关联规则的开采与更新*

周海岩

(太原师范专科学校 太原 030001)

摘要 对关联规则的增量式更新算法——IUA(incremental updating algorithm)进行了分析,指出其存在的问题,并给出一个改进的算法 NEWIUA(new IUA).NEWIUA 算法对已存在的和本次更新时新产生的频繁项目集都加以充分的利用,因此,在保证算法有效的同时提高了效率.文章提出了 3 种关联规则开采的并行算法,并对各算法进行了分析和讨论.

关 键 词 数据开采,关联规则,增量式更新,频繁项目集,并行算法.

中图法分类号 TP311

数据开采(data mining)是数据库中的知识发现 KDD(knowledge discovery in databases)的核心步骤,是一种从大规模数据库或数据仓库中提取隐藏的预测性信息的新方法.此方法的提出让人们有能力最终认识数据的真正价值,即数据中潜在的可用信息和知识.数据开采是目前在国际上数据库、数据仓库和信息决策领域最前沿的方向之一,引起了国内外学术界和工商业界的广泛关注.国际上许多研究机构和实验室都在这个领域开展了各种各样的研究.研究的主要目标是发展有关的方法论、理论工具,以支持从大量数据中提取有用的和让人感兴趣的知识、模式和规则.

在数据开采的研究领域,对于关联规则开采的研究开展得比较积极和深入.关于关联规则开采的一般对象、主要应用领域及关联规则在实际问题中的含义在文献[1~3]中均有详细叙述,在此不再重述.本文第 1 节给出了问题的详细描述.第 2 节指出了 IUA(incremental updating algorithm)算法存在的问题,对 IUA 算法的效率进行了分析,并给出了一种改进算法.第 3 节提出了 3 种关联规则开采的并行算法,并进行了相应的分析与讨论.第 4 节作出总结.

1 问题描述

为了使行文方便和完整,将关联规则开采问题的形式化描述如下[1,2].

设 $I=\{i_1,i_2,\ldots,i_m\}$ 是 m 个不同项目的集合,给定一个事务数据库 D,其中每一个交易 T 是 I 中一些项目的集合,即 $T\subset I$.每一个交易 T 都与一个唯一的标识符 TID 相联.如果对于 I 中的一个子集 X,有 $X\subset T$,我们就说一个交易 T 包含 X.一条关联规则(association rule)就是一个形如 $X\Rightarrow Y$ 的蕴涵式,其中 $X,Y\subset I$,而且 $X\cap Y=\emptyset$.X 称作规则的前提,Y 是结果.

一般把一些项目的集合称作项目集.在一个项目集中所含项目的个数称为该项目集的长度,即 X 为项目集,X 的长度 $|X|=X$ 中项目的个数.对于 $X\subset I$,如果 D 中包含 X 的交易数目为 s,则称 s 为 X 的支持度.若用 $Support(X)$ 表示 X 的支持度,则 $Support(X)=s$.关联规则 $X\Rightarrow Y$ 的支持度定义为 $Support(X\Rightarrow Y)=Support(X\cup Y)$,而一个关联规则也有其衡量标准,称其为"置信度"(confidence),定义为 $Confidence(X\Rightarrow Y)=Support(X\cup Y)/Support(X)$.关联规则的开采问题就是在 D 中筛选出所有具有用户指定的最小支持度和最小可信度的关联规则,即这些关联规则的支持度和可信度分别不小于最小支持度和最小可信度.

---

* 本文研究得到国家自然科学基金和太原师范专科学校科研基金资助.作者周海岩,1957 年生,讲师,主要研究领域为数据开采,信息存储,计算机算法,数据库理论.

本文通讯联系人:周海岩,太原 030001,太原师范专科学校计算机教研室

本文 1998-08-10 收到原稿,1998-10-14 收到修改稿

关联规则的开采问题可以分解为以下两个子问题:

① 筛选出事务数据库 D 中所有具有用户指定的最小支持度的项目集(itemset,I 的非空子集).具有最小支持度的项目集称为频繁项目集,反之就称为非频繁项目集.

