基于 Spark 的 Apriori 并行算法优化实现

王 青[1], 谭 良[1,2], 杨显华[3]

(1. 四川师范大学 计算机科学学院 四川 成都 610101; 2. 中国科学院 计算技术研究所 北京 100190;
3. 四川省计算机研究院 四川 成都 610041)

摘要: 针对传统 Apriori 算法处理速度和计算资源的瓶颈,以及 Hadoop 平台上 Map-Reduce 计算框架不能处理节点失效、不能友好支持迭代计算以及不能基于内存计算等问题,提出了 Spark 下并行关联规则优化算法. 该算法只需两次扫描事务数据库,并充分利用 Spark 内存计算的 RDD 存储项集. 与传统 Apriori 算法相比,该算法扫描事务数据库的次数大大降低;与 Hadoop 下 Apriori 算法相比,该算法不仅简化计算,支持迭代,而且通过在内存中缓存中间结果减少 I/O 花销. 实验结果表明,该算法可以提高关联规则算法在大数据规模下的挖掘效率.

关键词: Spark; 并行化; 数据挖掘; 关联规则; Apriori

中图分类号: TP301.6 **文献标志码**: A **文章编号**: 1671-6841(2016)04-0060-05

DOI: 10.13705/j.issn.1671-6841.2016667

0 引言

关联规则挖掘是用来描述事物之间的联系和挖掘事物之间的相关性,它是在数据库中搜索两个项目之间存在的显示或者隐式关系,有助于管理和决策. Apriori 算法是最为经典的关联规则挖掘算法,该算法的核心是生成最大项目集,通过迭代方式逐层搜索频繁项集,直至没有更大项目集生成,但每次搜索都需要完整地扫描一次数据库,这种传统串行方式效率非常低. 随着云计算技术的发展,Hadoop 在分布式集群环境下对离线批处理作业表现出优势,但由于其处理数据必须先存储后运算,不能同时进行并行化操作,影响数据处理的实时性. 而 Spark 拥有 Map-Reduce 框架所有的优点,且所有计算结果都可以保存在内存中,它的快速数据处理能力可以有效减轻海量数据下发现挖掘任务的压力,提高迭代运算的效率. 基于 Spark 下的并行 Apriori 算法可以解决传统关联规则算法遇到的难题、单一并行化计算模式的瓶颈以及 Hadoop 平台不能很好支持迭代计算的缺陷. 因此,本文结合 Spark 计算平台,提出了基于 Spark 的 Apriori 并行优化算法,提高了关联规则算法在大数据规模下的挖掘效率.

1 相关工作

为了提高 Apriori 算法的性能,文献[1]在最大项集和闭项集的基础上,提出了元项集挖掘算法,减少频繁项集结果的冗余;文献[2]构建了基于领域知识的项相关性模型,简约划分数据库并映射至一种压缩树形结构中,缩小事务规模;文献[3]利用缓存数据库提高 Apriori 算法的效率. 这些算法在事务集小且事务维度不高的情况下,能发挥较好的作用. 但随着事务集越来越大、事务维度越来越高,上述算法性能明显降低.

随着云计算技术和大数据分析处理技术的兴起,为了提高挖掘效率,Apriori 算法优化主要围绕并行化进行研究[4],包括 MPI 并行化以及基于 Hadoop 平台的并行化研究. 文献[5-6]把云计算技术的两个重要步骤 Map 和 Reduce,分别引入到 Apriori 算法的连接和剪枝步骤中,并对优化算法进行 Map-Reduce 模型并行化,达到了 Apriori 算法并行化的目的. 但 Apriori 算法需要多次迭代才能发现频繁项集,当采用 Hadoop 并行

收稿日期:2016-07-23

基金项目:国家自然科学基金资助项目(61373162);四川省科技支撑项目(2014GZ007).

