分布式序列模式发现算法的研究*

邹 翔[1,2+], 张 巍[2], 刘 洋[2], 蔡庆生[2]

[1](公安部第三研究所 科研中心,上海 200031)

[2](中国科学技术大学 计算机科学系,安徽 合肥 230027)

Study on Distributed Sequential Pattern Discovery Algorithm

ZOU Xiang[1,2+], ZHANG Wei[2], LIU Yang[2], CAI Qing-Sheng[2]

[1](Research Center, The Third Research Institute of Ministry of Public Security, Shanghai 200031, China)

[2](Department of Computer Science, University of Science and Technology of China, Hefei 230027, China)

+ Corresponding author: Phn: +86-21-64336810, Fax: +86-21-64333563, E-mail: xiangz@ustc.edu, http://www.trimps.ac.cn

Received 2003-11-13; Accepted 2005-02-03

Zou X, Zhang W, Liu Y, Cai QS. Study on distributed sequential pattern discovery algorithm. *Journal of Software*, 2005,16(7):1262−1269. DOI: 10.1360/jos161262

Abstract: Algorithm FDMSP (fast distributed mining of sequential patterns) is proposed in order to deal with mining sequential patterns in distributed environment and its properties are analyzed. The algorithm utilizes prefix-projected technique to divide the pattern searching space, utilizes polling site associated with prefix to get a global support, and utilizes local pruning, poll pruning and count pruning to decrease candidate sequences. It is divided into three sub-procedures which run asynchronously. As a result, the algorithm has lower I/O cost, memory cost and communication cost, and global sequential patterns are generated with higher efficiency. The experiments show that it outperforms the algorithm GSP after centralizing data by 68.5% to 99.5% and scaleable over LAN with huge amount of data.

Key words: data mining; sequential pattern; distributed algorithm

摘 要: 提出算法 FDMSP(fast distributed mining of sequential patterns),以解决分布式环境下的序列模式挖掘问题.首先对分布式环境下序列模式的性质进行了分析.算法采用前缀投影技术划分模式搜索空间,利用序列模式前缀指定选举站点统计序列的全局支持计数,利用局部约减、选举约减、计数约减等方法减少候选序列数,同时将算法分为 3 个子过程异步运行,使得算法具有较低的 I/O 开销、内存开销和通信开销,从而高效地生成全局序列模式.实验结果显示,在具有海量数据的局域网环境中,FDMSP 算法的性能优于将数据集中后采用 GSP 算法 68.5%~99.5%,并且 FDMSP 算法具有良好的可伸缩性.

关键词: 数据挖掘;序列模式;分布式算法

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.70171052, 90104030 (国家自然科学基金)

作者简介: **邹翔**(1977−),男,安徽马鞍山人,博士,助理研究员,主要研究领域为机器学习,知识发现;**张巍**(1975−),男,博士,主要研究领域为机器学习,知识发现;**刘洋**(1978−),男,硕士,主要研究领域为知识发现;**蔡庆生**(1938−),男,教授,博士生导师,主要研究领域为人工智能,知识发现.

序列模式(sequential pattern)的发现是数据挖掘研究的重要内容.在预测、生物医学、市场分析及通信网络分析等领域具有广泛的应用前景.

随着计算机应用技术的不断推广和使用的不断深入,海量的数据使得传统的单计算机系统在功能和性能上已不能满足对数据处理能力的需要;由网络联接多台计算机所构成的分布式系统已成为当今的主流系统,因此数据往往呈分布式状态.对于分布式的序列数据,在各单机系统上执行挖掘算法所得到的序列模式只是针对局部数据有效的,无法获得对分布式系统中所有数据全局有效的序列模式.一种直接的方法是将分布式环境中所有数据集中在一台计算机上执行挖掘算法,但在具有海量数据的分布式环境中,这将造成巨大的通信开销.如何有效地处理分布式环境下的序列模式发现问题,成为当前序列模式挖掘研究中亟待解决的问题.

1 问题定义

分布式环境下的序列模式挖掘问题的形式化描述如下.

