DOI:10.15923/j.cnki.cn22-1382/t.2018.2.08

基于 Hadoop 的 2FP-Growth 算法

王泽儒， 王红梅*， 李芬田

（长春工业大学 计算机科学与工程学院，吉林 长春 130012）

摘 要： 在 2FP-Grwoth 算法的基础上提出并行处理算法，基于 Hadoop 平台下的 Map/Reduce编程模式对 2FP-Growth 算法进行处理。通过实验和 FP-Growth 算法、COFI 算法以及 2FP-Growth 算法验证了基于 Hadoop 的并行 2FP-Growth 算法的正确性和高效性。

关键词： 2FP-Growth；Map/Reduce 模式；频繁项集；内存溢出

中图分类号： TP 311.13 **文献标志码：** A **文章编号：** 1674-1374(2018)02-0150-06

2FP-Growth algorithm based on Hadoop

WANG Zeru， WANG Hongmei*， LI Fentian

(School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China)

Abstract: A parallel algorithm based on the 2FP-Grwoth algorithm is proposed, which is dealt with Map/Reduce programming mode based on Hadoop platform. With FP-Growth algorithm, COFI algorithm and 2FP-Growth algorithm simulation, the parallel 2FP-Growth algorithm is verified.

Key words: 2FP-Growth; Map/Reduce mode; frequent item-sets; memory overflow.

0 引 言

目前，在数据挖掘领域关联规则[1]是比较重要的一个研究课题，它反映了大量数据中项目与项目之间的联系或者关系。频繁项集的产生是关联规则挖掘应用中的最重要一步。近年来，在频繁项集挖掘中，许多学者先后提出了挖掘算法，例如：Apriori、FP-Growth、PARTITION 等挖掘算法，在众多挖掘算法中，FP-Growth 算法最为著名，因为它在前一个算法 Apriori 的基础上提出，并且在挖掘效率上有一个数量级的改善，FP-Growth 算法的思想是：

1)将事务数据集压缩成一棵 FP 树；

2)根据 FP 树产生后缀模式和项头表，以此找出所有条件模式基，遍历条件模式基构造出类似频繁项 1-项集合 L 的频繁项集合 L_i；

3)根据 L_i 再遍历条件模式基从而构造新的条件 FP-tree，以此来进行迭代挖掘。通过上述过程来看，FP-Growth 算法每次构造出新的 FP 树之前都要两次遍历条件模式基。

在大数据时代背景下，由于数据呈现指数增长趋势，经典的 FP-Growth 算法在生成新的条件

收稿日期： 2018-03-25

基金项目： 吉林省科技厅科技发展计划基金资助项目(20160203010GX)

作者简介： 王泽儒(1993－)，男，汉族，山东临沂人，长春工业大学硕士研究生，主要从事数据挖掘方向研究，E-mail:wangzeru1993@163.com. *通讯作者：王红梅(1968－)，女，汉族，吉林长春人，长春工业大学教授，博士，主要从事数据挖掘方向研究，E-mail:wanghm@ccut.edu.cn.

