第 46 卷 第 11 期 计 算 机 工 程 2020 年 11 月
Vol. 46 No. 11 Computer Engineering November 2020

·先进计算与数据处理· 文章编号：1000-3428(2020)11-0109-08 文献标志码：A 中图分类号：TP391

# 基于哈希存储与事务加权的并行 Apriori 改进算法

李 洁[1,2]，朱洪亮[1,2]，陈玉玲[2]，辛 阳[1,2]

（1. 北京邮电大学 网络空间安全学院，北京 100876；2. 贵州大学 贵州省公共大数据重点实验室，贵阳 550025）

**摘 要：** Apriori 算法能够挖掘事物之间的关联关系，但传统 Apriori 算法每计算一次候选集的支持度，都需要遍历原始事务数据库，多次扫描数据库导致其效率较低。为此，提出一种基于哈希存储与事务加权的改进算法。通过哈希存储的去重特性对事务进行去重，以减少冗余计算。将项目与项集的映射存储到哈希结构中，避免计算候选集的支持度时多次扫描事务数据库。同时开启多个线程，并行计算候选集的支持度，从而提高 Apriori 算法的运行效率。在开源数据集上的实验结果表明，当数据集中事务条数以及重复事务数越多时，该算法相较于传统 Apriori 算法的性能提升越明显，其运行时间与 FP-Growth 算法相近但避免了 FP-Growth 算法内存占用过大的问题。

**关键词：** 关联规则；频繁项集；哈希存储；事务加权；并行计算

**开放科学(资源服务)标志码(OSID)：**

**中文引用格式：**李洁，朱洪亮，陈玉玲，等. 基于哈希存储与事务加权的并行 Apriori 改进算法[J]. 计算机工程，2020，46(11)：109-116.

**英文引用格式：**LI Jie，ZHU Hongliang，CHEN Yuling，et al. Improved parallel Apriori algorithm based on hash storage and transaction weighting[J]. Computer Engineering，2020，46(11)：109-116.

# Improved Parallel Apriori Algorithm Based on Hash Storage and Transaction Weighting

LI Jie[1,2]，ZHU Hongliang[1,2]，CHEN Yuling[2]，XIN Yang[1,2]

（1. School of Cyberspace Security，Beijing University of Posts and Telecommunications，Beijing 100876，China；
2. Guizhou Provincial Key Laboratory of Public Big Data，Guizhou University，Guiyang 550025，China）

**【Abstract】** The Apriori algorithm can mine the association relationships between things，but the traditional Apriori algorithm needs to traverse the original transaction database every time the support of the candidate set is calculated，which reduces the efficiency of the algorithm. To address the problem，this paper proposes an improved algorithm based on hash storage and transaction weighting. The algorithm uses the deduplication feature of hash storage to deduplicate the transactions to reduce redundant calculations. At the same time，the mapping between the transaction set and the itemset is stored in the hash structure to avoid scanning the transaction database for multiple times during the calculation of the support of the candidate set. In addition，the support of the candidate set is calculated in parallel using multiple threads to improve the efficiency of the Apriori algorithm. Experimental results on open-source datasets show that the performance improvement of the proposed algorithm over the traditional Apriori algorithm increases with the number of transactions and repeated transactions in the dataset. Its running time is similar to that of the FP-Growth algorithm while the excessive memory consumption is avoided.

**【Key words】** association rule；frequent itemset；hash storage；transaction weighting；parallel computing

**DOI：**10.19678/j.issn.1000-3428.0056714

## 0 概述

数据挖掘是指从海量、不完整以及模糊的实际应用数据中，提取出人们事先不知道但又可能有价值的信息和知识的过程[1]。关联规则挖掘是数据挖掘领域中的一个重要研究方向，其最早由 AGRAWAL

**基金项目：**国家重点研发计划(2017YFB0802300)；贵州省科技重大专项(20183001)；贵州省公共大数据重点实验室开放课题(2018BDKFJJ008，2018BDKFJJ020)。

**作者简介：**李 洁(1993—)，女，硕士研究生，主研方向为网络安全、数据挖掘；朱洪亮，讲师、博士；陈玉玲，副教授；辛 阳，教授。

**收稿日期：**2019-11-26 **修回日期：**2020-01-04 **E-mail：**cherry.beijing@foxmail.com

等人[2]针对超市购物篮问题分析提出,目的是为了发现超市交易数据库中不同商品之间的关联关系。经典关联规则挖掘算法包括 Apriori 算法[2]和 FP-Growth 算法[3]。Apriori 算法扩展性较好,可以应用于并行计算等领域,但是其多次扫描事务数据库,每次利用候选频繁集产生频繁集,需要很大的 I/O 负载。FP-Growth 算法利用树形结构,无需通过候选频繁集而直接产生频繁集,大幅减少了扫描事务数据库的次数,从而提高了算法效率。目前,Apriori 和 FP-Growth 2 种算法均广泛应用于市场营销、网络安全和生物信息学等领域。

随着信息化时代的到来,网络交互呈爆炸式增长,需要处理的数据量从 GB 量级增长到了 PB 量级,其中,非结构化数据占数据总量的 80%~90%,且其价值密度较低,这对数据挖掘算法的性能提出了更高的要求。传统 Apriori 算法由于每计算一次候选集的支持度都需要遍历原始事务数据库,因此需要多次扫描数据库,效率较低,不能满足大数据处理的要求,对 Apriori 算法进行改进显得尤为必要。

本文针对传统 Apriori 算法运行效率低的问题,提出一种基于哈希存储与事务加权的并行 Apriori 算法。利用哈希结构对事务进行去重,同时将项目与项集的映射存储在哈希结构中,避免计算候选集的支持度时多次扫描事务数据库。同时,通过多线程并行计算候选集的支持度来提高 Apriori 算法的运行效率。

