

基于 Hadoop 的并行 FP-Growth 算法的研究与实现

施 亮, 钱雪忠

(江南大学 物联网工程学院, 江苏 无锡 214122)

摘 要: 在 PFP(Parallel FP-Growth)算法的基础上提出了一种负载均衡并行的挖掘算法 LBPFP(Load-Balanced Parallel FP-Growth)算法,该算法在 Hadoop 框架实现并行计算的同时,在数据分发中利用负载均衡策略,使主节点均衡地向子节点分配数据.除此之外,在子节点进行数据处理的过程中利用剪枝策略,减少数据的处理量,该算法在提高并行计算能力的同时也大大缩小了数据的处理量.最后,通过实验分析表明该算法在大数据的处理中具有较好的效果,证实了该算法的可行性.

关键词: 关联规则;数据挖掘;负载均衡;FP-Growth;并行计算

中图分类号: TP301.6

文献标识码: A

文章编号: 1000-7180(2015)04-0150-05

Research and Implementation of Parallel FP-Growth Algorithm Based on Hadoop

SHI Liang, QIAN Xue-zhong

(College of The Internet of Things Engineering, Jiangnan University, Wuxi 214122, China)

Abstract: This paper, we propose a load-balanced and parallel FP-Growth algorithm based on Map/Reduce, which evenly parallelizes FP-Growth in the MapReduce approach. LBPFP(Load-Balanced Parallel FP-Growth) adds into PFP(Parallel FP-Growth) the load balance feature and the effectively pruning strategy, which improves parallelization and thereby improves performance. Finally, the experimental result shows the algorithm has good effect in the large data processing. It is proved the feasibility of the algorithm.

Key words: association rules; data mining; load-balanced; FP-Growth; parallel computing

1 引言

频繁项集的挖掘(FIM)是关联规则分析、数据统计、因果关系及序列模式及其他重要的挖掘任务中相关重要的一部分.传统的频繁项集挖掘算法,如 Apriori^[1] FP-Growth 等,在处理大数据的挖掘任务中效率极其低下.为此,人们提出了并行计算以此来减少每台计算机的内存和 CPU 消耗.而 Apriori 算法在挖掘任务中需要耗费大量的时间来多次扫描数据库,一些并行算法则存在高通信量的问题,降低了并行计算的性能. PFP^[2] (Parallel FP-Growth)算法是一种利用 Map/Reduce 编程模型来实现挖掘任务

的并行化处理,通过借助 Hadoop 框架^[3]的分布式优势,该算法在并行化计算方面有了较大的提高.但是, PFP 算法在数据分发过程中并没有充分考虑均衡分配的问题,而均衡分配在处理海量数据中却是十分重要.因此本文中提出一种负载均衡地并行的 FP-Growth 算法—LBPFP(Load-Balanced Parallel FP-Growth).该算法首先测算每个挖掘单元的任务负载量,然后均衡的将这些单元分成几组,已达到各组的负载均衡.通过将负载均衡策略应用到 PFP 算法中,有效地改善了各子节点数据分配不均衡的问题,使得数据的并行化效率得以提高,同时在挖掘任务中利用剪枝策略大大降低了数据的处理

收稿日期: 2014-07-02; 修回日期: 2014-08-28

基金项目: 国家自然科学基金资助项目(61103129, 61202312); 江苏省科技支撑计划资助项目(BE2009009)

量,有效地提高了算法的处理效率.

2 FP-Growth 算法

2.1 相关工作介绍

最早的一些并行 FP-Growth 算法是在共享内存的环境下工作.但是,当数据量较大时,这些算法就会面临内存不够用的问题,而一些分布式算法通过运行在一个集群中来解决内存不足的问题,但是这些算法也同样面临一些十分棘手的问题,如高通信消耗高内存消耗,高读写等.除此之外,这些分布式算法也存在难以扩展及难于维护的问题.

PFP 算法通过 Hadoop 框架的 Map/Reduce 并行编程模型实现 FP-Growth 算法的并行化^[4],其原理是 PFP 算法将一个挖掘任务分成若干个相对独立的子任务,然后利用 Hadoop 框架将子任务分发给各个子节点进行处理,该算法在数千机器组成的集群中处理数据挖掘任务时性能得到了很大的提升,但是该算法并未充分考虑负载均衡的问题,在处理海量数据时该问题是十分的重要,直接影响算法的整体性能.

