一种基于 Hadoop 的关联规则挖掘算法

丁 勇 朱长水 武玉艳
(南京理工大学泰州科技学院 江苏 泰州 225300)

摘 要 传统的并行关联规则算法对每一次迭代都定义一个 MapReduce 任务,以实现候选项集的生成和计数功能,但多次启动 MapReduce 任务会带来极大的性能开销。文中定义了一种并行关联规则挖掘算法 PST-Apriori,该算法采取分治策略,在每个分布式计算节点定义一个前缀共享树,通过递归调用的方式将事务 T 生成的候选项集逐层压缩到前缀共享树(PST)中。然后广度遍历 PST,逐层将每个节点对应的〈*key*,*value*〉作为 map 函数的输入,并由 MapReduce 框架自动按照 *key* 值进行聚集。最后调用 reduce 函数对多个任务的处理结果进行汇总,得到满足最小支持度阈值的频繁项集。算法只使用两个 MapReduce 任务,且 PST 按照 *key* 值排序便于 Mapper 端的 shuffle 操作,提高了运行效率。

关键词 关联规则,Hadoop,MapReduce,前缀共享树
中图法分类号 TP311 **文献标识码** A

Association Rule Mining Algorithm Based on Hadoop

DING Yong ZHU Chang-shui WU Yu-yan
(Taizhou College of Science and Technology, Nanjing University of Science and Technology, Taizhou, Jiangsu 225300, China)

Abstract The traditional parallel association rule algorithm defines a MapReduce task for each iteration to implement the generation and counting function of the candidate set, but multiple startup of the MapReduce task brings great performance overhead. This paper defined a parallel association rule mining algorithm (PST-Apriori). This algorithm adopts a partition strategy, defines a prefix shared tree in each distributed computing node, and compresses the candidate items generated by each transaction T to the prefix shared tree (PST). Then the breadth traversal algorithm is used, and the 〈*key*,*value*〉 corresponding to each node are used as input of the map function, and the MapReduce frame is automatically gathered according to the *key* value. Finally, the reduce function is called to aggregate the processing results of multiple tasks, and the frequent itemsets satisfying the minimum support threshold are obtained. The algorithm only usestwo MapReduce tasks, and PST is sorted according to key value to facilitate shuffle operation at Mapper, which improves the efficiency of operation.

Keywords Association rule, Hadoop, MapReduce, Prefix shared tree

1 引言

关联规则是数据挖掘技术中的重要算法之一,其目的是挖掘事务之间存在的某种隐含关系。传统的算法包括 Agrawal 等[1] 提出的 Apriori 算法,以及 Han 等[2] 提出的 FP-Growth 算法。但是随着挖掘数据量的不断增长,传统的关联规则算法存在单一计算节点无法存储所有数据以及运行内存不足等问题,不能满足大数据挖掘的需求。因此,基于 Hadoop 分布式计算框架的并行关联规则挖掘算法成为解决上述问题的有效方法。

Li 等[3] 提出了基于 Hadoop MapReduce 计算模型的 1 阶段并行 Apriori 算法,该算法只定义一个 MapReduce 任务,在 map 函数中对每一个事务 T 生成所有可能的候选项集,如 1 个数据分片中某个事务 T 有 n 个项集,则生成 $2n-1$ 个候选项集,项集被发送给对应的 reduce 函数进行统计,最后由 reduce 函数对所有计数进行合并,得到频繁项集。这种算法简单且容易实现,但是当事务较多时,map 阶段会产生大量的候选项集,造成大量的内存消耗,甚至导致程序无法运行。

Li 等[4] 提出了基于 MapReduce 的 k 阶段并行 Apriori 算法——PApriori,该算法首先使用一个 MapReduce 任务对所有事务 T 中的数据项进行计数,得到频繁 1 项集 F_1;然后按照一定的规则串接生成候选 2 项集 C_2,并将结果临时写入 HDFS;再由第二个 MapReduce 任务对 C_2 进行计数,生成频繁 2 项集 F_2。每一次迭代都定义一个 MapReduce 任务,map 阶段对上一阶段产生的候选项集进行计数,reduce 阶段对所有项集的支持度进行汇总得到频繁项集。但是,多次启动 MapReduce 任务会带来极大的性能开销,且算法只实现了计数的并行化,并未实现连接操作的并行化,不能充分发挥分布式计算的优势。

