序列模式数据挖掘算法的并行化研究

王宗江

（潍坊学院计算机与通信工程学院　山东潍坊 261061）

摘　要　序列模式在许多领域都有着重要的应用，大量的数据和模式需要高效的、可扩展的并行算法。针对目前序列模式挖掘算法存在的普遍问题，在对串行序列模式数据挖掘算法研究的基础上，本文提出了一种并行的序列模式数据挖掘算法。通过理论分析与实验验证可知：该并行数据挖掘算法，在海量数据的情形下，能很好地提高数据挖掘的效率。

关键词　数据挖掘，序列模式，最小支持度，并行算法，频繁序列集

Parallel Research of Sequential Pattern Data Mining Algorithm

WANG Zong-jiang

(School of Computer and Communication Engineering, Weifang University, Weifang, Shandong 261061, China)

Abstract　Sequential pattern has important applications in many areas and a large number of data and patterns need efficient and scalable parallel algorithm. In light of the widespread problem in current sequential pattern data mining algorithm and on the basis of researching the data mining algorithm of serial sequential pattern, a data mining algorithm of parallel sequential pattern is presented in this paper. Through theoretical analysis and experimental verification, the parallel data mining algorithm can well improve the efficiency of data mining in massive data circumstances.

Keywords　Data mining, Sequential pattern, Minimal support, Parallel algorithm, Frequent sequence aggregate

1　引言

在数据库中，发现知识（Knowledge discovery in databases，简称 KDD），亦称数据挖掘（Data Mining），已是当今国际上人工智能和数据库研究方面最富活力的新兴领域，其目标是为了满足用户目标，自动处理大量的原始数据，从中识别出重要的和有意义的模式，并将其作为知识加以表达。由于其强大的应用潜力以及可广泛用于存在于各种数据库中的大量数据上，因此 KDD 成为一个具有迫切现实需要的热点研究课题。序列模式（Sequential Pattern）的发现由 Agrawal 和 Srikant 于 1995 年首先提出，是数据挖掘研究的重要内容。本文对数据挖掘中的序列数据进行了分析，提出了一种并行的序列挖掘算法。

2　基本概念

序列模式挖掘的侧重点在于分析数据间的前后序列关系，它能发现数据库中形如“在某一段时间内，顾客购买商品 A，接着购买商品 B，而后购买商品 C，即序列 A→B→C 出现的频度较高”之类的知识。为讨论问题方便，对文中涉及的概念进行如下定义：

定义 1　设 $I=\{a_1,a_2,\cdots,a_k\}$，其中 $a_i(1\leqslant i\leqslant k)$ 为数据项，若 $I\neq\Phi$，则称 U 为项集。

定义 2　序列是项集的有序表，记为 $S=\langle s_1,s_2,\cdots,s_n\rangle$，其中 $s_i\subseteq I,(1\leqslant i\leqslant n)$，序列的长度为：$L=\sum_{1\leqslant i\leqslant n}|s_i|$。

定义 3　令序列 $S_1=\langle a_1,a_2,\cdots,a_n\rangle$，$S_2=\langle b_1,b_2,\cdots,b_m\rangle$，若存在整数 $i_1\leqslant i_2\leqslant\cdots\leqslant i_n$，使得 $a_1\subseteq b_{i1},a_2\subseteq b_{i2},\cdots,a_n\subseteq b_m$，则称 S_1 是 S_2 的字序列，或者说 S_2 包含序列 S_1。在一组序列中，如果某序列 S 不包含在其它任何序列中，则称 S 是该组中最长序列。一组序列中可能存在多个最长序列。

定义 4　函数 $c:I\to I^+$（正整数集），称为项集 I 的标识函数。函数 $t:I\to I^+$，称为项集 I 的时间函数，它表示项集 I 对应的时刻。事务 T 是由标识函数值及时间函数值标识的项集，标识函数 c 称为事务的标识号，记为 $c(T)$。时间函数值称为事务发生的时刻，记为 $t(T)$。数据以 $(c(T),t(T))$ 表的形式存储，事务序列的集合称为事务数据库。

定义 5　给定序列 S，事务数据库 DB，则 S 在 DB 中的绝对支持度为 DB 中包含 S 的元组数目，相对支持度为 DB 中包含 S 的元组在整体数据库元组中所占的百分比。支持度大于最小支持度的 K-序列，称为 DB 上的频繁 K-序列，记为 F_k。

