基于信息熵与遗传算法的并行关联规则增量挖掘算法

毛伊敏[1]，邓千虎[1]，陈志刚[2]

（1. 江西理工大学信息工程学院，江西 赣州 341000；2. 中南大学计算机学院，湖南 长沙 410083）

摘 要：针对大数据环境下基于 Can 树的增量关联规则算法存在树结构空间占用过大、支持度阈值无法动态设置以及 Map 与 Reduce 阶段数据传输耗时等问题，提出了一种基于信息熵和遗传算法的并行关联规则增量挖掘算法 MR-PARIMIEG。首先，该算法设计基于信息熵的相似项合并策略（SIM-IE）来合并相似数据项，并根据合并后的数据集进行 Can 树构造，从而减少树结构的空间占用；其次，提出基于遗传算法的 DST-GA 策略获取大数据环境下相对最优的动态支持度阈值，根据此阈值进行频繁项集挖掘，避免了冗余的频繁模式挖掘导致的时间消耗；最后，在 MapReduce 并行化运算过程中使用并行 LZO 数据压缩算法对 Map 端输出数据进行压缩，从而减少传输的数据规模，最终提升算法的运行速度。实验仿真结果表明，MR-PARIMIEG 在大数据环境下进行频繁项集挖掘时具有较好的性能表现，适用于对较大规模的数据集进行并行化处理。

关键词：Can 树；信息熵；大数据；增量挖掘；数据压缩

中图分类号：TP311

文献标识码：A

DOI: 10.11959/j.issn.1000−436x.2021052

Parallel association rules incremental mining algorithm based on information entropy and genetic algorithm

MAO Yimin[1], DENG Qianhu[1], CHEN Zhigang[2]

1. School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China

2. College of Computer Science and Engineering, Central South University, Changsha 410083, China

Abstract: Aiming at the problems that in the big data environment, the Can-tree based incremental association rule algorithm had problems such as too much space occupation of the tree structure, inability to dynamically set the support threshold, and too much time consumption during the data transfer process between the Map and Reduce stages, the Map Reduce-based parallel association rules incremental mining algorithm using information entropy and genetic algorithm (MR-PARIMIEG)was proposed. Firstly, a similar items merging based on information entropy (SIM-IE) was designed to merge similar data items, and a Can tree based on the merged data set was constructed, thereby reducing the space occupation of the tree structure. Secondly, the dynamic support threshold obtaining using genetic algorithm (DST-GA) was proposed to obtain the relatively optimal dynamic support threshold in the big data environment, and frequent itemset mining was performed according to this threshold to avoid the unnecessary time consumption caused by mining redundant frequent patterns. Finally, in the process of MapReduce parallel operation, the parallel LZO data compression algorithm was used to compress the output data of the Map stage, thereby reducing the size of the transmitted data, and finally improving the running speed of the algorithm. Experimental simulation results show that MR-PARIMIEG has better performance when mining frequent item sets in the big data environment, and it is suitable for parallel processing of larger data sets.

Keywords: Can-tree, information entropy, big data, incremental mining, data compression

收稿日期：2020−11−13；**修回日期**：2021−02−04

基金项目：国家自然科学基金资助项目（No.41562019, No.61762046）；国家重点研发计划基金资助项目（No.2018YFC1504705）

Foundation Items: The National Natural Science Foundation of China (No.41562019, No.61762046), The National Key Research and Development Program of China (No.2018YFC1504705)

1　引言

关联规则是数据挖掘的一个主要研究领域，其目的在于发现数据集中有价值的潜在频繁模式[1]。目前，关联规则挖掘已被广泛应用于购物推荐、网站点击分析、电子商务、金融和医疗诊断等领域，产生了极大的社会与经济效益[2-3]。静态关联规则挖掘是在固定的数据集以及支持度阈值下，对数据集中的频繁模式进行挖掘，如 Apriori[4]、FP-Growth[5]、Eclat[6]等。但这些静态关联规则算法并未考虑到数据库中的事务发生变化或是关键的支持度阈值发生变化的增量挖掘工作，这将导致算法需对整个数据集进行重复处理，造成大量的时空间消耗。针对此问题，Leung 等[7]提出了一种基于 CATS（compressed and arranged transaction sequences）树改进的 Can 树（Canonical order tree）来简化增量挖掘工作，极大地提高了算法的运行效率。但在大数据环境下，随着数据量的指数级增长，运算时间过长和内存占用过高已成为传统关联规则算法处理海量数据的重要瓶颈[8]。因此，提高算法的并行化效率使之能高效地对海量数据进行处理是目前迫切需要解决的问题。

MapReduce 是 Google 公司为解决海量数据处理提出的一种分布式并行运算框架[9]，具有使用简单、成本低廉、系统扩展性好以及负载均衡等优点，目前已被广泛应用于大数据分析与处理等领域。基于此，Song 等[10]提出了一种大数据环境下的并行频繁项集挖掘算法 IncMiningPFP，结合 MapReduce 计算框架实现了 Can 树向并行化方向的迁移，极大地提高了基于 Can 树的增量挖掘算法的并行化运算能力。但该算法在使用 Can 树结构存储所有数据信息以加快后续增量挖掘的同时，也会导致最终产生的树结构极其庞大。针对此问题，胡军等[11]使用一种基于数据量排序的 Can 树构造方法，将各事务中的数据项按出现频次进行排序，然后根据排序后的事务进行树结构构建，使各个事务中相同的数据项尽量共用一个树节点，从而降低最终生成的树结构的空间占用。但此方法在适用性与可行性方面具有较大的局限性，因此对 Can 树结构空间占用过大的问题仍然有待进一步改进。

此外，支持度阈值的设定对关联规则算法而言也是极其关键的问题。大数据的数据规模更大，但数据的价值密度却随之下降，当设定的支持度阈值较小时，将会产生大量冗余的无效频繁项集；当使用相对较大的支持度阈值时，则可能在减少冗余项集的同时丢失一些高质量的频繁项信息，使最终生成的频繁项的整体质量偏低。针对该问题，Ragaventhiran 等[12]在应用 MapReduce 框架并行化挖掘 Can 树频繁模式的同时，设计了基于多支持度的频繁模式挖掘方法，在不同的支持度阈值下进行频繁模式挖掘，根据挖掘结果来获取最优的支持度阈值。但该算法仅支持特定的阈值取值范围，并未对大数据环境下的阈值取值进行优化，难以根据相对最优的动态支持度阈值获取最终结果。

对于基于 Can 树的并行挖掘算法，除了 Can 树结构空间占用过大以及支持度阈值难以动态设定的问题外，在 MapReduce 的并行化运算中，Map 与 Reduce 进程之间的数据传输过程也会对系统的整体性能产生极为重要的影响。Map 任务在内存中产生的数据需要先存储到磁盘，再由 Reduce 进程从中读取数据，但磁盘与内存之间 I/O 速度的极大差异将会导致这一数据传输过程消耗大量时间等待内存区数据写入磁盘，最终影响算法的运行速度。因此，申玲艳[13]设计了针对 Map 端输出数据的优化策略，将多个 Map 任务同时产生的输出数据进行合并，在内存区数据量达到阈值后将数据写入磁盘，使 Reduce 任务能够获取更加紧凑的 Map 端输出数据，从而提高数据传输效率，加快数据传输过程；Cao 等[14]提出了一种优化 Map 中间输出数据的通信方法，对 MapReduce 任务中的数据通信活跃度进行量化，使用 Bayes 算法进行分类预测，根据分类结果将 Map 端输出的通信活跃的数据映射到同一分区中，通过优化传输流程来加快 Map 与 Reduce 之间的数据传输。但这些算法都存在一定的局限性，当 Map 端单个节点输出的数据规模已经达到内存区阈值或是输出数据的活跃度相差不明显时，这些改进算法对整体的性能提升效果并不明显。

综上，尽管上述算法取得了一定成效，但如何有效降低 Can 树结构的空间占用、获取大数据环境下相对合适的动态支持度阈值、加快 Map 与 Reduce 阶段的数据传输等问题仍然是目前亟待解决的。针对这些问题，本文提出一种基于信息熵和遗传算法的并行关联规则增量挖掘算法 MR-PARIMIEG。本文主要贡献如下。

1) 设计基于信息熵的 SIM-IE 策略来合并数据

集中的相似数据项，根据合并后的数据集进行 Can 树构造，降低了最终生成的 Can 树结构的空间占用。

2) 提出基于遗传算法的 DST-GA 策略，以获取大数据集中的相对最优的动态支持度阈值，根据此动态阈值进行频繁模式挖掘，避免了冗余的频繁模式挖掘导致的时间消耗。

3) 使用并行 LZO 数据压缩算法对 Map 端输出的数据进行压缩来减少传输的数据规模，从而减少了内存与磁盘之间的 I/O 操作频次，最终加速 Map 与 Reduce 阶段的数据传输过程。

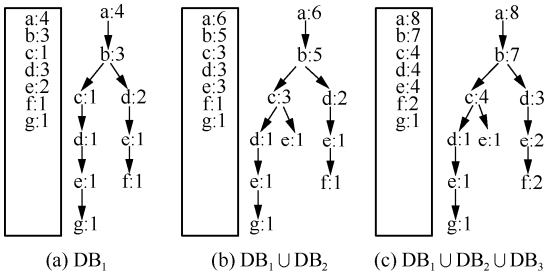
2 相关概念介绍

2.1 Can 树

Can 树是一种适用于增量挖掘的树形数据结构，在进行增量挖掘时，其挖掘过程不受最小支持度或数据信息变化的影响，且具有较高的挖掘效率。构造 Can 树时首先将事务中所有的数据项按照某种特定顺序（字典序、字母序等）进行排序，然后进行树结构构造，在后续对增量关联规则进行挖掘时只需要将增量数据直接更新至原有的 Can 树中[15]。给定如表 1 所示的数据集，Can 树的构建过程如图 1 所示。