FP 树时必须要遍历条件模式基两次，这样使系统反复读取数据库服务器中相同的海量数据，有以下两个缺点：

1)降低了算法的挖掘效率；

2)对数据库服务器产生高负荷，不利于数据库服务器正常运作。

随着数据发展趋势，国内外许多学者对 FP-Growth 算法进行了改进，如文献[2]为了解决数据量指数增长趋势，使得传统 FP-Growth 算法受到限制，所以在 PFP 算法[3]的基础上提出负载均衡的并行算法；文献[4]在现有的并行 FP-Growth 算法基础上，提出负载均衡的算法，并且伴随着剪枝策略，可以解决并行分组数据冗余以及负载不均衡的问题；文献[5]是将 FP-tree 的改进算法 Cantree 在 Hadoop 平台中 Map/Reduce 模式下进行并行化计算；文献[6]在提出并行 FP-Growth 算法的同时，对数据进行分割，然后通过结合的方式对事务进行分片实现并行化，解决了 PFP 在大数据下不能处理的问题；文献[7]在并行算法 PFP 上，对挖掘子节点进行剪枝来减少对数据的处理，以此来提高挖掘效率；文献[8]直接对 FP-Growth 算法进行改进，提出一种只需要扫描一次数据库的节点表算法，该算法不生成项目头表，新增加一个与 FP-tree 相关联的节点表，解决 FP-Growth 算法扫描两次数据库并且会产生大量模式基的问题；文献[9]在不同于 Hadoop 平台的 spark 平台下进行并行化处理，提出基于 spark 平台的并行 FP-Growth 算法；文献[10]先将数据按照垂直排列，然后通过扫描删除不频繁项，并且并行建立 FP-Tree，最后通过迭代生成频繁项集。

FP-Growth 算法存在如下缺点[11]：

1)在第一次扫描数据库时，只对频繁 1-项集进行支持度计数统计，但是计算机扫描数据集时，时间消耗是很大的；

2)FP 树只是单纯的将相同前缀的路径进行合并，并没有考虑剪枝。

针对以上缺点，提出 2FP-Growth 算法。文中在改进算法 2FP-Growth 基础上设计并行运算。最后通过实验对 2FP-Grwoth、FP-Growth 以及 COFI 进行比对，验证并行 2FP-Growth 算法的高效性以及正确性。

1　相关理论

T 为一个数据集，$t_1,t_2,\cdots,t_{10}$是数据集中的每一条事务，见表 1。

表 1　数据集 T

事务	项集	事务	项集
t_1	A,B,C,D,E,F	t_6	B,D,E,G
t_2	A,B,D	t_7	A,C,E,G
t_3	B,D,F	t_8	A,F
t_4	C,D	t_9	A,C,D
t_5	A,B,D,E	t_{10}	A,D,E

1.1　2FP 森林

定义 1　满足下列特性的树结构称为 2-项集频繁模式增长树（2-items frequent-pattent growth tree，简称 2FP 树）：

1)根结点是频繁 2-项集，其数据结构为[2-项集：支持度计数]；

2)除根结点外，其余结点均是频繁 1-项集，其数据结构为[1-项集：支持度计数]；

3)设结点 B 的 1-项集是$\{\beta\}$，对于结点 B 的任意非根祖先结点 A 的 1-项集$\{\alpha\}$，则 2-项集$\{\alpha,\beta\}$是频繁的。

定义 2　由 $m(m\geqslant 0)$个互不相交的 2FP 树构成的森林称为 2-项集频繁模式增长森林(2-items frequent-pattent growth forest，简称 2FP 森林)。

设 min_sup 为 2，对表 1 所示数据集 T 删去非频繁项 G，构建的 2FP 森林如图 1 所示。

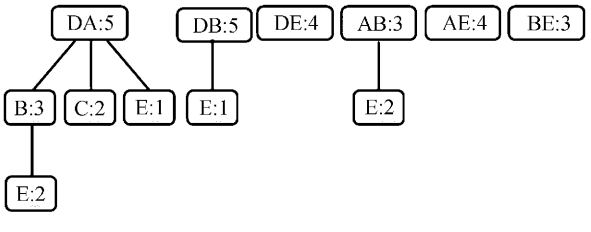


图 1　2FP 森林

定理 1　如果项集 $X=\{x_1,x_2,\cdots,x_k\}$是频繁的，$\forall\alpha\in X$，则 α 一定是频繁的。反之，$\exists x\in\alpha$，如果 α 是非频繁的，则项集 X 一定非频繁的。

定理 2　如果项集 $X=\{x_1,x_2,\cdots,x_k\}$是频繁的，$\forall\alpha\in X\wedge\beta\in X\wedge\alpha\neq\beta$，则 2-项集$\{\alpha,\beta\}$一定是频繁的。反之，$\exists\alpha\in X\wedge\beta\in X\wedge\alpha\neq\beta$，如果 2-项集$\{\alpha,\beta\}$非频繁，则项集 X 一定非频繁的。