3　序列模式挖掘的相关技术

3.1　采用前缀投影技术生成局部序列模式

所有的序列模式按其搜索次序形成了一棵序列树，树的根标记为 NULL，第 1 层为 L_1 序列模式，第 2 层为 L_2 序列模式，…。对树中处于第 1 层以下的任意节点，设长度为 L，其父节点是其前缀，长度为 $L-1$；其子节点以它为前缀，长度为 $L+1$。序列树可以根据 L_1 序列模式划分为多个子树，我们称这些子树为 L_1 子树（相应地，长度为 k 的序列模式所对应的子树记为 L_k 子树）。

在各数据站点 $S_i(i=1,2,\cdots,m)$ 上，按字典序依次生成各个 L_1 子树。我们将各数据站点生成的子树称为局部子树（local subtree），则各数据站点生成的 L_1 子树称为局部 L_1 子树。而将最终生成的全局频繁序列所构成的子树称为全局子

王宗江　副教授，主要从事计算机控制、软件工程研究。

树(global subtree),全局子树中所有序列模式以子树根节点对应序列模式为前缀。此外,我们在生成 L_1 投影数据库时删除了所有非频繁项,因为非频繁项在并不出现在局部序列模式和全局序列模式中,进一步降低了投影数据库规模。

3.2 局部序列模式与全局序列模式之间存在的特殊性质

定义 6 对于站点 $S_i(1\leqslant i\leqslant m)$ 上的一个局部序列模式 S_e,如果 S_e 同时也是全局序列模式,我们称 S_e 为 S_i 上的全局——本地序列模式,记为 *gl-seq*。

引理 1 对于任意一条全局序列模式 S_e,存在站点 S_i,S_e 和其所有子序列都是 S_i 上的 *gl-seq*。

证明:假设不存在这样的站点 S_i,则由问题定义知:$Count_i(S_e)<\min Count_i(i=1,2,\cdots,m)$。因此,DB 中包含 S_e 的序列总数为:$Count(S_e)=Count_1(S_e)+\cdots+Count_m(S_e)<\min Count_1+\cdots+\min Counbt_m=\min sup p\times|DB|$,则 S_e 不满足最小支持度,故假设不成立。由 Apriori 性质可知 S_e 的所有子序列都是 S_i 上的 *gl-seq*。

定义 7 对任意一个全局 L_1 序列模式 x,对应的全局 L_1 子树记为 $\{x\}seq$;如果它在站点 S_i 上是 *gl-seq*,则在 S_i 上对应的局部 L_1 子树记为 $\{x\}\text{-}seq_i$,对应的 L_1 投影数据库记为 $\{x\}-DB_i$。将所有局部 L_1 子树的集合记为 UL_1,$UL_1=\bigcup_{x\subseteq L_1}\bigcup_{i=1}^{m}\{x\}\text{-}seq_i$。

定理 1 所有全局序列模式的集合 F_G 是所有局部 L_1 子树的集合 UL_1 的子集。

证明:$UL_1=\bigcup_{x\subseteq L_1}\bigcup_{i=1}^{m}\{x\}\text{-}seq_i=\bigcup_{i=1}^{m}\bigcup_{x\subseteq L_1}\{x\}\text{-}seq_i$,根据 PrefixSpan 算法,$\bigcup_{x\in L_1}\{x\}\text{-}seq$ 即为站点 S_i 上的所有局部 L_1 子树集合;因此 $F_G\subseteq\bigcup_{i=1}^{m}\bigcup_{x\subseteq L_1}\{x\}\text{-}seq_i$,即 $F_G\in UL_1$。

3.3 全局序列模式生成技术

在各站点上,采用前缀投影技术生成了局部序列模式。为判断它们是否为全局序列模式,我们需要得到这些局部序列模式的全局支持计数。

如果我们采用广播方式统计所有局部序列模式的全局支持度,则计算一条局部序列的全局支持度的通信次数是 $O(m^2)$。通常情况下,很少有局部序列在所有站点上均是局部频繁的。因此,在通过广播方式得到全局 L_1 序列模式后,我们将每个局部 L_1 子树 $\{x\}\text{-}seq_i$ 拆分为多个 L_2 子树,使用一个分配函数,如哈希散列方法,将 L_2 子树分配到相应站点,该站点称为该子树上所有序列的选举站点,负责统计它们的全局支持计数。每个局部序列的选举站点是唯一的,计算一条局部序列的全局支持计数的通信次数是 $O(m)$。