设分布式环境中存在 m 个站点 $S_1,S_2,\ldots,S_m$,每个站点都是一台独立的计算机,站点之间通过网络互联.以 S 代表所有站点的集合,$S=\{S_1,S_2,\ldots,S_m\}$.站点 $S_i(i=1,2,\ldots,m)$上的数据序列集合记为 $db_i(i=1,2,\ldots,m)$,所有站点上数据序列的集合记为 DB,$db_i\subset DB$,$db_1\cup db_2\cup\ldots\cup db_m=DB$ 且 $db_1\cap db_2\cap\ldots\cap db_m=NIL$.

站点上的数据序列记为(Seq_ID,Trans_List),其中 Seq_ID 代表序列标识,Trans_List 是事务列表,包含多个按时间或其他顺序排列的事务.$Trans\_List=(Trans_1,Trans_2,\ldots,Trans_n)$,$Trans=(trans\text{-}time,Itemset)$,其中 trans-time 代表事件发生时间或其他顺序标识,Itemset 是一个项目集.$Itemset=(i_1,i_2,\ldots,i_n)$,$i_k\in T(k=1,2,\ldots,n)$,$T$ 代表所有项目或事件的集合.列表中的事务按 trans-time 的升序排列.序列(sequence)记为$\langle se_1,se_2,\ldots,se_n\rangle$,$se_k(k=1,2,\ldots,n)$代表一个项目集.设序列 $A=\langle se_1,se_2,\ldots,se_n\rangle$,序列 $B=\langle se'_1,se'_2,\ldots,se'_m\rangle$,若存在 $i_1<i_2\ldots<i_n$ 使得 $se_1\subseteq se'_{i_1}$,$se_2\subseteq se'_{i_2}$,…,$se_n\subseteq se'_{i_n}$,则称序列 A 是序列 B 的子序列.若 se 包含在一个数据序列中,我们称该数据序列支持 se.

一个序列 s 的支持计数 $count(s)$记为所有包含 s 的数据序列的总数.s 的支持度 $supp(s)$记为所有包含 s 的数据序列的总数与数据库中的数据序列总数之比.在分布式环境中,站点 $S_i(i=1,2,\ldots,m)$上包含 s 的数据序列总数称为 s 在站点 S_i 上的局部支持计数,记为 $count_i(s)$;分布式环境中包含 s 的数据序列总数称为 s 的全局支持计数,记为 $count(s)=\sum_{i=1}^{m}count_i(s)$.最小支持度 min$Supp$ 是一个阈值,一般由用户指定,全局最小支持计数 min$Count$=$|DB|\times\min Supp=(|db_1|+|db_2|+\ldots+|db_m|)\times\min Supp$.满足 $count(s)\geq\min Count$,称之为全局频繁序列或全局序列模式(global sequential pattern);局部序列模式(local sequential pattern)满足与全局序列模式相同的最小支持度 min$Supp$,在站点 S_i 上的局部最小支持计数记为 $\min Count_i=|db_i|\times\min Supp$.

以 F_G 代表分布式环境中所有序列模式的集合,对任意 $se\in F_G$,满足 $count(se)\geq\min Count$.以 F_{Li} 代表站点 S_i 上局部序列模式的集合,对任意 $se\in F_{Li}$,有 $count_i(se)\geq\min Count_i$.

我们认为,有效的分布式序列模式挖掘算法必须具备以下条件:

(1) 挖掘结果是有效的,即分布式序列模式挖掘算法最终得到的全局数据序列集合必须与在单计算机系统中将所有数据集中执行现有算法所得到的数据序列集合完全一致.

(2) 在分布式环境中,一般网络速度相对于站点计算速度要慢得多.因此,分布式序列模式挖掘算法必须具有较小的通信开销.

(3) 对于具有海量数据的分布式环境,分布式序列模式挖掘算法应当具有比将所有数据集中到单计算机系统中执行现有算法相同或更好的性能.

2 相关研究

文献[1]给出了序列模式挖掘的定义,文献[2]提出了一种泛化序列模式挖掘算法 GSP,文献[3]考虑了挖掘循环关联规则的方法,文献[4]提出了一种基于约束的序列模式挖掘方法,文献[5,6]提出了基于前缀投影的序列模式挖掘算法 Freespan 和 Prefixspan,文献[7]提出了采用深度优先搜索策略生成候选序列的序列模式挖掘算法 SPAM,文献[8−10]提出了几种有效的序列模式维护算法,解决序列模式的增量式更新问题,文献[11]提出了基于

树投影技术的两种不同的并行算法来解决分布内存并行计算机的序列模式发现问题,文献[12]提出了算法pSPADE处理共享内存计算机上的序列模式发现问题.

