一种基于 MapReduce 的频繁闭项集挖掘算法*

陈光鹏 杨育彬 高 阳 商 琳

(南京大学 计算机软件新技术国家重点实验室 南京 210093)

摘 要 频繁闭项集的挖掘是发现数据项之间关联规则的一种有效方式. 当前以 MapReduce 模式为基础的云计算平台为解决海量数据中的关联规则挖掘问题提供新的解决思路. 文中提出并实现一种基于 Hadoop 云计算平台的频繁闭项集的并行挖掘算法. 该算法主要包括并行计数、构造全局频繁项表、并行挖掘局部频繁闭项集和并行筛选全局频繁闭项集四个步骤. 在多个数据集上的实验表明,该方法能较大提高数据挖掘的效率,具有较好的加速比.

关键词 云计算,并行算法,数据挖掘,频繁闭项集,MapReduce

中图法分类号 TP 311

Closed Frequent Itemset Mining Based on MapReduce

CHEN Guang-Peng, YANG Yu-Bin, GAO Yang, SHANG Lin

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

ABSTRACT

Closed frequent itemset mining is an useful way for discovering association rules from data. Cloud computing infrastructure based on MapReduce provides a promising solution to address the problem. A parallel algorithm for mining closed frequent itemset is presented based on the Hadoop cloud computing platform. The method consists of four steps: parallel counting, global F-List constructing, parallel mining of local closed frequent itemset and parallel filtrating of global closed frequent itemset. The experimental results validate the method and show that it is effective with a satisfied speedup.

Key Words Cloud Computing, Parallel Algorithm, Data Mining, Closed Frequent Itemset, MapReduce

1 引 言

在海量数据库中挖掘关联规则,是数据挖掘领域的一个十分重要,并且具有挑战性的研究内容. 它在适用性、挖掘效率、准确性和可理解性等方面常优于其他数据挖掘算法. 通常,挖掘关联规则的问题可归结为挖掘频繁项集[1-3]. 然而,从大型数据集中挖掘频繁项集的主要挑战是这种挖掘常常产生大量满

* 国家自然科学基金项目(No. 61035003,60875011,60721002)、国家 973 计划项目(No. 2010CB327903)、科技部国际科技合作计划项目(No. 2010DFA11030)和江苏省自然科学基金项目(No. BK2010054)资助
收稿日期: 2011 - 02 - 14; 修回日期: 2011 - 07 - 05
作者简介 陈光鹏,男,1985 年生,硕士研究生,主要研究方向为数据挖掘. 杨育彬,男,1977 年生,博士,副教授,主要研究方向为数据挖掘、机器学习. E-mail: yangyubin@nju.edu.cn. 高阳,男,1972 年生,教授,博士生导师,主要研究方向为数据挖掘、机器学习. 商琳,女,1973 年生,博士,副教授,主要研究方向为数据挖掘、机器学习.

足最小支持度的项集，当最小支持度 *min_sup* 设得很低时尤其如此，项集的个数都太大，无法计算和存储. 频繁闭项集是频繁项集的一种压缩，由于它包含频繁项集的完整信息，而且其数量远小于频繁项集，因此当前通常的做法是用挖掘频繁闭项集，来代替挖掘频繁项集[4-5].

Closet 算法是 Pei 等[6]提出的基于 FP-tree(Frequent Pattern-tree)技术的频繁闭项集挖掘算法. 该方法只需遍历两次原始数据集来建立频繁模式树，进而在该树上挖掘频繁闭项集. 这种方法挖掘密集型数据集时，效率较好，但在挖掘稀疏型数据集时的效率较低[7]. Closet + 算法[7]是 Closet 的改进，它根据 FP-tree 的特点在 Closet 的基础上引入更多的剪枝策略，同时，针对密集型和稀疏型两种不同数据集结构类型，设计不同的投影 FP-tree 结构，来挖掘频繁闭项集. AFOPT(Ascending Frequency Ordered Prefix-tree)算法[8-9]是另一类频繁项集挖掘算法，与前两种方法相似，都是基于树的数据结构来挖掘频繁闭项集，不同点在于频繁模式树的构造方式. Closet 和 Closet + 算法构造 FP-tree 时，将事务中的各项按照其频繁次数降序的次序添加到频繁树中. 而在 AFOPT 算法中，添加事务到频繁树时，将事务中的各项按照其频繁次数升序的次序添加到频繁树中. 因此，越靠近根节点的数据项，其频繁次数越小，挖掘频繁闭项集时，也是从频繁次数最小的项集开始挖掘. 该方法是目前效率较高的方法之一，本文则基于此方法，提出挖掘局部频繁闭项集的 AFOPT 算法.

