计算机科学
COMPUTER SCIENCE

基于N-list和DiffNodeset结构的频繁项集并行挖掘算法

张阳, 王瑞, 吴贯锋, 刘弘毅

引用本文

张阳, 王瑞, 吴贯锋, 刘弘毅. 基于N-list和DiffNodeset结构的频繁项集并行挖掘算法[J]. 计算机科学, 2023, 50(11): 55-61.

ZHANG Yang, WANG Rui, WU Guanfeng, LIU Hongyi. Parallel Mining Algorithm of Frequent Itemset Based on N-list and DiffNodeset Structure [J]. Computer Science, 2023, 50(11): 55-61.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221000011

基于 N-list 和 DiffNodeset 结构的频繁项集并行挖掘算法

张　阳[1,2]　王　瑞[3]　吴贯锋[1,2]　刘弘毅[1,2]
1 西南交通大学数学学院　成都 611756
2 西南交通大学系统可信性自动验证国家地方联合工程实验室　成都 611756
3 航天物联网技术有限公司　北京 100094
(1170922259@qq.com)

摘　要　频繁项集挖掘是数据挖掘中的一个基本问题,在许多数据挖掘应用中发挥着重要作用。针对并行频繁项集挖掘算法 MrPrePost 在大数据环境存在密集数据集下算法效率下降、计算节点负载量不均衡和冗余搜索等问题,提出了基于 N-lists 和 DiffNodeset 两种结构的并行频繁项集挖掘算法(Parallel Mining algorithm of Frequent Itemset based on N-list and DiffNodeset structure,PFIMND)。首先,根据 N-list 和 DiffNodeset 在存储不同数据集上的优势,设计了稀疏度估计函数(Sparsity Estimation,SE),根据数据集稀疏程度灵活选取其中之一压缩数据集,相比采用单一存储结构消耗的内存更少;其次,提出了计算量估计函数(Computation Estimation,CE)来估计频繁 1 项集 F-list 中每一项的负载量,并根据计算量进行均匀分组;最后采用集合枚举树作为搜索空间,为避免组合爆炸和冗余搜索问题,设计了超集剪枝策略和基于宽度优先搜索的剪枝策略,生成最终的挖掘结果。实验结果表明,相比同类算法 HP-FIMBN,PFIMND 算法在 Susy 数据集上挖掘频繁项集的效果提升了 12.3%。

关键词:频繁项集;负载估计;MapReduce;稀疏度估计;集合枚举树
中图法分类号　TP311

Parallel Mining Algorithm of Frequent Itemset Based on N-list and DiffNodeset Structure

ZHANG Yang[1,2],WANG Rui[3],WU Guanfeng[1,2] and LIU Hongyi[1,2]
1 School of Mathematics,Southwest Jiaotong University,Chengdu 611756,China
2 National-Local Joint Engineering Laboratory of System Credibility Automatic Verification,Southwest Jiaotong University,Chengdu 611756,China
3 Aerospace Internet of Things Technology Co.,Ltd,Beijing 100094,China

Abstract　Frequent itemset mining is a basic problem of data mining and plays an important role in many data mining applications. In order to solve the problems of the parallel frequent itemset mining algorithm(MrPrePost) in big data environment,such as algorithm efficiency degradation,unbalanced load of computing nodes and redundant search,this paper proposes a parallel frequent itemset mining algorithm(PFIMND),which is based on N-lists and DiffNodeset. Firstly,according to the advantages of N-list and DiffNodeset data structures,the data set sparsity estimation function(SE) is designed,and one of them is selected to store data according to the data set sparsity. Secondly,the computational estimation function(CE) is proposed to estimate the load of each item in the frequent 1-item set F-list,and the load is evenly grouped according to the computational cost. Finally,the set enumeration tree is used as the search space. In order to avoid combination explosion and redundant search problems,the superset pruning strategy and the pruning strategy based on width first searches are designed to generate the final mining results. Experimental results show that compared with the similar algorithm(HP-FIMND),the effect of PFIMND algorithm in mining frequent itemsets on Susy dataset is improved by 12.3%.

Keywords　Frequent itemset,Load estimation,MapReduce,Sparse estimation,Set-enumeration tree

1　引言

关联规则挖掘[1]是数据挖掘的重要分支之一,旨在发现各数据项之间的相互关系,被广泛应用于市场营销、社交网络等领域。频繁项集挖掘是关联规则挖掘中的一项基本任务[2],它负责提取数据中频繁发生的事件、模式或项。来自这种模式分析的见解在决策过程中有着重要作用。然而,挖掘这类模式的算法时,其计算复杂度随着数据中项的数量呈

到稿日期:2022-10-07　返修日期:2023-02-22
基金项目:国家自然科学基金(62106206)
This work was supported by the National Natural Science Foundation of China(62106206).
通信作者:吴贯锋(wgf1024@swjtu.edu.cn)

