·云计算与大数据专题· 文章编号:1000-3428(2019)03-0032-04 文献标志码:A 中图分类号:TP391

基于负载均衡的并行 FP-Growth 算法

高 权,万晓冬

(南京航空航天大学 自动化学院,南京 211106)

摘 要: 针对 FP-Growth 算法查找操作时间复杂度较高的问题,提出一种新的算法 LBPFP。在 PFP 算法基础上,将哈希表加入链头表以实现项地址的快速访问,并设计基于前缀长度的计算量模型,优化并行流程,提升算法的执行效率。在 webdocs.dat 数据库上进行对比实验,结果表明,LBPFP 算法比 PFP、HPFP、DPFP 算法具有更高的频繁项集挖掘效率。

关键词: Spark 平台; 频繁模式增长; 并行; 负载均衡; 链头表; 计算量模型

中文引用格式: 高权,万晓冬.基于负载均衡的并行 FP-Growth 算法[J].计算机工程,2019,45(3):32-35,40.

英文引用格式: GAO Quan, WAN Xiaodong. Parallel FP-growth algorithm based on load balance[J]. Computer Engineering,2019,45(3):32-35,40.

Parallel FP-Growth Algorithm Based on Load Balance

GAO Quan,WAN Xiaodong

(College of Automation Engineering,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China)

【Abstract】 Aiming at the problem that the lookup operation of FP-Growth algorithm has a high time complexity, this paper proposes a new algorithm named LBPFP. The algorithm is based on PFP algorithm, which is added a hash table to the head table to achieve fast access to item and is designed a workload model based on the prefix length to optimize the parallel process and improve the efficiency of the algorithm. The comparison experiments in the webdocs.dat database show that the LBPFP algorithm has better performance than the PFP, HPFP and DPFP algorithms.

【Key words】 Spark platform; FP-Growth; parallel; load balance; head table; workload model

DOI: 10.19678/j.issn.1000-3428.0049606

0 概述

随着数据规模的不断增大,串行算法在时间和空间上的开销迅速增长。面对大规模数据带来的处理速度过慢和存储容量不足等问题,基于集群的算法并行化方案是有效的解决方案。文献[1]提出 PFP 算法,在 MapReduce[2] 框架下实现 FP-Growth 算法[3]的并行化,成为一种典型的并行化实现方案,也是 Hadoop、Spark 平台的算法库成员。随后,众多研究学者在 PFP 算法的基础上深入研究。文献[4]提出用布尔型矩阵减少事务数据库的扫描次数,从而优化了 PFP 算法的执行流程。文献[5]使用二维表压缩记录频繁项的支持度,避免产生大量的条件 FP 树。文献[6]根据项合并策略减小搜索空间规模,优化 FP-Growth 算法的挖掘效率,并利用 Hadoop 平台的 MapReduce 编程模式实现并行化,进一步提升大数据环境下的挖掘效率。文献[7]针对事务共享前缀过长的特殊情况拆分模式树,优化 FP-Growth 算法,并在 Spark 平台上完成并行化。文献[8]提升 PFP 算法挖掘最大频繁项集的效率。文献[9]改进 PFP 算法,并在数据库增量更新下挖掘关联规则。文献[10]在 PFP 算法中引入负载均衡策略,根据计算量模型估算负载大小再分组,大幅提升了并行执行效率。文献[11]在考虑负载均衡的同时,对 FP-Growth 算法使用剪枝策略,合并多路径中的同名项,从而减少挖掘 FP 树的递归次数。

本文以 PFP 算法为基础,针对 FP-Growth 算法中原始链头表的数据结构对排序、查找等操作产生的较高时间复杂度问题,以及现有计算量模型估算不准确带来的负载不均问题进行优化,提出一种并行算法 LBPFP,以提升并行效率,降低时间成本。