## 1 关联规则挖掘相关研究

目前,国内外学者对关联规则挖掘算法进行了大量研究,对挖掘算法的改进主要分为以下 4 类:

1)基于数据存储结构的改进算法。该类改进主要利用矩阵、树和图等数据结构对事务信息进行存储,以达到减少数据库扫描次数的目的[3-5]。文献[6]提出一种改进的 Apriori 算法,其通过生成布尔矩阵减少事务数据库的遍历次数,降低了关联规则挖掘的时间和空间复杂度,但是当数据量较大时,该算法对内存的要求较高。文献[7]提出一种基于矩阵与权重向量的 Apriori 改进方法,该方法通过权重计算在一定程度上压缩了数据矩阵,但当数据量较大时,布尔矩阵的计算性能会显著降低。文献[8]提出一种 NSFI 算法以挖掘关联规则,其通过使用哈希表来存储与频繁 1 项集相关联的 $N$ 个事务,计算候选集的支持度时只需查找对应 2 个列表之间的交集,从而提高了算法的运行效率和内存使用率,但该算法取交集的效率较低,且未考虑事务重复的情况,当事务重复较多时,其会存储大量的冗余数据,导致性能降低。

2)基于多种算法相结合的改进算法。将关联规则算法与其他算法相结合的改进方式[9],可以发挥关联规则算法与其他算法的优势。文献[10]提出一种基于遗传算法来寻找频繁项集的 GNA 算法,其结合 Apriori 算法和遗传算法的特点,设计 $k$ 步挖掘过程,利用交叉算子产生候选项集和变异算子从而筛选频繁项集,在避免多次扫描数据库的同时减少冗余。文献[11]提出一种 Apriori 算法和遗传算法相结合的方法,其采用支持度、置信度和覆盖度作为规则的评价指标,与传统 Apriori 算法相比,该方法可以在相同的支持度下找到更多满足用户期望的规则,并减少无用规则的生成。文献[12]将 Apriori 算法和图形计算相结合,提出一种图形计算方法 ANG 来进行频繁项集挖掘。当 $k$ 很小时,使用 Apriori 算法计算 $k$ 项候选集的支持度,当 $k$ 增大时,使用图形计算方法计算 $k$ 项候选集的支持度,并给出公式来确定何时从 Apriori 算法转换到图形计算方法。相较于传统 Apriori 算法和图形计算方法,ANG 方法在性能上有很大提升。但是,基于多种算法相结合的改进算法大多计算过程复杂,且混合算法中的参数容易受到数据集的影响。

3)基于 MapReduce[13]等并行处理技术的改进算法。该类改进算法主要利用并行处理技术,对频繁集的查找过程做并行计算,从而缩短算法的运行时间。文献[14]提出一种并发关系关联规则挖掘(CRAR)方法,该方法利用并发性有效缩短了挖掘关联规则的时间,但是其缩放效率和并行运算效率不高,还需进一步优化。文献[15-17]利用开源框架 Hadoop 中的 MapReduce 并行处理技术,将数据集按事务进行分组,利用分布式计算的优势使各节点并行地完成候选项集生成与剪枝操作,从而提高 Apriori 算法的性能,但是此类改进算法采用 HDFS(Hadoop Distributed File System)存储系统,每次计算的中间结果都会写入磁盘,造成很多不必要的I/O 负载,降低了算法的执行效率。针对上述问题,文献[18]将关联规则算法 FP-Growth 与 Spark 框架进行结合,计算的中间结果不再写入磁盘,并针对挖掘频繁项集过程中分组不均衡的问题给出解决方案,提高了 FP-Growth 算法的并行运算效率,但此类改进算法并行计算的效果参差不齐且不稳定,容易受数据分组算法的影响,且应用框架不够轻量,不利于应用的迁移和扩展。

4)基于约束条件的改进算法。该类改进算法主要针对关联规则挖掘算法中的频繁集补充约束条件,降低无意义的候选频繁集产生。文献[19]提出一种基于 SET-PSO 的正负关联规则挖掘方法,其通过粒子群优化从数据库中生成关联规则,同时考虑

属性的正相关性和负相关性,从而减少候选频繁集的产生,提升 Apriori 算法的运行效率,但是该算法生成正负关联规则的相关系数往往需要人工确定。文献[20]针对序列数据边界难以确定的问题,提出一种基于模糊关联规则挖掘(DOFARM)的新型参数和度量动态优化方法,其通过使用一定范围内的分界值平滑地分离 2 个连续的分区,并为原始数据集生成模糊集制定相应的隶属函数,有效解决了连续数据的分组问题,同时对分组参数进行动态优化,在实验数据集上取得了较好的效果。基于约束条件的改进算法多数是针对某种情况下的特定问题而提出。

## 2　并行 Apriori 改进算法

### 2.1　传统 Apriori 算法

关联规则挖掘算法主要用来查找和分析隐藏在项集之间的类似 $X\Rightarrow Y$ 的规则,Apriori 算法[2]是该领域的最经典算法之一,用来查找海量数据中有价值的隐藏关联知识。Apriori 算法的经典应用是通过对超市交易数据的关联分析,发现“尿布”和“啤酒”之间的关联关系,从而使超市获得了可观的经济收益。Apriori 算法中的相关概念[21]如下:

1)项目和项集。设 $I=\{i_1,i_2,\cdots,i_m\}$ 为项的集合,即项集。项集中的每个 $i_k(k=1,2,\cdots,m)$ 称为项目(item)。项集的长度就是项集 $I$ 含有的项目数量,长度为 $k$ 的项集在本文中称为 $k$-项集($k$-itemset)。