指数级增加，而且挖掘过程将消耗大量内存。因此，有必要提出高效的解决方案。

传统的频繁项集挖掘算法主要分为3类。1)产生-测试：此类算法通过排列组合的方式，从频繁 k-项集迭代地生成候选 $(k+1)$-项集，再遍历数据库对其计数，然后根据支持度阈值筛选出频繁 $(k+1)$-项集。代表算法为 Ariori[1]，其缺点是需要频繁扫描数据库。2)模式增长：此类算法不生成候选项集，而是将全体数据进行压缩，减少内存消耗并提高搜索效率。代表算法为 FP-growth[3]，其缺点是当数据较大时，压缩结构占用大量内存，搜索效率迅速降低。3)垂直数据集：此类算法将水平数据集转换成垂直数据集格式，将发生的事务列表与每个项集相关联。因此，两个项集的支持度可以简单地通过将它们的列表相交获得。代表算法为 Eclat[4]，其缺点是不适合处理需要较大事务列表相交的数据集。

在大数据环境下，运行时间和内存消耗成为这些传统方法的瓶颈。为此引入并行计算的思想。并行处理是减少大数据处理时间的有效技术。

大数据处理主要采用分治法，即将大数据问题分解成规模较小的子问题求解，然后合并子问题的解，从而得到最终解。基于此，Google 公司研发了一种专门处理大数据的编程模型和实现框架 MapReduce，它具有简单、高效、易伸缩以及高容错性等特点，因此迅速成为大数据处理的主流计算平台[5]。由于 MapReduce 具有自动并行化、高容错性和负载平衡等优点，现在已有许多并行数据挖掘算法被移植到 MapReduce 框架上[6]。

文献[7-9]实现了 Apriori 算法的并行化，可处理一定量级的大数据。但由于该算法需要频繁扫描数据库，会造成大量的 I/O 开销，同时激增的候选集也会消耗大量内存。基于 MapReduce 实现的并行 FP-Growth 算法[10]较好地解决了该问题。首先，FP-Growth 算法仅需扫描数据库两次，且无须生成候选项集。其次，在整个挖掘过程之中，集群中各计算节点是相互独立的，它们无须信息交互。文献[11-12]考虑到部分节点由于计算量的差异可能出现空闲等待、浪费计算资源的情况，设计了负载估计函数，使各计算节点的负载量相对均衡。文献[13-15]基于垂直格式的 Elact 算法提出了它的并行化算法。该类算法设计简单，通过大量的交运算来计算 $k(k>1)$项集的支持度，在一定程度上克服了前面两类算法面对大数据计算力不足的问题。但并行 Elact 算法在将水平上数据集转换成垂直格式后，仍然需要使用 Apriori 算法挖掘频繁项集。

为了减少并行计算中单个节点的内存需求与节点之间的通信量，Liao 等[16]结合传统算法 FP-growth 和 Eclat 的优点提出了 MRPrePost 算法。该算法执行一次 MapReduce 任务后得到频繁 1-项集 F-list，然后访问数据库生成 PPC-Tree[17]，并生成频繁 1-项集的 N-list 压缩结构，最后在集群中的多个计算节点挖掘频繁项集。此过程无须将 PPC-tree 保存在内存中，既提高了项集支持度的计算效率，又减少了内存消耗。但该算法存在不足：处理密集数据时，N-list[17]的基数会非常大，内存消耗剧增，并且项集支持度的计算效率随之降低；未充分考虑计算节点负载不均对算法整体性能的影响；该算法根据类 Apriori 算法原理迭代计算频繁项集，将产生大量的冗余搜索和无效计算。

在最新的研究中，HP-FIMBN[18]算法提出了更合理的负载均衡策略；PFIMD[19]算法采用 DiffNodeset[20]结构代替 N-list 完成对数据集的压缩，以避免 N-list 有时基数过大的问题。

2 相关概念

定义 1(PPC-tree[17]) PPC-tree 是一种树状数据结构，树中的每个节点均由节点名称 *item-name*、节点计数 *count*、孩子节点集合 *children-list*、先序遍历序号 *pre-order* 和后序遍历序号 *post-order* 5 部分组成。

定义 2(PP-code[17]) PP-code 即先序后序编码，它由 *pre-order*，*post-order* 和 *count* 3 部分组成。对于 *PPC-tree* 中的任意节点 N，其 PP-code 为 $(N.\ pre\text{-}order, N.\ post\text{-}order, N.\ count)$。

定理 1(祖先孩子关系) 设 N_1 和 N_2 是 PPC-tree 中任意的两个节点 $(N_1 \neq N_2)$，若满足 $N_1.\ pre\text{-}order < N_2.\ pre\text{-}order$ 且 $N_1.\ post\text{-}order > N_2.\ post\text{-}order$，则称节点 N_1 是 N_2 的祖先节点，节点 N_2 是 N_1 的孩子节点。

定义 3("≺"关系[17]) 给定 i_1 和 i_2 为频繁 1 项集中的任意两项，若 i_1 的支持度小于 i_2，则表示为 $i_1 \prec i_2$。

2.1 N-list 相关定义

定义 4(频繁 1 项集的 N-list[17]) 在 PPC-tree 中，将 *item-name* 属性相同的所有节点对应的 PP-code 按照 *pre-order* 属性升序排列得到的编码序列集合称为频繁 1 项集的 N-list。

