软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(4):1078−1093 [doi: 10.13328/j.cnki.jos.005419]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

分布式多数据流频繁伴随模式挖掘*

于自强[1], 禹晓辉[2], 董吉文[1], 王 琳[1]

[1](济南大学 信息科学与工程学院,山东 济南 250022)

[2](山东大学 计算机科学与技术学院,山东 济南 250101)

通讯作者: 王琳, E-mail: ise_wanglin@ujn.edu.cn

摘 要: 多数据流频繁伴随模式是指一组对象较短时间内在同一个数据流上伴随出现,并在之后一段时间以同样方式出现在其他多个数据流上.现实生活中,城市交通监控系统中的伴随车辆发现、基于签到数据的伴随人群发现、基于社交网络数据中的高频伴随词组发现热点事件等应用都可以归结为多数据流频繁伴随模式发现问题.由于数据流规模巨大且到达速度快,基于单机的集中式挖掘算法受到硬件资源的限制难以及时发现海量数据流中出现的频繁伴随模式.为此,提出面向大规模数据流频繁伴随模式发现的分布式挖掘算法.该算法首先将每个数据流划分成若干个 segment 片段,然后构建适合部署在分布式计算平台上的多层挖掘模型,并利用多计算节点以并行方式对大规模数据流进行处理,从而实时发现频繁伴随模式.最后,在真实数据集上进行充分实验以验证算法性能.

关键词: 多数据流;频繁伴随模式;分布式挖掘算法

中图法分类号: TP311

中文引用格式: 于自强,禹晓辉,董吉文,王琳.分布式多数据流频繁伴随模式挖掘.软件学报,2019,30(4):1078−1093. http://www.jos.org.cn/1000-9825/5419.htm

英文引用格式: Yu ZQ, Yu XH, Dong JW, Wang L. Distributed mining of frequent co-occurrence patterns across multiple data streams. Ruan Jian Xue Bao/Journal of Software, 2019,30(4):1078−1093 (in Chinese). http://www.jos.org.cn/1000-9825/5419.htm

Distributed Mining of Frequent Co-occurrence Patterns across Multiple Data Streams

YU Zi-Qiang[1], YU Xiao-Hui[2], DONG Ji-Wen[1], WANG Lin[1]

[1](School of Information Science and Engineering, University of Ji'nan, Ji'nan 250022, China)

[2](School of Computer Science and Technology, Shandong University, Ji'nan 250101, China)

Abstract: A frequent co-occurrence pattern across multiple data streams refers to a set of objects occurring in one data stream within a short time span and this set of objects appear in multiple data streams in the same fashion within another user-specified time span. Some real applications, such as discovering groups of cars that travel together using the city surveillance system, finding the people that are hanging out together based on their check-in data, and mining the hot topics by discovering groups of frequent co-occurrence keywords from social network data, can be abstracted as this problem. Due to data streams always own tremendous volumes and high arrival rates, the existing algorithms being designed for a centralized setting cannot handle mining frequent co-occurrence patterns from the large scale of streaming data with the limited computing resources. To address this problem, FCP-DM, a distributed algorithm to mine frequent co-occurrence patterns from a large number of data streams, is proposed. This algorithm first divides the data streams into segments, and then constructs a multilevel mining model in the distributed environment. This model utilizes multiple computing nodes for detecting

* 基金项目: 国家自然科学基金(61702217, 61771230, 61772231, 61873324); 山东省重点研发计划(2017GGX10144, 2018GGX101048, 2017CXGC0701, 2016ZDJS01A12); 山东省自然科学基金(ZR2017MF025); 济南大学科技发展计划(XKY1737, XKY1734)

Foundation item: National Natural Science Foundation of China (61702217, 61771230, 61772231, 61873324); Key Research and Development Program of Shandong Province (2017GGX10144, 2018GGX101048, 2017CXGC0701, 2016ZDJS01A12); Natural Science Foundation of Shandong Province of China (ZR2017MF025); Scientific and Technologic Development Program (XKY1737, XKY1734)

收稿时间: 2017-04-11; 修改时间: 2017-06-09, 2017-08-25; 采用时间: 2017-09-25

massive volumes of data streams in a parallel pattern to discover frequent co-occurrence patterns in real-time. Finally, extensive experiments are conducted to fully evaluate the performance of the proposal.

Key words: multiple data stream; frequent co-occurrence pattern; distributed mining algorithm

1 引 言

随着移动终端和无线网络的广泛应用,现实中许多应用面临大规模数据流的处理.例如,社交网络中的微博数据,电子商务领域的顾客浏览、交易数据,城市交通管控系统的过车数据,环境监测领域的传感器数据等都包含了大规模数据流.在海量数据流中,通常隐藏着大量用户感兴趣的特定模式,发现这些特定模式对于改善应用服务质量,提升数据应用价值具有重要意义.多数据流中的频繁伴随模式是由 Yu 等人[1]首次提出来的,现实中若干应用都可以抽象为频繁伴随模式的发现问题.本文的研究目标是设计分布式挖掘算法,实时发现海量数据流中的频繁伴随模式.为便于描述,下文将采用 FCP 来表达频繁伴随模式.

给定一个无界数据流的集合 $S=\{s_1,s_2,\ldots,s_n\}$和一个元素集合 $O=\{o_1,o_2,\ldots,o_k\}$,如果集合 O 是一个 FCP,那么它将满足以下条件:(1) 该集合中的元素在小于τ的时间段内出现在至少θ个数据流上;(2) 该集合的元素在每一个数据流上出现的时间间隔不大于ξ.这里,τ、θ、ξ都是用户指定的参数[1].频繁伴随模式表达了一组对象在多个数据流中的紧密关系,许多应用都可以归结为数据流上的频繁伴随模式发现问题[1].

伴随车辆发现.当前城市主要道路和路口安装了监控抓拍设备,车辆每次经过时,抓拍设备都会生成一条结构化的过车记录(vehicle passing record,简称 VPR).这样,每个抓拍设备将产生一个由连续过车记录组成的数据流,而一些伴随行驶的车辆将会出现在由多个抓拍设备所产生的数据流中.因此,从过车记录中发现伴随车辆相当于从这些数据流中发现 FCP.现实中,及时发现伴随车辆对于预防和打击罪犯嫌疑人驾驶多部车辆协同作案、尾随受害人作案等案件具有重要意义.

基于微博的热点话题发现.在新浪微博等社交平台,可以认为每个用户对应着一个由自己发布的微博组成的数据流.较短时间内,在许多用户微博中频繁出现的词汇组合通常预示着一个新的热点话题的出现.那么从海量用户对应的数据流中挖掘频繁伴随出现的热点词汇组合其本质也是多数据流的 FCP 发现问题.

位置信息服务.在一些签到应用中(如 Foursquare),用户会在到达一个地点之后签到,并向数据中心报告自己的位置.在这类应用中,可以认为每个签到的地点对应着一个数据流,而每个数据流是由用户的签到信息组成.在这些数据流中实时挖掘 FCP,能够发现短时间内共同到达某些签到地点的用户群,而发现这些用户具有潜在的商业价值,例如可以作为某些广告更好地推送对象.

以上事例所要解决的基本问题都是发现较短时间内出现在多个数据流上的 FCP(车辆、人、关键词等).与多数据流频繁伴随模式发现较为相似的问题是数据流中的频繁模式发现问题,但两者有明显区别.数据流中的频繁模式是指某一个元素集合在一个数据流的多个“事务(transaction)”中出现,且出现的频率大于给定的阈值;本文所研究的频繁伴随模式则是指一个元素集合在多个数据流中出现,并且所在数据流的数目和出现的间隔需要满足指定的条件.由于问题本身不同,因此,已有数据流频繁模式挖掘算法无法直接用来解决本文所研究的问题,原因将在下一节进行详细阐述.