2)事务和事务数据库。每个事务(transaction)都是项集 $I$ 的一个子集,记为 $T$,即 $T\subseteq I$。本文使用事务 ID 来区分不同的事务,以方便频繁集的查找和计数。事务数据库 $D$ 是全部事务的集合,本文用 $|D|$ 来表示 $D$ 中包含的事务个数。

3)项集的支持度。对于项集 $X,X\subset I$,本文使用 count($X\subseteq T$)表示数据库 $D$ 中包含 $X$ 的事务数量,则项集 $X$ 的支持度定义为:

$$\text{support}(X)=\frac{\text{count}(X\subseteq T)}{|D|} \tag{1}$$

4)项集的最小支持度和频繁集。在查找关联规则的过程中,项集要满足一个指定的支持度阈值,这个指定的支持度阈值就是项集的最小支持度,记为 $\sup_{\min}$。当一个项集的支持度不低于 $\sup_{\min}$ 时,称该项集为频繁项集,即频繁集;不满足此条件的项集,称为非频繁集。$k$-项集的支持度若不低于 $\sup_{\min}$,通常称为 $k$-频繁集,记作 $L_k$。

5)关联规则。本文定义关联规则的形式如下:

$$R:X\Rightarrow Y \tag{2}$$

其中,$X\subset I,Y\subset I$,且 $X\cap Y=\varnothing$。规则 $R$ 表示当一个事务中出现 $X$ 时,在某一概率下该事务中也会出现 $Y$。本文称 $X$ 为规则 $R$ 的条件,$Y$ 为规则 $R$ 的结果。关联规则 $R:X\Rightarrow Y$ 反映了如下规律:当 $X$ 中的项目出现时,$Y$ 中的项目也会随之出现。

6)关联规则的支持度。对于关联规则 $R:X\Rightarrow Y$,$X\subset I,Y\subset I$,且 $X\cap Y=\varnothing$,事务数据库中同时包含 $X$ 和 $Y$ 的事务个数与所有事务个数之比为规则 $R$ 的支持度,记为 support($X\Rightarrow Y$),表示为:

$$\text{support}(X\Rightarrow Y)=\frac{\text{count}(X\cup Y)}{|D|} \tag{3}$$

7)关联规则的可信度。对于规则 $R:X\Rightarrow Y,X\subset I$,$Y\subset I$,且 $X\cap Y=\varnothing$,其可信度是指同时包含 $X$ 和 $Y$ 的事务个数与包含 $X$ 的事务个数的比值,记为 confidence($X\Rightarrow Y$)。可信度反映了若事务中出现 $X$ 则事务中同时出现 $Y$ 的概率,表达式如下:

$$\text{confidence}(X\Rightarrow Y)=\frac{\text{support}(X\cup Y)}{\text{support}(X)} \tag{4}$$

8)连接和剪枝。当 2 个长度相同的频繁项集只有一个项不同时,将它们连接在一起产生候选频繁集的过程称为连接。根据 Apriori 算法的性质“频繁集的任一子集都是频繁的”来修剪候选频繁集的过程称为剪枝。

传统 Apriori 算法是一个迭代挖掘频繁模式的过程[22],挖掘过程中会不断地产生候选频繁集,然后计算候选集的支持度,由候选集产生频繁集,再由频繁集经过连接、剪枝步骤生成新的候选集,如此重复,直到无法产生新的频繁集则算法终止。对于项集 $X,X\subset I$,若 support($X$) $\geqslant\sup_{\min}$,则 $X$ 为频繁集。传统 Apriori 算法流程如图 1 所示。传统 Apriori 算法每次计算候选频繁集的支持度,都需要遍历一次事务数据库,存在扫描数据库频繁、产生候选项集多、耗时较长等问题,在实际应用中往往运行效率较低。本文针对 Apriori 算法的不足,提出一种基于哈希存储与事务加权的并行 Apriori 改进算法,以提升传统 Apriori 算法的性能。

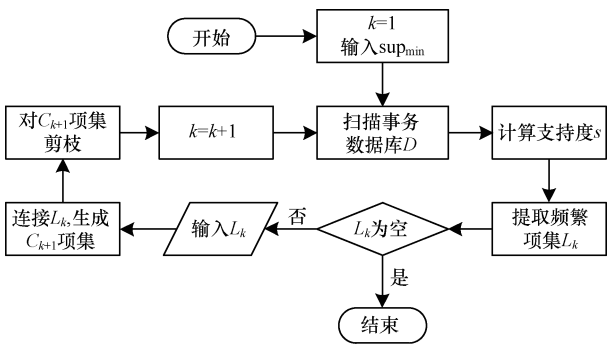


**图 1　传统 Apriori 算法流程**

**Fig. 1　Procedure of traditional Apriori algorithm**

### 2.2　Apriori 算法改进

本文利用哈希存储的去重特性对事务进行去重,在一定程度上压缩事务集,同时将项集与事务集的映射存储在HashMap中,计算支持度时只需计算

项集对应事务集的权重,从而减少事务数据库的扫描次数。然后利用 MapReduce 编程思想开启多个线程,并行计算候选频繁集的支持度,从而提高算法的运行效率。

2.2.1 算法描述

原始 Apriori 算法每计算一次候选集的支持度,都需要遍历一次原始事务数据库,因此需要多次扫描原始事务数据库,当数据库较大时,会进行大量的 I/O 操作,导致算法性能下降。对此,本文采用以下改进方式:

1)遍历原始事务数据库,利用 HashSet 的去重特性来存储事务集,并对重复的事务进行计数,将其记为该事务的权重。当事务列表重复项较多时,该步骤能大幅压缩事务数据库。

