改进的 FP-Growth 算法及其分布式并行实现

马月坤， 刘鹏飞， 张振友， 孙 燕， 丁铁凡
(华北理工大学 信息工程学院，河北 唐山 063000)

摘 要: 针对传统 FP-Growth 算法在大规模数据环境下挖掘效率低下的问题，提出了一种改进的 FP-Growth 算法. 该算法主要是通过基于频繁闭项集策略对完备模式树进行剪枝进而减小搜索空间规模，达到提高算法挖掘效率的目的. 并将改进后的 FP-Growth 算法的分治策略与分布式计算框架 Hadoop 的 MapReduce 编程模式有机结合，进一步提高了大数据环境下的挖掘效率. 实验证明，基于 Hadoop 的改进 FP-Growth 算法的效率较传统 FP-Growth 算法有所提高.

关键词: 分布式并行; 改进 FP-Growth 算法; 剪枝; MapReduce 编程模式
DOI: 10.15938/j.jhust.2016.02.004
中图分类号: TP311.1 **文献标志码:** A **文章编号:** 1007-2683(2016)02-0020-08

Improved FP-Growth Algorithm and Its Distributed Parallel Implementation

MA Yue-kun， *LIU Peng-fei*， *ZHANG Zhen-you*， *SUN Yan*， *DING Tie-fan*
(School of Information, North China University of Science and Technology, Tangshan 063000, China)

Abstract: The main problem solved in the research is that the traditional FP-Growth algorithm had a low efficiency and memory overflow under the big data environment. Aimed at the problem, this paper put forward a improved FP-Growth algorithm, which reduces the search space through complete pattern tree pruning by using frequent closed itemset to improve the mining efficiency of the algorithm. Then the FP-Growth algorithm is realized by using MapReduce a programming pattern of Hadoop. Thereby the mining efficiency is improved further. Experiments show that the efficiency of the improved FP-Growth algorithm implement by Hadoop is superior to traditional FP-Growth algorithm.

Keywords: distributed parallel; improved FP-Growth; pruning; MapReduce programming

0 引 言

R. Agrawal 等[1]于 1993 年提出了关联规则的概念，用于挖掘顾客交易数据库中的频繁模式. 关联规则挖掘的基本算法有 Apriori 算法和 FP-Growth 算法. R. Agrawal 提出了经典的 Apriori 算法来挖掘数据集中的频繁模式，从而挖掘出数据项集之间的关联规则[2-3]. 为了提高挖掘效率，有研究人员提出了基于散列的技术、事务压缩、抽样以及动态项集计数等改进算法[4]，但是因算法有反复扫描数据库和产生大量候选项集的缺点. 于是提出了 FP-

收稿日期: 2015-06-03
基金项目: 河北省科技支撑计划项目(15210110D); 唐山市科技支撑计划项目(14130233B).
作者简介: 马月坤(1976—)，女，博士，副教授;
刘鹏飞(1991—)，男，硕士研究生.
通信作者: 孙 燕(1990—)，女，硕士研究生，E-mail: sunyan19900930@126.com.

Growth 算法[5-6]，该算法的优势表现为挖掘全部频繁项集却不产生大量候选集. 随着大规模数据的产生，在存储和计算两个方面对单机环境下数据挖掘的处理能力提出了挑战，并行计算环境成为解决大数据处理问题的首选方案. 有研究者提出的基于多线程的并行算法[7]，虽然在很大程度上缓解了存储及计算的压力，但是内存资源的局限成为了算法扩展的瓶颈; 有研究人员在 MPI 并行环境上，使用进程间通信的方法来协调并行计算，其结果是并行计算效率较低、内存开销大并且不能够实现多节点的横向扩展[8-9]; 为了在某种程度上降低并行计算的通信消耗，邹翔等提出一种基于前缀树的并行算法[10]. 但该算法仍会受到内存消耗的限制. 针对海量数据的数据挖掘问题，本文提出了一种并行的改进 FP-Growth 算法.

1　传统的 FP-Growth 算法

经典的数据挖掘 Apriori 算法是关联规则模式挖掘的先驱，但它需要反复访问事务数据库，并且在对长模式挖掘时会产生大量的候选集. 针对 Apriori 算法存在的 I/O 频繁访问和存储空间大这两个主要问题，FP-Growth 算法通过利用减少全量扫描事务数据库的次数和不产生候选集的策略提高了关联规则挖掘效率. FP-Growth 算法中使用了一种称为频繁模式树的数据结构. FP-Tree 是一种特殊的前缀树，由频繁项头表和项前缀树构成[11-12].

假设 $I=\{I_1,I_2,\ldots,I_m\}$ 是项的集合. 给定一个订单数据库 ORDER，其中每个事务(order) T 是 I 的非空子集，即，每一个订单都与一个唯一的标识符 TID(Order ID) 对应，如表 1 所示.

