关联规则挖掘算法的多核并行优化*

吴华平,郑晓薇,张建强
(辽宁师范大学 计算机与信息技术学院,辽宁 大连 116081)

摘 要:分析了并行关联规则挖掘算法存在的不足,提出了一种改进的关联规则挖掘的多核并行优化算法。该算法对 Apriori 算法的压缩矩阵进行了改造,并在多核平台下利用 OpenMP 技术和TBB 技术对串行程序进行循环并行化和任务分配的并行化设计,最大限度地实现并行关联规则挖掘。

关键词:关联规则;Apriori 算法;频繁项集矩阵;OpenMP;TBB;多核并行

中图分类号:TP311 **文献标识码**:A **文章编号**:1674-7720(2011)01-0004-03

Matrix compression based on multi-core TBB parallel algorithms for mining association rules

Wu Huaping, Zheng Xiaowei, Zhang Jianqiang
(College of Computer and Information Technology, Liaoning Normal University, Dalian 116081, China)

Abstract: This paper analyzes the parallel algorithm for mining association rules exist, the paper proposes an improved multi-core parallel association rule mining algorithm. The algorithm transforms the compression matrix of Apriori algorithm, and uses OpenMP and TBB technology under multi-core platform to complish cycle of serial procedures and task allocation in parallel of parallel design, to maximize the parallel association rule mining.

Key words: association rules; Apriori algorithm; frequent itemsets matrix; OpenMP; TBB; multi-core parallel

海量数据中隐藏着大量的不为人知的模式和知识,寻找有价值的数据模式和知识是数据挖掘研究的主要内容[1]。关联规则的挖掘是数据挖掘中的一项重要的基础技术。经典的关联规则挖掘算法主要有 Agrawal 等提出的基于 Apriori 算法的频繁集方法,该算法以递归统计为基础,以最小支持度为依据剪切生成频繁集。随着数据容量的增加,为了提高关联规则的挖掘效率,研究人员提出了并行挖掘算法[2-3]。这些并行算法都是基于 MPI 和机群系统实现的,虽然具有速度快、容易实现、要求各节点间同步次数较少等优点,但仍然存在着可扩展性差、网络通信量大、负载不平衡、处理器空转、规则合成难度高等缺点。

目前市场上多核处理器已成为主流,比较有代表性的支持多核处理器的并行计算平台之一的线程构建模块 TBB(Thread Building Blocking),可以在其他多核化工具支持下对串行程序中的可并行化部分进行线程的并行化重构,提升多核处理器平台的效能,简化应用程序的并行化过程。本文针对 TBB 的多核并行编程的优势,结合 OpenMP 并行编程,提出了一种改进的关联规则挖掘多核并行优化算法。该算法对 Apriori 算法的压缩矩阵进行了改造,只需扫描一次数据库,并利用 TBB 技术最大限度地压缩矩阵,使矩阵的运算规模逐步减小。它不需要 Apriori 算法中的自联接和剪枝,直接通过支持矩阵行向量的按位"与"运算并行地找出频繁集,减少了数据移动带来的额外开销,提高关联规则挖掘效率。与分布式系统的 Apriori 并行算法相比,该算法采用多核 TBB 并行技术,不存在节点间的通信与同步、负载平衡和规则合成难度高等问题。实验证明该算法具有高效的并行挖掘效率和较高的多核 CPU 利用率。

1 挖掘关联规则的串行算法

关联规则的核心是基于两阶段频繁项集思想的递推算法。发现频繁项集是关联规则挖掘应用中的技术和步骤。支持度和置信度是关联规则挖掘中的两个重要指

* 基金项目:国家自然科学基金项目(No.60603047)

标，为了计算支持度，需要访问数据库。而 Agrawal 等人提出的挖掘关联规则串行算法 Apriori 是首先扫描数据库，计算每个数据项的支持度，并根据支持度阈值产生频繁 1-项集 L_1；L_1 用于找频繁 2-项集 L_2，L_2 而用于找 L_3，如此逐层迭代的搜索，直到不能找到频繁 k-项集。Apriori 算法存在一些难以克服的缺陷：(1)可能产生大量的候选项集，没有排除不应该参与组合的元素；(2)多次扫描数据库，大大增加系统的 I/O 开销，并且数据库有些可以删除的项或事务被多次扫描；(3)连接程序中相同的项重复较多。针对 Apriori 算法的缺点，参考文献[4]将事务数据库转换为基于内存的矩阵，在矩阵上找出所需的频繁项集，从而大大减少了数据库的扫描次数，但没有对矩阵进行压缩。参考文献[5-6]对矩阵进行了压缩，但不彻底，而且矩阵数据结构不合理，还额外增加了转置矩阵。

