基于 Hadoop 的 Apriori 改进算法研究

黄 剑[1] 李明奇[1] 郭文强[2]

(电子科技大学数学科学学院 成都 611731)[1] (新疆财经大学计算机科学与工程学院 乌鲁木齐 830012)[2]

摘 要 对于规模庞大的事务数据库,传统的并行 Apriori 算法在挖掘中会在数据 IO 上有较大的时间开销。从压缩事务、减少扫描次数、简化候选集生成3个方面对 Apriori 算法进行改进。提出了以元素"0"和"1"表示事务的布尔矩阵模型,并引入权值维度,压缩了相同事务的矩阵规模。同时,动态地进行剪枝,矩阵的"与"运算用于候选集合的生成。将改进后的算法在 Hadoop 框架上进行并行化实现,实验表明该算法适合大规模数据挖掘且具有良好的伸缩性与有效性。

关键词 Apriori 算法,事务数据库,布尔矩阵,Hadoop

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.07.046

Reseach on Improved Apriori Algorithm Based on Hadoop

HUANG Jian[1] LI Ming-qi[1] GUO Wen-qiang[2]

(School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu 611731, China)[1]

(School of Computer Science and Engineering, Xinjiang University of Finance & Economics, Urumqi 830012, China)[2]

Abstract Traditional mining based on parallel Apriori algorithms needs much more time in data IO with the increasing size of large transaction database. In this paper, we improved Apriori algorithm in three aspects: compression in the transaction, reducing the number of scanning areas, and simplifying the candidate set generation. We proposed "0" and "1" as the entries to describe the transaction Boolean matrix model, and introduced the weight dimensions to compress the matrix size of the transaction. Meanwhile, dynamic pruning matrix is adopted, and "and" operation of matrix is applied to generate a candidate set. The experiments of the improved algorithm running parallel in Hadoop framework show that the algorithm is suitable for large-scale data mining, and the algorithm has good scalability and effectiveness.

Keywords Apriori algorithm, Transaction database, Boolean matrix, Hadoop

数据挖掘是一个知识发现的过程,通过一定的算法获取隐藏于数据背后的规律[1]。当今互联网行业盛行,大公司积累的数据量巨大,例如国内百度已经达到数十 PB 的规模,传统的简单数据处理方式已经不能满足数据挖掘的需求,单机串行挖掘已经逐渐退出企业级应用的舞台。海量数据挖掘系统成为了新的发展趋势[2],能为企业更好地提供决策支持。

Apriori 算法[3-4]是 Agrawal 等人于 1994 年设计出的经典关联规则算法,是一种逐层次的搜索迭代方法,采用 k-项目集合来构成 $k+1$-项目集。该算法的核心思想为:先从事务数据库统计所有频集,该频繁集的支持度必须不小于最小支持度;然后进入到强关联规则生成过程,规则需要同时满足支持度和置信度阈值;后续使用第一步过滤符合期望的规则,保留只包含集合项的所有规则,一旦这些规则被保留生成,只有大于或者等于最小置信度的规则才能被保留下来。

Hadoop 框架[5]的设计源于 Apache 组织基金会开发的一个开源项目,因其具有跨时代的意义,在国内外信息领域得到了广泛应用。Hadoop 框架中具有重大意义的两大模块为[6]:分布式文件系统 HDFS 和分布式计算框架 MapReduce。HDFS 作为分布式文件系统,实现了数据存储的功能,它将协同计算框架 MapReduce 工作,为数据计算提供底层支持。MapReduce 思想[7-8]是由 Google 的一篇论文所提出的,简而言之,其核心方法就是"任务的分解与结果的规约"。

1 Hadoop 平台

HDFS 以 Master/Slave 作为架构,由 1 个管理节点(NameNode)和 N 个数据节点(DataNode)组成[9]。NameNode 提供 API 与 Client 的交互,在底层实现文件的 Block 分割,并存储于 DataNode。存储过程引入了冗余存储机制来实现容错容灾。Rack 是指机柜,1 个 Block 的 3 个副本通常会保存到 2 个或者 2 个以上的机柜中,其目的是防灾容错,因为发生一个机柜掉电或者其交换机挂掉了的概率相对较高。Client 先将数据写到第一个节点,在第一个节点接收数据的同时,又将其所接收的数据推送到第二个节点,第二个再将其推送到第三个节点,若存在多个节点则依次类推,其架构如图 1 所示。

到稿日期:2016-06-20 返修日期:2016-09-14 本文受国家自然科学基金(61163066)资助。

黄 剑(1988—),男,硕士,主要研究方向为数据挖掘及应用,E-mail:huangjian502871597@qq.com;李明奇(1970—),男,博士,副教授,主要研究方向为信息论及应用;郭文强(1975—),男,博士,教授,主要研究方向为计算机通信、信号处理。

