引用格式:朱颢东,薛校博,李红婵,等. 海量数据下基于 Hadoop 的分布式 FP-Growth 算法[J]. 轻工学报,2018,33(5):97 – 102.
中图分类号:TP301 **文献标识码**:A
DOI:10.3969/j.issn.2096 – 1553.2018.05.013
文章编号:2096 – 1553(2018)05 – 0097 – 06

海量数据下基于 Hadoop 的分布式 FP-Growth 算法

Distributed FP-Growth algorithm based on Hadoop under massive data

朱颢东,薛校博,李红婵,孟颍辉
ZHU Haodong, XUE Xiaobo, LI Hongchan, MENG Yinghui

郑州轻工业学院 计算机与通信工程学院,河南 郑州 450001
School of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou 450001, *China*

关键词:
FP-Growth 算法;Hadoop;数据分区;分布式计算

Key words:
FP-Growth algorithm; Hadoop; data partition; distributed computing

摘要:针对大数据环境下的关联挖掘问题,采取两次扫描数据库,将事务添加到相互独立的数据分区的方式,对传统 FP-Growth 算法进行分布式改造,进而提出了基于 Hadoop 框架的分布式 FP-Growth 算法以实现海量数据的频繁模式 FP 挖掘. 仿真结果表明,在数据处理量逐渐增大的过程中,该算法相比较传统算法其运行时间和内存消耗的优势愈加明显,当数据处理量达到 70 万条时,该算法比传统算法节省约 2/3 的运行时间,而内存消耗仅为传统算法的 1/5. 说明该算法在处理海量数据时,能够显著提高 FP 的挖掘效率并降低内存的消耗量.

收稿日期:2018 – 05 – 16
基金项目:国家自然科学基金项目(61501405);河南省科技计划项目(152102210149,152102210357);郑州轻工业学院校级青年骨干教师培养对象资助计划项目(XGGJS02);郑州轻工业学院研究生科技创新基金资助项目

作者简介:朱颢东(1980—),男,河南省虞城县人,郑州轻工业学院副教授,博士,主要研究方向为智能信息处理、智能计算.

Abstract: In view of the large data problem of association mining by the method of taking two times of scanning database and adding the transaction to the independent data partition, distributed renovation of traditional FP-Growth algorithm was taken, the distributed FP-Growth algorithm based on Hadoop framework was then put forward so as to realize the frequent pattern FP huge amounts of data mining. The simulation results showed that in the process of increasing data processing, the algorithm was compared with the traditional algorithm advantages of its running time and memory consumption were becoming ever more obvious. When the amount of data processing reached 700,000 items, the algorithm saved about 2/3 running time than the traditional algorithm, while the memory consumption was only 1/5 of the traditional algorithm. It showed that the algorithm could significantly improve the mining efficiency of FP and reduced the memory consumption when dealing with massive data.

0　引言

在大数据环境下,传统的关联挖掘算法难以满足要求,面临的主要难题有:单机情况下爬取海量数据消耗的时间太长,人们往往难以忍受[1];单台服务器的存储空间远远无法满足海量数据的存储要求[2];单机根本无法胜任对中间结果的计算[3]. 因此,利用分布式存储和计算框架来处理海量数据下传统挖掘算法的并行挖掘工作遂成为研究热点.

FP-Growth 算法是 J. W. Han 等[4]在 2000 年提出的关联规则挖掘算法,它把数据存储在一种称为频繁模式 FP(frequent pattern)树的紧凑数据结构中. 目前,国内外已有诸多关于FP-Growth 算法分布式方面的研究成果. 黄明[5]使用格网划分结构对数据进行分区,其思想是使用平行于各数据轴的若干条分割线对数据空间进行划分,格网划分结构将数据空间划分成多个均匀或者不均匀的子数据空间,并且能够方便地表征这些子数据空间的大小和位置. 这种划分结构可以使数据分区聚类的处理更加方便和直接. 该分区方法虽然能保证各数据分区相互独立,但会导致数据有较高的冗余度. 邱勇等[6]为了提高传统的 FP-Growth 算法的挖掘效率,提出了一种新的并行频繁模式 FP-Tree 构造算法 (PFPTC),该算法并发地创建多个子 FP-Tree,但这实际上并不能算作真正意义上的并行 FP-Growth 算法,而是一种 FP-Growth 算法内部各 FP-Tree 的合并,虽然一定程度上提高了数据挖掘能力,但挖掘效率和预期效果相距甚远[7]. 茹蓓等[8]提出了针对少量实时数据流完全频繁项集的改进 FP-Growth 算法,使用改进的 FP-Tree 兼容地表示滑动窗口中的所有事务,建立一个完整的基树;利用事务的字母顺序实现基树的插入与删除操作;利用分组 Tree 结构对基树进行由上而下的遍历来建立项目树,以较低的计算成本发现完全的频繁项集. 这种改进算法处理的是实时数据流,且基于最小支持度和运行时间这两个参数,确实降低了计算成本,但它通过遍历来处理数据,是一种单向、串行的方式,处理速率较慢,内存占用较大,容易使集群卡顿从而影响数据处理的效率,并不适合处理海量数据.