定义 5(k-项集的 N-list[17]) 设有频繁 1 项集 F，$i_k \in F$，$\forall k$。相同后缀的两个频繁 $k-1$ 项为 $i_k X: i_k i_{k-2} i_{k-3} \cdots i_2 i_1$ 和 $i_{k-1} X: i_{k-1} i_{k-2} i_{k-3} \cdots i_2 i_1$，其中 $i_{k-1} \prec i_k$，$k \geqslant 2$。则其对应的 N-list 结构分别表示为：

$$N\text{-}list(i_k X) = \{(x_{11}, y_{11}, z_{11}), (x_{12}, y_{12}, z_{12}), \cdots, (x_{1m}, y_{1m}, z_{1m})\} \tag{1}$$

$$N\text{-}list(i_{k-1} X) = \{(x_{21}, y_{21}, z_{21}), (x_{22}, y_{22}, z_{22}), \cdots, (x_{2n}, y_{2n}, z_{2n})\} \tag{2}$$

则 k-项集 $i_k i_{k-1} X$ 的 N-list 定义如下：首先，对于 $\forall (x_{1p}, y_{1p}, z_{1p}) \in N\text{-}list(i_k X)\ (1 \leqslant p \leqslant m)$，$\forall (x_{1q}, y_{1q}, z_{1q}) \in N\text{-}list(i_{k-1} X)\ (1 \leqslant q \leqslant n)$，如果满足条件 $x_{1p} < x_{2q}$，且 $y_{1p} > y_{2q}$，则将 (x_{1p}, y_{1p}, z_{2q}) 加入到 $i_k i_{k-1} X$ 的 N-list 当中，得到初始的 N-list。其次，遍历 $i_k i_{k-1} X$ 的 N-list，合并 *pre-order* 和 *post-order* 属性值相等的 PP-code，得到 k-项集 $i_k i_{k-1} X$ 最终的 N-list。

定理 2(频繁项集的支持度) 给定项 I 的 N-list 为 $\{(x_1, y_1, z_1), (x_2, y_2, z_2), \cdots, (x_m, y_m, z_m)\}$，则 I 的支持度为 $\sum_1^m z_k$。

2.2 DiffNodeset 相关定义

定义 6(2 项集的 DiffNodeset[20]) 设 i_1 和 i_2 $(i_1 \prec i_2)$ 为频繁 1 项集中的任意两个项，其 N-list 结构分别为 N_1 和 N_2。记 2 项集 $i_2 i_1$ 的 DiffNodeset 结构为 DNS_{21}，其定义如下：

$$DN_{12} = \{(x.\ pre\text{-}order, x.\ post\text{-}order, x.\ count) \mid x \in N_2 \wedge \neg(y \in N_1)\} \tag{3}$$

其中，节点 x 和 y 是祖先孩子关系。

定义 7（k-项集的 DiffNodeset[20]）　设 k 项集 $P=i_k i_{k-1}\cdots i_2 i_1$（$i_1\prec i_2\prec\cdots\prec i_{k-1}\prec i_k$），若频繁 $k-1$ 项集 $P_1=i_{k-1}\cdots i_2 i_1$，$P_2=i_{k-2}\cdots i_2 i_1$，对应的 DiffNodeset 分别为 DN_1 和 DN_2，记 k 项集 P 的 DiffNodeset 为 DNp，其计算方式如下：

$$DN_p=DN_2/DN_1 \tag{4}$$

定理 3（k-项集的支持度[20]）　给定 $k(k>1)$ 项集 $P=i_k i_{k-1}\cdots i_2 i_1$，其 DiffNodeset 结构为 $DN_p=\{(x_1,y_1,z_1),(x_2,y_2,z_2),\cdots,(x_s,y_s,z_s)\}$。若 $k-1$ 项集 $P_1=i_{k-1}\cdots i_2 i_1$ 的支持度为 $support(P_1)$，则项集 P 的支持度定义如下：

$$support(P)=support(P_1)-\sum_1^s z_k \tag{5}$$

3　PFIMND 算法

PFIMND 算法主要包含搜索频繁 1 项集 F-list、频繁 1 项分组和并行挖掘频繁项集 3 个步骤：1）开启一次 MapReduce 任务搜索频繁 1 项集 F-list；2）集群中计算节点若负载不均衡则会导致算法整体性能下降，因而提出计算量估计函数 CE 来估计频繁 1-项的计算量，并以此动态分组后得到分组表 G-list；3）并行挖掘频繁项集分为 Map 和 Reduce 两个阶段。Map 阶段根据 F-list 和 G-list 划分搜索空间，并对事务数据库中的每条事务进行筛选和排序。Reduce 阶段首先拉取 Map 阶段的输出结果构建组内 PPC-tree；其次，考虑到 N-list 和 DiffNodeset 两种数据结构对不同数据集的压缩效果不同，设计了数据稀疏度估计函数（SE），以选择占用内存较少的结构来保存数据；最后采用 Apriori 思想获取频繁 2-项集，采用集合枚举树挖掘频繁 k 项集。同时为避免组合爆炸和冗余搜索问题，设计了超集剪枝策略和宽度优先剪枝策略。

3.1　搜索频繁 1 项集

基于 MapReduce 框架搜索频繁 1 项集 F-list，主要分为如下几步：首先采用 Hadoop 自带的文件分块策略将原始数据 DB 切割成大小相等的文件块 block；其次是 map 阶段，以 block 为输入，将数据中的每个项 *item* 映射为〈*key*＝*item*，*value*＝1〉键值对的形式输出并缓存；然后是 Reduce 阶段，拉取缓存中相同 *key* 的键值对，将对应的 *value* 属性值累加得到项 *key* 的支持度；最后根据支持度阈值 *min-sup* 筛选出频繁 1 项集，并按照支持度以非递增排序得到 F-list。

