

基于 Hadoop 的大数据频繁模式挖掘算法

李校林^{1,2}, 杜 托¹, 谢 勇¹

(1 重庆邮电大学 通信新技术应用研究中心, 重庆 400065; 2 重庆信科设计有限公司, 重庆 400021)

摘 要: 针对传统的频繁模式挖掘算法不能满足大数据环境下的挖掘需要, 提出一种高效挖掘大型数据库中频繁模式的并行算法 H_PrePost. 首先从压缩数据库、简化数据表示以及采用高效的连接和剪枝策略等方面对 PrePost 算法进行改进, 以提高单机模式下的挖掘效率. 然后将改进算法迁移到 Hadoop 平台上, 利用 MapReduce 模型进行并行计算, 同时提出一种负载均衡策略保证集群高效运行. 最后使用 kulczynski 度量和不平衡比对所挖掘的频繁模式进行评估, 以确保所挖掘模式具有实际应用价值. 实验结果表明, H_PrePost 算法可以有效挖掘大数据集中的频繁模式.

关键词: Hadoop; 频繁模式; 大数据

中图分类号: TP311

文献标识码: A

文章编号: 1000—7180(2018)09—0014—06

Algorithm for Mining Frequent Patterns in Big Data Based on Hadoop

LI Xiao-lin^{1,2}, DU Tuo¹, XIE Yong¹

(1 Chongqing University of Posts and Telecommunications, New Technology Application Research Center, Chongqing 400065, China; 2 Chongqing Information Technology Designing Co. Ltd, Chongqing 400021, China)

Abstract: Aiming at the traditional frequent pattern mining algorithm can not meet the needs of mining in big data environment, a parallel algorithm for efficiently mining frequent patterns in large databases is proposed. Firstly, PrePost algorithm is improved from compressing database, simplifying data representation and using efficient connection and pruning strategy, which improve the efficiency of mining in stand-alone mode. Then, the improved algorithm is migrated to the Hadoop platform and the MapReduce model is used for parallel computing. A load balancing strategy is proposed to ensure the efficient operation of the cluster. Finally, the frequent pattern mining is evaluated using kulczynski metric and unbalance ratio to ensure that the mining pattern has practical value. Experimental results show that this algorithm can effectively mine the frequent patterns in big data sets.

Key words: Hadoop; frequent pattern; big data

1 引言

在当今大数据环境下, 如何从大量数据中挖掘出有价值的知识与规则显得尤为重要. 关联规则挖掘是数据挖掘研究的重要内容, 也是近年来的研究热点. 关联规则挖掘一般分为两个步骤: 从数据库中挖掘出符合支持度要求的频繁模式; 由频繁模式组合得到最终的关联规则. 由于后者与具体应用场景

关系较为密切且是一个排列组合过程, 因此学者们将研究重点放在第一步. 频繁模式挖掘算法主要分为两类, 一是基于数据库的 Apriori 算法^[1]及其改进算法, 如 BitTableFI^[2]、DHP^[3]等, 此类算法易于实现, 但是需要频繁扫描数据库, 挖掘效率不高. 二是基于模式树结构的 FP-growth 算法^[4]及其改进算法, 如 FP-growth*^[5]、PrePost^[6]等, 此类算法解决了类 Apriori 算法需要多次扫描数据库的问题且

挖掘效率有所提高,但是存在建树复杂,没有高效的连接和剪枝策略,面对大型数据库时存在内存溢出的问题,不能满足大数据关联规则挖掘的需要. 本文首先使用高效的建树算法和连接、剪枝策略对现有效率较高的 PrePost 算法进行改进,然后将改进算法迁移到 Hadoop 平台上,提出一种高效挖掘大型数据库中频繁模式的并行算法,有效解决了传统算法不能挖掘大数据中频繁模式的问题.

