挖掘最大频繁项集的并行算法*)

李庆华[1] 王 卉[1,2] 蒋盛益[1]

(华中科技大学计算机学院 武汉430074)[1] (通信指挥学院 武汉430010)[2]

摘 要 频繁项集的挖掘是数据挖掘的核心内容。本文提出挖掘最大频繁项集的并行算法P-MinMax,它采用数据库的垂直表示和基于前缀关系的等价类划分,利用因子项集的完全包含关系在处理机之间贪心分配等价类,根据等价类的需要相应地划分和有选择地复制数据库记录,使各处理机得以异步计算,达到了较好的负载平衡。分析和实验表明,P-MinMax有较好的可扩展性,其性能优于已有同类算法。

关键词 频繁项集,并行算法,数据挖掘

Parallel Algorithm for Mining Maximal Frequent Itemsets

LI Qing-Hua[1] WANG Hui[1,2] JIANG Sheng-Yi[1]

(Computer School, Huazhong University, Wuhan 430074)[1] (College of Communication, Wuhan 430010)[2]

Abstract Mining frequent itemsets is a crucial issue in data mining applications. The complexity of the problem has been shown as NP-hard. Parallel techniques are widely used to improve the efficiency of mining algorithms. A novel and powerful parallel algorithm for mining maximal frequent itemsets, called P-MinMax, is proposed in this paper, which is based on its serial version MinMax. The new algorithm decomposes the search space by prefix-based equivalence classes, distributes work among the processors by complete inclusive relation between equivalence class gene itemsets and selectively duplicates databases in such a way that each processor can compute the frequent itemsets independently. These techniques eliminate the need for synchronization, drastically cutting down the I/O overhead. The analysis and experimental results demonstrate the superb efficiency of the approach in comparison with the previous work.

Keywords Frequent itemsets, Parallel algorithm, Data mining

1 引言

频繁项集的挖掘是关联规则挖掘和序列挖掘等诸多数据挖掘的核心内容,它耗费数据挖掘算法绝大部分时间,其复杂性最坏情况下可以是数据库项数的指数函数,因此,有许多研究致力于缩短挖掘频繁项集的时间[1~3]。

由于数据库的规模不断扩大,并行技术被用于提高挖掘频繁项集的效率[4~6]。R. Agrawal 最早在文[4]中提出了Apriori的三种并行算法CD,DD和CaD,代表了数据库水平表示下的并行化实现策略。后续的许多并行算法都是在此基础上的改进。计数分布算法CD是Apriori的一个直接的并行化实现。该算法将数据库在各处理机之间大致相等地划分,将候选集在各处理机上完整复制,各处理机计算完整候选集的局部支持度,在每步迭代结束时,同步、汇总,产生全局支持度。同步是CD算法的一个明显的缺点。针对这一缺点,数据分布算法DD将候选集划分成互不相交的子集分配给各处理机,各处理机为其候选子集计算全局支持度,为此需要访问全局数据库,造成巨大的数据通信量。候选分布算法CaD采用选择性地复制数据库,克服了DD算法通信量大的问题,但由于其串行算法本身固有的特性,每次迭代都需要扫描局部数据库,因而算法效率不高。Zaki在文[5]提出了四种并行算法:ParEclat, ParMaxEclat, ParClique和ParMaxClique。这四种算法采用了相似的并行化策略,不同的搜索策略和等价类划分技术,其数据库采用垂直表示形式。由于在调度等价类时没有考虑剪枝的因素,未能发挥剪枝的作用;在划分数据库时没有针对等价类对局部数据库的需要,导致了不必要的数据库记录复制;以等价类的长度作为权值,不能较好地度量搜索的代价,因而,没有得到好的负载平衡。

基于此,本文提出挖掘最大频繁项集的算法MinMax[3]的一个并行化实现,称之为P-MinMax。P-MinMax采用数据库的垂直表示和基于前缀关系的等价类划分,以等价类长度的指数函数为权值,并利用因子项集的完全包含关系在处理机之间贪心分配等价类,根据等价类的需要相应地划分和复制数据库记录,使各处理机得以异步计算,达到了较好的负载平衡、较高的剪枝效率和较少的数据库记录复制,减少了算法的执行时间。分析和实验表明,P-MinMax有较好的可扩展性,其性能优于已有同类算法。

