·先进计算与数据处理· 文章编号:1000-3428(2016)10-0069-06 文献标志码:A 中图分类号:TP338.6

基于 Hadoop 平台的改进关联规则挖掘算法

王英博,马 菁,柴佳佳,赵 彬

(辽宁工程技术大学 软件学院,辽宁 葫芦岛 125105)

摘 要: 数据采集方式的增多导致单处理器下的关联规则挖掘受到 I/O 和内存的限制。针对该问题,对传统挖掘算法进行改进。借助 Hadoop 平台的优势,通过累加迭代的方法降低算法的时间复杂度,并利用 MapReduce 编程特点,通过一次遍历和 MapReduce 任务调度完成频繁项集挖掘,在强关联挖掘中通过 Sqoop 组件将外部表 Hive 中的数据迁移到 Redis,实现数据的高速读取。实验结果表明,该方法可有效提高挖掘效率,提高幅度随数据集规模同步增大,并且具有较好的加速比和扩展性。

关键词: Hadoop 平台;MapReduce 编程;关联规则;大数据;数据挖掘

中文引用格式: 王英博,马 菁,柴佳佳,等. 基于 Hadoop 平台的改进关联规则挖掘算法[J]. 计算机工程,2016,42(10):69-74,79.

英文引用格式: Wang Yingbo,Ma Jing,Chai Jiajia,et al. Improved Association Rule Mining Algorithm Based on Hadoop Platform[J]. Computer Engineering,2016,42(10):69-74,79.

Improved Association Rule Mining Algorithm Based on Hadoop Platform

WANG Yingbo,MA Jing,CHAI Jiajia,ZHAO Bin

(School of Software,Liaoning Technical University,Huludao,Liaoning 125105,China)

【Abstract】 Development of ways for data acquisition leads to limit of traditional association rule mining by I/O and memory. Aiming at this problem,this paper puts forward an improved method,which uses advantages of the Hadoop platform,reduces the time complexity of the algorithm by incremental iterative method,and makes full use of the features of MapReduce programming. It completes the frequent itemset mining through traverse and MapReduce task scheduling,which improves the efficiency of processing. In the mining of strong association,with the help of Sqoop,the external tables are migrated from Hive to Redis,which makes the data read more efficient. Experimental results show that the proposed method can improve processing efficiency. With the data increasing,the advance becomes more obvious,and improved algorithm also has better speedup and scalability,which is able to quickly mine the association rules in large data.

【Key words】 Hadoop platform;MapReduce programming;association rule;big data;data mining

DOI: 10.3969/j.issn.1000-3428.2016.10.012

1 概述

关联规则挖掘[1-2]是针对购物篮问题(Market Basket Analysis)提出的,通过对事务集频繁项集的挖掘进而发现事物之间的关联规则,在零售业、金融、电子商务等诸多领域都得到了广泛应用。

目前处在一个数据爆炸的大数据时代,据不完全统计,淘宝网每天新增的交易数据可达 10 TB,eBay 平台每天处理约 100 PB 数据量,YouTube 每分钟上传 60 h 的视频数据[3],在处理超大规模的数据时,单处理器下的关联规则算法受到 I/O 和内存的限制,近年来,不少专家学者开始探究使用分布式计算的方法来改进该算法。文献[4]提出了分布式并行关联规则挖掘算法,针对分布式应用数据架构全局频繁模式树(Frequent Pattern tree,FP-tree)可能过大而内存无法容纳的问题,提出只产生局部 FP-tree 的算法,在各个主要步骤上都实现了并行处理,显著提高了算法的执行效率和处理能力。文献[5]针对序列算法在关联规则挖掘时节点之间沟通成本过高的问题提出了解决方案。这些算法都在一定程度上提高了分布式关联规则挖掘算法的效率,但是无法避免节点失效给计算带来的问题。

