文章编号:1000-5013(2014)05-0523-05 **doi**:10.11830/ISSN.1000-5013.2014.05.0523

FP-Growth的并行加权关联规则挖掘算法

李翔,刘韶涛

(华侨大学 计算机科学与技术学院,福建 厦门 361021)

摘要: 基于FP-Growth算法,提出一种并行加权的关联规则挖掘(PWARM)算法,证明其满足加权向下封闭性.使用MapReduce计算模型,在分布式集群中并行挖掘出关联规则.实验结果表明:该算法可以满足数据权重不同的需求,且在处理大数据集时能有效地提高挖掘的效率.

关键词: 关联规则挖掘; 并行加权; FP-Growth算法; MapReduce; 加权频繁项集

中图分类号: TP 311 **文献标志码**: A

Apriori算法[1]和FP-Growth算法[2]都带有2个默认假设:一是数据库中的数据重要性相同;二是数据在数据库中分布均匀.然而在现实中,数据库中数据并非如此.为了解决这个问题,文献[3]提出了一种加权模型,并引入k-支持期望.文献[4]提出一种基于Apriori算法的加权模型;文献[5]提出了基于最小支持度的加权模型.文献[6]基于文献[4]的加权模型,提出基于FP-Growth算法的加权模型,并进行归一化处理,提高了算法性能.现有的ARM算法在处理大数据集时会遇到内存使用和计算消耗的瓶颈.并行的关联规则挖掘(association rule mining,ARM)算法[7-10]降低了每台机器运行时的内存和计算消耗.文献[7]提出了并行化FP-Growth算法,并进行均衡处理.文献[10]使用MapReduce并行化FP-Growth算法,优化了并行FP-Growth算法的效率,提高了容错性.本文提出一种基于FP-Growth并行加权关联规则挖掘算法,适应在具有给定权重的大数据集中的关联规则挖掘.

1 并行加权关联规则挖掘(PWARM)算法

$I=\{i_1,i_2,i_3,\cdots,i_k\}$是由$k$个不同项目组成的集合,DB$=\{T_1,T_2,\cdots,T_n\}$是事务数据库,其中每个事务$T_i(i=1,2,\cdots,n)$包含唯一的事务标志TID和$I$的一个子集.

定义1 假设项目集$I=\{i_1,i_2,i_3,\cdots,i_k\}$中的每个项目$i_j(j=1,2,\cdots,k)$有一个权重$W(i_j)$,且$0<W(i_j)<1$.项目有了权重,其所属项集$X$($X$是$I$的一个子集)也就有权重$W_I(X)=\sum_{i_j\in X}W(i_j)/|X|$.D B中的交易事务权重为$W_T(t)=\sum_{i_j\in t}W(i_j)/|t|$.关联规则$A\Rightarrow B$的加权支持度记为$W_{SUP}(A\cup B)$,定义为同时包含$A$和$B$的事务权重之和除以所有事务权重之和,即$W_{SUP}(A\cup B)=\sum_{t\in T\&A\cup B\subset t}W_T(t)/\sum_{t\in T}W_T(t)$.

1.1 加权模型

定义2 设最小加权支持度为$W_{min,sup}$,若项集X是加权频繁的,则其加权支持度不小于最小加权支持度,即$W_{sup}(X)\geqslant W_{min,sup}$.

定理1 如果项集X和Y是I的一个子集,即$X\subset I$,$Y\subset I$且$X\subset Y$,则有$W_{sup}(X)\geqslant W_{sup}(Y)$.

证明 在一个交易数据集中,包含X和Y的交易事务集T_X,T_y,$t\in T_y$,必然有$\forall t\in T_x$.因此

收稿日期: 2013-06-27

通信作者: 刘韶涛(1969-),男,副教授,主要从事软件体系结构与软件复用的研究.E-mail:shaotaol@hqu.edu.cn.

基金项目: 国务院侨办科研基金资助项目(09QZR02)

$\sum_{t\in T\&X\subset t} W_T(t) \geqslant \sum_{t\in T\&Y\subset t} W_T(t)$，易得 $W_{sup}(X)\geqslant W_{sup}(Y)$. 证毕.

定理 2 (加权向下封闭性)如果项集 X 和 Y 是 I 的一个子集，即 $X\subset I, Y\subset I$ 且 $X\subset Y$，若 X 不频繁，则 Y 也不频繁.

证明 X 不频繁，由定义 2 可知 $W_{sup}(X)<W_{min,sup}$，再由定理 1 可知 $W_{sup}(Y)\leqslant W_{sup}(X)$，由此可得 $W_{sup}(Y)<W_{min,sup}$，Y 不频繁. 证毕.

