文章编号：2096-1618(2019)04-0369-06

基于 MapReduce 的改进 Eclat 算法

向春梅， 陈 超
(成都信息工程大学通信工程学院，四川 成都 610225)

摘要：关联规则挖掘一直都是数据挖掘的重要任务，然而随着大数据时代的到来，数据规模呈指数形式增长，传统的串行挖掘算法已经面临着内存和计算资源不足等问题。针对上述问题，提出了一种基于 MapRedcue 并行编程模型的改进 Eclat 算法——IMREclat 算法。IMREclat 算法使用 2 个 MapReduce 任务，主要分为 3 个阶段：首先，平均划分事务数据库，并行挖掘频繁 2 项集。然后，将频繁 2 项集转化为垂直数据格式并利用二进制存储事务列表，按照等价类和其权重值分组。最后，将分组后的数据作为输入，通过利用预剪枝性质改进后的 Eclat 算法并行挖掘所有的频繁项集。实验表明，IMREclat 算法在运行时间上优于现有的 MREclat 算法，并有良好的扩展性能。

关 键 词：数据挖掘；关联规则；频繁项集；MapReduce 模型；Eclat 算法
中图分类号：TP301.6 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2019.04.008

0 引言

数据挖掘一直被认为是在大型事务数据库中有效发现未知模式的一个十分重要的研究课题。它以隐藏在大型事务数据库中的有趣信息吸引了众多研究者的关注。关联规则挖掘是数据挖掘中最重要的领域之一，主要任务是查找频繁项目集和生成关联规则[1]。第二个步骤相对简单，因此大部分研究人员将注意力放在了第一个步骤，即如何从数据库中准确且高效地找出频繁项集。挖掘频繁项集的算法按照数据存储格式可以分为基于水平型数据(Tid-Itemlist)的算法和基于垂直型数据(Item-Tidlist)的算法。Apriori[2] 和 FP-Growth[3] 是前者的典型算法。Apriori 算法需要通过多次扫描数据库，从而统计出项目对应的支持度来筛选候选项集形成频繁项集，这个过程会产生大量的候选项集。FP-Growth 算法则只需要扫描一次数据库，采用 FP-tree 把数据库中的项目信息压缩到内存中，由 FP-tree 递归挖掘频繁项集。但是，该算法在处理稀疏型数据集的时候，其性能也会大大降低[4]。Eclat[5] 算法则是基于垂直型数据的典型算法，仅需扫描一遍数据库，并且可以快速求出项集的支持度，其计算性能一般优于 Apriori 等水平型数据的算法。

随着数据集的扩大，传统的串行算法面临着两个问题：一是受单机内存的限制，很难一次性将大数据集放入内存中；二是受单机计算能力的限制，从大数据集中找到频繁项集需要较长的时间[6]。而分布式和并行计算的方法可以有效地解决上述两个问题，并且还具备资源共享、高透明性、高性价比、高可靠性、高度灵活性等特点，是处理大数据集的最佳策略[7]。目前，将 Eclat 算法移植到 MapReduce 计算模型的研究相对其他算法而言较少。李伟卫等[8] 提出了 PEclat 算法，算法需要迭代运行多个 MapReduce 任务，节点之间的通信时间增加，导致性能很低。Moens 等[9] 提出了 distEclat 算法，采用 3 个 MapReduce 任务，分别执行发现频繁 1 项集、产生频繁 k 项集，挖掘频繁项集子树从而得到频繁项集。Zhang 等[10] 提出了 MREclat 算法，将相同前缀的项目划分到一组，然后采用 Eclat 算法对每组中的数据进行处理。但该算法在数据量很大时，运算量会呈指数型增长，影响了算法的实用性，K. Keerthi等[11] 将频繁 2 项集作为 Eclat 算法的输入，依然使用了多个 MapReduce 任务，而且没有考虑负载均衡问题，导致算法效率不高。

文中提出一种基于 MapReduce 的改进 Eclat 算法——IMREclat 算法，将事务数据库转化为垂直结构的数据集，即 Item-Tidlist；利用二进制存储事务列表来压缩存储空间和简化运算；采用等价类并结合负载均衡思想将频繁 2 项集的项集分组；然后采用改进后的 Eclat 算法挖掘每个组中的频繁项集。