表 1　事务数据表

TID	商品 ID 的列表
T100	I_1,I_2,I_5
T200	I_2,I_4
T300	I_2,I_3
T400	I_1,I_2,I_4
T500	I_1,I_3
T600	I_2,I_3
T700	I_1,I_3
T800	I_1,I_2,I_3,I_5
T900	I_1,I_2,I_3

FP-Growth 算法可以分为两步: 创建模式 FP-Tree 和挖掘 FP-Tree[13]，具体步骤如下:

第一次全量扫描数据库，统计导出 $k=1$ 项集以及对应的支持度计数，并且按照支持度计算倒排. 结果记为 L. 支持度设置为 2，则 $L=\{\{I_2:7\},\{I_1:6\},\{I_3:6\},\{I_4:2\},\{I_5:2\}\}$.

接下来，建立 FP-Tree，首先创建树的根节点，记为 root. 第二次全量扫描数据库，每条事务数据按照 L 中的次数排序，并且以 root 为根节点创建一条路径.

如此，遍历整个数据库，将每条订单数据进行排序，创建对应的路径，但是，如果创建路径时遇到相同的节点，该节点计数就增加 1. 为了形成的 FP-Tree 能够方便被遍历，创建一个项头表，表的字段为项 ID，支持度计数，节点链[14]. 所以，经过上述建立 FP-Tree 的过程，最终的 FP-Tree 结构如图 1 所示. 创建 FP-Tree 流程图如图 2 所示.

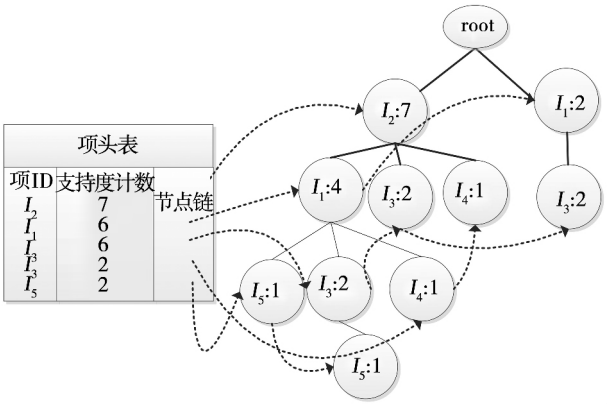

图 1　建立好的 FP-Tree

表 2　通过创建条件模式基挖掘 FP-Tree

项	条件模式基	条件 FP-Tree	生成的频繁模式
I_5	$\{I_2,I_1:1\}$, $\{I_2,I_1,I_3:1\}$	$<I_2:2,I_1:2>$	$\{I_2,I_5:2\},\{I_1,I_5:2\}$, $\{I_2,I_1,I_5:2\}$
I_4	$\{I_2,I_1:1\},\{I_2:1\}$	$<I_2:2>$	$\{I_2,I_4:2\}$
I_3	$\{I_2,I_1:2\}$, $\{I_2:2\},\{I_1:2\}$	$<I_2:4,I_1:2>$, $<I_1:2>$	$\{I_2,I_3:4\},\{I_1,I_3:4\}$, $\{I_2,I_1,I_3:2\}$
I_1	$\{I_2:4\}$	$<I_2:4>$	$\{I_2,I_1:4\}$

FP-Growth 算法最后一步(也是最重要的一步) 为挖掘 FP-Tree. 挖掘过程如下: 遍历 $F1$ 频繁项集，构建每个项的条件模式基(即子数据库) ，通过它的前缀路径来构建条件 FP-Tree，根据项头表自底向上递归挖掘，链接后缀模式与条件 FP-Tree 生成的频

繁模式为最终的频繁项集. 对建立的 FP-Tree 的挖掘过程总结发如表 2 所示.