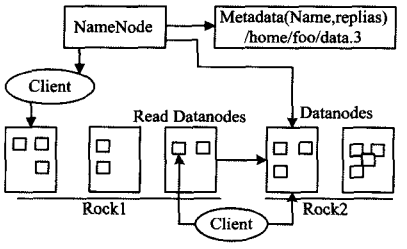

图 1　HDFS 的架构图

Mapreduce 框架[10]是高效的大规模数据计算模型，Map/Reduce 是其核心。一个文件被切分成小粒度的块文件，将这些块文件分配到对应的 Block 的同时为其分配一个 map 任务，所有任务执行完成后进入 shuffle 阶段[11]，然后根据哈希算法分配 reduce 任务，执行完成后输出最后的结果。

2　Apriori 算法

Apriori 算法是一种逐层次的搜索迭代方法，由 k-项目集合来构成($k+1$)-项目集。首先获取频繁 1-项目集合(简称为 L_1)，L_1 可以生成频繁的 2-项目集合 L_2，L_2 又可以生成频繁的 3-项目集合 L_3，按此规律，当不能找到频繁 k-项目集时，该算法结束[12-13]。

具体操作如下：

(1)遍历初始事务数据库，统计候选项集合出现的频次，其结果即为项目的支持度。所有支持度不低于支持度阈值的项目生成频繁 1-项目的集合 L_1。

(2)算法使用 L_1 jion L_1，形成候选 2-项目的集合 C_2。

(3)利用 C_2 中的项目，再次遍历数据库，可以获取到每个候选集合的支持度。所有的支持度不低于支持度阈值的项目生成频繁 2-项目的集合 L_2。

(4)算法使用 L_2 join L_2，形成候选 3-项集的集合 C_3。

(5)利用 C_3 中的项目，再次遍历数据库，可以获取到每个候选集合的支持度。所有的支持度不低于支持度阈值的项目生成频繁 3-项目的集合 L_3。

上述过程迭代执行，直到候选集合 C_k 为空为止。Apriori 算法会对数据库做多次 IO 操作，每个阶段都由两部分操作构成，即连接和剪枝。

3　基于 Hadoop 的 Apriori 改进算法设计

本文提出了一种 Apriori 的并行化改进算法 Apriori-TMC，其基本思想是遍历数据库中的每个事务，将事务数据库转换为矩阵，矩阵中的每行向量表示事务数据库中的一个事务，每列对应事务中的项。不用实际存储矩阵中的每一个项目，只需要用“0”或“1”表示该项目是否存在。这样所有项目都被保存在矩阵中，在后续进行频繁集的挖掘时只需要对矩阵进行内部运算。在结构化数据中存在较多冗余事务，采取引入权值的方式，以用一条向量代表多条交易记录，从而起到压缩事务数据库的作用。基于布尔矩阵，需要对矩阵做或运算来完成项目的拼接，有效地降低了时间复杂度。在候选集合生成过程中，对事务矩阵做动态剪枝处理，能够更大程度地减小挖掘的规模。

3.1　AprioriTMC 算法的描述

设项目 $I=\{I_1,I_2,\cdots,I_n\}$，事务数据库 $D=\{T_1,T_2,\cdots,T_m\}$。其中 T_i 表示事务数据库 D 中的第 i 个事务，其由 I 中的多个项目组成，即 $T_i\subseteq I$，并用事务标识号 T_id 来表示该事务。

定义 1　将每个项 I_j 的向量定义为：

$I_j=(t_{1j},t_{2j},\cdots,t_{mj})^{\mathrm{T}}$

D 中的事务 $T_i=\{t_{i1},t_{i2},\cdots,t_{in}\}$，其中事务 t_{ij} 表示项目 I_j 的存在情况，如果存在则有 $t_{ij}=1$；否则 $t_{ij}=0$。项目 I_j 的支持度 $support(I_j)=\sum_{i=1}^{m}t_{ij}$。

定义 2　事务数据库 D 可以转换成一个加权布尔矩阵：

$$D_{m\times(n+1)}=\begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1n} & w_1 \\ d_{21} & d_{22} & \cdots & d_{2n} & w_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} & w_m \end{pmatrix}$$

其中，$d_{ij}=0$ 或者 $d_{ij}=1(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$，$w_k$ 代表权值，其值等于相同事务的重复度。

定义 3　一项集合 $S_1=\{I_i\mid I_i\in I\}$，二项集合 $S_2=\{I_i,I_j\mid I_i,I_j\in I\}$，依次类推可以得到 k 项集。

定义 4　二项集$\{I_i,I_j\}$的向量定义为 D_{ij}：

$D_{ij}=D_i\wedge D_j=(d_{1i}\wedge d_{1j},d_{2i}\wedge d_{2j},\cdots,d_{mi}\wedge d_{mj})^{\mathrm{T}}$