面对海量的数据，单机模式下挖掘频繁闭项集的算法常常显得力不从心. 并行化是提高效率的有效方法[1-3]. 云计算是并行计算的一种有效实现方式. Hadoop 就是一种典型的云计算平台(http://cwiki.apache.org/confluence/display/MAHOUT/Mahout+Wiki)，它是一个分布式系统基础架构，由 Apache开发，由 HDFS、MapReduce 和 Hbase 组成. 用户可在不了解分布式底层细节的情况下，开发分布式程序，充分利用集群的力量来达到高速运算和存储[10]. Li 等[10]提出的 PFP(Parallel FP-Growth)算法就是在 Google 的 MapReduce 基础架构上，实现频繁项集挖掘的方法. 该方法利用 FPGrowth 算法中可并行操作的部分，将其 MapReduce 化，进而挖掘出 top-k频繁项集. 在云计算平台下，并行挖掘频繁闭项集是一个富有挑战性且非常有意义的工作，然而，目前针对的该方面的研究仍然很少. 本文提出一种基于云计算平台 Hadoop 的频繁闭项集的挖掘方法. 该方法主要分 4 个步骤: 并行计数，构造全局 F-List(Frequency List)，并行挖掘局部频繁闭项集和并行筛选全局频繁闭项集，最终挖掘出满足最小支持度 *min_sup* 的频繁闭项集. 实验表明，该方法在保证数据挖掘结果正确性的同时，较大提高数据挖掘的效率，具有较好的加速比.

2　挖掘频繁闭项集的并行算法

2.1　AFOPT 算法介绍

AFOPT 算法[8-9]是由 Liu 等提出的一种频繁项集挖掘算法. 该算法被扩展后，可分别挖掘极大频繁项集和频繁闭项集，其中，用于挖掘频繁闭项集的 AFOPT-closed 算法的输入是一个 AFOP-tree(Ascending Frequency Ordered Prefix-tree)[9]. 构造该树时，将事务中的各项按照其频繁次数升序的次序添加到频繁树中. 因此，越靠近根节点的数据项，其频繁次数越小，挖掘频繁闭项集时，也是从频繁次数最小的项集开始挖掘. 首先，在根节点的直接子结点中，找一个频繁次数最小的结点，以其为前缀，将其子女按照局部的 F-List 的次序，构造新的局部的 AFOP-tree 树. 如果该树是单分支结构，则验证该分支所携带的频繁项集，相对于已挖掘到的频繁闭项集是否是闭合的. 如果是，则将其存入 CFI-tree(Closed Frequent Itemset-tree)中. 然后执行更新操作，即将该频繁次数最小的结点删除，将其子女和其兄弟结点中，具有相同数据项的进行合并. 然后，再在根节点的直接子结点中，找一个频繁次数最小的结点，对其子树进行挖掘. 如此递归，直至所有的频繁闭项集都被挖掘出来. 显然，由 AFOP-tree 的结构可知，每次新挖掘出的频繁项集的长度一定小于或等于之前挖掘出的频繁闭项集的长度，所以在验证新挖掘的频繁项集相对于 CFI-tree 中的项集是否闭合时，只需要子集检查，而无需超集检查.

