优势也较为明显.

7 结 论

本文提出的 FMMFI 算法利用局部 FP-tree 快速挖掘局部最大频繁项集, 然后中心节点与各个节点交互, 采用自上而下的剪枝策略, 最终获得全局最大频繁项集. 实验结果表明 FMMFI 算法是有效且可行的. 下一步的研究任务是在数据库动态更新或支持度变化的情况下实现全局最大频繁项集的更新.

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: 5-8.
(Chen Z B, Han H, Wang J X. Data warehouse and data mining[M]. Beijing: Tsinghua University Press, 2009.)

[2] 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.)

(下转第1223页)