2 相关概念

2.1 问题描述

设 $I = \{i_1, i_2, \dots, i_m\}$ 是数据库中所有项目集合, DB 是一个事务数据库,事务 $T = \{i_1, i_2, \dots, i_n\}$ 代表不同项目的集合. 如表 1 为一个典型的事务数据库

表 1 事务数据库 DB

项目	频繁 1- 项集
1 $i_1 i_2 i_7$	$i_1 i_2$
2 $i_1 i_2 i_3 i_4$	$i_3 i_1 i_2 i_4$
3 $i_1 i_3 i_5$	$i_3 i_1 i_5$
4 $i_1 i_2 i_3 i_5$	$i_3 i_1 i_2 i_5$
5 $i_3 i_4 i_5 i_6$	$i_3 i_4 i_5$
6 $i_3 i_4$	$i_3 i_4$

定义 1:(支持度) 项集 X 的支持度定义为 DB 中含有项集 X 的事务数目与总事务数之比, 记作 $\sigma(X)$. 设 $|DB|$ 代表事务总数, $\sup(X)$ 代表含有项集 X 的事务数目, 则项目集 X 的支持度 $\sigma(X) = \sup(X) / |DB|$.

性质 1:(Apriori 性质) 频繁项集的所有子集一定是频繁项集; 非频繁项集的任一超集一定是非频繁项集.

利用上述相关定义与性质, 就可以将频繁模式挖掘问题转换为挖掘事务数据库中存在的且满足最小支持度要求的项集问题.

2.2 构造遍历树 TB-tree

构造遍历树 TB-tree (Traversal when Building) 结构^[7]是一种新颖的数据结构, 可以高效压缩事务数据库. 每个 TB-tree 包含一个 Root 节点和项目的前缀子树节点. Root 节点为根节点, 将其置为 NULL. 每个项目的前缀子树节点由五个部分构成分别为: item_name, count, parent_pointer, start_build, finish_build. 其中 Item_name 代表项目名称, count 代表包含此节点的事务数目, parent_pointer

指向其父节点, start_build 代表此节点构造的开始顺序, finish_build 代表此节点构造的结束顺序. 图一为表一数据库对应的 TB-tree. 以节点 c 为例, item_name 为 c, count 为 5, start_build 为 3, finish_build 为 10, parent_pointer 指向 Root 节点.

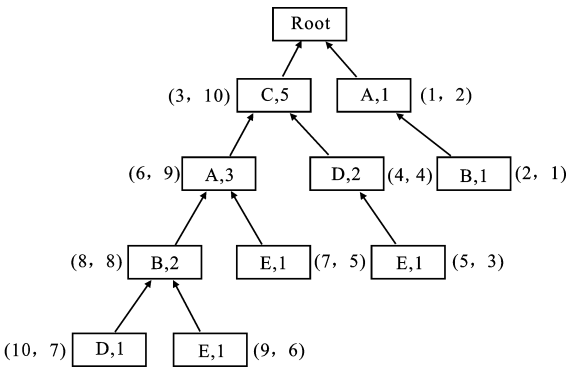


图 1 DB 对应的 TB-tree

由于 TB-tree 可以在构造阶段得到 start-build 与 finish-build 的值, 相较于 PrePost 算法采用的 PPC-tree 数据结构, 不需要两次遍历来得到所有前缀子树节点的 pre_order 与 post_order 的值, 因此构建 TB-tree 比 PPC-tree 效率更高.

2.3 构造链表 B-list

B-list 是在 TB-tree 的基础上提出的一种数据结构, 其优势是可以简化数据表示, 快速得到频繁项集的支持度, 提高挖掘效率. 下面逐步给出其定义与性质.

由于 B-list 是由 B-info-code 构成的, 首先给出 B-info-code 的定义与性质.

定义 2:(B-info-code) 对 TB-tree 中除根节点以外的任一节点 M , 定义 $[(M.start_build, M.finish_build), M.count]$ 为 M 的 B-info-code.

性质 2: 对 TB-tree 中除根节点以外的任意两个节点 M 和 N , 当且仅当 $M.start_build < N.start_build$ 与 $M.finish_build > N.finish_build$ 均成立, M 节点才是 N 节点的祖先.

证明: 在构建 TB-tree 时, 祖先节点的开始构建顺序总是在孩子节点之前, 且其结束构建顺序总是在孩子节点之后, 因此祖先节点的 start_build 值总是小于其孩子节点, finish_build 值反之. 得证.

除此之外, 由于 TB-tree 中除根节点外每一个节点与其对应的 B-info-code 具有一一对应关系, 因此可以用性质 2 来表示 B-info-code 之间的孩子—祖先关系, 具体见定义 3.

