文章编号：2096-1472(2018)-08-10-05
DOI:10.19644/j.cnki.issn2096-1472.2018.08.004

数据流中闭频繁项集的并行挖掘算法

冯忠慧，尹绍宏
(天津工业大学软件工程系，天津 300387)

摘　要：闭频繁项集包含了关于频繁项集的完整信息，可显著减少频繁项集挖掘所产生的模式数量，在一定程度上降低了内存开销、提高了时间效率。数据流的特性决定了它需要更高效的挖掘算法，为此使用分治策略，提出一种并行化闭频繁项集挖掘算法PCFI。该算法采用垂直数据格式存储项集的事务，通过对事务集的集合运算，可快速得到项集的支持度计数，合并具有相同事务集的频繁项，得到初始生成子，降低了搜索空间的规模。采用分治策略对初始生成子进行并行处理，得到约简前序集和约简后序集，在挖掘过程中不断地对每一生成子的搜索空间进行减枝，得到更小的约简后序集，从而减少对冗余数据的处理。实验分析表明，该算法的性能优于先前设计的算法。

关键词：数据流；滑动窗口；垂直数据格式；并行计算；闭频繁项集
中图分类号：TP311.5　**文献标识码**：A

Parallel Mining Algorithm of Closed Frequent Itemsets in the Data Stream

FENG Zhonghui,YIN Shaohong
(*School of Computer Science and Software Engineering*,*Tianjin Polytechnic University*,*Tianjin* 300387,*China*)

Abstract: The closed frequent itemsets contain complete information about frequent itemsets,which can significantly reduce the number of patterns generated by frequent itemsets mining,to a certain extent,decreasing the memory overhead and improving the time efficiency.The characteristics of the data stream determine that it needs a more efficient mining algorithm.To solve this problem,the paper proposes a parallel closed frequent itemsets mining algorithm,PCFI.This algorithm uses the vertical data format to store the items in a set.By collecting the set of transactions,the support counts of the items can be quickly obtained,and the frequent items with the same set of transactions are merged to obtain the initial generation and reduce the size of the search space.The partitioning strategy is adopted to process the initial generator in parallel,and the sets of pre-reduction sequence and the post-reduction sequence are obtained.In the mining process,the search space of each generator is continuously reduced,and the reduction sequence set becomes smaller,thus reducing the redundant data processing.Experimental analysis shows that the performance of this algorithm is superior to the previously designed algorithm.

Keywords:data stream;sliding window model;vertical data format;parallel computing;closed frequent itemsets

1 引言(Introduction)

数据流[1]是一串快速到达、无界的数据序列，广泛存在于日常生活各领域。数据流的特性决定了对其进行挖掘将面临着更多的挑战。首先，由于存储器的有限性，无法通过一次扫描存储所有传入的数据。其次，对于数据的处理效率也有更高的要求，即在新事物到来之前，需完成对当前时间段内数据的处理。因此，传统数据挖掘算法无法直接应用于数据流挖掘，需进行适当的改进和扩展。

尽管对于如何高效的挖掘频繁项集已取得不少研究成果[2-5]，但当最小支持度阈值设置过低或数据集存在长模式时，仍将会产生大量频繁项集，影响着数据流挖掘的时效性。针对此问题，1999年，Pasquier等人首次提出闭频繁项集概念[6]来减少存储空间和处理时间。闭频繁项集，规模小，包含频繁项集的完整信息，作为频繁项集的替代，可从大量的频繁项集中进一步提取有用知识，提高挖掘效率。

Chi等人首次提出基于滑动窗口的闭频繁项集挖掘算法，Moment算法[7]引入闭合计数树CET结构，用来检测闭频繁项集及与其他项集的边界，并通过CET中的相应节点类型转换来处理数据流的概念变化。该算法面临两个问题：其一，相对于闭频繁项集的数量而言，CET节点数目非常高。其二，以减少内存占用并加快对事物的搜索，采用FP树存储滑动窗口事务，并在此结构中搜索新的频繁项集。因此需要维护大