Zhou 等[5] 在 PApriori 算法的基础上进行了改进,增加了

本文受 2015 江苏省高校自然科学研究面上项目(15KJB520016),2017 年度江苏省高校"青蓝工程"资助。
丁 勇(1980—),男,硕士,副教授,主要研究方向为数据库理论与数据挖掘,E-mail:4383526@qq.com;朱长水(1981—),男,硕士,副教授,主要研究方向为虚拟现实、图像处理,E-mail:shui_zc@163.com;武玉艳(1981—),女,硕士,助理研究员,主要研究方向为信息技术基础、教学管理,E-mail:1718427921@qq.com。

由频繁 k 项集 F_k 生成候选 $k+1$ 项集 C_{k+1} 的并行化处理，但是额外启动了一个 MapReduce 任务，且数据分片、分发以及临时读写 HDFS 的时间消耗大大影响了算法执行的效率。

2 相关定义

2.1 关联规则

关联规则的思想是：当某些数据在值域上存在着高重复出现率时，说明这些数据之间存在着某种关系，这种关系被称作为关联规则。

支持度：给定事务数据库 D，项集 X 出现的次数占 D 中总事务的百分比称为 X 的支持度，如式(1)所示：

$$Support(x)=\frac{\|\{y\in Y|X\in y\}\|}{\|Y\|} \tag{1}$$

其中，X 表示数据库中的某项集合，Y 表示总的事务数据库，包含 X 的记录数为 $Support(X)$。

置信度：给定事务数据库 D，在支持项集 X 的事务中，支持项集 Y 的事务所占百分比称为 $X\Rightarrow Y$ 的置信度，记为 $confidence(X\Rightarrow Y)$，如式(2)所示：

$$Confidence(X\Rightarrow Y)=\frac{Support(X\bigcup Y)}{Support(X)} \tag{2}$$

2.2 前缀共享树

设两个项集 $X=\{i_1,i_2,\cdots,i_k,\cdots,i_m\}$，$Y=\{i_1,i_2,\cdots,i_k,\cdots,i_n\}$，它们的前 k 项相同，则$\{i_1,i_2,\cdots,i_k\}$称为项集 X 和 Y 的 k-前缀。如图 1 所示，项集$\{A,B,C,D\}$和项集$\{A,B,E,F\}$具有相同前缀$\{A,B\}$。

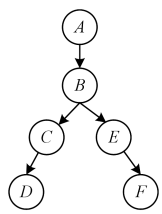

图 1 前缀共享树

2.3 MapReduce

MapReduce 是 Hadoop 分布式框架中定义的一种分布式计算模型，用于海量数据的并行运算，主要包括 Mapper 和 Reducer 两个任务。其中，Mapper 接受〈*key*，*value*〉格式的数据流，并产生一系列同样是〈*key*，*value*〉格式的输出，这些输出经过处理，形成〈*key*，{*value list*}〉形式的中间结果。Reducer 将 Mapper 产生的中间结果作为输入，将相同 *key* 值的{*value list*}做相应的处理，最终生成〈*key*，*value*〉形式的数据，并将其写入 HDFS，原理如图 2 所示。

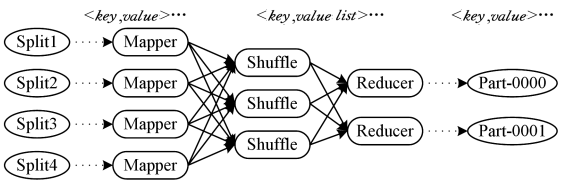

图 2 MapReduce 计算模型

3 并行挖掘算法(PST-Apriori)