定义 3:(B-info-code 关系) 设 $X: [(x_1, y_1), z_1]$

为节点 M 的 B-info-code, $Y: [(x_2, y_2), z_2]$ 为节点 N 的 B-info-code, 当且仅当 $x_1 < x_2$ 与 $y_1 > y_2$ 均成立, X 才是 Y 的祖先。

基于 B-info-code 的定义与性质, 结合其孩子祖先关系, 下面给出频繁 1-项集、2-项集及 k -项集对应的 B-list 定义。

定义 4(频繁 1-项集对应的 B-list) 对 TB-tree 中除根节点以外的节点来说, 所有具有相同 Item_name 节点的 B-info-code 构成的集合为 1-项集对应的 B-list. 其中 B-info-code 按照 start-build 升序排列。

定义 5:(2-项集对应的 B-list) 设 X, Y 为频繁 1-项集中不同的两项, X 在 Y 之前, X 的 B-list 为 $\{[(x_{11}, y_{11}), z_{11}], [(x_{12}, y_{12}), z_{12}], \dots [(x_{1n}, y_{1n}), z_{1n}]\}$, Y 的 B-list 为 $\{[(x_{21}, y_{21}), z_{21}], [(x_{22}, y_{22}), z_{22}], \dots [(x_{2m}, y_{2m}), z_{2m}]\}$, 则 XY 的 B-list 可用如下方法确定:

首先判断 $[(x_{1i}, y_{1i}), z_{1i}] (1 \leq i \leq n)$ 与 $[(x_{2j}, y_{2j}), z_{2j}] (1 \leq j \leq m)$ 的孩子-祖先关系, 如果前者是后者祖先, 则将前者加入到 XY 的 B-list 中, 否则继续进行下一项的判断, 判断完毕后可以得到 XY 的 B-list. 但此时的 B-list 中很多 B-info-code 具有相同 start-build 与 finish-build 值, 可以对其进行合并操作以减少其数量. 如 $\{[(x_1, y_1), z_1], [(x_1, y_1), z_2], \dots, [(x_1, y_1), z_n]\}$, 可以合并为 $[(x_1, y_1), z_1 + z_2 + \dots + z_n]$.

k -项集的 B-list 产生方法与 2-项集类似, 此处不再赘述。

性质 3: 设项集 M 的对应的 B-list 为 $\{[(x_1, y_1), z_1], [(x_2, y_2), z_2], \dots, [(x_n, y_n), z_n]\}$, 则项集 M 的支持数 $\text{sup}(M) = z_1 + z_2 + \dots + z_n$.

证明: 由于项集 M 对应的 B-list 中包含有一个或多个 B-info-code, 而 B-info-code 与 TB-tree 中的节点具有一一对应的关系, 因此 M 的支持数就为 TB-tree 中所有同名节点的支持数之和, 即 $\text{sup}(M) = z_1 + z_2 + \dots + z_n$.

3 基于 Hadoop 的频繁模式挖掘算法

针对传统频繁模式挖掘算法挖掘效率较低且不能满足大数据频繁模式挖掘需要的问题, 本章首先对传统频繁模式挖掘 PrePost 算法进行改进, 提出一种高效的单机频繁模式挖掘算法. 然后将改进算法与 Hadoop 平台结合, 提出一种大数据环境下的频繁模式挖掘算法 H_PrePost, 能够有效挖掘出大

数据中的频繁模式。

3.1 基于 B-list 的改进 PrePost 算法

PrePost 算法是现有效率较高的频繁模式挖掘算法, 但是存在构建 PPC-tree 复杂、缺乏高效的连接和剪枝策略等问题. 本节结合上一章的 TB-tree 与 B-list 等新的数据结构, 从压缩数据库、简化数据表示以及采用高效的连接和剪枝策略等方面对 PrePost 算法进行改进, 以提高单机模式下的挖掘效率, 为将算法迁移到 Hadoop 平台以挖掘大数据中的频繁模式打好基础. 算法具体描述如下。

步骤一: 扫描一次事务数据库, 得到频繁 1-项集。

步骤二: 对所有的频繁 1-项集按支持度降序排列, 得到频繁 1-项集列表 F-list, 剔除事务中的非频繁项, 将剩余频繁项按 F-list 进行排序, 构建 TB-tree, 得到频繁 1-项集对应的 B-list。