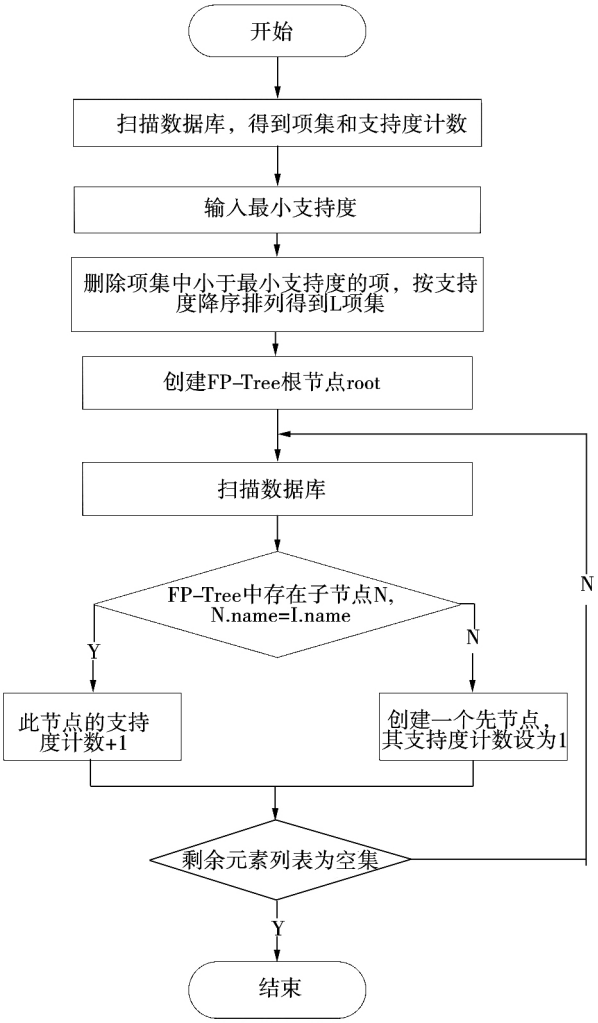

图 2 创建 FP-Tree 的流程图

FP-Growth 算法主要分为建立 FP-Tree 和挖掘 FP-Tree,从而在不产生候选集的条件下实现了频繁项集的挖掘. 同时,把数据压缩成树形结构,大大的减少了存储的空间. FP-Growth 算法有分治的策略,所以为分布式并行计算提供了依据[15-16].

2 改进的 FP-Growth 算法

为了进一步优化 FP-Growth 算法的性能,通过项合并策略对 FP-Growth 算法的 FP-Tree 进行剪枝.

项合并策略的理论依据: 如果包含频繁项集 A 的每个事务都包含项集 B,但不包含 B 的任何真超集,则 $A \cup B$ 形成一个闭频繁项集,并且不必再搜索包含 A 但不包含 B 的任何项集[17].

上述原则描述的是项合并的剪枝策略的理论依据,通过项合并策略能够大大的减小搜索空间的规模,提高挖掘频繁树的效率,也在很大程度上减少了频繁项集的数量.

改进的 FP-Growth 算法主要分为两步:

第 1 步: 通过扫描整个事务数据库,计算出 F 频繁项集,并且按项的频繁计数从大到小排序,记为 F-List. 再次全量查询事务数据库,遍历每条事务数据,根据 F-List 做一次升序遍历,然后创建节点名 root 为 FP 树的根节点,按照图 2 所示的建立 FP-Tree 的流程生成一颗完备的模式树.

第 2 步: 开始挖掘已建立的 FP-Tree,采用项合并的策略来减少挖掘 FP-Tree 时产生的条件模式基,达到剪枝的效果. 改进的挖掘 FP-Tree 的流程如图 3.

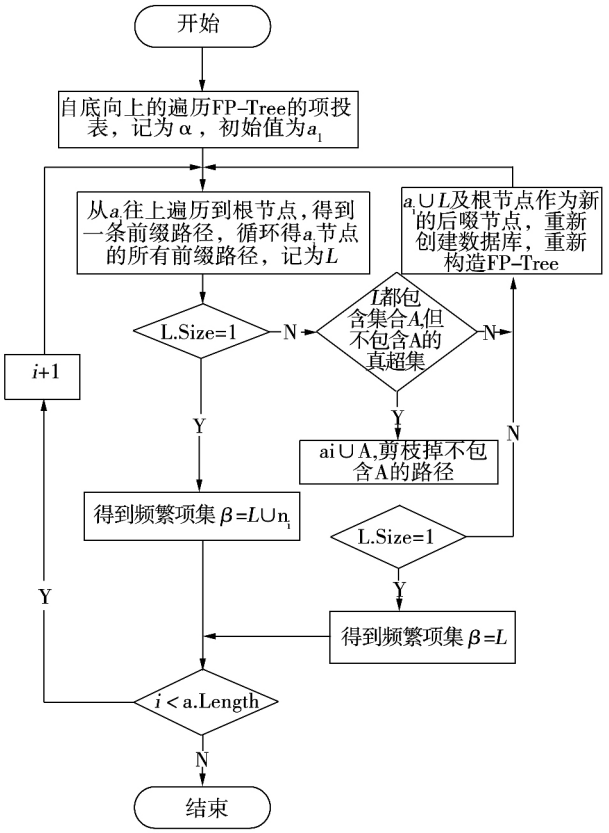

图 3 改进的挖掘 FP-Tree 的流程图

改进的挖掘 FP-Tree 的步骤如下:

第 1 步: 自底向上的方式遍历生成的 FP-Tree 的项头表,从而得到此节点的所有前缀路径,以此节点为后缀模式,得到所有包含此节点在 FP-Tree 的路径.

第 2 步: 如果上步得到单条路径,前缀路径的每个元素和后缀模式合并生成频繁项集. 否则,需找前缀路径中是否存在项合并中的情况. 若存在,剪枝后合并; 否则,转到第 3 步.

第 3 步: 以上述得到的路径中包含的所有后缀节点及根节点作为新的后缀节点,重新创建数据库,按照创建 FP-Tree 的流程重新创建一棵新的 FP-Tree.

第 4 步: 把第 3 步生成的树作为第一步输入的数据源,反复迭代,直到所有的项只存在一条路径.

