一种基于后缀项表的并行闭频繁项集挖掘算法*

唐颖峰[1,2], 陈世平[1]
(1. 上海理工大学 管理学院, 上海 200093; 2. 上海对外经贸大学 教务处, 上海 201620)

摘　要: 对现有的基于 MapReduce 的并行频繁项集挖掘算法进行了研究,提出一种基于后缀项表的并行闭频繁项集挖掘算法,通过后缀项表的引入及以闭频繁项集挖掘的形式,减少组分间的数据传送量,提高挖掘效率。实验表明,该算法可以有效缩短平均挖掘时间,对于高维大数据具有较好的性能。

关键词: 频繁项集挖掘; 并行挖掘算法; MapReduce; 闭频繁项集; 后缀项表
中图分类号: TP31; TP301.6　　**文献标志码**: A　　**文章编号**: 1001-3695(2014)02-0373-05
doi:10.3969/j.issn.1001-3695.2014.02.013

Parallel closed frequent itemset mining algorithm with postfix-table

TANG Ying-feng[1,2], CHEN Shi-ping[1]
(1. *Business School*, *University of Shanghai for Science & Technology*, *Shanghai* 200093, *China*; 2. *Academic Affairs Section*, *Shanghai University of International Business & Economics*, *Shanghai* 201620, *China*)

Abstract: Based on current frequent itemsets mining using parallel FP-Growth algorithm with MapReduce framework, this paper proposed a parallel closed frequent itemsets mining algorithm with a postfix-table based on MapReduce framework. The algorithm generated closed frequent itemsets instead of all frequent itemsets. With a postfix-table structure, the algorithm could reduce the amount of data transfer between mappers and reducers efficiently. The experimental results show that the algorithm can shorten mining time efficiently. The algorithm has good performance especially in long- transction mode.

Key words: frequent itemsets mining; parallel mining algorithm; MapReduce; closed frequent itemsets; postfix-table

0　引言

关联规则挖掘是数据挖掘中的一个重要课题,最近几年已被业界广泛研究[1]。Agrawal 等人[2]于 1993 年提出了挖掘顾客交易数据库中项集间的关联规则问题,并提出了经典的 Apriori 算法。但是该算法存在多次遍历数据库及生成大量候选集等问题,使得算法在处理较大数据时效率低下。对此,Han 等人[3]于 2000 年提出了基于对 FP-Tree 进行挖掘生成频繁项目集的 FP-Growth 算法。该算法只需对数据库进行两次扫描,且不生成候选集,极大地提高了频繁项目集的挖掘效率。尽管如此,在海量数据处理的背景下,FP-Growth 的单机处理能力依然非常有限。因此,算法的并行化是发展的必然趋势。Zaïane 等人[4]于 2001 年提出了一种基于多处理器并行计算机的并行 FP-Growth 算法。此后又有多位学者提出一些基于并行计算机的 FP-Growth 算法[5,6]。但是并行计算机可扩展性较为有限,在一台计算机内扩展处理器数量及存储空间的做法代价昂贵。MapReduce[7]是在云计算背景下被提出的一种高扩展性和高容错性的并行计算框架。Li 等人[8]于 2008 年提出了一种 MapReduce 框架下的并行 FP-Growth 算法 PFP(parallel FP-Growth algorithm)。但是 PFP 算法中 mapper 需要向 reducer 复制传送大量事务数据,特别是当事务数据模式较长、最小支持度阈值较低时,组分间的数据传送量会成倍增长。对于 MapReduce 这样的分布式系统而言,网络带宽和磁盘 I/O 是其瓶颈所在,巨大的数据传送量势必对其产生巨大压力,导致系统效率大大降低。PFP 需要传送大量数据的一个重要原因在于,其试图挖掘事务数据中全部频繁模式,因此不得不对 F-List 中每一项的频繁模式进行搜索。当事务数据的模式较长时,对于 F-List 中的每一项都要传送大量组内依赖事务(group-dependent transactions)。此外,当支持度阈值较小时,挖掘算法会产生数量巨大的短频繁模式,导致存储和处理出现困难。因此在实际处理时(如 PFP 的 Mahout 实现[9])常采用取 TOP-*K* 频繁模式这种折中的方式。