步骤三: 挖掘频繁 2-项集

文献[8]表明, 在频繁模式挖掘算法中, 候选 2-项集的数量远远多于其他候选项集, 对其进行支持度计算会耗费大量时间与系统资源. 利用 TB-tree 特性可以有效解决这一问题. 以开始构建顺序 start-build 对 TB-tree 进行遍历, 对任一节点 M , 由于 TB-tree 中具有孩子-祖先关系的项目必定在一条路径上, 因此只需要将具有孩子-祖先关系的节点进行连接便可以产生所有的候选 2-项集. 这种方式可以大大减少 2-候选项集的数目, 从而加快算法执行. 然后将候选 2-项集的支持度与最小支持度进行比较, 剔除所有不满足要求的候选 2-项集, 便可得到所有的频繁 2-项集. 对频繁 1-项集的 B-list 进行连接, 即可得到频繁 2-项集的 B-list。

步骤四: 挖掘频繁 k -项集

由于频繁 k -项集长度较长、挖掘过程复杂, 是消耗计算资源与时间的重要部分. 经过研究, 本文利用集合枚举树、Apriori 性质、B-list 连接策略来高效挖掘频繁 k -项集。

(1) 产生候选 k -项集

PrePost 算法使用 Apriori 算法的自连接策略产生候选 k -项集, 这种方法会产生大量非频繁项集, 效率不高. 本文利用集合枚举树来产生候选项集. 在产生频繁 $k-1$ -项集后, 集合枚举树中大量的非频繁项集节点及其子节点可以删除掉, 从而简化集合枚举树, 然后利用集合枚举树中剩余的节点产生候选 k -项集, 可以减少候选项集的数目。

(2) 剪枝候选 k -项集

由性质 1 可知,频繁项集的所有子集一定是频繁项集,因此利用频繁 $k-1$ 项集对候选 k -项集进行剪枝. 如果候选 k -项集中的任一子集不是频繁的,则将其剔除候选 k -项集.

(3) 产生频繁 k -项集

得到候选 k -项集之后,计算其支持度,然后将其与用户设置的支持度进行比较,将不满足用户支持度要求的项集删除,即可得到频繁 k -项集. 通过对产生候选 k -项集的两个 $k-1$ -频繁项集的 B-list 进行连接,得到的候选 k -项集 B-list 的 count 值即为其支持度,如果大于用户设置的最小支持度,则为频繁 k -项集,反之,则剔除. 在此过程中,连接两个候选 k -项集 B-list 的算法复杂度较高,因此本文使用如下一种具有线性复杂度的连接策略:

设候选 $k-1$ -项集 M 的 B-list 为 $\{[(x_{11}, y_{11}), z_{11}], [(x_{12}, y_{12}), z_{12}], \dots, [(x_{1n}, y_{1n}), z_{1n}]]\}$, 候选 $k-1$ -项集 N 的 B-list 为 $\{[(x_{21}, y_{21}), z_{21}], [(x_{22}, y_{22}), z_{22}], \dots, [(x_{2m}, y_{2m}), z_{2m}]]\}$, 首先利用定义 3 判断 $[(x_{1i}, y_{1i}), z_{1i}]$ 与 $[(x_{2j}, y_{2j}), z_{2j}]$ 之间的孩子-祖先关系. 如果 $[(x_{1i}, y_{1i}), z_{1i}]$ 是 $[(x_{2j}, y_{2j}), z_{2j}]$ 的祖先,则将 $[(x_{1i}, y_{1i}), z_{2j}]$ 添加到候选 k -项集 MN 的 B-list 中. 然后搜索判断 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 是否存在,如果存在,则继续判断 $[(x_{1i}, y_{1i}), z_{1i}]$ 与 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 之间的孩子-祖先关系. 如果 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 不存在,则连接结束. 如果 $[(x_{1i}, y_{1i}), z_{1i}]$ 不是 $[(x_{2j}, y_{2j}), z_{2j}]$ 的祖先,则存在两种可能, x_{1i} 大于 x_{2j} 或 x_{1i} 小于 x_{2j} .

如果 x_{1i} 大于 x_{2j} ,若 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 存在,则继续判断 $[(x_{1i}, y_{1i}), z_{1i}]$ 与 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 之间的孩子-祖先关系. 若 $[(x_{2j+1}, y_{2j+1}), z_{2j+1}]$ 不存在,则连接结束.