目前,对于多数据流 FCP 发现问题的研究还很少.文献[1]提出该问题的同时,给出了 CooMine 挖掘算法.该算法的目的是通过减少内存消耗和索引维护代价快速发现多数据流中的 FCP.值得注意的是,CooMine 算法是针对单个计算节点设计的集中式挖掘算法,而单个计算节点受到硬件资源的限制,很难应对大规模数据流.为解决这一问题,最直接的办法是采用多个计算节点,令每个计算节点独立运行 CooMine 算法,处理一部分数据流.该策略虽然能够处理大规模数据流,但缺点是形成某些 FCP 的数据流可能被分在不同的计算节点,导致部分结果丢失.另一策略是将 CooMine 算法在多个计算节点上并行化实现,但是 CooMine 算法采用的 Seg-tree 树型索引结构很难有效地并行化部署到多个计算节点上.因此,当在分布式计算环境下解决大规模数据流 FCP 发现问题时无法直接采用 CooMine 算法,仍需设计高效的分布式挖掘算法.

与集中式算法相比,设计分布式挖掘算法面临以下挑战.

(1) 由于每个计算节点只存储部分数据,这给 FCP 的生成和比对带来很大困扰.例如,对于一个分布在计算节点 n_1 上的长度为 k 的 FCP,计算节点 n_1 无法得知该 FCP 应和哪些计算节点的长度为 k 的 FCP 合并,生成长度为(k+1)的 FCP.

(2) 采用多个计算节点处理海量数据流时,可能出现不同计算节点上数据负载不均的情况,从而影响整体的处理效率.这是因为,每个 FCP 的生成需要多个计算单元协同合作,负载失衡时,负载较轻的计算单元往往需要等待负载较重的计算单元的计算结果,从而影响了整个系统的挖掘效率.

(3) 由于大规模数据流连续到达,新到达的数据不断地和已有数据共同构成新的FCP,这就要求挖掘算法必须能够随着数据流的连续到达实时发现新生成的 FCP,实现 FCP 的增量挖掘.

为应对以上挑战,本文提出 FCP 分布式挖掘方法(FCP distributed mining approach,简称 FCP-DM).该方法的核心思想是采用基于服务器集群的分布式计算模式,从大规模数据流中实时连续发现 FCP.FCP-DM 首先将每个连续到达的数据流划分成若干 segment 片段,将问题转化为从不同数据流的 segment 中发现 FCP;然后基于 Actor-Model 计算模型构建多级分布式挖掘框架,由多计算节点实现对不同数据流的 segment 片段的逐级并行处理;最后根据 Apriori 算法的迭代思想,设计不同层级的数据分发策略,通过对 segment 的多层计算和比对,最终得到 FCP.

本文的主要贡献如下:

- 提出 FCP-DM 分布式挖掘方法并给出相应的分布式多级计算框架,能够实现对大规模数据流的并行处理,通过多层迭代计算和比对发现所有的 FCP.FCP-DM 方法具备良好的可扩展性,仅通过增加硬件资源就可实现处理能力的线性增长.
- 解决了分布式环境下多数据流 FCP 挖掘所面临的负载迁移、多计算节点之间 FCP 分发策略以及面向连续数据流的 FCP 增量挖掘等问题.
- 在开源流数据处理平台 S4 上实现了 FCP-DM 方法,并且以山东省会城市济南某天所有卡口的过车记录为测试数据集进行大量实验,充分验证了该算法的各项性能.

2 相关工作

多数据流频繁伴随模式发现问题是由 Yu 等人首次提出来的[1],并给出两种集中式挖掘算法:DIMine 算法和 CooMine 算法.这两种算法通过设计不同的索引结构来提高挖掘效率并达到节省存储空间的目的.然而,这两种算法都是基于单机计算环境而设计,难以直接应用到分布式环境中,可扩展性较差,无法应对当前规模急剧增长的海量数据流.

已有工作中,与多数据流 FCP 发现问题最为相似的是数据流上的频繁模式挖掘问题.数据流上的频繁模式(frequent pattern,简称 FP)[2]是指给定一个由连续的 transaction 组成的数据流(每个 transaction 包含多个元素),如果指定的某段时间内该数据流共有 n 个 transaction,而一个元素集合在其中的 m($m\leqslant n$)个 transaction 中出现并且 $\frac{m}{n}\geqslant sup$,那么该元素集合就是一个频繁模式,sup 是用户指定的阈值.下文采用 FP 表示频繁模式.

近年来,流数据 FP 挖掘问题受到国内外学者的广泛关注[2−12].Manku 和 Motwani 提出了粘性抽样和有损耗的计数算法来计算数据流元素集合的近似频率[4].Yu 等人提出了 FDPM 算法[5],该算法能够使用有界的内存挖掘数据流中的频繁元素集合.该算法是以假的消极结果为导向的,即某些特定的频繁元素集合可能不会被发现.这两项工作都是基于 landmark 模型,它们的目标都是挖掘数据流从开始到当前时间所有的频繁模式.此外,它们并不保证发现所有频繁模式,而是保证获得一个错误率小于指定参数的近似结果集.

另外一些方法[2,5−12]是能够从数据流中获得当前精确的频繁模式集合.Chang 等人[2]提出了一种挖掘算法,该算法通过自适应地减小过期事务的影响从而从在线数据流中发现频繁模式.Leung 和 Khan 等人[6]提出树型索引结构(DSTtree),该索引能够从数据流中捕获重要信息从而精确地挖掘频繁模式.Mozafari 等人则将数据流划分成滑动窗口,提出 versification 这一新的计数概念,并基于此概念设计了 SWIM 算法.该算法是一种精确算

法,其性能和扩展性能够根据窗口的大小进行调整[8].KARP 等人提出了一个简单、精确的数据流上的频繁项集发现算法,并证明了该算法的时间复杂度为线性且空间复杂度为 $1/\theta$,θ为用户指定的支持度[9].Chi 等人[10]主要研究内存受限情况下的数据流上闭合频繁项集挖掘问题,并给出 Moment(maintaining closed frequent itemsets by incremental updates)算法对数据流的闭合频繁项集进行持续监测.Silva 等人提出了 Star FP Stream 算法,用以解决从多维数据模型所产生的海量星型数据模式中发现频繁模式[11].李海峰等人研究了数据流上频繁项集挖掘时所采用的滑动窗口模型,提出了基于事务的可变窗口滑动模型,并在此基础上提出了频繁项集的挖掘算法 FIMoTS[12].

以上这些方法虽然能够从数据流中挖掘频繁模式,但是无法被直接用来解决FCP挖掘问题.首先,FP和FCP的定义有着很大的不同.FP挖掘问题是关注一个元素集合在某个数据流上出现的频率,而FCP挖掘问题则关注一个元素集合在数据流内部和多个数据流之间的伴随关系.此外,上述方法主要针对单个数据流的 FP 挖掘问题,而本文要解决的是多数据流 FCP 发现问题,需要对多个数据流同时处理,两者之间存在明显区别.

值得注意的是,文献[13]提出了 H-stream 算法,目标是发现多个数据流上的频繁模式.表面上看,H-stream 算法要解决的问题似乎与本文的问题非常相似,但事实上有很大不同.虽然 H-stream 算法的目标是挖掘在多个数据流出现的 FP,但本质上还是先查找每个数据流的 FP,然后筛选符合条件的 FP.而本文中,FCP 是由多个数据流共同生成,因此必须同时处理多个数据流才能发现 FCP.此外,H-stream 是一种近似算法,而本文的目标是求解精确结果.因此,H-stream 算法并不能解决本文研究的问题.此外,毛宇星等人虽然也提出一种多层关联规则挖掘方法[14],但是分层的目的是对每层的数据进行聚类,而本文所设计的多层挖掘模型的目的是充分发挥多计算节点在处理多数据流时的并行计算能力,研究的侧重点有很大不同.