作者简介: 王英博(1964—),男,教授、博士,主研方向为数据挖掘、数字化矿山、大数据;马 菁、柴佳佳,硕士研究生;赵 彬,硕士。

收稿日期: 2015-10-29 **修回日期:** 2015-12-04 **E-mail:** 231767911@qq.com

Hadoop[6]是 Apache 软件基金会旗下的一个开源分布式计算平台。以分布式文件系统 HDFS[7]和 MapReduce[8]为核心的 Hadoop 平台提供了一个底层细节透明的分布式基础架构。Hadoop 通过自动将失败的任务重新分配,有效解决了节点失效给计算带来的问题。Hadoop 平台因其具有可靠性、扩展性、高效性、容错性等优点,在处理大数据的各个领域得到了广泛应用。目前,基于 Hadoop 平台的关联规则挖掘已有一些研究,文献[9]针对单处理器内存和 CPU 受限的问题提出了改进的基于 MapReduce 的先验算法,有效地提高了 Hadoop 上关联规则算法计算效率。文献[10]通过构建事务数据库分块索引和动态负载均衡的方法来提高关联规则算法在 Hadoop 平台的处理能力。文献[11]通过 MapReduce 编程框架实现了分布式 CD(Candidate Distribution,候选集分布)算法,有效解决了分布式挖掘关联规则算法存在的瓶颈,但仅考虑频繁项集的计算,且需要产生多次 MapReduce 任务,严重影响了关联规则的挖掘效率。

针对以上问题,本文提出利用 Hadoop 生态系统来改进关联规则算法。通过 2 个 MapReduce 任务完成频繁项集和强关联规则的挖掘,采用 Hive 外部表和 Redis 数据库实现数据的存储,使用 Sqoop 进行数据迁移,借助 Oozie 实现算法流程的调度。

2 相关技术

2.1 关联规则

设 E 是所有项目元素的集合,$E=\{e_1,e_2,\cdots,e_n\}$。数据集合 $D=\{d_1,d_2,\cdots,d_n\}$,数据集合 D 中的任意一个元素 d_i 都是由集合 E 中元素构成的集合,且 $d_i\subseteq E$。

定义 1 集合 A 由 E 中的 k 个元素构成,若数据集合 D 中满足 $A\subseteq d_i$ 的个数为 m,则集合 A 在数据集合 D 中的支持度定义为 $sup(A)=m/n$。若 $sup(A)$ 大于用户指定的最小支持度阈值 s,则集合 A 为数据集合 D 的频繁 k 项集。

定义 2 若集合 A 和集合 B 满足 $A\subseteq E\cap B\subseteq E$,则 $A\rightarrow B$ 的置信度为 $sup(A\cup B)/sup(A)$。

关联规则的目的就是挖掘数据集合中满足用户设定的最小支持度和最小置信度的 $A\rightarrow B$。由此关联规则挖掘算法主要分为 2 步:(1)挖掘满足最小支持度的频繁项集合。(2)根据频繁项集合挖掘满足最小置信度的强关联规则。

2.2 基于 Hadoop 实现的关联规则算法

根据文献[11]中的理论和实验对比可得,在众多的并行关联规则算法中,CD 算法的性能最好,但是 CD 算法有 2 个严重的缺陷,即该算法不允许任何一个计算节点失败,否则将导致整个计算失败;若计算节点的速度存在差异较大时,计算时间由计算性能差的节点决定,这样会导致计算资源的浪费,影响了运行效率。

在 Hadoop 平台上运行 MapReduce 程序时,若某一个节点计算失败,Hadoop 系统会把它的计算任务重新分配到其他计算节点。Hadoop 平台通过资源管理器可以合理地分配资源,达到负载均衡,有效解决计算资源浪费的问题。文献[11]通过借助 Hadoop 平台的 MapReduce 编程框架实现了 CD 算法,解决了上述缺陷,其主要计算步骤如下:

(1)主进程通过($k-1$)项频繁项集产生 k 项候选集,并分发给每个节点。

(2)每个工作节点通过 Map 函数计算其读取的部分数据中的 k 项候选集的支持数。

(3)Reduce 进程把 Map 进程获取的 k 项候选集支持数合并获取全局 k 项候选集的支持数。

(4)主进程根据 k 项候选集的支持数获取($k+1$)项频繁项集。

(5)通过 k 次迭代计算频繁项目集。

基于 Hadoop 平台实现的 CD 算法,有效解决了分布式挖掘关联规则算法存在的瓶颈,但是该算法仍然存在以下缺陷:

(1)该算法并未完全结合 Hadoop 平台的特点,MapReduce 编程框架无法将中间结果保存在内存中,只能保存到 Hadoop 分布式文件系统(Hadoop Distributed File System,HDFS)中,同时每次提交计算任务都需要重启资源,所以,MapReduce 编程框架并不适合多次迭代,而 CD 算法需要产生多次 MapReduce 任务,严重降低了关联规则的挖掘效率。

(2)该算法相对于非并行化关联规则算法,在效率和稳定性上都有了明显的提高。然而该算法仍然存在一个缺陷,就是需要多次遍历原始数据集。当处理海量数据集时,将严重影响处理数据的效率。