闭频繁项集是频繁项集的一种无损压缩,它在保留频繁项集完整信息的前提下,可以显著减少频繁项集挖掘产生的模式数量[1]。Wang 等人[10]于 2012 年将闭频繁项挖掘算法融入 PFP,提出基于 MapReduce 的闭频繁项挖掘算法 PFP-C。但是 PFP-C 的数据传送方式与 PFP 相同,因此并未解决数据传送量大的问题。

本文在 PFP 的基础上提出一种基于后缀项表的并行闭频繁项挖掘算法 PFP-P(parallel FP-Growth algorithm with postfix-table),由闭频繁项挖掘的特点,引入后缀项表的概念;通过后缀项表的构建,删去 F-List 中不必要传送数据的项,有效地缩减了处理过程中的数据传送量;且在最小支持度阈值较低时,只产生少量闭频繁项集,可保留频繁项集的完整信息。

收稿日期: 2013-05-13; **修回日期**: 2013-06-27　　**基金项目**: 国家自然科学基金资助项目(61170277);上海市教委科研创新重点项目(12zz137);上海市一流学科建设项目(S1205YLXK)

作者简介: 唐颖峰(1983-),男,甘肃嘉峪关人,博士研究生,主要研究方向为云计算、数据挖掘(tangyf1983@yahoo.com.cn);陈世平(1964-),男,博导,博士,主要研究方向为云计算、计算机网络通信、信息检索.

1 PFP 算法

PFP 算法是由 Google 北京研究院的 Li 等人于 2008 年提出的一种基于 MapReduce 框架的并行 FP-Growth 算法。得益于 MapReduce 框架的高扩展性以及高容错性,算法在处理大数据的具有较好的表现。算法的基本思想是:将事务数据库变换成一个新的组内依赖事务数据库,并将其分发给对应的 Reducer,在 Reducer 递归构建条件 FP-Tree 的过程中,通过不同的组内依赖事务生成的本地 FP-Tree 是相互独立的。

1.1 PFP 算法描述

PFP 算法将数据库分成 P 个分块,由 P 个 mapper 进行处理。该算法共需进行三次 MapReduce 操作,分别为:

a)扫描一遍数据库,生成一阶频繁项集 F-List,并对项集按支持度大小进行排序。将 F-List 中的 i 个频繁项分成 Q 个组,称为 G-List,每个组分配一个 gid。

b)再次扫描数据库,将每条事务中的项按 F-List 中的支持度大小进行排序,并将事务中不存在于 F-List 中的项删除。

c)对处理后的数据进行并行的 FP-Growth 挖掘。

步骤 c)的 MapReduce 操作中:

(a)对于 mapper

①由 G-List 生成一个 hash 表 H,表的 key 为频繁项,表的 value 为频繁项所对应的 gid。

②遍历每条事务数据,在 H 中查找事务数据中的项,并根据项对应的 gid 把该项左边的全部项分发到对应的组;然后删除 H 中该 gid 对应的行,防止该项左边的数据重复分发。

(b)对于 reducer

①每个 reducer 获取 G-list 中的一个组,并根据组的 gid 拖取 mapper 分发完成的数据。

②拖取完成后,根据拖取的数据构建 FP-Tree,对 FP-Tree 调用 FP-Growth,生成 K 阶频繁项集。

步骤 c)MapReduce 操作的伪代码见算法 1 和 2。

算法 1 PFP 算法步骤 c)中的 Map 算法

```
void Map (key k,value T_i)
{   G-List. Load( );
    HashTable H = G-List. getHashTable( );
    Item a[ ] = T_i. Split( );
    for j = a. length-1 to 0
    {   HashNum = H. getHashNum( a[j] );
        if( HashNum!  = Null)
        {   H. deleteRowByVal( HashNum);
            OutPut(〈HashNum, a[0] + a[1] + ⋯ + a[j]〉);
            }
    }
}
```