定理 3　设某事务包含的项集为 $X=\{x_1,x_2,\cdots,x_m\}$，在 x 中删去非频繁项，并将剩余项按

支持度非升序排列为项集 $Y=\{y_1,y_2,\cdots,y_k\}$ $(k\leqslant m)$，$\forall\alpha\in Y-\{y_k\}$，在该事务不存在2-项集 $\{\alpha,\beta\}$ 的频繁项集。

2FP-Growth算法将提供频繁项的数据集压缩存储到2FP树中，其思想是将频繁2-项集作为树的根节点，然后利用剪枝策略对2FP树进行剪枝，最后把这些2FP树构成2FP森林。2FP-Growth算法过程如下(以表1为例)：

1)FP-Growth算法第一遍扫描数据集仅计算1-项集的支持度，考虑到对数据集扫描的时间代价，2FP-Growth算法在扫描第一遍数据集时统计所有1-项集和2-项集的支持度计数。对于表1所示数据集 T，扫描一遍数据集计算1-项集和2-项集的支持度，见表2。

表2 1-项集和2-项集的支持度计数

item	A	B	C	D	E	F	G
A	7	3	3	5	4	2	1
B		5	1	5	3	2	1
C			4	3	2	1	1
D				8	4	2	1
E					5	1	2
F						3	0
G							2

2)FP-Growth算法根据剪枝定理1删去所有非频繁模式，2FP-Growth算法根据剪枝定理2删去了一定不会产生大于等于频繁2-项集的频繁模式。对于表1所示数据集 T 依据表2的统计结果，设 min_sup=2，根据剪枝定理1删去了项 G(非频繁模式)，根据剪枝定理2删去了项 F(所有包含 F 的2-项集均非频繁)，将剩余的频繁模式按支持度非升序排列，得到 $I'=\{D,A,B,E,C\}$。

3)FP-Growth算法中没有2-项集对FP树的剪枝作用，这样由于剪枝不充分对FP树的规模控制较低，反而增加了后续遍历FP树及构造条件FP树的代价。由于2FP-Growth算法在第一次扫描数据集后得到了所有频繁2-项集，在第二次扫描数据集时只需挖掘频繁 k-项集($k\geqslant 3$)，因此，以频繁2-项集为根结点，根据剪枝定理2，若某2-项集 X 非频繁，无须建立以 X 为根结点的2FP树。例如，2-项集 $\{E,C\}$ 非频繁，则所有以 $\{E,C\}$ 为前缀的项集均非频繁，则2FP森林中没有以 $\{E,C\}$ 为根结点的树(见图1)。

4)根据剪枝定理3，对于模式集 I'，设 $\{\gamma\}$ 是 I' 的最后项，则不存在包含 $\{\gamma\}$ 为根结点的2FP树。例如 $I'=\{D,A,B,E,C\}$，则2FP森林无须建立以 $\{D,C\}$、$\{A,C\}$、$\{B,C\}$ 和 $\{E,C\}$ 为根结点的2FP树(见图1)。

5)根据剪枝定理2，在构建2FP树时剪掉所有非潜在频繁3-项集对应的项。例如，对事务 $t_1=\{D,A,B,E,C\}$ 构造以 $\{A,B\}$ 为根结点的2FP树时，对于项 C，由于2-项集 $\{B,C\}$ 非频繁，则直接剪掉项 $\{C\}$，如图2所示。

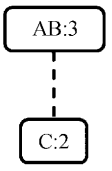
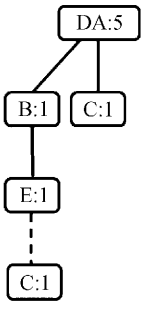
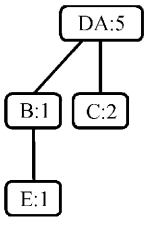