表 1 数据集

数据库	事务	数据项
DB_1	t_1	{a, b, d, g, e, c}
	t_2	{d, f, b, e, a}
	t_3	{a}
	t_4	{d, b, a}
DB_2	t_5	{a, c, b}
	t_6	{a, c, b, e}
DB_3	t_7	{a, c, b}
	t_8	{a, b, d, e, f}



(a) DB_1 (b) $DB_1 \cup DB_2$ (c) $DB_1 \cup DB_2 \cup DB_3$

图 1 Can 树构造过程

2.2 信息熵

信息熵是一种反映不确定性的度量方式，通过信息量的规模来体现不确定程度。在不同数据项之间的相似度判断中，信息熵的值能相对合理地反映数据项中各属性之间的相似性。即信息熵的数值越大，信息量的规模也相对越大，从而数据项之间的相似性越高[16]。

定义 1 信息熵。令 $x=\{x_1,x_2,\cdots,x_n\}$ 为事务集，$P(x_i)$ 为对应事务的概率值，那么事务集 x 的信息熵 $H(x)$ 可表示为[16]

$$H(x)=-\sum_{i=1}^{n}P(x_i)\log(P(x_i)) \tag{1}$$

2.3 Dempster 组合规则

Dempster 组合规则是一种基于多维信息的组合规则，对具有少量冲突信息的多维信息有较高的处理效率。在对相似度较高的数据集的概率进行合并的问题中，各数据项的属性在形式上具有较大的相似性，冲突项较少。因此该问题在形式上符合需求，可以使用 Dempster 组合规则进行求解[17]。

定义 2 Dempster 组合规则。设 A、B 为 2 个数据集，C 为集合 A、B 进行数据合并后的集合，f_1、f_2 分别是集合 A、B 的概率分配函数，Ω 是集合 A、B 不相交的概率总和，φ 为 A、B 的交集，那么 A、B 合并后获得的集合 C 的概率 $P(C)$ 为

$$P(C)=\frac{\sum_{A\cap B=C}f_1(A)f_2(B)}{1-\Omega} \tag{2}$$

其中，$\Omega=\sum_{A\cap B=\varphi}f_1(A)f_2(B)$。

2.4 LZO 数据压缩算法

LZO 数据压缩算法是一种基于字典模型的无损压缩算法，可实现对数据的高效压缩。并行 LZO 数据压缩算法针对现代计算机的多核多线程 CPU 结构，使用多线程技术来高效利用 CPU 的运算能力，通过控制线程（control）、压缩线程（compression）以及重构线程（reconstruction）3 种线程，实现对数据块的并行化高效压缩[18]。在 MapReduce 分布式计算框架中，利用集群的并行运算能力来实现并行化的 LZO 数据压缩算法，从而通过压缩数据减少传输的数据规模，加快 Map 与 Reduce 阶段的数据传输过程。并行 LZO 数据压缩算法的具体实现流程如图 2 所示。

3 MR-PARIMIEG 算法

3.1 算法思想

MR-PARIMIEG 主要分为 3 个阶段，分别为相

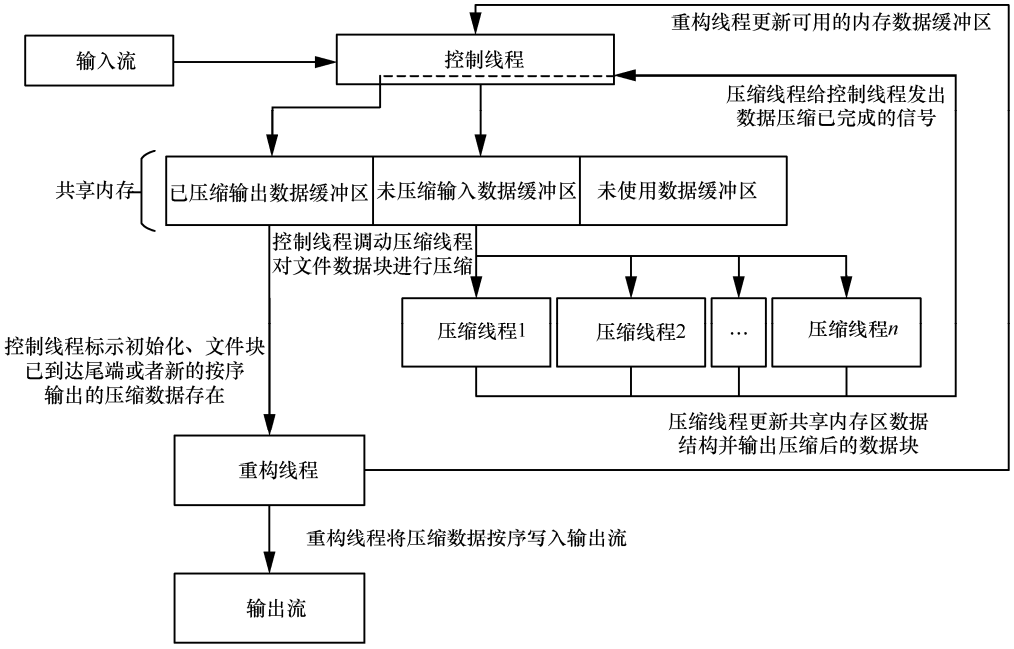


图 2　并行 LZO 数据压缩算法的具体实现流程

似项合并、动态支持度阈值获取以及频繁项集并行挖掘，各阶段的主要任务如下。1) 在相似项合并阶段，首先对数据集进行划分，然后使用相似性评估公式对划分数据集的相似性进行评估，最后根据相似度阈值合并相似项并更新全局概率值。2) 在动态支持度阈值获取阶段，根据数据集构造支持度函数，然后基于遗传算法的求解过程构造动态阈值公式来获取支持度函数的最优解，即动态支持度阈值。3) 在频繁项集并行挖掘阶段，将数据挖掘任务向 MapReduce 平台进行迁移，并使用并行 LZO 数据压缩算法对 Map 端输出数据进行压缩，来加快 Map 与 Reduce 阶段的数据传输过程，最后在 Reduce 阶段根据动态支持度阈值从树结构中并行挖掘频繁项集。

3.2　SIM-IE 策略合并相似项

基于 Can 树结构的增量关联规则挖掘算法在构建树结构时会存储所有的数据项，因此在数据量较大时最终生成的树结构将极其庞杂。基于此，本节提出 SIM-IE 策略以合并数据集中的相似项，从而简化最终生成的树结构，降低树结构的空间占用。SIM-IE 主要包含以下 3 个步骤。1) 数据集划分：计算数据项的平均差，并将平均差与数据项之间的差值进行比较，从而对数据集进行划分。2) 相似度计算：根据划分后的数据集计算信息熵和条件熵，以获取各数据集之间的相似度。3) 相似项合并：将各数据集之间的相似度与相似度阈值进行比较，由此进行相似项合并，同时计算并更新合并后的数据集的全局概率。

3.2.1　数据集划分

对于构成 Can 树的数据项而言，对其进行排序后数据项将按值大小进行分布，那么相邻的数据项之间的差值可以直观地反映数据的邻近性，即相邻项的差值越小，那么这 2 个数据项就越邻近，因此可根据差值大小对数据项进行分类。平均差反映了整个数据集的平均邻近距离，根据平均差将具有平均邻近性的数据项划分为一类，而相邻项之间的差值大于平均邻近距离的项则划分到不同的数据集合中，可以将整个集合中的数据相对合理地划分到不同的数据集中。因此，将数据项的平均差作为评价标准以进行数据划分是较可行的一种划分方法，这种平均的划分方法首先将数据集进行合并与排序，然后计算各数据项之间的差值，从而求得平均差，最后根据平均差进行数据划分。具体的划分过程如下。

①对需要进行相似性评估的数据集中的数据项进行合并，然后按序排列，求出各相邻数据的差值总和 sum。令数据项数为 n，那么求得的平均差为 avg=sum/(n−1)。

②求得平均差 avg 后，即可将排序后的数据集根据 avg 进行划分。设划分数为 d，如果相邻的数

据值之差小于平均差，那么就将前一个划分获得的数据与当前的数据之间的所有数据项归为一个分区，重复执行比较与分区操作，直至所有数据都被划分到对应的数据分区。

3.2.2 相似度计算

获得划分好的数据之后，即可根据相似性评估公式计算对应的数据集之间的相似度，具体的计算过程如下：首先获取划分数、被划分的不同数据集的数量以及数据集规模，然后根据相似性评估公式 SAF 求得对应的条件熵以及信息熵，最终获得数据集之间的相似度。

定理 1 相似性评估公式 SAF。令 A、B 为 2 个相似性待判断的数据集，S 为决策模式属性集，C 为不确定匹配模式关系集，C 与 S 交集为空，那么可根据对应的条件熵以及信息熵求得 A、B 之间的相似度。

$$H(S\mid C)=-\sum_{c\in C}P(c)\sum_{s\in S}P(s\mid c)\log(P(s\mid c)) \tag{3}$$

$$H(S)=-\sum_{i=1}^{n}P(x_i)\log(P(x_i)) \tag{4}$$

$$\mathrm{sim}(A,B)=\frac{H(S\mid C)}{H(S)} \tag{5}$$

证明 令 O 为决策模式属性有限集，那么可求得信息熵的不确定率为 $\mu=1-(H(C)+H(S|C)-H(S))/(\log(O)-H(S))$。设 R_1 和 R_2 为模式集 O 中的等价关系，$H(S_1)$ 和 $H(S_2)$ 为其对应的信息熵，则有如下属性。