算法 2 PFP 算法步骤 c)中的 Reduce 算法

```
void Reduce (key gid,value DB_gid)
{   G-List. Load( );
   nowGroup =  G-List. getGroupByGid( gid);
   LocalFPtree = new FP-tree( Null);
   foreach T in DB_gid
     LocalFPtree. insert_tree( T,LocalFPtree. root,G-List);
     foreach item in nowGroup
     {   Heap HP = new Heap( );
      LocalFPtree. FPGrowth( LocalFPtree. root,item, HP);
      foreach pattern in HP
          OutPut(〈null,pattern + pattern. supp( )〉);
     }
}
```

1.2 PFP 算法分析

由于组分间的数据通信是分布式系统的最主要瓶颈,因此本文着重分析算法在执行过程中的数据传送量,即数据通信复杂度。设数据库中事务数据数为 n,第 i 个事务的长度为 l_i,则整个事务数据的数据量可表示为

$$M_{\mathrm{DB}} = \sum_{i=1}^{n} l_i \tag{1}$$

PFP 算法中步骤 a)计算 F-List 过程中的数据传送量为 MDB,步骤 b)只需要 Map 操作而无须 Reduce 操作,因此数据传送量为 0。

步骤 c)过程中需要传送的数据量可表示为

$$\begin{aligned} M_{\mathrm{Dup}} &= \sum_{i=1}^{n}(f(I_1 I_{l_i}) + 2f(I_2 I_{l_i}) + \cdots + l_i f(I_{l_i} I_{l_i})) = \\ &\sum_{i=1}^{n} l_i + \sum_{i=1}^{n}(f(I_1 I_{l_i}) + 2f(I_2 I_{l_i}) + \cdots + (l_i - 1) f(I_{l_i} I_{l_i})) = \\ &M_{\mathrm{DB}} + \sum_{i=1}^{n}\sum_{j=1}^{l_i - 1} j f(I_j I_{l_i}) \end{aligned} \tag{2}$$

其中:I 表示事务中的项。函数 $f(I_j, I_k)$ 定义为

$$f(I_j, I_k) = \begin{cases} 1 & I_j、I_k \text{ 不在同一 group 中,或 } I_j = I_k \\ 0 & I_j、I_k \text{ 在同一 group 中} \end{cases} \tag{3}$$

将步骤 c)的数据传送量加总,可知算法全过程的数据传送量为

$$M = M_{\mathrm{DB}} + M_{\mathrm{Dup}} = 2M_{\mathrm{DB}} + \sum_{i=1}^{n}\sum_{j=1}^{l_i - 1} j f(I_j I_{l_i}) \tag{4}$$

由式(3)和(4)可知,PFP 的数据传送量与 G-List 的分组相关。当分组数 $Q = 1$,即 G-List = F-List 时,$M = 2M_{\mathrm{DB}}$,取得最小值,但此时 PFP 会退化为一个单机算法。设 F-List 中项的个数为 m,当分组数 $Q = m$ 时,$M = (3/2)M_{\mathrm{DB}} + (1/2)\sum_{i=1}^{n} l_i^2$,取得最大值,但此时系统的并行度最高。

对于一个分布式系统而言,组分数越大代表着系统的并行度越高,处理效率越高。然而这样一般会同时导致数据传送量的增大。从对 PFP 算法的分析可知,当事务数据的平均长度 l 较长时,传送的数据量会随着组分数的增加急剧增大,导致系统运行效率降低,成为系统运行的瓶颈。

2 PFP-P 算法

PFP-P 算法的基本出发点是,以挖掘闭频繁项集来替代挖掘全部频繁项集。与挖掘全部频繁项集不同,挖掘闭频繁项集并不需要事务数据中全部项的组内依赖事务,而只需要部分后缀项的组内依赖事务即可,以此可以达到减少数据传送量的目的。此外,挖掘闭频繁项集的另一个好处是,在保证信息完整度的前提下,可以大量减少挖掘的结果数据,便于存储和进一步处理。

2.1 后缀项表

为方便讨论,本文将事务数据映射为事务模式,即将每条事务数据中的非频繁项删减后,再将按 F-List 中项的顺序进行排序后的事务数据,用 T 表示;事务 T 中的项用 I 表示;模式 T 及子模式的支持度用 sup( )表示,满足频繁模式的最小支持度用 sup_min 表示。