2 基本概念与有关约定

设$T=\{i_1,i_2,\cdots,i_m\}$是一个由m个不同项组成的集合,$T=\{t_1,t_2,\cdots,t_n\}$是n个交易的集合,对于$x\in T$,$tid_list(x)=\{t_i|t_i\in T$且$t_i$中含有$x\}$为项$x$参加的所有交易的集合,于是$D=\{tid_list(x)|x\in T\}$是一个垂直表示的交易数据库。对于$x\in T$,称$f(x)=|tid_list(x)|$为项$x$的支持度。称$X=\{x_1,x_2,\cdots,x_k\}(x_i\in T,i=1,2,\cdots,k)$为长度为$k$的项集,简称$k$项

*)基金项目:国家自然科学基金(60273075)。李庆华 教授,博导,主要研究领域为数据挖掘和并行算法等。王 卉 博士研究生,副教授,主要研究领域为数据挖掘和并行算法等。蒋盛益 博士研究生,副教授,主要研究领域为数据挖掘和并行算法等。

集。称 $f(X)=|\bigcap_{x_i\in X} tid\_list(x_i)|$ 为项集 X 的支持度。设 ε 是用户指定的最小频繁阈值，称满足 $f(X)>=\varepsilon$ 的项集为频繁项集，频繁的1项集为频繁项。D 中所有频繁 k 项集的集合记做 F_k，所有频繁项集的集合记做 F，$F=\bigcup_{k\geq 1} F_k$。称满足 $X\in F\wedge(\neg\exists\ Y(Y\supset X)\wedge Y\in F)$ 的项集 X 为最大频繁项集。所有最大频繁项集的集合记做 M。于是有 $M\subseteq F$ 且有如下性质成立：

性质1 $\forall\ X(X\in F)\rightarrow(Y\subset X\wedge Y\neq\varnothing\rightarrow Y\in F)$

性质2 $\forall\ X(X\notin F)\rightarrow(\forall\ Y(Y\supset X)\rightarrow Y\notin F)$

性质3 $\forall\ X(X\in M)\rightarrow|\{Y|\ Y\subset X\wedge Y\neq\varnothing\wedge Y\in F\}|=2^{|X|}-2$

由此可知，$|M|<<|F|$。

频繁项集的挖掘可以看作一个搜索问题。搜索空间中的每一个点由 head 和 tail 两部分组成，表示为 node(head，tail)，node 为节点名，head 是在该节点上已经确定的频繁项集，tail 是对 head 可能的扩展。在初始状态下，根节点是唯一的节点，其 head=$\varnothing$，tail=F_1。设 node.tail=$\{a_1,a_2,\cdots,a_n\}$，node.head$\cup\{a_i\}$($i=1,2,\cdots,n$)是频繁的，那么，节点 node(head，tail)可扩展的下一层节点有 nextlevel(node.head$\cup\{a_i\}$，$\{a_{i+1},a_{i+2},\cdots,a_n\}$)($i=1,2,\cdots,n-1$)以及 nextlevel(node.head$\cup\{a_n\}$，$\varnothing$)。Tail=$\varnothing$的节点为叶节点。遍历整个搜索空间得到的所有节点的 head 的集合是 F。

为叙述方便，本文提出以下定义和定理。

定义1(频繁项的不频繁能力) 频繁项 x 的不频繁能力 $g(x)$定义为项 x 构成的不频繁2项集的个数：$g(x)=|y|x\in F_1\wedge y\in F_1\wedge f(\{x,y\})<\varepsilon|$ 。