(3)该算法仅考虑了频繁项集的计算,并未考虑强关联规则的挖掘。当处理大规模数据集时,同样需要大量的计算来挖掘强关联规则。

因此,本文针对这些缺陷,在基于 Hadoop 实现的 CD 算法(以下称该算法为改进前算法)的基础上进行了改进,具体的改进方案在第 3 节中进行介绍。

3 基于 Hadoop 的改进关联规则算法

针对 2.2 节中所述缺陷,本文设计一种新的结合整个 Hadoop 平台和 MapReduce 特点的关联规则挖掘思路,具体分为以下两部分:频繁项集的挖掘和强关联规则的挖掘。

3.1 频繁项集挖掘

利用 Map 函数计算项目集的子集,利用 Reduce

函数汇总所有子集的支持度,进而计算频繁项的支持度,实现数据集中的频繁项集挖掘,具体执行流程如下:

输入　原始数据集 D 输入路径,最小支持度 α

输出　满足最小支持度的频繁项集 HDFS 文件

Map 任务:

步骤 1　根据输入文件的路径,从 HDFS 上读取原始数据集合并将其水平划分 N 个规模相当的数据子集,对数据子集每一行进行格式化,产生 $<Key, Value>$ 键值对。*Key* 值为其字符偏移量,*Value* 为其数据内容。

步骤 2　Map 函数读取数据子集的每个 $<Key, Value>$ 键值对,利用 split 函数将 *Value* 内容解析到集合中。

步骤 3　计算集合的所有子集作为输出 *key*,将每个子集的 *Value* 值设为 1,即在 Map 函数中将频繁项集的候选集的支持度记为 1。

步骤 4　调用可选的 Combine 函数。在大规模数据中,每个 Map 端都会产生大量 *Key* 值相同的键值对,若将这些键值对都通过网络传送到 Reduce 端,会耗费大量的网络资源,增加系统的 I/O,降低算法的执行效率。通过调用可选的 Combine 函数,在 Map 端合并相同 *Key* 的键值对,从而减少数据在网络的传输量,减少 Reduce 端计算候选集支持度的工作量,提高算法的执行效率。

Reduce 任务:

步骤 1　Hadoop 平台会自动对 Combine 函数提交的键值对进行排序,将 *Key* 值相同的合并形成 $<Key, List(Value)>$,Reduce 函数读取 $<Key, List(Value)>$,对 List 集合中值进行累加就可得到 *Key* 集合在数据集合 D 的支持数,即在 Reduce 端实现频繁候选项集全局支持度的计算。

步骤 2　根据最小支持度,将大于等于最小支持度的候选项集及其支持度输出到 Hive[12] 外部表中,其中 Hive 是基于 HDFS 文件系统之上的数据仓库架构,用户可以通过 HiveQL(Hive SQL)查询频繁项集,同时由于 Hive 是基于 HDFS 文件系统之上的,所以外部表 Frequent_items 文件也是分布式文件系统的 HDFS 文件,可以直接作为 MapReduce 程序的输入或输出文件。

核心伪代码如下:

```
map(Object key,Text value,Context context)
throws IOException {
for(each value)//读取 1 行数据
GetSubsets(value)//计算该行数据元素子集
context.write(keytext,new IntWritable(1));
//输出集合的所有子集并将其 value 值设为 1
}
reduce(Text key,Iterable < IntWritable > values,Context
context) throws IOException {
    for(IntWritable value:values){
    sum = sum + 1;}//计算每个 Key 对应 List(Value)的和
    sup = sum/alldata;//计算该 Key 值对应的支持度
    context.write(key,Sup);
    }
```

设原始集合 $s = <a, b, c>$,求其子集过程为:第 1 次迭代:$<a>$,第 2 次迭代:$<a, ab, b>$,第 3 次迭代:$<a, ab, b, ac, abc, bc, c>$。每次迭代,都是上一次迭代的结果加上当前迭代的元素。第 k 次迭代的迭代次数为 $2^k - 1$。若集合元素的个数为 M,则需要迭代 M 次,总的遍历次数为 $2^{M+1} - (2 + M)$,即其时间复杂都为 $O(2^M)$。若有 N 行数据,则本文提出的频繁项集挖掘算法的时间复杂度为 $O(2^{MN})$,而改进前算法的时间复杂度为 $O(M!\ N)$。同时本文提出的算法仅需提交一次 MapReduce 任务,仅需遍历一次原始数据集,每条数据仅需读入内存一次,频繁项目子集在内存中计算完成。而改进前的算法需要提交 M 个 MapReduce 任务,每次只能计算一个子集,其需要遍历 M 次数据,M 次在 HDFS 上进行数据存储和读取,而 Hadoop 平台并不适合多次提交任务,严重降低了频繁项集的挖掘效率,因此,相对于改进前的算法,本文算法可大幅提高频繁项集的挖掘效率。

