分布式并行化数据流频繁模式挖掘算法

马 可,李玲娟,孙杜靖
(南京邮电大学 计算机学院,江苏 南京 210003)

摘 要:为了提高数据流频繁模式挖掘的效率,文中基于经典的数据流频繁模式挖掘算法 FP-Stream 和分布式并行计算原理,设计了一种分布式并行化数据流频繁模式挖掘算法—DPFP-Stream (Distributed Parallel Algorithm of Mining Frequent Pattern on Data Stream)。该算法将建立频繁模式树的任务分为 local 和 global 两部分,并设置了参数“当前时间”;将到达的流数据平均分配到多个不同的 local 节点,各 local 节点使用 FP-Growth 算法产生该单位时间内本节点的候选频繁项集,并按照单位时间将候选频繁项集及其支持度计数打包发送至 global 节点;global 节点按“当前时间”合并各 local 节点的中间结果并更新模式树 Pattern-Tree。在分布式数据流计算平台 Storm 上进行的算法实现和性能测试结果表明,DPFP-Stream 算法的计算效率能够随着 local 节点或 local bolt 线程的增加而提高,适用于高效挖掘数据流中的频繁模式。

关键词:数据流;频繁模式;分布式并行化;Storm
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2016)07-0075-05
doi:10.3969/j.issn.1673-629X.2016.07.16

Distributed Parallel Algorithm of Mining Frequent Pattern on Data Stream

MA Ke,LI Ling-juan,SUN Du-jing
(School of Computer,Nanjing University of Posts and Telecommunications,Nanjing 210003,China)

Abstract:In order to improve the efficiency of mining frequent pattern on data stream,a Distributed Parallel Algorithm of Mining Frequent Pattern on Data Stream,named DPFP-Stream,is designed in this paper based on the ideas of classical FP-Stream and the distributed parallel computing. It divides the task of building frequent pattern tree into two parts:local and global,and introduces a new parameter “current time”. The arrival data will be equally distributed into different local nodes. Then every local node uses FP-Growth algorithm to produce candidate frequent items,and packages them with relevant support count according to unit time,and sends them to the global node. The global node combines the results produced by local nodes according to the “current time” and updates the global Pattern-Tree. The results of implementing DPFP-Stream algorithm and testing its performance on Storm,a distribution data stream computing platform,show that the computing efficiency of DPFP-Stream can increase linearly with the increasing of local nodes or the local bolts,and DPFP-Stream is applicable to effectively mine frequent pattern from data stream.

Key words:data stream;frequent pattern;distributed parallelization;Storm

0 引 言

数据流是按时间顺序到达的数据所组成的一个序列,其中的数据是动态的,数据量潜在无界、数据到达速率快。对此类数据的收集过程和挖掘过程是同时进行的,不允许反复扫描历史数据,需要用一次扫描算法(single-scan algorithm)来处理[1]。

流数据的挖掘有分类、聚类、关联分析等多种任务[2-4]。在流数据的关联规则挖掘算法中,经典的 FP-Stream 算法实现了对流数据的频繁模式挖掘。该算法将挖掘任务分为在线挖掘单位时间的候选频繁项集与离线处理历史频繁项集两个部分,通过倾斜时间框架存储候选频繁项集,并可以按照用户输入的参数查询相应时间的频繁项集[5-7]。

MapReduce 是一种分布式计算框架,将一个算法抽象成 Map 和 Reduce 两个阶段进行处理,非常适合数据密集型计算,但是它是批处理的。Storm 是一种典型的在线流式数据分布式计算架构,可以用来在线处理源源不断流进来的数据,也可以通过设置滑动时

收稿日期:2015-10-10 **修回日期**:2016-01-20 **网络出版时间**:2016-06-22
基金项目:国家自然科学基金资助项目(61302158,61571238);中兴通讯产学研项目
作者简介:马 可(1991-),男,硕士研究生,CCF 会员,研究方向为流数据挖掘、信息安全;李玲娟,教授,CCF 会员,通讯作者,研究方向为数据挖掘、信息安全、分布式计算。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20160621.1701.014.html