文献[13]提出了FDM(fast distributed mining of association rules)算法以解决分布式环境中的关联规则挖掘问题,该方法利用在局部频繁项目集与全局频繁项目集之间存在的性质来减少需要传输的信息量,并利用散列方法指定局部频繁项目集的轮询站点缩小了所需的通信次数,从而快速、有效地生成全局频繁项目集.

文献[14]提出了CDM(collective data mining)框架以解决分布式环境中的分类学习问题,该文指出对分布式环境中的站点直接应用现有机器学习和统计算法生成的局部模型可能是不正确的,与全局模型不一致.该方法使用正交基函数进行局部分析,再将所有局部生成的正交基函数组合,形成全局数据模型.

3 分布式环境下序列模式的性质分析

定义 1. 对于站点 $S_i(i=1,2,\ldots,m)$上的一个局部序列模式 se,如果 se 同时也是全局序列模式,我们称 se 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 se,存在站点 S_i,se 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:$Count_i(se)<\min Count_i(i=1,2,\ldots,m)$.因此,$DB$ 中包含 se 的序列总数为 $Count(se)=Count_1(se)+Count_2(se)+\ldots+Count_m(se)<\min Count_1+\min Count_2+\ldots+\min Count_m=\min supp\times(|db_1|+|db_2|+\ldots+|db_m|)=\min supp\times|DB|$,则 se 不满足最小支持度,$se\notin F_G$,故假设不成立.se 都是 s_i 上的 *gl-seq*.由Apriori性质可知,se 的所有子序列都是 S_i 上的 *gl-seq*. □

定义 2. 长度为 $k(k=1,2,\ldots)$的序列模式,记为 Lk 序列模式.

定义 3. 满足最小支持度的所有序列模式可以构成一棵序列树,树的根标记为 *NIL*,第1层为 $L1$ 序列模式,第2层为 $L2$ 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 $L+1$.序列树根据 $L1$ 序列模式可以划分为多个子树,我们称这些子树为 $L1$ 子树;相应地,长度为 k 的序列模式所对应的子树记为 Lk 子树.

定义 4. 我们将各站点上局部序列模式构成的子树称为局部子树(local subtree),而将全局频繁序列所构成的子树称为全局子树(global subtree).对任意一个全局 $L1$ 序列模式 x,对应的全局 $L1$ 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 $L1$ 子树记为 $\{x\}$-seq_i,对应的 $L1$ 投影数据库记为 $\{x\}$-DB_i.将所有局部 $L1$ 子树的集合记为 $UL1$,$UL1=\bigcup_{x\in L1}\bigcup_{i=1}^{m}\{x\}\text{-}seq_i$.

定理 1. 所有全局序列模式的集合 F_G 是所有局部 $L1$ 子树的集合 $UL1$ 的子集.

证明:$UL1=\bigcup_{x\in L1}\bigcup_{i=1}^{m}\{x\}\text{-}seq_i=\bigcup_{i=1}^{m}\bigcup_{x\in L1}\{x\}\text{-}seq_i$,$\bigcup_{x\in L1}\{x\}\text{-}seq_i$ 即为站点 S_i 上的所有局部 $L1$ 子树集合;根据引理1,$\forall se\in F_G,\exists S_i,se\in\bigcup_{x\in L1}\{x\}\text{-}seq_i$,因此,$F_G\subseteq\bigcup_{i=1}^{m}\bigcup_{x\in L1}\{x\}\text{-}seq_i$,即 $F_G\subseteq UL1$.

4 分布式序列模式挖掘算法 FDMSP

本文提出算法FDMSP(fast distributed mining of sequential patterns)来解决分布式环境下的序列模式挖掘问题.其主要思想是:在分布式环境中的各站点上,利用前缀投影技术划分模式搜索空间,降低需扫描数据库的规模,生成局部序列模式;利用序列模式前缀指定选举站点降低通信开销;利用局部序列模式与全局序列模式之间存在的特殊性质减少候选序列数;多个子过程异步运行,提高算法的并行性,从而高效地生成所有全局序列模式.

4.1 采用前缀投影技术生成局部序列模式

文献[6]提出了基于前缀投影的序列模式挖掘算法PrefixSpan,该算法按以下步骤挖掘序列模式:

第1步.扫描 DB 一次,找到所有频繁项,即 $L1$ 序列模式.