作者简介:王青(1992—),女,湖南衡阳人,硕士研究生,主要从事大数据处理与分析、数据挖掘以及机器学习研究;通讯作者:谭良(1972—),男,四川成都人,教授,主要从事可信计算、网络安全以及云计算和大数据处理研究,E-mail: tanliang2008cn@126.com.

引用本文:王青,谭良,杨显华. 基于 Spark 的 Apriori 并行算法优化实现[J]. 郑州大学学报(理学版),2016,48(4):60-64.

化的 Apriori 算法时,需要为每次迭代产生一个新的 Map-Reduce 去读取 HDFS 上的中间结果,产生额外负载. 文献[7]提出了将 Apriori 基于 Spark 进行并行化实现的 YAFIM 算法,解决了基于 Hadoop 并行化存在的编程模式问题,性能明显提高,但 YAFIM 算法也存在经典 Apriori 算法本身的一些问题. 文献[8]提出了 Spark 平台上并行化的 R-Apriori 算法,但 R-Apriori 算法仅通过优化 YAFIM 算法的第二次迭代过程提高 YAFIM 的效率,仍然存在额外的 I/O 负载. 因此,进行基于 Spark 的 Apriori 算法并行化优化具有研究意义.

2　基于 Spark 的 Apriori 算法优化(SP-Apriori)

2.1　Apriori 算法简介

Apriori 算法的主要思想是通过迭代的方法逐层搜索,用$(K-1)$项集去搜索大于最小支持度的 K 项集,直到没有满足条件的$(K+1)$项集生成. 对于事物 A、B,规则是否有效是由支持度 $s_{\text{support}}(A\rightarrow B)=P(A\cup B)$决定. Apriori 算法具体步骤如下:

输入:数据集 Datasets;最小支持度阈值 $m_{\text{min\_support}}$.

输出:K-项频繁集 L_K;

1) 首次扫描 Datasets 生成候选集 C_1,通过逐层扫描统计候选集中每个项集 X 的支持度 s_{support},删除 $X.s_{\text{support}}<m_{\text{min_support}}$的项集得到频繁集 L_1.

2) 频繁集 L_1再进行自身连接生成候选集 C_2,再次通过逐层扫描 Datasets,删除 $X.s_{\text{support}}<m_{\text{min_support}}$的项集得到频繁集 L_2.

3) 对 $K>2$ 的每个候选集 C_K,重复 2),最终得出最大频繁项集 L_K.

可以看出,算法效率非常低下,主要存在以下问题:① 资源消耗大. 算法每次搜索都需要完整扫描一次数据库,挖掘海量数据时,CPU 时间和内存消耗问题更加突出;② 规则挖掘模型较复杂. 单一方式搜索候选集,挖掘海量数据时,候选集数量巨大,产生候选集模型无法适应大数据环境.

2.2　基于 Spark 的 Apriori 算法优化过程

2.2.1　*Apriori 算法的改进*　对 Apriori 算法进行了如下改进:在挖掘过程中,利用频数表示支持度,易于比较并减少频繁计算支持度概率;利用组合策略得到总的规则类别,便于获得各项集 k_{key};利用此算法的两个重要性质(① 若 X 是频繁项集,则 X 的所有子集是频繁项集;② 若 X 是非频繁项集,则 X 的所有超集都是非频繁项集)去掉多余项集 k_{key}来压缩搜索空间. 改进 Apriori 算法的步骤描述如下:

1) 扫描事物数据库得到所有 1-item 项集 K 个,以及事物总数 n_{nums}.

2) 对各个 1-item 进行计数,记录频数最大的 i_{item}并去除产生 1 项候选集 C_1.

3) 根据业务需求和经验设置关联规则阈值:$m_{\text{min\_support}}$(最小支持度),即最小支持频数为 $m_{\text{min\_sup}}=n_{\text{nums}}*m_{\text{min\_support}}$.

4) 令 $i=1$,i 作为搜索第 i 项集的迭代控制变量,满足 $i<K$.