定义 1 闭频繁模式。设有子模式 $X = I_1, I_2, \cdots, I_k$, sup

$(X)=u\geqslant \sup_min$;若 $\forall I_p$,其中 $p\geqslant k+1$,均有 $\sup(I_1,I_2,\cdots,I_k,\cdots,I_p)<u$,则称 X 为项 I_k 的闭频繁模式,I_k 称为 X 的后缀。闭频繁模式与闭频繁项集一一对应。

定义 2　后缀项。对 T 的子模式 $X=I_1,I_2,\cdots,I_k$,满足 $\sup(I_1)=\sup(I_2)=\cdots=\sup(I_k)=u$,则有:

定义 3　若 $\forall I_p$,其中 $p\geqslant k+1$,均有 $\sup(I_p)<u$,则称 I_k 为 T 的 u 支持度后缀项。

定义 4　对于 X 中任意项 I_i 及任意后缀项 I,$I_i\neq I$,若均有 $\sup(I_iI)<u$,则称 I_i 为 T 的 u 支持度后缀项。

定理 1　闭频繁模式的后缀必为后缀项。

证明　用反证法证明。假设 T 的子模式 $X=I_1I_2\cdots I_k$ 为闭频繁模式,$\sup(X)=u$,且 I_k 不为后缀项;则根据定义 3,T 中必存在项 I_p,其中 $p\geqslant k+1$,满足 $\sup(I_p)\geqslant u$,由于 T 中项的支持度单调不增,故 $\sup(I_p)=u$;根据定义 4 可知,$\sup(I_kI_p)=u$;由此则 T 中必存在一子模式 $X'=I_1I_2\cdots I_kI_p$,使得 $\sup(X')=u$;根据定义 1,X 不是闭频繁模式,与假设矛盾,原结论得证。

根据定理 1,在 mapper 向 reducer 分发数据时,只需向后缀项分发数据,就可以有足够的信息构造闭频繁项集,而无须向非后缀项分发数据。由事务数据中所有后缀项组成的列表称为后缀项表,它是 F-List 的一个子集。

2.2　后缀项表的构造

后缀项表的构造需要在 F-List 构造完成以及将事务数据进行删减排序之后才能进行。但是构造后缀项表并不需要单独进行一次 MapReduce 过程,可以将这个过程放在对事务数据库进行删减排序的 MapReduce 过程中一并处理。

根据定义 3,将每条事务模式 T 中的项按支持度分成若干集合,同一支持度的项的最后一项则成为第一个后缀项;然后按照定义 4 递归地判别其余项与后缀项组合模式的支持度,若存在与项的支持度相等的组合模式支持度,则删去该非后缀项,否则将其加入后缀项表。

该方法虽然流程简单,易于理解,但是实际操作却是不可行的。非后缀项与后缀项组合模式数量巨大,判定组合模式的全局支持度,需要数据的二阶支持度矩阵进行支持;而生成二阶支持度矩阵代价巨大,结果可能不仅没有减少数据的传送量,反而使其增大。

因此,本文提出一种基于垂直格式挖掘的后缀项表构造算法,只需遍历一次事务模式,即可构造出后缀项表,并且该过程同样可以在事务数据预处理的 MapReduce 过程中一并处理。

算法的基本思想是:用项所在的位置特征(如项所在 TID 集合)标志每一项,根据后缀项的定义可知,同一支持度下多个具有相同位置特征的项,其最右的项一定是后缀项,而其余项一定不是后缀项,因此可以删去,由此即可构造后缀项表。

该算法描述如下:

a)对于 mapper

(a)读取 F-List;

(b)遍历每条事务模式 T,将 T 的 TID 按一定规则转换为相应的整数值 ID_NUM;由 F-List 求得 T 中项 I 的支持度 sup;

(c)输出 sup 作为 key,值对〈I, ID_NUM〉作为 value;

b)对于 reducer

(a)读取 F-List;

(b)对于输入 value 中相同的 I 进行汇总,将其 ID_NUM 由小到大排列并拼接为一个字符串 P 作为其位置特征(若该字符串过长,可考虑使用 MD5 等算法提取其特征);