1 相关工作

PFP 算法的执行流程如图 1 所示,主要由 3 个 MapReduce 任务完成。

基金项目: 国家部委基金。

作者简介: 高 权(1990—),男,硕士研究生,主研方向为大数据分析、数据挖掘;万晓冬,副研究员。

收稿日期: 2017-12-07 **修回日期:** 2018-01-08 **E-mail:** gaoquan_gq@163.com

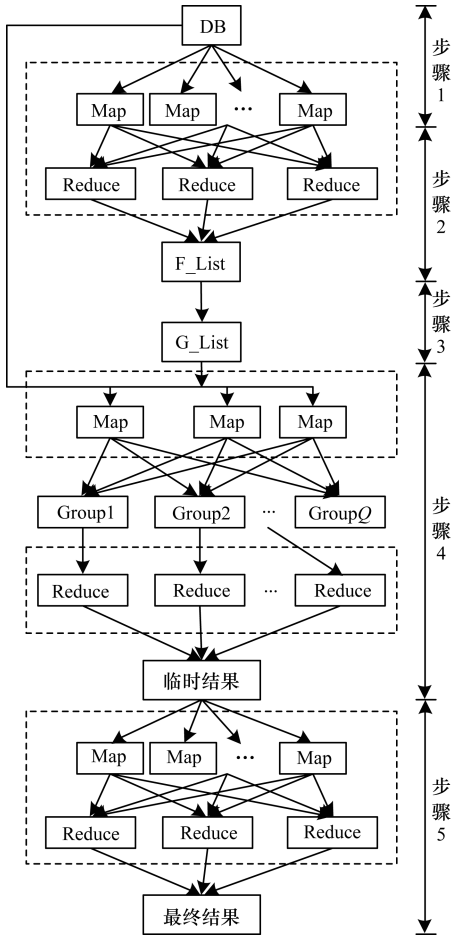

图 1　PFP 算法的并行流程

PFP 算法详细描述如下:

步骤 1　利用 Map 任务将事务数据集 DB 划分到不同节点上,并将事务 T_i 映射成键值对(a_j,1),其中 a_j 属于 T_i。

步骤 2　利用 Reduce 任务统计各项的支持数,生成频繁项目表 F_List。

步骤 3　将 F_List 中的项分组,结果存储在 G_List中,由单节点完成。

步骤 4　根据 G_List,用 Map 任务生成组间独立的事务,然后由 Reduce 任务对组内事务进行 FP-Growth 挖掘,返回局部支持数最高的前 K 项模式。

步骤 5　聚合步骤 4 的计算结果,先用 Map 任务合并相同键值,再用 Reduce 任务计算全局支持数最高的前 K 项模式。

然而,PFP 算法仅采用了等量分组策略,并没有考虑不同频繁项会产生不同计算量的问题[12]。在数据量增大的情况下,不同分组之间的负载差异会增大,出现部分节点完成少量计算任务后空闲等待的情况,明显降低了整个并行算法的性能。而文献[5-6]给出一种主流的负载均衡策略,用以估算每个频繁项的负载大小,然后在分组中利用贪心策略,将负载大的项放到负载总和最小的组以实现负载均衡。计算量模型计算公式如下[13]:

$$L_i = \mathrm{lb}(P_i) \tag{1}$$

其中,项目 i 对应的计算量为 L_i,在链头表中的位置为 P_i。这种计算量模型也被文献[14-15]使用到各自的负载均衡策略中。

2　LBPFP 算法

本文提出一种 LBPFP 算法,通过优化 FP-Growth 算法中链头表的数据结构,降低时间复杂度,改进负载均衡策略中的计算量模型,缩小分组带来的计算时间差异,提升并行执行的效率,并给出 Spark 下优化算法 LBPFP 的实现过程。

2.1　链头表数据结构优化

根据表 1 给出的事务数据库,构建优化前后的链头表,如图 2 所示。

表 1　PFP 算法的应用样例

事务 T	排序后的事务	独立的事务片段
r z h k p	z r	r:z
z y x w v u t s	z x s t y	y:z x s t t:z x s s:z x x:z
s x o n r	x r s	s:x r r:x
x z y m t s q e	z x s t y	y:z x s t t:z x s s:z x x:z
z	z	∅
x z y r q t p	z x r t y	y:z x r t t:z x r r:z x x:z