频繁情节挖掘(frequent episode mining)是在频繁模式挖掘基础上衍生出来的又一问题,它是指从一些序列数据中发现有价值的或者用户感兴趣的模式序列[15,16].目前,已有若干工作研究多个行业(如通信、制造业、金融、生物信息)的频繁情节挖掘问题,但是绝大部分工作都是采用离线方式挖掘频繁情节,这些方法也难以用于实时发现多数据流的频繁伴随模式.虽然 Ao 等人提出了基于 episode trie 的在线频繁情节挖掘方法[15],然而该方法所关注的频繁情节也是基于单个数据流定义的,与本文研究的问题有很大区别.此外,该方法是针对单机计算环境所设计,episode trie 索引结构很难直接部署到分布式计算环境,也难以解决本文所研究的问题.

当前,也有一些学者开始研究大数据环境下的模式发现[17−20],但是这类工作通常关注的是静态数据集上的模式发现问题,而本文关注的是从海量动态数据流上发现特定模式.

3 问题定义

为便于算法描述,本文引入并使用文献[1]所给出的相关定义.

定义 1(数据流(data stream)). 数据流是一组连续的按时间顺序到达的元素序列.对于数据流中的任意元素 o_i(i 为该元素在数据流中的序列号),它的标识符和时间戳分别是 id_i 和 t_i,其中,t_i 表示该元素出现的时间.

定义 2(伴随模式(co-occurrence pattern,简称 CP)). 给定数据流 s_i 上的一个元素集合 $O=\{o_1,o_2,\ldots,o_k\}$,如果 $t_{\max}^{o^{s_i}}-t_{\min}^{o^{s_i}}\leqslant\xi$,则认为集合 O 是一个伴随模式 CP,这里,$t_{\max}^{o^{s_i}}=\max\{,\ldots,\}$,$t_{\min}^{o^{s_i}}=\min\{,\ldots,\}$,$\xi$是用户指定的阈值.$CP_k$($k\geqslant 2$)表示长度为 k 的 CP,即该 CP 包含 k 个元素.

定义 3(频繁伴随模式(frequent co-occurrence pattern,简称 FCP)). 如果一个伴随模式 CP 在 l 个数据流中出现,l 个数据流可以表示为集合 $S=\{s_1,s_2,\ldots,s_l\}$.如果该 CP 满足以下条件:

(1) $l\geqslant\theta$,

(2) $T_o^{\max}-T_o^{\min}\leqslant\tau$,

那么,该 CP 就是一个 FCP.其中,$T_o^{\max}=\max\{t_{\max}^{o^{s_1}},\ldots,t_{\max}^{o^{s_l}}\}$,$T_o^{\min}=\min\{t_{\min}^{o^{s_1}},\ldots,t_{\min}^{o^{s_l}}\}$,$\theta$和$\tau$是用户指定的参数,并且$\tau>>\xi$(见定义 2).$FCR_k$($k\geqslant 2$)表示长度为 k 的 FCP,即该 FCP 包含 k 个元素.

定义 4(segment 片段). 给定一个数据流 s_i,一个 segment 片段 $G(o_1,o_2,\ldots,o_m)$是数据流 s_i 的子序列,并且 G 满

足以下条件.

(1) $|t_i-t_j|\leqslant\xi$,这里,o_i,o_j 是 G 中任意两个元素,t_i,t_j 分别是 o_i,o_j 出现的时间;

(2) 数据流 s_i 中不存在子序列 G',使得 G'是一个 segment 片段并且 G 是 G'的一个严格的子序列.

图 1 给出由多个元素构成的一个数据流,假设每个元素出现的时间如下:t_a=2,t_c=5,t_d=13,t_g=16,t_e=20,t_b=26,并假设ξ=10,那么该图中{a,c,d}和{d,g,e}为两个 segment.元素 a 是 segment 片段 G_0 的第 1 个元素,因为 $t_d-t_a<\xi,t_g-t_a>\xi$,所以元素 d 是 G_0 的最后一个元素.当以元素 c 为第 1 个元素构建 segment 时,由于 $t_g-t_c>\xi$,{c,d,g}无法构成 segment.进一步地,我们以元素 d 为第 1 个元素构建 segment 时,便可得{c,d,e}是一个 segment.

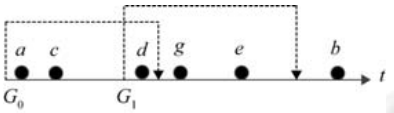

Fig.1 Segment partition

图 1 Segment 片段

4 FCP-DM 挖掘方法

引入 segment 之后,每个数据流被划分成多个 segment 片段,那么 FCP-DM 的目标就是从海量数据流的 segment 中发现 FCP.根据 CP 和 FCP 的定义,可以推断出一个数据流的某个 segment 中的任意 CP 都可能与其他数据流的 segment 中的 CP 形成 FCP.因此,为了得到精确结果,FCP-DM 算法需要对所有数据流的 segment 所包含的 CP 进行比对.

4.1 FCP-DM算法

FCP-DM 算法中,每一个数据流首先被划分成多个 segment.由于 segment 分布在不同的计算节点上,那么如何比对不同 segment 中的 CP 是一个首先要解决的问题.

在本文构建的分布式计算环境中,一个物理节点可以运行若干个逻辑计算单元(processing element,简称 PE).FCP-DM 算法的思想是以 CP 本身为 key 构建分布式哈希索引(DH-index),每个 CP 对应 DH-index 的一个索引单元.DH-index 中每个索引单元的 value 值包括对应 CP 的出现时间以及它所在的 segment 和数据流的相关信息.这种情况下,不同 segment 中相同的 CP 被插入到 DH-index 时,这些 CP 将被映射到同一个索引单元.当 FCP-DM 被部署到分布式计算环境时,令物理节点的一个 PE 负责一个索引单元,该索引单元每增加一条新的记录(即来自某个 segment 的 CP),PE 都会判断该索引单元对应的 CP 是否为 FCP.某个 CP 一旦满足 FCP 定义的条件,FCP-DM 会实时发现该 FCP.

在图 2 中,给出分别属于数据流(s_1,s_3,s_2)的 3 个 segment(G_0,G_1,G_2),根据给定的 segment,得到所有长度为 2 的 CP 集合.根据图 2 的 CP_2 集合建立 DH-index 索引,如图 3 所示.FCP-DM 算法对 DH-index 进行扫描,可以得到全部长度为 2 的 FCP(FCP_2).例如,设定θ=3,那么 FCP-DM 只需对{o_1,o_2}进行判断.如果{o_1,o_2}出现在 s_1,s_3,s_2 的时间间隔小于τ,那么{o_1,o_2}就是一个 FCP.FCP-DM 算法在获得 FCP_2 后,便利用 Apriori 启发式思想[3],逐步地由 FCP_k($k\geqslant2$)的集合推导出 FCP_{k+1}($k\geqslant2$)的集合.