3.1 算法思想

首先使用 Hadoop HDFS 实现对数据集的自动分片及分布式存储，调用第一次 MapReduce 任务对所有项集进行计数，删除不满足最小支持度阈值的项集，得到新的数据集 D，并对 D 进行预处理，将每个事务的数据项按指定类型排序；然后调用第二次 MapReduce 计算任务，对 D 进行数据分片，采取分治策略依次读取分片中的每个事务的数据项，通过递归调用的方式，逐层生成候选项集，并将之压缩到前缀共享树 PST 中；最后通过广度遍历每个计算节点的 PST，逐层将对应的〈*key*，*value*〉作为 map 函数的输入，由 MapReduce 框架自动按照 *key* 值进行聚集，并调用 reduce 函数将多个任务的处理结果进行汇总，得到所有频繁项集。

算法采用广度遍历的方式，按照 *key* 值排序便于 Map 和 Reduce 端的 shuffle 操作，大大减少了每个分片输出的候选项集的数量，也减少了在第二阶段进行计数的候选项集的数量，提高了算法的效率。算法流程如图 3 所示。

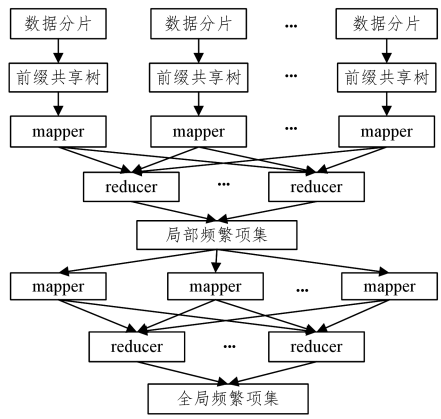

图 3 算法流程图

PST-Apriori 算法定义的 PST 用于压缩各个数据项，树中每个节点 α 对应一个数据项，包括属性：α. name 数据项的名称，α. count 数据项的支持度，α. pre 数据项的前缀结点，α. children 数据项的孩子节点，α. lay 数据项对应的层。通过一个 HashMap 结构的层头表记录 PST 的层信息，以便逐层对应 map 函数的键-值对，如图 4 所示。

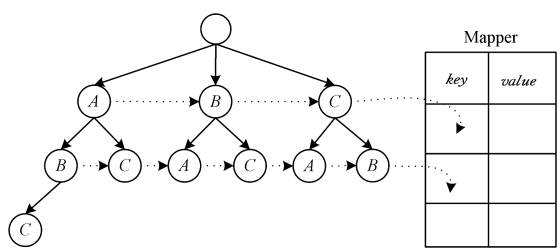

图 4 PST 数据结构

MapReduce 计算模型主要包括 map 和 reduce 两个函数。在 map()函数中，首先扫描数据分片 Data，对每个事务的数据项进行分割，生成第一层频繁项集 L_1，并按照指定类型将其排序。然后，将 L_1 作为初始输入项，通过调用 CreatePST()函数逐层扩展，生成每个计算节点的前缀共享树。最后，广度遍历 PST 中的每个节点 α，将 α 的前缀节点 α. pre. name 以及 α. name 作为 map 函数的 *key*，将 α. count 作为 map 函数的 *value*，MapReduce 框架自动按照 *key* 值进行聚集，将 *key* 值相同的数据统一交给一个 reduce()函数处理。reduce()函数把分解后多个任务的处理结果进行汇总，得到所有候选项集。对于满足最小支持度阈值的项集，将其添加至 *context* 上下文对象作为频繁项集的输出。算法的伪代码如下。

算法 1 函数 Mapper

输入：数据分片 Data

输出:Context〈key/value〉

1. PST=∅
2. Scan Data
3. For each data∈Data
4. L_1=data. split() //分割
5. sort(L_1) //排序
6. PST=CreatePST (L_1) //创建前缀共享树
7. For each i∈PST. totalLay //广度遍历
8. For each $\alpha \in L_i$
9. α. name+=α. preName
10. context. write(α. name,α. count)