为了更加清晰地描述此剪枝策略,对图 2 中建立的 FP-Tree 进行挖掘,图 2 的根节点 root 有两个子节点,I_2 和 I_1,分别对应两棵子树. 基于 FP-Tree 的频繁项集挖掘过程如下:

1) **根据图 2 中项头表的排列顺序,倒序遍历,则遍历顺序为 "I_5,I_4,I_3,I_1". I_5 出现在图 2 中的 FP-Tree 的两条分支中,这些分支形成的路径分别为 $<I_2,I_1,I_5:1>$ 和 $<I_2,I_1,I_3,I_5:1>$,因此,以 I_5 为后缀,它的两个前缀路径是 $<I_2,I_1:1>$ 和 $<I_2,I_1,I_3:1>$,这两个路径中都包含项集 $\{I_2,I_1\}$,但不包含项集 $\{I_2,I_1\}$ 的真超集,根据项合并,将项集 $\{I_5\}$ 和 $\{I_2,I_1\}$ 合并,生成的频繁项集为 $\{I_2,I_1,I_5:2\}$,剪掉项集 $\{I_3\}$. I_5 项剪枝结果如图 4 所示.**

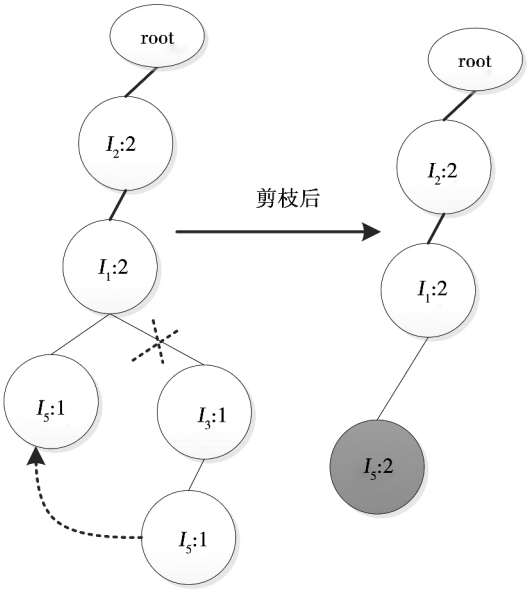

图 4　项 I_5 剪枝结果图

I_4 在图中有两条分支,路径分别为 $<I_2,I_1,I_4:1>$ 和 $<I_2,I_4:1>$,因此,I_4 的前缀路径有两条,分别为 $<I_2,I_1:1>$ 和 $<I_2:1>$,根据项合并,生成的频繁项集为 $\{I_2,I_4:2\}$.

I_3 在图中有 3 条分支,路径分别为 $<I_2,I_1,I_3:1>$、$<I_2,I_3:2>$ 和 $<I_1,I_3:2>$,I_3 的前缀路径分别为 $<I_2,I_1:1>$、$<I_2:2>$ 和 $<I_1:2>$,对于前两条路径中,都包含集合 $\{I_2\}$,根据项合并,将 $\{I_3\}$ 和 $\{I_2\}$ 合并,生成频繁项集 $\{I_2,I_3:3\}$,经过该剪枝后,发现 I_3 的前缀路径仍然有两条 $<I_2,I_3:3>$ 和 $<I_1,I_3:2>$,那么需要重建订单数据库. 此时新生成的订单数据库如表 3 所示.

I_1 在图中有两条分支,分别为 $<I_2,I_1:4>$ 和 $<I_1:2>$,因此 I_1 的前缀路径为 $<I_2:4>$,那么生成的频繁项集为 $\{I_2,I_1:4\}$.

表 3　新生成的订单数据库 T'

T'ID	商品 ID 的列表
T1001	I_1,I_2,I_5,root
T2001	I_2,I_4,root
T3001	I_2,I_3,root
T4001	I_2,I_4,root
T5001	I_1,I_3,root
T6001	I_2,I_3,root
T7001	I_1,I_3,root
T8001	I_1,I_2,I_5,root
T9001	I_2,I_3,root

2) **全量扫描数据库 T',统计导出 $k=1$ 项集以及对应的支持度计数,并且按照支持度计算倒排. 结果记为 L'. 支持度设置为 2,则 $L'=\{\{root,9\},\{I_2:7\},\{I_1:5\},\{I_3:4\},\{I_4:2\},\{I_5:2\}\}$,根据创建 FP-Tree 的流程图重新创建订单数据库 T' 的 FP-Tree,如图 5 所示.**

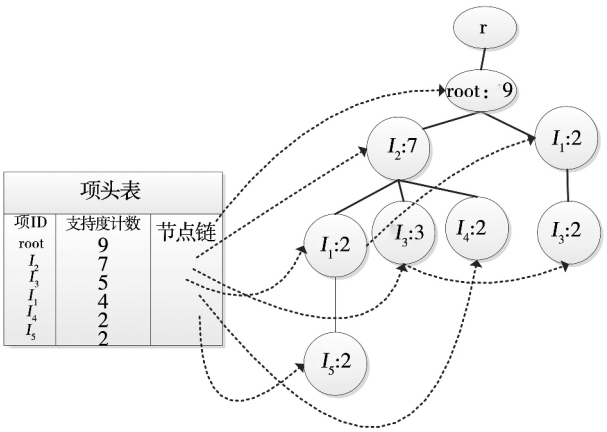

图 5　订单数据库 T' 生成的 FP-Tree

I_3 在图中有两条分支,路径分别为 $<root,I_2,I_3:2>$ 和 $<root,I_1,I_3:2>$,因此,I_3 的前缀路径为 $<root,I_2:2>$ 和 $<root,I_1:2>$,每个路径中都包含项集 $\{root\}$,根据项合并策略,$\{root\}$ 与 $\{I_3\}$ 进行合并,行成闭频繁项集 $\{root,I_3:4\}$,剪枝后生成的 FP-Tree 如图 6 所示.

