

基于 Bigtable 与 MapReduce 的 Apriori 算法改进

魏 玲 魏永江 高长元

(哈尔滨理工大学管理信息系统重点实验室 哈尔滨 150000)

摘 要 为提高 Apriori 算法挖掘频繁项目集的效率,引进了 Bigtable 技术与 MapReduce 模型来对 Apriori 算法进行优化,设计出大数据环境下挖掘频繁项目集的新算法 BM-Apriori 算法。与单纯基于 MapReduce 模型的 Apriori 改进算法相比,新算法利用 Bigtable 的时间戳属性代替了键/值对的产生,只需扫描数据库一次即可,节约了模式匹配的时间。同时,BM-Apriori 算法在项集列表中新增事务标号列,自动获取事务标号以计算支持度。将 BM-Apriori 算法在 Hadoop 平台上进行了实验,结果表明 Bigtable 技术的融入使得 BM-Apriori 算法具有更高的效率与可拓展性。

关键词 Apriori 算法, Bigtable, MapReduce, 大数据

中图法分类号 TP301.61 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.042

Improved Apriori Algorithm Based on Bigtable and MapReduce

WEI Ling WEI Yong-jiang GAO Chang-yuan

(Key Laboratory of Management Information System, Harbin University of Science and Technology, Harbin 150000, China)

Abstract BM-Apriori algorithm was designed for big data to address the poor efficiency problem of Apriori in mining frequent item sets. BM-Apriori takes advantages of Bigtable and MapReduce together to optimize Apriori algorithm. Compared with the improved Apriori algorithm simply based on MapReduce model, timestamp of Bigtable is utilized in this algorithm to avoid generating a large number of key/value pairs. It saves the pattern matching time and scans the database only once. Also, to obtain transaction marks automatically, transaction mark column is added to set list for computing support numbers. BM-Apriori was executed on Hadoop platform. The experimental results show that BM-Apriori has higher efficiency and scalability.

Keywords Apriori algorithm, Bigtable, MapReduce, Big data

1 引言

随着 IT 技术与物联网、互联网、云计算三网融合的飞速发展,数据爆发式的增长向 IT 行业发起了挑战。传统的关联规则挖掘技术已经不能满足大数据的需求,技术的融合与改进成为了诸多学者关注的焦点。数据挖掘是指从大量的数据中提取有用的、潜在的、有着特殊关联性的知识的过程^[1],它是知识发现 KDD 最重要的步骤^[2],其中最为经典的是 Apriori 算法。

Apriori 算法最初于 1994 年由 Agrawal^[3] 提出。通过反复扫描数据库,Apriori 算法逐层搜索候选项集,最终目的在于迭代地产生频繁项集。由 Apriori 算法的先验性质可知,频繁项集的所有非空子集一定是频繁的^[4]。利用该性质,Apriori 算法的精髓可概括如下:在第 k 次扫描时产生出长度为 k 的频繁项集 L_k ,通过 L_k 自身连接而产生候选集 C_{k+1} ;在进行第 $k+1$ 次扫描时统计 k 项频繁项集 L_k 产生长度为 $k+1$ 的候选集 C_{k+1} 所发生的次数,直到无法再找到频繁项集。可见,每找出一个频繁项集 L_k ,需要完整地扫描一次数据库;并且,这样产生的候选项集也极为庞大。例如,长度为 1000 的

频繁集 $\{x_1, x_2, x_3, \dots, x_{1000}\}$,将会产生 21000 个候选项集。由此可知,Apriori 算法的时间消耗主要存在于以下 3 个方面:

(1) 频繁项集自身连接生成候选项集时,所产生的候选项集数目庞大。

(2) 执行支持度计算,挖掘频繁项集时,须对海量数据库进行多次全局扫描。

(3) 扫描数据库对候选项集与事务进行模式匹配时,时间开销巨大。

为了弥补以上 Apriori 算法效率低的缺陷,国内外学者选取了各式技术与 Apriori 算法融合改进,诸如基于 MapReduce、基于布尔矩阵^[5]、基于压缩事务矩阵相乘^[6]、基于十字链表^[7]、基于 HMT 和哈希树的改进算法^[8]等。然而,面对大数据,这些算法从不同角度上又增加了新的负荷,影响了 Apriori 算法效率的大幅提高。例如:基于压缩事务矩阵相乘的改进算法在矩阵相乘上花费了较多的时间;基于十字链表的改进算法需要进行模式匹配,限制了算法效率的提高;基于 HMT 和哈希树的算法,在大数据的环境下,若没有一台足够强大的计算机支撑数据分析,串行算法则无能为力;MapRe-