符号“$\wedge$”表示“逻辑与”运算，“$\times$”表示逻辑与结果 0 或 1 与 w_k 相乘，计算顺序是先做“$\wedge$”运算，再做“$\times$”运算。因此，$\{I_i,I_j\}$的支持度为：

$$support\{I_i,I_j\}=\sum_{k=1}^{m}d_{ki}\wedge d_{kj}\times w_k$$

同理，k 项集$\{I_1,I_2,\cdots,I_k\}$的支持度为：

$$support\{I_1,I_2,\cdots,I_k\}=\sum_{s=1}^{m}(d_{s1}\wedge d_{s2}\wedge\cdots\wedge d_{sk}\times w_s)$$

多个项目做逻辑与计算时，只要有一个项目为零，逻辑与的结果就为零。

定理 1　若事务项 T_i 的项目包含的最大长度为 L，且 $L<2$，则可以将其所在的行向量删除。

证明：由于事务项 T_i 所产生的最大项目集合的长度为 k，因此不能产生 $k+1$ 项集合，可将其删除以减小数据库规模。

定理 2　存在项集 A 和 B，且 $A\subseteq B$，如果 B 是频繁集，则 A 也是频繁集合。

证明：设事务数据库中的最小支持度为 min_sup，B 为频繁集合，那么 B 的支持度 Sup_B 满足：$Sup\_B\geqslant Min\_sup$。在事务中，由于 $A\subseteq B$，显然有 $Sup\_A>Sup\_B$，因此有 $Sup\_A>Min\_sup$，故 A 也为频繁集合。

定理 3　若 k 阶频繁集 L_k 的个数 $L(k)$ 少于 $k+1$，则在事务数据库中不可能存在 $k+1$ 阶频繁集。

证明(反正法)：假设 $k+1$ 阶频繁集是存在的，根据定理 2，一定有该 $k+1$ 阶频繁集的所有 k 项子集都是频繁 k 项集。而 k 项集的个数为 $L(k)=C_{k+1}^{k}=k+1$，这与已知矛盾，因此不可能存在 $k+1$ 项频繁集。

推论 1　存在两个项目集 A 和 B 满足关系 $A\subseteq B$，若 A 是非频繁集，那么 B 也是非频繁集合。

证明：假设最小支持度为 min_sup，A 为非频繁集合，那么 A 的支持度 Sup_A 满足：$Sup\_A<Min\_sup$。在事务中，由于 $A\subseteq B$ 且显然 $Sup\_A>Sup\_B$，因此有 $Sup\_B<Sup\_A<Min\_sup$，故 B 也为非频繁集合。

3.2 AprioriTMC 算法的执行流程

根据上述定理和推论，AprioriTMC 算法的执行流程图如图 2 所示。

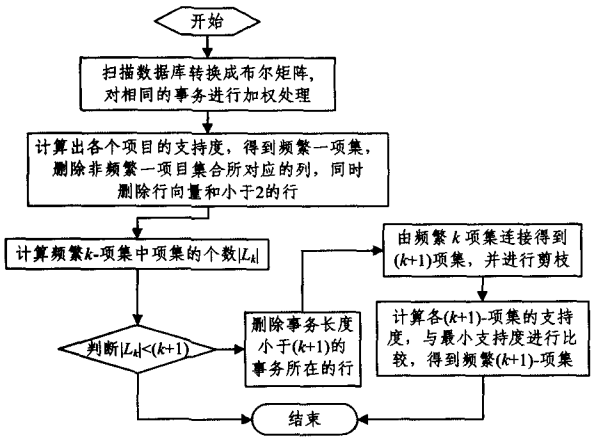

图 2 AprioriTMC 算法流程图

(1)遍历源数据库，根据数据库中的对应关系，按定义 1、定义 2 的原则构造矩阵 D，假如事务项目有 n 个，事务行数为 m，权重占据一行，那么构造出来的矩阵为 $D_{m\times(n+2)}$，多出的一行代表事务长度，起到压缩矩阵的作用。

(2)根据下式生成 1 项频繁集合：

$$support(I_j)=\sum_{k=1}^{n}(d_{kj}\times w_k)$$

得到支持度后，将大于最小支持度阈值的归纳到频繁 1 项集合并保存，对小于最小支持度阈值的进行列向量删除；同时还需要删除事务长度小于 2 的行向量事务。

(3)遍历更新的矩阵，将 1 项集进行拼接以得到二阶候选集，下一步将计算支持度，其计算公式为：

$$support\{I_i,I_j\}=\sum_{k=1}^{n}d_{ki}\wedge d_{kj}\times w_k$$

得到支持度后，更新 2 项目集的支持度，同时删除行向量小于 3 的事务。

(4)生成 $k+1$ 阶频繁集合时需要遍历 k 阶频繁集。由定理 3 可知，需要对 k 项集合进行统计，假如其个数少于 $k+1$，则不会存在 $(k+1)$-项频繁集合；若不少于 $k+1$，则对 k 阶频繁集合进行拼接，计算 $k+1$ 阶频繁支持度的公式为：