5) 对于 i 到 K,搜索第 i 项集,可以预测 i 项集总类别为C_K^i.

6) 所有候选集 C_i频数 $n_{\text{num_}L_i}$满足规则$(n_{\text{num_}L_i}>m_{\text{min_sup}})=>1$ 项频繁项集 L_i.

7) 如果 $n_{\text{num\_}L_i}<m_{\text{min\_sup}}$,去掉该项集,以及包含该子项集的 L_{i+1}-L_K项集.

8) 去掉频繁集 L_i中频数最小的 i-item,产生有趣第 i 项频繁集 F_i,令 $L_i=F_i$.

9) 对 L_i进行趋势(平稳、下降、上升、随机)分析 $=>L_i$,更新项集并存储,$i++$.

10) 逐次迭代 5)~9)直到产生 K 项候选集 C_K,如果存在 $K+1$ 项候选集,则继续迭代执行,如果不存在,则最终得到有趣 K 项频繁集 L_K,产生关联规则.

对于步骤 5),改进 K-项集的发现方法,将采用组合策略生成所有 K-项集,即可发现所有的 K-项集频度,K-项集总的个数为:$C_K^1+C_K^2+\cdots+C_K^K=2^K-1$,这种方法可以产生所有可能的候选项集,为每次迭代的输入提供了候选结果集,减少了运行复杂度. 例如,针对事务 $T\{I_1,I_2,I_3\}$对应的项目集,那么 K-项集的个数为 $2^3-1=7$,K-项集与二进制对应关系如表 1 所示. 利用算法性质,当$\{I_1,I_2\}$发生 n 次时,对应 011,1-项集$\{I_2\}$,$\{I_3\}$也将同时发生 n 次,将对应的所有〈K-项集,n〉写入 transformation 操作的〈k_{key}, v_{value}〉.

表1 *K*-项集与二进制对应关系

Tab.1 The correspondence between *K*-item and binary

$T\{I_1,I_2,I_3\}$对应二项式	*K*-项集	计数 C_{Count}
001	$\{I_1\}$	$\langle\{I_1\},C_{\text{Count}}\rangle$
010	$\{I_2\}$	$\langle\{I_2\},C_{\text{Count}}\rangle$
100	$\{I_3\}$	$\langle\{I_3\},C_{\text{Count}}\rangle$
011	$\{I_1,I_2\}$	$\langle\{I_1,I_2\},C_{\text{Count}}\rangle$
101	$\{I_1,I_3\}$	$\langle\{I_1,I_3\},C_{\text{Count}}\rangle$
110	$\{I_2,I_3\}$	$\langle\{I_2,I_3\},C_{\text{Count}}\rangle$
111	$\{I_1,I_2,I_3\}$	$\langle\{I_1,I_2,I_3\},C_{\text{Count}}\rangle$

对于步骤5)~9),把传统算法抽象成循环迭代算法,每次搜索项集候选项集确定,迭代次数确定并小于*K*,它不仅减少了运行复杂度,且可以把每次搜索任务分摊到多个处理器上同时运行,便于并行化计算.

2.2.2 *基于Spark的Apriori算法并行化设计* Spark引入弹性分布式数据集RDD数据模型,并整合了内存计算基元,支持节点集群将数据集缓存在内存中,缩短了访问延迟.除了能够提供交互式查询外,还可以优化迭代工作负载,当需要反复操作的次数越多、读取的数据量越大时,相对于Hadoop,Spark在性能方面更适用于需要多次操作特定数据集的应用场合.Spark是Map-Reduce的扩展,它提供两类操作:transformation(得到新的RDD)和action(得到结果)多种API,不再需要使用Hadoop唯一DataShuffle模式,编写程序更具灵活性,使上层应用开发效率提升数倍.Spark大数据编程模型如图1所示.