3.2　强关联规则挖掘

挖掘强关联规则就是就是挖掘频繁项目集中满足最小置信度的元素组合,由 2.1 节中置信度的定义可知,频繁项集的子集和其差集之间关联度的置信为两者支持度的比值。如若频繁项集 $\{A, B\}$ 的支持度为 a,$\{A\}$ 的支持度为 b,则 $A -> B$ 置信度为 a/b,即在计算置信度时需要实时获取频繁项集子集的支持,而这些频繁项集的子集都是存储 Hive 外部表中的频繁项集,这就要求在计算频繁项集的置信度时需要实时获取其他频繁项集的支持度。为了解决这个问题,本文选择通过 Hadoop 系统中的数据迁移工具 Sqoop[13] 组件将 Hive 外部表中的频繁项目集数据导入支持高速读取的 Redis[14] 数据库中,从而实时获取任何频繁项集支持度。

结合 MapReduce 编程模型,强关联规则算法的具体步骤如下:

输入　将 Hive 外部表 Frequent_items 的存储路径,最小置信度 β

输出　满足最小置信度 β 的强关联规则

Map 任务:

步骤 1　Map 函数在执行之前会自动调用其 setup 方法,本文通过 setup 方法获取 Redis 数据库的连接。

步骤 2　将 Hive 表中的频繁项集文件水平划分 N 个规模相当的数据子集,对数据子集进行格式化,

将每行数据解析成 < *Key*, *Value* > 键值对, *Key* 值为其字符偏移量, *Value* 为其数据内容, 即频繁项目集和其支持度的组合字段。

步骤 3 将 *Value* 中频繁项集字段元素解析到集合中, 列如其 *Value* 值为(*A*, *B*, *SupValue*), 则将元素(*A*, *B*)存储到集合中。

步骤 4 计算频繁项集集合元素的子集 *A*, 在 Redis 数据库读取该子集的支持度 *sup*(*A*), 频繁项子集与该频繁项集集合种除该子集元素之外元素间的置信度即为该频繁项集支持度值与 *sup*(*A*)的比值, 即*A* -> *B*之间的置信度为 *SupValue*/*sup*(*A*)。

步骤 5 若置信度大于设定的最小置信度, 则该子集与该频繁项集集合种除该子集元素之外元素的之间为强关联规则, 将子集和差集组合为一个 *Key* 值, 置信度值为 *Value* 保存到输出文件中, 其核心伪代码如下所示:

```
setup(Context context) {
key_sup = RedisUtils. findMap("Frequent_items");
//获取 Redis 数据库数据的连接池
}
map(Object key,Text value,Context context)
throws IOException,InterruptedException {
for(each value);//读取一行数据;
String a[] = Split(value);//解析频繁项集和其支持度
subset = GetSubsets(a[0…lemhth -2]);//计算子集
sup = key_sup. get(subset);//获取该子集支持度
con = sup/a[length -1];
//计算频繁项集子集与其差集之间的置信度
difset = filter(a,subset);//计算该子集的差集
context. write(new Text(subset + ' -> ' + difset),
newDoubleWritable(con));}
```

这种强关联规则的挖掘方法充分利用了 Redis 将全部的数据存储在内存中的特点, 在执行频繁项集支持度获取的事务时, 与之前的直接从 Hive 外部表的读取相比效率明显提高。

4 实验结果与分析

在本文设计的关联规则挖掘系统中需要产生不同的工作调度。为了对流程进行监控, 对处理过程和结果进行监督, 优化处理算法的执行过程, 引入 Hadoop 生态系统的工作流调度工具 Oozie[15], 通过 Oozie 保证了系统的快速高效运行。

4.1 实验环境和数据

本文实验环境是在 Linux 环境下搭建 Hadoop 集群, 共 4 台计算机, 其中 1 台作为提供 Resource Manager, NameNode 服务的 Master 节点, 其他 3 台作为 DataNode 节点, 3 台机器安装的操作系统都为 CentOS-6.4-x86_64, 为简化集群的安装过程和实现运算算过程可视化, 通过 CM5.1.1 搭建 CDH5.1.3。实验环境架构如图 1 所示。