1.2 基于 MapReduce 的加权 FP-Growth 算法

FP-Growth 采用分而治之的思想，只需 2 次扫描数据库. 第 1 次扫描数据库，根据项目的出现次数排序产生一个排序好的项目集，记为 F-List；第 2 次将数据库构建成一个 FP-tree. FP-Growth 算法可以根据已经构建好的 FP-tree，针对每一个支持度大于最小支持度的项目生成相应的条件样式树.

基于 MapReduce 的加权 FP-Growth 算法简化了 FP-Growth，省去了构建 FP-tree 和 Head 表的过程. 给定数据库中的项目权重，表 1 为一个实例数据库中的项目权重. 表 1 中：项目权重在 1～10 之间；1 表示权重最小，10 表示权重最大.

表 1 项目名称及权重

Tab. 1 Name and weight of items

项目号	项目	权重
1	a	1
2	b	2
3	c	2
4	f	1
5	m	3
6	p	4

Mapper 过程为数据库中的项目计数，统计出项目的支持度. 统计出事务的权重 $W_T(t_i)$ 和总的事务权重和 $SUM(t)=\sum_{t_i\in T} W_T(t)$，做归一化处理，事务归一化后的权重为 $W_{TN}(t_i)=W_T(t_i)/SUM(t)$，输出条件事务集为〈Key=$i_j$，Value=$\{\{i_0,i_1,\cdots,i_j\},W_{TN}(t_i)\}$〉.

Reducer 过程接受 Mapper 过程的输出，聚集 Key 值相同的输出，并统计出集合中项目的加权支持度，筛选出加权支持度大于最小加权支持度的项目作为条件 FP-tree.

基于 MapReduce 加权 FP-Growth 算法模型，算法伪码如表 2 所示. DB 是一个包含 5 条交易记录的实例数据库，最小加权支持度 $W_{min,sup}=0.5$.

表 2 MapReduce 加权 FP-Growth 算法实例

Tab. 2 Instance of MapReduce weighted FP-Growth algorithm

Map input(transactions) key=value	Sorted transactions $WT(t_i)$, $WT_N(t_i)$	Map outputs(conditional transactions) key=i_j, value=$\{\{i_0,i_1,\cdots,i_j\},WT_N(t_i)\}$
facmp	fcamp|2.2,0.212	p:fcam|0.212 m:fca|0.212 a:fc|0.212 c:f|0.212
abcfm	fcabm|1.8,0.174	m:fcab|0.174 b:fca|0.174 a:fc|0.174 c:f|0.174
bf	fb|1.5,0.145	b:f|0.145
bcp	cbp|2.66,0.257	p:cb|0.257 b:c|0.257
afcmp	fcamp|2.2,0.212	p:fcam|0.212 m:fca|0.212 a:fc|0.212 c:f|0.212

Redice inputs key=i_j, value=$\{\{i_0,i_1,\cdots,i_j\},WT_N(t_i)\}$	Conditional FP-trees
p:{{fcam|0.212},{fcam|0.212},{cb|0.257}}	{c:|0.681}|p
m:{{fca|0.212},{fca|0.212},{fcab|0.174}}	{{f|0.598},{c|0.598},{a|0.598}}|m
b:{{fca|0.174},{f|0.145},{c|0.257}}	{}|b
a:{{fc|0.212},fc|0.174},{fc|0.212}}	{{f|0.598},{c|0.598}}|a
c:{{f|0.212},{f|0.174},{f|0.212}}	{f|0.598}|c

```
Procedure:MapReduce-FP-Growth(DB, W_min,sup)
Procedure Mapper(DB,W_min,sup)
Define and clear F-list F[];
Define and clear SUM(T_i);
Define and clear W_TN(T_i) ;
  foreach Transaction T_i in DB do
    Define and clear W_T(T_i);
    foreach Item i_j in T_i do
        F[i_j]++;
        (W_T(T_i))+=W(i_j)/|T_i|;
    End
Procedure Reducer(key=i_j,value=S({{i_0,i_1,…,i_j},W_TN(T_i)}))
    foreach Item i_m in S({i_0,i_1,…,i_j}) do
        Define and clear SUM_N(i_m);
        SUM_N(i_m)+=M_TN(t_m);
        if SUM_N(i_m)>=W_min sup
            S(i_m,SUM_N(i_m)+={i_m,SUM_N(i_m)};
    End
Call Outputs(〈Key=i_j,Value={S(i_m,SUM_N(i_m))}〉)
```

```
SUM(T_i) +=(W_T(T_i);
End
Sort F[];
foreach Transaction T_i in DB do
    W_TN(T_i) =W_T(T_i)/SUM(T_i);
    foreach Item i_j in T_i do
    Call Outputs
    〈key=i_j,value={i_0,i_1,…,i_j},W_TN(T_i)}〉
    End
End
```

1.3　PWARM 算法

对于给定的 DB，PWARM 算法共 5 个步骤，使用 4 次 MapReduce 并行化加权 FP-Growth 算法，加权 FP-Growth 算法使用加权模型.