鉴于此,本文拟提出一种海量数据下基于 Hadoop 的分布式 FP-Growth 算法:通过两次扫描数据库,将事务添加到所对应的数据分区中,同时数据库被划分成几个独立的数据分区,在很大程度上降低了数据的冗余度;在 Hadoop 框架下,几个分区同时进行 FP 的挖掘,把不同机器上的结果聚合起来,筛选出低于阈值的频繁项集,以期解决海量数据下庞大的 FP-Tree 无法驻留在内存的问题,高效地实现在 Hadoop 框架下 FP 的分布式挖掘.

1 基于 Hadoop 的分布式 FP-Growth 算法

1.1 传统的 FP-Growth 算法

对于输入的事务数据库 D,统计所有事项中各元素的出现频次,即每个 1 项集的频数,并将各元素按照频数降序排列,删除那些出现频数少于设定支持度 min_sup 的元素,形成列表 L,留下来的元素就构成了频繁 1 项集(这是对数据集的第一遍扫描). 对数据集中每个事务的元素按照列表 L 排序(按支持度降序排列),开始构造 FP-Tree. 树的根节点为空,每个事务中的所有元素形成一条从根节点到子节点的路径. 若几个事务的元素按列表 L 排序后,具有相同的前 N 个元素,则它们在 FP-Tree 中共享前 N 个元素代表的节点. 树中每个节点的计数为路径经过该节点的事务集的个数(这是对数据集的第二遍扫描). 整个挖掘的过程分为两步:首先,扫描数据库的所有事务以获得频繁项集,依据频繁项集构造项目头表,根据项目头表构造出 FP-Tree;然后,挖掘 FP-Tree 中的所有频繁项集[9]. 其流程如图 1 所示.

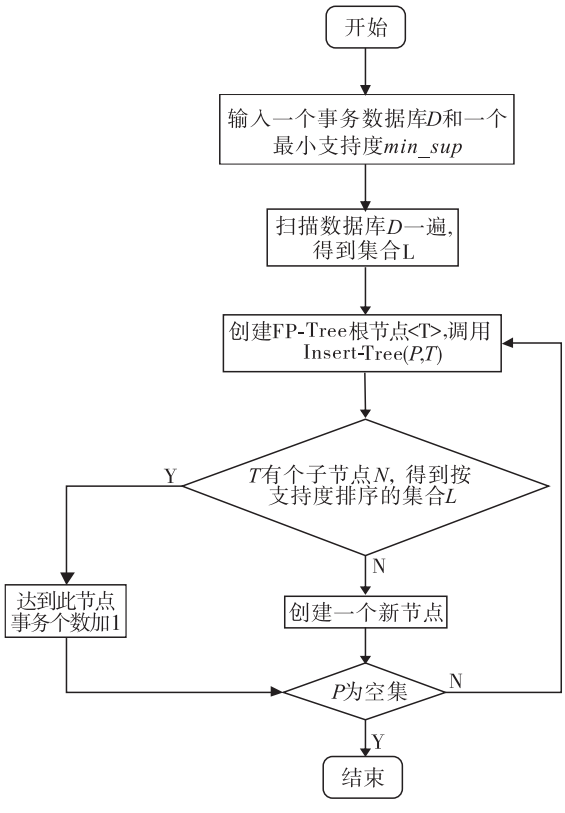

图 1 FP-Growth 算法流程图

Fig. 1 FP-Growth algorithm flow chart

1.2 基于 Hadoop 的分布式 FP-Growth 算法

1.2.1 算法思路 传统的 FP-Growth 算法的核心思想是根据原始数据库在内存中构造出 FP-Tree 这个精巧的数据结构,再对 FP-Tree 不断地进行递归挖掘[10]以得到完备的 FP. 但在海量数据的现状下,FP-Tree 已经大到无法保存在计算机的内存中,这就不可能通过不断地递归挖掘来获得频繁项集. 因此,并行化是唯一的选择.

