doi:10.3969/j.issn.1001-4616.2016.04.005

一种新的基于 FP_Growth 的频繁项目集并行挖掘算法

孙鸿艳,吉根林

(南京师范大学计算机科学与技术学院,江苏 南京 210023)

[**摘要**] 频繁项目集挖掘用于发现项目之间的关联规则. 为了高效求解面向大数据的频繁项目集,本文提出一种新的基于 FP_Growth 的频繁项目集并行挖掘算法 NPFP_Growth(New Parallel algorithm based on FP_Growth),该算法对频繁模式树的存储结构进行改进,基于 Map/Reduce 并行计算模型,利用 HDFS 实现数据存储,在各自计算节点上构造局部频繁模式树,求解该局部频繁模式树中每个分支的最长全局频繁项目集;对于全局非频繁项目集,计算其支持数,发送至相应计算节点进行支持度统计,从而以较为简单的算法实现频繁项目集并行挖掘. 实验表明,NPFP_Growth 算法具有较高的计算效率和良好的可伸缩性.

[**关键词**] 频繁项目集,关联规则,FP_Growth,Hadoop,Map/Reduce
[**中图分类号**]TP311.11　[**文献标志码**]A　[**文章编号**]1001-4616(2016)04-0019-06

New Parallel Algorithm for Mining Frequent Item Sets Based on FP_Growth

Sun Hongyan,Ji Genlin

(School of Computer Science and Technology,Nanjing Normal University,Nanjing 210023,China)

Abstract:Mining of frequent item sets is used to find the association rules between items. In order to get frequent item sets of big data efficiently,this paper proposes a new parallel algorithm for mining frequent item sets based on FP_Growth,named NPFP_Growth(New Parallel algorithm based on FP_Growth). The storage structure of local frequent pattern tree is improved and created in each node based on parallel computing model Map/Reduce and distributed storage system HDFS,and then longest global frequent item sets are mined in each branch of the tree. Finally,Support for item sets which does not meet global minimum support is computed and then sent to corresponding computing node to count. Parallel mining algorithm NPFP_Growth is implemented. The experimental results show that the algorithm have high computing efficiency and good scalability.

Key words:frequent item sets,association rule,FP_Growth,Hadoop,Map/Reduce

求解频繁项目集是挖掘关联规则的关键,人们提出了很多求解频繁项目集的算法,其中 J.Han 等人于 2000 年提出的 FP_Growth 算法[1]是一种经典高效的算法,该算法将事务数据压缩成一棵频繁模式树即 FP_Tree,然后在该 FP_Tree 上挖掘出全部的频繁项目集,文献[2-5]提出了基于 MPI 编程模型的并行 FP-Growth 算法. 金桃等人[6]于 2010 年提出一种简单有效的并行化频繁项集挖掘算法 SP-FP-Growth,通过建立以根结点为前缀的模式森林达到对 FP_Growth 的计算效率的改进. 文献[7]提出了基于 Map/Reduce 的并行 FP_Growth 算法 PFP,章志刚等人[8]于 2014 年提出了一种基于 FP_Growth 的频繁项目集并行挖掘算法 FPPM,通过局部频繁项目集得到全局频繁项目集,具有较小的网络通信量.

文献[7]提出的 PFP 算法只能求出支持度最高的 K 个频繁项目,结果不完备;文献[8]提出了 FPPM 算法对 PFP 算法进行改进,能够挖掘完备频繁项目集;然后 FPPM 算法通过构造条件 FP 树实现频繁项目集的递归挖掘,算法时间效率仍有提高的空间. 为了进一步提高 FP_Growth 频繁项目集并行挖掘算法的时间效

收稿日期:2016-03-20.
基金项目:国家自然科学基金(41471371).
通讯联系人:吉根林,博士,教授,博士生导师,研究方向:数据挖掘与云计算. E-mail:glji@njnu.edu.cn

率,本文提出了一种新的基于 FP_Growth 的频繁项目集并行挖掘算法 NPFP_Growth,将原先的 FP 树的挖掘转换成单分支的排列组合,简化频繁项目集的求解过程. 该算法基于 Hadoop 大数据平台、Map/Reduce 计算模型、HDFS 分布式存储系统,通过在各个节点上建立频繁模式树,根据频繁模式树记录的最长频繁项目集以及非频繁项目集,然后对每条记录使用单分支法实现频繁项目集挖掘. 由于单分支频繁项目集挖掘较频繁模式树挖掘简单,因此可以减少计算量,实验表明,该算法能够保证算法运行过程中较高的效率和较好的可伸缩性.