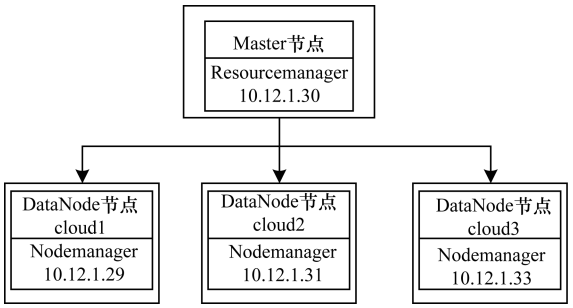

图 1 Hadoop 集群架构

本文实验数据是通过 Weka 中提供的关联数据集 weather, 通过 Java 程序将数据属性值随机组合自动生成了 4 个数据集, 如表 1 所示。

表 1 本文实验数据

数据集	文件大小	数据记录行数/10^6
C	197 MB	1
D	395 MB	10
E	1.93 GB	50
F	3.86 GB	100

4.2 效率对比实验

效率对比实验在改进前算法与本文所提出的改进后的关联规则挖掘算法上进行, 比较 2 个算法在处理相同数据集时的运行效率, 其中设置参数最小支持度 $s = 0.5$, 实验结果如图 2 所示。

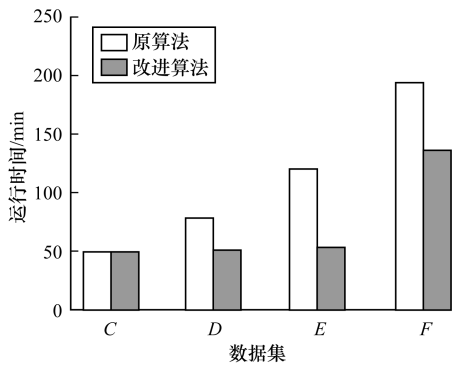

图 2 集群对比实验结果

从图 2 可以看出, 本文改进算法在运行效率上明显高于改进前算法, 且随着数据量的增大其效率差会进一步增大, 其原因是本文算法降低了 MapReduce 程序启动的次数, 减少了中间结果 I/O, 进而提高了算法的执行效率。同时本文算法仅需遍历一次原始数据, 而改进前算法需要多次遍历原始数据, 随着数据规模的增大, 其效率差将会进一步增大。

4.3 集群扩展性和加速比实验

集群的扩展性是指随着数据量的增大, 当运算性能遇到瓶颈时, 是否可以通过增加节点的方式来提高吞吐量, 进而提高运行效率。集群是否具有良好的扩展性体现了该算法是否能够有效处理海量

数据。

集群加速比是指针对同一个数据集在串行处理和并行处理运行消耗的时间比率,可以用来衡量并行系统或程序并行化的性能和效果[16]。计算公式如下:

$$Sp = T_1/Tp \tag{1}$$

其中,Sp 表示加速比;T_1 是单节点串行的运行时间;Tp 是在有 p 个节点并行处理的运行时间。加速比越大表明并行算法的执行效率越高。

集群的扩展性和加速比实验可以通过一个实验来验证,其内容为统计数据集(C,D,E,F)在不同节点下的运行时间,结果如图 3、图 4 所示。

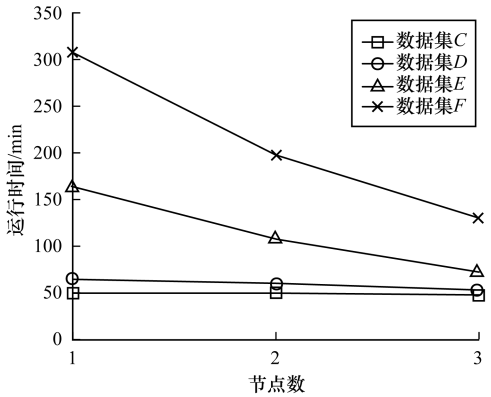

图 3　集群的扩展性实验结果

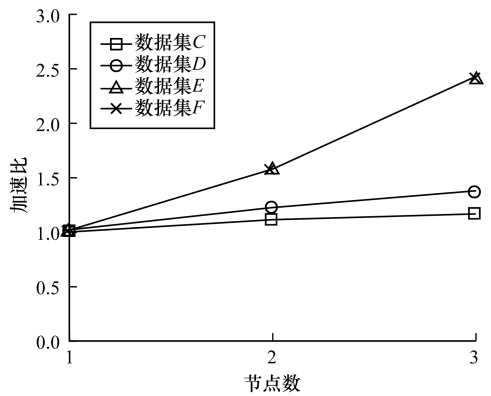

图 4　集群加速比

从图 3 中可以看出,本文算法在数据量小于 395 MB时,随着并行节点的增加其运行效率并没有明显的提高。而当数据量达到 1.93 GB 时,随着并行节点的增加其运行效率有了明显了提高,表明集群针对大规模数据数据集具有良好的扩展性,也说明了基于 Hadoop 平台改进的关联规则挖掘算法适合处理大规模数据集。