① 若 R_1=R_2，那么 $H(S_1)=H(S_2)$，故 μ 满足不变性。

② 对于 $\forall C,S\in O$，均有 $(H(C)+H(S\mid C)-H(S))/(\log(O)-H(S))\in[0,1]$，即 $\mu\in[0,1]$，故 μ 满足非负性。

③ 若 $R_1>R_2$，那么 $H(S_1)>H(S_2)$，故 μ 满足单调性。

综上所述，SAF 中信息熵的不确定因子 μ 同时满足不变性、单调性以及非负性，因此 SAF 可以作为不确定数据集相似性评估的基本准则公式。

证毕。

3.2.3 相似项合并

根据 SAF 获取所有划分数据集之间的相似性结果后，再根据提前设定的相似度阈值 δ 判断是否需要进行相似项合并，即将经过计算求得的相似度数值 $\mathrm{sim}(A,B)$ 与 δ 进行比较，若 $\mathrm{sim}(A,B)\geqslant\delta$，则进行相似项合并，同时对相似项合并后的全局概率值进行计算并更新。具体运行过程如下。

①获取所有的相似项并循环合并这些项集，将所有的数据元组放入同一个数据表。

②从合并后的项集中获取具有相同数据的元组，保留一个元组并删除其他重复项。

③通过概率合并公式 PM-DCR 对相似项的概率进行合并计算，得到删除重复项之后的项集的全局概率值。

定理 2 概率合并公式 PM-DCR。令 p_1、p_2 分别为项集 S_1、S_2 的全局概率，p_1p_2 为 S_1、S_2 相交的概率，$p_1(1-p_2)$、$(1-p_1)p_2$ 均为 S_1、S_2 不相交的概率，S_1、S_2 合并后的项集的概率为

$$P(S_1,S_2)=\frac{p_1p_2}{1-p_1(1-p_2)-(1-p_1)p_2} \tag{6}$$

证明 令 H_1、H_2、H_3 为识别框架 Θ 的独立证据体，m_1、m_2、m_3 分别为 H_1、H_2、H_3 的信任指派函数，那么 PM-DCR 有如下属性。

① 对于 H_1、H_2，$P(H_1,H_2)$=$P(H_2,H_1)$，即 $m_1\oplus m_2=m_2\oplus m_1$，故 PM-DCR 满足交换律。

② 对于 H_1、H_2、H_3，$P(H_1,H_2,H_3)$= $P(P(H_1,H_2,H_3)$=$P(H_1,P(H_2,H_3))$，即 $m_1\oplus m_2\oplus m_3=(m_1\oplus m_2)\oplus m_3=m_1\oplus(m_2\oplus m_3)$，故 PM-DCR 满足结合律。

③ 如果 m_1、m_2 均为单调函数，使用 PM-DCR 对其进行融合时，有 m_3=$m_1\oplus m_2$，那么 m_3 也是单调的，故 PM-DCR 满足单调性。

④ 使用 PM-DCR 对 Θ 中的证据元素进行融合后，随着单个子集合的信任指派值增加，对应元素的信任指派值会随之降低，故 PM-DCR 满足极化性。

综上，PM-DCR 同时满足交换律、结合律、单调性以及极化性，是一种可行的 Dempster 组合规则改进方法，因此 PM-DCR 可对相似项的概率值进行合并计算，证毕。

相似项合并过程的伪代码如算法 1 所示。

算法 1 相似项合并

输入 集合 A、B，数据规模 M、N，划分数 K，相似度阈值 δ

输出 合并相似项后的数据集 D

1) set division = dataSetDivide(A、B、K) //获取划分

2) for (int i =0; division[i] !=null; i ++)

3)　set number =第 i 个划分的数据项数量，countA =0 // countA 为划分中包含的集合 A 的数据项数

4)　for (int j=0; j<number ; j++)

5)　division[i][j] ∈ A? countA++:countA

6)　end for

7)　set condition =getCondition (countA, K, number)//获取条件熵

8)　ratio =getRatio(sum, K)//获取信息熵

9)　end for

10) set sim=condition / ratio //获取相似度

11) if(sim≥δ)//判断相似性是否大于阈值

12)　set D =getResByFunc//获取合并相似项后的数据集

13) end if

14) probabilityStandardizeByDCR(D)//调整合并后的概率

15) return D //输出最终结果

3.3　动态支持度阈值获取

一般而言，挖掘关联规则时的支持度阈值需预先设定好且始终保持不变，但对于大数据集来说，由于其数据价值密度降低并且数据始终处于动态变化之中的新特性，固定的支持度阈值已不足以反映其关联规则的变化情况。为了获取相对最优的动态支持度阈值，MR-PARIMIEG 设计了一种 DST-GA 策略，实现过程如下：支持度函数构造，根据具体的数据集信息构造支持度函数；动态阈值获取，基于遗传算法的收敛性求取函数的最优解，即动态支持度阈值。

1) 支持度函数构造

大数据环境下，为适应数据的新特性，需要对支持度阈值做出调整以使其逐渐趋于用户感兴趣的或是有价值的关联规则，这一动态交互过程的实现即对动态支持度阈值的求解过程。基于此，DST-GA 提出了支持度函数的定义，并将这一过程公式化为基于极值定理的最优值问题求解。

定理 3　支持度函数 SF。令 m 表示数据集 D 的属性数，$P(x_i)$ 表示项目 x_i 在 D 中出现的概率，$\mathrm{Weight}(x_i)$ 表示 x_i 的权重，$r(x_1,x_2,\cdots,x_m)$ 表示修正函数，那么支持度函数 SF 可表示为

$$\mathrm{SF}=\frac{\sum_{i=1}^{m}P(x_i)\mathrm{Weight}(x_i)}{|D|}r(x_1,x_2,\cdots,x_m) \tag{7}$$

证明　支持度函数为连续有界函数，最终的支持度阈值主要取决于各个数据项的概率值 $P(x_i)$ 以及对应的权重值 $\mathrm{Weight}(x_i)$。对于 $\sum_{i=1}^{m}P(x_i)\mathrm{Weight}(x_i)$，数据集的项目概率值的变化将直接反映到最终的公式化计算中。因此，当数据集信息发生变化时，SF 可以根据改变后的信息对 $\sum_{i=1}^{m}P(x_i)\mathrm{Weight}(x_i)$ 进行计算，同时更新 $|D|$，最终获得适用于更新后数据集的支持度阈值。证毕。

2) 动态阈值获取

构造支持度函数 SF 后，利用遗传算法的收敛性与不失一般性，对其进行迭代优化运算以求得最优解[19]，即相对最优动态支持度阈值。具体的求解过程如下。

① 根据极值定理可求得连续的支持度函数在其定义域内的极小值 ξ_1 与极大值 ξ_2，所要求的动态支持度阈值必然介于 ξ_1、ξ_2 之间。

② 基于遗传算法的收敛性，在数学问题的优化求解过程中，当循环迭代的运算次数足够多时最终结果将会收敛于特定值。

③ 将遗传算法应用于支持度函数的极值问题求解过程中，提出求解动态阈值的 min SF 公式将这一迭代优化过程具象化，经过多次迭代后最终 ξ_1、ξ_2 的差值趋近于 0，最终结果收敛于特定值，即相对最优动态支持度阈值。

定理 4　动态阈值公式 min SF。令数据变量集 $x=(x_1,x_2,\cdots,x_m)^{\mathrm{T}}$，$x$ 中任意数据项 x_i 的取值范围为 $[a_i,b_i]$，那么根据遗传算法求取支持度函数中动态最优值的计算式为

$$\min \mathrm{SF}(x)=\min \max_{1\leqslant i\leqslant m}\{f_i(x)\} \tag{8}$$

证明　令封闭 m 维立方体 $C=\{(x_1,x_2,\cdots,x_m)\,|\,x_i\in[a_i,b_i],i=1,2,\cdots,m\}$ 为式(8)中变量 x 的取值范围，由遗传算法相关原理可知，第 n 次加速循环的搜索取值为 $C_n=\{(x_1,x_2,\cdots,x_m)\,|\,x_i\in[a_i^n,b_i^n],i=1,2,\cdots,m\}$，且 $0\leqslant a_i^n-b_i^n\leqslant a_i^{n-1}-b_i^{n-1}$。令收缩区间比 $k_i^n=(a_i^n-b_i^n)/(a_i^{n-1}-b_i^{n-1})<1$，那么当循环次数趋于无穷时，$0\leqslant\prod_{n=1}^{\infty}k_i^n\leqslant\prod_{n=1}^{\infty}a\to 0$ (a<1,i=1,2,⋯,m)，故可得 $C_0\supset C_1\cdots\supset C_n$。此时，对任意 $x_n,y_n\in C_n$，它们的差值趋近于 0，即可知经过循环迭代最终获得的结果具有收敛性。证毕。

算法 2 给出了获取动态支持度阈值时，整体运行流程的伪代码。

算法 2 动态支持度阈值获取

输入 数据集 D，各数据项对应概率数组 P，GA 迭代次数 N

输出 动态支持度阈值 DST

1) set count=0, W = null, SF=null //初始化数据项数 count，权重数组 W

2) for(int i =0; $i < d$.length ; i++)

3) count++ //统计数据项数

4) set $W[i]$=用户指定权重值//给数据项指定权重

5) end for

6) SF =getSupportFunction(D,P,W)//根据已知条件获取支持度函数

7) set ratio =1//记录收缩区间比

8) for(int $j = 1$; $j < N + 1$; j++)//循环进行遗传算法迭代

9) set $C_j = \{x_1, x_2, \cdots, x_{\text{count}}\}$ //算法运行空间

10) for(int k = 0 ; $k < C_j$.length ; k++)