1　FP_Growth 算法

FP_Growth 算法主要是实现将事务数据库中的数据压缩为一棵频繁模式树 FP_Tree,然后通过该 FP_Tree 求解出其中所有的频繁项目集. 如表 1 所示为事务数据,TID 是事务 ID 号,用于唯一标识各个事务,项目列表是事务中包含的项目的 ID 号,对表 1 中数据建立频繁模式树,如图 1 所示.

表 1　事务数据

Table 1　Transaction data set

TID	项目列表	TID	项目列表
T1	I1,I2,I5	T6	I2,I3
T2	I2,I4	T7	I1,I3
T3	I2,I3	T8	I1,I2,I3,I5
T4	I1,I2,I4	T9	I1,I2,I3
T5	I1,I3		

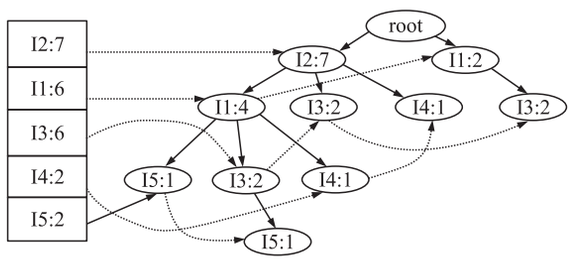

图 1　频繁模式树

Fig. 1　Frequent pattern tree

FP_Growth 算法主要包含两个核心步骤:

步骤 1　FP-树构造:

(1)扫描事务数据库,找出 1-项频繁项目集,并对其按支持度降序进行排序.

(2)建立树的根结点,以 null 标记. 扫描事务数据库,对每个事务中的每个项目按照 1-项集支持度由大到小进行排序,并插入到频繁模式树中.

步骤 2　挖掘,挖掘过程通过调用如下 FP_G(Tree,null)算法实现,其伪代码如下:

```
FP_G(fp_Tree,con_mode)//fp_Tree 为频繁模式树;con_mode 是为条件模式基,初始值为空.
  if fp_Tree 包含单条路径 path,then
    for each 路径 path 中结点的每个组合(记为 comb)do
      产生模式 comb∪con_mode,其支持度计数等于 comb 中结点的最小支持度计数;
      若其支持度大于等于最小支持度阈值,则将其输出;
    end
  else
  for fp_Tree 的头表中的每个 con_mod e_i do
    产生模式 comb=con_mod e_i ∪con_mode,其支持度数等于 con_mod e_i 的支持度数;
    构造 comb 的条件模式基,然后构造 comb 的条件 FP 树 fp_Tree_comb;
    if fp_Tree_comb ≠φ then
      FP_G(fp_Tree_comb,comb);
    end
  end
end
```

2　NPFP_Growth 算法

2.1　算法基本思想

本文提出的 NPFP_Growth 算法主要基于以下结论:

(1)频繁项目集的子集是频繁的.

(2)设 sup_i(item)是 item 在节点 i 上的支持数,n 是全局事务数,minsup 是最小支持度,若某个局部节

点 i 中存在项目 item,$\frac{\sup_i(\text{item})}{n} \geqslant \text{minsup}$,则局部节点 i 中的 item 是全局频繁的.

各个计算节点由事务数据库建立局部频繁模式树,通过挖掘局部频繁模式树求解各个分支的最长全局频繁项目集以及非频繁项目集,对于全局非频繁项目集,计算其支持数,发送至相应计算节点进行统计支持度,从而以较为简单的算法实现频繁项目集并行挖掘. 同时,对频繁模式树的存储结构进行改进,如图 2 所示,每个结点增加了指向父结点的指针,从而提高挖掘效率.