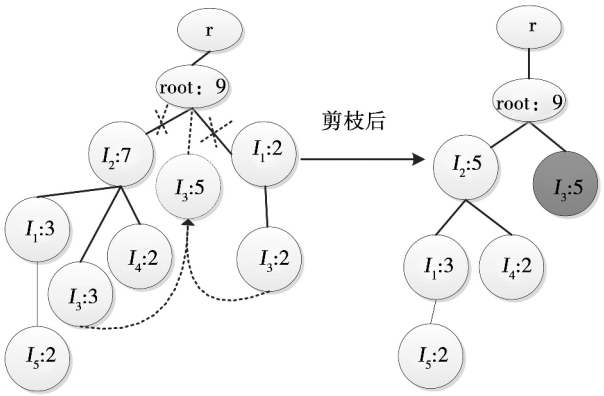

图 6 项 I_3 剪枝后结果图

通过上述的过程，整个 FP-Tree 挖掘的过程也就完毕．挖掘出来的频繁项集都是闭频繁项集．整个过程对搜索闭项集的搜索空间做了相应的优化，减少了树的路径，从而提高了算法的计算速度．

3 基于 Hadoop 的并行 FP-Growth 实现

本文提出来基于分布式计算框架 Hadoop 的并行计算的 FP-Growth 算法．利用 Hadoop 自身的各种处理大数据的优势，很好的解决了传统 FP-Growth 算法的问题．图 7 为分布式 FP-Growth 算法的流程图．

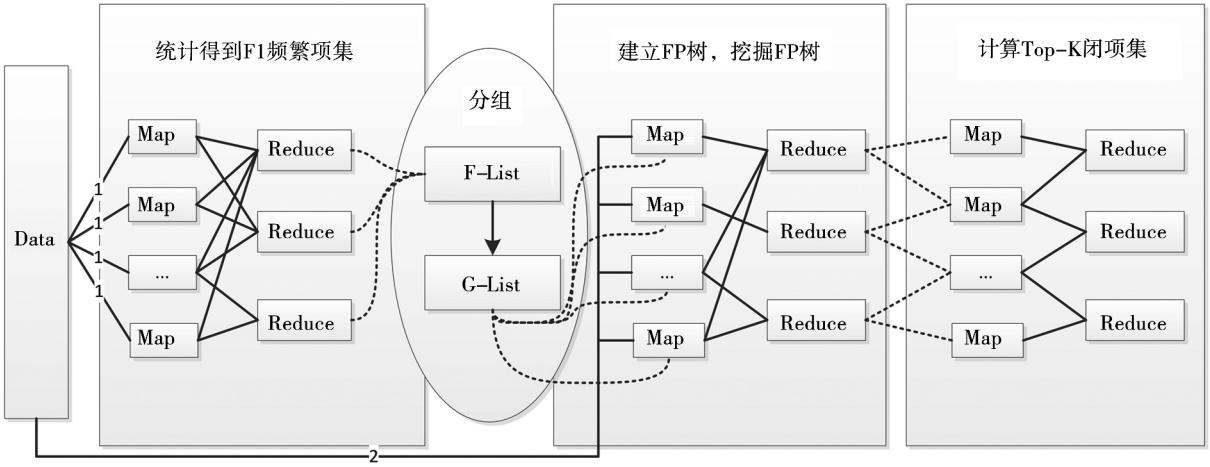

图 7 分布式 FP-Growth 算法的流程图

通过图 7 可知：分布式 FP-Growth 算法结合了 MapReduce 的编程思想[18-19]，利用 3 个串行的 MapReduce 任务来完成，其中，结合分布式缓存机制来存储 F-List 表能够提供访问效率，降低相应的 I/O 操作．下面就是分布式 FP-Growth 算法的主要步骤的详细描述：

1) 统计 F1 项频繁集

统计 F1 项频繁集是并行算法的第一步．此步就是统计每个项在数据库中出现的次数，并且过滤掉出现次数小于设置的最小支持度的项，再把剩余的项通过出现的次数进行降序排序，最终得到 F-List 集合．其中 F-List 集合中不仅存储了项的唯一标识，而且也储存了出现的频率，即项的计数．

统计 F1 项集的具体的处理步骤如下：

Map 过程：第一次全量扫描事务数据，输入的 key 为每行的偏移量，value 为每条事务数据．针对 value 的值进行分割后，输出的 key 为每个项，value 为每个项首次出现的计数为 1.