11) ratio = ratio(($C_j[k+1] - C_j[k]$) / ($C_{j-1}[k+1] - C_{j-1}[k]$))

12) end for

13) end for

14) DST =getValueByGA (ratio, N, D, P, SF, C_N) //调用基于 GA 的最优值求解方法获取最终结果

15) return DST

3.4 频繁项集并行挖掘

经过前述工作，已经获得了适用于增量挖掘的动态支持度阈值以及合并相似项后的数据集。为了并行化挖掘频繁项集，将挖掘过程向 MapReduce 计算平台迁移，同时使用并行 LZO 压缩算法对 Map 阶段输出的数据进行压缩，以加快 Map 与 Reduce 阶段的数据传输过程，最终加速算法运行。

MR-PARIMIEG 结合 MapReduce 并行挖掘频繁项集的过程分为初始挖掘与增量挖掘 2 个部分，2 个过程的主要区别在于增量挖掘时需要根据新增数据集的事务项对初始挖掘阶段构造的 Can 树进行更新，然后从更新的树结构中挖掘频繁项集。MR-PARIMIEG 算法并行挖掘频繁项集的具体过程如下所示，并在图 3 中给出了总体的运行过程。

1) Map 阶段。扫描合并相似项后的原始数据集或新增数据集的所有数据项，并根据具体的集群节点配置情况对数据进行分块，最后将划分好的数据块映射到每个计算节点进行 Map 运算。

2) 数据压缩阶段。将 Map 阶段的输出数据使用并行 LZO 数据压缩算法进行压缩，具体的数据压缩过程如下。①扫描 MapReduce 集群，以获取处于空闲状态的计算节点，基于负载均衡策略将 Map 端输出的数据分配给所有可用节点进行处理。②在内存中创建 3 种线程，分别是控制线程、压缩线程以及重构线程，使用信号量保护的共享内存进行 3 种线程间的信号交流。③将数据分块并输入内存，通过主控线程控制数据流向并初始化压缩线程，将所有的压缩线程相对均衡地分布到可用的处理器核心中，然后使用压缩线程各自独立地对数据块进行并行压缩。④使用重构线程获取各个压缩线程中生成的压缩数据块，并按输入顺序输出所有的压缩数据，最后在压缩数据输出到磁盘后进行解压供 Reduce 任务调用。

3) Reduce 阶段。在初始挖掘时，根据 Map 阶段的输出数据并行构造 Can 树，并使用 Hash 表记录所有数据项在树结构中的相对位置以加快数据

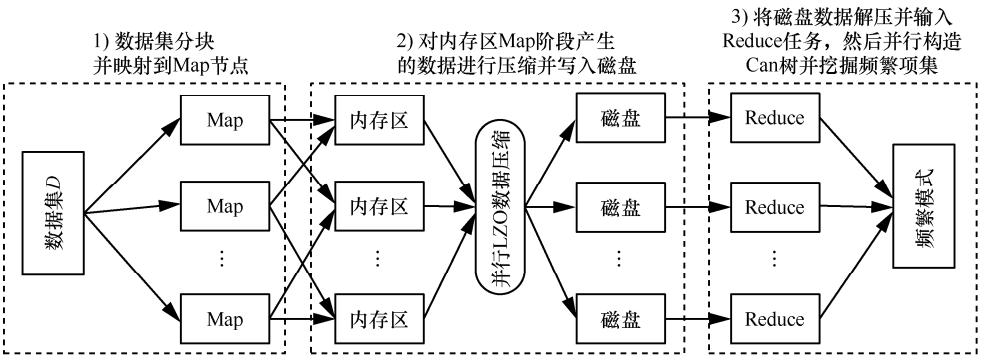


图 3 频繁项集并行挖掘过程

查找速度，最后根据动态支持度阈值从树结构中挖掘频繁项集；在增量挖掘时，根据 Map 阶段的输出数据对 Can 树以及存储数据项位置信息的 Hash 表进行更新，根据更新后的 Can 树和动态支持度阈值并行挖掘频繁项集。

频繁项集并行挖掘的执行过程伪代码如算法 3 所示。

算法 3　频繁项集并行挖掘

输入　数据集 D，支持度阈值 ST

输出　频繁 N 项集

1) Set　N=max ItemSize, freItemList=null //初始化频繁项集最大项项数、频繁项集列表

2) Map(key, value)

3) Scan the dataset D //数据集扫描

4)　D .division()//数据集划分

5) end

6) compress data by parallel LZO//压缩 Map 阶段的输出数据

7) data input the disk and unzip//数据输入磁盘并解压

8)　Reduce(key, value)

9)　construct or update the Can-tree//构建或更新 Can 树

10) for　item in Can-tree

11)　　for(int　i=1 ; $i < N+1$; $i++$)

12)　　if (item.size==i&&item.freq≥ST)//判断项集项数以及是否是频繁项

13)　　freItemList .add(item)

14)　　end if

15)　　end for

16) end for

17) end

18) return　freItemList

3.5　算法分析

对于本文提出的 MR-PARIMIEG 算法，其时间复杂度主要由相似项合并、动态支持度阈值获取以及频繁项集并行挖掘 3 个部分构成，这些阶段的时间复杂度如下所示。

1) 进行相似项合并时需要扫描数据集中的每个数据项并对数据集进行划分，然后计算相似度。令总数据项数为 n，划分数为 d，那么时间复杂度为 $T_1=O(d^2+n^2)$。

2) 获取动态支持度阈值时，根据遗传算法的遗传迭代获取最优解思想，令迭代次数为 M，其中每次迭代进行遗传演化时均需要对整个数据集进行处理，那么该阶段的时间复杂度为 $T_2=O(nM)$。

3) 关联规则并行挖掘阶段需要多个运算节点对数据进行压缩以加快内存与磁盘的数据交换过程，然后并行构建或更新 Can 树结构并从中获取频繁项集。令 MapReduce 分布式集群的节点数为 r，那么其时间复杂度为 $T_3=O(r\log n)$。

因此，对于 MR-PARIMIEG，其总的时间复杂度为 $T=O(d^2+n^2+nM+r\log n)$，其中 $r<<n$、$M<<n$，所以最终的时间复杂度近似为 $T=O(d^2+n^2)$。

对于 FPM-HCG 算法[8]，在数据准备阶段，由于其采用了滑动窗口技术来对数据进行扫描，然后将数据更新至 Can 树结构中，同时需要维持一个映射 Can 树进行频繁项挖掘的 GTree，令滑动窗口大小为 k，那么该阶段的时间复杂度为 $O(kn^2)$；在频繁项集并行挖掘阶段，由于未对磁盘与内存之间的数据传输过程进行优化，因此在集群节点数为 r 时，该阶段的时间复杂度为 $O(rn)$。因此，FPM-HCG 的时间复杂度近似为 $O(kn^2)$。

对于 IncbuildingPFP[10]算法，需要对数据进行 2 次扫描以获取所有的数据信息，该阶段的时间复杂度为 $O(n^2\log(n))$；在对频繁项进行并行挖掘时，其时间复杂度与 FPM-HCG 基本相同。因此，IncbuildingPFP 的时间复杂度近似为 $O(n^2\log(n))$。

对于 MR-PARIMIEG，由于 d 为对数据集进行划分的划分数，因此 $d<n$，故其时间复杂度 $O(d^2+n^2)<O(2n^2)$；对于 FPM-HCG，k 为滑动窗口大小，决定了每次对数据库进行扫描能保存的数据量，其数值小于 $\log(n)$ 但大于 2。由于 $O(d^2+n^2)<O(2n^2)<O(kn^2)<O(n^2\log(n))$，故相较于 FPM-HCG 算法和 IncbuildingPFP 算法，MR-PARIMIEG 算法具有更理想的时间复杂度。

4　实验与分析

4.1　实验环境

本节对本文提出的 MR-PARIMIEG 算法设计了一系列实验来验证其性能。具体的实验硬件配置为主从结构的分布式集群，其中包含一个主节点和 4 个从节点。所有节点的硬件配置均为 Intel core i9-9900K 八核处理器、16 GB 内存、1 TB 固态硬盘，各节点通过 500 Mbit/s 的网络进行连接，且均处于

同一局域网中；软件配置则统一安装 Ubuntu16 操作系统、jdk1.8 版本的 Java 环境以及 Hadoop2.7.5 分布式计算平台。

4.2 实验数据

本文中实验所用数据为 4 个真实的数据集，分别是 RetailRocket、Accident、Susy 和 Jester[20]。RetailRocket 是一个记录电子商务网站用户行为的数据集，记录了时间跨度为 4～5 个月的网站访问者的操作数据，操作分为商品点击浏览、加入购物车和购买等，总计有 2 756 101 个操作事件，数据量较多且较离散；Accident 是一个交通事故数据集，包含 340 183 条数据，数据量较少，数据分布相对均匀；Susy 是记录粒子探测数据的数据集，数据项数为 28，总的数据记录数为 500 万，数据项数少且数据长度相对均匀；Jester 是一个评分数据集，记录了 73 421 个用户对 100 个笑话给出的 410 万个评分数据，数据连续性强且取值区间较小。这些数据集的详细信息如表 2 所示。

表 2 数据集的详细信息

数据集	记录数/条	数据项数/个	规模/MB
RetailRocket	2 756 101	5	299.6
Accident	340 183	63	35.5
Susy	5 000 000	28	880.5
Jester	4 100 000	13	96