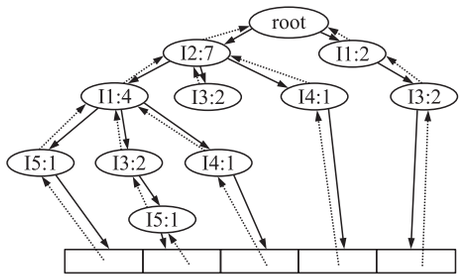

图 2　改进后的频繁模式树

Fig. 2　Improved frequent pattern tree

NPFP_Growth 算法主要包含 4 个核心步骤:

步骤 1　计算事务数据库中所有 1-项集的支持数. 此步骤用 1 个 Map/Reduce job 实现.

步骤 2　扫描事务数据库,根据 1-项集对各事务数据进行过滤和排序,各事务中具有最大支持数的项作为 key 值,其余项作为 value 值,此步骤用 1 个 Map/Reduce job 实现,结果存储在 HDFS 分布式文件中.

步骤 3　根据步骤 2 的结果在各个计算节点建立如图 2 所示的局部频繁模式树. 各分支的最长全局频繁项目集作为 key 值,非频繁项及其支持数作为 value 值,若某分支中只存在非频繁项,则 key 为 NULL. 此过程用 1 个 Map/Reduce job 实现,结果存储在 HDFS 分布式文件中.

步骤 4　读取上述步骤 3 中的分布式文件,挖掘全局频繁项集. 方法如下:如果 value 为空,则对 key 中各个项进行各种组合,得到一些全局频繁项目集;如果 value 不为空,则 key 中各个项进行各种组合,之后再与 value 中各项组合,形成候选项集,各候选项集的支持度为该项集中所包含项目的最小支持度;如果 key 为空,即 NULL,则只对 value 中的项目进行各种组合,其支持度为该项集中所包含项目的最小支持度. 此步骤用 1 个 Map/Reduce job 实现.

2.2　1-项集支持数统计

1-项集支持数统计过程,用一个 Map/Reduce 实现,其伪代码如下:

```
Mapper 过程:
map(key,value){//key 为事务 ID 号,value 为事务数据
for each word_i ⊆ value do
  output<word_i,1>  //输出,word_i 为项目,且该项目的个数记为 1
}
Reducer 过程:
Reduce(key,value){  //key 为各个项目,value 是 Mapper 中该项目的个数的集合
  count=0;
  for each v_i in value do
    count=count+v_i;
  end
  if count/‖D‖≥minsup then //D 为总事务数目,minsup 是最小支持度
    output<key,count>  //输出,key 为项,value 为该 key 对应的支持数 count
  end
}
```

2.3　数据预处理

扫描事务数据库,对事务按照 1-项集的支持数由大到小过滤和排序,最大支持度的项则为 key 值,此步骤用一个 Map/Reduce 实现,其伪代码如下:

```
Mapper 过程:
map(key,value){  //key 为事务 ID 号,value 为事务数据
  for each line ⊆ value do
    Filter_Sort(line,1-itemSets,item[])  //对各个事务 line 按 1-项集 1-itemSets 的支持度由大到小进行过滤和排
                                         序,结果用于更新 item 数组
```

```
    output<item[0],item-left>  //输出,支持度最大的项作为 key 值,其余项 item-left 作为 value 值
    end
}
Reducer 过程:
reduce(key,value){  //key 是各个项,value 是 Mapper 中所有其余项的集合
  count=null;
  For each v_i in value do
    count=count+分隔符+v_i;  //key 相同的各个分支以分隔符分隔,记为 count
  end
  output<key,count>  //输出,key 为项,value 为以分隔符分隔的各个分支
  end
}
```