第 2 步.将 *DB* 按照上一步中找到的序列模式作为前缀进行划分,形成多个投影数据库,例如:设⟨*a*⟩为 *L*1 序列模式,以⟨*a*⟩为前缀的投影数据库中的数据序列由 *DB* 中所有包含 *a* 的数据序列的 *a* 第 1 次出现之后的部分组成,称为 *L*1 投影数据库.

第 3 步.对各投影数据库的前缀进行扩展.不失一般性,我们认为 *DB* 中的项目存在一个字典顺序,按照这个字典顺序进行序列扩展,例如:对于以⟨*a*⟩为前缀的投影数据库,⟨*b*⟩,⟨*c*⟩为其余长度为 1 的序列模式,则其扩展的顺序为⟨(*a*)(*a*)⟩,⟨(*ab*)⟩,⟨(*a*)(*b*)⟩,⟨(*ac*)⟩,⟨(*a*)(*c*)⟩.扫描投影数据库一次,找到该前缀长度为 2 的序列模式,然后按照第 2 步对投影数据库作进一步划分,递归地进行挖掘.最终找到所有序列模式.

我们采用 PrefixSpan 算法,按字典序依次生成各个 *L*1 子树,即以位于该子树根节点所对应序列模式为前缀的所有序列模式的集合.并且,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的规模.

4.2 全局序列模式的生成

为了判断各站点上生成的局部序列模式是否为全局序列模式,需要得到这些局部序列模式的全局支持计数.

我们按照如下步骤生成全局 *L*1 序列模式:首先生成所有局部 *L*1 序列模式集合的并集;然后,各站点分别向所有其他站点广播并集中所有序列模式在该站点上的局部支持计数,各站点将这些局部支持计数累加起来,其和大于 min*Count* 的即为全局 *L*1 序列模式.

如果我们采用这种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 $\{x\}$-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,判断它们是否是全局频繁的.*X* 的选举站点与它是否在该站点上频繁无关,并且分配函数保证每个局部序列的选举站点是唯一的,相应计算一条局部序列的全局支持计数的通信次数是 $O(m)$.

选举站点收到所有站点发送的 *L*2 子树集合,将具有相同前缀的 *L*2 子树合并为一个 *L*2 子树.合并方法为:将每个 *L*2 子树看做是一个集合,执行集合的并操作,集合之间相同的节点的支持计数相加,同时记录每个节点的已知支持计数值的站点.对于合并后的 *L*2 子树中的每个节点,向其支持计数未知站点发送以其为根的子树的支持计数请求.

各站点接收到所有选举站点的计数请求,扫描相应局部 *L*1 投影数据库一次,得到序列计数,将结果传回选举站点.选举站点收到所有站点传回的计数值,生成相应的全局 *L*2 子树,并将其向所有其他站点广播.

在局部站点上,各局部 *L*1 子树将按字典序依次生成,每生成一个 *L*1 子树,我们就将其拆分为多个 *L*2 子树,并发送到相应的选举站点,由选举站点统计它们的全局支持计数;同时,局部站点上继续生成新的 *L*1 子树.利用分布式系统的内在并行性,同时执行通信过程和挖掘过程,提高了算法的效率和本地资源利用率.

4.3 候选序列的约减方法

我们将生成全局序列模式之前所有局部序列都称为候选序列,可以采用以下方法对候选序列进行约减:

(1) 在站点 S_i 对投影数据库的前缀按字典顺序进行序列扩展时,设当前处理的为局部 *L*1 序列模式 *x* 对应的 *L*1 子树.这时,对每条长为 *L* 扩展序列 *se*,如果 *se* 的第 2 个元素的字典序位于 *x* 之前,则根据 Apriori 性质,其自第 2 个元素开始的长为 *L*−1 的子序列必然存在于已经得到的全局 *L*1 子树中.如果不存在,则该扩展序列不可能为全局序列模式,可以将该扩展序列删除.这种约减称为局部约减(local pruning).