图1 Spark大数据编程模型

Fig.1 Big data programming model of Spark

结合Spark特性,基于"分而治之"的思想,本文算法的并行化设计是把事物数据库均衡分发给多个子节点,以局部查找频繁项集、剪枝代替全局操作,避免全局查找出现内存无法容纳的问题,并且可以实时实现数据集计数、过滤支持度低的项集以及排序等,实现对整个挖掘频繁项集和生成规则以及评价规则等各个处理过程的并行化.并行化设计步骤如下:

1) Master利用Spark提供的算子$t_{\text{textFile}}()$扫描存储在HDFS上的事务数据库,即为一个RDD.

2) Worker利用$C_{\text{Count}}(r_{\text{rdd}},n_{\text{num}})$操作求1项集的集合$L_1$和候选1项集$C_1$.

3) RDD被平分成n个数据块,且这些数据块被分配到m个worker节点进行处理.

4) 根据worker节点上1-项Item,采用优化算法步骤7)的方式生成所有局部*K*-项集Part_ L_K.

5) 通过函数$f(i_{\text{iter}})=>i_{\text{iter}}.f_{\text{filter}}(_>=M_{\text{Max_}L_1})$对$w_{\text{worker}}$中的所有数据进行过滤.

6) 设置关联规则标准的阈值最小支持度$m_{\text{min_sup}}$.

7) 根据Part_ L_K生成局部支持度频数,利用局部剪枝性质,删除局部支持度频数小于局部支持度阈值的项.

8) 利用$m_{\text{map}}(w_{\text{worker}},C_K)$、$r_{\text{reduceByKey}}(w_{\text{worker}},C_K)$、$f_{\text{filter}}(w_{\text{worker}},C_K>m_{\text{min_sup}})$组合操作进行每一轮局部剪枝操作.

9) 针对剪枝触发提交job进行$f_{\text{foreachRDD}}(i_{\text{iter}}.\text{步骤}8)=>a_{\text{add}}(w_{\text{worker}},C_K)=>P_{\text{Part_}L_K}$局部连接,然后$u_{\text{union}}(\text{worker},P_{\text{Part_}L_K})=>C_K$进行全局连接.

10) 结合频繁项集时序性规则挖掘趋势进行filter(-,-)产生有趣频繁项集.

11) 全局$f_{\text{filter}}(C_K>m_{\text{min_sup}})$触发SparkContext产生有趣规则$L_K$.

以上t_{textFile},C_{Count},f_{filter},m_{map},$r_{\text{reduceByKey}}$算子都是Spark为用户编程提供的接口API,其中$f(i_{\text{iter}})$函数是自定义迭代函数,去除小于支持度的项集.

2.2.3 *基于Spark的Apriori算法的实现* 迭代式Apriori算法并行化实现的核心是迭代调用transformation和action操作,每次迭代中利用上一次迭代结果来进行求解,算法并行化实现步骤如下:

输入:数据源路径i_{inpath};最小支持度阈值$m_{\text{min_sup}}$.

输出:*K*-项频繁集;*K*-项频繁集输出路径*K*-outpath.

1) 获取总事务集$i_{\text{items}}=A_{\text{Apriori}}(i_{\text{inpath}})$//构造函数,对数据源进行预处理.

2) 获取总事务数$n_{\text{nums}}=g_{\text{getNums}}(i_{\text{items}})$//计算1项集总类别数.

3）获取 1 到 K-项集 K-items 集,去掉 $m_{\text{maxCount}}(i_{\text{items}})$的 1 项集合//计算得到最大 1 项集.

4）$K=1$.

5）$o_{\text{outpath}} = g_{\text{getFirstFreq}}(i_{\text{inpath}}, K, n_{\text{nums}}, m_{\text{min\_sup}})$//从 i_{inpath}获得所有 1 项集 L_1,并将产生的 $L_1 => C_2$输出到新的 K-outpath 中.