如果 x_{1i} 小于 x_{2j} ,若 $[(x_{1i+1}, y_{1i+1}), z_{1i+1}]$ 存在,则继续判断 $[(x_{1i+1}, y_{1i+1}), z_{1i+1}]$ 与 $[(x_{2j}, y_{2j}), z_{2j}]$ 之间的孩子-祖先关系. 若 $[(x_{1i+1}, y_{1i+1}), z_{1i+1}]$ 不存在,则连接结束.

(4) 若没有频繁 $k+1$ -项集产生,输出挖掘到的所有频繁项集,算法结束.

3.2 基于 Hadoop 的并行挖掘算法

由于传统频繁模式挖掘算法不能满足大数据环境下的挖掘需要,本节结合上节中的改进 PrePost 算法,提出一种基于 Hadoop 的大数据频繁模式挖掘算法 H_PrePost. 首先利用 HDFS 对大数据集进行分组,然后利用 MapReduce 模型进行并行计算,挖掘出所有的频繁模式. 针对集群中的负载均衡问

题,提出一种保证集群高效运行的负载均衡策略. 最后使用 kulczynski 度量和不平衡比对所挖掘的频繁模式进行评估,以确保所挖掘模式具有实际应用价值. 算法整体过程如图 2 所示.

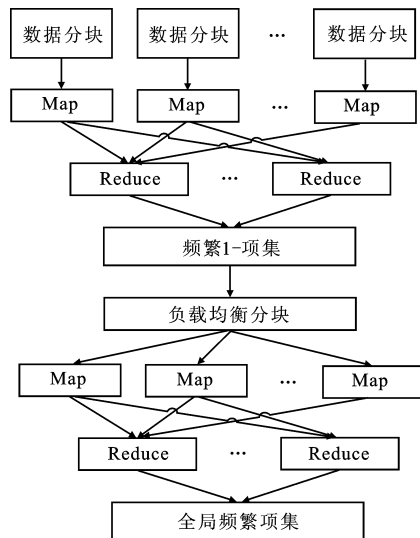


图2 算法总体框图

步骤一:对大数据集进行分块

将改进 PrePost 算法迁移到 Hadoop 平台上,首先要对待挖掘的大数据集进行分块,这是基于 Hadoop 平台进行并行算法设计分而治之的思想^[9]的一个具体体现. 此步骤利用 HDFS 的默认分组策略,如有其他需要,可自定义设置其值.

步骤二:利用 MapReduce 任务挖掘 F_1

将步骤一中的每一个分块 Block 作为一个 Map 的输入,Map 统计每个 Block 中项的出现次数,为减少节点间的通信次数,利用 Combine 将所有具有相同 Key 值的项进行合并操作,然后将结果输入到 Reduce 任务中,Reduce 对所有 Map 的结果进行合并,按最小支持度要求生成所有频繁 1-项集 L_1 并将其降序排列得到频繁 1-项集列表 F-list.

步骤三:对 F-list 进行负载均衡分组

由 3.1 节可知, TB-tree 的深度对算法影响较大,因此需要保证集群中的每个节点所构建的 TB-tree 具有相近的深度. 而对两个候选频繁项集的 B-list 进行合并操作是算法中耗时最多的一步,其算法复杂度为对应 B-list 长度之和,而 B-list 与频繁 1-项集具有对应关系,因此最终分组归结为对 F-list 进行分组,分组后的结果为 G-list. 具体分组过程如下:

(1) 设集群中的节点数为 N , 如果 N 多于频繁 1-项集的个数,则每个频繁项为一组,分组结束. 反之继续进行如下步骤.

(2)计算每一个频繁 1-项集的负载量并降序排列. 对 G-list 的 N 个分组每组添加一个频繁 1-项集,其负载量为对应频繁 1-项集的负载量.

(3)对剩余频繁 1-项集进行分组. 将其添加到负载量最小的一组并更新其负载量.

(4)重复此步骤,直到所有频繁 1-项集均得到对应分组,将 G-list 保存至 HDFS,分组结束.

步骤四:利用 MapReduce 任务挖掘频繁 k -项集
首先利用 Map 任务将负载均衡的事务发送到各个节点:

(1)读取 HDFS 中的 G-list,根据 G-list 建立 HashTable,以项为键值对中的 key,组号为 value.