4.3 评价指标

1) 加速比

为了确定并行化算法对数据进行并行处理的效率，可以使用加速比作为评价指标以衡量算法的并行化运算能力。加速比是通过并行计算以降低总体的运行时间而获得的性能提升的数值化表示形式，其定义为

$$S_n=\frac{T_1}{T_n} \tag{9}$$

其中，T_1 表示算法在单节点中的运行时间；T_n 表示算法在 n 个节点中的运行时间；S_n 表示加速比，S_n 越大，说明算法的并行化效率越高。

2) 推荐非空率

推荐非空率是指在用户访问的所有项集中，能够给出有效推荐的被访问项集的比例，是一种对关联规则算法挖掘的频繁项的有效性进行评价的指标，能够对算法挖掘的所有频繁项的总体质量进行评估。推荐非空率的定义为

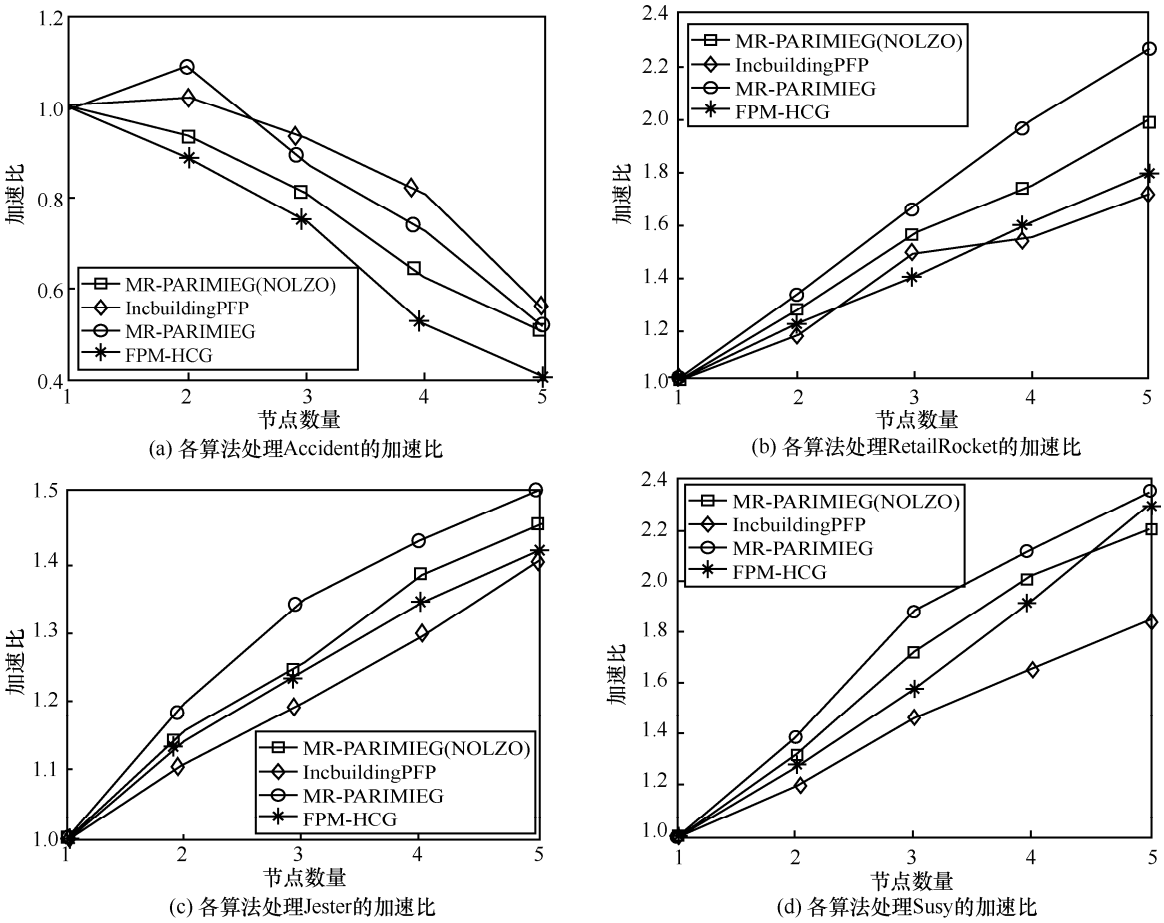
$$\mathrm{RecNon}=\frac{|\mathrm{UI}|\cap\mathrm{RI}}{|\mathrm{UI}|} \tag{10}$$

其中，UI 为用户访问的所有频繁项集的集合，RI 为给出了有效推荐的项集的集合，RecNon 为推荐非空率。

4.4 算法可行性分析

为验证 MR-PARIMIEG 算法、FPM-HCG[8]、IncbuildingPFP[10]算法在大数据环境中进行频繁项集挖掘的可行性，以及 LZO 压缩算法对运行效率的有效提升，以加速比作为衡量指标，应用上述 3 种算法以及不使用 LZO 数据压缩技术的 MR-PARIMIEG (NOLZO)分别对上述 4 个数据集进行处理。同时，为确保实验结果的准确性，各算法对每个数据集进行处理时均运行 10 次，并取最后运行时间的平均值作为实验结果，然后计算加速比。最终的实验结果如图 4 所示。

从图 4 中可以看出，在处理 RetailRocket、Susy 和 Jester 这样规模相对较大的数据集时，各算法运行的加速比随着节点数的增加而逐渐增加，并且 MR-PARIMIE 始终具有最高的加速比，而未进行数据压缩的 MR-PARIMIEG(NOLZO)的加速比始终小于 MR-PARIMIE。但在处理 Accident 这样规模相对较小的数据集时，仅有 MR-PARIMIEG、IncbuildingPFP 的加速比在 MapReduce 节点数为 2 时分别达到了 1.09 和 1.03，而在其他情况下各算法的加速比均随着节点数的增加而不断降低并且始终小于 1。这是因为各算法在对 Accident 这样规模相对较小的数据集进行并行化处理时，将数据分布到各个计算节点会导致各节点间的通信时间开销快速增长，而通过并行化运算获得的运行速度提升却极其有限。而在处理 RetailRocket、Susy 和 Jester 这样规模相对较大的数据集时，各算法的加速比随着节点数的增加而逐步上升。特别是在 Susy 和 RetailRocket 中，MR-PARIMIEG 的加速比在节点数为 5 时最高达到了 2.35 和 2.23，相比于节点数为 1 时分别提升了 1.35 和 1.23，相较于未进行数据压缩的 MR-PARIMIEG(NOLZO)分别提升了 0.14 和 0.26。这是因为在规模相对较大的数据集中，各算法通过高效的并行化运算减少总体运行时间的优势被逐渐放大，并且 MR-PARIMIEG 所采用的 LZO 数据压缩技术通过加快 Map、Reduce 阶段的数据传输过程有效地提升了算法的加速比。这说明 MR-PARIMIEG、FPM-HCG 和 IncbuildingPFP 算法适用于对规模相对较大的数据集进行并行化处理，



(a) 各算法处理Accident的加速比　(b) 各算法处理RetailRocket的加速比

(c) 各算法处理Jester的加速比　(d) 各算法处理Susy的加速比

图 4　各算法处理各数据集的加速比

并且 MR-PARIMIEG 使用的 LZO 数据压缩技术能有效提升算法运行效率。

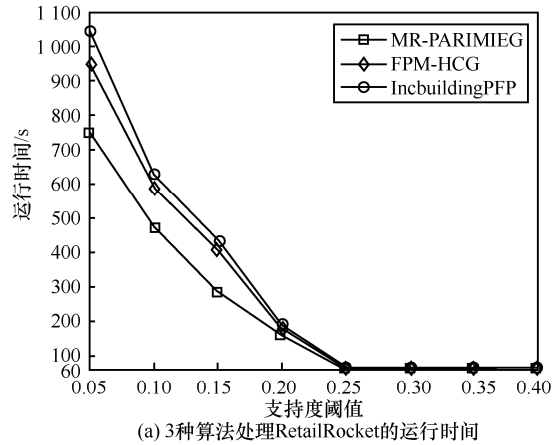
4.5　算法性能比较分析

1) 运行时间比较

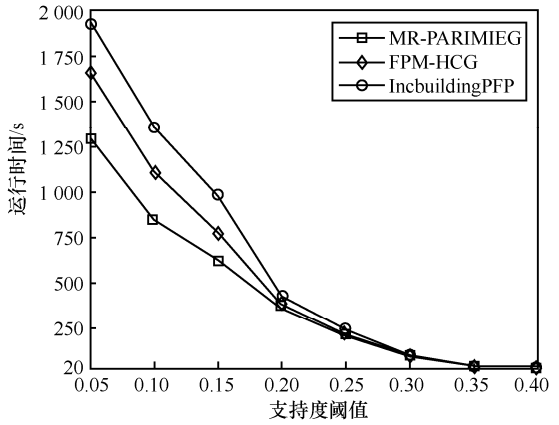
将 MR-PARIMIEG 分别与 FPM-HCG、IncbuildingPFP 算法在 RetailRocket、Susy 和 Jester 数据集中进行对比实验，实验比较了 MR-PARIMIEG、IncbuildingPFP 以及 FPM-HCG 算法在 MapReduce 节点数为 5 的集群中，在不同的初始支持度阈值下对数据集中的频繁项集进行挖掘所需的时间，实验结果如图 5 所示。