量节点，影响算法运行效率。

Lucchese等人提出的DCI−Closed算法[8]，通过生成子的前序集和后序集，来判定该生成子是否保序的方法，进行大量剪枝操作，减少了无效的检测，实现了对频繁项集格跳跃式的搜索闭频繁项集。优于CLOSET+算法[9]。宋威等人在此基础上，提出改进的DCI−CLOSED−INDEX算法[10]，采用二进制矩阵存储数据集，索引数组组织数据，并提出约简前序集和约简后序集概念，进一步的缩小了搜索空间。但通过递归的调用方式挖掘闭频繁项集使得效率低下。

数据流中闭频繁模式挖掘的研究问题主要集中在两点：(1)引入高效的数据结构存储数据集。提高空间利用率同时提高数据存储效率及数据处理效率。(2)算法本身性能的提升。本章通过对DCI−CLOSED−INDEX算法的改进，设计一种并行化的数据流闭频繁项集挖掘算法。引入垂直数据格式组织数据，通过集合运算对项集进行集合操作计数判断。对初始生成子采用分治策略，利用ForkJoin框架对其进行并行化处理。

2 相关知识(Related knowledge)

2.1 基本概念和性质

定义1：数据流$DS=\{T_1,T_2,T_3,\cdots,T_n\}$，由以一定速度连续到达的事务$T_i$组成。$I=\{i_1,i_2,i_3,\cdots,i_n\}$代表数据流中数据项的集合，事务$T_i$由数据流中部分或全体数据项构成，$T_i=\{i_1,i_2,i_3,\cdots,i_k\}$ $(1\leqslant k\leqslant n)$，对于任意$T_i$，都有$T_i\subseteq I$。

定义2：滑动窗口[11]$SW=\{S_0,S_1,S_2,\cdots,S_{w-1}\}$，构成数据流的一个瞬时抽样，窗口宽度$w(w>0)$，即窗口内包含事务集的数量，由用户预定义。随时间推进，新事务的到来，滑动窗口采取可重写循环方式不断更新窗口内数据。

定义3：闭项集。如果项集X的直接超集都不具有和它相同的支持度计数，则称项集X为闭项集。

定义4：闭频繁项集。给定最小支持度阀值min_sup，若闭项集X的支持度$\sup(X)\geqslant$min_sup，则X是一个闭频繁项集，其中$\sup(X)=|X|/W$，$|X|$表示窗口中事务集包含项集X的数量。

性质1：对于窗口中的任意频繁项集X，若$X\cap T_{\text{new}}=\emptyset$，则$T_{\text{new}}$的到来对于当前窗口中已存在的频繁项集无影响，其中$T_{\text{new}}$代表新事务。

性质2：如果项集$X\subseteq Y_{\text{closed}}$，则$\text{Sup}(X)=\text{Sup}(Y_{\text{closed}})$。

2.2 数据结构

2.2.1 事务矩阵

用一个$m\times n$的矩阵存储窗口中的事务，其中m代表事务量，n代表数据流中项的量。设定滑动窗口大小为w，则事务矩阵大小固定为$w\times n$。窗口未满时，按照事务到达次序，依次对矩阵赋值，若事务T_i包含项目ij，则将TM_{ij}置为1，否则置为0；窗口滑动后，采用新事务覆盖最旧事务的方式[12]，对事务矩阵进行更新。

2.2.2 垂直数据格式

对频繁的项集采用事务集表示，即{items:transactions}，其中items代表频繁项集，transactions代表包括items的事务集合。通过对事务集进行交集运算，便于进行支持度计数。同时采用该方式以紧凑方式进行存储，也可避免以bit值存储大量0所导致的空间浪费，对于数据的存储和处理在时空效率上均有所提高。

2.3 ForkJoin框架

ForkJoin[12]框架是由Java7提供的基于多核计算的并行处理框架，充分利用多核CPU的优势，提高程序处理能力。主要思想是通过设置阈值和工作线程数量，将总任务分割成多个子任务并行处理，其中工作线程数量根据CPU和任务需要进行设置，阈值取决于要进行划分的任务规模和工作线程数量。最后，通过对子任务的结果进行合并，可得到全局结果。其原理如图1所示。