既然传统的 FP-Growth 算法不能应对海量数据的挖掘任务,就需要对其进行分布式改造[11]. 首先扫描一次数据库,得到每个项的出现频率(即频繁项),删除频率小于最小支持度[12]的项,依据频繁项构造一个频繁项列表 F_list,并将 F_list 分组,形成对应数量的 G_list;第二次扫描数据库,取出第一条事务数据[13],构建 FP-Tree 的第一条路径(项的排序与第一次扫描中得到的频繁项集合的排序一致),然后取出第二条事务数据,如果前面节点上有与第一条路径相同的元素,则该节点所在元素数目值加 1,如果后面的节点上没有相同的元素,则创建新的路径. 依次类推,取出数据库中剩余的若干条事务数据,就形成了含有若干子节点的 FP-Tree 数据结构. 这样,数据库中的事务数据就被压缩成为简洁的 FP-Tree 数据结构,G_list 则被包含在对应的数据分区中,数据库被划分成了几个数据分区. 这种对数据库的分区方法在保证各分区独立性的同时,也在很大程度上降低了数据的冗余度[14].

在 Hadoop 框架下,几个分区同时进行 FP 挖掘,最后通过主节点将统计结果进行聚合,筛

选出不低于阈值的数据. 这样就可以解决海量数据下庞大的 FP-Tree 无法驻留在内存的问题,高效地实现在 Hadoop 框架下 FP 的分布式挖掘.

1.2.2　FP-Growth 算法的算法流程　在 Hadoop框架中,其 MapReduce 主要由 Map 和 Reduce 这两个函数组成[15],Map 函数把任务分解成多个子任务,Reduce 函数把分解后的多个子任务的处理结果聚集起来,Hadoop 的工作示意图如图 2 所示.

在 Hadoop 框架下,基于 Hadoop 的分布式 FP-Growth 算法流程图如图 3 所示,具体步骤如下.

步骤 1　数据库分区:将原始的整个数据库进行分区,这些分区是连续的,每一台服务器上都有一个分区,这些分区称作 *shard*.

步骤 2　计算 *F_list*,也就是所有 *item* 的 *support count*,通过 Hadoop 框架来完成.

步骤 3　*item* 分组:将 *F_list* 里的 *item* 分成 *N* 组,行成一个 *group_list*,其中的每一个 *group* 都被分配一个 *group_id*,每个 *group_list* 都包含一组 *item* 的集合,去除结果中常用的单字词,将剩下的词语序列放入候选词集中.

步骤 4　FP-Growth 算法的分布式操作:在 Hadoop 框架中,利用第一步 *shard* 的数据库分区,逐个地去处理所在数据库分区中的每一条事务,将单个的事务 *transaction* 分成多个 *item*,根据 *group_list* 把单个 *item* 映射到所对应的 *group* 里. 依照这样的映射规则,利用 Map 函数,将相同 *group* 的 *item* 集合聚合到一台服务器上,产生完备数据集. 接下来利用 Reduce 函数处理上一步获得的中间结果.

步骤 5　聚合:通过主节点将各从机的统计结果进行聚合,筛选出不低于阈值的数据.

2　仿真实验

2.1　实验平台与测试数据

为了验证本文算法的有效性,将其与传统的 FP-Growth 算法和 PFPTC 算法进行对比实验. 实验平台为 4 台安装 Linux 系统服务器的 Hadoop 2.0.4 平台,主设备节点采用 Intel i7 处理器,内存为 8 GB;从设备节点采用 Intel i7 处理器,主频为 3.6 GHz,内存为 4 GB. 测试数据是用 python 爬虫程序爬取的 30 万条微博数据和通过新浪微博官方 API 接口获取的 40 万条微博数据,共计 70 万条微博数据,将微博的用户数据和发布的微博内容从网络上下载到本地数据库存储,在存入数据库之前要经过数据的