1 相关知识

1.1 基本概念和性质

设项目集合为 $I=\{I_1, I_2, \cdots, I_n\}$，数据 D 是数据库事务中与任务相关的事务集合，$|D|$ 为数据库中事务的总数，$D=\{T_1, T_2, \cdots, T_i, \cdots, T_n\}$，其中 T_i 为事务，每个事务都有一个唯一的标识符，称作 TID。假设用户规定的最小支持度为 min_sup，即 I 在数据 D 中出现

收稿日期：2019-01-03
基金项目：四川省科技计划资助项目(18ZDYF3278)

的最小次数;最小支持度阈值为 S,即 I 在数据 D 中出现的最小次数与 $|D|$ 的百分比;$I.support$ 为 I 的支持度,表示 I 在数据 D 中出现的次数。

定义 1 k 项集。由 k 个项目组成的项目集合称为 k 项集。

定义 2 频繁 k 项集。如果现有一个 k 项集,记做 X,其支持度记为 $X.support$,并且满足 $X.support \geqslant min\_sup$,那么该 k 项集为频繁 k 项集。

性质 1[1] 若项集 X 是频繁的,项集 X_1, X_2 是 X 的子集,则项集 X_1, X_2 都是频繁的。

性质 2[1] 若项集 X 是非频繁的,项集 X_1, X_2 是 X 的超集,则项集 X_1, X_2 是非频繁的。

性质 3 如果频繁 $k-1$ 项集中存在项目 A,并且 A 在所有的频繁 $k-1$ 项集中出现的次数少于 $k-1$ 次,那么任何一个频繁 $k-1$ 项集与包含 A 的频繁 $k-1$ 项集连接生成的 k 项集一定是非频繁的。

对性质 3 采用反正法证明:假设生成的所有的 k 项集是频繁的。由性质 1 可知,频繁 k 项集的所有子集都是频繁的,所以其所有的 $k-1$ 项集是频繁的。对于一个包含项目 A 的频繁 k 项集有且仅有 $k-1$ 个包含 A 的频繁 $k-1$ 集,那么就有 $k-1$ 个,并且包含 A 的频繁 k 项集可能不止一个,所以 A 在所有的频繁 $k-1$ 项集中出现的次数必定大于或等于 $k-1$ 次。这与性质 3 中的条件矛盾,故性质 3 是成立的。

1.2 Eclat 算法

Eclat 算法的主要思想是:首先,扫描 D 将它转化成垂直的数据格式 D' 来存储数据,然后从下而上,通过项目所对应的事务集进行集合的交运算,从而得到项集和该项集的支持度[5]。该算法在实际应用中有较好的效果,是一种性能较好的频繁项集挖掘算法。以下是 Eclat 算法的伪代码。

```
输入:F_k = {I_1, I_2, ···, I_n} //频繁 k 项集
输出:频繁 l 项集,l>k
Eclat(F_k){
    For Each I_i ∈ F_k, F_{k+1} = ∅;
    For Each I_j ∈ F_k, i<j
        /* 将 I_i, I_j 两个项目所在的事务求交集,就是项目集 I_iI_j 所在的事务集 */
    N = I_i ∩ I_j;
        /* 满足最小支持度,则将项目集加入 k+1 项频繁集中 */
    If N.support ≥ min_sup Then
    F_{k+1} = F_{k+1} ∪ N;
        End
    End
    End
    Out(F_{k+1});
    If F k+1 ≠ ∅ Then
    Eclat(F_{k+1}); //根据频繁 k+1 项集,继续查找频繁的 k+2 项集
    End
}
```

在 Eclat 算法中可以直接计算两个项集的 TID 的交集从而得到候选项集和其支持度,同时利用性质 2 提前删除不满足最小支持度的项集获得频繁项集,可以达到一定的压缩结果集的效果。然而,传统的 Eclat 算法并不能利用性质 2 对现有的频繁项集进行剪枝。这种不足在数据量较小时体现不大,但是当数据量较大时,就会产生大量冗余的项集导致算法性能急剧下降[12]。而且,传统的 Eclat 算法应用在规模较大的数据集上,会因为利用集合的交运算求支持度的操作而消耗大量时间和大量系统内存[13]。