原始链头表

数组	
项	链头节点
z	Head-link_z
x	Head-link_x
r	Head-link_r
s	Head-link_s
t	Head-link_t
y	Head-link_y

优化后的链头表

数组		哈希表	
项	标志位	键	值
z	F	r	(2,Head-link_r)
x	F	s	(3,Head-link_s)
r	F	t	(4,Head-link_t)
s	F	x	(1,Head-link_x)
t	F	y	(5,Head-link_y)
y	F	z	(0,Head-link_z)

图 2　链头表数据结构示意图

原始链头表采用数组结构,按支持度由高到低的顺序存储频繁项和节点链的头节点。假设链头表含有 m 个频繁项,则一个含有 k 个元素的事务在排序中需要比较 $k \cdot m$ 次,在插入到 FP 树中需要比较 $m \cdot m$ 次,时间复杂度为 $O((k+m) \cdot m) = O(m^2)$。

优化后的链头表包含一个数组和一张哈希表。其中,数组按支持度由高到低的顺序存储由频繁项

和标志位组成的二元组;哈希表存储键值对,键为频繁项,值为二元组,其中第 1 个元素是频繁项在数组中的位置,第 2 个元素是节点链的头节点。此时,排序逻辑为:根据项目名查找哈希表,获取数组的下标值,并将数组中对应元组中的标志位置为 True,再获取数组中标志位为 True 的项,并将标志位复位为 False。排序的时间复杂度为 $O(k)$,插入 FP 树的时间复杂度为 $O(m)$,总的时间复杂度为 $O(k+m)=O(m)$,较优化前降低了一个数量级。

2.2 计算量模型优化

式(1)给出的计算量模型仅与链头表中项目的位置有关,并不能反映数据自身的特征。如图 3 所示,在 FP 树中挖掘以 r 为模式后缀的频繁项集,产生的负载量应比 s 大,而现有模型估算的 r 计算量却小于 s,由此进行的分组仍存在一定程度的负载不均衡现象。

典型分组

项	lb P_i	gid
z	lb 1	2
x	lb 2	1
r	lb 3	1
s	lb 4	2
t	lb 5	2
y	lb 6	1

优化分组

项	前缀长度	gid
z	1	1
x	3	2
r	7	1
s	6	2
t	8	2
y	10	1

图 3 分组示意图

为此,本文提出基于前缀长度的计算量模型,同时反映项目在 FP 树中由深度与多路径产生的计算量。

考虑前缀长度作为后验知识带来的计算任务,本文对 PFP 算法流程作出改进,以保证事务数据库的扫描次数不变,提升并行化效率。具体的步骤如下:

1)对排序后的事务进行 flatMap 操作,生成 <key:项,value:事务片段 >,再通过 groupByKey( ) 生成独立的条件事务集,用于构建每一项的条件 FP 树,计算前缀长度,并将前缀长度和条件 FP 树保存到项对应的键值对中。

2)根据链头表中从下向上的顺序逐项对前缀长度进行分组计算:每次将新项目的前缀长度累加到总长度最小的组中,同时赋予该项相应的组号 gid,直到分组结束。

3)根据哈希结构 G_List 中保存的(key:组号,value:划分到该组的各项对应的条件 FP 树),调用 Growth( ) 函数挖掘以各项为模式后缀的频繁项集。

在 LBPFP 算法中,只扫描原始事务数据库2 次,直接由以项为键、事务片段为值的键值对构建各频繁项的条件 FP 树,省去构造 FP 树的过程。同时,前缀长度更加准确地衡量出负载的大小,通过均衡策略分组后,可降低各节点之间计算时间的差距。LBPFP 算法在 Spark 下的执行过程如算法 1 所示。