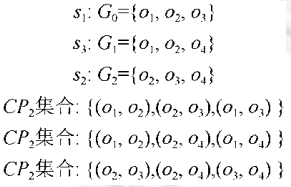

Fig.2 Segments and the corresponding CP_2 collections

图 2 生成 segment 的 CP_2 集合

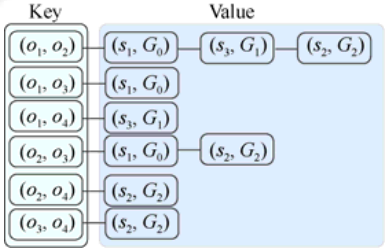

Fig.3 The structure of DH-index

图 3 DH-index 示意图

实际应用中,海量数据流将产生规模巨大的 CP,超出了单个计算节点的处理能力.为此,本文将 FCP-DM 算法部署到分布式计算环境,通过利用多计算节点的存储和计算资源,采用并行计算和比对方式,实现海量数据流中 FCP 的实时发现.下面介绍 FCP-DM 算法的分布式计算框架.

4.2 FCP-DM分布式计算框架

本文所提出的 FCP-DM 分布式计算框架(FCP-DM distributed framework,简称 FCP-DMDF)采用 Actor-Model 分布式计算模型,FCP-DMDF 中基本逻辑处理单元是 PE(processing element),每个物理节点可以运行任意多个 PE.PE 之间通过发送和接收事件(event)进行数据传输.一个 Event 被表示为⟨type,key,value⟩,type 表示 Event 的类型,key 和 value 分别表示 Event 的键值和内容.每个 PE 指定其所接收的 Event 的 type 和 key 值.FCP-DMDF 根据 PE 所指定的 type 和 key 值为其分发 Event.任意一个 PE 可以接受其他 PE 发送的 Event,也可以将本地计算结果以 Event 的形式发送给其他 PE.对于一个新的 Event,如果已有的 PE 无法对其进行处理,那么 FCP-DMDF 将自动构建一个新的 PE 处理该 Event.

FCP-DMDF 是一个多级框架,主要包括数据接收层、数据分割层和算法实现层.图 4 是 FCP-DMDF 框架示意图.

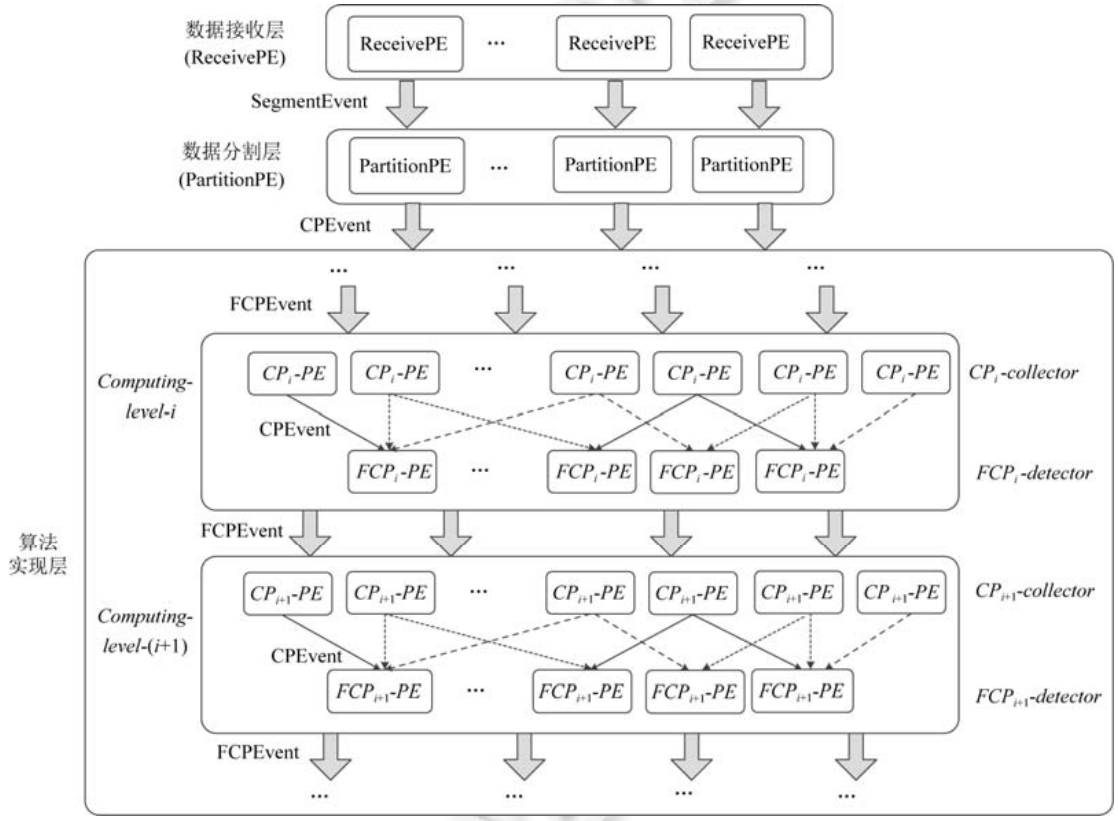

Fig.4 The framework of FCP-DMDF

图 4 FCP-DMDF 示意图

数据接收层的任务是接收持续到达的数据流.每个数据流由一个 ReceivePE 负责,ReceivePE 的任务是接收数据流并随时间延续将每个数据流分割成若干 segment,然后将 segment 以 SegmentEvent 的形式发送至数据分割层的 PartitionPE.

数据分割层的任务是生成每个 segment 所有长度为 2 的 CP.该层中每个 PartitionPE 的任务是接收来自不

同数据流的 segment,并计算每个 segment 包含的所有长度为 2 的 CP.然后以 CP 本身为 key 值将每一个 CP 以 CPEvent 的形式分发至算法实现层的 PE.在该层中,每个 PartitionPE 负责一个或多个 segment,并能够根据负载对 segment 进行数据迁移,第 4.3.1 节将详细讨论该问题.

算法实现层是 FCP-DMDF 的主体部分.该层包含多个 *Computing-Level*,*Computing-Level-i* 表示第 i 层 *Computing-Level*,负责发现长度为(i+1)的 FCP.FCP-DM 算法在产生长度为 i 的 FCP 时包含两步操作:CP_i 的聚合和 FCP_i 的检测,因此,每层 *Computing-Level-i* 包含两类 PE:CP_i-*PE* 和 FCP_i-*PE*.当 $i\geqslant 3$ 时,由 FCP_{i-1} 集合得到 FCP_i 集合的迭代计算步骤如下.

① CP_i-*PE* 将可能构成 CP_i 的一组 FCP_{i-1} 进行聚合,生成待检测的 CP_i,并将符合条件的 CP_i 以 CPEvent 的形式发送至 FCP_i-*PE*.分布式环境下,如何将可能构成 CP_i 的一组 FCP_{i-1} 由不同的 PE 汇聚至同一个 CP_i-*PE* 是一个不小的挑战,为此,本文设计了基于⟨key,value⟩的 FCP 分发策略(该策略将在第 4.3.2 节详细加以讨论);

② FCP_i-*PE* 接收来自不同 CP_i-*PE* 的 CP_i 进行检测并判断其是否为 FCP_i,然后将发现的 FCP_i 以 FCPEvent 的形式发送至 *Computing-Level*-(i+1)层的 CP_{i+1}-*PE*.

③ 重复步骤①和②,CP_{i+1}-*PE* 将接收的 FCP_i 合并生成 CP_{i+1},然后由 FCP_{i+1}-*PE* 检测 CP_{i+1} 是否为 FCP_{i+1}.

根据上述步骤,随着 i 的增长,算法实现层能够通过逐步迭代的方式发现所有的 FCP.