3.2　频繁 1 项分组

首先，引入计算量估计函数（CE）来预测频繁 1 项集中每个项的计算量，按计算量降序排序得到〈频繁 1 项-计算量〉的映射表 L-map，并根据分组数 *gid* 计算各节点的负载平均值 *avg*；然后先将计算量大于 *avg* 的频繁项分组，再将剩余频繁项在剩余组中均匀分布；最后将分组结果 G-list 缓存到分布式文件系统 HDFS 中，使得 Hadoop 集群中的任意节点均可访问。

定义 8（计算量估计函数（CE））　设 *item* 是频繁 1 项集 F-list 中的项，其支持度为 *count*，且 *item* 在 F-list 中的位置为 l，$0\leqslant l\leqslant s-1$，$s$ 为 F-list 中的元素个数。则 *item* 的计算量定义如下：

$$CE(item)=\min\{count,2^l\} \tag{6}$$

说明：本文对项集的支持度计数采用 N-list 结构的交运算或 DiffNodeset 结构差集运算完成，其计算量直接依赖于频繁 1 项 *item* 对应的 PP-code 的个数，它等价于 *item* 在 PPC-tree 中的节点个数，显然节点数的最大值不超过 *item* 的支持度 *count*。另一方面，PPC-tree 是对原始数据集无损压缩的结果，在 PPC-tree 的同一条路径中不会出现两个相同 *item-name* 属性的节点 。若频繁 1 项 *item* 在 F-list 中的下标位置为 $l(0\leqslant l\leqslant s-1)$，那么 PPC-tree 中包含 *item* 的不同路径最多有 2^l 条，此时 *item* 对应 N-list 的最大长度为 2^l。

综上所述，频繁 1 项 *item* 的计算量应不超过 $\min\{count,2^l\}$。

本文算法通过向每个组分配大致相等的计算量以实现负载均衡。通过 CE 计算得到频繁 1-项集的计算量映射表 L-map，若有计算量不小于平均值的频繁 1 项，则将其单独分为一组，否则根据动态分组原则进行分组。分组算法如算法 1 所示。

算法 1　根据计算量对频繁 1 项均匀分组

```
输入：L-map，分组个数 gid
输出：频繁 1 项分组表 G-list
1. G-list←Ø
2. if gid≤size(L-map) do
3.    //计算频繁 1-项集的总负载量
4.    totalLoad←0
5.      for each item in G-list do
6.         totalLoad←totalLoad+item.load
7.      end for
8.      //首先，将计算量不小于平均值项单独分组
9.      avgLoad←totalLoad/gid
10.     index←0
11.     for each item in G-list do
12.          if item.compitition≥avgLoad do
13.             G-list[index]←item
14.             Remove item in L-map
15.             index←index+1
16.       end if
17.     end for
18.     gid←gid-(index+1)
19.     //其次，将余下的频繁 1-项均匀划分到其他空组
20.     if gid>0 do
21.       for each G-list[index] in G-list do
22.          groupLoad←0
23.          i←0
24.          for each item in L-map do
25.             //基于贪心思想对 item 进行动态分组
26.               if abs(groupLoad+item.load-avgLoad)
27.                  ≤abs(groupLoaad-avgLoad) do
28.                  groupLoad←groupLoad+item.load
29.                  G-list[index]←item
30.                  Remove item in L-map
31.               else do
32.                  i←i+1
33.               end if
34.            end for
35.         end for
```

```
36.        end if
37.      end if
38. return G-list
```

3.3 Map 阶段:划分搜索空间

在并行频繁项集挖掘的 map 阶段,其主要任务是将预处理后的数据根据频繁 1-项集分组,并将结果映射到集群中各个计算节点上,即完成对整个数据集搜索空间的划分,使得每个计算节点可以独立完成分配数据的挖掘工作,减少节点之间的通信,从而减少算法的时间消耗。这一阶段是根据频繁 1-项集的分组信息表 G-list,再次扫描数据库,将数据库中的事务划分到每个计算节点上。首先,根据频繁 1-项集 F-list 对每一个事务进行筛选,然后按照支持度对 F-list 中的项降序排序,得到有效记录 t。此时,t 对应 PPC-tree 中的一条分支。其次,按照划分搜索空间的思想进行分组,即每个分组中有且只能有一条该路径的子路径,且该分支中的非叶子节点包含在组内频繁 1-项集中。

3.4 并行挖掘频繁项集

分组后,每个分组都对应集群中的一个计算节点,然后每个计算节点独立完成组内频繁项集挖掘。首先需要判断划分到组内的数据是否密集,其次根据判断结果选择压缩结构进行频繁项集挖掘。具体步骤为:1)在拉取划分到本组的事务数据时,记录每一条数据的长度,即其中项的个数,得到组内事务的长度集合;2)根据稀疏度估计函数计算组内数据是否稀疏,稀疏数据集采用 N-list 结构挖掘频繁项集,密集数据则采用 DiffNodeset 结构挖掘频繁项集。