由定义1可知，如果 $g(x1)>g(x2)$，那么，$x1$比 $x2$造成更多的不频繁项集。

定理1 如果 $f(x1)<f(x2)$，那么，$x1$比 $x2$造成更多的不频繁项集。

证明：设 $t\in T,x1,x2,x,y\in I$，设 $p(x)$是 $t\in tid\_list(x)$ 的概率，$p(\{x,y\})$是 $t\in tid_list(x)\cap tid_list(y)$的概率。由于 x 和 y 是独立的，因此 $p(\{x,y\})=p(x)*p(y)$。由 $f(x1)<f(x2)$，可知 $p(x1)<p(x2)$，因此 $p(\{x1,y\})<p(\{x2,y\})$，于是 $f(\{x1,y\})<f(\{x2,y\})$，$x1$比 $x2$造成更多的不频繁2项集。因此，$x1$比 $x2$造成更多的不频繁项集。证毕。

定理2 设 p 为以 node(head，tail)为根节点的子树中的一个最大频繁项集，如果 p=node.head$\cup$node.tail，则该子树中所有的节点皆为频繁项集。

证明：根据扩展的过程可知，对于以 node(head，tail)为根节点的子树中的任意节点 son_node(head，tail)，有 son_node.head$\cup$son_node.tailnode.head$\cup$node.tail 成立。

由于 p=node.head$\cup$node.tail，故有 son_node.head$\cup$son_node.tailp 成立。由于 p 是频繁的，据性质1，son_node(head，tail)频繁。证毕。

定义2(前缀函数) 定义一个函数 $h:p(I)\times N\mapsto p(I)$，$p(I)$为 I 的子集的集合，$h(x,k)=x[1:k]$为 x 的长度为 k 的前缀。

定义3(等价类$[x]_{\theta_k}$) 定义 $p(I)$上的等价关系 θ_k 为：$\forall\ X,Y\in p(I),X\theta_kY\Leftrightarrow h(X,k)=h(Y,k)$。称 θ_k 为基于 k 前缀的等价关系。满足等价关系 θ_k 的任意元素 $x\in p(I)$的等价类记做$[x]_{\theta_k}=\{y|y\in p(I)\wedge x\theta_k y\}$。

定义4(等价类的权) 设$[x]$是 F_2上的基于 θ_1的一个等价类，等价类$[x]$的权定义为：$weight([x])=2^{|[x]|}$。

定义5(等价类的因子项集) 设$[x]=\{xy_1,xy_2,\cdots,xy_m\}$，等价类的因子项集定义为：$[x]^+=\{x,y_1,y_2,\cdots,y_m\}$。

定义6(处理机的负载) 设$[x_i](i=1,2,\cdots,m_i)$是处理机 P 上的全部等价类，处理机 P 的负载定义为：$workload(P)=\sum_{i=1}^{m_i} weight([x_i])$ 。

3 串行算法 MinMax

MinMax[3]对以($\varnothing$，F_1)为根节点的树进行深度优先搜索，返回 M。其基本思想是：尽早地找到最大频繁项集，并用它们来剪去其子集。MinMax 用一个栈 s 来保持搜索空间，根据定理2执行多步回退剪枝。栈记录的结构设为：(head，tail，flagbits)，flagbits 中的每一位对应于 tail 的每一项，0表示对应项尚未扩展，1表示对应项已经被扩展。栈的初始状态为：($head_0$，$tail_0$，00……0)，$head_0=\varnothing$，$tail_0=F_1$，结束状态为空。在初始状态下，根据定义1和定理1对 $tail_0$按 $g\downarrow f\uparrow$ 进行排序。这样，可以先扩展 tail 较小的节点，尽快达到一个最大频繁项集。