2.2　挖掘频繁闭项集的并行算法总体框架

我们提出一种基于云计算平台 Hadoop 的频繁闭项集的挖掘方法. 对于给定的一个事务数据库，本方法共需要 4 步来完成频繁闭项集的挖掘，如图 1 所示. 控制程序 driver 操控整个流程，将可并行操作的步骤以任务的形式提交给 Hadoop 平台，具体步骤如下.

step 1　并行计数. 并行扫描一次数据库，统计其中每一项的频繁次数.

step 2　构造全局 F-List. 此步骤不是并行的，以

上一步并行计数的输出结果为输入，构造全局 F-List.

step 3 并行挖掘局部频繁闭项集. 此步骤是并行操作的，再次扫描事务数据库，在各个结点用 AFOPT 算法挖掘局部频繁闭项集.

step 4 并行筛选全局频繁闭项集. 此步骤也是并行操作的，在各个结点过滤掉局部是闭的，但全局不是闭的频繁闭项集.

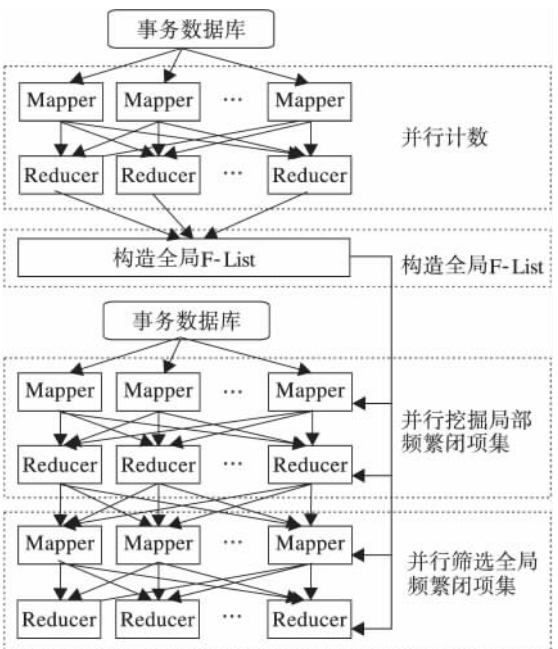


图 1 挖掘频繁闭项集的并行算法总体框架

Fig. 1 Framework of parallel algorithm for mining closed frequent itemset

2.3 并行计数

为了构造全局频繁项表 F-List，首先得统计数据库中每个数据项出现的次数. 此步骤并行扫描一次数据库，按行读取，统计其中每项的频繁次数. 主要借鉴 Hadoop 平台上的典型算法 WordCount (http://hadoop.apache.org)，其伪代码如下所示.

算法 1 并行计数

```
Mapper(LongWritable offset, Text value = transaction)
    {   for (each item A_i in value)
        output(A_i, 1);
    }
Reducer(Text key, Iterable <IntWritable> values)
    {   int sum = 0;
        for(IntWritable value : values)
        {   sum = sum + 1;
        }
        output (key, sum);
    }
```

2.4 构造全局 F-List

根据上一步得到的每个数据项出现的频繁次数，可以很容易地构造出全局 F-List. 这一步是非并行的，由控制程序 *driver* 完成. 它以上一步并行计数的输出结果为输入，取满足最小支持度 min_sup 的项，按其频繁次数由大到小进行排序，结果存放在 F-List 中. 其伪代码如下所示.

算法 2 构造全局 F-List

```
constructDescendFList (DB_frequency, min_sup)
Input:  DB_frequency is the results of Parallel Counting;
        min_sup is the minimum support threshold;
Output: F-List is the frequent item list;
Description:
{   ArrayList < Pair < string , int > > F-List;
    For ( Pair < string , int >  itemWithCount : DB_frequency)
    {   if ( itemWithCount.count >= min_sup)
        {   insert itemWithCount into F-List by descend order of
            itemWithCount.count ;
        }
    }
    return F-List;
}
```

2.5 并行挖掘局部频繁闭项集

接下来，就可在上一步得到的全局 F-List 的帮助下，并行挖掘局部频繁项集.