1.3 MapReduce 并行编程模型

MapReduce 是 Google 于 2004 年重新引入的一种程序设计模型,用于大型数据集上以大规模并行方式在计算机集群上支持分布式计算[14]。MapReduce 计算模型充分利用了分而治之的思想,将大量数据的处理过程拆分为 Map 阶段和 Reduce 阶段。Map 阶段主要是对输入进行整合,通过定义的输入格式获取文件的信息和类型,并确定读取方式,最终将读取的内容以键值对的形式保存。Reduce 阶段用来对 Map 阶段产生的结果进行二次处理和归并排序,从而计算出最终结果。MapReduce 有许多优点[15]:首先,MapReduce 可以将完整的数据划分为相等大小的数据块,并自动将它们分发到分布式文件系统中,以便用户可以从数据的位置和分布中解脱出来。其次,MapReduce 可以重新执行面临崩溃的任务并获得良好的容错能力。第三,MapReduce 可以通过将较慢节点中未完成的任务重新分配给集群中的空闲节点来增强总的吞吐量。MapReduce 的框架如图 1 所示。

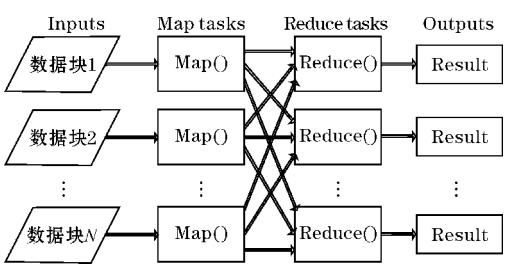

图 1 MapReduce 并行编程模型

1.4 等价类划分

使用 MapReduce 并行框架模型会涉及数据划分,如果直接水平划分数据会导致最终结果不准确。Zaki 等[16]提出一种等价类划分技术,并将该技术应用到并

行 Eclat 算法中,得到了很好的效果。等价类定义为:若集合中存在相同长度且长度为 k 的多个项集,其中前 $k-1$ 项都相同的项,则这些 k 项集都可以互为等价类。

例如:假设有频繁 2 项集 $F_2=\{AB,AC,AD,AE,BC,BD,BE,CD,CE,DE\}$,并且 F_2 是按字典顺序排列的,则其中[B]AB,AC,AD,AE 为等价类[A],BC,BD,BE 为等价类,CD,CE 为等价类[C],DE 不是等价类。对于频繁 2 项集要产生 3 项集,传统的做法是将 F_2 中的项集两两求交集可得到。F_2 中共有 10 个元素,要找到所有的 3 项集需要 $C_{10}^2=45$ 次运算,并且发现找到的 3 项集只有 10 个,这说明 45 次的运算有很多次找到的 3 项集都是重复的,因此多消耗了很多时间。如果利用等价类来查找 3 项集,在[A]中需要 C_4^2 次,在[B]中需要 C_3^2 次,在[[C]]中需要 C_2^2 次,共需要 10 次运算,正好找到所有的 3 项集。

从以上描述来看,使用等价类来挖掘频繁项集会节约很多不必要的运算时间。

2　IMREclat 算法

2.1　算法的基本思想

2.1.1　存储方式

传统的 Eclat 算法采用垂直类型的数据格式并利用矩阵形式来存储垂直数据(Item-Tidlist)。这种存储方式用于存储密集型数据集具有很好的效果,但是当存储稀疏的数据集还是会浪费很多存储空间。文中提出的算法也主要采用垂直类型的数据格式,并利用二进制的形式来存储 Tidlist,减小了存储空间,并且可以直接利用二进制按位与的操作来获取候选项集,减少了运算的时间。

2.1.2　输入数据

目前提出的很多算法都是将现有的水平数据集转化为频繁 1 项集和其对应的事务列表,直接转换为垂直数据集会因频繁 1 项集所对应的列长度较大而造成空间的浪费;且本身转化和由 1 项集求交生成 2 项集需要更多的时间。

若一个水平的数据集包含 100000 条事务数据,共 100 个项目,平均每条事务包含的项目个数为 10,则总的项目个数为 100000×10。将其转化为垂直数据集后,每个项目所对应的事务列表的平均个数为 100000×10÷100=10000。生成 2 项集的过程中需要进行 $C_{100}^2\times2\times10000\approx10^8$ 次计算。如果直接在原水平数据集中查找 2 项集,则需要进行 $C_{10}^2\times100000=4.5\times10^6$。所以采用从水平数据集中直接查找 2 项集,根据最小支持度得到频繁 2 项集。将频繁 2 项集转化为 Item-Tidlist 后作为算法的输入。