从图 5 可以看出，在对不同数据集的频繁项进行挖掘时，各算法的运行时间随着支持度阈值的增大而快速减少，MR-PARIMIEG 在 3 个数据集中始终具有最少的运行时间，并且支持度阈值越小，MR-PARIMIEG 相较于 FPM-HCG、IncbuildingPFP 在运行时间上的优势越明显。而当支持度阈值达到相对较高的数值时，各算法的运行时间差异并不明显。在数据分布较离散且相似项较多的 RetailRocket 数据集中（如图 5(a)所示），当初始支持度阈值为 0.05 时，MR-PARIMIEG 的运行时间相比于 IncbuildingPFP 和 FPM-HCG 分别降低了 28.05%和 21.27%。随着数据集规模的增长，在处理数据项长度相差较小的 Susy 数据集中（如图 5(b)所示），当初始支持度阈值为 0.05 时，MR-PARIMIEG 的运行时间相比 IncbuildingPFP 和 FPM-HCG 分别降低了 33%和 22.15%。在 Jester 这样数据较连续的数据集中（如图 5(c)所示），当初始支持度阈值为 0.05 时，MR-PARIMIEG 的运行时间相比于 IncbuildingPFP 和 FPM-HCG 分别降低了 30.58%和 27.74%。产生这种结果的主要原因如下。首先，MR-PARIMIEG 采用了 SIM-IE 策略来合并数据集中的相似项，通过减小数据集规模来加快后续 Can 树结构的构造；其次，MR-PARIMIEG 在并行挖掘频繁项集时，所采用的并行 LZO 数据压缩算法会对数据规模进行有效的压缩，从而显著加快了 Map 与 Reduce 阶段的数据传输过程；最后，MR-PARIMIEG 使用了 DST-GA 策略，根据具体的待挖掘增量数据设置相

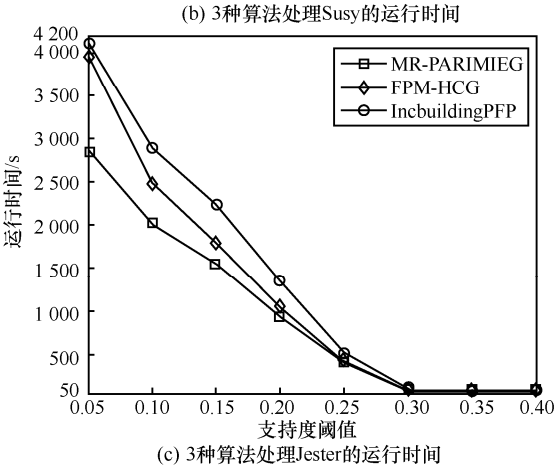
对合适的动态支持度阈值，并由此进行后续的增量频繁项集挖掘，在一定程度上避免了过多的频繁项集挖掘导致的时间消耗，从而减少了总体的算法运行时间。当支持度阈值设置为相对较大的数值时，满足该阈值的频繁项数量极少，因此各算法对数据集进行处理的运行时间差距较小。



(a) 3种算法处理RetailRocket的运行时间



(b) 3种算法处理Susy的运行时间

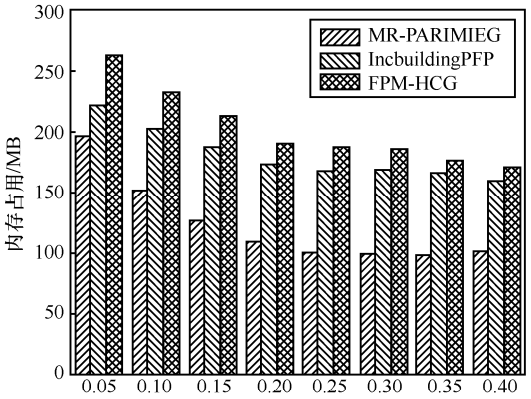


(c) 3种算法处理Jester的运行时间

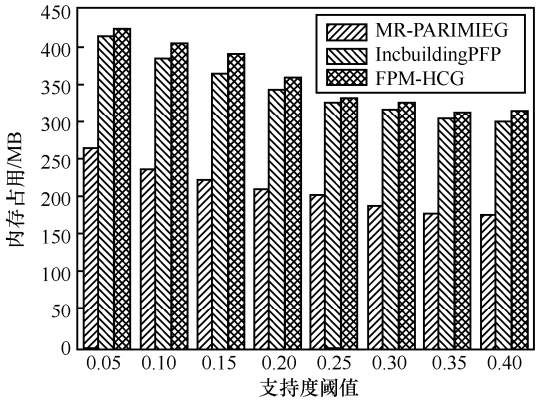
图 5　3 种算法处理各数据集的运行时间

2) 内存占用比较

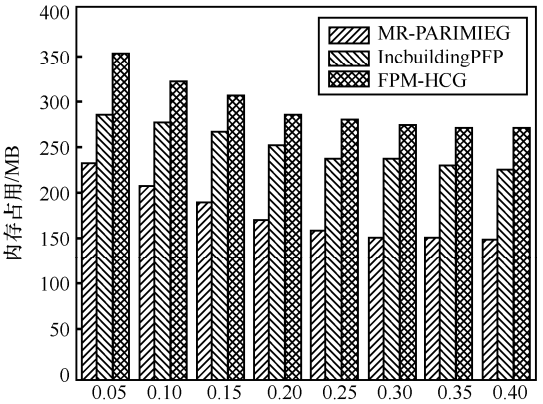
将 MR-PARIMIEG 算法分别与 FPM-HCG、IncbuildingPFP 算法在 RetailRocket、Susy 和 Jester 这 3 个数据集中进行对比实验，实验比较了各算法在不同的初始支持度阈值下对数据集进行挖掘的内存占用，实验结果如图 6 所示。



(a) 各算法处理RetailRocket的内存占用



(b) 各算法处理Susy的内存占用



(c) 各算法处理Jester的内存占用

图 6　3 种算法处理各数据集的内存占用

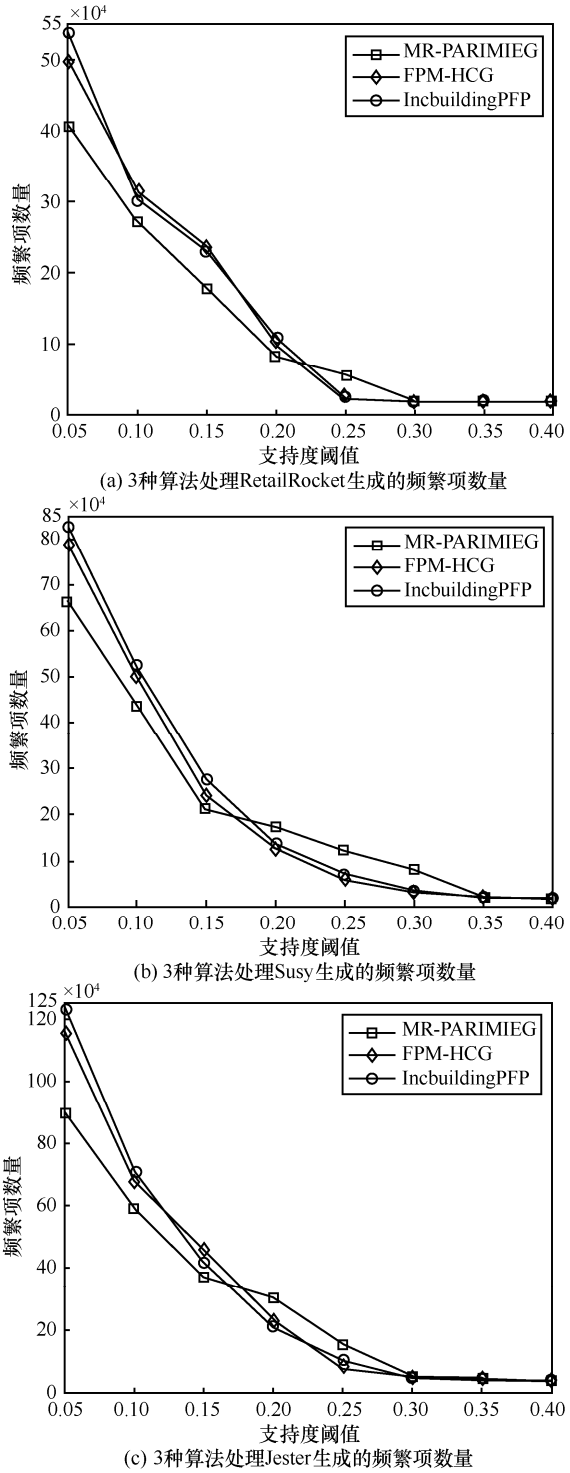
从图 6 可以看出，随着支持度阈值的增大，各算法的内存占用不断下降，并且在 3 种算法中，MR-PARIMIEG 在处理各数据集时始终具有最低的内存占用。在处理数据分布较离散的 RetailRocket 且初始支持度阈值为 0.4 时，MR-PARIMIEG 的内

存占用相比 IncbuildingPFP 和 FPM-HCG 分别降低了 36.7%和 41.86%；在处理数据比较连续的 Jester 且初始支持度阈值为 0.4 时，MR-PARIMIEG 的内存占用相比 IncbuildingPFP 和 FPM-HCG 分别降低了 33.04%和 42.73%；在处理数据规模相对最大的 Susy 且初始支持度阈值为 0.4 时，MR-PARIMIEG 的内存占用相比 IncbuildingPFP 和 FPM-HCG 分别降低了 40.6%和 43.45%。这一方面是因为 MR-PARIMIEG 采用了 SIM-IE 策略合并数据集中的相似项，使根据合并相似项的数据集生成的树结构的内存占用有明显的降低；另一方面则是因为在算法的并行化频繁项集挖掘阶段，使用了并行 LZO 数据压缩算法对 Map 端输出数据进行压缩，在数据量不变的情况下有效地减小了传输的数据规模，降低了最终的内存占用。IncbuildingPFP 算法虽然使用了分布式的方式对数据进行并行化挖掘，但并未解决 Can 树结构存储所有的数据信息而导致的空间占用过大的问题，导致算法最终的内存占用过大。FPM-HCG 算法使用了滑动窗口技术来加快数据项查找过程，但会使用额外的数据表来存储所有的已遍历数据项信息，这会导致在最终生成的树结构规模相同的情况下，还需要额外的内存空间来存储这些表结构，最终造成 FPM-HCG 在处理各数据集时均有最多的内存占用。