$$\sum_{s=1}^{n}(d_{s1}\wedge d_{s2}\wedge\cdots\wedge d_{sk}\times w_s)$$

若计算出的支持度大于支持度阈值，则将候选集归并到 k 阶频繁，同时调整矩阵的维度。若事务的长度小于 $k+1$，则将其删除，以起到压缩矩阵的作用。

(5)对得到的 $k+1$ 阶频繁集合进行统计，假如其包含的频繁集合个数少于 $k+2$，根据定理 3，说明不存在 $k+2$ 阶频繁集合，则算法终止。

3.3 AprioriTMC 进行 Hadoop 并行化设计

AprioriTMC 的并行化步骤如下：

(1)执行输入数据分片过程。

根据 HDFS 调度机制，调用 jobClient 获取作业，数据文件按照 inputformat 格式被分成了 n 个规模相当的数据块并以 inputsplit 的方式逻辑存储。这些数据块分别被分配到 m 个节点上。

(2)生成频繁一项集 L_1。

HDFS 调度完成后，开始执行 map 函数，每个数据块产生 pair 〈item,1〉。由于每个数据块会产生很多相同的item，因此可以使用 combiner 函数进行局部累加操作，然后输出〈item,n〉进入到 shuffle 阶段，之后利用 hash 函数进行分桶后，再调用 reduce 函数规约。在 reduce 中，输出为〈itemsets sum(sup)〉，其中 sum(sup)表示局部支持度。

(3)根据 1 阶频繁集生成 2 阶候选集。

步骤(2)中将 L_1 保存在 HDFS 上，可以将其组合成 C_2 以后再广播到各个节点。

(4)节点构建矩阵，输出局部候选集合的支持度。

再次读取事务数据库，每个节点根据本地数据建立矩阵，同时进行加权和剪枝处理。对每个候选集在本地矩阵做“与”运算并乘以权值，即可得到候选频繁集合的频次。求 2 阶候选集的支持度公式为：

$$support\{I_i,I_j\}=\sum_{k=1}^{n}d_{ki}\wedge d_{kj}\times w_k$$

重复步骤(2)、步骤(3)即可生成 L_2。

(5)根据 L_2 生成 C_3，进入迭代，重复步骤(4)。

(6)当生成的 k 项频繁集的个数少于 $k+1$ 时，算法终止。

AprioriTMC 的并行化算法的形式化描述如下：

1)mapreduce job 1 阶段的作用主要是生成 1 阶频繁集合，同时将 L_1 写入到 HDFS 以便于下次迭代。其伪代码如下：

```
Map:
  Mapper(key,value=itemsets)
  Foreach item itemsets//遍历每个项目集合
        Combiner//进行局部支持度合并
      Output(item,sum(sup)) //输出局部支持度
Reduce:
  Setup:Pre_key="null"
  Mapper(key,value=sup)
      If(key==pre_key)
        Sum_sup+=value
      else
  Output(item,Sum_sup) //输出局部支持
```

2)mapreduce 2 阶段的作用是建立事务矩阵，根据 k 阶频繁集合生成 $k+1$ 阶频繁集，这里需要使用 L_1 的结果，过程如下：

```
Map:
  Mapper(key,value=itemsets)
  Itemsets.prune(L1)//根据 L1 进行剪枝，删除频繁集合和事务长度小于 k+1 的项目
  Output(taskid,itemsets_vector)) //输出局部支持度
Reduce:
  Reducer(taskid,value=itemsets_vector)
    Foreach item itemsets_vectpr// 遍历每个向量
      Build(Matrix G).fresh()// 进行矩阵构建
```

Foreach k_column_items in Matrix G

$$local\_sup\_C_k=\sum_{k=1}^{n} d_{ki}\wedge d_{kj}\times w_k$$

Output(C_k,local_sup_C_k)

3.4 AprioriTMC 并行算法的性能分析

在分布式算法与传统算法的比较中,常常是以加速比作为指标的。所谓加速比是指任务在串行与并行时的时间消耗比。改进算法是需要中心节点指派任务的,那么必然有通信时差,设中心节点给每个节点派发任务的时间为 T_f。假设有 n 个项目,则通信总时间为 $T_f=nt_f$,整个算法的加速比为:

$$S=\frac{m(2^n\times T_s)}{nt_f+2^nT_s}=\frac{m}{1+\frac{n}{2^n}\frac{T_s}{t_f}}$$

