doi: 10.12052/gdutxb.150120

IFAMR：一种基于MapReduce的高效频繁项挖掘算法

刘祥佳，程良伦
（广东工业大学 计算机学院，广东 广州 510006）

摘要: 针对单机无法高效地利用现有的并行计算框架来加快对海量数据进行频繁项挖掘的问题，根据Apriori算法基本原理，结合MapReduce并行计算模型的优势，在FAMR算法的基础上提出了一种改进的高效频繁项挖掘算法IFAMR. 该算法首先采用AprioriTID算法来对原始数据进行预处理，删除所有的低频1-项集，然后计算出每次处理集(L)和最小支持度(N)的长度来确定Map操作结束后的最大合并候选集合. IFAMR算法减少了Map函数中的低频项集的生成，通过与已有的算法进行实验对比表明，该算法有效地减少了内存占用，提高了算法的挖掘效率.

关键词: 频繁项集挖掘； MapReduce； FAMR； Apriori； Hadoop
中图分类号: TP311 **文献标志码:** A **文章编号:** 1007–7162(2017)02–0086–06

IFAMR: An Efficent Frequent Itemset Mining Algorithm Based on MapReduce

Liu Xiang-jia, Cheng Liang-lun
(School of Computers, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: Considering that single host in the existing parallel computing framework is not efficient in accelerating massive data mining frequent item, an improved efficient algorithm for mining frequent item IFAMR is proposed, combining the advantages of the MapReduce parallel computing model according to the basic principle of Apriori algorithm and based on the algorithm FAMR. This algorithm first uses AprioriTID algorithm to preprocess the raw data, deleting all the low-frequency 1-itemsets, and then calculate the length of each transaction set (L) and minimum support (N) to determine the maximum merger candidate Map sets of operations. IFAMR algorithm reduces the Map function to generate a low-frequency item set, and the algorithms are proved by experimental comparison to have greatly reduced memory footprint and effectively improved the efficiency of the mining process.

Key words: FIM (frequent itemset mining); MapReduce; FAMR (frequent itemset mining algorithm based on MapReduce); Apriori; Hadoop

随着数据挖掘算法的广泛应用，数据挖掘的数据流越来越大，数据种类越来越复杂，快速地从海量数据流中挖掘出有价值的信息变得越来越重要. 当单机变成了算法效率的瓶颈时，很多学者提出了利用网格计算，云计算以及Hadoop和MapReduce并行框架等并行计算模型的数据挖掘算法[1-5]. 频繁项集挖掘算法是一种普遍使用的数据挖掘算法，比如：基于MapReduce框架的一些Aprioir算法，其中比较常见的有One-Phase算法（基于一次MapReduce过程的Apriori算法）[6]，K-phase算法（基于K次MapReduce过程的Apriori算法）[7]以及FAMR算法[8].

One-phase算法由Li和Zhang在2011提出，该算法利用MapReduce并行框架来进行复杂计算，提高挖掘速度. 通过一次MapReduce操作，在Map阶段会产生所有的候选集，候选集的形式是以<Key，Value>来表示，然后在Reduce阶段根据不同的key值来对候选

收稿日期： 2015-11-14
基金项目： 国家基金广东省联合基金重点项目(U2012A002D01)；国家自然科学基金青年科学基金资助项目(61502110)
作者简介： 刘祥佳(1989–)，男，硕士研究生，主要研究方向为大数据处理、数据挖掘.

项集进行分类，通过扫描高频项集来计算出最小支持度来对候选集进行计数，改进传统的Apriori算法. 由于必须在Map阶段产生所有的候选集，这将会占用大量的内存，从而降低了算法的挖掘效率[9].

K-Phase算法是由Ning等学者在2013年提出的一种基于Hadoop平台的新的Apriori算法[10]，可以有效地解决一次MapReduce产生的内存消耗过大的问题. 多次MapReduce处理类似于传统的Apriori算法操作过程，经过多次MapReduce处理才产生所有的频繁项集，这样就可以大大地减少内存的占用，但是每次Map和Reduce操作之间的连接都要耗费一定的时间，这样也会降低整个算法的执行效率.