(c)对于串 P 长度相同的项比较 P 值,对于 P 值相同的项,只保留最右的一项,并将其余项删除。

(d)输出所有被保留的项 I。

c)对于 driver

汇总各 reducer 输出的项 I,将 F-List 中非 I 的项删除,即得到后缀项表 P-List。

Map 和 Reduce 算法的伪代码见算法 3 和 4。

算法 3　后缀项表构造 Map 算法

```
void Map(key k,value T_i)
{F-List. Load();
  int ID_NUM = T_i. TID. toInteger();
  Item a[ ] = T_i. Split();
  for j = a. length-1 to 0
        {   int sup = F_list. getSup(a[j]);
            OutPut(sup,〈a[j], ID_NUM〉);
        }
}
```

算法 4　后缀项表构造 Reduce 算法

```
void Reduce (key key,value[ ] Pairs)
{F-List. Load();
  pairlist〈item,string〉 plist = new pairlist();
  foreach pair in Pairs
   {if(plist. find(pair. first))
        plist. getPair(pair. first). second.
       concatenateInOrder(pair. second);
    else
        plist. addPairInOrder(pair,F-List);
}
foreach pair in plist
{if(pair. compareBySecondPart(plist.  getRests(pair))
      plist. delete(pair);
}
   OutPut(〈null,plist〉);
}
```

2.3　PFP-P 算法描述

与 PFP 算法类似,PFP-P 算法同样进行三步 MapReduce 操作,前两步执行与 PFP 算法稍有不同:

a)只需生成 F-List 即可,不需要生成 G-List。

b)在对事务数据进行排序和删减后,执行后缀项表构造算法,生成 P-List;然后将 P-List 中的 i 个频繁项分成 Q 个组,作为 G-List,每个组分配一个 gid。

c)MapReduce 操作中的 mapper 端进行与 PFP 算法相同的操作。

对于 redecer:

(a)每个 reducer 获取 G-List 中的一个组,并根据组的 gid 拖取 mapper 分发完成的数据。

(b)拖取完成后,对于 G-List 中的每个项获取其条件数据 DB_{item}。

(c)对每个 DB_{item} 调用闭频繁项集挖掘算法,构建 FP-Tree,生成闭频繁项集。本文中的闭频繁项集挖掘算法采用较有代表性的基于 FP-Tree 挖掘的 CLOSET 算法[11]。

算法步骤 c)中 Reduce 操作的伪代码见算法 5。

算法 5　PFP-P 算法步骤 c)中的 Reduce 算法

```
void Reduce (key gid,value DB_gid)
{G-List. Load();
```

```
    nowGroup = G-List.getGroupByGid(gid);
    foreach item in nowGroup
    {   DB_item.get(item, DB_gid);
        Heap HP = new Heap();
        CLOSET(item, DB_item, G-List, HP);
        foreach pattern in HP
            OutPut(〈null, pattern + pattern.supp()〉);
    }
}
```

2.4 PFP-P 算法分析

本节对 PFP-P 的数据通信复杂度进行分析。算法的步骤 a)传送的数据量与 PFP 相同,为 M_{DB}。步骤 b)构造 P-List 需要传送的数据量同样为 M_{DB}。

步骤 c)的数据传送量为

$$M_{Dup} = M_{DB} + \sum_{i=1}^{n}\sum_{j=1}^{l_i-1} jf(I_jI_{l_i})g(I_j) \tag{5}$$

其中:函数 $g(I)$ 定义为

$$g(I) = \begin{cases} 0 & I\text{ 为非后缀项} \\ 1 & I\text{ 为后缀项} \end{cases} \tag{6}$$

将三个步骤的数据传送量加总,可知算法全过程的数据传送量为

$$M = M_{DB} + M_{Dup} = 3M_{DB} + \sum_{i=1}^{n}\sum_{j=1}^{l_i-1} jf(I_jI_{l_i})g(I_j) \tag{7}$$