4 串行的序列模式挖掘

4.1 串行数据挖掘算法

串行数据挖掘算法利用一个按字典序排列的投影树数据结构来表示所有项目集的集合。树中的某个节点对应于一个特定的项目集,而该节点在树中所处的层数对应于项目集的长度。

只有当对应于频繁项目集的节点生成时,才可用投影树算法以广度优先或深度优先的方式对生成树进行扩展。投影树算法的一个主要特征是通过使用数据库投影方法,在第 $k-1$ 层的双亲节点对原始事务数据进行投影,来决定在算法的第 k 次迭代过程中所产生的候选项目集的支持度。具体做法是对于树中的每一个节点,保留一张项目表,表中的每个项目可以在对应于后辈节点的项目集中找到。这些项目称为节点的动态项目。当一个事务投影到某个节点上时,只有出现在动态项目表中的那些项目才保留下来,而剩余的项目则被剔除,图 1 示例了一颗投影树扩展到第三层的情况。

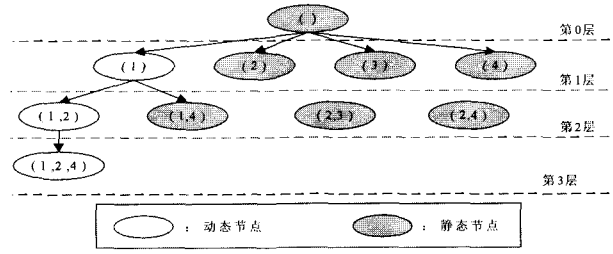

图 1 项目集投影树实例

4.2 基于串行算法的序列模式挖掘

发现序列模式的投影树算法也是使用一个字典序的树结构来表示所有可能的序列。树中的每个节点对应于一个特定的序列而且第 k 层的节点对应于长度为 k 的序列。序列节点与其子节点之间的关系本质上类似于原投影树算法中的项目集之间的关系。然而,有所不同的是,每个节点有两种类型的子节点,它是通过在该序列的最后一个项目集中增加一个项目来获得,但所增加项目的字典序必须比最后一个项目集中的所有项目要大;另一种类型的子节点可以通过在序列的最后增加一个大小为 1 的新项目来获得,此时对项目的字典序没有限制。这两种类型节点的获得过程分别称为项目集扩展和序列扩展。

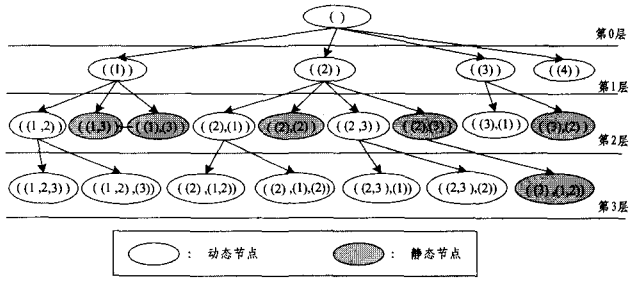

图 2 序列模式发现的投影树

类似地,只有当对应于频繁序列的节点生成时才对字典树进行扩展。此外,可以通过使用双层候选序列生成方式来提高算法的效率。在此方式中,第 $k-1$ 层的节点可以用于生成第 $k+1$ 层的候选序列。通过将数据库序列不断地沿着从根节点到第 $k-1$ 层的节点的路径进行投影操作,来确定候选序列的频度。由于序列节点可以通过项目集和序列扩展两种方式来进行扩展,因而必须保持两个动态项目集,其一用于项目集扩展,另一用于序列扩展。在投影过程中,不是这两个集合中的项目则被剔出。图 2 示例了一颗用于序列挖掘的投影树,节点〈(2)〉的动态项目集是{1,3},其中{1}是动态序列扩张,{3}是动态项目集扩展。

不同长度为 $k+1$ 的序列的频度的确定如下所述。每个节点,表示为序列 $s=\langle t_1,t_2,\cdots,t_m\rangle$,在第 $k+1$ 层保留四个计数矩阵。其一用于计算候选序列扩展$\langle t_1,t_2,\cdots,(t_m,i,j)\rangle$的频度,其中项目 i 和 j 是节点的动态项目集扩展;其二用于计算候选序列扩展$\langle t_1,t_2,\cdots,t_m,(i),(j)\rangle$的频度,其中项目 i 和 j 是节点的动态序列扩展;其三用于计算候选序列扩展〈t_1,