通常情况下 n 较大,因此加速比可以接近 m。对算法效率而言,$E_p=S/m$,其中 S 为加速比,m 是并行节点个数。

$$E_p=1+\frac{n}{2^n}\frac{T_s}{t_f}$$

由上式可知算法效率与 n 呈现负相关,这说明改进算法具有良好的伸缩性。

4 仿真实验结果与分析

使用 6 台计算机作为 Hadoop 分布式计算平台,其中 1 台作为 NameNode 和 TaskTracker,其余 5 台作为 DataNode 和 TaskTracker。

硬件环境:Intel(R) Core2 Duo CPU T6400 @2.00GHz;2GB 内存;250G 盘;100Mbps 网口。

软件环境:Ubuntu 10;JDK1.7.0_0;eclipse 3.7。

4.1 AprioriTMC 算法的示例

(1)表 1 中 TID 表示事务序号,items 表示每个事务包含的项目数。根据前面提到的定理,转换后得到表 2 所列的布尔关系矩阵,对于相同的事务,采取权重的方式进行处理。表 1 中事务 3 和事务 7 完全相同,那么就将事务 3 的权重 w 更新为 2,其中列向量 T 代表事务所对应的长度,而横向量 S 代表每个项目的支持度。

表 1　原始事务表

TID	items
1	I_1,I_3,I_5
2	I_2,I_5,I_7
3	I_1,I_2,I_5,I_7
4	I_1,I_3,I_6,I_8
5	I_1,I_2,I_3,I_4,I_5,I_7
6	I_1,I_2,I_3,I_5,I_6
7	I_1,I_2,I_5,I_7
8	I_8

表 2　加权布尔模型

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	T	W
1	1	0	1	0	1	0	0	0	3	1
2	0	1	0	0	1	0	1	0	3	1
3	1	1	0	0	1	0	1	0	4	2
4	1	0	1	0	0	1	0	1	4	1
5	1	1	1	1	1	0	1	0	6	1
6	1	1	1	0	1	1	0	0	5	1
8	0	0	0	0	0	0	0	1	1	1
S	6	5	4	1	6	2	4	2		

(2)根据算法设计思路,应该在第一步寻找一项频繁集,可以看出 I_4,I_6,I_8 的支持度分别为 1,2,2,均小于 3,因此应该删除相应的列。同时可以看出,表 1 中第 8 行记录只有 I_8 一条记录,长度小于 2,对后面找寻频繁集合没有影响,因此应该删除 $T\_id=8$ 的横向量。剪枝后的布尔加权模型如表 3 所列。

表 3　剪枝后的布尔加权模型

	I_1	I_2	I_3	I_5	I_7	T	W
1	1	0	1	1	0	3	1
2	0	1	0	1	1	3	1
3	1	1	0	1	1	4	2
4	1	0	1	0	0	2	1
5	1	1	1	1	1	5	1
6	1	1	1	1	0	4	1
S	6	5	4	6	4		

1 项频繁集 $L_1=\{\{I_1,6\},\{I_2,5\},\{I_3,4\},\{I_5,6\},\{I_7,4\}\}$

(3)根据 L_1,可以得到候选 2 项集 $C_2=\{\{I_1,I_2\},\{I_1,I_3\},\{I_1,I_5\},\{I_1,I_7\},\{I_2,I_3\},\{I_2,I_5\},\{I_2,I_7\},\{I_3,I_5\},\{I_3,I_7\},\{I_5,I_7\}\}$。

由定义 4 得 $sup\{I_1,I_2\}=\sum_{k=1}^{6}(d_{k1}\wedge d_{k2}\times w_k)=4$,同理可得其余候选合的支持度:$sup\{I_1,I_3\}=4$,$sup\{I_1,I_5\}=5$,$sup\{I_1,I_7\}=3$,$sup\{I_2,I_3\}=2$,$sup\{I_2,I_5\}=5$,$sup\{I_2,I_7\}=4$,$sup\{I_3,I_5\}=3$,$sup\{I_3,I_7\}=1$,$sup\{I_5,I_7\}=4$。那么可以得到频繁 2 项集 $L_2=\{\{I_1,I_2\},\{I_1,I_3\},\{I_1,I_5\},\{I_1,I_7\},\{I_2,I_5\},\{I_2,I_7\},\{I_3,I_5\},\{I_5,I_7\}\}$。

(4)可以将得到的 L_2 拼接成 3 项候选集,枚举为 $C_3=\{\{I_1,I_2,I_3\},\{I_1,I_2,I_5\},\{I_1,I_2,I_7\},\{I_1,I_3,I_5\},\{I_1,I_3,I_7\},\{I_1,I_5,I_7\},\{I_2,I_3,I_5\},\{I_2,I_5,I_7\},\{I_3,I_5,I_7\}\}$。同理可以得到 C_3 的支持度:$sup\{I_1,I_2,I_5\}=4$,$sup\{I_1,I_3,I_5\}=3$,$sup\{I_1,I_5,I_7\}=3$,$sup\{I_2,I_5,I_7\}=4$,其余的支持度均小于 3。$L_3=\{\{I_1,I_2,I_5\},\{I_1,I_3,I_5\},\{I_1,I_5,I_7\},\{I_2,I_5,I_7\}\}$。