图2 剪枝示例1

在构建2FP树时保证路径上的所有结点均是频繁2-项集。例如，对于事务 $t_1=\{D,A,B,E,C\}$ 构造以 $\{D,A\}$ 为根结点的2FP树时，对于项 C，由于2-项集 $\{B,C\}$ 非频繁，则将 $\{C\}$ 作为根结点 $\{D,A\}$ 的孩子，剪枝前后对比如图3所示。



(a) 剪枝前



(b) 剪枝后的合并现象

图3 剪枝前后对比

将结点 $\{C\}$ 从路径 {DABEC} 剪掉不仅减少了路径长度，而且还可以与其他事务的相同前缀路径进行合并，例如，对于事务 $t_9=\{D,A,C\}$，结点 $\{C\}$ 作为根结点 $\{D,A\}$ 的孩子，可以和 t_1 路径上的结点进行合并。

6)根据剪枝定理3，对于每一个事务在构建

2FP 树时，并不需要组合最后一项，减少了组合次数，从而提高了构建 2FP 森林的时间性能。例如，对于事务 $t_5=\{D,A,B,E\}$，无须更新以 $\{D,E\}$、$\{A,E\}$和$\{B,E\}$为根结点的 2FP 树。

1.2　Map/Reduce 模式

Map/Reduce 模式是由 Google 公司基于 Hadoop 平台下提出的编程模式，该模式的主要思想是：输入一个＜key，value＞的输入键值对，然后产生一个＜key，value＞的结果键值对。这个过程需要定义 Map 和 Reduce 函数，其中 Map 函数用来对输入的键值对进行分片，然后产生中间键值对。Reduce 函数是将相同键值对进行组合生成最终结果。

2　改进着眼点

2.1　基于 Map/Reduce 下并行 2FP-Growth 算法

2FP-Growth 算法在串行上体现出了优势，但是当数据量过大时，2FP-Grwoth 算法仍然是不能实现的。因此，文中提出并行 2FP-Growth 算法，解决 2FP-Gtowth 在大数据下进行频繁模式挖掘不能实现的问题。此算法在 MapReduce 编程模式下，对 2FP-Grwoth 算法进行并行化挖掘。算法主要分为 3 个步骤：

1）统计频繁 1-项集与频繁 2-项集。计算数据集中 1-频繁项集和 2-频繁项集的支持度计数，运行一个统计支持度计数的 Map/Reduce 工程，并将结果保存到分布式缓存中。

2）建立 2FP 树，获取局部频繁项集。这一步是整个并行挖掘算法的重要步骤，该过程设置一个 Map/Reduce 工程。其中，在 Mapper 函数中构造局部 2FP 森林，并对其挖掘得到局部频繁项集 $L2FPSet_i$。Reducer 函数中，将会对所有 $L2FPSet_i$ 进行合并操作，这样将会得到全局频繁项集 GFPSet，并将剩下的不确定是否为全局频繁项集集合中的元素保存到分布式文件中。

3）对存放的候选全局频繁项集并行统计其支持度计数。设置一个 Map/Reduce 工程统计步骤 2）中存放在系统分布式文件中的候选频繁项集的支持度计数，将满足最小支持度计数的频繁项集加入到全局频繁集中。

最后将 2）与 3）所得到的结果合并生成的频繁项集就是整个数据集的全部全局频繁项集。

2.2　统计频繁 1-项集与频繁 2-项集

1-项集和 2-项集的求解过程利用 Map/Reduce来统计关键词出现的次数的应用，可以很容易实现。其伪代码实现如下：

Mapper 过程：

```
map (key, value) {value 为事务 ti
1.for each ai ∈ti do
2.  output＜ai, 1＞;
3.end
4.}
```

Reduce 过程：

```
Reduce(key,value){//key 是 1-项集与 2-项集，value 是其支持度计数列表
1.C=0;
2.For each vi in value do
3.  C+=vi;
4.End
5.If C≥minsup then
6.  Output＜key,C＞;//输出该 1-项集频繁集和支持度计数
7.End
8.}
```