2.4 建立局部频繁模式树,求解各个分支最长全局频繁项目集和非频繁项目集

该过程由一个 Map/Reduce 实现,伪代码如下:

```
Mapper 过程:
map(key,value){  //输入,key 为支持数最大的项,value 是以分隔符分隔的各个分支
  builtTree(Tree,key);  //建立频繁模式树 Tree,根结点为空,且 key 值为根结点的孩子结点
  for each branch ⊆value  //branch 是以分隔符进行分隔的分支
    updata _tree(Tree,branch,List);  //以 branch 更新 Tree
                                     同时用 List 记录叶结点位置
}
cleanup(){
  for each item_i ⊆ List  //item_i 是叶结点的位置
    ergodic(item_i,Max_fre,infre);  //根据 item_i,求解最长全局频繁项目集
                                    Max_fre 和全局非频繁项目集及支持度 infre
    if(Max_fre≡null)
      output<NULL,infer>;  //输出,若 Max_fre 为空,则 key 设置为 NULL,infer 为 value
    else
      output<Max_fre,infer>;  //输出,key 值为 Max_fre,value 值为 infer
    end
}
Reducer 过程:
reduce(key,value){  //key 是 Mapper 中的 Max_fre,value 是相同 key 的 infer 的集合
  combine=null;
  for each v_i in value do
    combine=combine+分隔符+v_i;  //将具有相同 key 的各个 value 合并,并用分隔符进行分隔
  end
  output<key,combine>  //输出,key 为最长全局频繁项目集,combine 为以该最长全局频繁项目集为前缀的全局非
                       频繁项目集及其支持度
  end
}
```

2.5 挖掘全局频繁项集

该过程的伪代码如下:

```
Mapper 过程:
  map(key,value){  //输入,key 为最长全局频繁项目集,value 值为对应的全局非频繁项目集及其支持度
    for each v_i in value do
      if(v_i≡φ)
    result=perm_comb(key);  //对 key 中各个项进行各种组合,得到一些全局频繁项目集
for each res_i in result do
```

```
        output<res_i,minsup>;  //输出组合结果 res_i 以及最小支持数 minsup
    else
      if(key≠NULL)
        result=perm_comb(key);//key 中各个项进行各种组合,得到一些全局频繁项目集
        for each res_i in result do
          output<res_i,minsup>;  //输出组合结果 res_i 以及最小支持数 minsup
          result_value=perm_comb(res_i,value)  //之后再与 value 中各项组合
          for each res_val_i in result _value do
            output<res_val_i,sup>;  //输出组合结果,支持度为该组合中各项具有的最小支持数
    else
      result=perm_comb(value);  //只对 value 中的项目进行各种组合
      for each res_i in result do
        output<res_i,sup>;  //输出组合结果 res_i,支持度为该组合中各项具有的最小支持数
      end
    end
  end
  }
  Reducer 过程:
  reduce(key,value){  //输入的 key 是组合结果,value 是该组合结果的支持数的集合
    c=0;
    for each v_i in value do
      c=c+v_i;
    if c/‖D‖≥minsup then  //D 为总事务数
    output<key,null>  //输出的 key 是满足最小支持度阈值的频繁项目集,value 为空
  end
  }
```

3 实验结果

为了验证 NPFP_Growth 算法的有效性,实验采用的是 18 台服务器构成的并行计算平台,处理器是 Intel Ⓡ Xeon Ⓡ CPU E5645,主频是 2.40 GHz,操作系统是 64 位的 CentOS realise 6.4 版. 分别利用 Hadoop -2.6.0 进行 FPPM 算法[8]和本文提出的 NPFP_Growth 算法的实现. 测试数据集由 IBM QUEST Market-Basket Synthetic Data Generator 产生,项目总数达到 1 000 种,每个事务的平均长度为 10.

在 18 个节点,事务数为 12×10^6 的情况下,对应不同最小支持度 FPPM 算法和 NPFP_Growth 算法的运行时间如图 3 所示,从图中可以看出,本文提出的 NPFP_Growth 算法比 FPPM 算法在相同的最小支持度下运行时间要少,即计算效率更高.