算法 1 LBPFP 算法

输入 事务数据库 DB 的 HDFS 存储路径 filePath,min_sup

输出 全局频繁项集 all

```
1. val sc = Initialize SparkContext;
2. val inputRDD = sc.textFile(filePath);
3. val FRDD = inputRDD.mapPatition(p = > {
val F_list = getItem(p)
/* 统计频繁项的频数,生成频繁项目表 */
sort(p)/* 根据项头表对原始事务中的项排序和剪枝 */
});
4. val rdd = FRDD.mapPartition(p = > {
val result = List[String,(Int,Null)]()
/* <项,(前缀长度,条件 fp 树根节点) > */
getReduce(p)/* 相同项的事务片段聚集在一起; */
for each ai in F_list do
build-conditional-fp-tree(root_ai,p)
/* 构建每一项的条件 fp 树 */
count(length_ai)  /* 计算项的前缀长度 */
end
});
5. G_list = group(rdd);  /* 分组 */
6. val loc = rdd.mapPartition(p = > {  /* p 代表一个数据块 */
getFrequent(p,G_list,min_sup);
/* 组内逐项对条件 fp 树挖掘频繁项集 */
});
7. val all = locToAll(loc);
/* 合并局部频繁项集,得到全局频繁项集 */
8. return all;
```

3 实验与结果分析

为验证 LBPFP 算法的优化效果,本文用4 台PC 机搭建 Spark 集群环境,具体参数如表 2 所示。实验使用公开数据集 webdocs. dat[16],其基本特征如表 3 所示。

表 2 集群环境参数

序号	机器名	运行进程	核数/内存	环境参数
1	cmaster0	Master/Worker	4 核/8 GB	CentOS 7.1 Spark-2.1.0 Scala-2.1.1
2	cslave0	Worker	2 核/4 GB	
3	cslave1	Worker	2 核/4 GB	
4	cslave2	Worker	2 核/4 GB	

表 3 webdocs. dat 数据集的基本描述

数据集	数据量/GB	记录数	基本项数	事务最大长度
webdocs. dat	1.48	1 692 082	5 267 656	71 472

本文实验以 IntelliJ IDEA 为开发环境,在 Spark 集群下实现 PFP 算法、HPFP 算法(在 PFP 算法基础上只优化链头表)、DPFP 算法(在 HPFP 算法中加入典型的负载均衡策略)以及本文提出的 LBPFP 算法(优化 DPFP 算法中负载均衡策略使用的计算量模型),并从数据规模、支持度、集群节点个数角度比较 4 种算法的挖掘效率,取 5 次实验数据的平均值作为最终结果。

3.1　数据规模对算法性能的影响

从 webdocs.dat 中随机抽取不同规模的样本数据,构建实验数据集 data1、data2、data3、data4,依次包含 40 万、80 万、120 万、160 万条记录,取最小支持度为 0.6,分别使用 4 种算法进行挖掘分析,结果如图 4 所示。

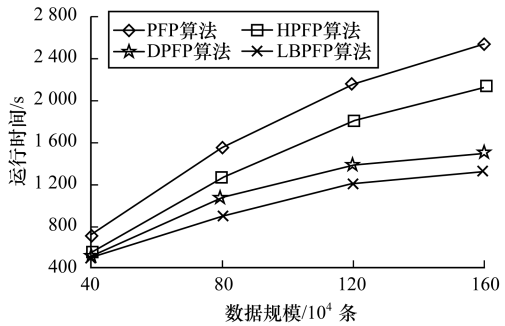

图 4　不同数据规模的实验结果

由图 4 可知,当数据量较小时,4 种算法的运行时间接近。随着数据量的增大,HPFP、DPFP、LBPFP 3 种算法相对于 PFP 算法都具有明显的优化效果,说明优化链头表与采用负载均衡策略都对算法性能有较大的提升。DPFP 算法与 LBPFP 算法在较大数据规模下表现出更高的算法性能,说明负载均衡策略对并行算法执行效率的提升有重要作用。LBPFP 算法相对 DPFP 算法表现出更好的挖掘效率,说明优化计算量模型提升了算法性能。