FAMR算法是Huang在2013年提出的一种基于MapReduce框架的频繁项集挖掘算法[11]，该算法的主要思想是利用AprioriTID算法[12]来对原始数据进行预处理，生成1-项高频候选集，删除低频1-项候选集，再将TID格式数据转换回原来的数据格式，压缩原始数据. 然后经过一次MapReduce过程来产生所有的(2~n)-项候选集. 由于只有一次MapReduce过程，所以连接代价减少，且Map过程减少了中间候选集的生成，从而降低了内存的占用，执行效率也就大大提升. 但是，一次Map操作同时产生所有的(2~n)-候选集，Map阶段将占用内存，从而影响算法的执行效率.

针对以上算法的不足之处，本文提出了一种改进的高效频繁项集挖掘算法IFAMR，该算法基于FAMR算法，首先采用AprioriTID算法来删除所有的低频1-项集候选集，然后根据每次计算出的处理集(L)和最小支持度(N)的长度来决定Map任务中的最大合并候选集数，这样就可以有效地减少Map阶段低频项集的生成，解决内存消耗过大和连接处理的开销过大问题，从而提高算法的执行效率.

1　问题描述

在MapReduce框架中，从程序设计的角度上来说，过程是：输入>Map>分开>排序>组合>Reduce>输出[13]. 从第一次Map操作到最终结果输出，复制阶段主要是将Map的结果复制当做Reduce阶段的输入，直到所有的Map操作完成，复制阶段结束.

为了预估在输入不同的数据量情况下，整个Mappers和Reducers阶段的连接时间开销，部署3个节点，然后将数据分别分割成6等份和3等份[14]. 采用FAMR算法来测试处理的数据量从D100K到D2000K（数据量从10 MB增大到70 MB）时Mappers和Reducers过程中的复制阶段所占算法执行时间的比例，结果如图1所示. 从图1中可以得到在运行过程中，复制时间所占整个过程的最小比例为48.7%，最大达到62.7%. 实验结果表明，复制阶段的运行时间在算法的过程中尤为重要.

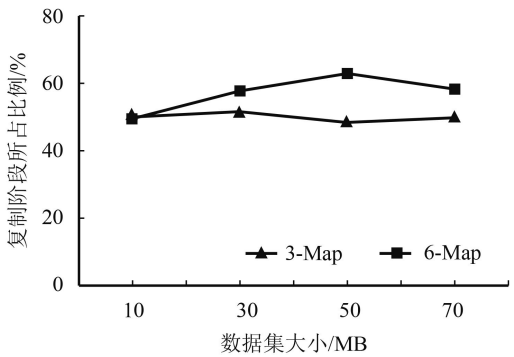

图 1　在不同的数据量下的复制阶段时间所占百分比

Fig.1　Communication overhead of all copy stages under different input data sizes

影响复制阶段执行效率的原因主要为：(1) 数据复制是远程复制. (2) 当使用HTTP协议进行数据传输时，大量的数据传输会产生网络延迟. (3) 因为Map阶段产生所有的候选项集，内存消耗会减慢数据的传输速度. 所以，为了减缓数据的传输压力，必须减少网络传输的时间以及Map阶段的数据吞吐量. 这样不仅可以减少Map阶段的内存占用，而且可以减少Reduce阶段需要计算的数据的传输量，降低网络连接的压力，从而提高整个算法的执行效率.

表1中的数据集是采用FAMR算法对原始数据经过TID格式转换，删除1-项低频项集后得到的新的数据信息. 如果最小支持度设置成0.3，那么高频项集至少有2个子项集合. 当平均交易集长度是3时，所有集合的长度是不同的. 因此，每次处理得到的候选项集也是不同的. 如表1所示：TransID=4时，可以合并成项集$\{A,C,D,E,F\}$，但是$\{A,C,D,E,F\}$明显不是一个高频项集. 因此FAMR算法过程中仍会产生大量的低频候选项子集，影响算法效率.

本文提出的IFAMR算法通过将TID格式转换成原始数据格式，计算交易集的长度和最小支持度来确定候选子集的最大长度，可以有效地减少Map阶

表 1　数据信息以及交易集的长度

Tab.1　New data record and length of transaction record

TransID	Itemsets	Length
1	$\{B, C\}$	2
2	$\{A, B, C\}$	3
3	$\{A, C, D\}$	3
4	$\{A, C, D, E, F\}$	5

段低频项集的产生.