(2) 在选举站点发送序列计数请求之前,对于由该站点处理的任意序列 *se*,如果全局支持计数已知且小于 min*Count*,则该节点的所有子节点都不可能是全局序列模式,将该序列及其对应子树删除;如果某节点 *sec* 的全局计数未知,而其父节点 *se* 的全局支持计数已知,可以近似计算 *sec* 的全局支持计数.设 *U* 代表 *sec* 支持计数已知站点集合,*V* 代表计数未知站点集合,计算公式为 $count^*(sec)=\sum_{i\in U} count_i(sec)+\sum_{j\in V} count_j(se)$,$count_i(sec)\le count_i$

(*se*),则 $count(sec) \le count^*(sec)$,如果 $count^*(sec)$小于 min*Count*,则将该子节点及其对应子树删除.这类约减称为选举约减(poll pruning).

(3) 在各站点接收到所有选举站点发送的序列计数请求时,这些请求由一系列序列子树组成,对于投影数据库的每条数据序列,自子树的根向下扫描子树,如果某节点不是该数据序列的子序列,则该节点的所有子节点都不可能是该数据序列的子序列.这类约减称为计数约减(count pruning).

4.4 算法详细描述

算法 FDMSP 在分布式环境的所有数据站点 $S_i(i=1,2,\ldots,m)$上执行,由 3 个子过程组成,子过程 Main 负责生成局部序列模式和将其发送至相应选举站点,并启动其他两个子过程;子过程 Polling 负责对局部序列模式全局支持计数的统计;子过程 Reply 负责接收序列计数请求,扫描投影数据库,并将结果返回.在实现上,每个子过程可以作为一个线程,异步执行.

1. Main 子过程

输入:DB_i:在站点 S_i 上分布的数据序列集合.

输出:F_G:所有全局序列模式的集合.

方法:

(1) $F_G \leftarrow null$;
(2) *L*1-*seq-set*←*Generate_L*1-*seq*(); //以广播请求方式生成所有全局 *L*1 序列模式
(3) $F_G \leftarrow F_G \cup L1\text{-}seq\text{-}set$;
(4) Start thread Polling and Reply; //启动 Polling 子过程和 Reply 子过程
(5) For each *se*_1 in *L*1-*seq-set* do
(6) If *se*_1 is *gl-seq* of S_i then
(7) {*se*_1}-seq_i←*Generate_Local-L*1-*Tree*(*se*_1); //生成{*se*_1}-seq_i
(8) partition {*se*_1}-seq_i into *m* *L*2-*Tree-set*$_i$*s*; //根据其选举站点划分子树
(9) for *j*=1,2,…,*m* do send *L*2-*Tree-set*$_i$(*j*) to Site S_j; //将发送 *L*2 子树集合
(10) wait for thread Polling and Reply; //等待 Polling 子过程和 Reply 子过程运行结束
(11) return F_G.

在 Main 子过程中,第(1)~(3)步生成全局 *L*1 序列模式并将其加入结果集 F_G;第(4)步启动 Polling 子过程和 Reply 子过程;第(5)~(9)步对 S_i 上的每个长为 1 的 *gl-seq se*_1,生成相应的局部 *L*1 子树{*se*_1}-seq_i,同时使用 local pruning 方法进行序列约减,并根据选举站点将其划分为 *m* 个 *L*2 子树,再将 *L*2 子树集合发送到相应的选举站点;第(10)步等待 Polling 子过程和 Reply 子过程运行结束;第(11)步输出 F_G.

2. Polling 子过程

方法:

(1) for *j*=1,2,…,*m* do
(2) receive *L*2-*Tree-set*$_j$(*i*); //接收各站点发送的 *L*2 子树集合
(3) merge(*L*2-*Tree-set*$_j$(*i*), *M_L*2-*Tree-set*$_i$(Prefix(*L*2-*Tree-set*$_j$(*i*))); //进行子树合并
(4) remove all node using poll proning;
(5) for each tree *MT* in *M_L*2-*Tree-set*$_i$ do //生成计数请求集合
(6) for each node *ND* in *MT* do
(7) insert *subtree*(*ND*) into all *RequestSet j* in *ND*.unknown_site;
(8) remove all *j* from unknown_site of *ND*.children;
(9) for *j*=1,2,…,*m* do send *RequestSet*(*j*); //发送支持计数请求
(10) for *j*=1,2,…,*m* do receive *ReplySet*(*j*); //接收计数结果
(11) for each tree *MT* in *M_L*2-*Tree-set*$_i$ do