间窗口等机制,在实时处理到达数据的同时,实现类似 MapReduce 的功能[8]。

文中首先基于经典数据流频繁模式挖掘算法 FP-Stream[9]和分布式并行计算的思想,设计了一种分布式并行化数据流频繁模式挖掘算法(Distributed Parallel algorithm of mining Frequent Pattern on data Stream, DPFP-Stream)。接着,考虑到将流挖掘算法部署到流平台上运行是算法实用化的前提,进一步基于 Storm 集群进行了 DPFP-Stream 算法的实现。为了评价该算法的性能,设计了线程处理压力测试实验,并分析了实验效果。

1 FP-Stream 算法分析

FP-Tree(频繁模式树)是 FP-Growth 算法建立的一种数据结构,虽然它不能直接用于数据流的关联规则挖掘,但通过对 FP-Tree 加以改进,可以将其运用在数据流上[10-11]。基于此思想,Giannella. C 等提出了 FP-Stream 模型,将频繁模式挖掘算法分为挖掘单位时间频繁项集与记录各时间段频繁项集两个部分[12]。第一部分设置了参数最大支持度误差(该值小于挖掘频繁项集的最小支持度),使用 FP-Growth 算法对单位时间内的数据进行挖掘,挖掘出支持度大于支持度误差的项集即候选频繁项集供第二部分处理;第二部分以 FP-Tree 为基础,引入倾斜时间窗口[13]建立 Pattern-Tree,用来记录不同时间粒度的频繁项集中间结果;算法对外设置了接口供用户输入参数,用户可以自由地设置最小支持度、置信度与查询时间,根据不同时间段参数方便地查询频繁项集及关联规则。

图 1 给出了 FP-Stream 算法在一个单位时间内的处理流程。

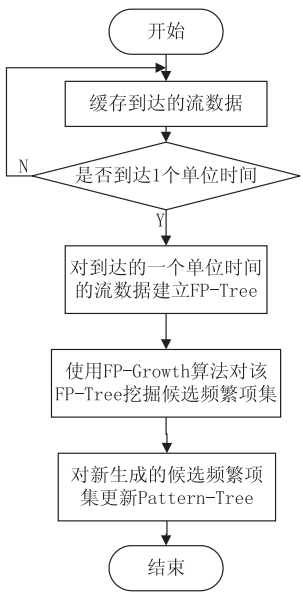

图 1 FP-Stream 算法单位时间内的处理流程

2 DPFP-Stream 算法设计

2.1 基本思想

FP-Stream 算法无法直接运用于分布式环境,因为当到达的数据流速过快,算法第一部分(用 FP-Growth 算法挖掘单位时间内的候选频繁项集)无法快速产生结果,为了提高挖掘速度必须提高最大支持度误差值,但这会影响挖掘精度。

针对这个问题,文中设计了 DPFP-Stream 算法。其基本思想是:将挖掘任务分为 local 和 global 两大部分,相应地,设置多个 local 节点和一个 global 节点,local 节点为局部计算节点,global 节点为全局合并节点。到达的流数据平均分配到不同的 local 节点,各 local 节点使用 FP-Growth 算法产生该单位时间内本节点的候选频繁项集,按照单位时间将候选频繁项集及其支持度计数打包发送至 global 节点;global 节点合并各 local 节点的中间结果并发送至 Pattern-Tree。此外,设置参数"当前时间"来保证被合并数据在时间上的对应性。

2.2 候选频繁项集的分布式并行化挖掘

候选频繁项集的挖掘以分布式并行化方式进行,到达的数据平均分配到各个 local 节点,每个节点设置一个缓存,接收一个单位时间的流数据,当接收时间到达一个单位时间,对这一块数据建立 FP-Tree,根据算法设定的最大支持度误差阈值,找到该单位时间内的候选频繁项集(支持度大于最大支持度误差的项集),并将其按照时间打包发送至 global 模块进行处理。图 2 为基于单位时间内的数据生成 FP-Tree 的流程。