如表1所示，每次操作的子集合的长度都是可以通过计算来确定的，假定最小支持度是0.4，对于每个候选集，如果存在至少2个交易集的长度会大于或者等于候选集的长度，那么候选集就是高频的. 否则，候选集便是低频的. 分析表1可得，候选集的最大长度是5，交易集中能够形成5-项候选集中肯定包含{4}，形成4-项候选集中也包含{4}，形成3-项候选集中包含{2,3,4}，形成2-项候选集中包含{1,2,3,4}.

因为3-项候选集交易集可以形成大于等于2的3-项子集和2-项候选集，但是形成大于3-项候选集的交易集的数量却小于2，所以得出：长度大于3-项的候选集都是低频项子集. 如表2所示，IFAMR算法使用Map/Reduce框架生成候选集时，通过计算出候选集的最大长度是3，因此仅仅只需要生成2-项子集和3-候选集. 而没必要去生成4-子集和5-候选集，对比FAMR算法减少了Map阶段4-候选集和5-候选集的生成.

表 2 交易集的候选集

Tab.2 Candidate itemset of transaction record

Trans ID	Candidate Itemsets
1	{*B*}{*C*}{*B*, *C*}
2	{*A*}{*B*}{*C*}{*A*, *B*}{*A*, *C*}{*B*, *C*}, {*A*, *B*, *C*}
3	{*A*}{*D*}{*C*}{*A*, *D*}{*A*, *C*}{*C*, *D*}{*A*, *C*, *D*}
4	{*A*}{*C*}{*D*}{*E*}{*F*}{*A*, *C*}{*A*, *D*}{*A*, *E*}{*A*, *F*} {*C*, *D*}{*C*, *E*}{*C*, *F*}{*D*, *E*}{*D*, *F*}{*E*, *F*}{*A*, *C*, *D*} {*A*, *C*, *E*}{*A*, *C*, *F*}{*A*, *D*, *E*}{*A*, *D*, *F*}{*A*, *E*, *F*}{*C*, *D*, *E*} {*C*, *D*, *F*}{*D*, *E*, *F*}{*C*, *E*, *F*}

2 IFAMR算法

IFAMR算法的关键就是计算出候选集的最大长度（即最大候选集的长度）. 所以，定义*D*为交易的数据集合，*H*是处理的次数. 每次操作的长度*Ti* (1≤*i*≤*H*)，交易集*D*中的最大操作集为*N*以及最小支持度为*S*, *No* (*i*)表示操作集长度等于*i*的所有的操作集数量. 因此，最大候选集*M*的长度应该同时满足：*No* (*M*)≥*H***S*和*No* (*M*+1)<*H***S*. 因为，*M*是候选集的可能最大长度，所以只需要在IFAMR算法的Map阶段形成所有的2-*M*项候选子集，则会减少Map阶段*M*+1-*N*候选集的形成.

IFAMR算法总体设计框架如下图2所示.

表3~表7给出了IFAMR算法执行步骤的伪代码. 首先，实现对原始交易数据集进行预处理，如表3所示，使用AprioriTID算法将原始数据转换成TID格式来找到所有的低频1-项候选子集，然后将1-项低频候选集从TID中删除后，再将新的TID格式中数据转换

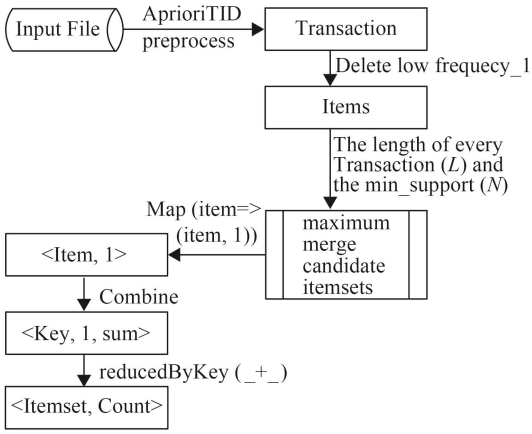

图 2 IFAMR算法总体框架

Fig.2 Algorithm architecture graph

成原始数据格式数据，生成新的数据信息.