2)在遍历项集时统计当前项目(item)对应的所有事务集合(transactions),对事务集合的权重进行累加即可快速获得支持度计数。

3)在迭代计算 $n$ 项集对应的事务集时,对组成它的 $n-1$ 项集对应的事务集直接做交集操作。此处不需要遍历原始数据库,只对项集对应的事务集合进行操作,从而有效减小了遍历长度。最后对事务集合(transactions)的事务权重进行累加即可快速获得支持度计数。为了提高取交集操作的速度,本文采用数据库视图的思想,不直接操作原始数据,而是先生成一个集合视图,集合视图是对原始集合的一系列逻辑操作,通过集合视图可以快速获取交集是否为空、交集大小是否符合要求等信息。若符合要求,再遍历集合视图,这样可以过滤掉大部分取交集的耗时操作,大幅提升取交集的效率。由于事务集合采用 HashMap 存储,2 个集合取交集时,只需遍历其中一个集合,然后直接通过哈希映射判断当前事务是否存在另一个集合中,保证取交集的遍历操作能在 $O(n)$ 的时间复杂度内完成。同时,由于集合视图只是一系列逻辑运算,因此大幅减少了内存占用。

4)哈希函数的选取。为了避免哈希函数的冲突,本文使所有项集的每个字符均参与哈希计算,首先将项集转为字符串,然后取其中每个字符的值作累加并左移 5 位,获得哈希值,这样可以尽量避免冲突。同时,为了减少重复计算,使每个项集缓存计算的哈希值。

5)在每次迭代计算项集的支持度时,开启多个线程并行计算。并行运算属于 CPU 密集型的运算,CPU 核数越多,并行度越高,计算速度越快。并行计算示意图如图 2 所示,Apriori 改进算法流程如图 3 所示。

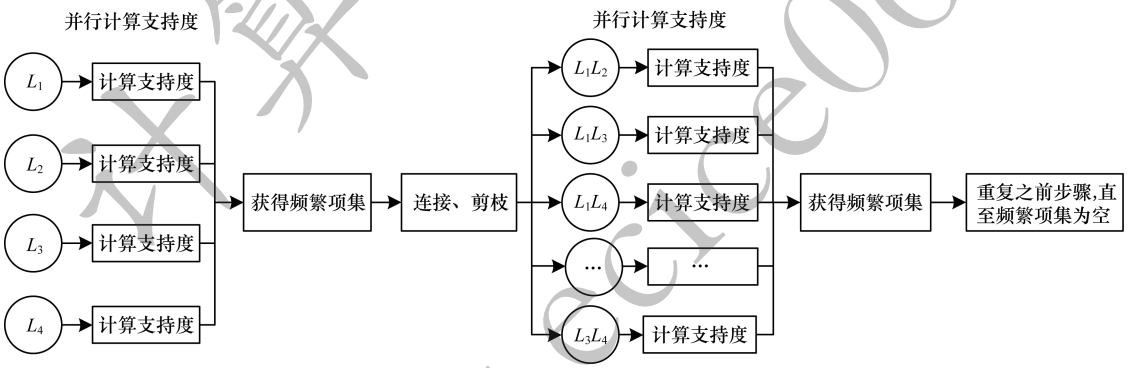


**图 2 并行计算示意图**

**Fig. 2 Schematic diagram of parallel computing**

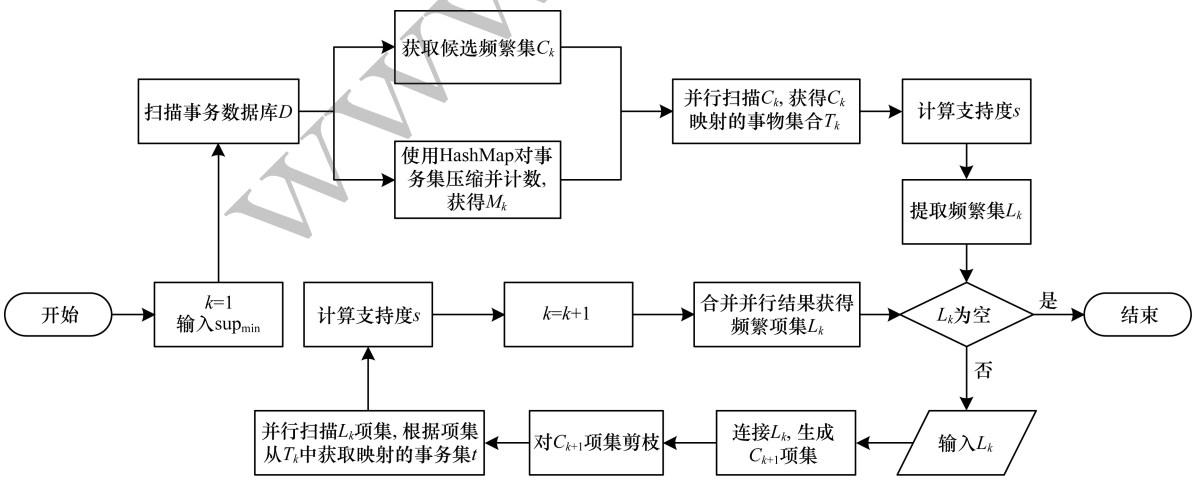


**图 3 Apriori 改进算法流程**

**Fig. 3 Procedure of improved Apriori algorithm**

Apriori 改进算法描述如下:

**算法 1**　Apriori_improve 算法

**输入**　事务数据库 $D$,最小支持度阈值 $sup_{min}$

**输出**　所有频繁集 $L$

1. 扫描事务数据库 D,记录项集 $C_1$、事务集 transactionList 以及项集到事务集的映射 transMap,同时对重复事务 T 计数,记为权重 w