算法 2　函数 Reducer

输入:〈key/values〉

输出:〈key/count〉

1. count=0
2. For each value∈values
3. count+=val. get()
4. If count≥ min_sup
5. Context. write(key,count)

函数 CreatePST()用于在每个计算节点中生成前缀共享树 PST,基本思想是对于第 L_i 层的每个节点 α,找出其在 L_i 的位置 k,然后将 $k+1$ 后面的每个节点 β 依次作为 α 的子节点,若此节点已经存在,则将该节点出现的次数加 1,通过递归函数调用逐层生成前缀共享树。

算法 3　CreatePST

输入:第 i 层节点 L_i

输出:前缀共享树

1. new L_{i+1}=∅
2. For each $\alpha \in L_i$
3. k=α. IndexOf(L_1)
4. For k+1 to length(L_1)
5. β=L_1. get(i)
6. If not exit(β)
7. α. child. add(β)
8. β. count=1
9. else
10. β. count++
11. If length(L_i+1)>0
12. BreadthMinner (L_{i+1}) //递归

3.2　关联规则提取

在 MapReduce 过程中,基于最终生成的 PST 提取关联规则,沿着树中由根节点到每个叶子节点的路径,每个叶子节点都创建一条规则,每个分割都成为规则中的一个前件,叶子节点中的节点名称为 Then 的后件,如算法 4 所示。

算法 4　Generate_Rules(Node)

输入:共享树(PST)

输出:规则(Rules)

1. Node=Root //根节点
2. Rules=∅ // 规则
3. If Node ≠ 叶子节点 then
4. For each Child in Node //每个子结点
5. Ruls. left=Node. name
6. Rules. right=Child. name
7. Generate_Rules (Child) //递归调用
8. End for
9. Return Rules //返回规则

3.3　算法复杂度分析

给定数据集 L,包含 k 个事务,数据项类型集合为 ε,事务的平均长度为 L_k,计算节点数量为 N。

时间复杂度:由 mapper 阶段 T_{map} 和 reduce 阶段 T_{reduce} 两部分组成。在 MapReduce 模型中,输入的数据和输出的计算结果都表示成键值对的形式,N 个节点并行化处理的时间复杂度为 $|L/N|$,每个节点对应一个 PST,生成 PST 的时间复杂度为 $O(|\varepsilon * L_k^2|)$,$T_{\text{map}}=|L/N| \cdot |\varepsilon * L_k^2|$;reduce 函数处理 H 层 PST,每层最多 ε 个节点,$T_{\text{reduce}}=|\varepsilon \cdot H|$。不考虑两个函数通信的时间开销,并行处理总的时间复杂度为 $O(|L/N| \cdot |\varepsilon * L_k^2+|\varepsilon \cdot H|)$。

空间复杂度:PST 每个节点存储一个数据项,共 N 个计算节点,空间复杂度为 $O(|L \cdot N|)$。

4　应用实例

给定一个计算节点的数据片 data,排序后的数据项包含 4 种类型,最小支持度 $min\_sup=2$,数据片如下所示:

*trans*1(*A*,*B*,*C*)

*trans*2(*A*,*B*,*C*,*D*)

*trans*3(*A*,*C*,*D*)

…

根据 PST-Apriori 算法,扫描第一个事务,在 root 节点下生成 3 层节点:L_1 包括〈*A*:1〉,〈*B*:1〉,〈*C*:1〉,L_2 包括〈*AB*:1〉,〈*AC*:1〉,〈*BC*:1〉,L_3 包括〈*ABC*:1〉。扫描第二个事务,L_1 包括〈*A*:2〉,〈*B*:2〉,〈*C*:2〉,〈*D*:1〉,L_2 包括〈*AB*:2〉,〈*AC*:2〉,〈*AD*:1〉,〈*BC*:2〉,〈*BD*:1〉,〈*CD*:1〉,L_3 包括〈*ABC*:2〉,〈*ABD*:1〉,〈*ACD*:1〉,〈*BCD*:1〉,L_4 包括〈*ABCD*:1〉。最终生成的 PST 如图 5 所示,从图中可以看出算法在 map()函数中添加了 11 个键值对。