2.2 FP-Growth 算法

FP-Growth 算法可以分为两个阶段:模式树(FP-Tree)的构建及频繁模式的挖掘^[5].在构建 FP-Tree 的过程中需要扫描两次事务数据库,第一次扫描事务数据库,统计事务数据库中的 1-频繁项集,并记录各频繁项在事务数据库中出现的次数,根据出现的次数按降序排列同时剔除出现次数少于最小支持度的频繁项,形成 F-List.第二次扫描事务数据库用以构建模式树(FP-Tree).首先,扫描事务数据库中的每条记录,按照 F-List 中各项的顺序,进行降序排列,并删除在 F-List 中不存在的项,然后将其插入到模式树(FP-Tree)中.在 FP-Growth 算法初始阶段需要输入 FP-Tree 及指定最小支持度,然后 FP-Growth 算法根据 F-List 的最后一个频繁项扫描模式树中的每个节点.在扫描各个节点时,FP-Growth 算法不断记录从该节点到根节点的路径上的项集,记录的这些项集便构成以该项为后缀的条件模式基.利用这些条件模式基按照上述步骤重新构成一个子模式树(FP-Tree).然后再在这个子模式树上进行 FP-Growth.这个过程不断递归迭代直到无条件模式基生成为止.

3 负载均衡的并行 FP-Growth 算法

给定一个事务数据库及最小支持度, LBPFP 算法需要两次 Map/Reduce 过程来并行化 FP-Growth 算法.相对于 PFP 算法,有三个主要的不同点:一是在给事务数据库分组阶段应用负载均衡策略,将整个挖掘任务分成若干个相对独立的子任务以提高并行化;二是 LBPFP 算法没有聚合的步骤,该算法意在挖掘出所有的频繁项集;三是在每个子挖掘任务中使用模式树(FP-Tree)剪枝策略,合并符合条件的子节点,减少 FP-Growth 算法的递归迭代次数,加快数据处理的速度.

3.1 数据划分

数据划分就是将整个事务数据库中的数据分成连续的若干数据片段,并将这些数据片段分发给若干个(P)不同的机器储存^[6],在使用 Hadoop 框架的 Map/Reduce 分布式编程模型进行数据处理时,只需要将事务数据库中的数据拷贝到 Hadoop 框架的分布式文件管理系统(DFS)中即可, Hadoop 框架会自动帮我们进行数据划分处理.

3.2 数据的并行统计

在这一阶段使用一个 Map/Reduce 任务来统计整个事务数据库中的 1-频繁项集的出现次数^[7].每个 mapper 实例处理一个事务数据库的数据块. mapper 实例接收的数据是一种键值对的形式,如 $\langle \text{key}, \text{value} = T_i \rangle$, 其中 T_i 是事务数据库(DB)中的每条记录.对于每个频繁项, mapper 实例将处理好的数据是以一种键值对的形式输出,如 $\langle \text{key}' = a_j, \text{value}' = 1 \rangle$.对于每个 mapper 实例的输出,用一个对应的 combiner 实例进行局部的整合,将相同键的值进行求和,通过此步可有效地减少节点之间的数据通信量^[8].在所有的 mapper 实例执行完毕后, Hadoop 框架会将具有相同键的键值对分发给同一个 reducer 实例进行处理, reducer 实例对具有相同键(key)的值进行求和运算,处理结束后同样以键值对的形式输出^[9],如 $\langle \text{key}', \text{sum}(\text{values}') \rangle$.本阶段处理结束后可获得一个以 1-频繁项集按其支持度降序排列的列表, F-List.

3.3 数据的均衡分组

将 F-List 中的频繁项根据均衡分组策略分成 Q 个组,使得数据在分发过程中达到负载均衡的目的.

的,依此来提高整个挖掘过程的并行化.

整个 FP-Growth 的负载量等于在所有频繁模式基上进行 FP-Growth 处理的子 FP-Growth 负载量的总和^[10],因此均衡分组可以分成两个步骤完成,第一步,计算以每个频繁项为后缀的条件模式基上进行 FP-Growth 任务的任务量,第二步,将所有 FP-Growth 任务的总任务量分成若干(Q)组.