算法描述如下：

```
MinMax(head0,tail0,M)
/* MinMax 输出最大频繁项集的集合 M,其输入为 head0=∅,tail0
   =F1,tail0按 g↓f↑排序 */
1. while 栈 s 非空 do
2. if s.flagbits 各位非全1
3. then /* 准备产生新节点 */
   选取 s.tail 最左边的 s.flagbits 对应为0的项 ai;
4. newnode.head←s.head∪{ai};
5. newnode.tail:={y|y∈s.tail∧y>ai∧f(newnode.head∪{y})>
   ε};
6. if ∃ p∈M (M⊇newnode.head∪newnode.tail)
7. then 给 s.flagbits 最左边的0位置1/* 新节点直接被剪枝 */
8. else if newnode.tail==∅/* 新节点是叶节点吗? */
9. then M:=M∪newnode.head;
10. while s.head∪s.tail==newnode.head do pop ;/* 多步回退剪
    枝 */
11. if 栈非空 then 给 s.flagbits 最左边的0位置1
12. else /* 新节点仍需扩展 */
    push(newnode.head,newnode.tail,00……0)
13. endif
14. endif
15. else {pop; if 栈非空 then 给 s.flagbits 最左边的0位置1}
16. endif
17. endwhile
18. return M
```

4 并行算法 P-MinMax

挖掘最大频繁项集的并行算法涉及项集的划分与调度和数据库的划分。算法 P-MinMax 按照 F_2上的等价关系 θ_1将项集划分成互不相交的等价类，分布到各处理机上，并将数据库有选择性地复制到相应的处理机上，各处理机异步执行 MinMax。汇总各处理机的计算结果，去除那些非全局最大的局部最大频繁项集，即为结果。

4.1 任务划分策略

并行挖掘最大频繁项集的任务划分问题也就是等价类的分配问题。P-MinMax 将等价类$[x]$按其长度$|[x]|$递减排序。以$2^{|[x]|}$为权值在各处理机之间按贪心算法分配等价类。在考虑当前等价类$[x]$时，依次在各处理机对应的等价类序列中，检查是否存在等价类$[y]$，其因子项集$[y]^+$完全包含等价类$[x]$的因子项集$[x]^+$。设等价类$[y]$已分配给处理机 P，且有$[x]^+\subset[y]^+$，则将等价类$[x]$也分配给处理机 P。否则，将等价类$[x]$分配给当前 workload(P_i)最小的处理机 P_i。将这种有完全包含关系的等价类分配到同一个处理机的好处有两点：第一，如果$[x]$的搜索树中有任何节点包含于$[y]$的某个

最大频繁项集，即可剪去该节点而无需继续搜索，这样就增加了剪枝的机会；反之，如果[x]中的一个节点被[y]的最大频繁项集所包含但[x]和[y]不在同一个处理机上，则会损失一次剪枝的机会。第二，由于[x]所需要的数据库记录都是[y]所需要的，因此，可以不必为[x]复制数据库记录，于是就减少了数据库记录的多余复制。

4.2 数据划分策略

P-MinMax 先分配等价类，再根据各处理机上等价类的需要，复制相应的数据库记录。这种数据划分方案较之独立于等价类的数据划分方案，可以使所有的数据记录都为该处理机上的等价类所需。设处理机 P 上的等价类[x]的因子项集 $[x]^+$ 中含有项 x_i，若处理机 P 上没有数据记录 $tid\_list(x_i)$，则复制该记录到 P 上。

4.3 并行算法描述

P-MinMax 包括初始化、异步计算和后处理三个阶段。

(1) 初始化：计算 F_1 和 F_2，得到各频繁项的 $f(x)$ 和 $g(x)$，按照 θ_1 将 F_2 划分成等价类，按4.1节分配等价类，按4.2节划分并选择性地复制数据库记录，为各处理机异步执行 MinMax 作准备。

(2) 异步计算：各处理机逐个处理等价类[x]：异步执行 MinMax($\{x\}$,$[x]^+-\{x\}$,M_i)，M_i 为处理机 P_i 的计算结果。