本文介绍一种改进的基于 Apriori 算法的挖掘关联规则的多核并行优化算法。本文改进了参考文献[4-5]中的矩阵的数据结构：在一个单纯的事务矩阵中，添加 2 个辅助行和 1 个辅助列，方便矩阵进行彻底的压缩，使矩阵的规模逐步减小，运算量也大为减少；同时为了配合查找频繁 k-项集(k>=2)的运算，设置了一个简单的辅助二维数组，用来记录下标组合情况。

定义 1 支持矩阵 R：设 A 为任一给定的事务数据库，m 为事务的个数，n 为项目的个数，事务集 $T_i(i \in m)$，项目集 $I_j(j \in n)$。如果 $I_j \in T_i$ $(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m)$，则支持矩阵 $r_{ij}=1$，否则 $r_{ij}=0$。为方便矩阵进行彻底地压缩，再为矩阵添加 1 列 sum_r，记录每个事务的事务数，即累计每行的项数；为矩阵添加 1 行 sum_c，记录每个项的项支持度；为矩阵添加一项，记录项的编码，则构造支持矩阵 $R_{(m+2)\times(n+1)}$。支持矩阵 R 如图 1 所示。

$R_{(m+2)\times(n+1)}=$

	sum_r	I_1	I_2	I_3	…	I_n
T_1	$r_{1,0}$	$r_{1,1}$	$r_{1,2}$	$r_{1,3}$	…	$r_{1,n}$
T_2	$r_{2,0}$	$r_{2,1}$	$r_{2,2}$	$r_{2,3}$	…	$r_{2,n}$
T_3	$r_{3,0}$	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$	…	$r_{3,n}$
…	…	…			…	…
T_m	$r_{m,0}$	$r_{m,1}$	$r_{m,2}$	$r_{m,3}$	…	$r_{m,n}$
	sum_c	$r_{(m+1),1}$	$r_{(m+1),2}$	$r_{(m+1),3}$	…	$r_{(m+1),n}$

图 1 支持矩阵 R

其中：第一行为记录项的编码；最后一行为每个项的支持度；第一列为每个事务的事务数；r_{ij}=1|0，$(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m)$。

定义 2 每个项 I_i 的支持度为 $r_{(m+1),i}=\text{support}(I_i)=\sum_{j=1}^{m} r_{ij}$（$r_{ij}$ 为支持矩阵 R 中的元素），即该 i 列向量中“1”的个数。每个事务 T_j 的事务数为 $r_{j,0}=\sum_{j=1}^{m} r_{ji}$（$r_{ji}$ 为支持矩阵 R 中的元素），即该 j 行向量中“1”的个数。

定义 3 k-项集支持度：对支持矩阵 R 中的任意 k 列向量(除去第一列)进行对位(同行的元素)“与”运算，运算结果中“1”的个数称为 k 列向量的 k-项集支持度。

根据 k-项集支持度的定义，结合 Apriori 性质，可以得到以下性质。

性质 1 X_k 是 k-项集，如果频繁$(k-1)$-项集 L_{k-1} 中包含 X_k 的$(k-1)$-子项集的个数小于 k，则 X_k 不可能是 k 维最大频繁项集。证明见参考文献[6]。

性质 2 设 k 为 k-项频繁项集，若 $T \subseteq I$，且支持矩阵 $R_{(m+2)\times(n+1)}$ 中 T 的事务数等于 k，由于 k-项集对于生成频繁$(k+1)$-项集没有作用，则求$(k+1)$-项集支持度时，该行事务 T 可删除。

性质 3 对于频繁 k-项集的集合 L_k，如果$|L_k|<k+1$，则被挖掘的事务数据库最大频繁项集的次数为 k。其中$|L_k|$是指 L_k 的频繁 k-项集的个数。

证明：对于频繁$(k+1)$-项集 $I_x=\{I_1, I_2, \cdots, I_{k+1}\}$，一定有 $k+1$ 个频繁 k-项子集，若频繁 k-项集的集合 L_k 元素个数小于 $k+1$，则被挖掘的事务数据库不存在频繁$(k+1)$-项集。利用该性质可以提前终止搜索频繁集的循环。