根据操作集的长度和最小支持度计算出每次处理数据集的长度，从而确定候选集*M*的最大长度，伪代码如下表4所示.

执行Map操作的伪代码如表5所示. Map阶段形成的所有的候选集长度小于等于*M*. 根据不同的键值，将Reduce阶段需要的候选集进行分类，以及根据相同的键值来对候选集进行计数，过滤掉高频候选集，最后将输出的数据存储在HDFS中.

由于Map-Reduce阶段的复制操作时间开销太大，为了减少Map-Reduce阶段的数据传输的时间开

表 3 删除1-项低频子集

Tab.3 Delete the first-order low-frequency itemset

```
First Task
Input : D=Database format，min=minimum support count
Output : T=Transforming the TID table into a Database format
Scan D and structure a TID table
For each item in TID table
    If (transaction. size<min)
    Delete the item;
    Output (T);
End; End;
```

表 4 最大的合并项候选集*M*

Tab.4 Largest merged orderd candidate itemset *M*

```
Second Task
Input: T=Database format，min=minimum support count
Output:M=the max merged ordered itemsets size
Scan D and compute everytransaction's length
For each orderded itemset
    For each transaction in T
        If (order.size<=transaction's length)
        Count++;
        If (Count<minmum support count)
        Delete the ordered set;
End; End;
M=Max (order.size)
OutPut (M);
End;
```

表 5　执行Map操作

Tab.5　Executing Map task

```
MapTask
Input: Si=split I, M
OutPut:<key，l>，key is all of candidate itemsets
C=candidate itemset，the itemset's size<=M
For each transaction i in I
Map (key, value (i))
    For each item C in i
        Output (C，l);
End; End;
```

表 6　执行Combine操作

Tab.6　Executing Combine task

```
Combine Task
Input:<Key1，value1>，keyl is one of candidate itemsets
Output:<key1，value2>
Reduce (key1，value1)
    Sum=0;
    For each value1 in key1 list
        Sum+=value1;
    End;
Output (key1, sum);
End;
```

表 7　Reduce函数处理

Tab.7　Executing Reduce task

```
Reduce Task
Input: <Key1，value1>，key1 is one of candidate itemset，minimum support
count
Output: <key1, value2>
Reduce (Key1，value1)
    Sum=0;
    For each value 1 in key1 list
        Sum+=value1;
End;
    If (sum>=minimum support count)
        Output (key1，sum);
End; End;
```

销，算法采用了combine操作来计算Map的输出量. 从而减少了Map-Reduce之间的传输代价，但是combine函数没有包括高频项集. combine函数实现伪代码如表6所示.

Reduce函数的输入数据全部是来自于各个主机的Map函数的输出. Reduce阶段根据相同的键值来将候选子集进行分类和计算，最后通过最小支持度进行压缩得到高频项集. Reduce函数的伪代码实现如表7所示.

3　仿真分析

为了验证IFAMR算法的执行效率，将其与其他3个算法进行了实验仿真对比. 实验采用IBM数据生成器来获取实验数据，利用Handoop平台来做分布式计算. 根据不同的数据量，最小支持度以及Map阶段来测试算法的执行时间，并对比了每个算法的内存消耗情况.

3.1　实验配置

实验运行在Hadoop环境中，部署3个节点来做分布式并行计算. 详细的硬件和软件环境为Inter (R) Xeno (R) CPU X3440@2.53 GHz*1，Intel (R) Xeno (R) CPU X3430@2.40 GHz*2双核CPU，12 G DDR3 内存，CentOS relaese5.6操作系统，Java/jdk 1.6.0_24编译器，Hadoop 1.0.0集群.

实验数据中T代表数据处理的平均时间，I代表频繁子集的最大平均长度，D表示交易数据集的数量，N表示子项数量. Hadoop实验环境由3个节点和默认大小是32 MB的区块组成，Map的任务是自动形成的. 由于One-phase算法执行时间最长，所以，这里只测试IFAMR算法和K-Phase算法、FAMR算法在不同最小支持度以及Map任务数量条件下的执行时间.