Combiner 过程：此 combiner 过程提高计算性能，在集群的每个节点上本地化的对每个项进行一次相加统计，输出的 key 依旧是项，value 变为本节点上项出现的次数，即计数．

Reducer 过程：收集 Map 阶段输出的中间结果，相加统计一下每个项在整个事务数据库中出现频率．输出的 key 为项的标识符，value 为数据中出现的频率．

图 8 为统计 F1 项集的数据处理流程图．

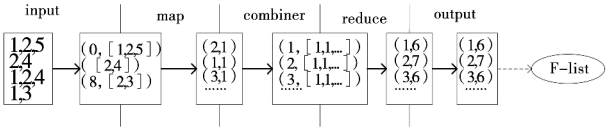

图 8 F1 项集的数据处理流程图

其实,统计 F1 项频繁集阶段得出的结果就是整个数据库的 1 项频繁集. 通过每个项的计数由高到低的排序后,就形成了 F-List 集合,并且把 F-List 集合存入 Hadoop 的分布式缓存,如此在整个集群中共享了此文件,方便了事务数据库分发数据,还减少了相应的 I/O 操作. 为了达到每组数据的平衡以及其独立性,利用 F-List 集合中 F1 项来分组.

2) 分布式并行 FP-Growth 算法实现

图 9 为并行 FP-Growth 算法的实现步骤,也是整个算法最主要的步骤.

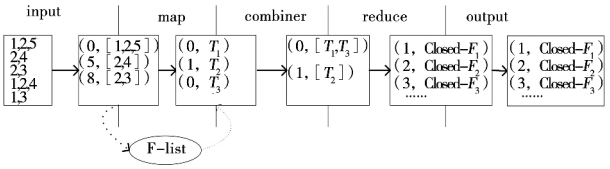

图 9　并行 FP-Growth 算法的过程图

通过图 9 可知,并行 FP-Growth 算法的具体的过程如下:

分组和 Map 过程: 在本阶段首先读取分布式缓存中 F-List 集合,并且存入 Map 中,并且对项对应一个编号映射,记为 i. 第二次全量地访问事务数据库,遍历每条事务数据中的项,通过 F-List 表中的 F1 频繁项集得到一个组标识符(记为变量 g),从而得到了 G-List 集合.

其中,分组的策略为: 首先得到每组中项的个数,记为 m. $m = \frac{|F|}{n}$,其中: m 为每组中项的个数; F 为 F-List 集合的模,集合中项的个数; n 为设定的组数. 如果不能整除,m 就加上 1.

遍历每条事务数据中项,依照 F-List 集合,先做一次排序,然后在映射表中找到项的映射数字,即 i. 这样,此项的组标识符计算公式: $g = \frac{i}{m}$. 其中: g 为此项将分入的组号; i 为项在 F-List 集合中的排序号; m 为每组中项的个数.

通过上述策略就找到了每个项对应的组,聚合每个组,从而得到 G-List 集合.

在此 Map 阶段输出的 key 值为 g,value 为整条数据的子项集(事务序列 P 的子序列). 这样分组不仅均衡了整个数据库的投影数据,同时保证了挖掘过程中数据的完备性.

Combiner 过程: 本地聚合每组中的事务数据,形成压缩的事务数据.

Reduce 过程: 此阶段是整个并行 FP-Growth 算法中具体对数据进行挖掘的一步. 就是在集群的节点上运行本地化的改进 FP-Growth 的算法,但数据已经不是整个事务数据库了,而是子数据库. 首先建立 FP-Tree,然后通过改进的挖掘算法得到频繁项集. 输出的 key 为组号,而 value 为挖掘出来的频繁项集.

3) 聚合

图 10 为分布式 FP-Growth 算法的最后一步,是挖掘 K 项最大的闭频繁项集的过程图.

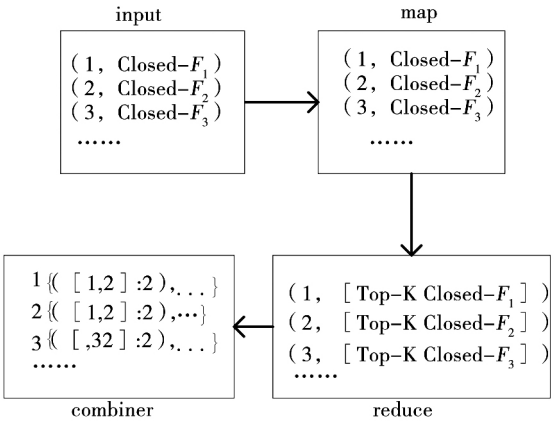

图 10　挖掘 K 项最大的闭频繁项集的过程图

挖掘 K 项最大的闭频繁项集的具体步骤如下:

Map 过程: 本过程输入的 value 为挖掘出来的频繁项集. 遍历频繁项集中的元素,输出的 key 为此元素,value 为此频繁项集.

Combiner 过程: 本地化对每个 item 的相应的频繁项集做一次排序并且选取频繁项集共同出现的频率最大的 K 项频繁项集.

Reduce 过程: 收集 combiner 过程产生的局部最大的 K 项频繁项集,再次做排序,选取整个数据库中计数最大的 K 项频繁闭项集.