到稿日期:2014-10-15 返修日期:2014-12-25 本文受国家自然科学基金资助项目(71272191)资助。

魏 玲(1965—),女,博士,教授,主要研究方向为信息管理与数据库, E-mail: weiling@hrbust.edu.cn; 魏永江(1990—),男,硕士生,主要研究方向为信息管理; 高长元(1960—),男,教授,博士生导师,主要研究方向为决策支持系统。

duce 算法生成了大量的键/值对,增添了 MapReduce 系统负荷,使得算法执行效率不高。大数据环境下,以上 Apriori 改进算法时间效率都不尽理想。云计算为提高海量数据挖掘效率提供了新思路。MapReduce, Bigtable 以及 GFS 同为 Google 云计算技术三宝。一部分学者应用 MapReduce 思想对 Apriori 进行了改进^[9]。MapReduce 高效的搜索与查询功能,及特有的 Map 和 Reduce 两个函数属性,使得 Apriori 算法能够快速生成候选项集,解决了上述 Apriori 算法的瓶颈问题(1)。因而,在诸多学者的 Apriori 改进算法中,本文选取了基于 MapReduce 模型的 Apriori 算法来进一步改进。

本文引用云计算中的 Bigtable 技术作为 MapReduce 模型的框架对原始 Apriori 算法并行化处理,继承 MapReduce 模型的优势,同时运用 Bigtable 技术三维属性加以改进,全面解决了上述 Apriori 算法的时间消耗的问题(2)与问题(3)。最后,将改进后的算法在 Hadoop 平台上进行实验,实验结果表明 Bigtable 技术的融入,使得 Apriori 算法运行效率有了质的飞跃。

2 经典 Apriori 算法与云计算技术的融合

2.1 经典 Apriori 算法描述

Apriori 算法是挖掘关联规则频繁项集的最经典的算法。利用频繁项集的先验性,使用逐层搜索的迭代方法,扫描数据库得到事务产生的全部频繁项集。算法描述如下。

输入:事务数据库 D,最小支持记数阈值 minsupport

输出:D 中事务生成的频繁项集集合 L

步骤:

1. $L_1 = \text{find_frequent_1-items}(D)$;
2. For($k=2; L_{k-1} \neq \emptyset; k++$) {
3. $C_k = \text{apriori_gen}(L_{k-1})$; //由 $(k-1)$ 项频繁项集 L_{k-1} 生成 k 项频繁候选项集
4. For each transaction $t \in D$ //扫描 D 进行计数
5. $\text{countCt} = \text{subset}(C_k, t)$ //得到候选项集的子集,即候选项集
6. For each candidate $c \in C_t$
7. $c.\text{count}++$;
8. $L_k = \{c \in C_k | c.\text{count} \geq \text{minsupport}\}$;
9. return $L = L \cup L_k$;

2.2 MapReduce 框架下的 Apriori 算法

MapReduce 的主体思想是“分而治之”,即先部分,后整体。MapReduce 模型的精髓可概括为系统将大数据集分割成小数据块,Map 函数并行处理各小块数据集,并交给 Reduce 函数进行归约。Map 函数将输入的一行数据构造按键值对 $\langle \text{key}, \text{value} \rangle$ 并按 key 值排序。Reduce 函数处理阶段是将 key 值相同的键值对合并,并输入到 Hadoop 上并行化处理,从而进一步提高算法速度^[10]。