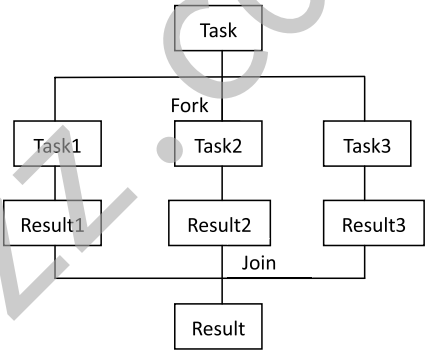

图1 ForkJoin框架工作原理

Fig.1 The working principle of ForkJoin frame

3 PCFI算法(PCFI algorithm)

该算法主要包括三部分：首先，使用事务矩阵存储滑动窗口数据，计算频繁1−项集，并按支持度计数降序排列以垂直数据格式结构进行存储。之后，对频繁1−项集进行处理，获得初始生成子。最后，将初始生成子分组以并行方式处理，计算约简前序集和约简后序集，得到局部闭频繁项集。合并子任务结果，得到全局闭频繁项集。

以表1所示事务数据流为例，介绍PCFI算法流程，设min_sup=0.4，w=5。

表1 事务数据流

Tab.1 Transactional data flow

Tid	Items
1	a,b,c
2	a,b,c,d
3	b,c,e
4	a,c,d,e
5	d,e
6	a,b,e
…	…

3.1 处理滑动窗口数据

表2为表1中的前五条数据构造事务矩阵，其中count一行记录项的支持度计数。

表2 事务矩阵

Tab.2 Transactional matrix

序号	a	b	c	d	e
1	1	1	1	0	0
2	1	1	1	1	0
3	0	1	1	0	1
4	1	0	1	1	1
5	0	0	0	1	1
count	3	3	4	3	3

扫描事务矩阵count一行，如果该值不小于最小支持度计数，则该项为频繁1-项集，可得到按支持度计数降序排列的频繁1-项集{c,a,b,d,e}。并以垂直数据格式的方式来存储频繁1-项集，如表3所示。其中TID-集表示包括该项集事务TID的集合。以项集c为例，被事务TID={1,2,3,4}所包含。

表3 频繁1-项集

Tab.3 Frequent 1-itemsets

项集	TID-集
c	1,2,3,4
a	1,2,4
b	1,2,3
d	2,4,5
e	3,4,5

3.2 初始生成子

初始生成子需满足两个条件：(1)没有直接超集的项，即闭频繁项集。(2)如果项集的事务集相同，则应将其合并。满足其一，即可作为初始生成子。对频繁1-项集矩阵进行处理，计算初始生成子。由于$Sup(ac)=Sup(a)$，$Sup(bc)=Sup(b)$，均不满足条件一，因此排除项a和b。经处理后可得到的初始生成子如表4所示，同时可得到闭频繁项集{c,d,e}。此步有效的减少了后期需要处理的数据量。

表4 初始生成子

Tab.4 Initial generator

初始生成子	支持度计数
c	4
d	3
e	3
d	2,4,5

3.3 并行挖掘闭频繁项集

全局闭频繁项集的挖掘部分以并行化方式执行。由于对每个初始生成子进行处理时相互之间不存在关联关系，所以根据分治策略，可将初始生成子分割成与CPU核数相匹配的子任务，对其并行处理。

3.3.1 前序集和后序集

以支持度计数降序排列的频繁1-项集矩阵为基准，位于某项之前的项归为该项的前序集，其后序项作为其后序集。以项b为例，其前序集为{c,a}，后序集为{d,e}。其中第一个项的前序集为空集，最后一个项的后序集为空集。

3.3.2 约简前序集和约简后序集

为缩减候选项集的规模，应对其用于扩展初始生成子的后序集进行约简，以降低后期处理的复杂度。因此引入约简前序集和约简后序集概念。

首先，对初始生成子的前序集进行约简。约简规则以保证约简前序集中项的事务集都不被其他项的事务集所包含。以初始生成子{e}为例，前序集{c,a,b,d},其中项a的事务集被项c事务集所包含，因此项a应被约简，同理约简项b，得到约简后序集{c,d}。