3) 频繁项数量比较

为了进一步有效地验证 MR-PARIMIEG 能够提高对频繁项的挖掘效率，并且具有相对较好的性能表现，在对各数据集处理的运行时间以及内存占用进行比较的基础上，对各算法在不同的初始支持度阈值下生成的频繁项数量进行了对比实验，实验结果如图 7 所示。

从图 7 可以看出，各算法挖掘的频繁项集数量随着支持度阈值的增大而逐渐减少。当支持度阈值相对较小时，FPM-HCG 算法和 IncbuildingPFP 算法所挖掘的频繁项集数量多于 MR-PARIMIEG 算法；当支持度阈值逐渐增大时，MR-PARIMIEG 算法挖掘的频繁项集数量反而多于其他 2 种算法；在初始支持度阈值相对较大时，各算法所挖掘的频繁项数量差距极小。其中，在对 RetailRocket 进行处理时，当初始支持度阈值为 0.05 时 MR-PARIMIEG 所挖掘的频繁项数量相较于 FPM-HCG 和 IncbuildingPFP 分别减少了 17.74%和 23.88%；当初始支持度阈值为 0.25 时，则分别增加了 134.58%和 139.57%。在对 Susy 进行处理时，当初始支持度阈值为 0.05 时，



(a) 3种算法处理RetailRocket生成的频繁项数量

(b) 3种算法处理Susy生成的频繁项数量

(c) 3种算法处理Jester生成的频繁项数量

图 7　3 种算法对各数据集进行处理生成的频繁项数量

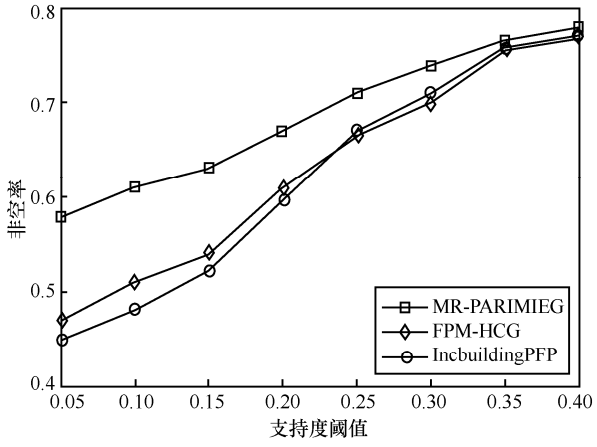
MR-PARIMIEG 所挖掘的频繁项数量相较于 FPM-HCG 和 IncbuildingPFP 分别减少了 16.41%和 20.43%；当初始支持度阈值为 0.2 时，则分别增加了 35.43%和 24.64%。在对 Jester 进行处理时，当初始支持度阈值为 0.05 时，MR-PARIMIEG 所挖掘的频繁项数量相较于 FPM-HCG 和 IncbuildingPFP 分别减

少了 22.43%和 27.48%；当初始支持度阈值为 0.2 时，则分别增加了 31.33%和 42.58%。这是因为 MR-PARIMIEG 在所指定的初始支持度阈值较小时，在增量挖掘阶段通过 DST-GA 策略获得了相对较大的阈值进行增量挖掘；而当所指定的初始支持度阈值逐渐增大时，DST-GA 策略在增量挖掘阶段又根据数据集信息设置了相对较小的阈值，最终根据该动态阈值获得了相对最多的频繁项数量。在初始支持度相对较大时，由于满足阈值的频繁项数量极少，因此各算法所挖掘的频繁项数量差距较小。同时，通过对比图 5～图 7 可以看出，即使是在支持度阈值逐渐增加的情况下，需要对更多的数据进行处理以挖掘相对较多的频繁项集时，MR-PARIMIEG 的运行时间和内存占用依然有着相对最优的表现，这也有效证明了 MR-PARIMIEG 具有最佳的频繁项挖掘效率。

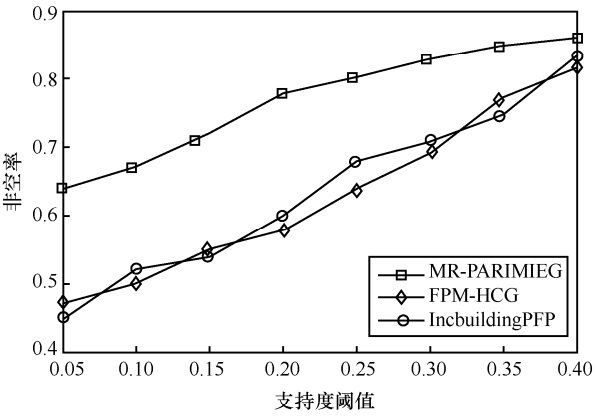
4) 推荐非空率比较

为了验证 MR-PARIMIEG 算法采用的 DST-GA 策略通过动态设置支持度阈值以挖掘更多高质量频繁项集的有效性，在比较各算法挖掘的频繁项数量的基础上，将推荐非空率作为评价指标对所挖掘的频繁项质量进行了比较，实验结果如图 8 所示。

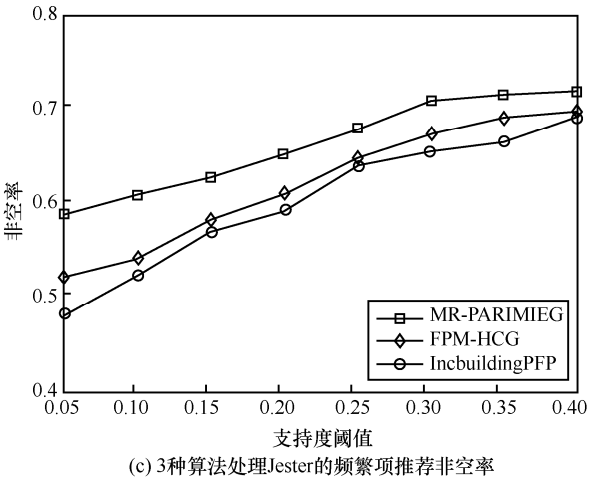
从图 8 可以看出，在对各数据集进行处理时，MR-PARIMIEG 所挖掘的频繁项始终有着最佳的推荐非空率，并且支持度阈值越小，其推荐非空率的优势相较于 IncbuildingPFP 和 FPM-HCG 越明显。其中，在 RetailRocket 中，当初始支持度阈值为 0.05 时，MR-PARIMIEG 的推荐非空率相比 FPM-HCG 和 IncbuildingPFP 分别增加了 0.11 和 0.13；在 Susy 中，当初始支持度阈值为 0.05 时，MR-PARIMIEG 的推荐非空率相比 FPM-HCG 和 IncbuildingPFP 分别增加了 0.13 和 0.15；在 Jester 中，当初始支持度阈值设置为 0.05 时，MR-PARIMIEG 的推荐非空率相比 FPM-HCG 和 IncbuildingPFP 分别增加了 0.07 和 0.12。这是因为 MR-PARIMIEG 采用了 DST-GA 策略，根据数据集信息实时对支持度阈值进行调整。当支持度阈值较小时，DST-GA 在增量挖掘阶段通过设置相对较大的阈值对冗余的无效频繁项集进行了剪枝；当支持度阈值逐渐增大时，又通过设置相对较小的阈值对数据集中的有效频繁项集进行了挖掘。因此，MR-PARIMIEG 对各数据集进行处理所产生的频繁项集始终具有相对较高的推荐非空率，即 MR-PARIMIEG 采用的 DST-GA 策略根据数据信息以及初始的支持度阈值动态调整阈值大小，既避免了初始支持度阈值较小时挖掘过多的无用频繁项集，又避免了过大的初始支持度阈值导致的有效频繁项丢失，使最终挖掘的频繁项集具有较高的质量。