MapReduce 主要用于生成候选项集。用户可以在云计算平台 Hadoop 上声明一个 Combiner,规约 Map 函数输出的结果,进而提高 MapReduce 整体运行速度。根据 Apriori 的性质,前 $k-2$ 项相同的两个 $(k-1)$ -频繁项集相连接产生 k -候选项集。因此,本文将 $(k-1)$ -频繁项集的前 $k-2$ 项作为键值,最后一项作为值,由 Map 函数输出。Reduce 函数将键值相同的数据规约到同一个 Reduce 函数中,快速地实现连接

操作,该方法在生成 K_3 的候选项集时效率提高得尤为明显,解决了引言中的问题(1)。例如,有这样 4 个频繁项集: $\langle A, B, C \rangle, \langle A, B, D \rangle, \langle A, B, E \rangle, \langle A, C, D \rangle$, 经由 Map 函数处理,输出的结果为: $\langle \langle A, B \rangle, C \rangle, \langle \langle A, B \rangle, D \rangle, \langle \langle A, B \rangle, E \rangle, \langle \langle A, C \rangle, D \rangle$, MapReduce 系统自动将其排序,生成的结果为: $\langle \langle A, B \rangle, [C, D, E] \rangle, \langle \langle A, C \rangle, [D] \rangle$, 此结果作为 Reduce 的输入,经由 Reduce 函数处理后,输出的结果为: $\langle A, B, C, D \rangle, \langle A, B, C, E \rangle, \langle A, B, D, E \rangle$, 此结果即为新生成的候选项集。 $\langle \langle A, C \rangle, [D] \rangle$ 由于并没有与其前两项相同的项集,因此没有生成新的候选项集。

2.3 引入 Bigtable 数据模式的改进思想

单纯基于 MapReduce 的 Apriori 算法策略,在大数据环境下运行时,统计候选项集的支持度会生成海量的 $\langle \text{itemset}, 1 \rangle$ 键/值对,不但耗时,而且对运行设备有很高的要求。Bigtable 是用于大规模数据的并行处理的分布式计算模型,也是多维度排序 Map。其三维分别是行键、列键和时间戳。在 Bigtable 表中,可以存储每一项数据的不同版本,并以时间戳属性予以区分和索引。时间戳可由系统自动分配,也可由用户赋值。Bigtable 在生成频繁项集与候选项集表时,自动生成事务项列。利用其独有的时间戳属性可以取代 MapReduce 中 $\langle \text{itemset}, 1 \rangle$ 键/值对的产生,从而实现扫描一次数据库即可,且无需重新返回事务数据库进行模式匹配,大大缩减了算法消耗的时间^[11]。

同时, Bigtable 的表会根据行键自动划分为片 Tablet, 其负载均衡的功能使得数据片被均匀地分配到各个节点上并行处理,读取效率大大增加,为大数据的繁杂海量的数据提供了便捷处理的平台^[12]。通过 Wrapper 类, Bigtable 作为 MapReduce 框架的输入和输出,更加高效地解决了引言中 Apriori 算法效率低的问题(2)与问题(3)。

3 基于 Bigtable 与 MapReduce 并行的 BM-Apriori 算法设计

3.1 基于 Bigtable 与 MapReduce 并行的 Apriori 算法详述

以表 1 中的事务数据库 D 为实例来说明算法。设最小支持度 $\text{minsupport}=2$ 。

表 1 事务数据库

TID	Items
1	ACD
2	BCE
3	ABCE
4	BE

步骤 1:生成候选 1-项集 C_1 ,如表 2 所列,计算支持度,删除项集中小于 minsupport 的记录,得到频繁 1-项集 L_1 ,如表 3 所列。数据进入 Bigtable 时系统为每个数据自动生成时间戳。时间戳可以由客户端赋值。本文设定每个事务的时间戳为其事务代码,便于下文利用时间戳作为索引来检索数据。例如,事务 1 的时间戳为 1。在生成候选项集 1-项集 C_1 时, BM-Apriori 算法在表中增加了每个项集所在的事务的标号列 TID。因而,在之后生成新的频繁项集时无需反复扫描事务数据库 D 来计算支持度及模式匹配,只需扫描数据库一次即可。