(3)后处理：汇总各处理机的结果 M_i，得到总的结果 M。

4.4 举例

这里举一个例子来说明并行算法 P-Minmax 的等价类分配和数据库的划分，以及各等价类对应的访问节点和剪枝情况。设有如下等价类因子项集：$[1]^+=\{1,2,3,4,5,6,7\}$，$[2]^+=\{2,5,6\}$，$[3]^+=\{3,4,7\}$，$[4]^+=\{4,5,6,7,8,9\}$，$[5]^+=\{5,6,7,8\}$，$[6]^+=\{6,7,8\}$，$[7]^+=\{7,8,9\}$。各等价类对应的搜索树及剪枝情况如图1所示。其中带×标志的为被剪掉的节点，其下方的等价类标志表示该节点被相应等价类剪掉。有下划线的节点表示最大频繁项集。设有 $P1$和 $P2$两部处理机，表1给出了不同的等价类分配方案下各处理机访问节点的总数以及损失的剪枝次数。假定在独立于等价类划分数据库的方案下，记录 $tid\_list(1)$，$tid\_list(3)$，$tid\_list(5)$，$tid_list(7)$和 $tid_list(9)$在处理机 $P1$上，$tid\_list(2)$，$tid\_list(4)$，$tid_list(6)$和 $tid_list(8)$在处理机 $P2$上，表2给出了不同等价类分配方案下数据库记录的冗余情况。可以看出，按$2^{|[x]|}$及包含关系分配等价类时剪枝状况和数据库记录的冗余状况优于其余两种分配方案，负载平衡状况也较好。

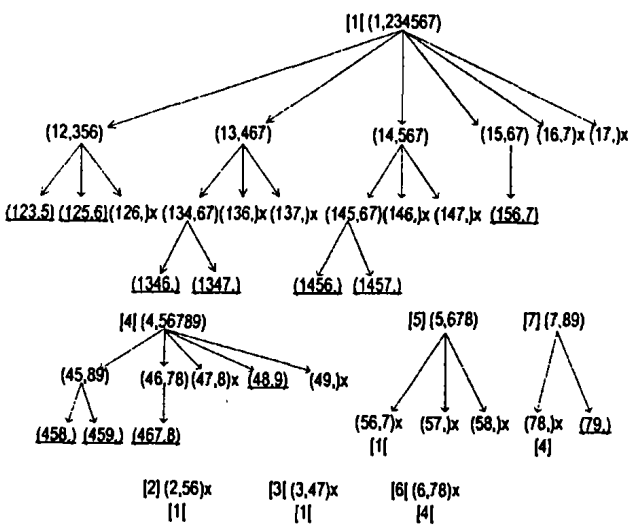

图1 等价类的搜索及剪枝图

表1 等价类的分配及访问节点数和剪枝损失情况

等价类分配方案	处理机	等价类序列	访问节点数	损失的剪枝次数
按|[x]|分配	P1	[1][2][3][7]	21+1+1+3=26	2
	P2	[4][5][6]	9+4+1=14	
按$2^{\|[x]\|}$分配	P1	[1]	21	3
	P2	[4][5][2][3][6][7]	9+4+2+2+1+3=21	
按$2^{\|[x]\|}$及包含关系分配	P1	[1][2][3]	21+1+1=23	1
	P2	[4][5][6][7]	9+4+1+3=17	

表2 不同的等价类分配方案下数据库冗余情况

等价类分配方案	独立于等价类划分数据库时的冗余记录数	根据等价类需要划分数据库时的冗余记录数
按|[x]|分配	6	6
按$2^{\|[x]\|}$分配	7	6
按$2^{\|[x]\|}$及包含关系分配	6	4

5 实验结果

我们在曙光3000并行计算机上对算法进行了试验。有关的配置情况为：3个节点由内部通信网高速互联，每个节点2GB 内存和9GB 硬盘由4个主频为375MHz 的处理机共享。实验数据采用合成数据库 T10.I4.D2084K，T15.I4.D1471K 和 T20.I6.D1137K。其中，T 表示平均交易长度，I 表示最大频繁项集的平均长度，D 表示数据库中交易的数目，取支持度阈值 $\epsilon=|D|*0.25\%$，数据库中频繁项集的分布如图2所示。算法在不同处理机个数的情况下的执行时间如图3所示。固定处理机个数3*1，即配置为3个节点，每个节点1个处理机，当数据库中交易的数目乘 r 倍变化时，算法执行时间的变化如图4所示。实验结果显示，算法 P-MinMax 具有较好的可扩展性。