为方便讨论,设事务数据的平均长度为 l,分组取最大值,则 PFP 和 PFP-P 算法的平均数据传送总量可由式(4)及(7)求得

$$\overline{M_1} = (3/2)nl + (1/2)nl^2 \tag{8}$$

$$\overline{M_2} = 3nl + n\sum_{j=1}^{l-1} jg(I_j) \tag{9}$$

由式(8)和(9)有

$$\overline{M_2} - \overline{M_1} = n\sum_{j=1}^{l-1} jg(I_j) - n(1/2)l(1-l) + nl \tag{10}$$

当全部 $g(I)$ 均为 1 时,即事务数据的所有项均为后缀项时,式(10)的值为第三项 nl,即此时 PFP-P 比 PFP 多传送一个 M_{DB} 的数据。当 $g(I)$ 为 0 的项,即非后缀项的系数 j 之和大于事务平均长度 l 时,PFP-P 的数据传送量将小于 PFP。当平均长度 l 增大时,式(10)中第三项的 nl 对总传送量的影响将迅速减小,起决定性作用的将是式中的前两项。在 l 较大时,只需很少的非后缀项,就可以使其系数之和大于 l。

根据后缀项表的构造算法,设满足支持度 sup 项的个数为 k_{sup},则支持度为 sup 的项中存在非后缀项的概率可表示为

$$P(\exists I \notin P\text{list}) = \min\left(1, \frac{k_{sup}-1}{n(n-1)(n-2)\cdots(n-\text{sup}+1)}\right) \tag{11}$$

设事务数据中频繁项的最小支持度为 minsup,项的最大支持度为 maxsup,则有

$$M_{DB} = \sum_{i=1}^{n} l_i = \sum_{i=\text{minsup}}^{\text{maxsup}} ik_i \tag{12}$$

假设在事务数据中 n 和各项的支持度均保持不变,由式(12)可知,l 增大时 k 随之增大;再由式(11)可知,事务数据中的项为非后缀项的概率也会随之增大。因此对于事务平均长度较大的数据,PFP-P 可以有效减少数据的平均传送量。

以数据 $\{T_1 = (a_1, a_2, \cdots, a_{50}), T_2 = (a_1, a_2, \cdots, a_{100})\}$ 为例,设最小支持度阈值为 1,采用最大分组。则在 PFP 中,需要将数据分为 100 个组,数据总传送量为 $150 \times (3/2) + (1/2) \times (50 \times 50 + 100 \times 100) = 6475$。而在 PFP-P 中,可知 a_{50} 为支持度 2 的后缀项,a_{100} 为支持度 1 的后缀项,其余项均为非后缀项;则只需分为 a_{50} 和 a_{100} 两组,数据总传送量为 $150 \times 3 + 49 + 99 = 589$。

3 实验及结果分析

实验基于 Hadoop 平台[12],共有八个计算节点,包括一个主节点(NameNode)和七个数据节点(DataNode)。每个节点采用 Intel Core2 2 GHz 处理器,2 GB 内存,260 GB 磁盘空间,通过 100 M 以太网相连,操作系统采用 CentOS6.2。

本文将 Mahout 开源项目中的 PFP 算法改写为 PFP-P 算法,在上述实验环境中与 PFP 和 PFP-C 算法进行了对比。为了方便描述,用 D#T#C#的形式来描述实验数据集,其中 D 表示事务的长度,即维数;T 表示事务的数目;C 表示事务的势,即每一维可取值的数目。采用的实验数据集有 3 个,分别代表三类数据:accidents 数据集[13](D570T340000C45)代表大量事务数目的低维数据;breast cancer 数据集[14](D24481T78C100)代表少量事务数目的高维数据;合成数据集 testdata(D15000T8000C100)代表大量事务数目的高维数据。