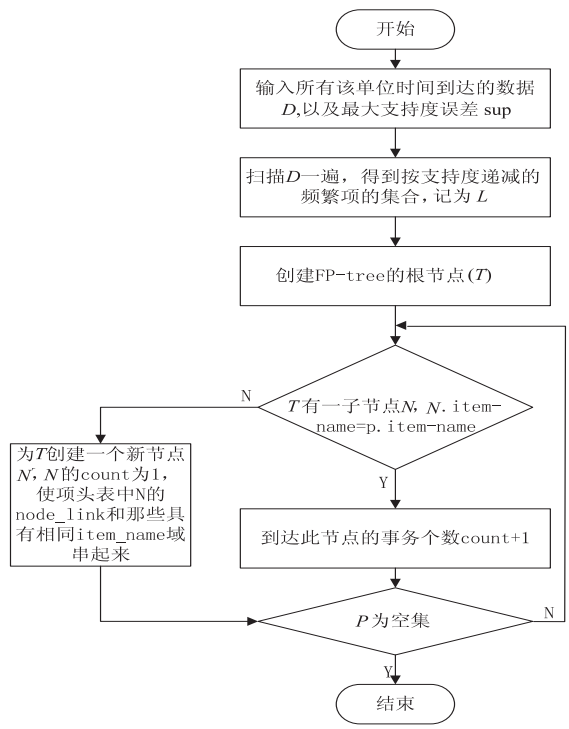

图 2 建立单位时间内的 FP-Tree 的流程

建立了单位时间内的 FP-Tree 之后,使用经典的 FP-Growth 算法[14]挖掘该棵 FP-Tree 中支持度大于最大支持度误差的频繁项集,即候选频繁项集,具体过程可以描述如下:

输入:待挖掘的 FP-Tree;

输出:所有的频繁项集。

步骤:

递归地挖掘每个条件 FP-Tree,累加后缀频繁项集,直到找到 FP-Tree 为空或者 FP-Tree 只有一条路径,首先调用 FP-Growth(Tree,null)。

过程 FP-Growth (Tree, x)可以描述如下:

procedure FP-Growth (Tree, x)

if Tree 含单个路径 P

then{

for 路径 P 中节点的每个组合(记作 b)

产生模式 $b \cup a$,其支持度 support 为 b 中节点的最小支持度;

}

else {

for each a_i 在 Tree 的头部(按照支持度计数由低到高顺序进行扫描)

{

产生一个模式 $b = a_i \cup a$,其支持度 support = a_i . support;构造 b 的条件模式基(即顺着 header table 中 item 的链表,找出所有包含该 item 的前缀路径,这些前缀路径就是条件模式基),然后构造 b 的条件 FP-Tree,即 Tree$b$;

if Tree$b$ 不为空

then 调用 FP_Growth(Tree$b$, b);

}

}

local 节点在生成一个单位时间内的候选频繁项集之后,将该单位时间内的所有频繁项集与记录总数、当前时间一起打包发送至 global 节点进行合并。

2.3　分布式并行化挖掘结果的合并

由于处理能力会有所不同,各 local 节点处理生成中间结果并发送至 global 节点的速度可能不一致,这使得 global 节点会错误地将不同时间段的中间结果合并至相同时间段。为了防止此类情况的发生,DPFP-Stream 算法对 global 节点与各 local 节点设置了参数“当前时间”,global 节点依据各 local 节点发送的“当前时间”对中间结果进行合并。

global 节点设置了阈值 threshold,其作用是控制 Pattern-Tree 的合并。global 节点合并一个 local 节点的中间结果至 Pattern-Tree 的过程可以描述如下:

输入:全局 Pattern-Tree,单个 local 节点的中间结果 MR,合并阈值 threshold;

输出:合并后的全局 Pattern-Tree。

步骤:

比较 MR 与 Pattern-Tree 的当前时间;

if MR. time = = PatterTree. time

then{

for each frequent item in MR

{

在 Pattern-Tree 中找到相应节点,将支持度计数加入该节点的第一块时间窗口(若 Pattern-Tree 中无相应节点,则新建节点插入相应信息);

}

将该单位时间内记录总数加入 Root 节点的第一块时间窗口;

}

else if MR. time = = PatternTree. time+1

then{

for each frequent item in MR

{

在 Pattern-Tree 中找到相应节点,将节点内窗口的数据向后滑动,并将支持度计数加入该节点的第一块时间窗口(若 Pattern-Tree 中无相应节点,则新建节点插入相应信息),Root 节点内窗口的数据向后滑动,并将记录总数加入 Root 节点的第一块时间窗口;

}

Pattern-Tree. time+1;