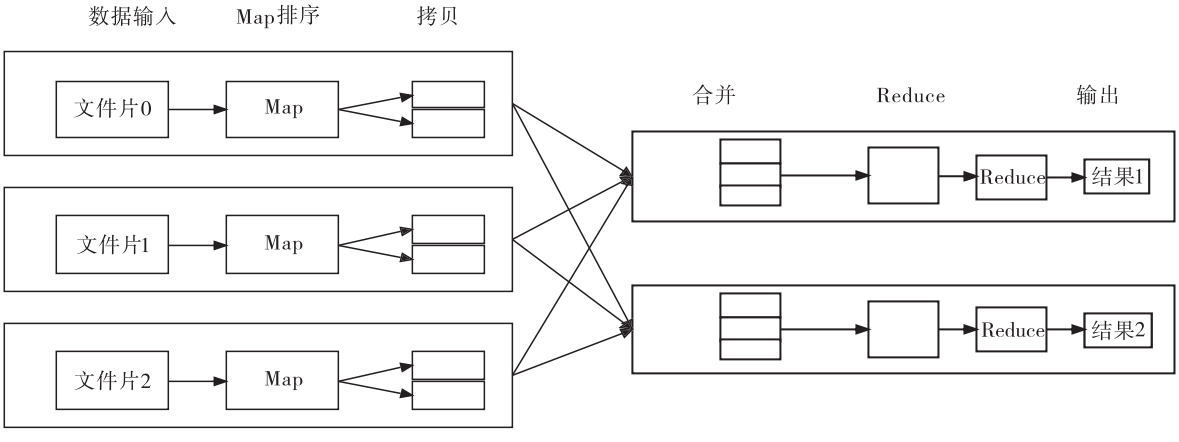

图 2　Hadoop 工作示意图

Fig. 2　Schematic diagram of Hadoop work

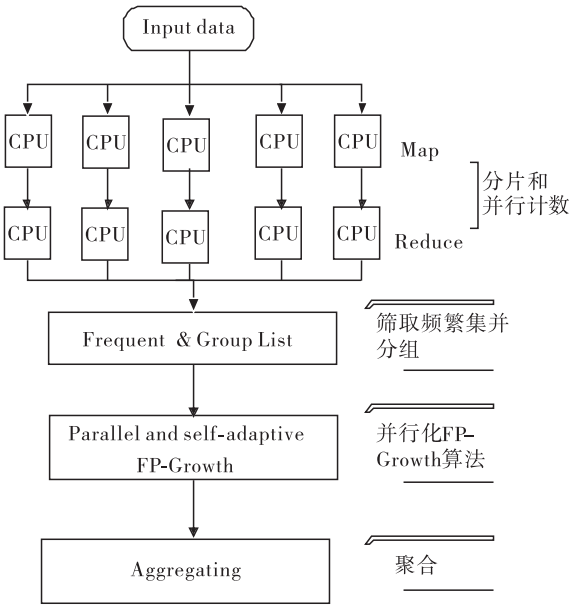

图 3 基于 Hadoop 的分布式 FP-Growth 算法流程图

Fig. 3 Flow chart of distributed FP-Growth algorithm based on Hadoup

筛选和清洗等操作. 网络爬虫能够从新浪微博中将所需要数据下载并通过一系列操作存入到磁盘中,爬取数据和清洗后的数据留作分析使用.

2.2 实验结果与分析

处理 10 ~ 70 万条微博数据的事务,将采用传统 FP-Growth 算法、PFPTC 算法与本文算法所需的运行时间和内存消耗量进行比较,结果分别见图 4 和图 5.

从图 4 可以看出,数据量仅为 10 万条时,3 种算法在运行时间上差别不大,主节点耗时统计结果(主节点只负责显示耗时量和内存消耗量,并不处理数据)分别为 15 s,10 s 和 6 s. 当处理的数据为 40 万条时,传统 FP-Growth 算法耗时44 s,每个从节点大约耗时十几 s;PFPTC 的算法耗时 24 s,各从节点大约耗时 6 s;本文算法耗时仅为 13 s,每个从节点大约耗时3 ~ 4 s. 当数据量达到 70 万条时,本文算法仅用时 20 s,而传统的 FP-Growth 算法耗时达到了