3.2　对比实验

1) 当最小支持度是0.1%时，测试了K-Phase算法，FAMR算法和IFAMR算法在T5I4D100KN100K、T514D400KN100K、T5I4D800KN100K、T5I4D2000KN100K以及T5I4D5000KN100K数据集上的执行时间.

2) 当数据集是T10I4D400KN100K，Map的任务数量是3（数据在HDFS上平均分成3等份），支持度分别是0.05%、0.15%、0.25%时，测试了K-Phase算法，FAMR算法和IFAMR算法的执行时间.

3) 当数据集是T10U4D400KN100K，最小支持度是0.1%，Map任务数是5、3、1（数据在HDFS上平分成5、3、1等份）时，测试了FAMR算法和IFAMR算法的执行时间.

4) 当最小支持度是0.1%，Map任务数分别是6、3时，在不同的处理数据量下，IFAMR算法整个Map和Reduce阶段下的所有复制阶段的时间开销占整个算法执行时间的百分比.

3.3　结果分析

如图3所示，数据集D100K~D5000K（数据量从10~90 MB）3种算法的执行时间随着任务量的增加而增大，然而，本文提出的IFAMR算法的执行时间总是低于另外2种算法的执行时间. K-Phase算法虽然节省了内存，但是其在MapReduce阶段产生所有的候选集，因此会增加算法的执行时间，连接的开销也会变大，从而影响整个算法的执行效率. FAMR算法使用TID格式来寻找1-项频繁项集，减少了Map阶段的候选集的形成，虽然减少了执行时间，但是它却在Map阶段产生所有的(2~n)-项候选集.

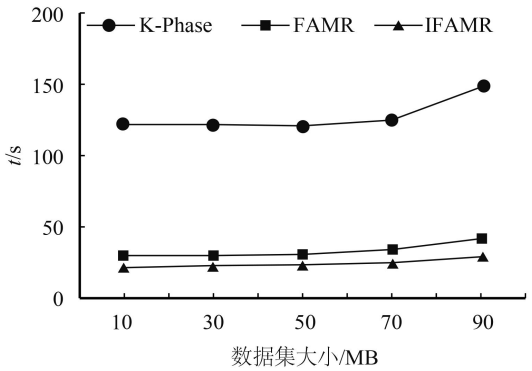

图3 3个算法在五类不同数据集上的执行时间

Fig.3 Execution time of the three kinds of algorithms on five kinds of databases

内存直接影响算法的执行效率. 本文算法通过减少在Map阶段结束时无效候选子集的产生，节省了内存，只经过一次MapReduce过程，这样执行时间和连接代价就会减少，进而提高整个算法的执行速度.

如图4所示，当最小支持度从0.05%提高到0.25%时，3种算法的执行时间都会减少，其中IFAMR算法的执行时间最少. 因为随着支持度的变大，高频项子集的数量也会减少. 对于K-Phase算法，虽然MapReduce的每个阶段只产生很少的候选子集，但是连接开销却很大. 而对于FAMR算法，通过TID格式将低频1-项候选集删除，可以明显减少Map阶段候选集生成的数量. 对于IFAMR算法，由于支持度的提高，最长候选集可能会从14倍减少到5倍的量级，同样减少了候选集的生成，甚至，当支持度为0.25%时，FAMR算法的执行时间和IFAMR算法由于支持度的变大执行时间相等. 在使用TID格式来预处理原始数据后，转换后的数据可以忽略不计. 当支持度提高到一定程度时，FAMR和IFAMR算法的最长候选子集相近，这

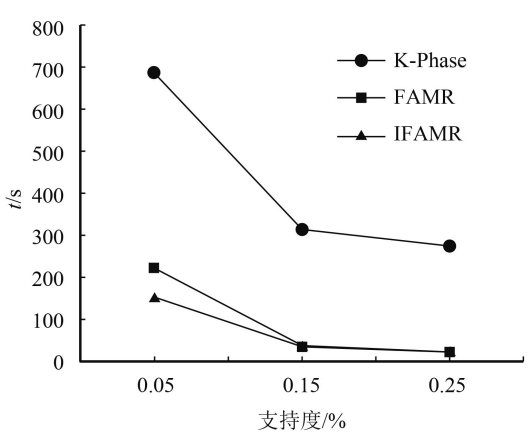

图4 3种算法在3种支持度下的执行时间对比

Fig.4 Execution time of the three algorithms on three kinds of support

两个算法的执行时间也大约相同.