步骤 1　碎片化. 将一个数据库碎片化成存储在多个分布 PC 上的局部数据库，每一个局部数据库称为一个碎片. 在现实中，大数据集往往分布存储在一个集群上，步骤 1 就可以省略.

步骤 2　并行计数. 使用 1 次 MapReduce，在每个碎片中并行计算数据库中的每个项目出现次数，即项目的支持度. 根据支持度排序结果集并存储在 F-list 中. F-list 中包含了 DB 的项集和项集中每个项目的支持度.

步骤 3　项集分组. 将 F-list 中的全部项集分成 Q 个分组，分组是为了便于并行计算. 根据实际情况中内存的大小来确定 Q 值的大小，保证每组中的内存都可以满足该组算法的运行. 每个组中包含 F-list 的一个子集，称为 G-list，并为每一个组赋予一个唯一的标记(gid).

步骤 4　并行加权 FP-Growth 是算法的重点，使用一次 MapReduce.

Mapper 过程：根据 G-list 中的每一个项目，并行将每个碎片中的事务派发到 Q 个分组中. Mapper 过程生成了 Q 个键值对，键为每个分组的 gid，值是一个对于 G-list 完整的事务集.

Reducer 过程：并行地在 Mapper 生成的组完整事务集中运行加权 FP-Growth 算法. 构建加权 FP-tree 和加权条件 FP-tree，挖掘出每一个项目的加权频繁项集.

步骤 5　整合. 将步骤 4 生成的结果使用一次 MapReduce 过程来整合，生成完整的频繁项集.

1.3.1　*并行计数*　针对每一个划分好的数据库碎片，使用一次 MapReduce. Mapper 过程的输入为数据库碎片中每一个事务，〈key，value$=T_i$〉($T_i \subset DB$). 输出是〈Key$'=i_i$，Value$'=1$〉($i_i \in I \&\& i_i \in T_i$). 当每一个 Mapper 实例全部计算完成以后，Reducer 过程将 Mapper 产生的输出集合起来产生输出〈Key$''=$i$_i$，Value$''=$Sum(Value$')_{\text{Key}'=\text{Key}''}$〉. 伪码如下所示.

```
Procedure:Mapper(key,value=T_i)
foreach item i_i in T_i do
    Call Output(〈Key'=i_i,Value'=1〉);
End
Procedure Reducer(〈Key'=i_i,Value'=1〉)
```

C←0;
Foreach item i_i in T_i do
C←C+1;
End
Call Output〈Key″=i_i, Value″=C〉

1.3.2 *并行加权* FP-Growth 算法的第3步将F-list划分成Q个组,每一个组都有一个唯一标识gid.本步骤在此基础上并行处理每个数据库碎片.根据G-list中的内容,派发数据库碎片中的每一个事务到一个组独立的事务数据库中.再在这些组独立的数据库中构建加权FP-tree,挖掘出加权的频繁关联规则.具体拆分为一个Mapper和Reducer过程.

Mapper过程.每一个Mapper实例启动时,输入一个数据库碎片,输入可以表示为〈Key,Value=T_i〉;对每一个T_i,将G-list哈希成哈希表H加载到内存中,倒序遍历T_i中的项目$i_j \in T_i$,搜索H中包含i_j的组G_k,从H中删除G_k的全部元素,输出〈key=gid, value=$\{i_1, i_2, \cdots, i_j\}$〉.

Reducer过程.将Mapper过程产生的结果,根据key值聚集在一起,生成Q个组独立的事务集;再并行地在每个组独立的事务集上,执行基于MapReduce的加权FP-Growth算法;生成碎片数据库的条件FP-tree.

2 实验结果和分析

实验用4台计算机,1台运行WFP(weighted FP-growth)算法,其他3台组成一个Hadoop集群,运行PWARM.实验环境是openSUSE Linux 11,Inter(R) Core(TM) i3-3220 CPU 3.30 GHz,2 G的内存,JDK-1.7.0, Hadoop-0.20.2.实验数据采用http://cdiac.ornl.gov/ftp/上的气象数据进行离散化处理后做为事务数据源,该数据是由中科院2个基站观测到的气象数据,项集包括平均气压,平均温度,平均最高气温等14个气候数据和时间(年月日)数据以及基站编号数据.使用一个随机生成函数给项目一个权重,权重分布在1至10之间.通过关联规则挖掘,找寻气象要素之间的关联规则关系.