3.4 挖掘任务中负载量的评估

由于在这一步骤中并不能获取完备的有效数据来精确地计算每个频繁项的负载量,因此需要做一些相应的评估.在此阶段需要进行两个评估过程.首先,通过统计 FP-Growth 任务在频繁项为后缀的条件模式基上进行的递归迭代量来计算 FP-Growth 任务在频繁项的条件模式基上进行的负载量,其次根据频繁项在 F-List 中的位置来评估该频繁项的最长模式基的长度.根据文献^[11],在频繁项的最长模式基上进行递归迭代量成指数增长,例如,频繁项(i)的负载量(T_i),根据 i 在 F-List 中的位置(L_i),可得如下结果:

$$T_i = \text{Log} L_i$$

首先,把 F-List 中的频繁项按照负载量进行降序排列,构成 L-List.然后利用 L-List 中的前 Q 项初始化 Q 组列表,每一项分配到一组,根据 Q 组中每组含有的频繁项的负载量初始化其所在分组的负载量.最后重复下面两个步骤直到 F-List 中的所有频繁项被分配到相应组中:

(1)将 L-List 中未被分组的第一项分配给负载量最小的组;

(2)将新加频繁项的负载量加到其所在组的负载量中.

本文中,使用最大堆算法来优化这一分组过程.算法 1 通过伪代码展示了均衡分组的相关过程,其中 F[] 表示 F-List, Q 表示分组数, G[] 表示 G-List.

```

Procedure: partition(F[], Q, G[])
if  $Q \geq F.Length$  then
Foreach ai in F do
g = Call CreateGroup();
else
L[] = Sort(F);
maxHeap = Call CreateMaxHeap(Q);
for i = 0 to Q - 1 do
g = Call CreateGroup();

```

```

g.add(L[i]);
maxHeap.add(g);
end
maxHeap.adjust();
for i = Q to L.length - 1 do
maxHeap.getTop().add(L[i]);
maxHeap.adjust();
end
end

```

3.5 FP-Growth 的并行化

本阶段利用一次 Map/Reduce 过程来进行处理.首先,在 Map 阶段,事务数据库中的数据被处理成组内依赖的数据块.然后在 Reduce 阶段每个 reducer 实例根据获取到的数据块构建子模式树(SubFP-Tree),利用 FP-Growth 在构建的子模式树上递归迭代地进行频繁模式的挖掘任务.在 mpper 实例的开始阶段,它会加载在均衡分组阶段生成的 G-List.在本文的实现中, G-List 被构建成一个 Hash 映射表,将其包含的每一项映射到相应的组的编号(group-id)上.在本阶段, mapper 的输入参数是事务数据库中的数据切片,每个数据切片以键值对,如 $\langle \text{key}, \text{Value} = T_i \rangle$, 的形式传递^[12].对于每个 T_i , mapper 都将会按照下面两个步骤进行执行:

(1)删除 T_i 中支持度小于指定最小支持度的项,然后按照降序排列;

(2)对于 T_i 中的每一项,定位该项左边的项集,记录为 L;

(3)输出一个键值对,形如 $\langle \text{key}' = \text{group-id}, \text{value} = \{T_i[1], T_i[2], \dots, T_i[L-1], T_i[L]\}$.

reducer 实例在开始执行时同样会加载 G-List.在这个阶段,每个 reducer 实例会逐个的读取和处理 $\langle \text{key}', \text{Value} = \text{DB}(\text{group-id}) \rangle$ 键值对,每一个 $\text{DB}(\text{group-id})$ 包含了同一组的所有事务数据块^[13].对于每个 $\text{DB}(\text{group-id})$, reducer 实例会递归地构建和挖掘子模式树(SubFP-Tree).为了更进一步地优化 FP-Growth 算法,对 FP-Tree 在递归构建和挖掘中,使用特定的剪枝策略来减低递归次数,提高数据处理速度.

FP-Tree 剪枝:根据由下到上、深度优先的原则^[14],从模式树中的最小子节点开始,对于每条路径中支持度小于最小支持度的子节点;如果它的父节点的兄弟节点或者父节点含有和该子节点相同的

模式,那么就合并它们并且删除该子节点,如果没有就直接删除该节点.按照上述过程处理树中的其他节点,通过此剪枝策略可进一步压缩模式树的规模,有效减少递归次数,加快 FP-Growth 算法的处理速度.

为了更清楚地表述此剪枝策略,文中使用一种简单的 FP-Tree 作为例子,如图 1 所示的 FP-Tree, Null 节点有两个子节点: I2、I1,分布对应两个子树.