表 2 候选 1-项集 C_1

Items	Sup	TID
A	2	1,3
B	3	2,3,4
C	3	1,2,3
D	1	1
E	3	2,3,4

表 3 频繁 1-项集 L_1

Items	Sup	TID
A	2	1,3
B	3	2,3,4
C	3	1,2,3
E	3	2,3,4

步骤 2:由频繁 1-项集 L_1 生成候选 2-项集 C_2 、频繁 2-项集 L_2 。表 3 的 Items 列项自身连接,生成表 4 所列候选 2-项集 C_2 的 Items 列项。由于 Bigtable 为每个事务的时间戳赋予了与事务标号相同的值,因此在查找每个项集存在于哪些事务中时,只需调用时间戳索引,搜索生成该项集的原项集的事务,时间戳值相同的即为该项集所在的事务。得到事务代码后,计算表 4 的 TID 列每项的个数,结果即为该项集的支持度。例如,表 4 项集中的 AC 项,由于 AC 项由 A 项与 C 项连接生成,因此,返回到表 3 中找到 A 和 C 所在行的 TID 列, A 项的事务代码为 1,3,C 项的事务代码为 1,2,3。调用时间戳索引,将时间戳值相同的事务提取出来,即 1,3 为 AC 的事务代码,支持度为 2。将表 4 中小于 minsupport 的项集删除,生成表 5 所列的频繁 2-项集 L_2 。

表 4 候选 2-项集 C_2

Items	Sup	TID
AB	1	3
AC	2	1,3
AE	1	3
BC	2	2,3
BE	3	2,3,4
CE	2	2,3

表 5 频繁 2-项集 L_2

Items	Sup	TID
AC	2	1,3
BC	2	2,3
BE	3	2,3,4
CE	2	2,3

步骤 3:连接并剪枝,生成表 6 所列的候选 3-项集 C_3 以及表 7 所列的频繁 3-项集 L_3 。利用前面提及的方法,对表 5 的项集进行连接,生成表 6 的 Items 列。在 MapReduce 模型中,表 5 的数据进入 Map 函数,项集第 1 项作为键值,最后一项作为值,由 Map 函数输出。Reduce 函数将键值相同的数据规约到同一个 Reduce 函数中,快速地实现连接操作。表 6 的 Items 列生成后,重复步骤 2,实现支持度以及事务项的计算。剪枝后,表 7 生成。表 7 不能再生成新的候选项集,计算结束。

表 6 候选 3-项集 C_3

Items	Sup	TID
ABC	1	3
ACE	1	3
BCE	2	2,3

表 7 频繁 3-项集 L_3

Items	Sup	TID
BCE	2	2,3

3.2 改进后的算法伪代码

输入:数据源路径 inputdatapath,最小支持度 minsup

输出:K-项频繁集存入路径 k_outputpath

Setup()

k=1;

outputpath=GetFirstFreq(inputdata-path,k,trans_num,minsup);

//获得一项频繁项集集合,并将结果放入路径 outputpath 中

candidatesetlist//读取 k-频繁项目集;

Map(key1=frequent(k-1)_itemset,value1=support_count)

P=frequent(k-1)_itemset;

output(key2=(p[0],p[1],...,p[k-2]),value2=p[k-1]);//将读取的频繁项集的前 k-2 项作为键值,最后一项作为值,由 Map 函数输出

Reduce(key2=(p[0],p[1],...,p[k-2]),iterable(value2)=iterable(value2))

Item[]=iterable(value2);

sort(item);//对项集按行排序

for i:0 to item.length do

for j:i+1 to item.length do

itemset=(p[0],p[1],...,p[k-2],item[i],item[j]);//生产新的(k+1)-候选项集

TIDlist=foreach(linkitem in ItemsList)