(12) for each node *ND* in *MT* do

(13) if(*Count*(*ND*)<min*Count*) then remove(*subtree*(*ND*));//删除非全局频繁序列

(14) broadcast($M\_L2\text{-}Tree\text{-}set_i$). //广播生成的全局频繁序列

在 Polling 子过程中,第(1)~(3)步接收各站点发送的 *L*2 子树集合,将具有相同前缀的子树合并,结果存放于 $M\_L2\text{-}Tree\text{-}set_i$;第(4)步使用 poll pruning 对合并后子树进行约减;第(5)~(8)步对 $M\_L2\text{-}Tree\text{-}set_i$ 中的每个子树,以宽度优先策略遍历合并后子树的每个节点 *ND*,用集合 unknown_site 记录支持计数未知站点,将以它为根的子树插入 unknown_site 中各站点对应的计数请求集合 *RequestSet*(*j*),并将 *j* 从 *ND* 的每个子节点的 unknown_site 中删去,避免对同一节点多次发送计数请求;第(9)步向各站点发送支持计数请求;第(10)步接收各站点发送的计数结果;第(11)~(13)步删除非全局频繁序列;第(14)步广播当前得到的全局频繁序列.

3. Reply 子过程

方法:

(1) for *j*=1,2,…,*m* do receive *RequestSet*(*j*) and add to *RequestSet*; //接收计数请求

(2) for each data sequence *DS* in $\{x\}\text{-}DB_i$ do //设当前投影数据库前缀为 *x*

(3) for each subtree *ST* in *RequestSet* do

(4) Increment each node *ND* in *ST* if *ND* in *DS* //进行序列计数

(5) for *j*=1,2,…,*m* do send *ReplySet*(*j*); //向各站点发送应答

(6) for *j*=1,2,…,*m* do

(7) receive($M\_L2\text{-}Tree\text{-}set_j$); //接收全局频繁序列广播

(8) $F_G=F_G\cup M\_L2\text{-}Tree\text{-}set_j$. //加入输出集 F_G

在 Reply 子过程中,第(1)步接收各站点发送的计数请求,并将所有子树放入集合 *RequestSet*;第(2)~(4)步扫描投影数据库 $\{x\}\text{-}DB_i$ 一次,对 *RequestSet* 中的每个子树上的每个节点 *ND*,如果 *ND* 是当前数据序列 *DS* 的子序列(使用 count pruning 加快判断过程),则其计数值加 1;第(5)步向各站点发送应答;第(6)~(8)步接收 Polling 子过程发出的全局频繁序列广播并将其加入结果集 F_G.

5 算法的性能分析和实验评估

下面对算法运行的 I/O 开销、内存开销和通信开销进行分析.

在所有数据站点 $S_i(i=1,2,\ldots,m)$上,Main 子过程生成局部序列模式时,首先扫描 db_i 一次,生成局部 *L*1 序列模式;在通过广播请求生成全局 *L*1 序列模式之后,采用全部 *L*1 序列模式划分搜索空间,扫描 db_i 一次,生成相对于各全局 *L*1 序列模式的投影数据库.投影数据库规模随着候选序列长度的增加而减小.一般情况下,可以将当前正在处理投影数据库放入内存.另外,Reply 子过程接收各站点发送的序列计数请求后,需要扫描这些序列对应的 *L*1 投影数据库,可以将当前待扫描的 *L*1 投影数据库也放入内存.因此,对于所有数据站点 $S_i(i=1,2,\ldots,m)$上的数据序列集合 db_i,我们仅需扫描两次;而 *L*1 投影数据库的个数为全局 *L*1 序列模式的总数,对于每个 *L*1 投影数据库,我们需要扫描 3 次,一次用于发现新的频繁序列,一次用于根据局部 *L*2 序列模式生成 *L*2 投影数据库,一次用于获得各站点发送的序列在本站点的支持计数值;对于 $Lk(k>1)$投影数据库,我们仅需扫描两次.投影数据库规模随着候选序列长度的增加而减小,从而使得算法具有较低的 I/O 开销.

在内存中,存放着当前处理的投影数据库和待扫描的 *L*1 投影数据库,Polling 子过程生成的合并后 *L*2 子树集合 $M\_L2\text{-}Tree\text{-}set_i$,Reply 子过程接收各站点发送的需要计数的序列集合 $RequestSet_i$.由于 Main 子过程、Polling 子过程、Reply 子过程异步运行,实际运行中,上述投影数据库和数据结构一般不会同时出现在内存中.此外,局部约减、选举约减、计数约减进一步降低了候选序列的规模.因此,算法具有较低的内存开销.