(5)由 L_3 可以得到 $C_4=\{\{I_1,I_2,I_3,I_5\},\{I_1,I_2,I_5,I_7\},\{I_1,I_3,I_5,I_7\}\}$,根据推论 1,$\{I_1,I_2,I_3,I_5\}$中的$\{I_2,I_3,I_5\}$并非为频繁 3 项目集合,因此$\{I_1,I_2,I_3,I_5\}$也不是频繁项目集合。

同理,$\{I_1,I_2,I_5,I_7\}$,$\{I_1,I_3,I_5,I_7\}$也不是频繁项目集,因此 L_4 为空。算法结束。

4.2 AprioriTMC 算法的可行性分析

本实验来自 frequent itemset mining dataset repository,数据集来自经典的实验数据 Webdocs.data。硬件设施为普通计算机,并行实验环境为由多台计算机组成的 Hadoop 集群,基础设施廉价可操作性强。

4.3 在单机条件下 AprioriTMC 的并行

本实验来自 frequent itemset mining dataset repository,实验内容是在单机上将传统的 Apriori 算法执行时间与串行的 AprioriTMC 算法、并行 AprioriTMC 算法作对比,选择 Webdocs.data 数据,再随机抽取其中的 100M,200M,300M,400M,500M 分别为 data1,data2,data3,data4,data5。实验中设定最小支持度为 5%。单机对比实验结果如图 3 所示。

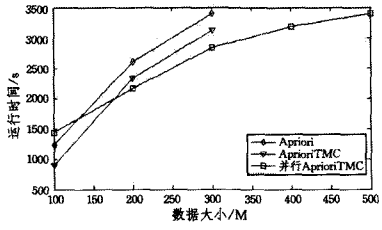

图 3 单机对比图

从图 3 中可以看出，串行 AprioriTMC 算法在性能上明显优于 Apriori 算法，其原因在于 AprioriTMC 算法采取了对事务数据库压缩的策略，而经典 Apriori 算法是没有压缩事务的，而且在 AprioriTMC 算法中会对候选集合进行动态的剪枝。在对 data4 和 data5 的数据块进行挖掘时，Apriori 与 AprioriTMC 均不能再对数据进行正常的挖掘，报错内存溢出，但是此时基于 Hadoop 的 AprioriTMC 算法却能够正常运行。由并行 AprioriTMC 运行时间走势可知，当规模为 data1 时，并行 AprioriTMC 算法在性能方面是不如单机 Apriori 和单机 AprioriTMC 的，其原因在于启动 Hadoop 以后，集群节点间需要相互通信从而会消耗较多时间，然后算法时长随着数据量的增加而延长。该实验表明，基于 Hadoop 的 AprioriTMC算法具有良好的伸缩性。

4.4 多节点集群测试

本节对改进算法的伸缩性与有效性进行验证。对于伸缩性，通常选择在不同大小数据集上的运行表现作为标准。而对于有效性，则在不同支持度下观察挖掘运行的表现形式是否一致。因此设计两个维度来进行实验，分别为数据规模大小不同和支持度大小不同。

(1)数据规模大小不同

并行计算中的应用程序分布在多个节点上。根据分布式的原理，计算过程中会发生通信，从而带来通信开销。因此不能采用过小的数据进行实验，抽取 Webdocs. data 的 200M，400M，600M，800M 分别设为 data1，data2，data3，data4。为了衡量伸缩性，采取通用的加速比的概念作为衡量标准，其计算公式如下：

$$Speed=\frac{\text{单节点算法执行时间}}{\text{多节点算法执行时间}}$$

将运行时间代入公式得到的结果如图 4 所示。在理论上加速比应该是线性相关的，即集群的节点数增加到 m 时，加速比也应该为 m。但集群之间会存在额外开销，包括通信开销、任务启动开销、任务调度和故障处理时间消耗等，因此真实加速比会低于预期的加速比，不过也在一定的节点范围内，随着节点数的增加，加速比得到提高。另外可以看出，数据规模越小，其加速比也越小，主要原因在于当计算规模小时，节点利用不充分。整体上，该算法是具备伸缩性的。

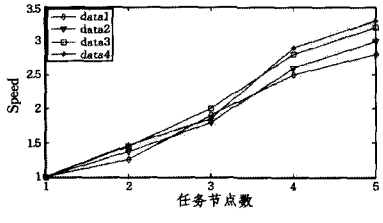

图 4 不同数据规模对比图

(2)支持度不同

选择 Webdocs. data 中的 500M 数据进行实验，支持度分别设为 5%，10%，15%。分别在节点数为 1～5 时，观察算法并行 AproiriTMC 的运行时间，其结果如图 5 所示。