2 多核并行编程技术

OpenMP 是共享存储系统编程的工业标准，它具有简单、移植性好和可扩展等优点。OpenMP 规范了一系列的编译制导、运行时库函数和环境变量来说明共享存储结构的并行机制。OpenMP 实现的是线程级的并行，线程间通过读/写共享变量实现，使用 Fork-Join 的并行执行模式。

TBB 是针对多核平台开发的一组开源的 C++的模板库，基于 GPLv2 开源证书，支持可伸缩的并行编程[7-8]。TBB 的编程模式通过使用模板来提供常见的并行迭代模式，使程序员即使在不具备很专业的同步、负载平衡、缓存优化等专门知识的情况下，也能够实现自动调度的并行程序，使得 CPU 的多个核心处于高效运转之中。TBB 完全支持嵌套的并行，程序员可以很容易地创建自己的并行组件，进而构建大型的并行程序。TBB 并行编程指定的是任务，而不是线程[9]，并以高效的方式将任务自动映射到线程，程序容易实现，且具有更优的可移植性和可扩展性。

3 关联规则挖掘算法的多核并行优化

本文在改进算法的同时，运用多核平台并行编程的优势，配合采用 OpenMP 的工作分区 sections 和并行库 TBB 的 tbb_parallel_for，可以实现每个工作段都由多个执行核并行执行和负载均衡的并行执行固定数目独立循环迭代体，用于提高查找频繁项集的效率。基本流程如图 2 所示。

(1)并行扫描数据库 D，建立支持矩阵 $R_{(m+2)\times(n+1)}$。TBB 先初始化对象，再对包含 m 个事务 n 个项目的事务数据库 D=$\{T_1, T_2, \cdots, T_m\}$，项目集 $I=\{I_1, I_2, \cdots, I_m\}$，运用

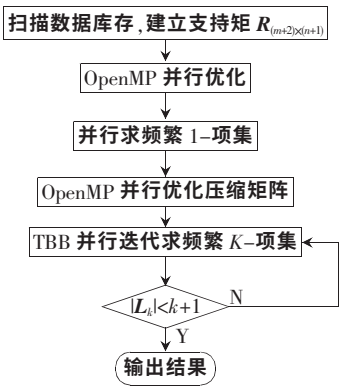

图 2 并行优化流程图

OpenMP 的工作分区 sections 并行优化扫描事务数据库 D，建立支持矩阵 $R_{(m+2)\times(n+1)}$。

```
task_scheduler_init init;                    //TBB 初始化对象
#pragma omp parallel sections                //工作分区
{  #pragma omp section
   for(i=1;i<=n;i++)                         //n,m,k 为全局变量
   {  R[0][i]=i;                             //填充项编号行
    R[m+1][i]=0;                             //填充 sum_c 行 }
   #pragma omp sectionfor(i=0;i<M;i++)
{   sum_r=0;
 for(j=1;j<=n;j++)
       if(Ij in Ti)
{   R[i+1][j]=1;
 sum_r++;                                    //统计项支持度
 R[m+1][j]++;                                //统计行的项数 }
  R[i+1][0]=sum_r; } }
```

(2)并行求频繁 1-项集。$k=1$，计算最小事务支持度 min_supsh=ceiling(min_sup×m)，压缩矩阵的列，余下的项都是频繁 1-项集。重新计算矩阵的 sum_r 列和 sum_c 行。

(3)$k=2$。

(4)并行压缩支持矩阵。按照性质 1，先统计每个项目 I_1 在频繁$(k-1)$-项集中出现的次数 b_i，若 b_i 小于$(k-1)$，则删除 R 中 I_i 对应的矩阵列。重新计算矩阵的 sum_r 列和 sum_c 行，按照性质 2，删除不含条件的列和行，直到不能再压缩为止。

(5)并行求频繁 k-项集。若压缩后支持矩阵 R 列数为 c_num(k<c_num)，则对支持矩阵中的后 c_num-1 列进行 k 维组合，生成 $P=C_{c\_num-1}^{k-1}$ 个组合对，每个组合对中的 k 个列向量(除去第一列)进行对位(同行的元素)“与”操作，对结果向量求 k 项集支持度，k 项集支持度不小于 min_supsh 的组合对所对应的项集则为频繁 k-项集。主要算法如下：

```
//R 支持矩阵   W_pxk 存放 P 个 k 维项组合对
// b[i]统计每个项目 I1 在频繁(k-1)-项集中出现的次数
long count=0;
                                  //统计频繁 k-项集中出现项集个数的变量
for(i=1;i<=n;i++) b[R[0][i]]=0;
parallel_for(blocked_range<size_t>(1, P), Found_K(R,b,
W,count)); //找出频繁 k-项集,对固定数目的独立循环迭
                                  //代体并行计算
class Found_K::void operator () (const blocked_range<
size_t> & r) const
 {for (size_t i = r.begin(); i != r.end(); ++ i)
    {  long suppnum=0;
       for(long j=1;j<=m;j++)
       {   long s=1;
           for(long h=0;h<k;h++)
              {s=s*_R[j][_W(i,h)];
                                  //对位“与”操作
              if(s==0)break;}
           suppnum+=s;   // 统计 k-项集支持度 }
       if(suppnum>=min_supsh)
        {  _count++;
           for(long h=0;h<k;h++)
              _b[_R[0][_W(i,h)]]++;}
    }  }
```