图 1 ~3 分别为三种算法在三个数据集的挖掘时间随最小支持度阈值(sup_min)的变化。

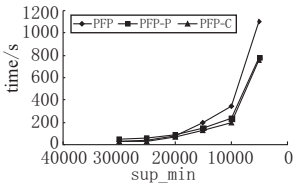

图1 三种算法挖掘accidents 数据集性能对比

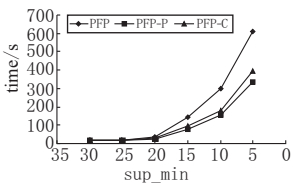

图2 三种算法挖掘breast cancer 数据集性能对比

由图 1 可以看出,对于低维数据,在 sup_min 值较大时,PFP-P 算法比其他两种算法花费更多时间进行挖掘;而随着 sup_min 减小,PFP 算法的挖掘时间急剧增加,PFP-P 和 PFP-C 算法的增长则相对缓慢,但三者的差别并不是很大。当 sup_min 较大时,由于 PFP-P 算法比其他两种算法多出一个后缀项表计算的步骤,而闭频繁项的数目与频繁项的数目差别较小,因此 PFP-P 算法并不具有优势;随着 sup_min 减小,频繁项的数目急剧增加,PFP-P 算法的优势逐步体现出来。另外,对于低维数据,由于非后缀项出现的概率较小,PFP-P 算法并不能大量地减少分组数据的传送量,因而 PFP-P 与 PFP 算法的挖掘时间差别并不是很大,并且整体性能略低于 PFP-C 算法。

由图 2 和 3 可以看出,PFP-P 算法的挖掘时间增长比 PFP 和 PFP-C 算法明显缓慢,而对于第三个数据集 testdata,PFP-P 算法显示出更为明显的优势。这主要是由于虽然对于高维数据,非后缀项出现的频率明显增加,使得 PFP-P 算法能够有效减少分组数据传送量,但是对于 breast cancer 这类事务数较小的数据集,其本身的数据量并不大,分组数据的传送并不是系统运行的主要瓶颈,因此 PFP-P 算法虽然能够减少分组数据传送所用的时间,但其对减少总挖掘时间的贡献并不大;而对于 testdata 这种高维海量数据,PFP-P 算法才能充分发挥其优势。

图 4 所示是三种算法在同一 sup_min 下挖掘 testdata 数据集的加速比对比。三种算法的加速比均低于理想加速比,PFP-C 算法的加速比与 PFP 相当,而 PFP-P 算法的加速比明显优于其余两种算法。这是由于 PFP-P 算法在挖掘时减少了分组数据的传送量,降低了节点间的 I/O 和网络带宽消耗,因而随着节点的增加,加速效果会明显优于 PFP 及

PFP-C 算法。

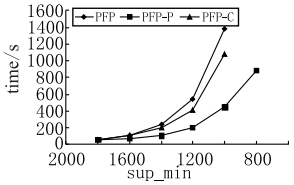

图3　三种算法挖掘testdata数据集性能对比

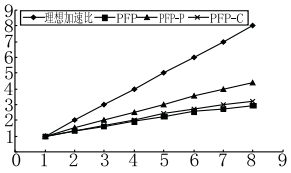

图4　三种算法加速比对比

4　结束语

本文在 PFP 算法的基础上,提出一种基于后缀项表的并行闭频繁项集挖掘算法 PFP-P。该算法以闭频繁项集的挖掘替代原算法中对全部频繁项的挖掘,以提高挖掘效率;并针对闭频繁项集的特点,在挖掘过程中引入后缀项表,以减少挖掘过程中分组数据的传送量,降低系统内耗。实验表明,该算法在平均性能上优于原算法,在最小支持度阈值减小时,能够有效缩短挖掘时间;算法能够有效降低节点间的通信消耗,具有较好的加速性;算法处理高维数据集比低维数据集更具优势,因此更适合海量高维数据的挖掘任务。

参考文献:

[1] HAN Jia-wei, CHENG Hong, XIN Dong, *et al*. Frequent pattern mining: current status and future directions[J]. Data Mining and Knowledge Discovery,2007,15(1):55-86.

[2] AGRAWAL R, IMIELISKI T, SWAMI A. Mining association rules between sets of items in large databases[J]. ACM SIGMOD Record,1993,22(2):207-216.

[3] HAN Jia-wei, PEI Jian, YIN Yi-wen. Mining frequent patterns without candidate generation[J]. ACM SIGMOD Record,2000,29(2):1-12.

[4] ZAÏANE O R, EL-HAJJ M, LU P. Fast parallel association rule mining without candidacy generation[C]//Proc of IEEE International Conference on Data Mining. 2001: 665-668.