如图5所示，基于同样的数据量，当Map任务数量级从1提高到5时，IFAMR算法的执行效率更高. 当数据量变大时，算法需要执行更多的Map任务，这样就可以充分地利用并行框架的计算优势来减少算法复杂计算的执行时间，从而提高算法的执行效率. 然而，数据量分成部分太多时，由于Map任务会提高JobTracker的任务调度，这样也会增加任务的执行时间. 并且由于MapReduce框架是基于网络协议比如：Tcp/IP，那么就会有RPC，以及传统HTTP的重连，导致整个框架的连接开销增大，影响算法的整个执行效率. 所以，在同样的硬件条件下，数据的分割（Map阶段的任务量大小）也会影响算法的计算效率.

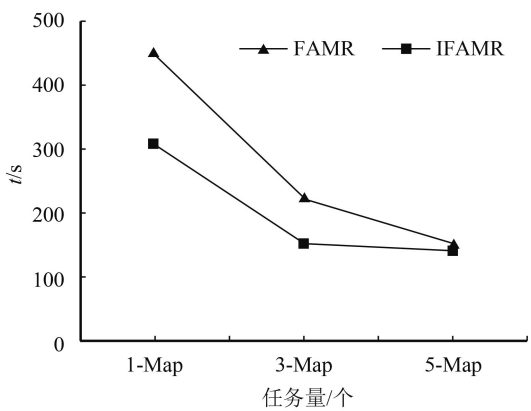

图5 不同的Map任务下的2种算法的执行时间对比

Fig.5 Execution time of the two algorithms on different number of Map tasks

如图6所示，当Map任务的数量是3时，数据集从D100K~D2000K（数据量从10~70 MB），最小的必要运行时间百分比是45.5%，最大是46.8%，但是当Map任务数增加到6时，最小的必要运行时间百分比是49.4%，最大为63.9%，所以，复制阶段的开销是影响整个算法执行效率的关键因素. 但是，算法在Map任务数6时的执行时间却要少于Map任务数为3时，复制阶段的连接开销直接影响了算法的执行效率. 这是因为当Map任务数是6时，Map-Reduce之间就需要连接6次，这代表越多的Map任务执行，复制阶段连接的次数就越多，所以连接代价也就越大.

通过以上实验仿真结果得出IFAMR算法根据交易集(L)来确定最长的候选子集，减少了Map阶段结束时候选集生成的数量，降低整个算法的执行时间和内存消耗. 同时，在Map结束时的数据量可以通过计算预估并加以充分利用，通过并行计算来避免在

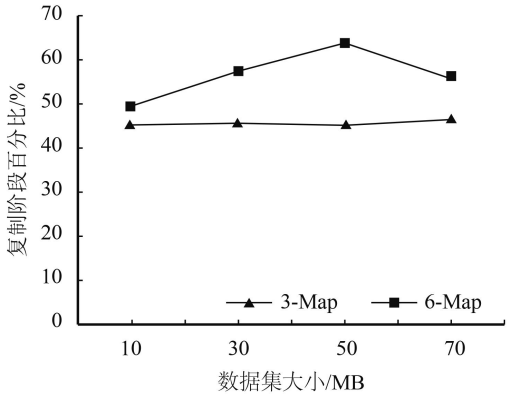

图 6　不同数据量时Map任务和Redcing执行时的连接开销

Fig.6　Communication overhead in Map tasks and Reducing execution time with different input data size

Map任务阶段数据分区太多造成的连接延迟，从而提高了整个算法的执行效率.

4　结论

为了提高单机的挖掘效率，本文提出了一种基于MapReduce框架的高效频繁项模式挖掘算法：IFAMR. 通过与已存的一些算法相比，提高了内存的使用. 与K-Phase算法相比，当处理的数据级大于20 MB时算法执行时间可以减少220 s以上，与FAMR算法相比，算法执行时间可以缩短大约60 s. IFAMR算法通过减少Map阶段结束时的候选子集数量，来节省Map阶段转换到Reduce阶段的候选子集所占用的内存，同时减少转换时间，进而提高整个算法的执行效率. 并且在Map阶段结束时可以产生最大的频繁项候选子集，充分利用MapReduce的并行计算框架来进一步减少连接处理开销.