定义 9(数据集稀疏度估计 SE) 假设有事务数据集 T,其事务长度集为 $L=\{l_1,l_2,l_3,\cdots,l_n\}$,则 T 的稀疏度估计如下:

$$SE=n(\max L-\min L)/\sum_{1}^{n}l_k \tag{7}$$

一般认为,当 $SE>1$ 时,T 为稀疏型数据;当 $SE\leqslant 1$ 时,T 为密集型数据。

定理 4 设有筛选后的事务集 Ω(其中每个项都是频繁的),若 Ω 为密集型数据,则使用 DiffNodeset 数据结构存放数据占用的内存较少;若 Ω 为稀疏型数据,则使用 N-list 数据结构存放数据占用的内存较少。

证明:采用 N-list 或 DiffNodeset 结构占用内存的大小取决于 PPC-tree 中"祖先孩子关系"的数量,因为 N-list 实际上就是记录这种关系,而 DiffNodeset 记录了不存在这种关系的节点信息。当事务集 Ω 稀疏时,PPC-tree 层级较高,节点比较分散,因而"祖先孩子关系"较少;反之,当 Ω 密集时,PPC-tree 层级较低,节点分布比较集中,从而产生了大量的"祖先孩子关系"。因此,当事务集 Ω 稀疏时,采用 N-list 结构挖掘频繁项集;当 Ω 密集时,采用 DiffNodeset 挖掘频繁项集较为合理。

首先,每个计算节点根据搜索空间划分情况拉取分配数据 groupData,估计该数据的稀疏度并构建组内 PPC 树;然后计算组内候选 2-项集;最后选取合适的结构开始组内频繁项集挖掘。具体算法流程如算法 2 所示。

算法 2 组内挖掘频繁项集

```
输入:groupData,组内候选 2-项集 candI2
      最小支持度 min-sup,频繁 1-项集 F-list
输出:组内频繁项集 localFI
localFI←Ø
步骤 1:建立组内 PPC-tree
root←null //初始化 PPC 树根节点
for each path in groupData do
    curNode←root
      for each item in path do
      //创建 PPC-tree 节点
      createTree(curNode,item)
    end for
end for
步骤 2:判断组内的数据是否密集
dense←false
Estimate se of groupData by Eq.(6)
if se<1 do
  dense←true
end if
步骤 3:挖掘组内频繁项集
//频繁 1 项的 N-list 和 DiffNodeset 压缩结构相同
setF1←getF1(root) //组内频繁 1-项集的压缩结构集合
for each item insetF1.keySet() do
    //将频繁 1 项添加到组内频繁项集合
    localFI=localFI∪item
end for
root←null //释放内存
//组内数据密集 分布
if dense is true do
    //计算频繁 2 项集的 DiffNodeset
    dnsF2←getF2(candI2,setF1,min_sup)
      for eachi1i2in dnsF2.keySet() do
        //创建集合枚举树
        et←createET(i1i2,dnsF2,F-list)
        for each node inet do
          //保存组内频繁项集
          localFI=localFI∪node.name
        end for
      end for
    //组内数据稀疏分布
    else do
      //计算频繁 2 项集的 N-list
      nlF2←getF2(candI2,setF1,min_sup)
      for eachi1i2in nlF2.keySet() do
        //创建集合枚举树
        et←createET(i1i2,nlF2,F-list)
        for each node in et do
          //保存组内频繁项集
          localFI=localFI∪node.name
        end for
      end for
    end if
    return localFI
```

由于 MrPrePost 算法是采用"产生-测试"的类 Apriori 算法的思想来挖掘频繁项集,因此这种方式不可避免地会产生大量冗余搜索的问题,使得搜索效率大大降低。为优化算法,

本文采用集合枚举树作为搜索空间，并提出超集剪枝策略来减少冗余搜索。另一方面，由于生成集合枚举树本身存在组合爆炸问题，因此提出了基于宽度优先搜索的剪枝策略，以减少枚举树在频繁项集搜索过程中的内存占用。

定理 5（超集剪枝策略）　若候选 k-项集 $P=i_k i_{k-1}\cdots i_2 i_1$（$i_1<i_2<\cdots<i_{k-1}<i_k$），设有 P 的子集 $P_1=i_{k-1}\cdots i_2 i_1$ 和 $P_2=i_k i_{k-2}\cdots i_2 i_1$，令 F_{k-1} 为频繁 $k-1$ 项集。那么当 $P_1\notin F_{k-1}$ 或者 $P_2\notin F_{k-1}$ 时，即可剪枝掉集合枚举树中名称为 P 的节点。

证明：根据频繁项集的反单调性质可知，非频繁项集的超集一定是非频繁的。

定理 6（宽度优先搜索剪枝策略）　以频繁 2-项集为根节点迭代生成集合枚举树，第 $k(k\geqslant 1)$ 轮迭代结束时，仅保留该树中频繁 $(k+2)$-项集对应的节点，第 $k+1$ 轮迭代之前删除数据结构集合 Ω 中所有 k-项集（$k\geqslant 2$）对应的数据存储结构。

证明：考虑到在“产生-测试”模式下，频繁 $k+1$-项集的产生测试仅与 k-项集有关，因此应该删除无关项集对应的数据结构（N-list 或 DiffNodeset），减少内存占用。另外，为避免反复搜索枚举树获取频繁 k 项集对应的数据结构，此算法设计了一个全局参数 Ω 随迭代向下转递，它最多包含相邻两层频繁项集对应的数据。