需要说明的是,当 i=2 时,由于 CP_2 由 PartitionPE 直接生成,因此,每个 CP_2-*PE* 接收到的是一组来自不同数据流的 CP_2,CP_2-*PE* 将该组 CP_2 进行聚合后发送至 FCP_2-*PE*.由于现实应用中 FCP 的最大长度无法预先确定,因此 FCP-DM 计算框架将根据 FCP 的长度自适应地增加或减少 *Computing-Level* 的层数,从而发现给定数据流中所有 FCP.

4.3 构建FCP-DMDF面临的挑战与解决方案

本节主要研究分布式环境下多数据流 FCP 挖掘所面临的负载迁移、FCP 分发策略以及面向连续数据流的 FCP 增量挖掘等挑战.

4.3.1 数据分割层的负载均衡策略

数据分割层将生成每一个 segment 片段所有的长度为 2 的 CP,并将 CP 进行分发.实际应用中,数据分割层的每个计算单元可能负责若干个 segment,如果数据分布不均匀,将造成不同计算单元之间的负载失衡,降低整个计算框架的挖掘效率.

为解决这一问题,本文设计动态迁移策略来保证计算单元之间的负载均衡.为便于描述,令 PE^r 表示数据接收层的任意一个 ReceivePE,令 PE_i^p 和 PE_{i+1}^p 表示数据分割层任意两个不同的 PartitionPE.此外,令$[w_h,w_k]$表示元素序列号的范围,如果某个 segment 片段首个元素 o_i 的 $id_i\in[w_h,w_k]$,那么该 segment 位于范围$[w_h,w_k]$.

对于数据分割层的任意逻辑计算单元 PE_i^p,假设其负责$[w_h,w_k]$范围内的 segment,那么令$[w_h,w_k]$为 PE_i^p 所能够处理 SegmentEvent 的 key 值范围.数据接收层的逻辑计算单元 PE^r 将根据 PE_i^p 的 key 值范围向其发送相应的 SegmentEvent.如果 PE_i^p 负载过大,即它所负责的 segment 数量大于系统设定的阈值 β,我们将对它的负载进行迁移,整个迁移过程包含以下 3 步.

(1) 对 PE_i^p 的 key 值范围$[w_h,w_k]$进行拆分,生成$[w_h,w_j]$和$[w_j,w_k]$($w_h<w_j<w_k$),使得$[w_h,w_j]$和$[w_j,w_k]$范围内的 segment 数量大致相等.

(2) PE_i^p 将$[w_j,w_k]$范围内未处理的 segment 以 SegmentEvent 的形式发送至数据分割层的新的逻辑计算单元 PE_{i+1}^p,PE_{i+1}^p 将由 FCP-DMDF 在 PE_i^p 发出 SegmentEvent 后创建. PE_{i+1}^p 可接收 SegmentEvent 的 key 值范围为$[w_j,w_k]$.

(3) PE_{i+1}^p 创建后,数据接收层的逻辑计算单元 PE^r 所生成的$[w_j,w_k]$范围内的 segment 将不再发送至 PE_i^p,而是发送至 PE_{i+1}^p. 这样,整个过程将 PE_i^p 约 1/2 的负载迁移至 PE_{i+1}^p.

图 5 是负载迁移执行过程的示意图.其中,图 5(a)表示初始状态下数据输入层的 PE^r 根据 PE_i^p 的 key 值范围向其发送相应的 segment.图 5(b)的①②③步则表示上述负载迁移 3 个步骤的执行过程.其中,

① 表示 PE_i^p 根据自身负载的 segment 数量将 key 值范围划分为$[w_h,w_j]$和$[w_j,w_k]$;

② 表示 PE_i^p 将位于$[w_j,w_k]$的 segment 发送至 PE_{i+1}^p, FCP-DMDF 发现 PE_{i+1}^p 不存在,将自动创建 PE_{i+1}^p;

③ PE^r 将后续到达的位于$[w_j,w_k]$的 segment 发送至 PE_{i+1}^p.

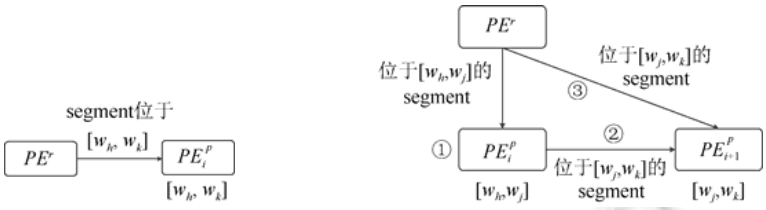

Fig.5 The procedure of workload transfer

图 5 负载迁移示意图

4.3.2 分布式环境下 FCP 分发策略

FCP-DM 算法逐步地由 FCP_k 的集合推导出 FCP_{k+1} 的集合,这在单机环境下不难实现.但在分布式环境下却面临新的问题.

(1) 由于数据分布在多个计算节点上,单个计算节点无法确定哪些 FCP_k 能够合并生成 FCP_{k+1}.

(2) 为解决第 1 个问题,需要将能够合并生成 FCP_{k+1} 的 FCP_k 分发至同一个计算单元,但是如何将这些 FCP_k 分发至同一个计算单元则又是一个新的挑战.例如,$\{a,b,c\}$和$\{b,c,d\}$是两个长度为 3 的 FCP,FCP-DM 算法将会对$\{a,b,c\}$和$\{b,c,d\}$进行合并,生成长度为 4 的 $CP\{a,b,c,d\}$,继而判断$\{a,b,c,d\}$是否为 FCP_4.分布式环境下,$\{a,b,c\}$和$\{b,c,d\}$可能分布在不同的计算节点 n_1 和 n_2 上,n_1 和 n_2 无法直接获取对方的数据,这为生成伴随模式$\{a,b,c,d\}$增加了很大难度.

为解决上述问题,本文设计基于⟨key,value⟩的 FCP 分发策略.该策略首先令能够合并生成 CP_{k+1} 的 FCP_k 具有相同的 key 值,然后令每个 key 值由 FCP-DMDF 中唯一的计算单元负责.此时,根据 key 值与计算单元的映射关系,可将具有相同 key 值的 FCP_k 分发至同一个计算单元,继而由该计算单元对这些 FCP_k 进行聚合,生成相应的 CP_{k+1},最后判断 CP_{k+1} 是否为 FCP_{k+1}.

假设 FCP_k 和 FCP'_k 分别表示两个不同的长度为 k 的 FCP,FCP_{k+1} 表示一个长度为$(k+1)$的 FCP.如果 $FCP_k \subset FCP_{k+1}$,$FCP'_k \subset FCP_{k+1}$ 且 $FCP_k \cup FCP'_k = FCP_{k+1}$,那么 FCP_k 和 FCP'_k 至少含有$(k-1)$个相同元素.将 FCP_k 的任意一个由$(k-1)$个不同元素构成的集合记作 e_k,那么 FCP_k 将包含 C_k^{k-1} 个不同的 e_k^i $(i=1,2,\ldots,k)$集合.对于任意集合 e_k^i,可能存在一个频繁伴随模式 FCP_k^i 满足 $FCP_k \cap FCP_k^i = e_k^i$,这种情况下,$FCP_k \cup FCP_k^i$ 可能产生 CP_{k+1}.因此,对于任意一个 FCP_k,以每个 e_k^i 为 key,以 FCP_k 自身为 value 生成 k 个形如(e_k^i, FCP_k)的元组,并将每个元组根据 e_k^i 分发到不同的 PE,这使得含有相同 e_k^i 的不同元组将被发送至同一个 PE.该类 PE 接收到这些元组之后,能够将这些元组进行聚合,构成 CP_{k+1},并将该 CP_{k+1} 发送至下一层计算单元由其判断该 CP_{k+1} 是否为 FCP_{k+1}.