(a) 3种算法处理RetailRocket的频繁项推荐非空率



(b) 3种算法处理Susy的频繁项推荐非空率



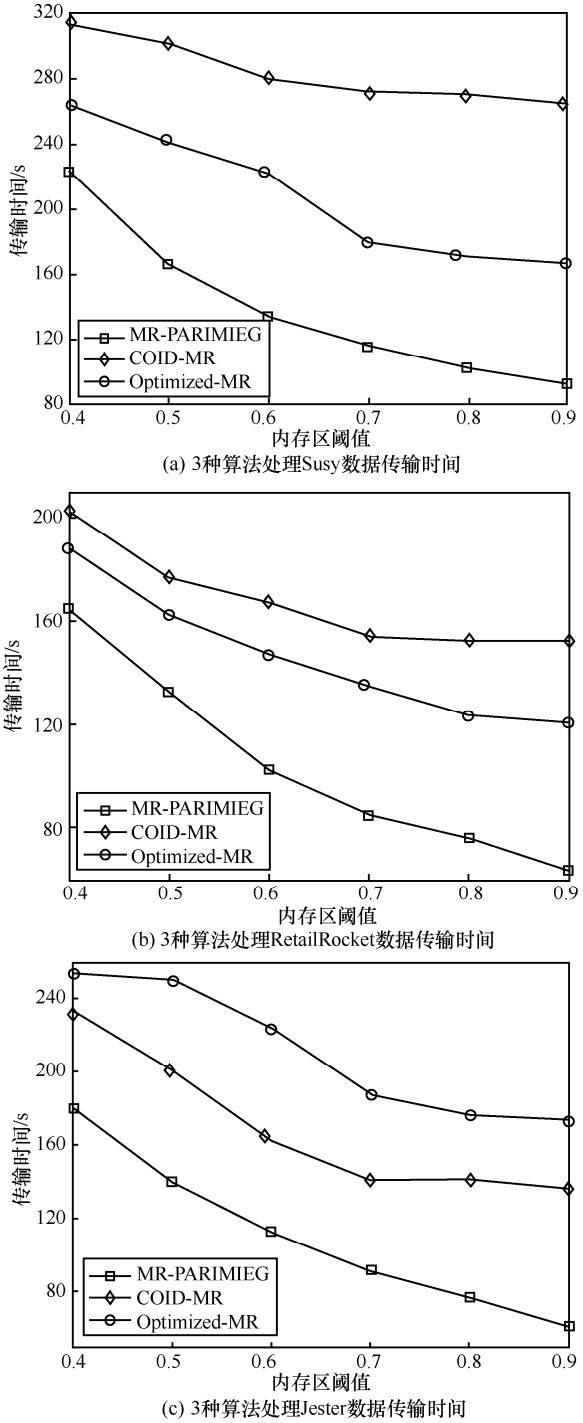
(c) 3种算法处理Jester的频繁项推荐非空率

图 8 各算法挖掘频繁项的推荐非空率

5) Map 与 Reduce 阶段数据传输时间比较

为验证本文所采用的并行 LZO 数据压缩算法的有效性，将 MR-PARIMIEG 分别与 Optimized-MR[13]、

COID-MR[14] 进行对比实验，比较各算法在 MapReduce 集群中对各数据集进行处理时，在不同的内存区阈值下 Map 与 Reduce 阶段的数据传输时间。同时，为保证实验结果的准确性，MR-PARIMIEG 在该实验中不使用 SIM-IE 策略对数据集中的相似项进行合并，确保各算法在处理同一数据集时 Map 端输出的总数据规模相同。实验结果如图 9 所示。



(a) 3种算法处理Susy数据传输时间

(b) 3种算法处理RetailRocket数据传输时间

(c) 3种算法处理Jester数据传输时间

图 9　3 种算法处理各数据集的数据传输时间

从图 9 可以看出，使用并行 LZO 数据压缩技术的 MR-PARIMIEG 算法在处理各数据集时，Map 与 Reduce 阶段的数据传输过程始终耗时最少，并且随着内存区阈值的增加，MR-PARIMIEG 的优势愈加明显。在对 Susy 进行处理且内存区阈值设置为 0.9 时，MR-PARIMIEG 的数据传输时间相比 Optimized-MR 和 COID-MR 分别减少了 43.98%和 64.82%；在对 RetailRocket 进行处理且内存区阈值设置为 0.9 时，MR-PARIMIEG 的数据传输时间相比 Optimized-MR 和 COID-MR 分别减少了 47.11%和 58.17%；在对 Jester 进行处理且内存区阈值设置为 0.9 时，MR-PARIMIEG 的数据传输时间相比 Optimized-MR 和 COID-MR 分别减少了 64.17%和 54.42%。这是因为并行 LZO 数据压缩采用了多线程技术，可以在单位时间内极大地提高数据压缩效率，并且对于数据大小的压缩率通常可以达到 50%左右，因此可以通过减少传输的数据规模来有效地提升数据传输速度，而数据在内存中的解压缩过程所消耗的时间与数据在磁盘与内存间进行传输所消耗的时间相比基本可以忽略不计。因此，MR-PARIMIEG 在对各数据集进行处理时，Map 和 Reduce 阶段的数据传输过程有着较少的时间消耗。

同时，从图 9 还可以看出，在对 Susy 和 RetailRocket 进行处理时，Optimized-MR 的表现好于 COID-MR；在对 Jester 进行处理时，Optimized-MR 却有着相对最差的性能表现。这是因为 Susy 和 RetailRocket 这类数据活跃度相差较小，并且在 Map 端每次产生的数据规模也相对较小，因此当内存区阈值较小时，Optimized-MR 每次进行磁盘与内存间的 I/O 操作所能传输的数据量随着内存区阈值的增加而快速增长，最终有效加快了数据传输速率；而 Jester 这类数据活跃度相差较大，COID-MR 所使用的基于数据活跃度的分类方式可以有效地对数据进行分类来优化传输流程，从而加快数据传输。相较于 Optimized-MR 和 COID-MR，MR-PARIMIEG 使用的并行 LZO 数据压缩算法的数据压缩过程不受 Map 端输出数据规模以及数据活跃度影响，并且在集群中其并行化压缩效率得到了极大地提升。因此，MR-PARIMIEG 的 Map 与 Reduce 阶段的数据传输过程始终有着最少的时间消耗。

5　结束语

针对传统的关联规则算法在大数据环境下的

不足，本文提出了一种并行关联规则增量挖掘算法 MR-PARIMIEG。首先，该算法提出了结合信息熵的相似项合并策略 SIM-IE 来对数据集中的相似项进行合并，以降低最终生成的 Can 树结构的复杂度以及空间占用；其次，使用 DST-GA 策略获取动态支持度阈值，并根据动态阈值进行关联规则挖掘，从而避免了冗余的频繁模式挖掘，提高了后续挖掘关联规则的速度与质量；最后，将频繁项集挖掘过程向 MapReduce 分布式计算平台迁移以并行化处理大数据集，同时使用并行 LZO 数据压缩算法对 Map 端输出数据进行压缩以加快 Map 与 Reduce 阶段之间的数据传输过程，最终提高算法的运行速度。为了验证 MR-PARIMIEG 的性能，本文设计了相关实验，在 RetailRocket、Susy 以及 Jester 这 3 个数据集中将 MR-PARIMIEG 分别与 FPM-HCG、IncbuildingPFP 进行比较。最终的实验结果与分析均反映出与其他基于 Can 树结构的并行化算法相比，本文提出的 MR-PARIMIEG 在处理大数据集时具有相对较好的性能表现，这也证明了 MR-PARIMIEG 的优越性。

参考文献：

[1] HAND D J, ADAMS N M. Data mining[EB]. (2015-06-22)[2020-11-13].

[2] KAMSU-FOGUEM B, RIGAL F, MAUGET F. Mining association rules for the quality improvement of the production process[J]. Expert Systems WithApplications, 2013, 40(4): 1034-1045.

[3] SÁNCHEZ D, VILA M A, CERDA L, et al. Association rules applied to credit card fraud detection[J]. Expert Systems WithApplications, 2009, 36(2): 3630-3640.

[4] BHANDARI A, GUPTA A, DAS D. Improvised apriori algorithm using frequent pattern tree for real time applications in data mining[J]. Procedia Computer Science, 2015, 46: 644-651.

[5] ZHANG W, LIAO H Z, ZHAO N. Research on the FP growth algorithm about association rule mining[C]//2008 International Seminar on Business and Information Management. Piscataway: IEEE Press, 2008: 315-318.

[6] LI Z F, LIU X F, CAO X. A study on improved eclat data mining algorithm[J]. Advanced Materials Research, 2011, 328: 1896-1899.

[7] LEUNG C K S, KHAN Q I, HOQUE T. CanTree: a tree structure for efficient incremental mining of frequent patterns[C]//Fifth IEEE International Conference on Data Mining. Piscataway: IEEE Press, 2005: 1-8.

[8] KUSUMAKUMARI V, SHERIGAR D, CHANDRAN R, et al. Frequent pattern mining on stream data using Hadoop CanTree-GTree[J]. Procedia Computer Science, 2017, 115: 266-273.

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

[10] SONG Y G, CUI H M, FENG X B. Parallel incremental frequent itemset mining for large data[J]. Journal of Computer Science and Technology, 2017, 32(2): 368-385.

[11] 胡军, 潘皓安. 基于 Can 树的关联规则增量更新算法改进[J]. 重庆邮电大学学报(自然科学版), 2018, 30(4): 558-563.
HU J, PAN H A. Improved incremental updating algorithm of association rules based on Can-tree[J]. Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition), 2018, 30(4): 558-563.

[12] RAGAVENTHIRAN J, KAVITHADEVI M K. Map-optimize-reduce: CAN tree assisted FP-growth algorithm for clusters based FP mining on Hadoop[J]. Future Generation Computer Systems, 2020, 103: 111-122.

[13] 申玲艳. MapReduce 计算模式的性能优化设计及其应用[J]. 信息与电脑(理论版), 2016(14): 49-50.
SHEN L Y. MapReduce computing model designed to performance optimization and applications[J]. China Computer & Communication, 2016(14): 49-50.

[14] CAO Y, WANG H. Communication optimisation for intermediate data of MapReduce computing model[J]. International Journal of Computational Science and Engineering, 2020, 21(2): 226-233.

[15] KIM J, HWANG B. Real-time stream data mining based on CanTree and Gtree[J]. Information Sciences, 2016, 367: 512-528.

[16] LIANG J, SHI Z, LI D, et al. Information entropy, rough entropy and knowledge granulation in incomplete information systems[J]. International Journal of General Systems, 2006, 35(6): 641-654.

[17] KOHLAS J, MONNEY P A. A mathematical theory of hints: an approach to the Dempster-Shafer theory of evidence[M]. Berlin: Springer Science & Business Media, 2013.

[18] KANE J, YANG Q. Compression speed enhancements to LZO for multi-core systems[C]//2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing. Piscataway: IEEE Press, 2012: 108-115.

[19] METAWA N, HASSAN M K, ELHOSENY M. Genetic algorithm based model for optimizing bank lending decisions[J]. Expert Systems With Applications, 2017, 80: 75-82.

[20] BART G. Frequent itemset mining dataset repository[EB]. (2020-03-01)[2020-11-13].

[作者简介]

毛伊敏（1970− ），女，新疆伊犁人，博士，江西理工大学教授、硕士生导师，主要研究方向为数据挖掘、大数据安全与隐私保护。

邓千虎（1998− ），男，湖北天门人，江西理工大学硕士生，主要研究方向为数据挖掘、大数据。

陈志刚（1964− ），男，湖南益阳人，博士，中南大学教授、博士生导师，主要研究方向为网络与分布式计算、机会网络。