此步骤是并行操作的，Mapper 按行读取数据，每行即一条事务. 将其中的各数据项以 F-List 的顺序进行再排序. 然后，从右向左依次以新事务中的数据项为 *key*，以出现在该数据项左边的数据项集为 *value* 发出[10]. Reducer 用接收到的所有事务，构造一棵 AFOPT[9]，这棵 AFOPT 是以 key 为条件的树. 然后用 AFOPT-Closed 算法，递归地在该树中挖掘频繁闭项集. 此时得到的频繁闭项集都是以 key 为条件的局部频繁闭项集. 其伪代码如下所示.

算法 3 并行挖掘局部频繁闭项集

```
Mapper( LongWritable offset, Text value = transaction)
  {   order value by F-List;
      String a [] = Split( value) ;
      for( i = a.length - 1; i >= 0; i - - )
          output( Text( a [i]) , Text( a [0] + a [1] + … +
          a [i]) ) ;
  }
Reducer( Text key , Iterable < Text > values)
  {   r = constructAFOPT( key, values) ;
      frequentClosedItemsTree = AFOPT-closed ( r, min_sup) ;
      for( frequentClosedItems : frequentClosedItemsTree)
```

```
    {
        output ( Text( key) ,Text( frequentClosedItems +
                frequentClosedItems ' support) ) ;
      }
    }
```

2.6　并行筛选全局频繁闭项集

上一步的结果只是局部频繁闭项集,其中只有一部分是全局闭项集,需要将其筛选出来.

此步骤也是并行操作的,用来过滤掉局部是闭的,但全局不是闭的频繁闭项集,将全局闭项集筛选出来. Mapper 将局部的频繁闭项集,以其中每一项为 *key*,广播出去. Reducer 把收到的项集按照其长度由大到小进行排序,然后再将其依次添加到 *frequentPatternTree* 中. 添加项集的时候,只需进行子集检查,无需超级检查,因为添加的次序是按照项集的长度由大到小依次添加的,所以当前的项集的长度必定小于或等于之前添加的项集的长度. 如果此项集对于 *frequentPatternTree* 中的所有项集都是闭合的,则将其添加到 *frequentPatternTree* 中,否则舍去. 最终,只将 *frequentPatternTree* 中,以该 Reducer 的 *key* 为其最后一项的频繁闭项集发出. 此时,得到的频繁项集是全局频繁闭项集. 其伪代码如下所示.

算法 4　并行筛选全局频繁闭项集

```
Mapper( LongWritable offset,Text value =
        frequentClosedItems + frequentClosedItems' support)
  {   itemset = value. frequentClosedItems ;
      for ( item : itemset)
        output( item, value) ;
  }
Reducer( Text key , Iterable < Text > values)
  {   frequentPatternTree = null;
      Sort the Order of itemset in values by their length from long
      to short;
      for ( itemset : values)
        {   if ( itemset is closed in frequentPatternTree)
              insert itemset into frequentPatternTree;
        }
        for ( itemset : frequentPatternTree)
          {   if ( key is the last item of this itemset)
                output ( Text ( key) , Text( itemset) ) ;
        }
    }
```

3　实验与结果分析

为了验证该并行算法的正确性和加速效果,我们在多个数据集上进行实验.

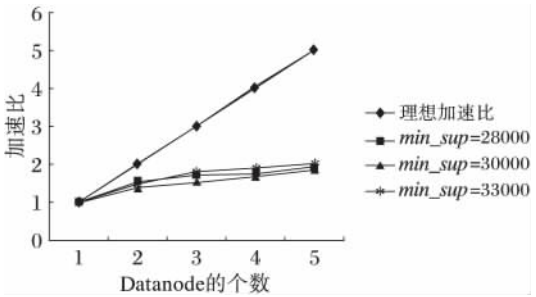
我们的实验环境采用云计算平台 Hadoop,共有 6 个结点: 1 个主控结点(namenode) ,5 个数据节点(datanode) . 每个结点的配置如下: 4 核处理器 Intel Core i7 860,4GB 内存,500GB,千兆网卡,操作系统为 Ubuntu 10.04. 每个 datanode 设置的 mapper 的最大负载(mapred. tasktracker. map. tasks. maximum) 为 3,Reducer 的最大负载(mapred. tasktracker. reduce. tasks. maximum) 也为 3. 整个集群的 hdfs 的配置容量为 2.41TB.