2.1.3　数据划分

如果直接将数据集水平划分,可能会导致某些全局频繁项集因不满足局部支持度而被删除,最终可能导致其支持度偏低,甚至导致该项集被错误地判定为非频繁项集而被删除。所以采用等价类划分频繁 k 项集并在等价类中挖掘 $k+1$ 项集来避免上述问题。

2.1.4　负载均衡

提出的改进算法在挖掘频繁 2 项集时,采用均分事务来划分数据集以获得负载均衡;在利用频繁 2 项集挖掘后续频繁项集时则使用等价类划分数据集。由于不同的等价类在挖掘频繁项集的运算复杂度不同,直接将每个等价类分成一组并划分给不同的节点来处理,会导致负载不均衡的问题,因此采用 $W_{[i]}$ 来表示频繁 2 项集中等价类[i]的权重值。

$$W_{[i]}=C_{n_{[i]}}^2=\frac{n_{[i]}\times(n_{[i]}-1)}{2}$$

其中[i]表示以项目集 i 为前缀的等价类,$n_{[i]}$ 表示等价类[i]的个数,$W_{[i]}$ 表示等价类[i]中频繁 2 项集两两运算产生候选 3 项集所需要的运算次数。

在计算频繁 2 项集中所有等价类的权重值后,按权重值降序排列,根据节点的个数,让每个节点分配尽量相同的权重累加值的等价类。

2.1.5　剪枝优化

Eclat 算法采取的是深度优先的搜索策略,可以利用 1.1 节提到的性质 2 来对候选项中不满足最小支持度的项集进行剪枝,但是不能直接对频繁集进行剪枝。文中提出的算法采用了 1.1 节提到的性质 3 对频繁 $k-1$项集中所包含的项目出现的次数少于 $k-1$ 的项集进行剪枝。

以频繁 2 项集 $F_2=\{\{AB\},\{AD\},\{AE\},\{AF\},\{BE\},\{EF\}\}$ 为例,其中 A 出现 4 次,B 出现 2 次,D 出现 1 次,E 出现 3 次,F 出现 2 次,由于 D 出现的次数小于 2 次,所以删除$\{AD\}$。

2.2　算法的步骤描述

IMREclat 算法主要分为 3 个阶段,以下是对该算法步骤的详细描述:

第一阶段:

扫描事务数据库将数据库平均分为 N 个片段,N 为 MapReduce 模型中的节点个数。其中,Map 阶段负责搜索每个部分中的 2 项集,输出<items,Tid>;Reduce 阶段负责合并计算所有分片中的 2 项集以及其支持度。根据最小支持度,删除不频繁的 2 项集,并输出<

items,Tidlist>。其中,items 代表频繁 2 项集,Tidlist 代表该频繁 2 项集所对应的事务列表,频繁 2 项集的支持度就是 Tidlist 的长度。此阶段使用一个 MapReudce 任务进行并行运算,下面是该阶段的伪代码:

输入:S_i// 第个分片,水平数据格式<Tid-Itemlist>
输出:<items,Tidlist>
Map(Tid,Itemlist){
For Each $I_k \in T_i$ //表示 T_i 中第 i 条事务
For Each $I_j \in T_i$ //j>k
Out($I_k \cup I_j$,Tid);
End
End
}
输入:所有的<2-items,Tid>
输出:<2-items,Tidlist>
Reduce(2-items,Tid){
Tidlist=∅;
sum=0;
For Each Tid∈<2-items,Tid>
sum++;
Tidlist=Tidlist∪Tid;
End
If sum≥min_sup Then
Out(2-items,Tidlist);
End
}

第二阶段:

将第一阶段得到的频繁 2 项集的垂直数据格式(Item-Tidlist)中的 Tidlist 用二进制存储。然后将 Item-Tidlist 按频繁 2 项集的字典顺序排列,划分等价类并计算等价类的权重,按权重的加权值将 Item-Tidlist 数据划分为 *N* 个片段,每个分片作为作为第三阶段的输入值。假设划分好的等价类有 *M* 个,一般情况下 *M*>*N*,此阶段不是并行计算的。下面是该阶段的伪代码:

输入:<2-items,Tidlist>,N,M
输出:arrays 数组,长度为 N,每个元素包含一组项目集
Divide(2-items,Tidlist,M,N){
CountAll($W_{[i]}$);
Order(2-items,Tidlist);
sum[ ]=0;//初始化等价类的权值数组
for(j=0;j<N;j++)
/*添加具有相同等价类的数据到数组中*/
<2-items,Tidlist>∈[i]组中*/
Add all in arrays[j]
sum[j]=sum[j]+$W_{[i]}$;
End
for(j=N;j<M;j++)
k=min_index(sum); //找到 sum 中最小值的下标
/*添加具有相同等价类的数据到数组中*/
Addall<2-items,Tidlist>∈[i] in arrays[k]
sum[j]=sum[j]+$W_{[i]}$;
End
Out(arrays);
}