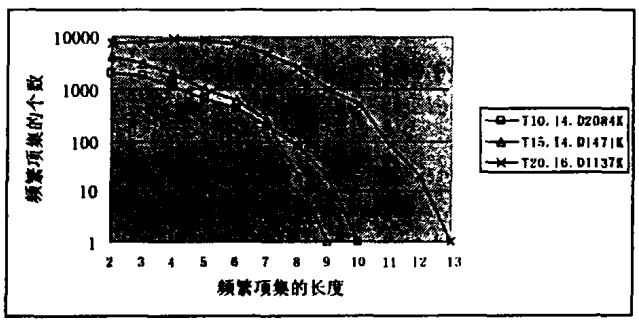

图2 数据库中频繁项集的分布 $\epsilon=|D|*0.25\%$

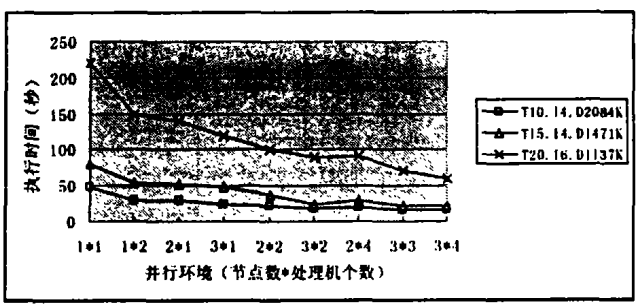

图3 数据库规模固定时，算法在不同处理机配置下的执行时间

（下转第188页）

833MHz 的 CPU、1G 内存，16K L1 D cache 和 16K L1 I cache。

优化的 gp 指令数——对 SPEC2000 基准程序的12定点程序的测试从图2中可以看出，ORC 编译器使用 O3 优化级别，经过程序调用共享连接技术优化后，对 gp 寄存器的保存和恢复指令数比原始数据都有不同程度的减少。

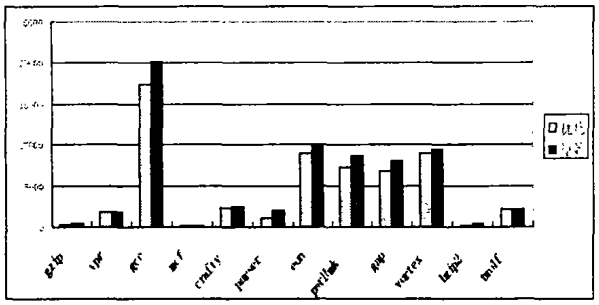

图2 程序共享连接技术优化的指令数

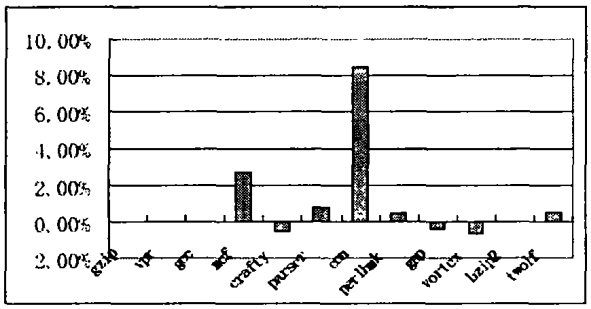

图3 程序共享连接技术性能加速比

O3 优化的性能改进——相对于图2来说，图3显示了这些定点程序经过 ORC 编译器在 O3 优化级别编译生成的二进制代码在 ITANIUM2计算机上测试运行的性能，这些程序的性能多数变好，最好的改善达到8.5%，一些程序的性能没有变化，而有少数程序的性能会有微弱的下降，这是因为一方面跟计算机运行的不稳定性有关，另一方面汇编代码的指令数的减少如果在比较小的基本块里，会导致跳转指令(br)的距离变近，影响了 CPU 的指令跳转预测，同时这里的指令减少又没有减少机器周期数，我们考虑到程序的总体性能有明显的改进，平均性能加速比达到1 个百分点，这对作为编译器的一个优化来说是非常可观的，特别对一个先进的编译器来讲，各种优化方法已经有了相当充分的研究。