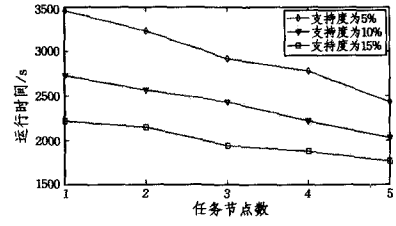

图 5 不同支持度对比图

从图 5 可以得到，在相同支持度下，并行 AprioriTMC 算法的计算时间是随着任务节点数的增长而递减的。同时可以看到，若支持度增大，算法的运行时间是减少的。在挖掘过程中，支持度增加会造成频繁集合的减少，这符合算法的规律，也验证了并行 AprioriTMC 算法的有效性，它是适用于大数据挖掘场景的。

结束语 对 Apriori 算法在事务压缩、简化候选集生成、减少扫描次数方面进行了改进，提出了以元素“0”和“1”表示事务的布尔矩阵模型，引入权值维度，压缩了相同事务的矩阵规模。动态地进行剪枝，以矩阵的“与”运算作为候选集合生成的计算方法。将改进算法结合 Hadoop 框架进行并行化实现，实验表明，该算法能够提高加速比，具备了良好的伸缩性和有效性，适用于大数据挖掘。

参 考 文 献

[1] 董志安，纪希禹，韩秋明，等. 数据挖掘技术与应用实例[M]. 北京：机械工业出版社，2009：12-48.

[2] XIE Y G，LIU R，QIAO R，et al. Resarch Literature on Big Data and Public Opinion[J]. New media and society，2014(4)：133-157. (in Chinese)
谢耘耕，刘锐，乔睿，等. 大数据与社会舆情研究综述[J]. 新媒体与社会，2014(4)：133-154.

[3] AGRAWAL R，SRIKANT R. Fast algorithm for mining association rules[C]// Proceedings of 20th Int. Conf. Very Large Data Bases (VLDB). Morgan Kaufman Press，1994：487-499.

[4] REN W J，YU B W. Improved Apriori Algorithm Based on Matrix Reducation[J]. Computer and Modern，2015，10：2-3. (in Chinese)
任伟建，于博文. 基于矩阵简约的 Apriori 算法改进[J]. 计算机与现代化，2015，10：2-3.

[5] GUNARATHNE T，WU T L，QIU J，et al. MapReduce in the Clouds for Science[C]// 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom). IEEE，2010：565-572.

[6] 陆嘉恒. Hadoop 实战[M]. 北京：机械工业出版社，2011：17-128.

[7] DEAN J，GHEMAWAT S. MapReduce：simplified data processing on large clusters [J]. Communications of the ACM，2008，51(1)：107-113.

(下转第 269 页)

$$\begin{aligned}&\| w^{k+1}(\alpha_k^*)-w^* \|^2\\&\leqslant \| w^k-w^* \|^2-(2\gamma\alpha^*\varphi(w^k,\tilde{w}^k,\xi^k)-\\&\quad \gamma^2(\alpha^*)^2 \| d(w^k,\tilde{w}^k,\xi^k) \|^2)\\&\leqslant \| w^k-w^* \|^2-\gamma(2-\gamma)\alpha^*\varphi(w^k,\tilde{w}^k,\xi^k)\\&\leqslant \| w^k-w^* \|^2-\gamma(2-\gamma)\alpha^*\delta_1 \| w^k-\tilde{w}^k \|^2\end{aligned}$$

从推论 1 可以看出，$\{w^k\}$是 Feje′r 单调的。与文献[11]中的证明方法类似，可以很容易地得到定理 1。

定理 1　由算法 1 产生的序列$\{w^k\}$收敛于变分不等式问题(1)一问题(3)的解。

5　非精确渐进点算法的数值实验检验

本节通过数值实验来验证算法 1 的有效性，将所提算法与文献[11]中的 IPSALM 算法和文献[12]中的 IADM 算法进行比较。取相关参数为：

$\varepsilon=10^{-5}$，$\beta=3^{-1}$，$H=I$，$\gamma=1.2$

考虑问题：

$$\min \frac{1}{2} \| X-C \|^2+\frac{1}{2} \| Y-C \|^2$$

$$\text{s. t. } X-Y=0$$

$$X\in\{H\in R^{n\times n} \mid H^{\mathrm{T}}=H, H\geqslant 0\}$$

$$Y\in\{H\in R^{n\times n} \mid H^{\mathrm{T}}=H, H_L\leqslant H\leqslant H_U\}$$