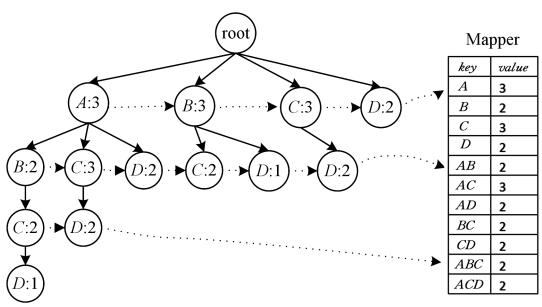

图 5　PST 与 Mapper

5　实验评估

5.1　实验环境

实验在 Hadoop 集群环境中完成,集群由 3 台 PC 机组成,其中 1 台 PC 机作为 master,另外 2 台 PC 机分别作为 slave1 和 slave2。机器硬件环境为 3.4 GHz Intel i5-7500 CPU、16 GB 内存。软件环境为 ubuntu 14.04 操作系统、Hadoop 2.6.1 分布式计算框架。程序基于 Eclipse 平台、MapReduce 插件,用 Java 语言实现。

5.2　实验分析

5.2.1　模拟数据

采用 IBM 数据合成器 QUEST 合成模拟数据,定义 1000 种长度为 6 的事件类型,以 500 MB 为步长,随机产生大小分别为 500 M,1 G,1.5 G,2 G,…的数据集作为测试数据,对于给

(下转第 416 页)

算法[J]. 计算机研究与发展,2007,44(2):296-301.
[8] 王吉源,黎晨,王婵娟. 用户属性加权活跃近邻的协同过滤算法[J]. 计算机应用研究,2016,33(12):3625-3629.
[9] 赵文涛,王春春,成亚飞,等. 基于用户多属性与兴趣的协同过滤算法[J]. 计算机应用研究,2016,33(12):3630-3633.
[10] ADOMAVICIUS G,TUZHILIN A. Toward the Next Generation of Recommender Systems:A Survey of the State-of-the-Art and Possible Extensions[J]. IEEE Transactions on Knowledge & Data Engineering,2005,17(6):734-749.
[11] HARPER F M,KONSTAN J A. The MovieLens Datasets:History and Context[J]. Acm Transactions on Interactive Intelligent Systems,2015,5(4):19.

(上接第 411 页)

定的支持度阈值,基于不同数量的计算节点,得到如图 6 所示的运行结果。

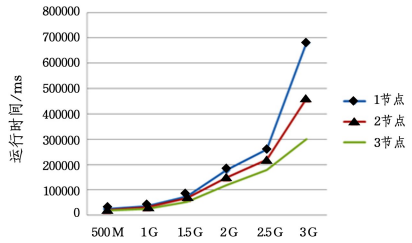

图 6 数据片与运行时间

从图 6 中看出,随着数据集大小的增加,运行时间开销逐渐增加。对于小数据集,MapReduce 计算模型并未充分发挥优势,图中 500 M 与 1 G 数据集运行 PST-Apriori 算法的时间差异性不大,这主要是因为在 Hadoop 平台下,数据在存入 HDFS 之前仅被分割为几个数据分片,且单个节点 DataNode 已经满足数据存储,mapper 操作大部分在内存中完成,不会频繁写入 HDFS,因此节点数量并不占太大优势。但是,随着数据集的增大和计算节点的增加,数据集实现了分布式存储与计算,算法的运行效率明显提升。

5.2.2 真实数据

采用 R 语言 arules 软件包中的 Groceries 数据集进行测试,该数据集是某一食品杂货店一个月的真实交易数据,共包含 169 种数据项。通过改变支持度阈值(min_sup=0.005~0.5),比较传统 Apriori 算法与 PST-Apriori 算法的执行效率,得到如图 7 所示的运行结果。