{linkitem[i].timestamp=linkitem[j].timestamp}

suppot_count=count(TIDlistset);//时间戳相同的检索出来生成事物列项,并计算事物数,即支持度

k_outputpath(key3=itemset,value3=suppot_count);

end

end

4 实验结果分析

4.1 实验环境

在 Linux 环境下建立 Hadoop 集群,Hadoop 平台版本为 2.2。本实验运行的云计算环境共有 15 台服务器,共 1 个主节点,20 个从节点,每个节点的处理器为英特尔(Intel)22 纳米酷睿 i5 四核处理器,4GB 内存。操作系统均采用 Ubuntu 12.04,JDK 版本为 Sun JDK 1.7。本文使用某超市销售数据作为实验数据集,共有 1.6GB,共 2492070 条事务,53 个不同的项,其中最长的事务有 42 项。本文将改进后的基于 MapReduce 和 Bigtable 技术的 BM-Apriori 算法在数据集上进行了实验,同时将其与基于 MapReduce 的 Apriori 的算法进行了实验比较,说明了 BM-Apriori 算法性能的优越性。

4.2 实验结果与分析

图 1 分别展示了基于 Bigtable 和 MapReduce 技术的 BM-Apriori 并行算法和基于 MapReduce 技术的 Apriori 算法在挖掘 K-频繁项集时的时间消耗的比较。由图 1 可知,随着挖掘频繁项集项目数的增加,基于 MapReduce 技术的 Apriori 算法的时间消耗迅速上升,远大于基于 MapReduce 和 Bigtable 技术的 BM-Apriori 并行算法,体现了改进算法的优越性。这是因为,基于 MapReduce 技术的 Apriori 算法在计算候选项目集支持数时,产生了大量的<itemset,1>键/值对。随着挖掘频繁项集项数的增加,候选项集的数量增加的速度甚至呈幂指数增长,尤其在处理海量数据时,大大增加了 MapReduce 框架的负担。例如,如果一个候选项集最后的支持数为 1000,那么该候选项集将产生 1000 个的<itemset,1>键/值对,

(下转第 243 页)

dings of the 48th Annual Meeting of the Association for Computational Linguistics, 2010, 2010: 384-394

- [9] Brown P F, deSouza P V, Mercer R L, et al. Class-based n-gram models of natural language[J]. Computational Linguistics, 1992, 18(4): 467-479
- [10] 刘远超, 王晓龙, 徐志明, 等. 文档聚类综述[J]. 中文信息学报, 2006, 20(3): 55-62
- Liu Yuan-chao, Wang Xiao-long, Xu Zhi-ming, et al. A survey of

document clustering[J]. Journal of Chinese Information Processing, 2006, 20(3): 55-62

- [11] Zhang Y, Lin H, Yang Z, et al. Biomolecular event trigger detection using neighborhood hash features[J]. Journal of Theoretical Biology, 2013, 318(2): 22-28
- [12] Miwa M, Sætre R, Kim J-D, et al. Event extraction with complex event classification using rich features[J]. Journal of Bioinformatics & Computational Biology, 2010, 8(1): 131-146

(上接第 210 页)

再交给 Reduce 规约为 1000。而基于 MapReduce 和 Bigtable 技术的 BM-Apriori 并行算法是将时间戳相同的数据项检索出来生成事物列项, 扫描一次数据库即可完成候选项集与事务模式匹配, 同时自动计算生成事务项对应的支持度, 显著提升了运行效率。

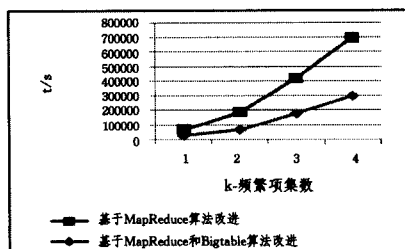


图 1 K-项频繁项集的时间消耗

图 2 为基于 MapReduce 和 Bigtable 技术的 BM-Apriori 算法和基于 MapReduce 技术的 Apriori 改进算法在不同数目节点上运行的结果。从中可以看出, 节点数目越多, 计算所需时间越短。这是因为, Bigtable 会根据行键自动划分为片 Tablet, 主服务器具有自动均衡节点功能, 可以高效有序地使所划分的片分布到不同的服务器节点上, 并行执行算法, 提高运行效率。但是随着节点数增加, 运行时间也会增长, 最终会使系统性能达到瓶颈。当节点数达到 6 时, 再增加节点数对减少运行时间已没有多大帮助。从图中可以看出, 在相同节点的情况下, 基于 MapReduce 和 Bigtable 技术的 BM-Apriori 算法比基于 MapReduce 技术的 Apriori 改进算法运行时间短, 体现了改进算法的优越性。