2. 根据映射 transMap 以及 w,并行计算 $C_1$ 中的项目支持度,得到 1-频繁集 $L_1$

3. 若 $L_1 = = \varnothing$,无频繁集,算法结束

4. for( $k = 2; L_{k-1} \neq \varnothing; k + +$ ){

5. $C_k$, transMap = Apriori_gen( $L_{k-1}$, transMap);//根据 //k - 1频繁集产生 k 候选集

6. for each $c \in C_k$ {//开启多线程并行计算

7. localTrans = transMap. get( c);

8. for each transaction ∈ localTrans{

9. supportCount + = weightMap. get( transaction) ;

10. }

11. support = supportCount/Len( transactionList)//计算 c //的支持度

12. if( support > = $sup_{min}$ ){

13. $L_k = L_k \cup c$

14. }

15. }

16. }

17. return $L = L \cup L_k$

Apriori_gen 算法的主要功能是对候选频繁集进行连接和剪枝,描述如下:

**算法 2**　Apriori_gen 算法

**输入**　上一次循环扫描的结果 $L_{k-1}$, transMap

**输出**　候选频繁集 $C_k$, transMap

1. for each $l_1 \in L_{k-1}$

2. for each $l_2 \in L_{k-1}$

3. if( $l_1[1] = = l_2[1] \&\& \cdots \&\& l_1[k-2] = = l_2[k-2] \&\& l_1[k-1] = = l_2[k-1]$ ){

4. $c = l_1 \oplus l_2$//将只差一项的 2 个项集连接在一起

5. trans1 = transMap. get( l1)

6. trans2 = transMap. get( l2)

7. trans = trans1 ∩ trans2

8. transMap. add( c, trans)

9. if 存在 c 的子集不在 $L_{k-1}$ 中//剪枝

10. delete c;

11. else $C_k = C_k \cup \{c\}$

12. }

13. return $C_k$

本文提出的 Apriori 改进算法通过扫描原始事务数据库,获取事务集以及项目到事务集的映射关系(Map),并将其存储在哈希映射(HashMap)中,同时记录事务重复出现的次数并开启多个线程,并行计算候选频繁集的支持度。哈希存储具有以下优势:

1)自动去重,减少程序运行所需要的空间,降低算法的空间复杂度。

2)由于改进算法存储了项目与事务之间的映射关系,在计算项目的支持度时,无需重新遍历原始数据库,只需取对应的事务集并做交集运算即可,有效提高了计算速度。

3)使用哈希运算获得局部锁,可以提升算法在计算过程中的并行性能。当数据库中重复的事务较多时,计算事务权重能够大幅压缩事务集,从而提高计算速度。

4)针对关联规则挖掘问题,对哈希函数进行优化,降低了哈希冲突的概率,同时对已计算出的项集对应的哈希值进行缓存,避免了重复计算。在优化取交集运算时,利用数据库视图的思想,利用集合视图能够快速获取交集是否为空、交集大小是否符合要求等信息,避免了直接操作原始数据,在大幅减少耗时操作的同时又避免了数据的复制,降低了内存的使用。在计算候选集的支持度时,开启多个线程并行计算支持度,这样能够充分发挥硬件设备的性能,并且在事务量足够大时提高算法的运行效率。

2.2.2　算法分析

本文对并行 Apriori 改进算法的正确性和运算效率进行理论分析,分析基于以下假设:

事务数据库为 $D$,项集为 $I = \{i_1, i_2, \cdots, i_m\}$,项目为 $i_k (k = 1, 2, \cdots, m)$,事务为 $T$,事务个数为 $|D|$,事务权重为 $w = \{w_1, w_2, \cdots, w_m\}$,其中,$T_i$ 的权重为 $w_i$,重复事务数与总事务数的比值记为重复率 $\delta$,$T = w_1 T_1 + w_2 T_2 + \cdots + w_m T_m$,关联规则 $R: X \Rightarrow Y$,其中,$X \subset I, Y \subset I$,且 $X \cap Y = \varnothing$。

对于项集 $X, X \subset I$,$X$ 的支持度为:

$$\text{support}(X) = \frac{\sum_{i \in \{i \mid X \subseteq T_i\}} w_i}{|D|} = \frac{\sum_{i \in \{i \mid X \subseteq T_i\}} \text{count}(T_i)}{|D|} = \frac{\text{count}(X \subseteq T)}{|D|} \tag{5}$$

关联规则 $R: X \Rightarrow Y$ 的支持度为:

$$\text{support}(X \Rightarrow Y) = \frac{\sum_{i \in \{i \mid X \subseteq T_i, Y \subseteq T_i\}} w_i}{|D|} = \frac{\sum_{i \in \{i \mid X \subseteq T_i, Y \subseteq T_i\}} \text{count}(T_i)}{|D|} = \frac{\text{count}(X \cup Y)}{|D|} \tag{6}$$

关联规则 $R: X \Rightarrow Y$ 的可信度为:

$$\text{confidence}(X\Rightarrow Y)=\frac{\sum_{i\in\{i|X\subseteq T_i,Y\subseteq T_i\}}w_i}{\sum_{i\in\{i|X\subseteq T_i\}}w_i}=\frac{\sum_{i\in\{i|X\subseteq T_i,Y\subseteq T_i\}}\text{count}(T_i)}{\sum_{i\in\{i|X\subseteq T_i\}}\text{count}(T_i)}=\frac{\text{count}(X\cup Y)}{\text{count}(X)}=\frac{\text{support}(X\cup Y)}{\text{support}(X)} \tag{7}$$