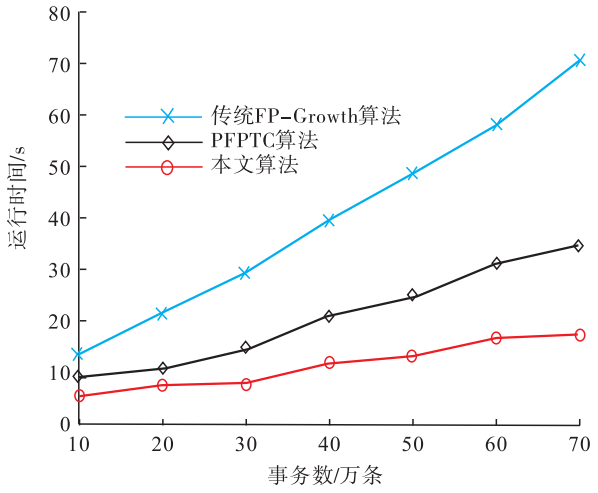

图 4 3 种算法的运行时间对比结果

Fig. 4 Running time comparison results of three algorithms

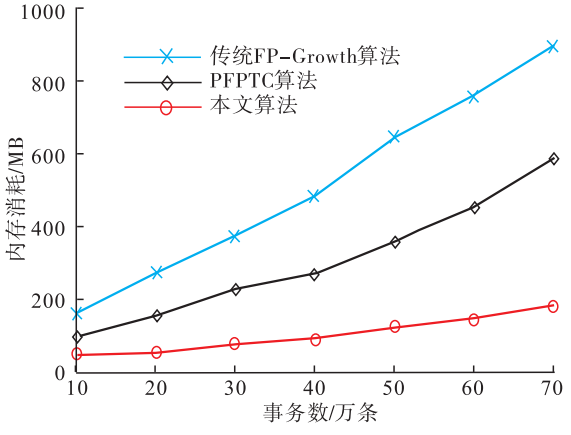

图 5 3 种算法的内存消耗对比结果

Fig. 5 Comparison of memory consumption between three algorithms

79 s,本文算法比传统的 FP-Growth 算法节省了近 2/3 的时间. 从图 4 中 3 条线的斜率还可以看出,随着处理数据量的增加,传统 FP-Growth 算法耗时的斜率最大,改进算法适中,而本文算法增速趋于平缓. 由此可知,本文并行算法在处理海量数据时,在运行时间上更具优势.

从图 5 可以看出,数据量为 10 万条时,使用 3 种算法处理所消耗的内存已有所差异. 主节点统计结果为:传统 FP-Growth 算法消耗了 158 MB 内存,PFPTC 算法为 100 MB 内存,而本

文算法仅消耗 47 MB 内存. 随着处理数据量的增长,当处理的数据量达到 70 万条时,传统 FP-Growth 算法消耗 893 MB, PFPTC 算法消耗 587 MB,本文算法消耗 183 MB,传统算法消耗内存是本文算法的 5 倍之多. 从图 5 还可以看出,传统 FP-Growth 算法和 PFPTC 算法处理数据所占用消耗的内存增速比较明显,而本文算法处理时对内存的消耗并没有明显的增长,这表明在处理海量数据时,并行处理算法是更具优势的.

分析其原因,本文算法与传统的 FP-Growth 算法和 PFPTC 算法相比,其主要差异在于对输入的原始数据进行分区,各分区的微博数据分别在集群的各个从节点上,利用 Hadoop 分布式框架进行实现. 这样,在海量数据下就实现了庞大数据结构 FP-Tree 的“拆分”,使其可以驻存于各服务器的内存中. 这不仅完成了在海量数据下频繁项集的挖掘任务,而且利用 Hadoop 并行框架能显著提高算法的执行效率,降低单个节点计算过程中的计算量,也降低处理数据时对内存的消耗量,从而提高算法的执行效率.