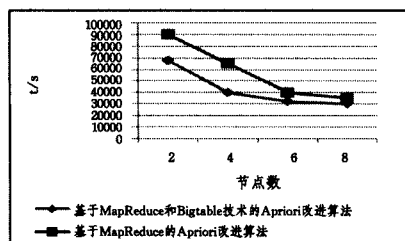


图 2 2 种算法在不同节点下的运行时间

结束语 本文对基于 MapReduce 的 Apriori 改进算法进行了研究整合, 在此基础上首创引入 Bigtable 技术实现的基于 MapReduce 与 Bigtable 技术并行的 BM-Apriori 算法, 并在 Hadoop 集群环境进行了实验与测试。实验证明, 该算法只需扫描一次数据库, 即可完成候选项集的模式匹配, 并避免了大量的 (itemset, 1) 键/值对的产生, 并行分布技术又进一步提升了 Apriori 算法的运行效率。逐个击破了 Apriori 算法时间消耗的症节点。在数据爆炸式增长的时代, BM-Apriori 算法将分布资源的计算能力整合, 对大数据具有高效的处理挖掘功能, 这为进一步研究 Apriori 数据挖掘算法的高效执行提供了

有力的支持, 具有良好的研究价值和应用前景。

参考文献

- [1] Hajian S, Domingo-Ferrer J. A methodology for direct and indirect discrimination prevention in data mining [J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(7): 1445-1459
- [2] Lara J, Lizcano D, Martinez A, et al. A UML profile for the conceptual modeling of structurally complex data: Easing human effort in the KDD process [J]. Information and Software Technology, 2014, 56(3): 335-351
- [3] Agrawal R, Imielinski T, Swami A. Database mining: a performance perspective [J]. IEEE Transactions on Knowledge and Data Engineering, 1993, 5(6): 914-925
- [4] 张震, 汪斌强, 陈庶樵, 等. 基于多维计数型布魯姆过滤器的大流检测机制[J]. 电子与信息学报, 2010, 32(7): 1608-1613
- Zhang Zhen, Wang Bin-qiang, Chen Shu-qiao, et al. A Mechanism of Identifying Heavy Hitters Based on Multi-dimensional Counting Bloom Filter[J]. Journal of Electronics & Information Technology, 2010, 32(7): 1608-1613
- [5] Wang B L, Shen Y G. Improvement of Apriori algorithm based on boolean matrix [J]. Advanced Materials Research, 2011, 159: 144-148
- [6] 罗丹, 李陶深. 一种基于压缩矩阵的 Apriori 算法改进研究[J]. 计算机科学, 2013, 40(12): 75-78
- Luo Dan, Li Tao-shen. Research on improved Apriori algorithm based on matrix compression [J]. Computer Science, 2013, 40(12): 75-78
- [7] 李晓虹, 尚晋. 一种改进的新 Apriori 算法[J]. 计算机科学, 2007, 32(4): 196-197
- Li Xiao-hong, Shang Jin. An improved Apriori algorithm [J]. Computer Science, 2007, 32(4): 196-197
- [8] Grudzinski P, Wojciechowski M. Integration of candidate hash trees in concurrent processing of frequent itemset queries using Apriori[J]. Control and Cybernetics, 2009, 38(1): 47-65
- [9] Jongwook W. Market Basket Analysis algorithms with MapReduce [J]. Wiley Interdisciplinary Reviews-Data Mining and Knowledge Discovery, 2013, 3(6): 445-452
- [10] Karim R, Hossain A, Rashid M, et al. A MapReduce Framework for Mining Maximal Contiguous Frequent Patterns in Large DNA Sequence Datasets [J]. IETE Technical Review, 2012, 29(2): 162-168
- [11] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data [J]. ACM Transactions on Computer Systems, 2008, 46(2): 205-218
- [12] Kim W. Web data stores (aka NoSQL databases): a data model and data management perspective [J]. International Journal of Web and Grid Services, 2014, 10(1): 100-110