Apriori 改进算法的正确性得证。

Apriori 算法的时间复杂性主要表现在对事务集的访问上，假设事务数 $|D|=n$，$b$ 为平均事务含有项目数，$q$ 为候选频繁集的平均集合大小，迭代次数为 $m$，则 $k$-频繁集 $L_k$ 产生 $k+1$ 候选频繁集的时间复杂度为 $O\left(\frac{L_k(L_k-1)}{2}\right)$。

传统 Apriori 算法总的访问次数为：

$$T(k)=nb+nbL_1^2+nbL_2^2+\cdots+nbL_{k-1}^2=nb\left(1+\sum_{j=1}^{k-1}L_j^2\right)=nb(1+mq^2) \tag{8}$$

因此，传统 Apriori 算法的时间复杂度为 $O(mq^2nb)$。

设本文 Apriori 改进算法中数据并行计算时分组为 $D=\{D_1,D_2,\cdots,D_p\}$，$p$ 为分组个数，则 $k$-频繁集 $L_k$ 产生 $k+1$ 候选频繁集的时间复杂度为 $O\left(\frac{L_k(L_k-1)}{2p}\right)$。

Apriori 改进算法的总访问次数为：

$$T(k)=n(1-\delta)b+n(1-\delta)bL_1^2+n(1-\delta)bL_2^2+\cdots+n(1-\delta)bL_{k-1}^2=n(1-\delta)b\left(1+\sum_{j=1}^{k-1}L_j^2\right)=n(1-\delta)b\left(1+m\left(\frac{q}{p}\right)^2\right) \tag{9}$$

因此，Apriori 改进算法的时间复杂度为 $O\left(m\left(\frac{q}{p}\right)^2nb\delta\right)$。

根据以上分析可以看出，本文提出的 Apriori 改进算法比原始 Apriori 算法效率至少提高 $(1-\delta)p^2$ 倍，且重复事务越多，并行度越高，效率提升效果越明显。

### 2.2.3 Apriori 改进算法的应用框架

在解决实际的数据挖掘问题时，先要对历史数据进行预处理以及特征提取，然后利用本文提出的 Apriori 改进算法对训练数据实现关联规则挖掘，找出频繁集，建立规则库，最后将待测数据与规则库中的规则进行模式匹配，并根据新信息动态更新所建立的模型。Apriori 改进算法的总体应用框架如图 4 所示。

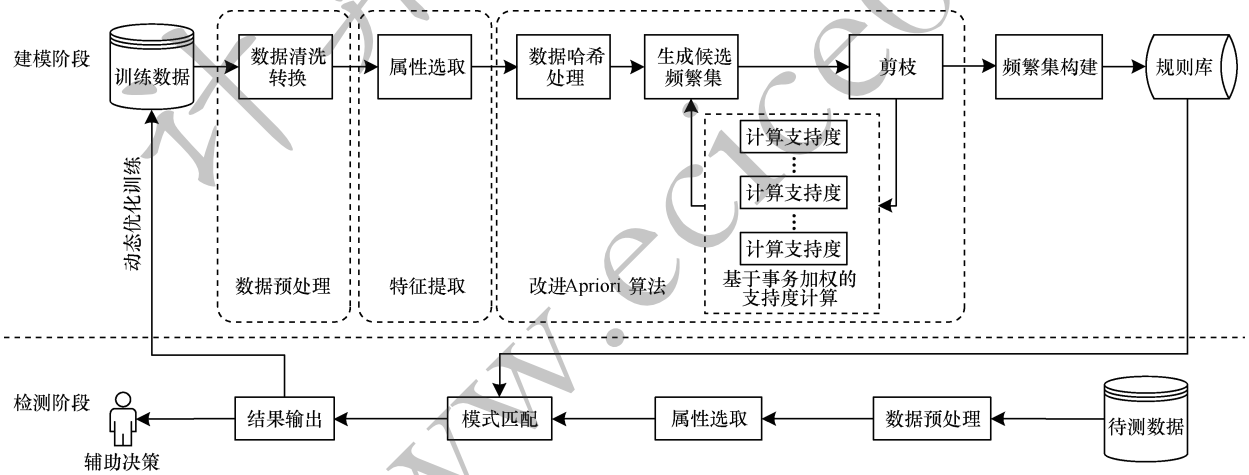


**图 4 Apriori 改进算法的应用框架**

**Fig. 4 Application framework of improved Apriori algorithm**

## 3 实验分析与验证

本文通过实验来验证 Apriori 改进算法的有效性，实验数据集采用 360 日志数据，该数据集是 2019 年奇虎 360 公司举办的网络攻防比赛所用数据，其中的日志数据包括 Web 警告信息、IP 基础信息、日常访问行为信息和终端行为信息等多个维度的网络行为数据。360 日志数据集共包含 2018 年 12 月份 31 天的日志信息，每天的日志信息量大小为 11.0 M ~ 1.7 G。实验环境：处理器为 Intel® Core i7-9700K 3.60 GHz 8 核，内存为 32 GB，硬盘为 1 T，操作系统为 Ubuntu 18.04.3 LTS，运行环境为 JDK 13。

为了验证本文所提算法的时间复杂度和空间复杂度，选取数据集中某 5 天的日志数据，使用传统 Apriori 算法、FP-Growth 算法和本文 Apriori 改进算法对实验数据进行关联规则挖掘，并统计和比较以上算法的运行时间和内存占用情况，结果如图 5、图 6所示。