将上述算法基于 MapReduce 模式在 Hadoop 云计算平台编程实现,采用的测试数据集为挖掘频繁项集领域常用的测试数据集中较大的数据集 connect、pumsb 和 accidents 分别进行实验. 对于相同数据集,还选择不同 *min_sup* 进行实验. 针对数据集 connect 的实验的最小支持度 *min_sup* 分别设置为 28 000、30 000 和 33 000; 针对数据集 pumsb 的实验的最小支持度 *min_sup* 分别设置为 36 000、38 000 和 40 000; 针对数据集 accidents 的实验的最小支持度 *min_sup* 分别设置为 100 000、140 000 和 170 000. Parallel Counting 的 *ReduceTasks* 的个数设为 1 时,并行挖掘局部频繁闭项集和并行筛选全局频繁闭项集的 ReduceTasks 的个数设为 20,得到的加速比如图 2 所示. 该方法在保证数据挖掘结果正确性的同时,较大提高数据挖掘的效率,具有较好的加速比.

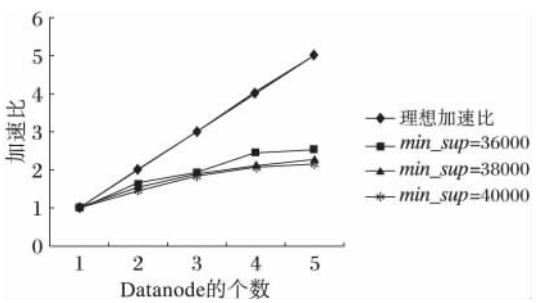
可以看出,对于同一数据集,不同的最小支持度得到的加速比不一样,但加速比的变化趋势是一致的,即随着 *datanode* 个数的增多而增加. 刚开始时,加速比增长比较快,但随着 *datanode* 个数的增多,通信的开销也越来越大,加速比的增长速度也随之降低,但较单结点来说,多结点的数据处理效率还是有很大优势的. 图 2(a) 中,不同最小支持度所得到的加速比性能相差不大,但在(b) 和(c) 中,随着最小支持度的逐渐升高,加速比性能明显呈下降趋势,最优加速比都是在较小的最小支持度时得到,即对于 pumsb 数据集,*min_sup* = 36 000 时加速比最优,对于 accidents 数据集,*min_sup* = 100 000 时加速比最优. 因为当最小支持度升高时,频繁闭项集的数量逐渐减少, step 3 和 step 4 需要处理的数据量都在减少,用于并行计算的时间开销所占整体开销的比重也随之减小,加速比的增长量也就自然降低.

同时,还可发现,在图 2 中,(c) 的最优加速比优于(b) ,(b) 的最优加速比优于(a) . 因为当数据集较小时(数据集大小: connect < pumsb < accidents) ,各个节点用于计算的时间较少. 随着 datanode 个数的增多,用于通信的时间开销所占的比重

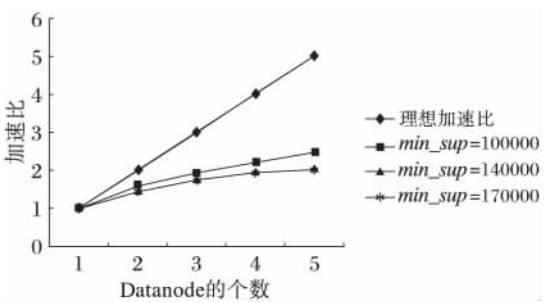
越来越明显. 而对于较大的数据集,各个节点的计算时间本身较长,用于通信的开销所占的比重相对要小很多. 可得出结论: 该算法在规模较大的数据集上取得的加速比要优于数据规模较小的. 这也显示出,该并行算法对于处理规模较大的数据集更有优势.