$t_2,\cdots,(t_m,i,j)\rangle$的频度，其中项目 i 和 j 是节点的动态序列扩展，并且项目 j 的字典序大于 i。最后一个矩阵用于计算候选序列扩展$\langle t_1,t_2,\cdots,(t_m,i),(j)\rangle$的频度，其中项目 i 是节点的动态项目集扩展，项目 j 是节点的动态序列扩展。

一旦每个序列都投影到第 $k-1$ 层上某个特定的节点，算法则立即更新相应矩阵的频度。可以通过使用稀疏矩阵使得更新效率得以提高。该矩阵的每行对应于某个项目集，而每列则对应于出现在序列中的不同项目。

5 并行序列模式挖掘

算法的目标机是分布式存储的并行计算机。我们假定消息传递是处理器间通信的唯一手段，还假定数据库太大，无法直接装入主存。这样的假定使得面临大规模数据集时算法具有较好的可扩展性。对于此类问题，一般有两种并行化的方法：一种是数据并行模式，另一种是任务并行模式。

5.1 序列挖掘的数据并行模式

数据并行模式的思想是将确定树中不同序列节点频度所需的计算量在不同的处理器间进行划分。下面给出并行算法的工作机制。令 P 是处理器的总数，则将原始数据库初始划分为 P 块大小相等的部分，每块分配给某个不同的处理器，并存储在其本地磁盘上。这些处理器协同工作，以先广扩展的方式每次将投影树扩展一层。而计算相应于树中第 $k+1$ 层节点的候选序列的频度分两步实施：第一步，每个处理器只是将数据库的本地子集投影到第 $k-1$ 层的节点，从而计算出序列的局部频度；第二步，通过全局规约操作将各自的频度累加起来，得到全局频度。各处理器利用这些全局频度值来确定第 $k+1$ 层的节点哪些满足最小支持度约束。应该注意的是，所有处理器使用的数据结构是相同的，与串行算法中所建立的树一样。

一般来说，由于数据库在各处理器间进行了等分，总的计算量会得到均衡分配。然而，如果某些处理器分配的子数据库所包含的序列模式相对较多，则会导致负载的不平衡。

5.2 序列挖掘的任务并行模式

投影树算法通过将数据库序列投影到第 $k-1$ 层的不同节点来计算第 $k+1$ 层的候选模式的频度。一旦数据库已经被投影到这些节点上，则实际的频度计算可以在每个节点独立地进行，于是第 $k-1$ 层各节点的计算量就成了独立的任务，通过将这些任务分配给不同的处理机从而进行算法的并行化。

可以使用水平和垂直模式来分配与树中不同节点相关联的任务。在水平模式中，每层的各节点依次生成，然后该层的节点任务被分配给不同的处理机。在不同层次的分解是不同的。而在垂直模式中，每个处理器分配到对应于 $k+1$ 层某些特定的节点的一系列任务，然后独立地生成扎根于这些节点的所有子树。在总体上，垂直模式在分布存储体系结构的并行机上的运行性能要优于垂直模式，而水平模式则更适合共享存储体系。下面我们就提出基于垂直模式的任务并行分解方式，该模式以下述方式在处理器间分配任务。首先，利用数据并行算法对投影树进行扩张，当扩展到第 $k+1$ 层后，第 k 层的不同节点被分配给各处理器，然后每个处理器就以分配给它的不同节点为树根来生成子森林。注意，算法是对第 k 层节点进行分配（而非第 $k+1$ 层），因为对第 $k+2$ 层的候选集进行频度计数时，需要将数据库投影到第 k 层节点上。

为了使每个处理器接下来能独立地进行工作，它必须访问数据库中可能会支持分配给该处理器节点所对应模式的序列。每个处理器 P_i 所要访问的数据库序列按下述方法确定。令 S_{nv} 为分配给 P_i 的节点的集合，A_i 为 S_{nv} 中所有动态项目节点的集合，B_{nv} 为 S_{nv} 中所有频繁序列节点所包含项目的集合。于是，每个处理器要访问所有满足以下条件的序列，该序列所包含的项目集是 $C_i=A_i\cup B_i$ 的子集。此外，由于它只计算对应于 S_{nv} 后辈节点所对应的频繁模式，因此只需保留原序列中出现在 C_i 中的项目。

在此算法中，一旦确定了第 k 层节点的分配方案，也就确定了集合 $C_0,C_1,\cdots,C_{p-1}$ 并以广播方式发送给各处理器，然后每个处理器开始读取数据库的本地部分，并将它切成 P 块，每个处理器一块，并将它发送到相应的处理器。当处理器收到与本地树中相匹配的序列时，则将它们写入盘中并独立地进行节点扩展。