(6)k++；重复步骤(4)~(5)，直到 count<k 停止。

4 实验及分析

为了验证基于多核并行技术的改进挖掘关联规则算法的性能，本文在 Intel(R)Pentium(R)D CPU 3.0 GHz、1.86 GHz、1 GB 内存的双核处理器系统上测试了参考文献[8]的 BBM 算法，改进的挖掘关联规则串行算法(以下称本文串行算法)及改进的挖掘关联规则的多核并行优化算法(以下简称多核并行算法)。从参考文献[10]选择数据集进行实验，事务数据库共 100 000 条事务，事务的平均长度为 12。实验测试结果见表 1，其中，加速比=本文串行执行时间/多核并行执行时间，CPU 运行效率=加速比/核数。

表 1 三种算法执行时间比较

最小支持度/%	BBM 算法	本文串行算法	多核并行算法	加速比	CPU 利用率/%
10	16.872	13.923	7.732	1.8	90
15	13.793	12.021	6.342	1.89	95
20	9.861	9.347	5.175	1.81	90.50
25	9.323	9.065	4.947	1.83	91.50
30	7.914	7.635	4.265	1.82	91.00

表 1 表明，支持度较高时，这三种算法的执行时间差别并不大；但当支持度逐渐降低时，与 BBM 算法相比，本文串行算法的执行时间要更短，而多核并行算法的执行时间几乎是本文串行算法的一半，具有高的并行挖掘效率。从加速比和 CPU 利用率分析，多核并行算法的多核 CPU 运行效率达到 90%左右，充分调度了两个

处理核心的资源，体现了计算机双核的优势。

关联规则技术是数据挖掘中的一种重要的基础算法，本文在深入研究 Apriori 算法的基础上，提出了一种改进的关联规则挖掘的多核并行优化算法，综合了布尔矩阵和多核并行编程的优点，节约了存储空间，减少了执行时间，具有较高的并行挖掘效率和多核 CPU 的利用率。本算法的设计方法对于相关算法的研究有较好的借鉴作用。

参考文献

[1] 何军，刘红岩，杜小勇.挖掘多关系关联规则[J].软件学报，2007，18(11).
[2] Boeyen SX.509 (2000): 4th Edition Overview of PKI&PMI Frameworks. http: //www. entrust. com.
[3] 何中胜.基于向量的并行关联规则挖掘算法[J].计算机系统应用，2009，18(3)：42-45.
[4] 曾万聘，周绪波，戴勃，等.关联规则挖掘的矩阵算法[J].计算机工程，2006，32(2)：45-47.
[5] 张月琴.基于 0-1 矩阵的频繁项集挖掘算法研究[J].计算机工程与设计，2009，30(20).
[6] 张素兰.一种基于事务压缩的关联规则优化算法[J].计算机工程与设计，2006，27(18)：3450-3453.
[7] Reinders J. Intel threading building blocks [M]. [s.l.]: O'REILLY 出版社，2007.
[8] 胡斌，袁道华.TBB 多核编程及其混合编程模型的研究[J].计算机技术与发展，2009，19(2)：89-101.
[9] Intel threading building blocks 基于任务编程 [OL]. http://www. cppprog. com/2009/0401/96_2. html.
[10] ALMADEN I. Quest synthetic data generation code. http://www. almaden. ibm. com/cs/quest/syndata. html.

(收稿日期：2010-09-05)

作者简介：

吴华平，男，1984 年生，硕士研究生，主要研究方向：并行计算，多核计算机系统。

郑晓薇，女，1957 年生，教授，主要研究方向：并行计算，多核计算机系统。

张建强，男，1981 年生，硕士研究生，主要研究方向：并行计算，多核计算机系统。