3.2　支持度对算法性能的影响

分别取最小支持度为 0.2、0.4、0.6、0.8,对所有数据进行挖掘分析,结果如图 5 所示。

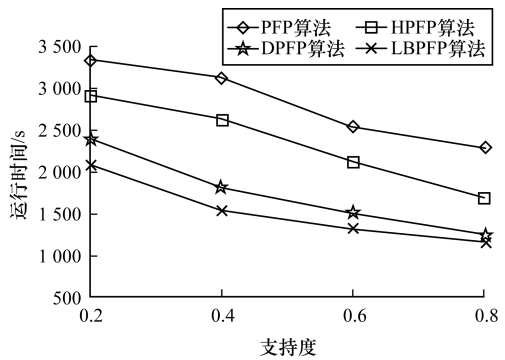

图 5　不同支持度的实验结果

由图 5 可知,随着支持度的增大,需要处理的挖掘任务必然减少,4 种算法的运行时间也都必然减小,但采用负载均衡策略的 DPFP 算法、LBPFP 算法能在数据量较大(即支持度较小)的情况下保持更好的性能。在 4 种算法的相互比较中可以发现,DPFP 算法相对于 HPFP 算法的提升效果整体上高于 HPFP 算法相对于 PFP 算法的提升效果,且 LBPFP 算法的性能相对 PFP 算法至少提升了 1 倍,说明优化的负载均衡策略有效提升了算法处理海量数据的能力。

3.3　节点个数对算法性能的影响

调整节点数分别为 1、2、3、4,对所有数据进行挖掘分析,结果如图 6 所示。

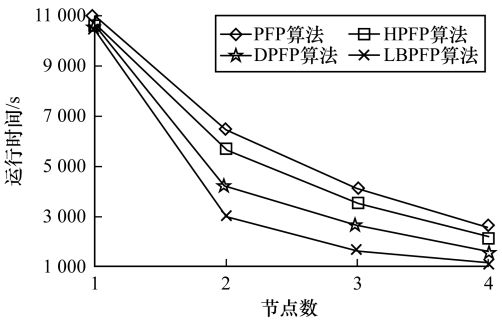

图 6　不同节点数的实验结果

由图 6 可知,随着节点数的增加,4 种算法的运行时间大幅减少,说明并行算法能有效地处理大规模数据。其中,LBPFP 算法性能明显优于其他 3 种算法,表明本文优化的负载均衡算法能在多个节点间更合理地分配计算任务,提升了并行执行的效率。

4　结束语

本文提出一种 LBPFP 算法,通过在链头表中引入哈希表提升项的访问速度,并针对负载均衡策略中负载估算不准带来的分组不均衡问题,建立基于前缀长度的计算量模型,改进并行化流程,提升算法性能。通过对比实验,从数据规模、支持度大小、节点数量的角度出发,证明 LBPFP 算法极大地提高了频繁项集的挖掘效率。下一步将考虑在 FP 树中引入剪枝策略、降低节点通信量,提升算法效率。

参考文献

[1] LI H, WANG Y, ZHANG D, et al. PFP: Parallel FP-Growth for query recommendation[C]//Proceedings of 2008 ACM Conference on Recom mended Systems. New York, USA: ACM Press, 2008: 125-137.

[2] LEE K H, LEE Y J, CHOI H, et al. Parallel data processing with MapReduce: A survey[J]. ACM SIGMOD Record, 2012, 40(4): 11-20.

[3] HAN J, PEI J, YIN Y. Mining frequent patterns without candidate generation[C]//Proceedings of 2000 ACM SIGMOD International Conference on Management of Date. New York, USA: ACM Press, 2000: 1-12.

(下转第 40 页)