② 利用频繁项目集生成所有的关联规则.对每一个频繁项目集 A,找出 A 的所有非空子集 a,如果比率 $Support(A)/support(a)>minconf$,就生成关联规则 $a\Rightarrow(A-a)$.其中 $minconf$ 为用户指定的最小可信度.

由于子问题②相对来说较为容易,因而目前研究的重点集中在第①个子问题上.

2 关于关联规则的增量式更新算法

在关联规则开采的过程中,为了发现事先未知的关联规则,用户必然需要通过对最小支持度和最小可信度这两个阈值的不断调整来逐步聚焦到那些真正令其感兴趣的关联规则上去,这将是一个动态的交互过程.因此,迫切需要高效的更新算法来满足用户对较快的响应时间的需求.

传统的关联规则的更新问题考虑的是,当基础数据库或数据仓库中数据发生变化时,如何高效地进行规则更新而勿需从头做起,而本文及文献[1]所考虑的是数据库 D 中数据保持不变,当用户所给定的阈值(主要指支持度)改变时关联规则的更新问题.

由于篇幅所限,在此对关联规则增量式更新算法 IUA 不作详细介绍,详情见文献[1],并且下文中所用符号及其含义除非特别说明,均与文献[1]相同.

2.1 IUA算法存在的问题

IUA 算法与 Apriori 算法(见文献[3])的主要区别在于 iua_gen 函数,但是在 iua_gen 函数的第 2 步修剪中,会将 C_k^3 中由 L_1 中的 $k-1$ 个项目与 L_1'' 中的一个项目所构成的 k-项目集和由 L_1 中的一个项目与 L_1'' 中的 $k-1$ 个项目所构成的 k-项目集全部修剪掉.这样导致关联规则在更新时会使一些频繁项目集以及一些有效规则开采不出来.

例:设数据库 D 有 4 次交易为 $T_1=\{A,B,C\}$,$T_2=\{A,B,D\}$,$T_3=\{A,D,E\}$,$T_4=\{A,B,C,D\}$.设旧的最小支持度 $s=3$,与之对应的所有频繁 k-项目集(即长度为 k 的频繁项目集)所构成的集合记为 $L_k(k=1,2,\ldots,m_1)$,这里 m_1 为所有频繁项目集中的最大长度,于是 $L_1=\{\{A\},\{B\},\{D\}\}$,$L_2=\{\{AB\},\{AD\}\}$,此时 $m_1=2$.

设新的最小支持度 $s'=2$.用 IUA 算法来进行关联规则更新得:

$L_1=\{\{A\},\{B\},\{D\}\}$, $L_1''=\{\{C\}\}$, $L_1^1=L_1$, $L_1^2=L_1''$, $L_1^3=\varnothing$, $L_1'=L_1\cup L_1''$, $C_2^1=apriori\_gen(L_{k-1}^1)-L_k=apriori\_gen(L_1^1)-L_2=\{\{BD\}\}$(此处 $k=2$),$C_2^2=apriori\_gen(L_1^2)=\varnothing$.

执行 iua_gen 函数来求 C_2^3,执行 iua_gen 函数的第 1 步拼接得 $C_2^3=\{\{AC\},\{BC\},\{DC\}\}$,执行 iua_gen 函数的第 2 步修剪,取 $\{AC\}\in C_2^3$,因为 $\{A\}$ 是 $\{AC\}$ 的 $k-1=1$ 个元素的子集,但 $\{A\}\in L_1^3=\varnothing$,则 $\{AC\}$ 在 C_2^3 中被删除,同样地,$\{BC\}$,$\{CD\}$ 通过 iua_gen 函数的修剪都将在 C_2^3 中被删除,最后得 $C_2^3=\varnothing$.于是得 $L_2^0=\{\{BD\}\}$,从而 $L_2^1=L_2^0\cup L_2=\{\{AB\},\{AD\},\{BD\}\}$,因为 $C_2^2=\varnothing$,$C_2^3=\varnothing$,则 $L_2^2=$,$L_2^3=\varnothing$,从而得 $L_2'=L_2^1\cup L_2^2\cup L_2^3=\{\{AB\},\{AD\},\{BD\}\}$,$C_3^1=apriori\_gen(L_2^1)-L_3=\{\{ABD\}\}$($L_3=\varnothing$),$C_3^2=apriori\_gen(L_2^2)=\varnothing$(因为 $L_2^2=\varnothing$).