在节点数为 18 的情况下,NPFP_Growth 算法在不同最小支持度下运行时间和事务数的关系如图 4 所示,从图中可以看出,随着事务数线性增加,算法运行时间基本上呈线性增加且最小支持度越大,算法的运

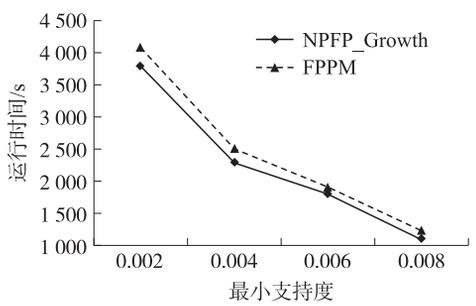

图 3 FPPM 算法与 NPFP_Growth 算法执行时间比较

Fig. 3 Algorithm execution time comparison of FPPM and NPFP_Growth

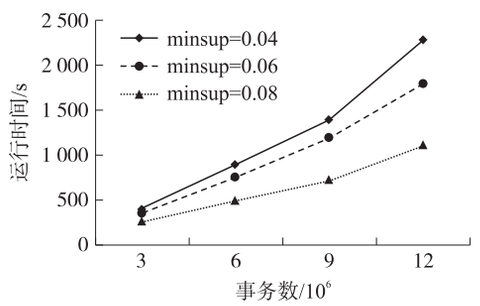

图 4 在不同支持度情况下 NPFP_Growth 算法的执行时间

Fig. 4 Execution time of algorithm NPFP_Growth at different support

行时间越小.

在最小支持度为 0.004,事务数一定的情况,NPFP_Growth 算法的运行时间与计算节点数的关系如图 5 所示. 从图中可以看出,随着节点数的增加,NPFP_Growth 算法的运行时间减少,且当节点数增加到一定的数量的时候,算法的运行时间趋于稳定.

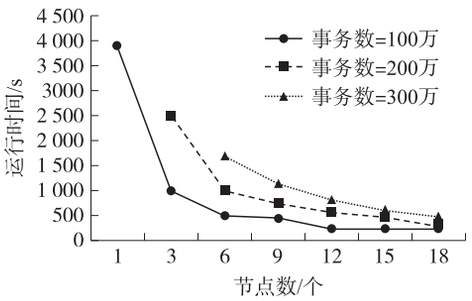

图 5　在不同计算节点情况下 NPFP_Growth 算法的执行时间

Fig. 5　Execution time of algorithm NPFP_Growth at different number of nodes

4　结论

本文提出了一种新的基于 FP_Growth 的频繁项目集并行挖掘算法 NPFP_Growth,该算法基于 Map/Reduce 并行计算模型,利用 HDFS 实现数据存储,在 Hadoop 平台上加以实现,与目前已有的相关并行算法相比,能有效简化挖掘过程. 实验结果表明 NPFP_Growth 算法在面向大数据的频繁模式挖掘中具有较高的计算效率和较好的可伸缩性.

[参考文献]

[1] HAN J,PEI J,YIN Y. Mining frequent patterns without candidate generation[J]. ACM SIGMOD Record,2000,29(2):1-12.

[2] ÖZDOGAN G Ö,Abul O. Task-Parallel FP_growth on cluster computers[C]//Proceedings of the International Symposium on Computer and Information Science,London,UK,2010:383-388.

[3] TANBEER S K,AHMED C F,JEONG B S. Parallel and distributed frequent pattern mining in large databases[C]//11th IEEE International Conference on High Performance Computing and Communications,Seoul,Korea,2009:407-414.

[4] SHEN X L,TAO L. Association rules parallel algorithm based on FP-tree[C]//2010 2nd International Conference on Computer Engineering and Technology,Century City New International Convention & Exhibition Center,Chengdu,China,2010,4:687-689.

[5] TU F,HE B. A parallel algorithm for mining association rules based on FP-tree. Advances in computer science,environment,ecoinformatics,and education[M]. Berlin,Heidelberg:Springer,2011:399-403.

[6] 金桃,何艳珊,宋伟国,等. 一种简单有效的并行化频繁项集挖掘算法[J]. 微计算机信息,2010(18):147-149.

[7] LI H,WANG Y,ZHANG D,et al. PFP:parallel FP-Growth for query recommendation[C]//Proceedings of the 2008 ACM Conference on Recommender Systems,Lausanne,Switzerland,2008:107-114.

[8] 章志刚,吉根林. 一种基于 FP-Growth 的频繁项目集并行挖掘算法[J]. 计算机工程与应用,2014(2):103-106.

[责任编辑:顾晓天]