参考文献：

[1] 李文俊. 基于MapReduce的Skyline查询算法研究[D]. 长沙: 湖南大学计算机学院, 2014: 22-45.

[2] 王乐. 数据流模式挖掘算法及应用研究[D]. 大连, 大连理工大学计算机学院, 2013: 33-50.

[3] 张启徽. 关联规则挖掘中查找频繁项集的改进算法[J]. 统计与决策, 2015, 23(4): 32-35.
ZHANG Q H. Lookup in association rules mining algorithm of frequent itemset[J].Journal of Statistics and Decesion, 2015, 23(4): 32-35.

[4] CHEN X S, ZHANG S, TONG T. FP-Growth algorithm based on boolean matrix and MapReduce[J]. Journal of South China University of Technology, 2014, 42(1): 135-141.

[5] 陈平, 王利钢, 李博涵. 基于项约束的关联规则挖掘研究综述[J]. 制造业自动化, 2014, 19(16): 46-53.
CHEN P, WANG L G, LI B H. Based on constrained association rules mining research were reviewed[J]. Journal of Manufacturing Automation 2014, 19(16): 46-53.

[6] 张巍, 刘峰, 滕少华. 改进的PrefixSpan算法及其在序列模式挖掘中的应用[J]. 广东工业大学学报, 2013, 30(4): 49-54.
ZHANG W, LIU F, TENG S H. Improved prefixspan algorithm and its application in sequential pattern mining[J].Journal of Guangdong University of Technology, 2013, 30(4): 49-54.

[7] 寇香霞, 任永功, 宋奎勇. 一种基于滑动窗口的数据流频繁项集挖掘算法[J]. 计算机应用与软件, 2013, 30(1): 143-146.
KOU X X, REN Y G, SONG K Y. A sliding window based on the data flow algorithm for mining frequent itemset[J]. Journal of Computer Applications and Software, 2013, 30(1): 143-146.

[8] LI N, Z ENG L, HE Q, *et al*. Parallel implementation of Apriori algorithm based on MapReduce[C]//Proc of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing. Bellinghan, USA: SNPD, 2012: 236-24.

[9] YAHYA O, HEGAZY O, EZAT E. An efficient implementation of apriori algorithm based on hadoop-mapReduce model[C]//International Journal of Reviews in Computing. Washington D C, USA: IJRIC, 2012: 760-771

[10] HUANG Y S. An efficient frequent patterns mining algorithm based on mapReduce framework[C]//Proc of the Human Computer Interaction with Mobile Devices and Services (HCIMD).Lisbon: ACM, 2013: 260-269.

[11] 吴建章, 韩立新, 曾晓勤. 一种基于多核微机的闭频繁项集挖掘算法[J]. 计算机应用与软件, 2013, 30(3): 44-46.
WU J Z, HAN L X, ZENG X Q. A multi-core microcomputer based algorithm for mining frequent closed itemset[J]. Journal of Computer Applications and Software, 2013, 30(3): 44-46.

[12] LIN M, LEE P, HSUEH S. Apriori-based frequent itemset mining algorithms on MapReduce[C]//Proc of the 16th International Conference on Ubiquitous Information Management and Communication. Washington D C, USA: ICUIMC, 2012: 1029-1040.

[13] 杨鹏坤, 彭慧, 周晓锋. 改进的基于频繁模式树的最大频繁项集挖掘算法-FP-MFIA[J]. 计算机应用, 2015, 35(3): 775-778.
YANG P K, PENG H, ZHOU X F. Improved based on frequent pattern tree algorithm for mining maximum frequent itemset--FPMFIA[J]. Journal of Computer Applications, 2015, 35(3): 775-778.

[14] 郑海雁, 王远方, 熊政. 标签集约束近似频繁模式的并行挖掘[J].计算机工程与应用, 2015, 6(3): 1-10.
ZHENG H Y, WANG Y F, XIONG Z. Label set constraint approximation parallel mining frequent patterns[J].Journal of Computer Engineering and Applications, 2015, 6(3): 1-10.