执行 iua_gen 函数第 1 步拼接得 $C_3^3=\{\{ABC\},\{ADC\},\{BCD\}\}$,经 iua_gen 函数的修剪最后得 $C_3^3=\varnothing$(因为 $L_2^3=\varnothing$,任取 $c\in C_3^3$,则 c 的任意一个有两个项目的子集 c' 均有 $c'\in L_2^3$ 故应在 C_3^3 中将 c 删除),于是得 $L_3^0=\{\{ABD\}\}$,从而 $L_3^1=L_3^0\cup L_3=\{\{ABD\}\}$,$L_3^2=\varnothing$(因为 $C_3^2=\varnothing$),$L_3^3=\varnothing$($C_3^3=\varnothing$),所以 $L_3'=L_3^1\cup L_3^2\cup L_3^3=L_3^1=\{\{ABD\}\}$.

这样,由 IUA 算法所得到的数据库 D 在最小支持度 $s'=2$ 下的所有频繁项目集为 $\{\{A\},\{B\},\{C\},\{D\},\{AB\},\{AD\},\{BD\},\{ABD\}\}$,而实际上数据库 D 在最小支持度 $s'=2$ 下的所有频繁项目集为 $\{\{A\},\{B\},\{C\},\{D\},\{AB\},\{AC\},\{AD\},\{BC\},\{BD\},\{ABD\},\{ABC\}\}$,IUA 算法没有把频繁项目集 $\{AC\}$,$\{BC\}$ 及 $\{ABC\}$ 开采出来.

由上例知,IUA 算法不能将大量的频繁项目集以及许多有效规则开采出来.故此算法不是关联规则的增量式更新的有效算法.

2.2 IUA算法效率分析

基本性质 1. 一个频繁项目集的任一非空子集必定也是频繁项目集.

① IUA 算法与 Apriori 算法一样主要利用了上述基本性质 1.根据这一基本性质可知,对于任一项目 i,如果 i 不是任一 $j(j<k)$项目集的元素,则 i 一定不是 k-项目集的元素,而在 IUA 算法的第⑥～⑧步(见文献[1])的循环中,每调用一次 iua_gen 函数,通过该函数的拼接将会使一些已明显不是频繁 k-项目集的 k-项目集成为候选 k-项目集 C_k^3 中的元素,从而给 iua_gen 函数中的修剪增加运算量,因此增加了算法的时间复杂性.

② IUA 算法在关联规则更新时,对 k-项目集的开采,只是注意到利用已存在的频繁 k-项目集的集合 L_k,没有注意基于基本性质 1 在本次更新时,对新产生的频繁$(k-1)$-项目集的集合 L_{k-1}'加以利用.这在 IUA 算法中对 C_k^3 的产生以及由 C_k^3 来确定 L_k^3 完全能够体现这一点.另一方面,由以上对 IUA 算法的分析及 iua_gen 函数存在的问题,于是 PIUA 算法也将失去其意义.

基于上述对 IUA 算法的分析和讨论,下面给出 IUA 算法的改进算法.

2.3 增量式更新算法NEWIUA

给定事务数据库 D,一个项目集的支持度可以认为就是所有包含该项目集的交易的数目.设旧的最小支持度为 s,L_k 为数据库 D 在最小支持度 s 下对应的长度为 k 的所有频繁项目集所构成的集合,$k=1,2,\ldots,m_1$,此处 m_1 为所有频繁项目集的最大长度.对于新的最小支持度 s',设 L_k'为对应的所有频繁 k-项目集的集合,$k=1,2,\ldots,m_2$,同样地,m_2 为所有具有最小支持度 s'的频繁项目集的最大长度.对于每一个项目集都有一个域 *count* 用来保存它的支持度计数.

当最小支持度发生变化时,只有两种情况:① $s'>s$,② $s'<s$.对于情况①,文献[1]中已得到较完美的处理,故 NEWIUA 算法只对情况②作出处理.