实验结果如表 1 所列。

表 1　3 种算法迭代次数和时间的对比

n	IPSALM		IADM		算法 1	
	迭代次数	迭代时间/s	迭代次数	迭代时间/s	迭代次数	迭代时间/s
100	81	0.84	74	0.63	59	0.59
200	105	5.61	109	4.81	73	3.54
300	124	17.28	121	13.80	79	10.79
400	133	43.10	136	34.65	95	23.51
500	148	89.55	151	72.24	97	48.83
800	192	412.25	173	311.52	111	212.55

结束语　本文在渐近点算法的基础上得到了一种非精确的渐近点算法，使得子变分不等式问题具有显式解，因而容易求解，数值实验也表明了算法的有效性。参数的选取对算法也有一定的影响，如何选取参数能加快算法的收敛速度还需进一步的讨论。

参 考 文 献

[1] BOYD S, PARIKH N, CHU E, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers[J]. Foun Trends Mach Learn, 2011, 3(1): 1-122.

[2] NAGURNEY A, ZHANG D. Projected Dynamical System and Variatioal Inequalities with Applications[M]. Kluwer Academic, Boston, 1996.

[3] FUKUSHIMA M. Application of the alternating directions method of multipliers to separable convex programming problems [J]. Computational Optimization and Applications, 1992, 1(1): 93-111.

[4] GLOWINSKI R, MARROCCO A. Sur l'approximation par éléments finisd' ordre un et la résolution par pénalisation-dualité d'une classe de problémes de Dirichlet non linéaires[J]. Journal of Equine Veterinary Science, 1975, 31(5): 41-76.

[5] ECKSTEIN J, BERTSEKAS D P. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators[J]. Mathematical Programming, 1992, 55: 293-318.

[6] HE B S, LIAO L Z, HAN D R, et al. A new inexact alternating directions method for monotone variational inequalities[J]. Mathematical Programming, 2002, 92(1): 103-118.

[7] YE C H, YUAN X M. A descent method for structured monotone variational inequalities [J]. Optimization Methods Software, 2007, 22(2): 329-338.

[8] HE B S, TAO M, YUAN X M. Alternating direction method with Gaussian back substitution for Separable convex programming[J]. SIAM J. Optimization, 2012, 22(2): 313-340.

[9] HE B S. Parallel splitting augmented Lagrangian methods for monotone structured variational inequalities Computational[J]. Optimization and Applications, 2009, 42(2): 195-212.

[10] CAI X J, GU G Y, HE B S, et al. A proximal point algorithm revisit on the alternating direction method of multipliers[J]. Science China Mathematics, 2013, 56(10): 2179-2186.

[11] TAO M, YUAN X M. An inexact parallel splitting augmented Lagrangian method for monotone variational inequalities with separable structures[J]. Computational Optimization and Applications, 2012, 52(2): 439-461.

[12] HE B S, LIAO L Z, YUAN X M. Alternating projection based prediction-correction methods for structured variational inequalities[J]. Computational Mathematics, 2006, 24(6): 693-710.

[13] CHEN Z M, WAN L, YANG Q Z. An Inexact Direction Method for Structured Variational Inequalities[J]. Journal of Optimization Theory & Applications, 2014, 163(2): 439-459.

[14] GU G Y, HE B S, YANG J F. Inexact Alternating Direction-Based Contraction Methods for Separable Linearly Constrained Convex Optimization[J]. Journal of Optimization Theory and Applications, 2013, 163(1): 105-129.

(上接第 266 页)

[8] LI W J, ZHAO H, ZHANG Y, et al. Research on massive data mining technology based on MapReduce[J]. Computer Engineering and Applications, 2013(20): 113-114. (in Chinese)
李伟卫，赵航，张阳，等. 基于 MapReduce 的海量数据挖掘技术研究[J]. 计算机工程与应用，2013(20)：113-114.

[9] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data [J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4-10.

[10] LIU S J. Research of Frequent Itemsets Mining Alogorithm Based on MapReduce Calculation Model[D]. Harbin: Harbin University of Science and Technology, 2015: 39-50. (in Chinese)
刘士佳. 基于 MapReduce 框架的频繁项集挖掘算法研究[D]. 哈尔滨：哈尔滨理工大学，2015：39-50.

[11] KUANG S H, LI B. Analysis Cloud Computing Architecture and its Application[J]. Computer&Digital Engineering, 2010, 3(6): 61-63. (in Chinese)
框胜徽，李勃. 云计算体系结构及应用实例分析[J]. 计算机与数字工程，2010，3(6)：61-63.

[12] BONCHI F, LUCCHESE C. Pushing tougher constraints in frequent patternvmining[M]// Advances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg, 2005: 114-124.

[13] BCHEUNG D W, HAN J, NG V T, et al. Maintenance of discovered association rules in largedatabases: An incremental updating technique[C]// Proceedings of the Twelfth International Conference on Data Engineering, 1996. IEEE, 1996, 106-114.