}

else if MR. time<PatternTree. time and PatternTree. time-MR. time<threshold

then{

依据 PatternTree. time-MR. time,找到该事件对应到倾斜时间窗口的具体位置,对 MR 中的所有频繁项集,在 Pattern-Tree 中进行更新;

}

else{

将 Pattern-Tree 当前时间发送至 local 节点,更新 local 节点当前时间,使之与 global 节点一致;

}

3　DPFP-Stream 算法在 Storm 平台上的实现

3.1　Storm 系统

Storm[15]是 Twitter 支持开发的一款分布式的、开源的、实时的、主从式大数据流式计算系统,是一种典型的流式数据计算架构,数据在任务拓扑中被计算,并输出有价值的信息。

任务拓扑是 Storm 的逻辑单元,一个实时应用的

计算任务将被打包为任务拓扑后发布,任务拓扑一旦提交后就会一直运行,除非显式地去中止。一个任务拓扑是由一系列 Spout 和 Bolt 构成的有向无环图,通过数据流实现 Spout 和 Bolt 之间的关联。如图 3 所示,Spout 负责从外部数据源不间断地读取数据,并以元组形式发送给相应的 Bolt,Bolt 负责对接收到的数据流进行计算,可以级联,也可以向外发送数据流。

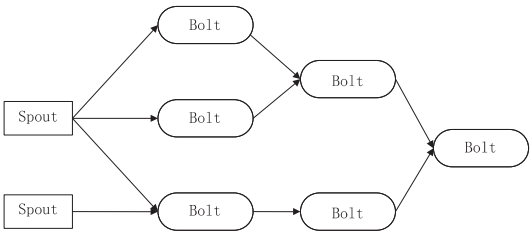

图 3 Storm 拓扑示例

3.2 DPFP-Stream 算法在 Storm 上的部署

基于 Storm 流计算框架的编程模型,文中设计了 DPFP-Stream 算法在 Storm 上的部署方案。

图 4 为 DPFP-Stream 算法在 Storm 上的拓扑示意图。

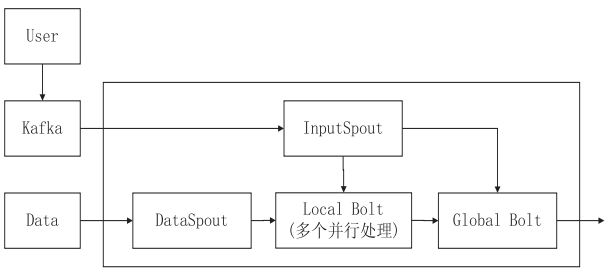

图 4 DPFP-Stream 的 Storm 拓扑图

如图 4 所示,Kakfa[16] 作为消息中间件,接收用户发送的配置参数与查询参数,发送至 InputSpout 供后续计算;DataSpout 接收待挖掘数据,将到达的数据打上时间戳标记并平均发送至各 Local Bolt 线程进行计算;InputSpout 接收用户输入的配置参数与查询参数,将所有参数(支持度、支持度误差、置信度、查询时间)发送至 Global Bolt 供挖掘计算与查询,同时将参数最大支持度误差发送至 Local Bolt 供生成候选频繁项集;Local Bolt 为算法在 Storm 上实现的并行化部分,按时间戳对单位时间内到达的数据使用 FP-Growth 算法挖掘候选频繁项集;Global Bolt 为算法在 Storm 上实现的合并部分,对各 Local Bolt 生成的中间结果进行合并,生成最新的 Pattern-Tree;用户可向系统输入查询参数查询最新 Pattern-Tree。

4 实验与结果分析

为了测试 DPFP-Stream 算法的分布式并行化效果,设计了如下实验。

(1)实验数据集与环境。

实验数据集是预处理过的超市购物数据集,实验中分别使用经典的 FP-Stream 算法与文中设计的 DPFP-Stream 算法对该数据集进行频繁模式挖掘。

实验环境:1 个 Nimbus 节点、2 个 Supervisor 的 Storm 集群,每台机器内存 8 GB,处理器为主频 2.70 GHz 的 i7 处理器,操作系统为 CentOS 6.4。使用 Kafka 作为消息中间件,设置一个 producer 每秒选取数据集中有特定关联规则的数据,打上相应时间戳,按照每秒 10 000 条的速率发送至 Kafka,算法的 Storm 拓扑从 Kafka 中获取数据进行相应挖掘计算。支持度设为 0.5,置信度设为 0.8,经典 FP-Stream 算法的支持度误差设为 0.3,DPFP-Stream 算法的支持度误差设为 0.1,线程数设为 3。