第三阶段:

Map 阶段负责使用改进后的 Eclat 算法搜索所有的频繁项集,在搜索过程中利用 2.1 节证明的性质 3 进行预剪枝,以缩小搜索空间。在得到相应的候选集后利用性质 2 删除不频繁的候选项集,以降低运算复杂度;Reduce 阶段负责合并并输出所有的频繁项集。此阶段使用一个 MapReudce 任务进行并行运算,下面是该阶段的伪代码:

输入:arrays[i] // 划分好的分组
输出:<items,support> //该组中的所有频繁集
Map(arrays[i]){
F_{k_i}=IEclat(arrays[i]);//第个分组的所有频繁 k 项集,k=1,2,3,…
Out(k-items,support)$_i$;
}
输入:<X,X.support>$_i$,i=1,2,3,…n //X 为项目集,X.support 为其支持度
输出:<X,X.support> ,k=1,2,3,… //全局的频繁项集和其支持度
Reduce(X,X.supprt){
For Each <X,X.support>$_i$
X.support=X.support+X.support$_i$;
End
Out(X,X.support);
}

其中 IEclat 方法的伪代码与 Eclat 基本一致,只是在第 7 行后添加了预剪枝的过程:

For all $I_n \in F_{k+1}$
If Count(I_n)<k+1 Then
Delete F_j($I_n \in F_j$,$F_j \in F_{k+1}$) From F_{k+1}
End
End

3 实验设计和分析

由于实验条件有限,实验采用一台内存 16G,处理

器为 Intel(R) Core9(TM) i7-6700HQ,硬盘为 1280G 的笔记本电脑搭建了 4 个节点(1 个 master 节点,3 个 slave 节点)的集群环境。每个节点配置相同,内存 2.5GB,处理器 1 核,硬盘 60GB,操作系统为 Ubuntu16.04。Hadoop 版本为 Hadoop-2.7.4,JDK 版本为 JDK1.8.0_144。使用 Java 语言实现算法。实验数据是从 Frequent Itemset Mining Dataset Repository(http://fimi.ua.ac.be/data/)下载的专门用于挖掘频繁项集的数据集,如表 1 所示。其中 retail 数据集是稀疏数据集,mushroom 和 chess 是密集数据集。

表 1　实验数据集

数据集名称	事务条数/条	大小/kb	事务长度/个
retail	88,163	4,070	/
chess	3,196	335	37
mushroom	8,124	558	23

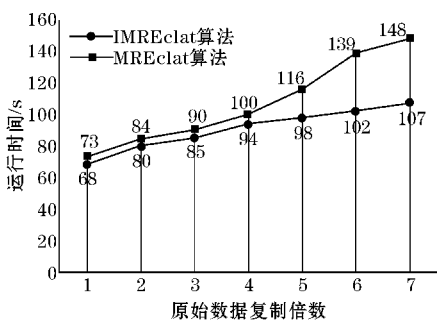

图 2　retail 数据集下的性能对比图(min_sup=0.1%)

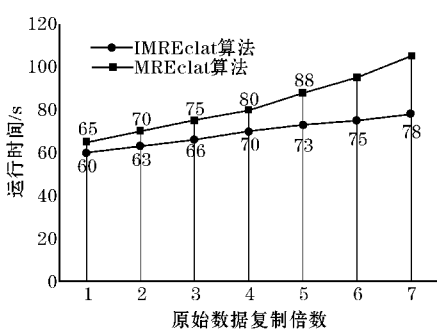

图 3　mushroom 数据集下性能对比图(min_sup=60%)

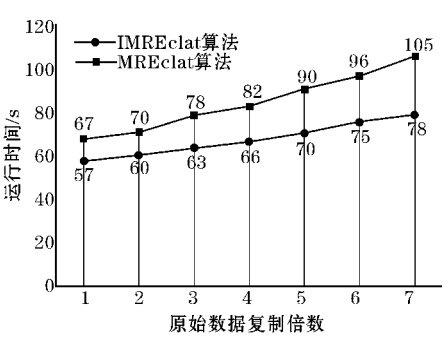

图 4　chess 数据集下性能对比图(min_sup=70%)