6）while(1){

K-outpath $= g_{\text{getKFreq}}(i_{\text{inpath}}, o_{\text{outpath}}, n_{\text{nums}}, m_{\text{min\_sup}})$

//通过数据源 i_{inpath}以及 L_1 获得 2-K 项集 L_2-L_K 结果集

如果 K-outpath 为空,则退出

否则:

$K=K+1$;

比对 K-items 集,去掉小于 $m_{\text{min_sup}}$项集;

o_{outpath} = K-outpath//作为下一次剪枝依据　}.

7）各计算节点将频繁模式 C_K增加趋势:$C_K = C_K -> t_{\text{trend}}(C_1, C_2, \cdots, C_K) => L_K$.

8）通过 u_{union}(K-outpath,$m_{\text{min\_sup}}$)汇集到 m_{master}节点,得出全局关联规则集合.

//子节点得到关联规则结果 => 全局关联规则结果.

3　实验和结果分析

3.1　实验环境

采用两台 PC 电脑,其中 1 台为 m_{master}节点,同时也作为 w_{worker}节点,另外 1 台为 w_{worker}节点,共 4 个节点,通过交换机组成一个局域网. 所用软件为 Intellij + Hadoop + Spark,分别实现了传统 Apriori 算法,Hadoop Map-Reduce 模式下 Apriori 改进算法(Mp-Apriori 算法),Spark RDD 模式下 Apriori 算法(S-Apriori 算法),Spark RDD 模式下 Apriori 改进算法(SP-Apriori 算法). 本实验数据由 IBM 数据生成器生成,由于实验硬件条件限制,数据量大小为 1.12 G,事务平均长度为 42 MB,共 100 个 i_{item}项集,包括约 100 万条事务数据记录.

3.2　实验结果

对 3.1 节数据进行随机采样,在支持度 0.75 下统计运行时间,采用子节点运行内存的 50% 来缓存 RDD,在此基础上开展两组实验. 实验一:采用传统 Apriori 算法以及保持 4 个节点不变的集群环境下的并行化 Mp-Apriori 算法、S-Apriori 算法和 SP-Apriori 算法,在挖掘数据集大小不同的情况下,计算各个算法的运行时间,结果如图 2 所示. 实验二:采用 100 万条数据集,增加一台机器,新增两个 w_{worker}节点,改变集群节点数目,测量节点可扩展性,分别测量节点数为 1, 2, 4, 6 时的 SP-Apriori 算法进行规则挖掘的执行时间,结果如图 3 所示.

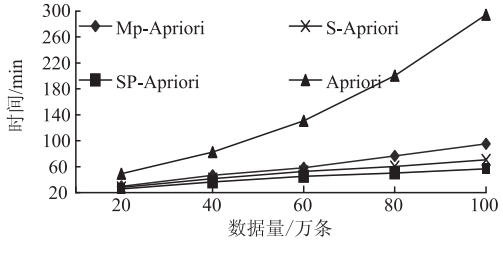

图 2　不同算法的运行时间

Fig. 2　The running time of different algorithms

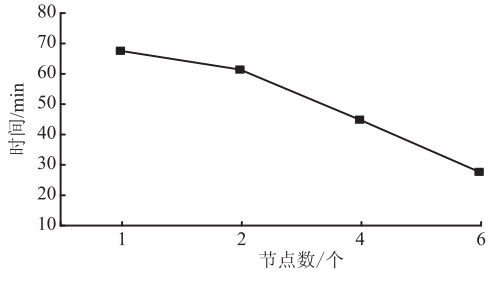

图 3　不同节点数的运行时间

Fig. 3　The running time of different workers

由图 2 可知,并行化算法比传统串行 Apriori 算法的效率更高,随着数据量的增加,并行化算法时间开销平稳增加,而传统串行 Apriori 算法时间开销成倍增加,说明相对于传统串行方式,并行化更适合大数据环境;当事务数据量不大时,基于 Spark 和 Hadoop 的算法运行时间差距不大,但随着事务数据量的增加,基于内存计算的 SP-Apriori 算法直接从内存中读取迭代时所需中间结果,大大减少了 Hadoop 计算时所需 I/O 读取时间,Spark 的优势越来越明显,改进的算法效果最好. 由图 3 可知,随着数据节点数增多,算法执行时间不

断缩短.数据节点也是影响算法效率的一个重要因素.因此,本文提出的优化对算法的性能有一定提高,同时随着节点数的增加、各节点内存容量变大以及对数据源进行预处理,算法的执行时间在理论上将大幅度减少.