5.3 并行算法的性能分析

一个并行算法的效率可以通过使用一系列诸如并行运行时间、加速比以及最大并发度等尺度来衡量。一个具有较好的可扩展性的并行算法，随着处理器数目的增加和问题规模的增加，仍然保持较高的并行效率。由于基于投影树的串行算法本身已很复杂，而且其运行时间取决于最小支持度和输入数据库的各种特性（比如不同项目的个数、所包含的频繁序列的个数等），因此文中在对并行算法的性能进行分析时做了一些必要的简化。

5.3.1 数据并行模式的算法性能分析

由于投影树的深度取决于最小支持度以及数据集的特性，我们的分析集中于在计算对应于投影树中第 $k+2$ 层的候选序列的频度时算法的并行效率。我们知道，第 $k+2$ 层的候选序列的频度的计算是通过将本地存储的数据库序列投影到第 k 层的节点上，并使用四个计算矩阵来计算其频度，然后通过全局规约操作将这些本地频度累加起来。这些矩阵的大小取决于树中每个节点动态项目的个数，而且在最坏情况下其上限为 $|I|$（I 表示数据库中不同项目的集合）。因此，令 m_k 是树中第 k 层节点的个数，则规约操作中涉及的数据量为 $O(|I|^2m_k)$，远大于处理器的数量，于是每个处理器在规约操作过程中的通信开销为 $T_1=O(|I|^2m_k)$。

串行算法用于计算第 $k+2$ 层候选序列频度所需的时间 T_2 取决于数据库中的序列数、不同项目的个数以及第 k 层节点的个数 m_k，即 $T_2=f(|D|,|I|,m_k)$。注意到 m_k 取决于 $|I|$ 以及最小支持度 σ，随着 $|I|$ 的增加或 σ 的减少，序列模式的数量也将增加，于是 m_k 也将增加。由于数据库中的每个序列都有可能被投影到第 k 层的节点上，因此在最坏的情况下，$f(|D|,|I|,m_k)=O(|D||I|^2m_k)$。而由于将数据库投影到某个节点上所需的工作会被其他有同样先辈的节点分担，于是 T_2 可表示为 $O(|D|^\alpha|I|^2m_k)$，其中 $0<\alpha<1.0$。虽然难以估算 α 的值，但将某个特定的序列投影到树中第 k 层的所有节点上所需的总工作量对于串行和并行算法是相同的。

令处理机的个数为 p，由以上的讨论可知，将数据库投影到第 k 层上的节点以及确定第 $k+2$ 层节点的频度所需的时间理论上为：

$$T_p=\frac{T_2}{p}+T_1=\frac{O(|D|^\alpha|I|^2m_k)}{p}+O(|I|^2m_k)$$

5.3.2 任务并行模式的算法性能分析

基于投影树算法的任务并行模式的性能主要取决于进行

（下转第 257 页）

Optimized Fingerprinting Tool for Highly Constrained Data // ACM Workshop on Multimedia and Security (MMSec). New-York, USA, August 2005:143-155

[8] Li Yingjiu, Swarup V. Fingerprinting Relational Databases -Schemes and Specialties. IEEE Transaction on Dependable and Secure Computing, 2005, 2(1):34-45

[9] Agrawal R, Kiernan J. Watermarking Relational Databases // The 28th VLDB Conference. Hong Kong, China, 2002

[10] 王彦,吕述望,徐汉良. 一种二进制数字指纹编码算法. 软件学报,2003,14(6):1171-1171

[11] Yen J C. Watermarks Embedded in the Permuted Image // The 2001 IEEE International Symposium on Circuits and Systems (ISCAS 2001). Sydney, Australia, 2001

[12] Sion R, Atallah M, Prabhakar S. Rights Protection for Relational Data. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(6):1509-1525

[13] Le Jia-jin, Zhu Qin, Zhu Ying. A Novel Robust Scheme of Watermarking Database // The 2nd International Conference on Software and Data Technologies (ICSOFT). Barcelona, Spain, 2007

(上接第 251 页)