集合枚举树的生成及其剪枝过程的伪代码如算法 3 所示。当所有节点均完成挖掘任务时，将每个节点所得的局部频繁项集汇总形成全体频繁项集。

算法 3　集合枚举树的生成

```
输入：频繁 2 项 i1i2（i1＞i2），频繁 2 项集对应的压缩结构的集合
      setF2，频繁 1 项集 F-list
输出：集合枚举树的根节点 root
root←i1i2//初始化集合枚举树
步骤 1：获取 i1i2 的前缀集合
prefixroot←getPrefix(root,F-list)
Function getPrefix(itemset,F-list)
prefix←Ø
for each item in F-list do
    i←itemset[0]
    if item≠i do
      prefix=prefix∪item
    else break
    end if
end for
Return prefix
步骤 2：调用方法生成枚举树
  createTree(root,prefixroot,setF2)
Return root
Function createTree(τ,prefix,Set)
for each node in Set do
    subset={item,comm}
    if node.level ＜τ.length−1 do
      Set.remove(node)//DP 剪枝
    end if
  end for
  comm←τ.substring(1)
  for each item in prefix do
    if subset not inSet.keyset()do
      //超集剪枝
      continue
    else do
      ns={item,τ}
    nodens←getNode(Set.get(τ),Set.get(subset))
    if nodens.support≥minsupdo
      τ.child.add(ns)
      Set.add(nodens)
    end if
  end if
end for
for each item in this.childNode do
    prefixitem←getPrefix(item,F-list)
    createNode(item,prefixitem,Set)
  end for
  Return τ
```

3.5　算法分析

PFIMND 算法主要包括搜索频繁 1-项集、频繁 1-项集分组、划分搜索空间和并行挖掘频繁项集这 4 个阶段，因此算法的时间复杂度主要由这 4 部分组成。

首先，搜索频繁 1-项集在 map 过程需要遍历到事务数据集中每条记录的每一项，假设数据集中总的记录数为 T_D，每条记录的平均长度为 L，则 map 过程的时间复杂度为 $O(T_D\times L)$。在 reduce 过程中需要将相同 key 值的键值对〈$key=item$，$value=1$〉进行合并以获得每个 1 项的支持度。假设 mapper 节点个数为 N，每个节点的键值对的平均个数为 M，则 reduce 过程的时间复杂度为 $O(M\times N)$，因此搜索频繁 1-项集阶段的总时间复杂度为 $O(T_D\times L+M\times N)$。频繁 1-项集分组是根据其中每一项的计算量估计值进行均匀分组，假设频繁 1-项的个数为 m，分组数为 G，那么这一阶段的时间复杂度为 $O(m/G)$，即 $O(m)$。第三阶段划分搜索空间需要再次扫描原始数据，仅保留每个事务中的频繁项，然后再根据频繁 1-项集分组情况对该事务进行搜索空间分组，即保证每个分组内有且仅有该事务的一个子集。假设数据集中总的记录数为 T_D，每条记录的平均长度为 L，分组数为 G，则第三阶段的时间复杂度为 $O(T_D\times(L+G))$。

第四阶段并行挖掘频繁项集的时间复杂度主要取决于 $(k+1)$-项集支持度的计算，假设用于生成 $(k+1)$-项集的两个 k-项集的 N-list 结构长度分别为 L_{na} 和 L_{nb}，DiffNodeset 结构长度分别为 L_{da} 和 L_{db}，那么并行挖掘频繁项集的时间复杂度为：$O(\sum\sum(\min(\max(L_{na},L_{nb}),\max(L_{da},L_{db}))))$。而在 MrPrePost 算法中，前 3 个阶段的时间复杂度基本相同，并行挖掘频繁项集阶段计算 $(k+1)$-项集的时间复杂度为 $O(\sum\sum(L_{na}+L_{nb}))$。因此 PFIMND 算法具有较低的时间复杂度。

4　实验结果与对比

4.1　实验环境

本文设计的实验环境包含 4 个计算节点，其中 1 个为主节点，3 个为辅助节点。所有节点的处理器均为 Inter i7，每个处理器拥有 8 个处理单元，运行内存均为 16GB。这 4 个节点处于同一局域网下，用 100 Mbps 的以太网连通。每个计算

节点装配 Hadoop 2.7.7,Java JDK 1.8.0 软件和 centos 7 操作系统。各计算节点的具体配置信息如表 1 所列。

表 1 节点配置信息

Table 1 Node configuration

节点类型	主机名称	IP 地址	节点功能
Master	master	192.168.31.101	NameNode
Slaver	s1	192.168.31.102	DataNode
Slaver	s2	192.168.31.103	DataNode
Slaver	s3	192.168.31.104	DataNode

4.2 实验数据

本文使用的 3 个实验数据集都可以在 SPMF[21] 文库下载,分别是 webdocs,kosarak 和 susy 数据集。其中 webdocs 数据集约有 169 万条记录和 526 万个项,具有数据量大和项目多等特点;kosarak 数据集约有 99 万条记录和 4 万个项,具有数据体量小和项数多等特点;susy 数据集约有 500 万条记录和 190 个项,具有数据量大、记录长度均匀且项数少等特点。这 3 个数据集的具体信息如表 2 所列。