上述策略采用(e_k^i, FCP_k)的形式对长度为 k 的 FCP 进行分发,使得能够形成 FCP_{k+1} 的任意两个 FCP_k(无论它们是否在同一个计算单元)一定被分发至同一个 PE,保证了挖掘结果的准确性和完整性.

4.3.3 FCP 连续增量挖掘方法

由于数据流持续到达且没有边界,新的数据流到达之后,可能与已有数据形成新的 FCP,因此需要对连续到达的数据流持续监控,设计增量挖掘方法实时发现由新到达的数据流所形成的 FCP.在增量挖掘方法中,我们始终维护 FCP-DMDF 框架,并将已处理的有效数据始终在内存中进行维护.新的数据流到达之后,只需将这些数据与已有的有效数据进行比对,即可发现新生成的 FCP.

由于数据流的规模无限扩大,导致无法在内存存储所有数据,需要对过期数据进行删除.由于形成 FCP 的所有元素的时间间隔不大于 τ,如果某个元素的产生时间与当前时间的间隔大于 τ,那么该元素可以被安全删除,原

因是它不可能和其他元素共同构成新的 FCP.从节省内存角度考虑,则应该实时删除过期元素.然而,如果令每个计算节点持续监视其维护的每条数据是否过期,那么该操作将产生很大的计算代价.为使内存使用效率和计算代价两方面达到一个平衡,本文设计了延迟删除策略.该策略中,每个计算节点每隔 Δt 时间令其所有的逻辑计算单元执行一次对过期数据的删除操作.实验部分,我们将 Δt 的默认值设置为 $\tau.\Delta t$ 值越大,删除操作的计算代价越小,但是内存消耗增大;Δt 值越小,内存消耗越少,但删除操作代价增大.因此,可以调整 Δt 的值来平衡内存使用效率和删除计算代价.实验部分将对 Δt 对于算法内存使用情况的影响进行测试.

由于 FCP-DMDF 始终在内存中维护有效数据,那么对于一个最新到达的 segment,如果它能够和已有数据形成 FCP,那么只需要在已有数据的基础上处理该 segment,便可发现由新到达的 segment 与已有 segment 所形成的所有 FCP,从而实现多数据流 FCP 的连续增量挖掘.为解决这一问题,本文设计了 Incremental-mining 算法并给出算法的伪代码.对于一个新生成的 segment,Incremental-mining 算法首先生成该 segment 的所有 CP_2,之后对应的 PE 将 CP_2 与已有数据进行比对,得到 FCP_2.然后,基于 Apriori 启发式思想,由对应的 PE 对 FCP_2 所能构成的 FCP 进行逐级并行检测,从而得到该 segment 与已有数据形成的所有 FCP.在 Incremental-mining 算法对 segment 的处理过程中,仅有少量 PE 参与计算,因此可节省计算资源.

Incremental-mining 算法.

输入:segment G;

输出:G 和已有 segment 新形成的所有 FCP.

$\mathcal{H}_i$ 表示存储 FCP_i 的集合($i\geqslant 2$);

1. $\mathcal{H}_2=\varnothing$;
2. PartitionPE 接收 G 并生成 G 的所有 CP_2;
3. 将 CP_2 发送至对应的 CP_2-PE;
4. CP_2-PE 将 CP_2 聚合,并将其发送至 FCP_2-PE;
5. FCP_2-PE 检测 CP_2 是否为 FCP_2,添加 FCP_2 至 $\mathcal{H}_2$,并将 FCP_2 发送至 CP_3-PE;
6. **For** (i=3; $\mathcal{H}_{i-1}\neq\varnothing$; i++)
7. $\mathcal{H}_i=\varnothing$;
8. CP_i-PE 接收 FCP_{i-1},将 FCP_{i-1} 与已有 FCP_{i-1} 进行聚合,构成 CP_i;
9. 发送 CP_i 至 FCP_i-PE;
10. **If** (CP_i 是 FCP_i) //由 FCP_i-PE 执行
11. 添加 FCP_i 至集合 $\mathcal{H}_i$;
12. 发送 FCP_i 至对应的 CP_{i+1}-PE;
13. **End If**
14. **End For**

图 6 给出在 FCP-DMDF 上利用 Incremental-mining 算法发现 FCP_3 的一个例子.在该例子中,假设数据流 S_1 至 S_i 的 segment 已被处理,此时数据流 S_{i+1} 产生一个新的 segment(G_{i+1}),下面是利用 Incremental-mining 算法发现由新产生的 segment 所形成的 FCP.首先由 ReceivePE 将每个数据流划分成不同的 segment;对于一个 segment,PartitionPE 生成其包含的所有 CP_2.CP_2-PE 将来自不同数据流的相同 CP_2 发送至 FCP_2-PE,然后由 FCP_2-PE 判断某个 CP_2 是否为 FCP_2.FCP_2-PE 根据第 4.3.2 节介绍的分发策略,将得到的 FCP_2 发送至对应的 CP_3-PE,每个 CP_3-PE 将收到的 FCP_2 进行聚合,生成 CP_3,最后由 FCP_3-PE 检测 CP_3 是否为 FCP_3.

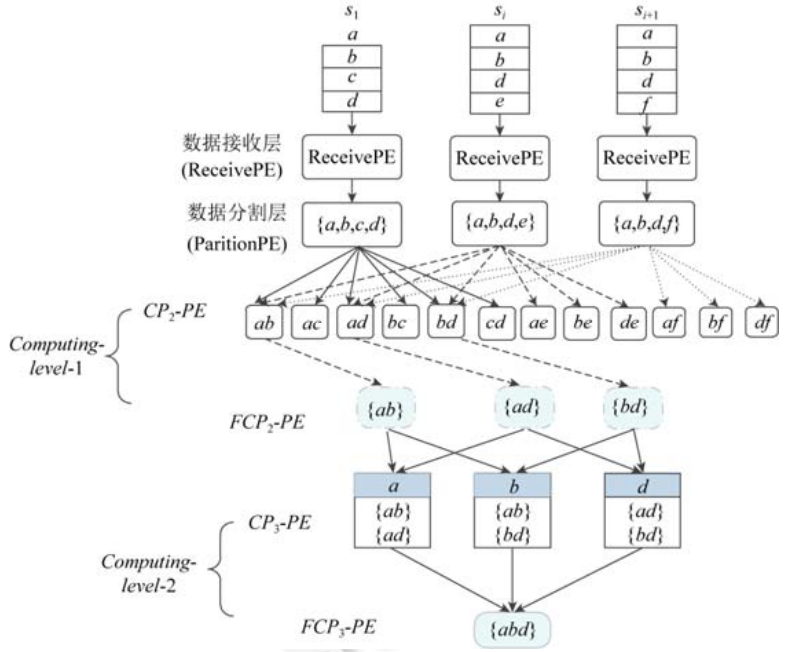

Fig.6 Procedure of mining FCP_3 (θ=3)

图 6 FCP_3 挖掘示意图(θ=3)

4.3.4 Incremental-mining 算法时间复杂度分析

分布式环境下,Incremental-mining 算法处理一个长度为 m 的 segment 片段 G 的时间复杂度的下界为 $O\left(\frac{m^2}{z}\right)$,最坏情况下的时间复杂度为 $O\left(\frac{m\times 2^m}{z}\right)$,$z$ 表示运行 Incremental-mining 算法的一层 CP_i-PE 或 FCP_i-PE 的数量.