在得到约简前序集的基础上，对后序集进行约简。如果后序集中项的事务集不被约简前序集中项的事务集所包含，则可加入约简后序集，从而缩小了可扩展项的数量。

经以上步骤处理后得到初始生成子的约简前序集和约简后序集如表5所示。

表5 约简前序集和约简后序集

Tab.5 Pre-reduction sequence set and post-reduction set

初始生成子	约简前序集	约简后序集
e	c,d	∅
d	c	e
c	∅	a,b,d,e

3.3.3 全局闭频繁项集

设置ForkJoin框架的工作线程数量n，将初始生成子分成n组并行计算，充分利用多核CPU资源，提高运算速度。

当初始生成子的约简后序集为空时，将其直接归为闭频繁项集，结束扩展。如初始生成子e，其约简后序集为空集，则e为闭频繁项集。

否则，利用约简后序集以广度优先方式对初始生成子进行扩展以避免递归式的调用。如果当前项集支持度计数大于扩展集支持度计数，则将当前项集加入全局闭频繁项集；如果扩展集支持度计数不小于最小阈值，则将其加入生成子，继续对其扩展；由性质2可知，非频繁项集的超集均为非频繁项集，对于支持度计数小于最小阈值的项集停止扩展。重复以上步骤，直到不再有新生成子时结束，即可获得以当前初始生成子为前项的闭频繁项集。以初始生成子c为例，用项b对其扩展，得到$Sup(cb)=3$，由于$Sup(c)!=Sup(cb)$，则c为闭频繁项集，同时项集{cb}作为生成子，继续扩展，得到候选项集{cbd}，因为$sup(cbd)<min_sup$，停止对其扩展。

合并对初始生成子扩展挖掘后得到的闭频繁项集，即可得到全局闭频繁项集。如图2所示。

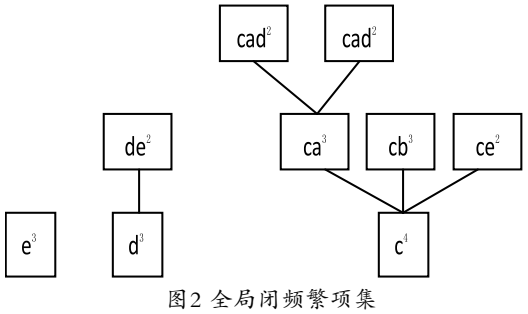

图2 全局闭频繁项集

Fig.2 Global closed frequent itemsets

3.4 PCFI算法描述

算法1：滑动窗口初始化，计算频繁1-项集。

输入：数据流DS，最小支持度min_sup，滑动窗口大小w。

输出：频繁1-项集

1　扫描DS前w条事务，存储到事务矩阵。

2　扫描事务矩阵最后一行

```
for(j=0;j<items;j++){
if(TM[last,j]>=min_sup)
    Fi.item.add(item);              /*加入项*/
    Fi.item.trans(transaction);     /*加入事务*/
}
```

3　按支持度计数降序排列频繁1-项集

4　输出以垂直数据格式存储的频繁1-项集矩阵

算法2：计算初始生成子。

输入：频繁1-项集矩阵

输出：初始生成子

```
for  each频繁1-项集
if(fi(i).trans和fi(j).trans相同)
/*合并两项，将合并后的项集作为初始生成子*/
if(fi(i).trans  fi(j).trans)
/*该项不能作为初始生成子*/
if(fi(i).trans不被任何项的事务集包含)
/*该项既是初始生成子，也是频繁1-项集*/
```

算法3：并行挖掘全局闭频繁项集。

输入：频繁1-项集，初始生成子，线程数量n

输出：全局闭频繁项集

1　将初始生成子分割成子任务

```
THRESHOLD=(end-start)/n;
canCompute=(end-start)<=THRESHOLD;
if(canCompute){/*核心处理部分2~6*/
}else{/*继续分割初始生成子*/
middle=(start+end)/2;
    ParallelProcessing(start,middle);
    ParallelProcessing(middle+1,end);
}
```