(2)读取每条事务中的最后一个频繁项,利用 HashTable 得到其组号,然后以组号为 key,其前面频繁项为 value 从而构成 Map 阶段的输出键值对.

(3)重复(2),直到所有事务处理完成.
然后利用 Reduce 任务构建每个分块的 TB-tree,并行挖掘出所有的频繁模式:

(1)利用 Map 任务输出的键值对构造每个分块的 TB-tree,然后得到其对应的频繁 1-项集的 B-list.

(2)采用上一节提出的高效的频繁模式挖掘算法挖掘出各个分块的频繁模式,然后对其进行合并操作,得到全局所有频繁模式.

步骤五:评估挖掘出的频繁模式
由于大数据环境下数据量大,所挖掘出的频繁模式数量也巨大,其中包含很多没有意义的规则,因此对其进行评估很有必要.

使用 kulczynski 度量和不平衡比评估频繁项集,可以有效剔除没有相关关系及具有负相关性的项集,确保项集之间具有正相关性. 需要说明的是,具体阈值选取需要依据具体情况来确定. 对于两个项集 A 和 B ,其 kulczynski 度量如下

$$kulc(A,B)=\frac{1}{2}(P(A|B)+P(B|A))$$

对于满足 kulczynski 度量的项集,使用不平衡比来评估其不平衡程度,不平衡比越小,效果越好,其公式如下:

$$IR(A,B)=\frac{|\sup(A)-\sup(B)|}{\sup(A)+\sup(B)-\sup(A\cup B)}$$

步骤六:输出评估后的所有频繁项集,算法结束.

4 实验结果与分析

为了验证本文算法的有效性,首先在单机条件下对 PrePost 算法、改进 PrePost 算法与 H_PrePost

运行时间进行比较,然后构建 Hadoop 平台对 H_PrePost 算法从负载均衡和加速比两个方面进行验证. 实验环境均为 ubuntu 14. 04 操作系统, intel core i5 处理器,主频 2. 70 GHz, 4 GB 内存, 500 G 硬盘.

4. 1 单机条件下算法比较

本节使用 Accidents 数据集,通过采用不同的支持度对 PrePost 算法、改进 PrePost 算法与 H_PrePost 算法在单机条件下的运行时间进行比较. 实验结果如图 3 所示.

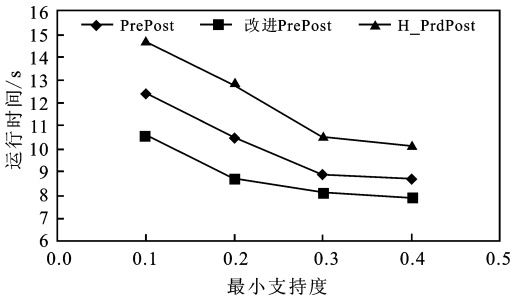


图 3 算法运行时间比较

从图 3 中可以看出,随着支持度的增加三种算法的运行时间都随之减少,这是因为随着支持度增加,满足要求的项集随之减少,因此算法运行时间会逐步降低. 但是在支持度升高到一定程度时,运行时间会趋于平衡,这是因为满足要求的项集不再减少或减少非常有限,对运行时间影响很小. 在这三种算法中,改进 PrePost 算法效率最高,其运行时间小于 PrePost 算法,验证了改进算法的有效性. H_PrePost 效率较低的原因是 Hadoop 平台节点间需要进行通信,占用了一定的计算资源. 这表明在数据量较小时,适合使用改进 PrePost 算法, H_PrePost 更适合于挖掘大数据集中的频繁模式.

4. 2 Hadoop 集群环境下性能测试

本节使用 webdocs 数据集从加速比和负载均衡两个方面对 H_PrePost 算法进行验证. 集群加速比定义为单节点算法执行时间与多节点执行时间的比值. 加速比越高,集群效率越高. 算法执行时间取决于集群中运行时间最长的节点,因此确保负载均衡是并行计算的一个重点. 负载均衡处理的越好,集群效率同样更高.