从图 4 可以看出,随着并行节点的增加,不同数据集的加速比都在增加。当数据量达到 1.93 GB 时,其加速比可以达到 2.5,这说明基于 Hadoop 平台实现的关联规则的挖掘有效提高了算法的执行效率。

4.4　集群优化实验

集群优化的目的是在硬件不变的情况下,通过调整集群的参数来提高集群的吞吐量,从而提高集群的处理数据的效率,使性能达到最优。

Hadoop 平台运行 Map 或者 Reduce 任务都是在 Java 虚拟机(Java Virtual Mechine,JVM)中运行的,默认情况下每一个 Map 或者 Reduce 任务都启动一个新的 Java 虚拟机。若在一个非常短的任务结束之后让后续的任务重用此虚拟机,就可以省下新任务启动 Java 虚拟机的时间,进而提高算法的执行效率。控制 JVM 重用的属性是 mapreduce. job. ubertask. enable,默认情况该属性为 false,将该属性值改为 true 则启用 JVM 重用。JVM 优化效果如图 5所示。

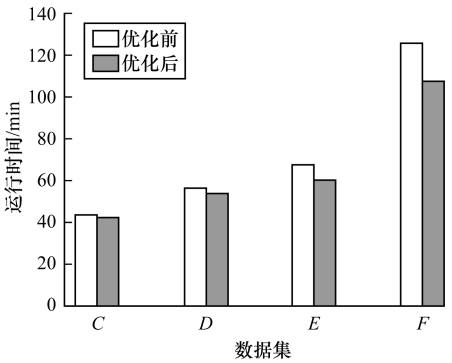

图 5　JVM 优化效果

从图 5 中可以看出,当数据量大于 1.93 GB 时,启动 JVM 明显加快了算法的运行效率,而数据量较小时则没有明显的优化效果,这是由于 Hadoop 平台是以块为单位进行存储和读取数据的,而 CDH5.1.3 默认情况下块为 128 MB。当数据规模较小时,Hadoop 不会产生太多的并行的任务,启动 JVM 重用不会产生优化的效果。块的大小可以通过在 MapReduce 程序中通过改变控制文件中的 mapred. max. split. size 属性值来改变块的大小,下面通过将块设为不同值来检验块大小对算法执行效率的影响,如图 6 所示。

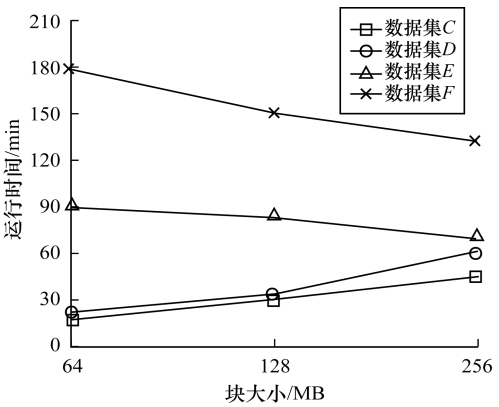

图 6　块大小对算法产生的影响

从图 6 可以看出块大小的设定对数据集(*C*,*D*)数据集(*E*,*F*)产生了相反的效果。这是因为针对小规模数据集(*C*,*D*),当块大小设定较小时则会产生更多的并行任务,提高了集群的利用率,进而提高算法的执行效率;而当块设置较大时则会无法发挥集群并行化的优势,浪费了集群资源,降低算法的执行效率。针对大规模数据集(*E*,*F*),当将块值设定较小则会产生超过集群容纳的任务,却增加了 Map 任务的分配过程以及运行作业所必须的寻址次数,增加了系统的总体消耗,降低了算法的执行效率,此时将块设定为 Hadoop 平台允许的最大值则会最大限度地发挥集群并行计算的优势,提高算法的执行效率。通过实验结果可以得出,基于 Hadoop 平台改进的关联规则挖掘算法可以根据需要处理数据量的大小来设定块的大小,进而最大限度地发挥集群并行计算的优势,提高算法的执行效率。

除了上述通过启动 JVM 重用和设置块大小的方式优化集群外,当 Hadoop 平台处理大规模数据集时,可以通过对文件进行压缩来优化集群。文件压缩一方面可以减少文件所需的存储空间,另一方面加快了文件在网络上或磁盘上的传输速度,文件压缩优化效果如图 7 所示。