(2)实验结果与分析。

在 FP-Stream 算法与 DPFP-Stream 算法的对比方面:各算法分别处理完 100 万条数据后,两种算法的挖掘结果一致,而 Storm 系统的 StormUI 中显示 FP-Stream 相应线程的 capacity(线程处理压力)为 0.143,DPFP-Stream 相应线程的平均 capacity 为 0.113。这说明,尽管 DPFP-Stream 的参数支持度误差设置的比较小,但是在得出一致的挖掘结果的同时,单个线程所承受的计算压力反而减小了。

在 DPFP-Stream 算法的线程处理压力随算法参数设置和线程数的变化方面:实验结果如图 5 所示,在每秒到达拓扑的数据流速率为 10 000 条不变的情况下,无论支持度误差为 0.1 或是 0.3,DPFP-Stream 算法的 capacity 都随着 Local Bolt 线程个数的增加呈倒数减小,说明算法的处理能力可随线程数的增加呈近线性增加;而当支持度误差为 0.1 时,虽然线程处理压力比支持度误差为 0.3 时大,但是可以挖掘出更多的频繁项集。

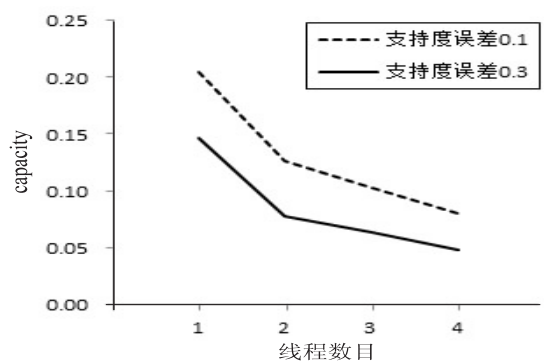

图 5 DPFP-Stream 算法多线程测试结果

从图中可以看出,DPFP-Stream 算法能有效地降低计算处理压力,并且不影响挖掘结果。此外,在线程处理压力不变的情况下,由于 DPFP-Stream 算法可以设置更低的支持度误差,故能在一定的情况下挖掘出 FP-Stream 算法挖掘不到的结果。

5　结束语

基于 FP-Stream 算法和分布式并行计算思想,文中设计了一种分布式并行化数据流频繁模式挖掘算法(DPFP-Stream),并在流计算平台 Storm 上进行了算法实现与性能测试。结果表明,该算法借助分布式并行化机制,能以更小的线程处理压力获得同样的挖掘精度,也说明了文中对 FP-Stream 算法所做的基于 Storm 的分布式并行化工作的可行性和有效性。

参考文献:

[1] Li Lingjuan, Li Xiong. An improved online stream data clustering algorithm[C]//Proceedings of second international conference on business computing and global informatization. Shanghai, China: [s. n.], 2012: 526-529.

[2] Gaber M, Zaslavsky A, Krishnaswamy S. Mining data streams: a review[J]. SIGMOD Record, 2005, 34(2): 18-26.

[3] Han J, Kamber M, Pei J. Data mining: concepts and techniques [M]. [s. l.]: Elsevier, 2006: 242-248.

[4] 孙大为,张广艳,郑纬民. 大数据流式计算:关键技术及系统实例[J]. 软件学报,2014,25(4):839-862.

[5] 孙玉芬,卢炎生. 流数据挖掘综述[J]. 计算机科学,2007,34(1):1-5.

[6] Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams[C]//Proceedings of automata, languages and programming. Berlin: Springer, 2002: 693-703.

[7] 李国徽,陈　辉. 挖掘数据流任意滑动时间窗口内频繁模式[J]. 软件学报,2008,19(10):2585-2596.

[8] Ma Ke, Li Lingjuan, Ji Yimu, et al. Research on parallelized stream data micro clustering algorithm[C]//Proceedings of ICCAET 2015. Zhengzhou, China: [s. n.], 2015: 629-634.