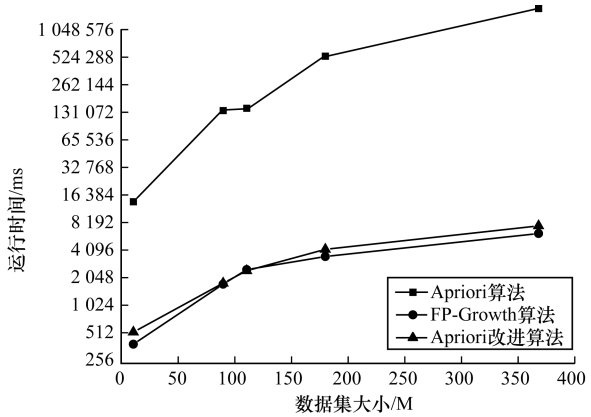


**图 5　不同数据集大小下的算法运行时间对比**

**Fig. 5　Comparison of algorithm running time under different dataset sizes**

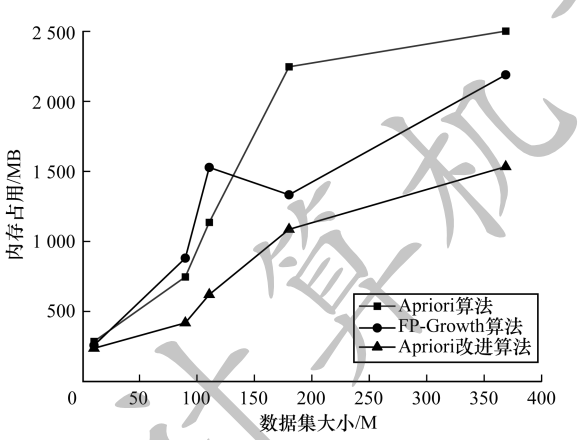


**图 6　不同数据集大小下的算法内存占用对比**

**Fig. 6　Comparison of algorithm memory consumption under different dataset sizes**

图 5、图 6 所示分别为最小支持度为 0.15、改进 Apriori 算法线程数为 8 时,不同事务数情况下 3 种算法运行时间和内存占用的对比情况。从中可以看出,相较于传统 Apriori 算法,本文 Apriori 改进算法性能提升明显。在算法运行时间上,Apriori 改进算法取得了与 FP-Growth 算法相近的效果,同时又避免了 FP-Growth 算法内存占用过大的问题。FP-Growth 算法需要递归生成条件 FP-tree,内存开销较大,本文 Apriori 改进算法采用哈希存储与集合视图相结合的改进方案,有效降低了算法的内存开销。

为了验证本文所提算法的稳定性并探究并行度对该算法的影响,针对不同的支持度和并行度进行实验,并统计和对比算法的运行时间,结果如图 7、图 8所示。

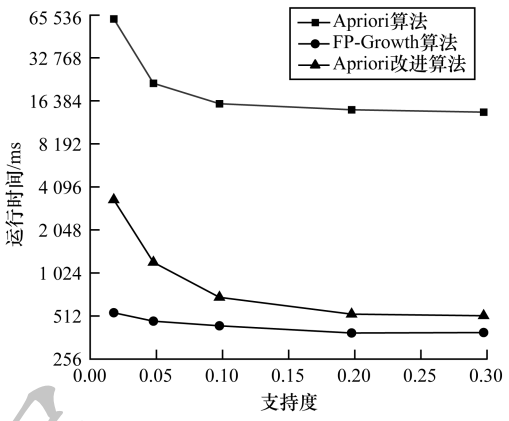


**图 7　不同支持度下的算法运行时间对比**

**Fig. 7　Comparison of algorithm running time under different support degrees**

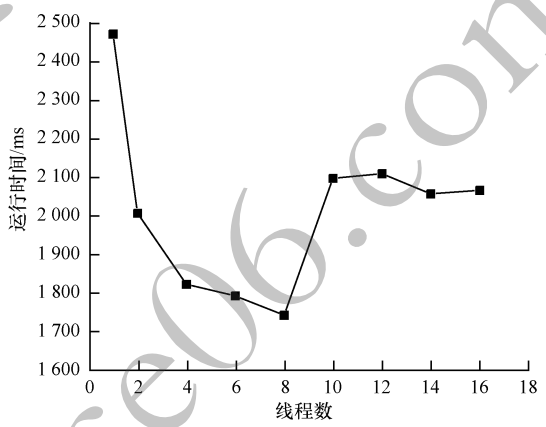


**图 8　不同并行度下的算法运行时间**

**Fig. 8　Algorithm running time under different parallel degrees**

图 7 所示为数据集大小为 11 M、最小支持度分别取 0.02 ~ 0.30 情况下算法的运行时间对比情况。从图 7可以看出,支持度越小,挖掘频繁项集所需时间越长,相较于传统 Apriori 算法,本文 Apriori 改进算法的运行时间随支持度的变化幅度较小,算法较为稳定。

图 8 所示为数据集大小为 91 M、最小支持度设置为 0.15 时改进算法的运行时间随并行度的变化情况。从图 8 可以看出,在线程数小于 8 时,Apriori 改进算法的运行时间随并行度的增加而降低,在线程数为 8 时,Apriori 改进算法的运行效率达到最优,在线程数大于 8 时,Apriori 改进算法的运行效率基本保持平稳状态。

## 4　结束语

本文针对传统 Apriori 算法运行效率低的问题,提出一种基于哈希存储与事务加权的并行 Apriori 改进算法。利用哈希存储减少对原始事务数据库的

扫描次数,记录事务的权重,对事务进行去重和压缩,并通过记录项目所在事务的 ID 建立映射关系,从而减少冗余计算。开启多个线程,并行计算候选频繁集的支持度,使硬件设备的性能得到充分发挥,从而提升算法的运行效率。选取 360 网络攻防大赛数据集进行验证,结果表明,该算法能够大幅降低关联规则挖掘所需的时间以及内存占用,有效提升传统 Apriori 算法的性能,在运行时间上取得与 FP-Growth 算法相近效果的同时又避免了 FP-Growth 算法内存占用过大的问题,在处理海量数据时,该算法能够大幅降低设备成本。

相较于分布式并行计算框架,本文并行 Apriori 改进算法更加轻量,易于移植,可扩展性较高,但并行计算所带来的性能提升受限于硬件设备 CPU 的核数,且线程数并非越多越好,线程切换会导致一定的时间开销。下一步将在硬件条件一定的情况下研究最佳并行度的确定方法,以提升本文算法的性能。

## 参考文献


[1] HAN J, KAMBER M. Data mining: concepts and techniques[M]. [S. l.]: Morgan Kaufmann Publishers Inc., 2005.