证明:Incremental-mining 算法处理 G 时,首先生成 C_m^2 个 CP_2,并检测每一个 CP_2 是否为 FCP_2.如果每一个 CP_2 均不是 FCP_2,则算法终止.为了便于描述,假设 C_m^2 个 CP_2 均匀分布到 z 个 PE 上进行并行处理,那么每个 PE 负责 CP_2 的数量约为 $\frac{C_m^2}{z}\left(\frac{C_m^2}{z}\geqslant 1\right)$,单个 PE 的计算时间为 $\frac{C_m^2}{z}\times t_d$ (t_d 为单个 CP_2 的检测时间).此时的时间复杂度为 $O\left(\frac{m^2}{z}\right)$.

下面讨论 Incremental-mining 算法最坏情况下的时间复杂度.当 G 能够形成 FCP_m 时,Incremental-mining 算法的计算时间最长.一般情况下,只有部分伴随模式能够构成频繁伴随模式.不失一般性,假设 G 的长度为 i 的子集是 FCP_i 的概率为 $p(0<p<1)$,那么 G 所构成的 FCP_i 的数量为 $p\times C_m^i$,其中,C_m^i 是 G 中包含 i 个元素的子集的个数.对于每个 FCP_i,Incremental-mining 算法将根据第 4.3.2 节的分发策略产生它的 i 个子集并将它们发送至对应的 PE,该步骤中单个 PE 的计算时间为 $\frac{i\times p\times C_m^i}{z}\times t_h$ (t_h 表示发送 FCP_i 一个子集的时间); $p\times C_m^i$ 个 FCP_i 至多形成 $p\times C_m^i$ 个 CP_{i+1},因此单个 PE 检测 CP_{i+1} 是否为 FCP_{i+1} 的时间为 $\frac{p\times C_m^i}{z}\times t_d$.因此,发现 G 中所有 FCP 总的时间为 $\sum_{i=1}^{m}\frac{i\times p\times C_m^i}{z}\times t_h+\frac{p\times C_m^i}{z}\times t_d$.由于 p、t_h、t_d 均为常数,因此,Incremental-mining 算法最坏情况下的时间复

杂度为 $O\left(\frac{m\times 2^m}{z}\right)$.

虽然 Incremental-mining 算法最坏情况下的时间复杂度为 $O\left(\frac{m\times 2^m}{z}\right)$,但它的实际计算时间要小很多,原因如下.

(1) 真实情况下 p 值非常小,因此需要检测的 FCP_i 数量远小于 C_m^i, 即 $p\times C_m^i < C_m^i$;

(2) Incremental-mining 算法运行在多个物理计算节点之上,能够利用大量的 PE 实现对 FCP_i 的并行检测,将大大缩短计算时间.因此,物理节点越多,总的 PE 的数量越多(即 z 值越大),计算时间越短,这与实验部分中图 10 所示结果一致.

5 实 验

首先,本文基于 Apache 开源流数据处理平台 S4[21]实现了 FCP-DMDF.S4 中的逻辑处理单元称为 PE,每个物理节点可以运行任意多个 PE.基于 S4 平台实现 FCP-DMDF 时,FCP-DMDF 中 PE 的功能可以由 S4 的 PE 来实现.S4 中 PE 之间通过发送和接收 Event 进行数据传输.每个 PE 指定其所接收的 Event 的类型和 key 值.任意一个 PE 可以接受其他 PE 发送的 Event,也可以将本地计算结果以 Event 的形式发送给其他 PE.对一个新产生的 Event,如果已有的 PE 根据 Event 的类型和 key 值都无法处理该 Event,那么 S4 将自动创建一个新的 PE 来处理该 Event.由于 S4 的并行处理思想被当前大多数流数据分布式计算平台所采用,因此,本文所设计的 FCP-DM 分布式挖掘方法也易于部署到 Storm、Spark Streaming 等平台.

5.1 实验配置

实验采用 8 台戴尔 R210 的服务器,每台服务器配置主频 2.4GHz 的 Intel 处理器和 8G 内存,服务器之间通过 1Gbps 带宽的以太网相连.分布式平台采用 S4-0.6.0 版本(http://incubator.apache.org/s4/download/).以山东省济南市 2015 年 5 月 1 日 7:00~12:00 产生的 320 万条过车记录(vehicle passing records,简称 VPR)为测试数据集,从中挖掘频繁伴随车辆.该数据集中每条过车记录可以看作是一个四元组$\langle r_i,l_i,m_i,tm_i\rangle$,$r_i$ 表示该记录的编号,l_i 表示该车的车牌号,m_i 表示该记录所对应的监控卡口编号,tm_i 表示该记录产生的时间.整个测试集中过车记录是由不同卡口产生的,如果将同一个卡口连续产生的过车记录看成一个数据流,那么若干卡口就对应若干数据流.实验中要查找的伴随车辆是指很短时间内连续通过某卡口,并在一段时间内以同样方式通过多个卡口的一组车辆.因此,在测试数据集中发现伴随车辆就相当于从多个卡口产生的数据流中发现 FCP.FCP 定义中相关参数(ξ、τ和θ)将对 FCP-DM 算法的效率和挖掘结果产生影响,因此,每组实验均标明各个参数的取值.缺省情况下,ξ=60(s),θ=4,τ=2(h).每组实验均重复 5 次,取 5 次结果的平均值作为最终实验结果.

5.2 实验内容

5.2.1 FCP-DM 算法效率评估

图 7 将测试数据集模拟为 523 个数据流,分别以 20 000VPR/s、40 000VPR/s、60 000 VPR/s、80 000 VPR/s、100 000 VPR/s 的速率发送至 FCP-DMDF,以测试系统的抗压性.实验结果表明,FCP-DM 算法的最大处理能力约为 60 000 条/s.在该组实验中,我们在内存中设置一个队列 Q 用以缓存到达的 VPR.*FCP-DM*(t)表示 FCP-DM 处理测试数据集所需总的时间,*FCP-DM*(Q)表示队列 Q 缓存 VPR 的最大数量.在图 7 中,当数据到达速率小于 60 000 条/s 时,随着数据速率的提高,*FCP-DM*(t)明显下降.此时,由于 FCP-DM 处理速率大于数据到达速率,所以 *FCP-DM*(Q)趋近于 0.当速率达到 60 000 VPR/s 后,FCP-DM(t)趋于平稳,表明算法是以最大处理能力运行的,*FCP-DM*(Q)开始明显增加.

图 8 表明,Yu 等人提出的 CooMine 算法的最大处理能力约为 8 000VPR/s(单个计算节点)[1].通过图 7 和图 8 的比较可以发现,FCP-DM 在 8 个计算节点上的处理能力几乎是 CooMine 算法的 8 倍.也就是说,FCP-DM 在 8 个计算节点的处理效率几乎相当于在 8 个计算节点上分别运行 CooMine 算法的效率总和.这是因为,CooMine

算法是针对单机环境设计的多数据流频繁伴随模式发现算法,该算法为了节省内存(压缩 segment)和减少索引维护代价,设计和采用 Seg-tree 树型索引结构.本文设计的 FCP-DM 算法是针对分布式计算环境,拥有较充裕的内存,因此采用了分布式哈希索引.虽然 CooMine 算法对基于 Seg-tree 的数据查找操作进行了大量优化,但是 FCP-DM 算法中基于哈希索引的查询操作(如查找元素、伴随模式等)要比 CooMine 算法中基于 Seg-tree 的查询操作具备更高的查询效率,而哈希索引的内存使用效率要低于 Seg-tree 索引结构.因此,在 8 台机器上运行 FCP-DM 算法的计算效率接近于在 8 个计算节点上单独运行的 CooMine 算法的计算效率之和.