4　结束语

通过对 Eclat 算法的研究和分析,将传统的串行 Eclat 算法生成的频繁 K 项集进行预剪枝,以提高算法效率,并将改进后的算法利用 MapReduce 并行编程模型实现。在实现 IMREclat 算法的过程中,使用垂直数据集并用二进制形式存储事务集以简化交集运算,采用频繁 2 项集作为算法的输入以降低计算时间,利用等价类及其权重值来划分频繁 2 项集以达到负载均衡。实验证明,IMREclat 算法具备较好的可扩展性,并且运行效率也高于 MREclat 算法。

参考文献:

[1] Agrawal R,ImielIński T,Swami A. Mining association rules between sets of items in large databases[C]. proceedings of the Acmsigmod record, F, 1993:207-216.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]. proceedings of the Proc 20th intconf very large data bases, VLDB, F, 1994:481-496.

[3] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]. proceedings of the ACM sigmod record, F, 2000:1-12.

[4] 何海. 基于 Spark 的关联规则算法研究与实现[D]. 北京:北京邮电大学,2017:1-59.

[5] Zaki M J, Parthasarathy S, Ogihara M, et al. New algorithms for fast discovery of association rules[C]. proceedings of the KDD, F, 1997:1-25.

[6] 周国军,龚榆桐. 基于 MapReduce 和矩阵的频繁项集挖掘算法[J]. 微电子学与计算机,2016,33(5):119-123.

[7] Feng Xingjie, Pan Xuan. Parallel Eclat algorithm based on Spark [J]. Journal of Computer Applications, 2019, 1(1):21-30.

[8] 李伟卫,赵航,张阳,等. 基于 MapReduce 的海量数据挖掘技术研究[J]. 计算机工程与应用,2013,49(20):112-117.

[9] Moens S, Aksehirli E, Goethals B. Frequent Itemset Mining for Big Data[C]. proceedings of the IEEE International Conference on Big Data, F, 2013.

[10] Zhang Z, Ji G, Tang M. Mreclat: an algorithm for parallel mining frequent itemsets[C]. proceedings of the Advanced Cloud and Big Data (CBD), 2013 International Conference on, F, 2013: 117-180.

[11] Keerthi K, Sarltha S J. ECLAT: Frequent itemset using MapReduce[C]. proceedings of the 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017.

[12] 徐卫,李晓粉,刘端阳. 基于命题逻辑的关联规则挖掘算法 L-Eclat[J]. 计算机科学,2017,44(12):211-215.

[13] 崔馨月,孙静宇. 改进的 Eclat 算法研究与应用[J]. 计算机工程与设计,2018,39(4):1059-1063.

[14] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[M]. ACM, 2008: 129-135.

[15] Farzanyar Z, Cercone N. Efficient mining of frequent itemsets in social network data based on MapReduce framework[C]. proceedings of the Ieee/acm International Conference on Advances in Social Networks Analysis and Mining, F, 2013: 1183-1188.

[16] Zaki J. Scalable algorithms for association mining[J]. IEEE transactions on knowledge and data engineering, 2000, 12(3): 372-390.

Improved Eclat Algorithm based on MapReduce

XIANG Chunmei, CHEN Chao

(College of Communication Engineering, Chengdu University of Information Technology, chengdu 610255, China)

Abstract: The mining of association rules has always been an important task of data mining. However, with the advent of the era of big data, the data scale has grown exponentially. The traditional serial mining algorithms have faced problems such as the insufficient of memory and computing resources. Regarding the issue above, the IMREclat algorithm is proposed, which is an improved Eclat algorithm based on the MapReduce parallel programming model. The IMREclat algorithm uses two MapReduce tasks, which are mainly divided into three phases: Firstly, the transaction database is divided equally, and the frequent 2-itemsets are drilled in parallel. Secondly, the frequent 2-itemsets are converted into a vertical data format, and the binary storage transaction list is used to group by the equivalence class and its weight value. Finally, the grouped data is used as input, and all frequent item sets are mined in parallel by using the improved Eclat algorithm with pre-pruning properties. The experiments show that the IMREclat algorithm outperforms the existing MREclat algorithm in running time and has good expansion performance.

Keywords: data mining; association rules; frequent itemsets; MapReduce model; Eclat algorithm