[2] AGRAWAL R. Mining association rules between sets of items in large databases[EB/OL]. [2019-10-10]. https://cs. fit. edu/ ~ pkc/ ml/ related/ agrawal-sigmod93. pdf.

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

[4] WANG Ling, MENG Jianyao, XU Peipei, et al. Mining temporal association rules with frequent itemsets tree[J]. Applied Soft Computing, 2018, 62: 817-829.

[5] CHANDA A K, SAHA S, NISHI M A, et al. An efficient approach to mine flexible periodic patterns in time series databases[J]. Engineering Applications of Artificial Intelligence, 2015, 44: 46-63.

[6] WANG Feng, LI Yonghua. An improved Apriori algorithm based on the matrix[C]//Proceedings of 2008 International Seminar on Future BioMedical Information Engineering. Washington D. C., USA: IEEE Press, 2008: 152-155.

[7] YANG Qinliu, FU Qunchao, WANG Cong, et al. A matrix-based Apriori algorithm improvement[C]// Proceedings of the 3rd International Conference on Data Science in Cyberspace. Washington D. C., USA: IEEE Press, 2018: 824-828.

[8] VO B, LE T, COENEN F, et al. Mining frequent itemsets using the N-list and subsume concepts[J]. International Journal of Machine Learning and Cybernetics, 2016, 7(2): 253-265.

[9] PADILLO F, LUNA J M, HERRERA F, et al. Mining association rules on big data through MapReduce genetic programming[J]. Integrated Computer-Aided Engineering, 2017, 25(1): 31-48.

[10] WEN Wu, GUO Youqing. Improvement of Apriori algorithm based on genetic algorithm[J]. Computer Engineering and Design, 2019, 40(4): 1922-1926. (in Chinese)
文武,郭有庆. 结合遗传算法的 Apriori 算法改进[J]. 计算机工程与设计,2019,40(4):1922-1926.

[11] DENG Xiaoheng, ZENG Detian, SHEN Hailan. Causation analysis model: based on AHP and hybrid Apriori-genetic algorithm[J]. Journal of Intelligent and Fuzzy Systems, 2018, 35(1): 767-778.

[12] ZHANG R, CHEN W G, HSU T C, et al. ANG: a combination of Apriori and graph computing techniques for frequent itemsets mining[J]. The Journal of Supercomputing, 2019, 75(2): 646-661.

[13] DEAN J, GHEMAWAT S. MapReduce[J]. Communications of the ACM, 2008, 51(1): 107-113.

[14] CZIBULA G, CZIBULA I G, MIHOLCA D L, et al. A novel concurrent relational association rule mining approach[J]. Expert Systems with Applications, 2019, 125: 142-156.

[15] LUNA J M, PADILLO F, PECHENIZKIY M, et al. Apriori versions based on MapReduce for mining frequent patterns on big data[J]. IEEE Transactions on Cybernetics, 2018, 48(10): 2851-2865.

[16] SINGH S, GARG R, MISHRA P K. Performance optimization of MapReduce-based Apriori algorithm on Hadoop cluster[J]. Computers and Electrical Engineering, 2018, 67: 348-364.

[17] XIAO Wen, HU Juan, ZHOU Xiaofeng. PFPonCanTree: a parallel frequent patterns incremental mining algorithm based on MapReduce[J]. Computer Engineering and Science, 2018, 40(1): 15-23. (in Chinese)
肖文,胡娟,周晓峰. PFPonCanTree:一种基于 MapReduce 的并行频繁模式增量挖掘算法[J]. 计算机工程与科学,2018,40(1):15-23.

[18] ZHANG Suqi, SUN Yunfei, WU Junyan, et al. A parallel frequent itemsets mining algorithm based on Spark[J]. Computer Applications and Software, 2019, 36(2): 24-28. (in Chinese)
张素琪,孙云飞,武君艳,等. 基于 Spark 的并行频繁项集挖掘算法[J]. 计算机应用与软件,2019,36(2):24-28.

[19] AGRAWAL J, AGRAWAL S, SINGHAI A, et al. SET-PSO-based approach for mining positive and negative association rules[J]. Knowledge and Information Systems, 2015, 45(2): 453-471.

[20] ZHENG Hui, HE Jing, HUANG Guangyan, et al. Dynamic optimisation based fuzzy association rule mining method[J]. International Journal of Machine Learning and Cybernetics, 2019, 10(8): 2187-2198.

[21] CHEN Zhipo. Data warehouse and data mining[M]. Beijing: Tsinghua University Press, 2009. (in Chinese)
陈志泊. 数据仓库与数据挖掘[M]. 北京:清华大学出版社,2009.

[22] QIAN Guangchao, JIA Ruiyu, ZHANG Ran, et al. One optimized method of Apriori algorithm[J]. Computer Engineering, 2008, 34(23): 196-198. (in Chinese)
钱光超,贾瑞玉,张然,等. Apriori 算法的一种优化方法[J]. 计算机工程,2008,34(23):196-198.


编辑 吴云芳