至此,完成了整个基于分布式策略的 FP-Growth 算法. 其分布式策略实现的核心是 MapReduce. 实现过程清晰,步骤明细,但是各关键步骤是被封装的,如果需要对一些策略进行性能上的提升,则要通过利用对 Hadoop 中 MapReduce 计算框架的参数进行设定以实现性能调优的方法来完成.

4　实验过程与实验结果

实验 1: 为了验证改进的 FP-Growth 算法的性

能,利用 mushroom. dat 数据来做实验. mushroom. dat 数据来自于 Frequent Itemset Mining Data Repository [20]. 实验通过 FP-Growth 算法和改进的 FP-Growth 算法作比较. 在相同的最小支持度下挖掘同一份数据的速率来做衡定,其中结果的横坐标数值为支持度阈值,那么最小支持度为整个 mushroom. dat 数据中包含的事务数据条数乘以支持度阈值,即 min_support = transactions. number * threshold. 图11为实验1的结果图.

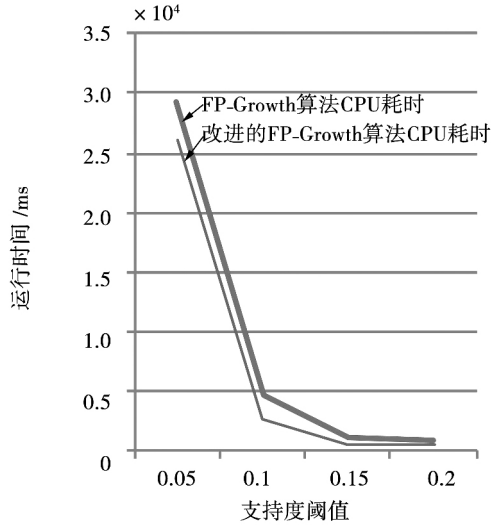

图11 挖掘速率比较图

从图11中可以看出,改进的 FP-Growth 算法在计算速度上有明显的提高,性能较好. 说明通过项合并的策略减少了搜索 FP-Tree 的迭代次数,实现了对搜索空间的剪枝. 阀值变大,相应的最小支持度计数也变大,从而得到的频繁项集的总量在减少,搜索的代价也随着降低,所以改进的 FP-Growth 算法和 FP-Growth 算法在挖掘速度上很接近. 从而可以得知,改进的 FP-Growth 算法的搜索空间规模远远小于传统 FP-Growth 算法,进而从计算资源的消耗上有所降低.

实验1的挖掘速度随着数据量的增加,挖掘速度急剧下降,到了一定程度就会内存溢出.

实验2:为了测试改进的 FP-Growth 算法在大规模数据处理能力方面的性能,进行了下一步实验(实验2). 实验2所采用的数据为电子商务订单数据,此实验数据的总条数为1 000 W条.

以下是实验2的过程描述.

1)为了反映在不同数据量情况下分布式算法的对应的计算速率和变化趋势,分别选取了10 W数量级中10 W和50 W,100 W数量级中100 W和500 W,1 000 W数量级的1 000 W,3 000 W,5 000 W,7 000 W和10 000 W作为测试数据对算法进行测试. 测试结果如图12所示.

2)为了展示在同样的数量级的数据中,挖掘最大的 K 项频繁项集花费的时间情况,选取了 $K=3$,5,8,10和12项频繁项集. 实验显示每一步的 CPU 计算时间的情况,使得结果更加的清晰. 实验结果如图13所示.

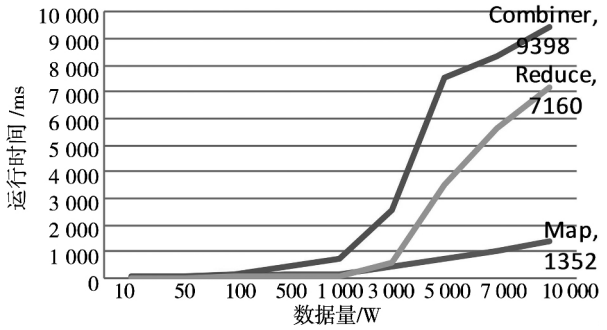

图12 数据量与计算速率的折线图

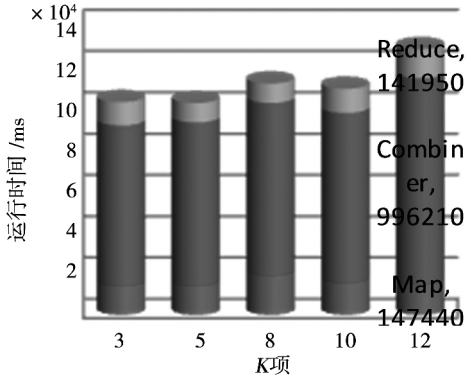

图13 K 项与计算速率的柱状图

通过图12可知:在不同的数量级下改进的 FP-Growth 算法计算的时间是不同的,其中 Map 过程在数据集的规模剧增的时候时间变化相对缓慢,其次是 Reduce 过程,最后是 Combiner 过程. 在本文的介绍过程中也提到了,Combiner 过程是整个分布式算法最关键的一步,也是最消耗时间的一步. 这步的计算任务是繁重的,但是在大规模数据下消耗的时间是可以接受的. 至少不会出现单机运行大规模数据出现内存溢出的情况.