4 小结

结合 Spark 计算平台,实现了一种基于 Spark 的并行 Apriori 优化算法,提高了处理海量数据的效率,适用于生产环境中对实时性要求较高的应用.由于没有事先对数据集进行预处理,无效数据过多,使得内存利用率降低;没有改变数据的存储结构,在实验过程中发现仍然有数据集本身数十倍甚至上百倍大小的中间结果需要保存在内存中.在接下来的研究中,将对算法的预处理和改变事务存储结构进行深入研究,并对并行过程进行严谨证明和理论推导,同时也会探讨 Spark 平台对实际应用场景的适用性,以期获得理想效果.

参考文献:

[1] 宋威, 李晋宏, 徐章艳, 等. 一种新的频繁项集精简表示方法及其挖掘算法的研究[J]. 计算机研究与发展, 2010, 47(2): 277 - 285.

[2] 毛宇星, 陈彤兵, 施伯乐. 一种高效的多层和概化关联规则挖掘方法[J]. 软件学报,2011,22(12):2965 - 2980.

[3] ASTHANA P, SINGH D. Improving efficiency of Apriori algorithm using cache database[J]. International journal of computer applications, 2013, 75(13):15 - 20.

[4] 陈玉哲,赵明华,李军,等. 基于移动 agent 和数据挖掘标准的分布式数据挖掘系统[J]. 郑州大学学报(理学版),2011,43(1):90 - 94.

[5] 伊瑶瑶, 茅苏. Hadoop 下的关联规则分析研究[J]. 计算机技术与发展,2015,25(9):84 - 88.

[6] 刘木林, 朱庆华. 基于 Hadoop 的关联规则挖掘算法研究:以 Apriori 算法为例[J]. 计算机技术与发展,2016,26(7):1 - 11.

[7] QIU H, GU R, YUAN C, et al. YAFIM: a parallel frequent itemset mining algorithm with Spark[C]// IEEE International on Parallel & Distributed Processing Symposium Workshops (IPDPSW). Phoenix, 2014: 1664 - 1671.

[8] YANG S, XU G, WANG Z, et al. The parallel improved Apriori algorithm research based on Spark[C]//9th International Conference on Frontier of Computer Science and Technology. Dalian, 2015:354 - 359.

Optimization of Apriori Parallel Algorithm Based on Spark

WANG Qing[1], TAN Liang[1,2], YANG Xianhua[3]

(1. *College of Computer Science*, *Sichuan Normal University*, *Chengdu* 610101, *China*;
2. *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*;
3. *Sichuan Institute of Computer Sciences*, *Chengdu* 610041, *China*)

Abstract: In view of the bottleneck of traditional Apriori algorithm in processing speed and computing resources, and that Map-Reduce on Hadoop could not handle node failures, friendly support iterative calculation, and calculate based on memory issues ,a parallel association rule optimization algorithm based on Spark was proposed. The optimization algorithm only needed to scan the transaction database twice and it took advantage of Spark's RDD storage structure. By comparing with the traditional Apriori and Apriori based on Hadoop, analysis showed that Apriori based on Spark more greatly reduced the number of scan database than that of traditional Apriori, and it used less I/O overhead than Apriori based on Hadoop, because it supported storing temporary results in memory and iterative calculation. Experimental results showed that Apriori based on Spark performed effectively on big data for mining association rules.

Key words: Spark; parallel processing; data mining; association rule; Apriori

(责任编辑:孔 薇)