为了验证集群的加速比,将 webdocs 数据集进行随机抽取,得到 3 个不同的数据集 web1、web2、web3,其大小分别为 100 MB、200 MB、300 MB. 通过改变集群中的节点数量计算其加速比,结果如图

4 所示。

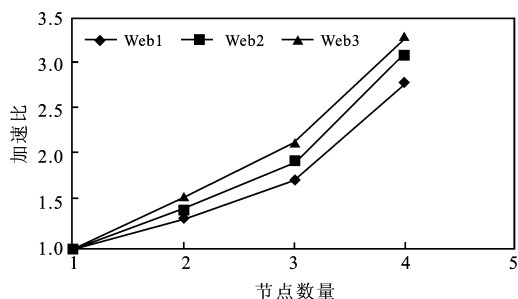


图4 算法加速比

从图4可以看出,加速比并不是严格理论上的线性关系,这是因为集群中节点之间的通信、任务调度等原因会消耗一定的时间.随着节点数目的提升,加速比逐步提升,最终接近线性关系.从图中还可以看出,随着数据规模的扩大,加速比也相应提高,这是因为此时集群中的节点的应用比较充分.实验结果说明了 H_PrePost 算法具有较好的加速比,具有良好的可扩展性.

为了验证 H_PrePost 算法中负载均衡策略的有效性,webdocs 数据集随机抽取 400 M 数据对含有负载均衡策略的 H_PrePost 算法与不含有负载均衡策略的 H_PrePost 算法进行比较,采用不同的支持度来进行实验,结果如图5所示.

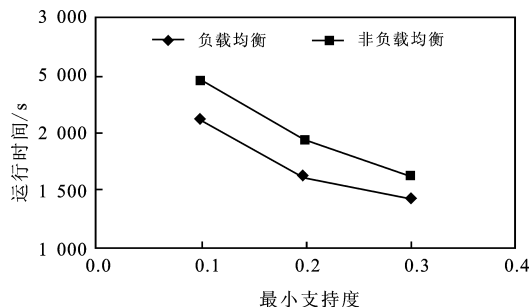


图5 负载均衡效果

从图5可以看出,在相同支持度条件下,采用负载均衡策略的 H_PrePost 运行时间少于不含负载均衡的 H_PrePost 算法且在三种不同的支持度条件下均是如此.因此, H_PrePost 算法采用的负载均衡策略可以有效保障集群稳定高效运行,挖掘出大数据中的频繁模式.

5 结束语

本文提出了一种高效挖掘大数据中频繁模式的算法 H_PrePost. 首先从压缩数据库、简化数据表示以及采用高效的连接和剪枝策略等方面对 PrePost

算法进行改进,然后结合 Hadoop 平台进行并行频繁模式挖掘.实验结果表明, H_PrePost 算法具有较好的加速比和负载均衡,可以有效挖掘出大数据中的频繁模式.

参考文献:

- [1] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]//Proc. 20th int. conf. very large data bases, VLDB. Zurich, Switzerland, 1994(1215): 487-499.
- [2] Song W, Yang B, Xu Z. Index-BitTableFI: An improved algorithm for mining frequent itemsets [J]. Knowledge-Based Systems, 2008, 21(6): 507-513.
- [3] Sahoo J, Das A K, Goswami A. An efficient approach for mining association rules from high utility itemsets [J]. Expert Systems with Applications, 2015, 42(13): 5754-5778.
- [4] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]// Acm Sigmod International Conference on Management of data. Dallas, Texas: ACM Press, 2000, 29(2): 1-12.
- [5] Pyun G, Yun U, Ryu K H. Efficient frequent pattern mining based on linear prefix tree [J]. Knowledge-Based Systems, 2014(55): 125-139.
- [6] 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.
- [7] Dam T L, Li K, Fournier-Viger P, et al. An efficient algorithm for mining top-rank-k frequent patterns[J]. Applied Intelligence, 2016, 45(1): 96-111.
- [8] Qibing Q, Long T. An effective hybrid algorithm for fast mining frequent itemsets[C]//Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2016 International Conference on. Zhengzhou, China, IEEE, 2016: 91-95.
- [9] Aji A, Wang F, Vo H, et al. Hadoop gis: a high performance spatial data warehousing system over mapreduce [J]. the VLDB Endowment, 2013, 6(11): 1009-1020.

作者简介:

李校林 男,(1968-),高级工程师,硕士生导师.研究方向为大数据、移动通信.

杜 托(通讯作者) 男,(1993-),硕士研究生.研究方向为大数据、数据挖掘. E-mail: dutuotuo@yeah. net.

谢 勇 男,(1994-),硕士研究生.研究方向为分布式计算、数据挖掘.