[9] Giannella C, Han J, Pei J, et al. Mining frequent patterns in data streams at multiple time granularities[J]. Next Generation Data Mining, 2003, 212: 191-212.

[10] 唐耀红. 数据流环境中关联规则挖掘技术的研究[D]. 北京:北京交通大学,2012.

[11] 刘学军,徐宏炳,董逸生,等. 挖掘数据流中的频繁模式[J]. 计算机研究与发展,2015,42(12):2192-2198.

[12] 程转流,王本年. 数据流中的频繁模式挖掘[J]. 计算机技术与发展,2007,17(12):53-55.

[13] Jin R, Agrawal G. An algorithm for in-core frequent itemset mining on streaming data[C]//Proceedings of fifth IEEE international conference on data mining. [s. l.]: IEEE, 2005: 210-217.

[14] Han J, Pei J, Yin Y, et al. Mining frequent patterns without candidate generation: a frequent-pattern tree approach[J]. Data Mining and Knowledge Discovery, 2004, 8(1): 53-87.

[15] Marz N. Storm: distributed and fault-tolerant realtime computation[EB/OL]. 2012. http://storm. apache. org.

[16] Apache. Apache Kafka: a high-throughput, distributed, publish-subscribe messaging system[EB/OL]. 2015. http://kafka. Apache. org.

(上接第 74 页)

(2): 56-60.

[2] 李秋洁,茅耀斌,王执铨. CAPTCHA 技术研究综述[J]. 计算机研究与发展,2012,49(3):469-480.

[3] Strathy N W, Suen C Y, Krzyzak A. Segmentation of handwritten digits using contour features[C]//Proceedings of the second international conference on document analysis and recognition. [s. l.]: IEEE, 1993: 577-580.

[4] Shi Z, Shrihari S N, Shin Y C, et al. A system for segmentation and recognition of totally unconstrained handwritten numeral strings[C]//Proc of international conference on document analysis and recognition. [s. l.]: [s. n.], 1997: 455-458.

[5] Lu Z, Chi Z, Siu W C, et al. A background-thinning-based approach for separating and recognizing connected handwritten digit strings[J]. Pattern Recognition, 1999, 32(6): 921-933.

[6] Congedo G, Dimauro G, Impedovo S, et al. Segmentation of numeric strings[C]//Proc of third international conference on document analysis and recognition. Montreal, Que: IEEE, 1995: 1038-1041.

[7] 曲金山. 基于形状上下文的验证码识别研究[D]. 哈尔滨:哈尔滨工程大学,2013.

[8] Chaudhari S K, Deshpande A R, Bendale S B, et al. 3D drag-n-drop CAPTCHA enhanced security through CAPTCHA [C]//Proceedings of the international conference & workshop on emerging trends in technology. [s. l.]: ACM, 2011: 598-601.

[9] Imsamai M, Phimoltares S. 3D CAPTCHA: a next generation of the CAPTCHA[C]//Proc of international conference on information science and applications. [s. l.]: IEEE, 2010: 1-8.

[10] Macias C R, Izquierdo E. Visual word-based CAPTCHA using 3D characters[C]//Proc of 3rd international conference on crime detection and prevention. [s. l.]: [s. n.], 2009: 1-5.

[11] Nguyen V D, Chow Y W, Susilo W. On the security of text-based 3D CAPTCHAs[J]. Computers & Security, 2014, 45: 84-99.

[12] Mitra N J, Chu H K, Lee T Y, et al. Emerging images[J]. ACM Transactions on Graphics, 2009, 28(5): 1-8.

[13] Ross S A, Halderman J A, Finkelstein A. Sketcha: a captcha based on line drawings of 3D models[C]//Proceedings of international conference on world wide web. [s. l.]: ACM, 2010: 821-830.

[14] YUNiTi. YUNiTi-do something good[EB/OL]. 2013-06-29. http://www. yuniti. com/register. php.

[15] Ye Q, Chen Y, Zhu B. The robustness of a new 3D CAPTCHA [C]//Proc of 11th IAPR international workshop on document analysis systems. [s. l.]: IEEE, 2014: 319-323.

[16] Charlotte C. Cafe charlotte[EB/OL]. 2013-06-29. http://www. cafe-charlotte. cz/en/fanclub.