NEWIUA 算法的基本框架与 IUA 算法和 Apriori 算法一致,对 $k=1,2,\ldots,m_2$,采用某种策略来生成候选 k-项目集的集合 C_k,然后扫描数据库来确定 C_k 中哪些 k-项目集是频繁项目集.

NEWIUA 算法与传统的增量式更新算法不同之处主要体现在以下两点.

① 因为有 $s'<s$,所以,原来所有在旧的最小支持度 s 下的频繁 k-项目集在新的最小支持度 s'下仍是频繁 k-项目集,因此在每一趟扫描数据库 D 计算候选 k-项目集的支持度计数时,就没有必要对 L_k 中的项目集重新再计算一次.因此 NEWIUA 算法在生成候选 k-项目集的集合 C_k 时不含 L_k 中的项目集.这在下列 NEWIUA 算法的基本框架描述中的第(1)步、第(8)步和第(8)′步中能够体现这一点.

② NEWIUA 算法在生成候选 k-项目集的集合 C_k 时,不但利用了已存在的频繁 k-项目集的集合 L_k,而且注意到,基于基本性质 1 对本次更新时新产生的频繁$(k-1)$-项目集的集合 L_{k-1}'加以充分利用.这一点由 NEWIUA 算法中的第(9)～(12)步来体现.

IUA 算法的改进算法——NEWIUA 算法的基本框架描述如下.

记 $L_k''=L_k'-L_k$,C_k 为所有候选 k-项目集所构成的集合.

NEWIUA 算法($s'<s$).

(1) $C_1=\{\text{all 1-itemsets } c \text{ of } I\}-L_1$
(2) for all transactions $t\in D$ do begin
(3) $C_t=subset(C_1,t)$; /* C_1 中包含于交易 t 的候选 1-项目集构成 C_t */
(4) for all candidates $c\in C_t$ do
$c.count$++; /* C_t 中候选 1-项目集的支持度记数加 1 */
(5) end
(6) $L_1^0=\{c\in C_1 \mid c.count\geq s'\}$; $L_1'=L_1^0\cup L_1$
(7) for (k=2; $L_{k-1}'\neq\varnothing$; k++) do begin
(8) $C_k=apriori\_gen(L_{k-1}')-L_k$
(9) for all itemsets $c\in C_k$ do /* 对 C_k 进行修剪 */
(10) for all (k−1)-subset s of c do
(11) if ($s\notin L_{k-1}'$) then
(12) delete c from C_k;
(13) for all transactions $t\in D$ do begin

(14) $C_t=subset(C_k,t)$ /* C_k中包含于交易 t 的候选 k-项目集构成 C_t */

(15) for all candidates $c\in C_t$ do

(16) $c.count$++; /* C_t中候选 k-项目集的支持度计数加 1 */

(17) end

(18) $L_k''=\{c\in C_k \mid c.conut\geq s'\}$;$L_k'=L_k''\cup L_k$;

(19) end

(20) $Answer=\cup_k L_k'$

我们也可以将上述算法的第(8)步改为如下形式:

(8)′ $C_k'=\cup c(c\in L_{k-1}')$;$C_k=\{$all k-subset c of $C_k'\}-L_k$.

在 NEWIUA 算法中选用(8)或(8)′要根据数据库 D 的特点来确定其优劣.

在上述算法中,第(1)步用来生成候选 1-项目集 C_1;第(2)~(5)步计算所有候选 1-项目集的支持度计数;第(6)步生成新的所有频繁 1-项目集;第(8)步生成所有候选 k-项目集的集合 C_k;第(9)~(12)步对 C_k 进行修剪;第(13)~(17)步计算所有候选 k-项目集的支持度计数;第(18)步生成新的所有频繁 k-项目集;第(20)步是结论.

3 关联规则开采的并行算法

数据开采问题的主要挑战性在于数据量巨大,因此算法的效率是关键.为了提高数据开采的效率,可有两种途径来解决此问题:一种是产生高效率的算法,另一种是计算机系统结构的改进.在目前的情况下,后者更具有吸引力.于是我们采用多处理机并行计算模型来进行关联规则的开采.所采用的多处理机并行计算模型是这样的:各处理机有各自的内存,事务数据库 D 储存于一个各处理机都能共享的大外部存储器中.在计算机互连网高度发达,高性能工作站通过局域网互连而构成工作站机群系统的今天,提出以上并行计算模型是很有意义的.