表 2 实验数据集

Table 2 Datasets used in experiment

数据集	记录数	项数	数据分布特征
webdocs	1692082	5267656	稀疏
kosarak	990002	41270	稀疏
susy	5000000	190	密集

4.3 评价指标

为评估 PFIMND 频繁项集挖掘算法的运算效率,本文采用加速比作为算法性能的衡量指标。加速比指在相同任务量的情况下,使用单一处理器进行运算与使用多个处理器运算所消耗时间的比值。加速比越大,说明算法在并行计算过程中消耗的时间越少,即算法的挖掘效率越高。

4.4 PFIMND 算法可行性分析

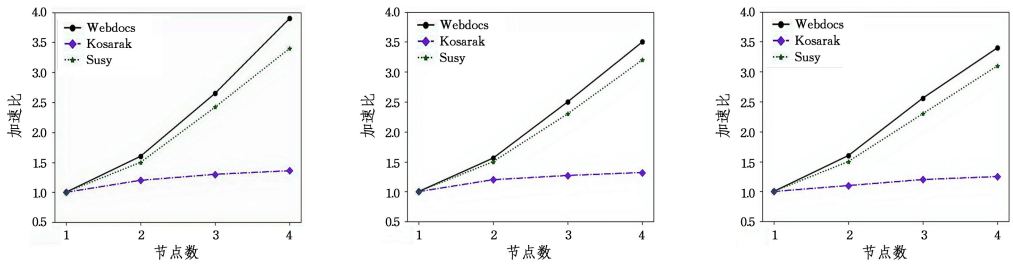
本节主要通过对比 3 个数据集在不同最小支持度阈值下的加速比,来验证 PFIMND 算法挖掘频繁项集的可行性。本文选取 1000,5000 和 10000 分别作为最小支持度阈值,将 PFIMND 算法分别应用于不同数据集上,并单独执行 10 次,取其均值作为最终结果。PFIMND 算法在不同数据集下的加速比如图 1 所示。

从图 1 可以看出,PFIMND 算法在处理体量较大的数据集时具有较高的加速比;相反,在处理体量较小的 kosarak 数据集时,加速比随计算节点增加获得的增幅并不明显。这是因为数据量较小时,现有的数据量不能将集群处理的数据量最大化,此时将数据依据算法中的规则划分到各节点上处理反而会增加时间开销。然而在大数据环境下,算法优化的关键在于有效地减小频繁 1 项集的规模以及快速准确地合并频繁项集。在大规模数据上,PFIMND 算法的挖掘加速比随着节点数的增加呈现出明显的线性增长趋势,说明该算法适用于对较大规模数据的处理,且具有较强的频繁项集挖掘潜力。

4.5 挖掘性能分析

本文分别在 3 个数据集上进行多次对比实验,在算法挖掘过程中将本文算法与 MrPrePost,HP-FIMBN 和 PFIMD 算法的运行时间进行对比。其中,HP-FIMBN 是基于 N-list 结构提出的并行算法,而 PFIMD 是基于 DiffNodeset 结构提出的并行挖掘算法。

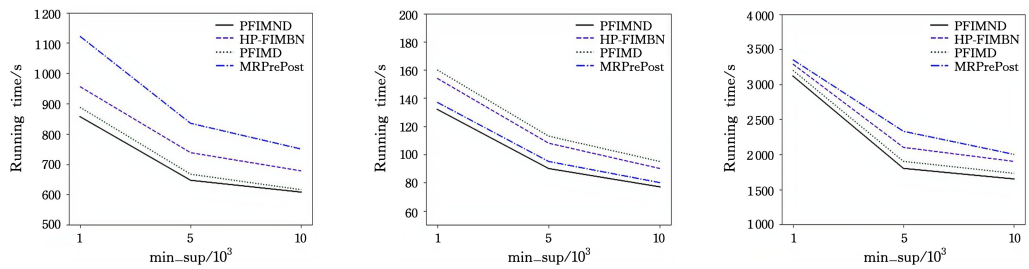
为验证 PFIMND 算法的有效性,本文分别在 webdocs,kosarak 和 susy 这 3 个数据集上将其运行时间与 PFIMND,MrPrePost,HP-FIMBN 和 PFIMD 算法对比。在独立运行 10 次后取平均值进行分析,通过对比这些算法总体的运行时间来评估 PFIMND 算法的性能。结果如图 2 所示。



(a)支持度为 1000 (b)支持度为 5000 (c)支持度为 10000

图 1 各算法在不同数据集上的加速比

Fig. 1 Acceleration ratio of each algorithm on different data sets



(a)susy (b)kosarak (c)webdocs

图 2 不同算法的执行时间

Fig. 2 Running time of different algorithms

从图 2 可以看出,相比 MrPrePost 和 HP-FIMBN 算法,PFIMND 算法在 3 个数据集上的整体运行时间均有减少。

在 susy 数据集上 PFIMND 算法的运行时间下降幅度最大，同比下降 22.6%和 12.3%。这是由于 PFIMND 算法采用了合理的分组策略，保证了集群的负载均衡，使每个分组具有相当规模的 PPC-Tree，有效减少了遍历所需的时间；另一方面是因为 susy 数据集具有整体呈密集分布，PFIMND 算法基本采用 DiffNodeset 结构挖掘频繁项集，因此占用的内存较少，且计算项集支持度的效率较高。