对于每条候选序列,由于采用选举站点进行计数,其通信开销为 $O(m)$.局部约减和选举约减降低了需传输的候选序列数;另外,由于各选举站点处理具有不同前缀的候选序列,因此所生成的全局频繁序列具有不同的前缀,每条全局频繁序列仅广播一次.因此,算法具有较低的通信开销.

综上所述,算法 FDMSP 具有较低的时间和空间复杂度,并且具有较低的通信开销.

为验证算法的性能,我们选取几台微机组成的局域网进行了测试,所有微机的运行环境均为 Pentium III 800,Windows 2000 Professional,内存 256 M,网速 10Mb.实验数据采用 IBM Almaden 实验室的 Quest 项目提供的人工数据生成程序 assocgen(http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html),数据的各项参数为:数据库大小为 308M,数据序列总数为 1 800 000,平均序列长度为 10,平均项目集长度为 2.5,项目的总数为 10 000.采用随机抽样将数据分割存放在这些微机(站点)上.

我们采用算法 DGSP 与我们的算法进行比较,DGSP 算法分为两步:第 1 步将分布在各数据站点的数据序列通过网络集中到一个特定站点上(该站点不是数据站点);第 2 步采用 GSP 算法[2]对集中到该站点上的所有数据序列的集合进行挖掘.因此,算法 DGSP 的运行时间由两部分组成:数据集中时间和序列挖掘时间.

实验结果如图 1~图 3 所示.图 1 为站点数变化时算法 FDMSP 和算法 DGSP 执行时间的比较,最小支持度为 0.01.测试结果显示,站点数变化时,算法 DGSP 的运行时间变化不大,而算法 FDMSP 需要更少的运行时间,算法 FDMSP 的执行效率比算法 DGSP 提高 68.5%~99.5%.图 2 为最小支持度变化时算法 FDMSP 和算法 DGSP 执行时间的比较,站点数为 5.测试结果显示,随着最小支持度的增加,算法 FDMSP 的运行时间快速降低;算法 DGSP 的运行时间受数据集中时间影响,变化较为缓慢.图 3 为数据量变化时算法 FDMSP 和算法 DGSP 执行时间的比较,站点数为 5.测试结果显示,随着数据量的增加,算法 FDMSP 运行时间的增加明显少于算法 DGSP.因此,算法 FDMSP 具有良好的可伸缩性,适用于分布式海量序列数据的挖掘.

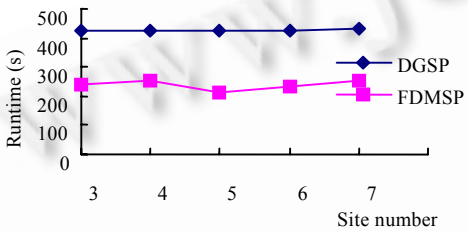

Fig.1 Runtime when site number varies

图 1 站点数变化时算法的运行时间

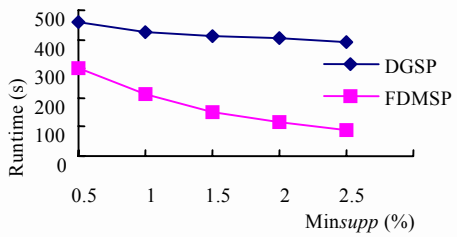

Fig.2 Runtime when min*supp* varies

图 2 最小支持度变化时算法的运行时间

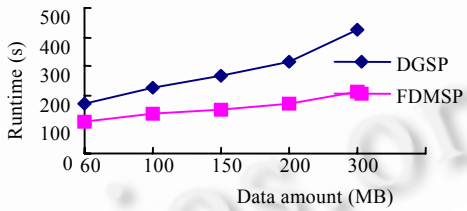

Fig.3 Runtime when data amount varies

图 3 数据集大小变化时算法的运行时间

从上面的实验结果我们得到,算法 FDMSP 明显优于算法 DGSP.因此,算法 FDMSP 满足第 1 节定义的分布式序列模式挖掘算法应当满足的第 3 个条件,即对于具有海量数据的分布式环境,分布式序列模式挖掘算法应当具有比将所有数据集中到单计算机系统中执行现有算法相同或更好的性能.