以下所提出的各关联规则开采并行算法的基本思想与 NEWIUA 算法一样.对于 k=1,2,...,首先采用某种策略来生成候选 k-项目集的集合 C_k 利用一些技巧对 C_k 进行修剪,然后扫描数据库确定频繁 k-项目集的集合 L_k,最后给出结果 $\cup_k L_k$.

3.1 多趟扫描数据库的并行算法

当处理机数远小于数据库的项目数时,采用以下的并行算法来进行关联规则开采.不妨设有两台处理机 P_1,P_2.设给定最小支持度为 s,关联规则开采的基本框架如下.

并行算法 1.

处理机 P_1 并行执行如下步骤:

(1) $C_1=\{$ all 1-itemsets s of I $\}$

(2) for all transactions $t\in D$ do begin

(3) C_{t_1}=subset(C_1,t) /* C_1中包含于交易 t 的候选 1-项目集构成 C_{t_1} */

(4) for all candidates $c\in C_{t_1}$ do

(5) $c.count$++; /* C_{t_1}中候选 1-项目集的支持度记数加 1 */

(6) end

(7) $L_1=\{c\in C_1 \mid c.count\geq s\}$

(8) for (k=3;$L_{k-2}\neq\emptyset$;k=k+2) do begin

(9) $C_k=\cup c\ (c\in L_{k-2})$

(10) $C_k=\{$all k-itemsets s of C_1 $\}$

(11) for all itemsets $c\in C_k$ do /* 对 C_k修剪 */

(12) for all (k–2)-subsets s of C

(13) if ($s\in L_{k-2}$) then

(14) delete c from C_k;

```
(15)    for all transactions t∈D do begin
(16)        C_t1=subset(C_k,t)      /* C_k中包含于交易 t 的候选 k-项目集构成 C_t1 */
(17)        for all candidates c∈C_t1 do
(18)            c.count++;          /* C_t1中候选 k-项目集的支持度计数加 1 */
(19)    end
(20)    L_k={c∈C_k | c.count≥s}
(21) end
```

处理机 P_2 并行执行如下步骤:

```
(1)  C_2={all 2-itemsets s of I}
(2)  for all transactions t∈D do begin
(3)      C_t2=subset(C_2,t)      /* C_2中包含于交易 t 的候选 2-项目集构成 C_t2 */
(4)      for all candidates c∈C_t2 do
(5)          c.count++;          /* C_t2中候选 2-项目集的支持度计数加 1 */
(6)  end
(7)  L_2={c∈C_2 | c.count≥s}
(8)  for (m=4; L_m-2≠∅;m=m+2) do begin
(9)      C_2=∪c(c∈L_m-2)
(10)     C_m={all m-itemsets s of C_2}
(11)     for all itemsets c∈C_m do
(12)         for all (m-2)-subsets s of C
(13)             if (s∉L_m-2) then
(14)                 delete c from C_m;
(15)     for all transactions t∈c do begin
(16)         C_t3=subset(C'_m,t)     /* C_m中包含于交易 t 的候选 m-项目集构成 C_t3 */
(17)         for all candidates c∈C_t3 do
(18)             c.count++;          /* C_t3中候选 m-项目集的支持度计数加 1 */
(19)     end
(20)     L_m={c∈C_m | c.count≥s}
(21) end
```

当 P_1,P_2 各自都完成计算后,由其中一台处理机计算 $Answer=\cup_k L_k$,则算法全部结束.

并行算法 1 采用的方法是处理机 P_1,P_2 并行执行.P_1 用来产生所有长度是奇数的频繁项目集,P_2 用来产生所有长度是偶数的频繁项目集,然后由其中一台处理机计算出最后结果.

设数据库 D 在最小支持度 s 下所有频繁项目集中最大的长度为 m_1,则 Apriori 算法需扫描数据库 m_1 次或 m_1+1 次.这种情况下,在并行算法 1 中,两台处理机分别并行地扫描数据库最多$[m_1/2]+1$ 次,故并行算法 1 与 Apriori 算法相比,执行效率提高将近 1 倍.而并行算法 1 也可以增加处理机的数量,以提高数据开采的速度.但在并行算法 1 中,并非处理机数量越多越好,这是因为当处理机数量增加时,会将很多根本不可能是频繁项目集的项目集添加到对应的候选项目集中,于是就增加了无谓的计算量.因此,在采用并行算法 1 进行关联规则开采时,应适当选用处理机数目.