在程序调用共享连接技术下，我们对函数符号属性的设置从而达到优化编译单元的冗余代码的目的，我们知道一个程序的符号表包括函数符号和数据符号，我们的研究计划一方面是研究函数调用的冗余代码删除，另一方面就是对数据变量的研究，因为考虑到如果一个全局数据变量符号是不可抢占受保护的，那么我们可以改善变量的布局，优化变量的存取，改进存储优化，对存储的优化是我们很感兴趣的方向，数据变量的布局优化对程序的性能的提升将有更大的帮助。

参 考 文 献

1 Intel Corporation. Intel® IA-64 Architecture Software Developer's Manual, Jan. 2000

2 Levine J R. Linkers and Loaders. Morgan Kaufmann Publishers, 2000

3 Intel Corporation. IA-64 Software Conventions and Runtime Architecture Guide, Jan. 2000

4 Intel Corporation. UNIX System V Application Binary Interface, Jan. 2000

5 Intel Corporation. Intel® Itanium®2 Processor Reference Manual, June 2002

6 Linux 下的动态连接库及其实现机制. http://www.antpower.org/

7 Aho A V, Sethi R, Ullman J D. Compilers: Principles, Techinques, and Tools. Addison Wesley, Pearson Education, 1986

(上接第134页)

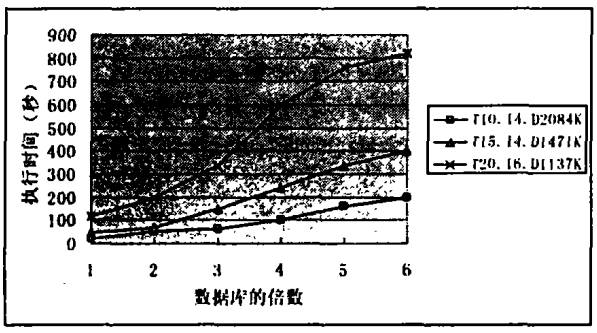

图4 处理机配置固定时，在不同数据库规模下算法的执行时间

结束语 本文提出了一个并行挖掘最大频繁项集的新算法 P-MinMax，它采用数据库的垂直表示和基于前缀关系的等价类划分，以等价类长度的指数函数为权值，并利用因子项集的完全包含关系在处理机之间贪心分配等价类，根据等价类的需要相应地划分和复制数据库记录，使各处理机得以异步计算，达到了较好的负载平衡、较高的剪枝效率和较少的数据库记录复制，减少了算法的执行时间。分析和实验表明，P-MinMax 有较好的可扩展性，其性能优于已有同类算法。

参 考 文 献

1 Burdick D, Calimlim M, Gehrke J. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In: Proc. of 17th Intl. Conf. on Data Engineering, Heidelberg, Germany, April 2001. 443～452

2 Gouda K, Zaki M J. Efficiently Mining Maximal Frequent Itemsets. In: Proc. of 2001 IEEE Intl. Conf. on Data Mining (ICDM'01), San Jose, California, November 2001. 163～170

3 Wang Hui, Li Qinghua, Ma Chuanxiang, Li Kenli. A Maximal Frequent Itemset Algorithm. Lecture Notes in Computer Science, Springer, 2003, 2639: 484～490

4 Agrawal R, Shafer J C. Parallel Mining of Association Rules. IEEE Transaction On Knowledge And Data Engineering, Dec. 1996, 8(6): 962～969

5 Zaki M J, Parthasarathy S, Ogihara M, Li Wei. New Parallel Algorithms for Fast Discovery of Association Rules. Data Mining and Knowledge Discovery: An International Journal. special issue on Scalable High-Performance Computing for KDD, Dec. 1997, 1(4): 343～373

6 Zaki M J. Parallel and Distributed Association Mining: A Survey. IEEE Concurrency, 1999, 7(4): 14～25