[3] ATENIESE G,BURNS R,CURTMOLA R,et al. Provable data possession at untrusted stores[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security. New York,USA:ACM Press,2007:598-609.
[4] ATENIESE G,KAMARA S,KATZ J. Proofs of storage from homomorphic identification protocols[C]//Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security. New York,USA:ACM Press,2009:319-333.
[5] ERWAY C C,PAPAMANTHOU C,TAMASSIA R. Dynamic provable data possession[J]. ACM Transactions on Information and System Security,2009,17(4):213-222.
[6] JUELS A,KALISKI B S. Pors:proofs of retrievability for large files[C]//Proceedings of ACM Conference on Computer and Communications Security. New York,USA:ACM Press,2007:584-597.
[7] WANG Q,WANG C,LI J,et al. Enabling public verifiability and data dynamics for storage security in cloud computing[C]//Proceedings of European Conference on Research in Computer Security. Berlin,Germany:Springer,2009:355-370.
[8] ZHU Y,HU H,AHN G J,et al. Efficient audit service outsourcing for data integrity in clouds[J]. Journal of Systems and Software,2012,85(5):1083-1095.
[9] ZHU Y,HU H,AHN G J,et al. Cooperative provable data possession for integrity verification in multicloud storage[J]. IEEE Transactions on Parallel and Distributed Systems,2012,23(12):2231-2244.
[10] 谭霜,贾焰,韩伟红. 云存储中的数据完整性证明研究及进展[J]. 计算机学报,2015,38(1):164-177.
[11] 周锐,王晓明. 基于同态哈希函数的云数据完整性验证算法[J]. 计算机工程,2014,40(6):64-69.
[12] 杨小东,刘婷婷,杨平,等. 基于代理重签名的云端数据完整性验证方案[J]. 计算机工程,2018,44(9):130-135.
[13] WANG H,ZHU L,WANG F,et al. An efficient provable data possession based on elliptic curves in cloud storage[J]. International Journal of Security and Its Applications,2014,8(5):97-108.
[14] YANG K,JIA X. An efficient and secure dynamic auditing protocol for data storage in cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems,2013,24(9):1717-1726.
[15] WANG C,WANG Q,REN K,et al. Privacy-preserving public auditing for data storage security in cloud computing[EB/OL]. [2017-10-10]. http://www.aensiweb.net/AENSIWEB/anas/anas/2016/Special/118-121.pdf.
[16] 何凯,黄传河,王小毛,等. 云存储中数据完整性的聚合盲审计方法[J]. 通信学报,2015,36(10):119-132.
[17] ATENIESE G,BURNS R,CURTMOLA R,et al. Provable data possession at untrusted stores[C]//Proceedings of ACM Conference on Computer and Communications Security. New York,USA:ACM Press,2007:598-609.

编辑 吴云芳

(上接第 35 页)

[4] 陈兴蜀,张帅,童浩,等. 基于布尔矩阵和 MapReduce 的 FP-Growth 算法[J]. 华南理工大学学报(自然科学版),2014(1):135-141.
[5] WEI F,XIANG L. Improved frequent pattern mining algorithm based on FP-tree[EB/OL]. [2017-11-11]. http://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=264,id.37.2015.
[6] 马月坤,刘鹏飞,张振友,等. 改进的 FP-Growth 算法及其分布式并行实现[J]. 哈尔滨理工大学学报,2016,21(2):20-27.
[7] 方向,张功萱. 基于 Spark 的 PFP-Growth 并行算法优化实现[J]. 现代电子技术,2016,39(8):9-13.
[8] 宁慧,王素红,崔立刚,等. 基于改进的 FP-tree 最大频繁模式挖掘算法[J]. 应用科技,2016,43(2):37-43.
[9] 程广,王晓峰. 基于 MapReduce 的并行关联规则增量更新算法[J]. 计算机工程,2016,42(2):21-25.
[10] ZHOU L,ZHONG Z,CHANG J,et al. Balanced parallel FP-Growth with MapReduce[C]//Proceedings of Information Computing and Telecommunications. Washington D.C.,USA:IEEE Press,2011:243-246.
[11] YANG Q,DU F,ZHU X,et al. Improved balanced parallel FP-Growth with MapReduce[C]//Proceedings of 2016 Joint International Conference on Artificial Intelligence and Computer Engineering and International Conference on Network and Communication Security. Lancaster,USA:Destech Publications,2016:1-5.
[12] 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):1-6.
[13] PRAMUDIONO I,KITSUREGAWA M. Parallel FP-Growth on PC cluster[C]//Proceedings of Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Berlin,Germany:Springer,2003:467-473.
[14] YAN X,ZHANG J,XUN Y,et al. A parallel algorithm for mining constrained frequent patterns using MapReduce[J]. Soft Computing,2015,21(9):1-13.
[15] 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.
[16] Frequent itemset ming dataset repository[EB/OL]. [2017-11-11]. http://fimi.ua.ac.be/data.

编辑 刘盛龄