3.2 分部分扫描数据库的并行算法

设有 n 台处理机 $P_1,P_2,\ldots,P_n$,我们可以采用下列并行算法来进行关联规则的开采.

并行算法 2.

(1) 将数据库 D 均匀地分为 n 个部分 $D_1,D_2,\ldots,D_n$;

(2) 处理机 P_1 生成 I 的所有 1-项目集构成候选 1-项目集的集合 C_1;

(3) 处理机 $P_j(j=1,2,\ldots,n)$并行扫描数据库 D 的对应部分 D_j,计算候选 1-项目集的支持度计数;

(4) 根据各处理机的计算结果,处理机 P_1 计算出所有候选 1-项目集的最后支持度计数,并生成 L_1;

(5) 循环(k=2;$L_{k-1}\neq\varnothing$;k++) do begin

(6) 由处理机 P_1 生成候选 k-项目集的集合 C_k,即 C_k=*apriori_gen*(L_{k-1});

(7) 各处理机 P_j(j=1,2,...,n)并行地扫描数据库 D 的对应部分 D_j,计算候选 k-项目集的支持度计数;

(8) 根据各处理机的计算结果,处理机 P_1 计算出所有候选 k-项目集的最后支持度计数,并生成 L_k;

(9) 循环结束;

(10) 处理机 P_1 最后生成 *Answer*.

注:各处理机在并行执行时由处理机 P_1 负责同步.

并行算法 2 扫描数据库的趟数与 Apriori 算法相同,但在每一趟扫描数据库时,由于各处理机分别并行地扫描数据库的一个部分,故从理论上讲,并行算法 2 扫描数据库的效率是 Apriori 算法的 n 倍,即并行算法 2 比 Apriori 算法的效率提高了近 n 倍.

本文提出的并行算法 2,在当前大规模并行计算机中以及互连网技术高度发展的情况下是一种非常实用和有效的算法.在分布式数据库中进行关联规则的开采,并行算法 2 也是一种有效的算法.在并行算法 2 中,扫描数据库以计算项目集的支持度计数的执行效率是非常高的,而此算法所面临的是在并行计算模型中的消息发送与接收的速度问题.若在一个消息发送与接收高效的系统中,利用并行算法 2 来对数据库(或分布式数据库)进行关联规则的开采则是非常高效的.

3.3 两趟扫描数据库的并行算法

设有 m 台处理机,分别为 P_1,P_2,P_3,...,P_m.与数据库 D 的项目数相等时,一种最简单明了的算法是:处理机 P_j(j=1,2,3,...,m)并行地执行如下操作:

(1) C_j={all j-subsets s of I}, /* I 的所有 j-项目集所构成的集合作为候选 j-项目集*/

(2) 扫描数据库 D,计算各候选项目集的支持度计数,

(3) L_j={$c\in C_j$ | $c.count\geq s$}.

当所有处理机都完成计算后,由其中一台处理机计算 *Answer*=$\cup_k L_k$,则算法结束.

采用上述算法虽然各处理机并行地扫描一次数据库即可完成关联规则的开采,但是此算法的计算量是非常大的,要计算 I 的 2^m-1 个非空子集的支持度.但是若事先已知 I 的某一项目所构成的集合不是频繁 1-项目集,则只计算 I 的 $2^{m-1}-1$ 个非空子集的支持度,运算量下降了 1 倍.在实际的关联规则开采时,一般情况下有下列两个特点:① 在由 m 个项目组成的项目集 I 中,所有项目构成的 1-项目集不可能全是频繁 1-项目集.② I 中所有项目所构成的 m-项目集不可能是频繁项目集.根据上述两个特点,为了减少计算量,我们提出以下并行算法.

并行算法 3.