通过图13可知:在挖掘不同最大 K 项频繁项集时花费的时间差距是不大的,各个步骤时间开销变

化率都比较缓慢. 说明分布式算法充分利用了集群的性能和内存空间,能够很好地完成大规模数据的频繁项集挖掘.

5 结　论

通过对分别针对经典数据集和大数据集的实验获得的数据进行分析,可知改进的 FP-Growth 算法在挖掘效率上比传统算法有明显的提高. 其主要原因在于挖掘 FP-Tree 的时候通过项合并策略实现剪枝,大大减少了树挖掘过程的迭代次数. 同时减少了内存空间的占用. 并将改进后的 FP-Growth 算法的分治策略与分布式计算框架 Hadoop 的 MapReduce 编程模式有机结合,进一步提高了大数据环境下的挖掘效率.

参 考 文 献:

[1] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining Association Rules between Sets of Items in Large Data Bases [C] // Proc of the 1993 ACM-SIGMOD International Conference on Management of Data (SIGMOD' 93). Washington, DC: ACM, 1993: 207 - 216.

[2] AGRAWAL R, SRIKANT R. Fast Algorithms for Mining Association Rules [C] // Proc of the 1994 International Conference on Very Large Data Bases (VLDB'94). Santiago, Chile: Conference Publications, 1994: 487 - 499.

[3] 文拯. 关联规则算法的研究 [D]. 长沙: 中南大学, 2009.

[4] 何月顺. 关联规则挖掘计数的研究及应用 [D]. 南京: 南京航空航天大学, 2010.

[5] HAN J, PEI J, YIN Y. Mining Frequent Patterns without Candidate Generation [C] // Proc of 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD'00). Dallas: Conference Publications, 2000: 1 - 12.

[6] 杨勇, 王伟. 一种基于 MapReduce 的并行 FP-growth 算法 [J]. 重庆邮电大学学报: 自然科学版, 2013, 25(5): 652 - 654.

[7] LI L, ZHANG Y. Optimization of Frequent Itemset Mining on Multiple-core Processor [C] //Proceedings of the 33th International Conference on Very Large Data Bases. Vien-na, Austria: VLDB Endowmen, 2007: 1275 - 1285.

[8] LAMINE M, NHIEN L, TAHAR M. Distributed Frequent Itemsets Mining in Heterogeneous Platforms [J]. Journal of Engineering, Computing and Architecture, 2007, 1(2): 1 - 12.

[9] MOHAMMAD E, OSMAR R. Parallel Leap: Large-Scale Maximal Pattern Mining in a Distributed Environment [C] //Proceedings of the 12th International Conference on Parallel and Distributed Systems, Minneapolis MN: Conference Publications, 2006: 135 - 142.

[10] 邹翔, 张巍, 刘洋, 等. 分布式序列模式发现算法的研究 [J]. 软件学报, 2005, 16(7): 1262 - 1269.

[11] 马丽生, 姚光顺, 杨传健. 基于改进 FP-Tree 的最大频繁项目集挖掘算法 [J]. 计算机应用, 2012, 32(2): 326 - 329.

[12] NAHAR J, IMAM T, TICKLE K S, et al. Association Rule Mining to Detect Factors Which Contribute to Heart Disease in Males and Females [J]. Expert Systems with Applications, 2013, 40(4): 1086 - 1093.

[13] LI H, WANG Y, ZHANG D, et al. PFP: Parallel FP-Growth for Query Recommendation [J]. ACM Recommender Systems, 2006(3): 43 - 51.

[14] 刘应东, 冷明伟, 陈晓云. 基于邻接矩阵的 FP-tree 构造算法 [J]. 计算机工程与应用, 2011, 47(7): 153 - 155.

[15] 晏杰, 亓文娟. 基于 Aprior& FP-Growth 算法的研究 [J]. 计算机系统应用, 2013, 22(5): 122 - 125.

[16] 祝孔涛, 李兴建, 王乐. 高效用项集挖掘算法 [J]. 计算机工程与设计, 2013, 34(12): 4220 - 4225.

[17] HAN Jiawei, KAMBER Micheline. 数据挖掘: 概念与技术 [M]. 3 版, 北京: 机械工业出版社, 2013.

[18] GREGORY Buehrer, SRINIVASAN Parthasarathy, SHIRISH Tatikonda, et al. Toward Terabyte Pattern Mining: Anachitecture-conscious Solution [J]. PPOPP, 2010(5): 103 - 111.

[19] 吕雪骥, 李龙澍. FP-Growth 算法 MapReduce 化研究 [J]. 计算机研究与发展, 2012(11): 211 - 242.

[20] Frequent Itemset Mining Dataset Repository [EB/OL]. [2012 - 10 - 21]. http://fimi.ua.ac.be/data/, 2012.

(编辑: 温泽宇)