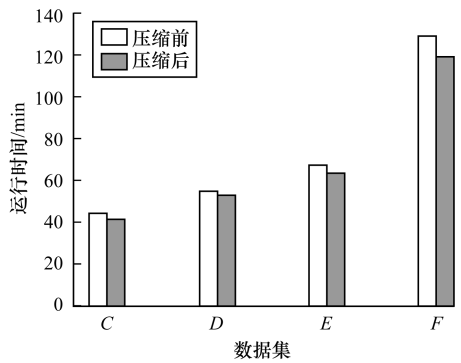

图 7 文件压缩优化效果

从图 7 可以看出,当数据集大于 2 GB 时,通过使用文件压缩明显提高了算法的执行效率,而对于数据量小于 400 MB 时,则不会产生优化的效果。从以上集群优化实验结果可知,Hadoop 平台需要根据处理数据的规模,采取不同的优化方案才能最大限度发挥集群的优势,使基于 Hadoop 平台改进的关联规则挖掘算法的性能达到最佳。

5 结束语

本文在对关联规则算法和基于 Hadoop 平台的 CD 算法研究的基础上,提出一种基于 Hadoop 平台改进的关联规则挖掘算法。该算法通过减少 MapReduce 任务提交的次数和元数据集遍历的次数来提高频繁项集挖掘的效率,并利用 Sqoop 组件将 Hive 外部表中的数据导入到支持高速读取的 Redis 中,增强了强关联规则挖掘时数据的实时性,从整体上提高了关联规则挖掘的效率。通过集群的扩展性和加速比实验,表明本文算法可以高效地处理大规模数据集。最后通过 JVM 重用、设置块大小以及文件压缩来优化集群,针对不同规模的数据集采取不同的集群优化方案,进而使算法的性能达到最佳。

本文提出的算法还存在改进空间,当面对超大规模的数据集时,Redis 数据库将无法支持强关联规则的挖掘,下一步的研究重点是利用 HBASE[17] 取代 Redis 数据库,提高系统对超大规模数据的处理能力。

参考文献

[1] Han Jiawei, Pei Jian, Yin Yiwen. Mining Frequent Patterns Without Candidate Generation[C]//Proceedings of ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2000:1-12.

[2] Agrawal R, Imielinki T, Swami A. Mining Association Rules Between Sets of Items in Large Database[C]//Proceedings of ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 1993:207-216.

[3] Borthakur D, Gray J, Sarma J S, et al. Aiyer: Apache Hadoop Goes Realtime at Facebook[C]//Proceedings of the 38th ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2011:1071-1080.

[4] 王智钢,王池社,马青霞. 分布式并行关联规则挖掘算法研究[J]. 计算机应用与软件,2013,30(10):113-119.

[5] Nguyen D, Vo B, Le B. Efficient Strategies for Parallel Mining Class Association Rules[J]. Expert Systems with Applications, 2014, 41(10):4716-4729.

[6] 陆嘉恒. Hadoop 实践[M]. 北京:机械工业出版社,2012.

[7] Lu Bingliang, Wei Shuchao. One More Efficient Parallel Initialization Algorithm of K-means with MapReduce[C]//Proceedings of the 4th International Conference on Computer Engineering and Networks. Berlin, Germany: Springer, 2014: 845-852.

[8] Mukhopadhyay D, Agrawal C, Maru D, et al. Addressing Name Node Scalability Issue in Hadoop Distributed File System Using Cache Approach[C]//Proceedings of 2014 International Conference on Information Technology. Washington D. C., USA: IEEE Press, 2014: 321-326.

[9] Yang Xinyue, Zhen Liu, Fu Yan. MapReduce as a Programming Model for Association Rules Algorithm on Hadoop[C]//Proceedings of the 3rd International Conference on Information Sciences and Interaction Sciences. Washington D. C., USA: IEEE Press, 2010: 99-102.

(下转第 79 页)

5　结束语

为提升实体信息集成效率,本文考虑结果相关性与重复度,提出一种新的深网数据源选择方法。基于内容主题与情感词构建数据源摘要,利用该摘要设计相应的最优数据源选择策略。实验结果验证了本文方法的有效性。未来将进一步研究面向数据质量的深网选择策略,以更好地满足深网数据集成的需要。

参考文献

[1] 李道申,刘　勇. 基于本体的 Deep Web 数据源发现方法[J]. 计算机工程,2012,38(4):52-54.

[2] 万常选,邓　松,刘喜平,等. Web 数据源选择技术[J]. 软件学报,2013,24(4):781-797.

[3] Balakrishnan R, Kambhampati S. Source Rank: Relevance and Trust Assessment for Deep Web Sources Based on Inter-source Agreement [C]//Proceedings of the 20th International Conference on World Wide Web. New York, USA:ACM Press,2011:227-236.