6 结 论

本文对在分布式环境中挖掘序列模式问题进行了研究,提出了一种称为 FDMSP 的分布式序列模式挖掘算法,采用前缀投影技术划分模式搜索空间,利用序列模式前缀指定选举站点统计序列的全局支持计数,利用局部约减、选举约减、计数约减等方法减少候选序列数,同时将算法分为 3 个子过程异步运行,使得算法具有较低的 I/O 开销、内存开销和通信开销,算法具有较高的运行效率.我们对算法的有效性和性能进行了证明和理论分析,实验结果也证明了上述结论.

今后的研究将着眼于在数据更新条件下的分布式序列模式挖掘问题以及算法在电子商务、网络入侵检测等领域的实际应用.

致谢 在此,我们向对本文的工作给予支持和建议的老师和同学表示感谢.

References:

[1] Agrawal R, Srikant R. Mining sequential patterns. In: Yu PS, Chen ASP, eds. Proc. of the 11th Int'l Conf. on Data Engineering. Washington DC: IEEE Computer Society Press, 1995. 3−14.

[2] Agrawal R, Srikant R. Mining sequential patterns: Generalizations and performance improvements. In: Apers PMG, Mokrane B, *et al*., eds. Proc. of the 5th Int'l Conf. on Extending Database Technology. Heidelberg: Springer-Verlag, 1996. 3−17.

[3] Ozden B, Ramaswamy S, Silberschatz A. Cyclic association rules. In: Proc. of the 14th Int'l Conf. on Data Engineering. 1998. http://citeseer.ist.psu.edu/ozden98cyclic.html

[4] Garofalakis M, Rastogi R, Shim K. Spirit: Sequential pattern mining with regular expression constraints. In: Atkinson MP, Orlowska ME, *et al*., eds. Proc. of the Int'l Conf. on Very Large Data Bases. Edinburgh: Morgan Kaufmann Publishers, 1999. 223−234.

[5] Han J, Pei J, Mortazavi-Asl B, Chen QM, Dayal U, Hsu MC. Freespan: Frequent pattern-projected sequential pattern mining. In: Ramakrishnan R, ed. Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2000. 355−359.

[6] Han J, Pei J, Mortazavi-Asl B, Pinto H, Chen QM, Dayal U, Hsu MC. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. of the 17th Int'l Conf. on Data Engineering. 2001. http://citeseer.ist.psu.edu/470226.html

[7] Ayres J, Gehrke J, Yiu T, Flannick J. Sequential pattern mining using a bitmap representation. In: Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2002. http://citeseer.ist.psu.edu/ayres02sequential.html

[8] Parthasarathy S, Zaki MJ, Ogihara M, Dwarkadas S. Incremental and interactive sequence mining. In: Gauch S, ed. Proc. of the 8th Int'l Conf. on Information and Knowledge Management. New York: ACM Press, 1999. 251−258.

[9] Masseglia F, Poncelet P, Teisseire M. Incremental mining of sequential patterns in large databases. Data & Knowledge Engineering, 2003,46(1):97−121.

[10] Zou X, Zhang W, Cai CS, Wang QY. A difficient incremental updating algorithm for discovering sequential patterns in large database. Journal of Nanjing University (Natural Sciences), 2003,39(2):165−171 (in Chinese with English abstract).

[11] Guralnik V, Garg N, Karypis G. Parallel tree projection algorithm for sequence mining. LNCS 2150, 2001. 310−320.

[12] Zaki MJ. Parallel sequence mining on shared-memory machines. Journal of Parallel and Distributed Computing, 2001,61:401−426.

[13] Cheung D, Han J, Ng VT, Fu AW, Fu YJ. A fast distributed algorithm for mining association rules. In: Proc. of the Int'l Conf. on Parallel and Distributed Inforamtion Systems. 1996. 31−44.

[14] Kargupta H, Park B, Hershberger D, Johnson E. Collective data mining: A new perspective toward distributed data mining. In: Advances in Distributed Data Mining. AAAI/MIT Press, 1999. http://citeseer.ist.psu.edu/article/kargupta99collective.html

附中文参考文献:

[10] 邹翔,张巍,蔡庆生,王清毅.大型数据库中的高效序列模式增量式更新算法.南京大学学报(自然科学版),2003,39(2):165−171.