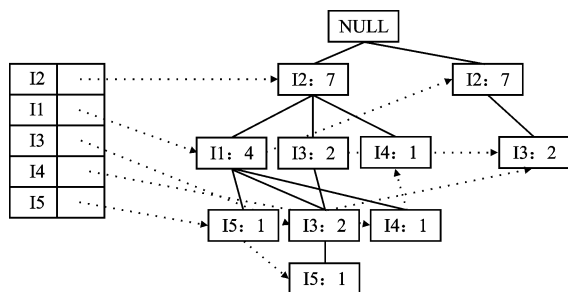


图 1 事务数据集 FP-Tree 表示图

对于 I2 子树中的子节点,如图 2 所示,主要检验支持度小于最小支持度的节点,按照由下到上的原则,从距离根节点最远的 I5 出发,因为 I5 的支持度低于最小支持度,因此将其与比它高一级的同名子节点 I5 合并,同时删除该 I5 节点.按照上述方法,距离节点 I2 较远的 I4,其支持度小于最小支持度,因此将其和上一级的 I4 合并,同时删除此 I4 节点,经过一系列的剪枝处理,最后得到最终的 I2 子树,如图 2 所示.

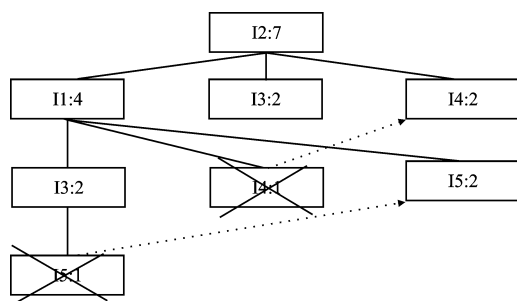


图 2 子 FP-Tree 表示图

4 试验结果与性能比较

4.1 集群配置部署

本文的实验环境是四台机器组成的集群,其中一台作为主节点,其余机器作为子节点.四台机器具有相同的硬件及软件配置,每台机器有两个 2.4 GHz Intel 处理器和 360 GB 内存,Ubuntu13.10 系

统,Jdk1.7,以及 0.20.2 版本的 Hadoop.

4.2 实验结果展示及分析

在本文试验中实现了三个算法: PFP 算法,本文提出的基于负载均衡和剪枝策略的 LBPFP 算法,以及为了说明负载均衡和剪枝策略各自方法对提高算法效率的有效性,在 PFP 算法的基础上增加了剪枝策略,为了区别于 PFP 算法,称其为 LBPFP-B 算法.通过对这三个算法的数据处理时间进行比较分析,依此来论证本文中提出算法的有效性.实验使用的数据集是由 IBM 的数据模拟生成器^[15]所生成,数据集含有 2 113 个数据项,48 000 条事务记录,事务平均长度 64,事务最长 73.

为了验证文中提出的均衡负载策略的有效性,图 3 中展示了 LBPFP 算法(基于负载均衡和剪枝策略两种方法)、LBPFP-B 算法(仅基于剪枝策略)两种算法在构建组内依赖的频繁模式树过程中子节点间的数据传输量,从图中可以看到,当数据量不是很大时,两种算法的数据传输量基本相同,随着处理的事务数量增加,两者的差距逐渐显现出来,LBPFP 算法在分组过程中由于增加了均衡分组策略,使得分组后各组的数据量基本保持一致,这样便可保证同一组的数据存储在同一节点中,由此便可在很大程度上减少子节点间的数据传输.

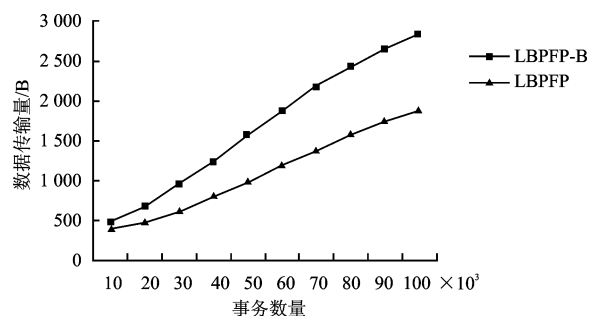


图 3 LBPFP 算法、LBPFP-B 算法子节点间的数据传输量比较

图 4 中给出了 LBPFP 算法、LBPFP-B 与 PFP 算法整体的处理效率对比,由图可知,在事务数量较低时,三种算法的运行时间并没有明显的差别,但是随着事务量的增加,LBPFP 算法相对 LBPFP-B 算法及 PFP 算法处理数据的效率明显提高,并且处理的数据量越大,LBPFP 算法的效率优势越明显.而 LBPFP-B 的数据处理效率亦明显高于 PFP 算法,却低于基于负载均衡及频繁树剪枝策略两种方法的

LBPFP 算法,由此说明两种策略均有助于提高 FP-Growth 算法的效率.通过上述实验分析说明了本文提出算法(LBPFP)的有效性.