[4] Dong X L, Saha B, Srivastava D. Less Is More: Selecting Sources Wisely for Integration [C]//Proceedings of the 39th International Conference on Very Large Data Bases. [S.l.]:Morgan Kaufmann Publishers,2013:37-48.

[5] Rekatsinas T, Dong X L. Finding Quality in Quantity: The Challenge of Discovering Valuable Sources for Integration[C]//Proceedings of the 7th Biennial Conference on Innovative Data Systems Research. New York, USA:ACM Press,2015:1-7.

[6] 邓　松,万常选,刘喜平,等. 基于用户反馈的深网数据源选择[J]. 小型微型计算机系统,2012,33(11):2367-2371.

[7] Rekatsinas T, Dong X L. Characterizing and Selecting Fresh Data Sources [C]//Proceedings of 2014 ACM SIGMOD International Conference on Management of Data. New York, USA:ACM Press,2014:919-930.

[8] 范　举,周立柱. 基于关键词的深度万维网数据库的选择[J]. 计算机学报,2011,34(10):1797-1804.

[9] Wang Ying, Zuo Wanli, He Fengling, et al. Ontology-assisted Deep Web Source Selection [J]. Computer Science for Environmental Engineering and EcoInformatics, 2011,159(2):66-71.

[10] 万常选,邓　松,刘德喜,等. 面向混合类型关键词查询的非合作结构化深网数据源选择[J]. 计算机研究与发展,2014,51(4):905-917.

[11] Nguyen K, Cao J. K-Graphs: Selecting Top-k Data Sources for XML Keyword Queries [C]//Proceedings of the 22nd International Conference on Database and Expert Systems Applications. Berlin, Germany: Springer-Verlag,2011:425-439.

[12] 朱冠胜,黄　浩,杨卫东. XML 关键字检索系统的数据源选择[J]. 小型微型计算机系统,2012,33(6):1183-1188.

[13] Markov I, Azzopardi L, Crestani F. Reducing the Uncertainty in Resource Selection [C]//Proceedings of the 35th European Conference on IR Research. Berlin, Germany: Springer-Verlag,2013:507-519.

[14] Hong D, Si Luo. Search Result Diversification in Resource Selection for Federated Search [C]//Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, USA:ACM Press,2013:613-622.

[15] 于　娟,尹积栋,费　庶. 基于句法结构分析的同义词识别方法研究[J]. 现代图书情报技术,2013,29(9):35-40.

[16] 阳爱民,林江豪,周咏梅. 中文文本情感词典构建方法[J]. 计算机科学与探索,2013,7(11):1033-1039.

编辑　金胡考

(上接第 74 页)

[10] 杨　勇,高松松. 基于 MapReduce 的关联规则并行增量更新算法[J]. 重庆邮电大学学报:自然科学版,2014,26(5):670-678.

[11] 马　盈. 基于 MapReduce 构造多维数据及关联规则挖掘算法的研究与应用[D]. 长春:东北师范大学,2013.

[12] Thusoo A, Sarma J S, Jain N, et al. Hive: A Petabyte Scale Data Warehouse Using Hadoop[C]//Proceedings of the 26th International Conference on Data Engineering. Washington D. C., USA:IEEE Press,2010:996-1005.

[13] Ji Zhanlin, Ganchev I, O'Droma M, et al. A Distributed Redis Framework for Use in the UCWW[C]//Proceedings of 2014 International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery. Washington D. C., USA:IEEE Press,2014:241-244.

[14] Chen Changai, Jiang Shan. Research of the Big Data Platform and the Traditional Data Acquisition and Transmission Based on Sqoop Technology [J]. The Open Automation and Control Systems Journal, 2015, 15(7):1174-1180.

[15] 范　飞,黄文明,邓珍荣. Oozie 工作流在 Mahout 分布式数据挖掘中的应用[J]. 桂林电子科技大学学报,2014,34(1):25-29.

[16] Wasi-ur-Rahman M, Huang Jian, Jose J, et al. Understanding the Communication Characteristics in HBase: What Are the Fundamental Bottlenecks? [C]//Proceedings of 2012 IEEE International Symposium on Performance Analysis of Systems and Software. Washington D. C., USA:IEEE Press,2012:122-123.

[17] Lee C, Zheng Y. Automatic SQL-to-NoSQL Schema Transformation over the MySQL and HBase Databases[C]//Proceedings of 2015 IEEE International Conference on Consumer Electronics. Washington D. C., USA: IEEE Press,2015:426-427.

编辑　金胡考