[5] PRAMUDIONO I, KITSUREGAWA M. Parallel FP-growth on PC cluster[C]//Proc of the 7th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Berlin: Springer-Verlag, 2003:467-473.

[6] LI Li, ZHAI Dong, JIN Fan. A parallel algorithm for frequent itemset mining[C]//Proc of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies. 2003: 868-871.

[7] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM,2008,51(1): 107-113.

[8] LI Hao-yuan, WANG Yi, ZHANG Dong, *et al*. PFP: parallel FP-Growth for query recommendation[C]//Proc of ACM Conference on Recommender Systems. 2008: 107-114.

[9] OWEN S, ANIL R, DUNNING T, *et al*. Mahout in action[M].[S. l.]: Manning Publications,2011.

[10] WANG Su-qi, YANG Yu-bin, CHEN Guang-peng, *et al*. MapReduce-based closed frequent itemset mining with efficient redundancy filtering[C]//Proc of the 12th IEEE International Conference on Data Mining Workshops. 2012: 449-453.

[11] PEI Jian, HAN Jia-wei, MAO Run-ying. CLOSET: an efficient algorithm for mining frequent closed itemsets[C]//Proc of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. 2000:21-30.

[12] WHITE T. Hadoop: the definitive guide[M].[S. l.]: O'Reilly Media,Inc., 2012.

[13] XIN Dong, HAN Jia-wei, YAN Xi-feng, *et al*. Mining compressed frequent-pattern sets[C]//Proc of the 31st International Conference on Very Large Data Bases. 2005:709-720.

[14] CONG Gong,TUNG A K H,XU Xin,*et al*. Farmer: finding interesting rule groups in microarray datasets[C]//Proc of ACM SIGMOD International Conference on Management of Data. 2004:143-154.

(上接第 353 页)

[3] KAZIENKO P, MUSIAL K, KAJDANOWICZ T. Multidimensional social network in the social recommender system[J]. IEEE Trans on Systems, Man and Cybernetics, Part A: Systems and Humans,2011,41(4):746-759.

[4] TRAUD A L, MUCHA P J, PORTER M A. Social structure of Facebook networks[J]. Physica A: Statistical Mechanics and Its Applications,2012, 391(16): 4165-4180.

[5] RADICCHI F, CASTELLANO C, CECCONI F, *et al*. Defining and identifying communities in networks [J]. Proceedings of the National Academy of Sciences of the USA,2004,101(9):2658-2663.

[6] SPAULDING T J. How can virtual communities create value for business[J]. Electronic Commerce Research and Applications, 2010,9(1):38-49.

[7] LIN Shi-zhuang, LIU Jing-yu, FANG Yan-jun. ZigBee based wireless sensor networks and its applications in industrial[C]//Proc of IEEE International Conference on Automation and Logistics. 2007: 1979-1983.

[8] BRÓDKA P, SAGANOWSKI S, KAZIENKO P. Group evolution discovery in social networks[C]//Proc of International Conference on Advances in Social Networks Analysis and Mining. 2011: 247-253.

[9] 李孝伟,陈福才,刘力雄. 一种融合节点与链接属性的社交网络社区划分算法[J]. 计算机应用研究,2013,30(5):1477-1480.

[10] ZHAO Zhong-ying, FENG Sheng-zhong, WANG Qiang, *et al*. Topic oriented community detection through social objects and link analysis in social networks[J]. Knowledge-Based Systems,2012,26(2): 164-173.

[11] ZHANG Zhong-feng, LI Qiu-dan, ZENG D, *et al*. User community discovery from multi-relational networks[J]. Decision Support Systems,2013,54(2):870-879

[12] BRÓDKA P, FILIPOWSKI T, KAZIENKO P. An introduction to community detection in multi-layered social network[M]//Information Systems,E-learning,and Knowledge Management Research. Berlin: Springer-Verlag,2013:185-190.

[13] BRÓDKA P, STAWIAK P, KAZIENKO P. Shortest path discovery in the multi-layered social network[C] //Proc of International Conference on Advances in Social Networks Analysis and Mining. 2011:497-501.

[14] NEWMAN M E J, GIRVAN M. Finding and evaluating community structure in networks [J]. Physical Review E, 2004, 69 (2): 026113.