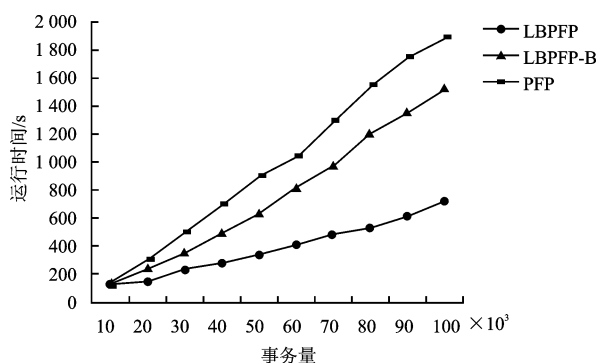


图4 LBPFP 算法、LBPFP-B 算法
与 PFP 算法下的性能比较

5 结束语

本文介绍了一种负载均衡并行的 FP-Growth 算法,该算法通过对事务数据库的均衡分组以及 FP-Tree 树的有效剪枝策略,显著地提高了算法的执行效率.实验结果比较分析说明 LBPFP 算法相对于 PFP 算法在性能方面具有显著的提升.为了进一步提高 FP-Growth 算法在每个节点的计算性能,将在条件模式树的递归迭代方便做进一步的研究工作.

参考文献:

- [1] Li Liu, Eric Li, Yimin Zhang, et al. Optimization of frequent itemset mining on multiple-core processor [C]// proceedings of the 33rd international conference on very large data based. [s. l.]: ACM, 2007: 1275-1285.
- [2] Mohammad El-Hajj, Osmar R, Zaiane. Parallel leap: large-scale maximal pattern mining in a distributed environment[C]// proceedings of the 12th international conference on parallel and Distributed Systems. Washington, DC, USA: IEEE, 2006: 135-142.
- [3] Gregory Buehrer, Srinivasan Parthasarathy, Shirish Tatikonda, et al. Toward terabyte pattern mining: an

architecture-conscious solution[J]. PPOPP, 2010(5): 103-111.

- [4] Lamine M Aouad, Nhien-An Le-Khac, Tahar M Kechadi. Distributed frequent itemsets mining in heterogeneous platforms[J]. Engineering, Computing and Architecture, 2012 (8):132-135.
- [5] H Li, Y Wang, D Zhang, et al. PFP: parallel FP-Growth for query recommendation[J]. ACM Recommender Systems, 2006(3):43-51.
- [6] M Zaharia, A Konwinski, A D Joseph, et al. Improving mapreduce performance in heterogeneous environments[J]. OSDI, 2012(2):65-78.
- [7] 郑启龙,房明,汪胜,等.基于 MapReduce 模型的并行科学计算[J].微电子学与计算机,2009,26(8):26-8.
- [8] C Yang, C Yen, C Tan, et al. Osprey: implementing mapreduce-style fault tolerance in a shared-nothing distributed database[C]// proceedings of the IEEE 26th International Conference on Data Engineering. [s. l.]: IEEE, 2010:657-668.
- [9] J Dean, S Ghemawat. Map reduce: a exible data processing tool[J]. Commun ACM, 2011(5): 72-77.
- [10] 刘怡,张戡.基于负载均衡和经验值的工作流任务分配策略[J].计算机工程,2012(2):231-234.
- [11] 应毅,任凯,刘正涛.基于云计算技术的数据挖掘[J].微电子学与计算机,2013,30(12):30-32.
- [12] 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.
- [13] 薛胜军,刘寅.基于 Hadoop 的气象信息数据仓库建立与测试[J].计算机测量与控制,2012(04):221-226.
- [14] 徐骁勇,潘郁,凌晨.云计算环境下资源的节能调度[J].计算机应用,2012(7):421-426.
- [15] IBM. IBM Quest Synthetic Data Generation Code[EB/OL] [2014-07-02] <http://www.almaden.ibm.com/>, 2009.

作者简介:

施亮男,(1987-),硕士研究生.研究方向为数据挖掘方向的研究. E-mail: shengjirenshe@163.com.

钱雪忠男,(1967-),硕士研究生,副教授.研究方向为数据库、数据挖掘方向的研究.