2.3　建立 2FP 树，获取局部频繁项集

在完成 2-项集过程后，下面的任务就是建立 2FP 树，并且对 2FP 树进行挖掘，得到局部的频繁项集。该过程是由另一个 Map/Reduce 实现。Map 过程首先构造并挖掘局部 2FP 树，将挖掘得到局部频繁集保存在 L2FPSet 中。Reduce 过程是用来把存到 L2FPSet 中的局部频繁项集合并，并且对其进行支持度计数，将所有合并后大于等于 minsup 的项集输出，支持度小于 minsup 的项集将会写入分布式文件，供进一步使用。伪代码如下：

Mapper 过程：

```
Map(key,value){ value 为事务 ti
1.insert_2FP(2FPT,ti); //针对 ti 更新局部 2FP 树
2.}
Createup(){
1.local2FPGrowth(2FPT,L2FPSet);
2.For each lfp in LFPSet do
3.Out＜lfp,sup(sfp)＞;
4.End
5.}
```

Reducer 过程：

```
Reduce(key,value){//key 项集，value 是其支持度计数列表
1.C=0;
2.For each vi in value do
```

3. C+=v_i;

4.End

5.If C≥minsup then

6. Output<key,C>;//输出该项集频繁集和支持度计数

7.else

8. Write key into a distribute file; //不确定是否为全局频繁项集,则写入分布式文件

9.end

10.}

2.4 并行计算部分候选全局频繁项集的支持度计数

对于步骤2)写入到分布式文件中的项目集,因为不能判断是否为全局频繁项集,所以要多建立一次Map/Reduce过程来计算这些候选项集的支持度计数,并且判断是否为全局频繁项集。Mapper过程中,readset()函数主要是为了读取这些项集,map函数则是统计支持度计数。伪代码如下:

Mapper过程:

Readset(){

1.LFPSets = loadLFP(); //读取部分频繁项集;

2.}

Map(key,value){ //value为事务 t_i

1.for each lfp in LFPSets do

2. If lfp in value then

3. Output<lfp,1>;

4.End

5.}

Reducer过程:

Reduce(key,value){ //key为全局候选项集,value为其支持度计数列表

1.C =0;

2.For each v_i invalue do

3. C+= v_i;

4.End

5.If C ≥ minsup then

6. Output<key,C>;

7.End

8.}

3 实验结果与分析

为了验证算法的正确性和高效性,在ubuntu16.04操作系统、主频2.5 GHz、内存4 G,使用基于Map/Reduce模型的Hadoop1.2.1作为平台搭建3台服务器实验集群,对数据集进行了如下3个实验。

实验1:在数据集mushroom上验证基于Hadoop的2FP-Growth算法的正确性,实验结果见表3。

表3 在mushroom下挖掘结果

min_sup	频繁项集个数			
	1-items	2-items	3-items	≥4-items
0.02	89	2004	13 496	6 780 018
0.04	79	1 458	8 566	1 796 526
0.08	58	880	4 524	385 819
0.1	56	763	3 992	347 087
0.2	43	376	1 423	49 912

实验1结果表明,基于Hadoop的2FP-Growth算法的频繁项集挖掘结果与FP-Growth算法的挖掘结果完全一致(误差小于1%),表明并行2FP-Growth算法的正确性。