3　结语

针对海量数据下传统的 FP-Growth 算法所构建的 FP-Tree 太大而增加挖掘负担和占用内存较多的问题,本文提出了基于 Hadoop 框架的分布式 FP-Growth 算法. 该算法首先对数据分区,扫描一次数据库,构造出一个频繁项列表并分组,之后再次扫描数据库中的每一条事务,如果该事务包含了一个组中的项,那么这条事务就被添加到这个组所对应的数据分区中,而每个数据分区由组和组列表构成,这样所获得的数据就具有完备性,而且几个数据分区也相互独立;然后对不同的数据分区在 Hadoop 框架下进行 FP-Growth 算法的分布式挖掘,由 Map 函数获得中间结果,Reduce 函数输出结果;最后主节点把各从节点的结果聚合,筛选出低于阈值的频繁项集,从而完成在海量微博数据下的挖掘工作. 仿真实验结果表明,处理同样的微博数据时,在得到相等的频繁项数的情况下,本文算法的挖掘效率和对内存的消耗量明显优于传统的 FP-Growth 算法和 PFPTC 算法. 但本文没有考虑各节点的负载均衡,有可能造成有些节点“负担过重”,而有些节点“过于清闲”,从而影响到执行效率,这也是后续工作需要研究的重点.

参考文献:

[1] 刘智勇. 关联规则挖掘的并行化算法研究[D]. 南京:东南大学,2016.

[2] 董金凤. 数据挖掘中关联规则算法的改进与并行化处理[D]. 哈尔滨:哈尔滨理工大学,2016.

[3] 孙兵率. 基于 MapReduce 的数据挖掘算法并行化研究与应用[D]. 西安:西安工程大学,2015.

[4] HAN J W, PEI J, YIN Y W. Mining frequent patterns without candidate generation[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data, New York: ACM, 2000:1.

[5] 黄明. 基于空间分区的空间聚类研究[D]. 武汉:武汉大学,2010.

[6] 邱勇,兰永杰. 高效 FP-TREE 创建算法[J]. 计算机科学,2004(10):98.

[7] 赵兰草. QAR 数据的异常检测与分析算法研究[D]. 天津:中国民航大学,2014.

[8] 茹蓓,贺新征. 高效的数据流完全频繁项集挖掘算法[J]. 计算机工程与设计,2017,38(10):2759.

[9] 王翔. 基于云计算棉花仓储海量数据挖掘算法研究[D]. 北京:首都师范大学,2014.

(下转第 108 页)

13.

[4] VELIKOVICH L, BLAIR-GOLDENSOHN S, HANNAN K, et al. The viability of web-derived polarity lexicons[C]// Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles: NAACL, 2010: 777.

[5] PANG B, LEE L, VAITHYANATHAN S. Thumbs up sentiment classification using machine learning techniques[C]// Proceedings of Conference on Empirical Methods in Natural Language Processing, Philadelphia: ACL, 2002: 79.

[6] KIM S M, HOVY E H. Automatic identification of pro and con reasons in online reviews[C]// Proceedings of the Meeting of the Association for Computational Linguistics, Sydney: ACL 2006: 483.

[7] READ J, CARROLL J. Weakly supervised techniques for domain-independent sentiment classification[C]// Proceedings of Conference on Information and Knowledge Management, Hongkong: ACL, 2009: 45.

[8] BENGIO Y, DUCHARME R, VINCENT P, et al. A neural probabilistic language model[J]. The Journal of Machine Learning Research, 2003 (3): 1137.

[9] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural Computarion, 1997, 9(8): 1735.

[10] COLLOBERT R, WESTON J. A unified architecture for natural language processing: deep neural networks with multitask learning[C]// Proceedings of International Conference on Machine learning, Helsinki: ICML, 2008: 160.

[11] KIM Y. Convolutional neural networks for sentence classification[C]// Proceedings of Empirical Methods in Natural Language Processing, Doha: ACL, 2014: 1746.

(上接第 102 页)

[10] 周诗慧. 基于 Hadoop 的改进的并行 Fp-Growth 算法[D]. 济南: 山东大学, 2013.

[11] 邵伟. 基于 FP-Tree 的关联规则挖掘算法研究[D]. 西安: 西安电子科技大学, 2010.

[12] 朱付保, 白庆春, 汤萌萌, 等. 基于 MapReduce 的数据流频繁项集挖掘算法[J]. 华中师范大学学报(自然科学版), 2017, 51(4): 429.

[13] 白川平, 杨志翀. 基于加权滑动窗口的数据流频繁项集挖掘算法[J]. 宁夏师范学院学报, 2017, 38(6): 49.

[14] 胡健, 吴毛毛. 一种改进的数据流最大频繁项集挖掘算法[J]. 计算机工程与科学, 2014, 36(5): 963.

[15] 刘慧婷, 候明利, 赵鹏, 等. 不确定数据流最大频繁项集挖掘算法研究[J]. 计算机工程与应用, 2016, 52(19): 72.