2　计算约简前序集

```
for初始生成子each前序集
if(pre_set(i).trans  pre_set(j).trans)
/*pre_set(i)被约简掉*/
```

3　得到约简后的前序集pre_set

4　计算约简后序集

```
for初始生成子each后序集
if(post_set(i).trans  pre_set(j).trans)
/*post_set(i)被约简掉*/
```

5　得到约简后的后序集post_set

6　挖掘全局闭频繁项集

```
if(post_set==NULL)/*将gen加入到全局闭频繁项集*/
else
    candidate.add(gen);
    for each candidate 用约简后序集进行扩展
      if(new_gen.post_set==null)
            continue;
    if(new_gen.sup>=min_sup)
    /* 将candidate.add(new_gen);*/
if(gen.sup>new_gen.sup)
/*将gen加入到全局闭频繁项集*/
```

4 实验结果及分析(Experimental results and analysis)

本文算法采用Java语言实现，以MyElipse为开发平台，部署Fork-Join框架，采用双核的并行方式对算法进行性能测试。实验环境为Windows 7旗舰版操作系统、AMD A4-6210 APU with AMD Radeon R3 Graphics1.80GHz、4GB内存。

采用三组数据集来模拟数据流环境进行测试，分别是稀疏集T10I4D100K、稠密集T40I10D100K和Mushroom。

为测试PCFI算法的有效性，本文将DCI-CLOSED-INDEX算法扩展为数据流中闭频繁项集挖掘算法，标记为DCI-CLOSED-INDEX-DS。

4.1 不同数据集下运行时间的比较

设定滑动窗口大小为w=1000，最小支持度为min_sup=0.02，测试两种算法在稀疏集T10I4D100K上的运行时间对比，如图3所示。

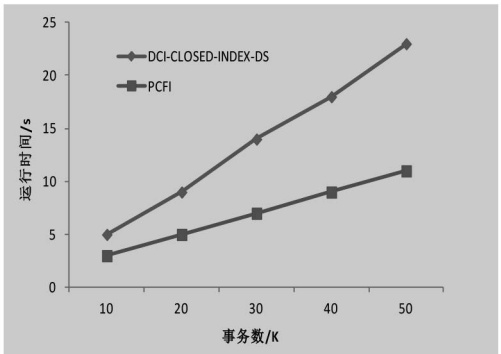

图3 T10I4D100K集两种算法的运行时间对比1

Fig.3 Running time comparison of two algorithms of T10I4D100K set 1

如图4所示，设定滑动窗口大小为w=1000,最小支持度为min_sup=0.05，测试两种算法在稠密集T40I10D100K上的运

行时间对比。

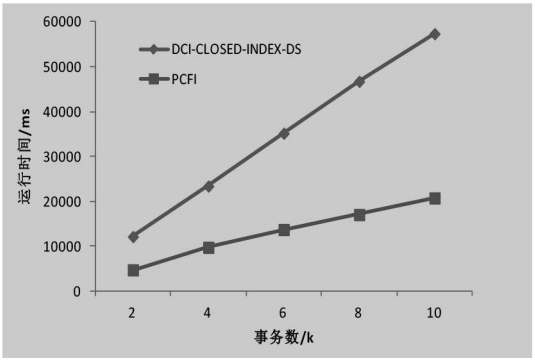

图4 T40I10D100K集两种算法的运行时间对比2

Fig.4 Running time comparison of two algorithms of T10I4D100K set 2

可以看出，两种算法在不同数据集下运行时间均随事务数的增多呈线性增长趋势，但相比之下DCI-CLOSED-INDEX-DS算法增长速度更快些，也说明了本算法的稳定性；在相同事务数下，无论是稀疏集还是稠密集，PCFI算法的时间效率均高于DCI-CLOSED-INDEX-DS算法。这是由于PCFI算法在挖掘闭频繁项集时避免了递归式的调用，降低了算法的时间复杂度；同时，在并行环境下，采用分治策略，利用多核CPU将初始生成子分成子任务并行处理，极大的提高了程序的执行效率。