任务分解以及在处理器间重新划分数据库所需的时间。一旦任务分配方案确定后,算法才进行数据库的再划分,再划分的通信开销取决于每个处理器所要接收的数据库的规模。在理想状态下,由于每个处理器平均收到的数据量大小为数据库的$\frac{1}{p}$,因此通信时间为$O(\frac{|D|}{p})$。然而,由于总会有部分的数据重叠,实际的通信开销可能会高一些。为了便于分析,我们假定每个处理器需要接收的数据库序列的平均数为$\beta\frac{|D|}{p}$($\beta \geqslant 1$),于是任务分解所需的时间为 $T=O\left(\beta\frac{|D|}{p}\right)$。而每个处理器 P_i 生成子森林所需的时间 T_i 取决于本地数据库的大小$\beta\frac{|D|}{p}$、其后辈节点的总数 d_i 以及 $|I|$(I 表示数据库中不同项目的集合),推理可知:

$$T_i=f(\frac{\beta|D|}{p},|I|,d_i)=O((\frac{\beta|D|}{p})^{\alpha}|I|^2d_i)$$

若 $d_i|I|>>1$,则算法用于任务分解的时间远小于处理器生成本地子森林的时间,这时任务并行模式将是非常有效的。

6 实验结果与分析

6.1 实验环境

机器类型:曙光 TC1700 并行机;节点数:4;处理器个数:8(每个节点有 2 个处理器);处理器的内存为:512;操作系统:Linux;计算环境:MPI;数据库规模为:200k;最小支持度:1%;网络环境:千兆以太局域网。

6.2 实验结果及分析

本实验进行两组测试:一组是串行的数据挖掘算法,另一组是采取本文提出的并行的数据挖掘算法。具体实验结果如表 1 所示。

表 1 串行与并行数据挖掘算法的执行时间比较

处理器个数	串行数据挖掘算法		并行数据挖掘算法	
	数据执行时间	任务执行时间	数据执行时间	任务执行时间
2	34.69	32.17	34.69	32.17
4	34.67	32.17	19.49	16.82
6	34.67	32.16	14.09	12.16
8	34.66	32.16	10.76	9.12

由上表可以看出,随着处理器数量的增加,两种并行算法的执行时间都明显减少,并取得了较好的加速比。同时我们注意到,随着处理器数目的增加,数据并行模式的并行效率随着下降,这是因为分配给每个处理器的工作量在减少,而归约操作所带来的通信开销是固定的;而在任务并行模式中,只要有足够多的节点和项目集,就能获得较好的负载均衡。

结束语 序列模式挖掘有着广泛且重要的应用前景。因为序列模式挖掘所面临的数据量往往非常大,所以整个系统的存储容量和挖掘效率就显得至关重要。为了进行有效挖掘,高效的并行算法尤为必要。本文提出了一种并行的序列模式数据挖掘算法。通过理论分析与实验验证可知:本文提出的并行数据挖掘算法,在数据量较大、多个处理器计算的秉性环境下,能很好地提高数据的挖掘效率。

参 考 文 献

[1] Jovanovic N, Milutinovic V, Obradovic Z. Foundations of predictive data mining; Neural Network Applications in Electrical Engineering // 2002 6th Seminar on. 2002:53-58

[2] Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: the RPRO Palgorithm // Neural Networks, 1993, IEEE International Conference on. 1993:586-591

[3] Onoda T. Neural network information criterion for the optimal number of hidden units. Neural

[4] Folino G, Pizzuti C, Spezzano G. Improving Induction Decision Trees with Parallel Genetic Programming[C] // Parallel, Distributed and Network-basedProcessing. Proceedings 10th Euromicro Workshop on. 2002

[5] Han E, Karypis G, Kumar V. Scalable parallel algorithms for mining association rules. IEEE Transactions on Knowledge and Data Eng., 2000, 12 (3)

[6] Zaki M J. An efficient algorithm for frequent mining sequences. Machine Learning Journal, 2001, 42:31-60

[7] Per J, Han J, Mao R. CLOSET: An efficient algorithm for mining frequent closed itemsets // Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD00). Dallas, TX, May 2000:11-20

[8] 王珊,等编著. 数据仓库技术与分联机分析处理. 北京:科学出版社,1998:1-17

[9] Wilkinson B, Allen M 著. 并行程序设计. 陆鑫达,等译. 机械工业出版社,2002:20-37

[10] 冯百鸣,经彤. BP 算法并行程序的自动生成与并行效率预测. 电光与控制,1997(2):1-5

[11] 任立勇,卢显良. 基于串-并行计算 BP 网络拓扑结构的研究与实现. 电子科技大学学报,2000,29(2):197-200