实验2:在数据集T10I4D100K上考察在相同支持度阈值下数据规模对算法效率的影响,实验结果如图4所示。

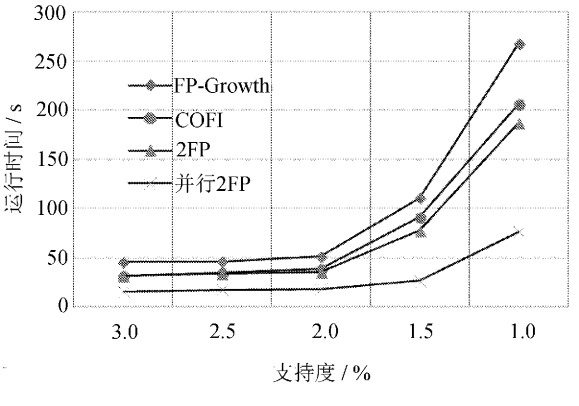


图4 算法运行时间对比

实验2结果表明,在数据集T10I4D100K上基于Hadoop的2FP-Growth算法在时间消耗上明显小于FP-Growth、2FP-Grwoth以及COFI算法,说明基于Hadoop的2FP-Growth算法的高效性。

实验3:在最小支持度5%下,不同大规模数据集下算法运行结果的比较见表4。

表 4　在大规模数据集下运行结果

数据集	数据大小/KB	运行时间/s			
		FP-Growth	2FP	COFI	并行 2FP
Pumsb_star.dat	11 028	内存溢出	202.75	内存溢出	98.27
Kosarak.dat	31 279	内存溢出	351.15	内存溢出	150.02
Accidents.dat	34 678	内存溢出	内存溢出	内存溢出	321.85
Webdocs.dat	1 448 580	内存溢出	内存溢出	内存溢出	5 346.35

实验 3 结果表明，当输入的数据集规模较大时，FP-Growth 算法及其改进算法会造成内存溢出，当 Hadoop 集群下建立 3 个计算机节点，表中的数据集将会解决内存溢出问题，因此，文中提出的基于 Hadoop 的 2FP-Growth 算法根据数据集的规模进行调整节点数是有效的。

4　结　语

提出基于 Hadoop 的 2FP-Growth 算法，并且在 Hadoop 平台下实现，取得了较好的实验结果。通过与 FP-Growth、COFI 以及 2FP-Grwoth 算法在数据集 Mmushroom 以及 T10I4D100K 比较正确性和挖掘效率可以看出，基于 Hadoop 的 2FP-Growth 算法明显高于 FP-Growth、COFI 以及 2FP-Grwoth。实验结果表明，基于 Hadoop 的 2FP-Growth 算法在数据规模较大调整计算机节点数有较好的高效性、正确性以及算法的应用价值。

参考文献：

[1] Agrawal R, Srikant. Fast algorithms for mining association rules[C]//Proceedings of the 20th International Conference on Very Large DataBases. Santiago: Chile,1994:487-499.

[2] Zhou L, Zhong Z, Chang J, et al. Balanced parallel FP-growth with MapReduce [C]//Information Computing and Telecommunications.2011:243-246.

[3] Mao W, Guo W. An improved association rules mining algorithm based on power set and hadoop [C]//International Conference on Information Science and Cloud Computing Companion. [S. l.]: IEEE,2014:236-241.

[4] 刘祥哲，刘培玉，任敏，等.基于负载均衡和冗余剪枝的并行 FP-Growth 算法[J].数据采集与处理，2016，31(1)：223-230.

[5] 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.

[6] 厍向阳，张玲.基于 Hadoop 的 FP-Growth 关联规则并行改进算法[J].计算机应用研究，2018，35(1)：109-112.

[7] 施亮，钱雪忠.基于 Hadoop 的并行 FP-Growth 算法的研究与实现[J].微电子学与计算机，2015，32(4)：150-154.

[8] 王建明，袁伟.基于节点表的 FP-Growth 算法改进[J].计算机工程与设计，2018，39(1)：140-145.

[9] 张稳，罗可.一种基于 Spark 框架的并行 FP-Growth 挖掘算法[J].计算机工程与科学，2017，39(8)：1403-1409.

[10] 邵梁，何星舟，尚俊娜.基于 Spark 框架的 FP-Growth 大数据频繁项集挖掘算法[J].计算机应用研究，2018(10)：1-6.

[11] 王红梅，李芬田，王泽儒.基于滑动窗口数据流频繁集挖掘模型综述[J].长春工业大学学报，2017，38(5)：484-490.