结束语　本文提出的基于 N-list 和 DiffNodeset 两种数据结构的并行频繁项集挖掘算法通过设计计算量估计函数对频繁 1-项集均匀分组，来解决节点负载不均匀的问题。为进一步提升算法运行效率，本文根据 N-list 和 DiffNodeset 两种数据结构的优缺点设计了离散度估计函数，判断分组后的数据是否离散，并根据判断结果采用相应的数据结构挖掘频繁项集，以加快项集支持度的计算，提高节点合并速度，并降低节点的负载压力。在此基础上，以频繁 2 项集作为根节点，构建超集剪枝和宽度优先搜索剪枝策略，在生成集合枚举树时同步进行剪枝，从而降低内存消耗。在 webdocs，kosarak 和 susy 这 3 个数据集上的实验结果验证了 PFIMND 算法的有效性，相比 MrPrePost，HP-FIMBN 等算法，PFIMND 算法对频繁项集的挖掘效果更好，能够在一定程度上提高频繁项集的挖掘效率。另一方面，此算法的效率在很大程度上依赖于数据集的分布特征，若能够更准确地判断每个节点上的数据分布情况，算法的整体性能将会得到进一步提高。

参 考 文 献

[1] AGRAWAL R，MANNILA H，SRIKANT R，et al. Fast discovery of association rules[J]. Advances in Knowledge Discovery and Data Mining，1996，12(1)：307-328.

[2] LUNA J M，FOURNIER-VIGER P，VENTURA S. Frequent itemset mining：A 25 years review[J]. Wiley Interdisciplinary Reviews：Data Mining and Knowledge Discovery，2019，9(6)：e1329.

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

[4] ZAKI M J. Scalable algorithms for association mining[J]. IEEE Transactions on Knowledge and Data Engineering，2000，12(3)：372-390.

[5] DEAN J，GHEMAWAT S. MapReduce：simplified data processing on large clusters[J]. Communications of the ACM，2008，51(1)：107-113.

[6] XIAO W，HU J. Paradigm and performance analysis of distributed frequent itemset mining algorithms based on MapReduce[J]. Microprocessors and Microsystems，2021，82：103817.

[7] QIN J，HAO T S，DONG Q Q. Parallel improvement of Apriori algorithm based on MapReduce[J]. Computer Technology and Development，2017，27(4)：64-68.

[8] GUO F F，LIANG X，WANG H Q，et al. A parallel Apriori algorithm based on multi tree[J]. Small Microcomputer System，2015，36(6)：1176-1180.

[9] NADIMI-SHAHRAKI M H，MANSOURI M. Hp-Apriori：Horizontal parallel-apriori algorithm for frequent itemset mining from big data[C]// 2017 IEEE 2nd International Conference on Big Data Analysis(ICBDA). IEEE，2017：286-290.

[10] LI H，WANG Y，ZHANG D，et al. PFP：parallel FP-growth for query recommendation[C]// Proceedings of the 2008 ACM Conference on Recommender Systems. 2008：107-114.

[11] ZHOU L，ZHONG Z，CHANG J，et al. Balanced parallel FP-growth with MapReduce[C]// 2010 IEEE youth Conference on Information，Computing and Telecommunications. IEEE，2010：243-246.

[12] GAO Q，WAN X D. Parallel FP-growth algorithm based on load balance[J]. Computer Engineering，2019，45(3)：32-35，40.

[13] MOENS S，AKSEHIRLI E，GOETHALS B. Frequent itemset mining for big data[C]// 2013 IEEE International Conference on Big Data. IEEE，2013：111-118.

[14] ZHANG Z G，JI G L，TANG M M. A new algorithm for parallel mining frequent item sets mreclat[J]. Computer Applications，2014，34(8)：2175-2178.

[15] FENG X J，PAN X. Parallel Eclat algorithm based on spark[J]. Computer Application Research，2019，36(1)：18-21.

[16] LIAO J，ZHAO Y，LONG S. MRPrePost—A parallel algorithm adapted for mining big data[C]// 2014 IEEE Workshop on Electronics，Computer and Applications. IEEE，2014：564-568.

[17] DENG Z H，WANG Z H，JIANG J J. A new algorithm for fast mining frequent itemsets using N-lists[J]. Science China Information Sciences，2012，55(9)：2008-2030.

[18] LIU W M，ZHANG C，MAO Y M. Hybrid parallel frequent itemset mining algorithm using n-list structure[J]. Computer Science and Exploration，2022，16(1)：120-136.

[19] MAO Y M，GENG J H，MWAKAPESA D S，et al. PFIMD：a parallel MapReduce-based algorithm for frequent itemset mining[J]. Multimedia Systems，2021，27(4)：709-722.

[20] DENG Z H. DiffNodesets：An efficient structure for fast mining frequent itemsets[J]. Applied Soft Computing，2016，41：214-223.

[21] FOURNIER-VIGER P，GOMARIZ A，SOLTANI A，et al. SPMF：OpenSource data mining platform[EB/OL]. http://www. philippe-fournier-viger. com/spmf.

ZHANG Yang，born in 1998，postgraduate，is a member of China Computer Federation. His main research interests include data mining and pattern discovery.

WU Guanfeng，born in 1986，Ph.D，is a member of China Computer Federation. His main research interests include intelligent information processing and parallel computing.

（责任编辑：何杨）