(a) Connect



(b) Pumsb



(c) Accidents

图 2 基于 MapReduce 的频繁闭项集挖掘算法分别在不同数据集上的加速比

Fig. 2 Speedup of algorithm for MapReduce based closed frequent itemset mining on different datasets

4 结 束 语

本文提出一种基于云计算平台 Hadoop 的频繁闭项集的挖掘方法. 该方法主要分 4 个步骤: 并行计数,构造全局 F-List,并行挖掘局部频繁闭项集和并行筛选全局频繁闭项集,最终挖掘出满足要求的频繁闭项集. 实验表明该方法是有效的. 本文的贡献主要有,提出一种基于云计算平台 Hadoop 的频繁闭项集挖掘方法的框架; 将 AFOPT 算法 MapReduce 化,用于挖掘局部的频繁闭项集; 提出一种并行筛选全局频繁闭项集的算法.

当然,该方法的不足之处是,面对较小数据集时,其效率往往不如单机的挖掘算法. 同时,本方法的 step 2 是单机处理,面对较小规模全局 F-List,其性能较好,没有并行化的必要. 但当需要构造的全局 F-List 的规模很大时,将会影响整个算法性能,这是以后需要改进的地方.

参 考 文 献

[1] Aouad L M, Le-Khac N A, Kechadi T M. Performance Study of Distributed Apriori-Like Frequent Itemsets Mining. Knowledge and Information Systems, 2010, 23(1): 55 - 72

[2] Yu Kunming, Zhou Jiayi, Hong T P, *et al*. A Load-Balanced Distributed Parallel Mining Algorithm. Expert Systems with Applications, 2010, 37(3): 2459 - 2464

[3] Shankar S, Purusothaman T. Utility Sentient Frequent Itemset Mining and Association Rule Mining: A Literature Survey and Comparative Study. International Journal of Soft Computing Applications, 2009, 4: 81 - 95

[4] Tao Limin, Huang Linpeng. Cherry: An Algorithm for Mining Frequent Closed Itemsets without Subset Checking. Journal of Software, 2008, 19(2): 379 - 388 (in Chinese)
(陶利民,黄林鹏. Cherry: 一种无须子集检查的闭合频繁集挖掘算法. 软件学报, 2008, 19(2): 379 - 388)

[5] Han Jiawei, Kamber M. Data Mining: Concepts and Techniques. London, UK: Morgan Kaufmann, 2006

[6] Pei Jian, Han Jiawei, Mao Runying. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets // Proc of the ACM SIGMOD International Workshop on Data Mining and Knowledge Discovery. Dallas, USA, 2000: 21 - 30

[7] Wang Jianyong, Han Jiawei, Pei Jian. CLOSET +: Searching for the Best Strategies for Mining Frequent Closed Itemsets // Proc of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, USA, 2003: 236 - 245

[8] Liu Guimei, Lu Hongjun, Yu J X, *et al*. AFOPT: An Efficient Implementation of Pattern Growth Approach [EB/OL]. [2003-12-19]. http://ftp.informatik.rwth-aachen.de/Publications/CEURWS/Vol-90/liu.pdf

[9] Liu Guimei, Lu Hongjun, Xu Yabo, *et al*. Ascending Frequency Ordered Prefixtree: Efficient Mining of Frequent Patterns // Proc of the 8th International Conference on Database Systems for Advanced Applications. Kyoto, Japan, 2003: 65 - 72

[10] Li Haoyuan, Wang Yi, Zhang Dong, *et al*. PFP: Parallel FP-Growth for Query Recommendation // Proc of the ACM Conference on Recommender Systems. Lausanne, Switzerland, 2008: 107 - 111