随数据集增大,WFP算法和PWARM算法的运行时间,如图1所示.当数据集中事务数超过8万时,WFP算法因为内存不足,算法无法运行下去.PWARM算法使用MapReduce分布式的计算模型,使用3台计算机并行的计算,其运行时间要小于WFP算法的运行时间.当事务集中事务数目增大时,PWARM算法可以使用集群中的共享内存,所以不会产生因为内存不足导致算法运行不了的情况.

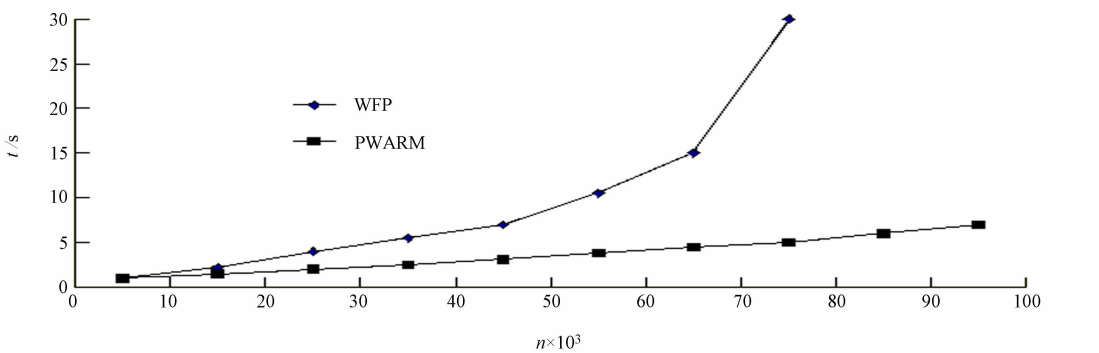

图1 算法运行时间比较

Fig.1 Comparing of running time

3 结束语

提出了基于FP-Growth算法的并行加权的关联规则挖掘算法,利用随机生成函数给每个项目设置不同的权重,从而解决交易数据库中数据项的重要性各不相同,且实际出现频率分布不均的问题.使用MapReduce计算模型来提高算法的运行效率.并行化的计算以解决处理大数据集时内存不足导致算法无法运行的问题.但是没有就权重对算法的影响进行实验,也没有与其他的并行加权算法做横向比较,因此,下一步将就此方面做深入研究,以完善算法.

参考文献：

[1] AGRAWAL R,SRIKANT R. Fast algorithms for mining association rules[C]// International Conference on Very Large Data Bases (VLDB 94). San Francisco: Morgan Kaufmann Publishers Inc,1994:487-499.

[2] HAN Jia-wei,PEI Jian,YIN Yi-wen,et al. Mining frequent patterns without candidate generation[C]// Data Mining and Knowledge Discovery. Hingham M A:Kluwer Academic Publishers,2000:53-87.

[3] CAI C H,ADA W C F,CHENGTAL C H . Mining association rules with weighted item[C]// Proc of International Symposium on Database. Washington D C:Engineering & Applications,1998:178-186.

[4] FENG Tao,MURTAGH F,FARID M. Weighted association rule mining using weighted support and significance framework[C]// Proc of 9ths ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. [S. l]:ACM Press,2003:661-666.

[5] 邹力鹍,张其善. 基于多最小支持度的加权关联规则挖掘算法[J]. 北京航空航天大学学报,2007,33(5):590-593.

[6] CHEN Wen. Mining algorithm for weighted frequent pattern based on FP-tree[J]. Computer Engineering,2012,38(6):24-27.

[7] PRAMUDIONO I,KITSUREGAWA M. Parallel FP-growth on PC cluster[C]// PAKDD. Heidelberg:Springer-Verlag Berlin,2003:141-144.

[8] ZAIANE O R,El-HAJJ M,LU P. Fast parallel association rule mining without candidacy generation[C]// ICDM. Washington D C:IEEE Computer Society,2001:21-26.

[9] ZHOU Le,Balanced parallel FP-Growth with MapReduce information computing and telecommunications[C]// 2020 IEEE. New York:ACM Press,2008:243-246.

[10] LI Hao-yuan,WANG Yi,ZHANG Dong. Chang PFP: Parallel FP-growth for query recommendation[C]// ACM. New York:ACM Press,2008:107-114.

A Parallel Weighted Association Rule Mining Algorithm on FP-Growth

LI Xiang, LIU Shao-tao

(College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China)

Abstract: Proposeing a parallel weighted association rule mining (PWARM) algorithm on FP-Growth algorithm. Testified that the algorithm is satisfy weighted downward closure property, using MapReduce mining association rules in parallel in a distributed cluster. Experimental analysis shows that this algorithm can satisfy the demand of mining the data with different weight in the database, and in dealing with large data sets to speed up the efficiency of mining.

Keywords: association rule mining; parallel weighted; FP-Growth algorithm; MapReduce; weighted frequent items

（**责任编辑**：陈志贤　　**英文审校**：吴逢铁）