4.2 不同支持度下运行时间的比较

如图5所示，设定滑动窗口大小为w=1000，采用数据集Mushroom测试两种算法在不同支持度下的运行时间对比。

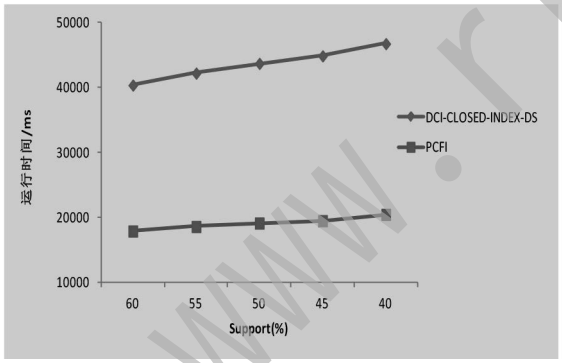

图5 不同支持度下两种算法的运行时间对比

Fig.5 Comparison of running time of two algorithms under different support degrees

图5显示当支持度设置较低时，PCFI算法运行时间比DCI-CLOSED-INDEX-DS快两倍多。随支持度变小，两种算法的运行时间波动均较小，可以反映出闭频繁项集的个数增加并不快，因此以闭频繁项集替代频繁项集也更适用于数据流中的频繁模式挖掘，能够以较快的运行速度挖掘出所需的完整信息。

5 结论(Conclusion)

本文针对频繁项集在数据流环境下挖掘效率低下的问题，对DCI-CLOSED-INDEX算法的改进和扩展，提出一种数据流中闭频繁项集的并行挖掘算法PCFI。该算法以垂直数据结构来存储频繁项集，通过事务集间的集合操作可以减少对事务矩阵的扫描，同时也能避免存储大量0元素对存储空间的浪费。根据频繁1-项集获取初始生成子，使用分治策略，与CPU核数及程序需求相匹配，对初始生成子进行分组并行计算，利用约简前序集得到约简后序集，对初始生成子进行扩展，直至扩展结束，将局部闭频繁项集合并得到全局闭频繁项集。实验结果表明，基于ForkJoin框架的PCFI算法在时间效率上，相对于已有算法有了明显的提高。

参考文献(References)

[1] Morales G D F,Bifet A,Khan L,et al.IoT Big Data Stream Mining[C].ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.ACM,2016:2119-2120.

[2] Aliberti G,Colantonio A,Pietro R D,et al.EXPEDITE:EXPress closED ITemset Enumeration[J].Expert Systems with Applications,2015,42(8):3933-3944.

[3] Subbulakshmi B,Dharini B,Deisy C.Recent weighted maximal frequent itemsets mining[C].International Conference on I-Smac.IEEE,2017:391-397.

[4] Huang J N,Hong T P,Chiang M C.An effective method for approximate representation of frequent itemsets[J].Intelligent Data Analysis,2017,21(3):597-616.

[5] 张伟,石纯一.Agent组织的一种递归模型[J].软件学报,2002,13(11):2149-2154.

[6] Meuer H,Simon H,Strohmaier E,et al.TOP500 supercomputer sites[EB/OL].http://www.top500.org,2011-10-15.

[7] Johnson T F,Tinoco E T,N.Yu N.Thirty years of development and application of CFD at Boeing commercial airplane seattle[R].USA:AIAA,2003:343.

[8] Zhang H,Li S K.Description logic of tasks: From theory to practice[J].Chinese Journal of Computers,2006,29(3):488-496.

[9] Wang J,Han J,Pei J.CLOSET+:searching for the best strategies for mining frequent closed itemsets[C].ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.ACM,2003:236-245.

[10] Graham S,Park C.Assignment of dual port memory banks for a cup and a host channel adapter in an infiniband computing node [P].US 6816889 B1,2004-10-09.

[11] Zhang Hui.Organizational coordinate behaviors modeling of virtual entity group[D].Changsha:National University of Defense Technology,2006.

[12] 梅杓春.新编英汉通信词典[M].南京:东南大学出版社,1996.

作者简介：

冯忠慧(1993-)，女，硕士生.研究领域：数据挖掘.

尹绍宏(1964-)，女，硕士，副教授.研究领域：数据挖掘.