(1) 各处理机 P_j 并行地扫描数据库,计算 1-项目集{i_j}的支持度,最后根据各处理机的计算结果由其中一台处理机确定出 L_1,并计算 $C=\cup c(c\in L_1)$或(1)′由其中一台处理机计算出 L_1,并计算 $C=\cup c(c\in L_1)$.设 L_1 中频繁 1-项目集的数目为 m_p,则 $m_p\leq m$;

(2) 各处理机 P_j(j=1,2,...,m_p-1)并行执行以下操作:

① C_{j+1}={all (j+1)-subsets s of C}; /* 产生候选(j+1)-项目集 */

② 扫描数据库 D,计算各候选集的支持度;

③ L_{j+1}={$c\in C_{j+1}$ | $c.count\geq s$}; /* 确定频繁(j+1)-项目集 */

(3) 由其中一处理机计算 *Answer*=$\cup_k L_k$.

一般情况下,经过并行算法 3 的第(1)步处理后,在执行算法的第(2)步,各处理机并行执行所产生的候选项目集数目的总和要比 2^m-1 少得多.这样,在各处理机分别并行地扫描数据库 D 以计算各候选项目集的支持度计数时,计算量会大幅度下降,加之并行算法 3 用于关联规则开采时仅对数据库扫描两次,从而大大提高了关联规则开采的效率.

在处理机数量非常多的并行计算模型中,对数据库进行开采时,本文提出的并行算法 3 是一种高速算法.此算法是以某些处理机执行无用的计算(即计算那些明显不是频繁项目集的支持度)为代价来换取速度的.以

机器的数量换取数据开采的速度,这在计算机硬件大幅度降价的今天,也是一种很有意义的方案.

本文提出的并行算法 3 可以有下列两种改进方案.(1) 当处理机数量不增加时,在并行算法 3 中,当对数据库第 2 次扫描时,由于有些处理机已处于空闲状态,这时可以让那些空闲的处理机去分担那些计算负荷较大的处理机的计算任务.这需要在算法中采用动态分配处理机的方法.(2) 当处理机数量是项目数量的若干倍时,我们可以将并行算法 2 和并行算法 3 结合起来,将数据库分为若干部分,每一部分利用并行算法 3 进行项目集的支持度计算.

对于本文提出的关联规则开采并行算法,都可以根据本文提出的关联规则更新算法 NEWIUA 中所提供的方法,修改成效率较高的关联规则更新的并行算法.

4 结束语

大量数据的产生和收集导致了信息爆炸,要从中发现有价值的信息或知识,达到为决策服务的目的,成为非常艰巨的任务.数据开采方法正是为此目的而提出的.

数据开采所面对的是大型数据库或大型数据仓库,其任务是从海量数据中挖掘有价值的信息或知识.因此,数据开采所面临的最大的挑战是计算效率问题,解决这一问题的途径是产生高效的数据开采算法与并行处理,而两者之间的结合更具有引吸力.也就是说,随着现代社会的信息量的急剧增长,迫切需要高效的并行数据开采算法的产生.

参考文献

1 冯玉才,冯剑琳.关联规则的增量式更新算法.软件学报,1998,9(4):301~306
(Feng Yu-cai, Feng Jian-lin. Incremental updating algorithms for mining association rules. Journal of Software, 1998,9(4):301~306)

2 胡侃,夏绍玮.基于大型数据仓库的数据采掘:研究综述.软件学报,1998,9(1):53~63
(Hu Kan, Xia Shao-wei. Large data warehouse-based data mining: a survey. Journal of Software, 1998,9(1):53~63)

3 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases. Santiago, Chile, 1994. 487~499

Data Mining and Incremental Updating on Association Rules

ZHOU Hai-yan

(*Taiyuan Teachers' College Taiyuan* 030001)

Abstract In this paper, the author analyzes the incremental updating algorithm (IUA) on association rules, points out its existing problems, and presents an improved algorithm, NEWIUA (new IUA), which takes full use of already existing and the current updated new frequent itemsets, therefore the efficiency is increased besides guaranteeing the validity of the algorithm. Three parallel algorithms for data mining on association rules are presented, the analysis and discussion on each algorithm are also presented.

Key words Data mining, association rules, incremental updating, frequent itemsets, parallel algorithm.