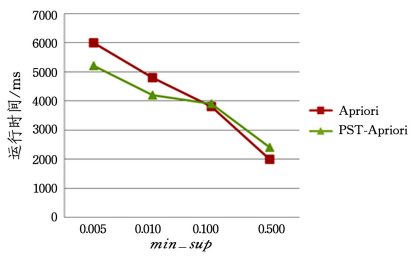

图 7 Apriori 与 PST-Apriori

从图 7 可以分析出,随着支持度阈值的增加,两个算法的时间开销均减少。当支持度阈值较大时,传统的 Apriori 算法比 PST-Apriori 算法的执行效率高,这是因为 Apriori 算法生成的候选项集少,迭代的次数少,而 PST-Apriori 算法需要逐层生成候选项集,并压缩到 PST 中,且需要分布式计数,增加了时间开销。当支持度阈值较小,PST 树中压缩的节点数量较多时,因为 PST-Apriori 算法采取广度优先搜索的策略,且按照 key 值排序,加快了 Mapper 和 Reducer 端的 shuffle 操作,充分发挥了 MapReduce 分布式计算的优势,因此提高了算法的执行效率。相比之下,对于较小的支持度阈值,PST-Apriori 算法的执行效率有了明显的提升。

结束语 本文提出的 PST-Apriori 算法主要用于解决传统并行关联规则算法多次启动 MapReduce 任务带来较大性能开销的问题,算法只需要两个 MapReduce 任务,第一个 MapReduce 任务用于分布式计数,统计频繁 1 项集;第二个 MapReduce 任务将每个事务 T 生成的候选项集逐层压缩到 PST 中,PST 中的每个节点对应 map 函数的键值对,由 MapReduce 框架自动实现聚集。最后调用 reduce 函数进行分布式汇总,得到满足支持度阈值的频繁项集。模拟和真实数据的实验证明,PST-Apriori 面对较大数据集时能有效地挖掘关联规则。

参 考 文 献

[1] AGRAWAL R,SRIKANT R. Fast algorithms for mining association rules(3rd ed)[M]// Readings in Database Systems. Morgan Kaufmann Publishers Inc.,1998:2299-2308.
[2] HAN J,PEI J,YIN Y. Mining frequent patterns without candidate generation[C]// ACM SIGMOD International Conference on Management of Data. ACM,2000:1-12.
[3] LI L,ZHANG M. The Strategy of Mining Association Rule Based on Cloud Computing[C]// International Conference on Business Computing and Global Informatization. IEEE,2011:475-478.
[4] LI N,ZENG L,HE Q,et al. Parallel Implementation of Apriori Algorithm Based on MapReduce[C]// Acis International Conference on Software Engineering,Artificial Intelligence,Networking and Parallel & Distributed Computing. IEEE,2012:236-241.
[5] ZHOU X,HUANG Y. An Improved Parallel Association Rules Algorithm Based on MapReduce Framework for Big Data[C]// International Conference on Fuzzy Systems and Knowledge Discovery. IEEE,2014:284-288.
[6] 郝天曙. 基于 Hadoop 的并行数据挖掘的研究[D]. 南京:南京邮电大学,2017.
[7] 张玲. 基于 Hadoop 平台并行关联规则挖掘算法研究[D]. 西安:西安科技大学,2017.
[8] 荀亚玲. 集群环境下的关联规则挖掘及应用[D]. 太原:太原科技大学,2017.
[9] 于跃. 基于 Hadoop 平台的并行化分布式关联规则挖掘算法研究[D]. 吉林:吉林大学,2017.
[10] 李若晨. 基于并行的 Apriori 数据挖掘算法的研究[D]. 吉林:吉林大学,2017.
[11] 叶璐. 基于 Spark 的改进关联规则算法研究[D]. 太原:太原科技大学,2017.
[12] 马连灯. 基于 HADOOP 平台的并行关联规则算法研究[D]. 天津:天津工业大学,2017.