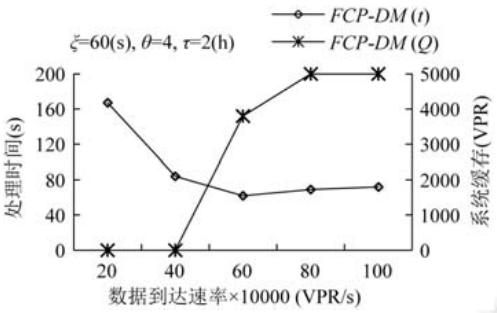

Fig.7 Processing capability of FCP-DM
图 7 FCP-DM 方法处理能力

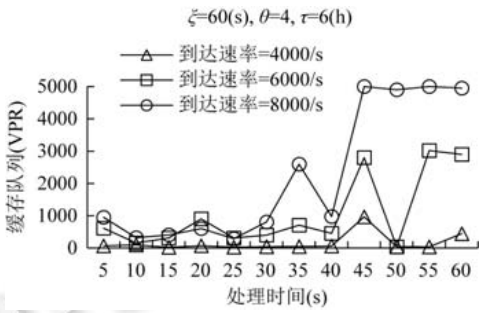

Fig.8 Processing capability of CooMine
图 8 CooMine 算法处理能力

为更加直观地比较 FCP-DM 算法和 CooMine 算法的性能,分别令这两种算法处理相同规模的过车记录数据集,并调整过车记录数据集的规模来比较两者的处理时间.该组实验是将整个数据集一次性地加载到系统中,算法处理时间是指从处理第 1 条数据开始至最后一条数据处理完成的时间跨度.图 9 所示的实验结果表明,FCP-DM 算法(采用 8 个物理节点)的处理时间要明显小于 CooMine 算法的处理时间,并且随着数据规模的扩大,FCP-DM 算法处理时间的增长速率明显小于 CooMine 算法的时间增长速率.

图 10 测试数据到达速率为 40 000 条/s 时 FCP-DM 算法处理不同规模数据时的可扩展性.该组实验中,分别采用 2、4、6、8 个计算节点处理不同规模的数据集.实验结果表明,随着计算节点个数的增加,FCP-DM 的处理时间明显下降.VPR 数量越多,处理时间的下降趋势越明显,表明 FCP-DM 方法在处理大规模数据流时具备良好的可扩展性.

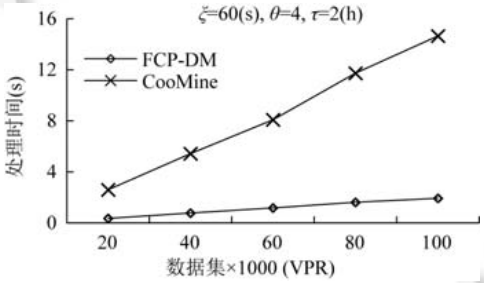

Fig.9 Comparison of FCP-DM and CooMine w.r.t. processing time
图 9 FCP-DM 与 CooMine 处理时间比较

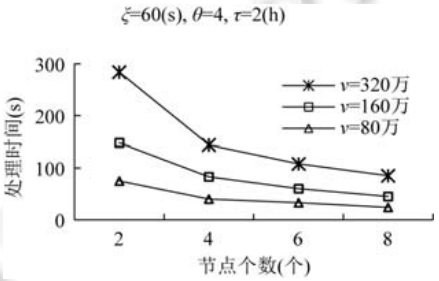

Fig.10 The scalability of FCP-DM
图 10 FCP-DM 的可扩展性

5.2.2 FCP-DM 算法相关参数对算法性能影响的评价

图 11 和图 12 主要对本文所采取的负载迁移策略的性能进行验证,主要测试单个 PartitionPE 所能承载的 segment 的最大数量β、数据到达速率以及数据集规模对于 PartitionPE 数量的影响.该组实验共采用 4 个物理计算节点.实验开始之前,我们对测试数据集中的所有元素的标识进行遍历,得到一个能够包含测试数据集中所有元素的标识范围$[w_s,w_e]$.实验开始时,初始化一个 PartitionPE,令其 key 值范围为$[w_s,w_e]$,也就是说,所有初始的 PartitionPE 能够接收所有的 segment.当初始 PartitionPE 接收但来不及处理的 segment 数量达到阈值β时,该

PartitionPE 就会分裂,这样就会产生多个 PartitionPE.这里需要注意的是,PartitionPE 一旦创建就不会被删除,本组实验将记录处理过程中所有的 PartitionPE 的数量.

图 11 的实验结果表明,当β值一定时,随着数据到达速率的增加,PartitionPE 的个数明显增多,这是因为数据到达越快,PartitionPE 所积压的未处理的 segment 越多,这样发生负载迁移的次数越多,导致更多的 PartitionPE 产生.当数据到达速率一定时(例如 30 000VPR/s 时),随着β值的增大,PartitionPE 的数量逐渐减少.这是因为随着β值的增大,PartitionPE 能够缓存的 segment 增多,使得 PartitionPE 的数量减少.图 12 中,令 FCP-DM 处理不同规模的数据集并观察 PartitionPE 的数量.当数据到达速率和β值固定时,我们发现数据集的规模对 PartitionPE 的数量没有显著影响,这表明 PartitionPE 的数量只与数据到达速率和β值有关,与待处理的数据集规模无关.

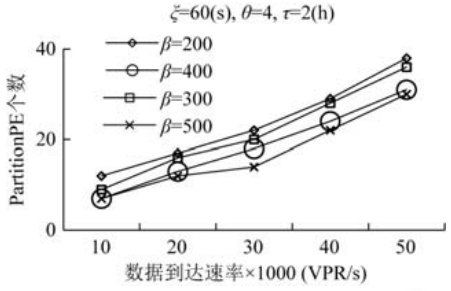

Fig.11 Number of PartitionPE w.r.t. β

图 11 参数β对 PartitionPE 个数的影响

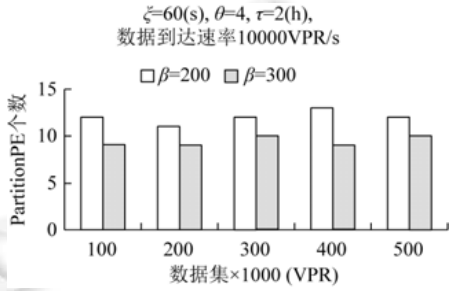

Fig.12 Number of PartitionPE w.r.t. scale of data

图 12 数据集规模对 PartitionPE 个数的影响

在延迟删除策略中,调整参数 Δt 的值可以改变对过期数据的删除周期.图 13 测试了参数 Δt 对于算法运行时内存使用状况的影响.实验策略是随机抽取 100 万条 VPR 作为测试数据集,令数据发送速率为 10 000VPR/s,然后观察不同时刻的所有计算节点的内存使用状态.图 13 的纵坐标表示所有计算节点运行 FCP-DM 算法时消耗的内存之和,横坐标表示算法的运行时间.实验结果表明,算法运行初期 Δt 取不同值时,内存使用量均增大,这是因为此时内存中没有过期数据,致使内存累积的数据增加.当算法运行 50s 之后,内存使用量出现明显波动,这是因为算法每隔 Δt 时间都会对过期数据进行删除.总的来说,Δt 取值越小,平均的内存使用量也越少,但是删除次数也相对增加.图 14 测试了参数ξ对 FCP-DM 处理时间的影响.图 14 所示实验结果表明,算法处理时间随着ξ值的增大而增加.这是因为,ξ越大,segment 长度越大,此时所产生的 CP 数量也越大.由于 FCP-DM 需要对所有的 CP 进行处理,因此,FCP-DM 算法的处理时间随着 CP 数量的增大而明显增加.

FCP-DM 算法的目标是准确发现给定数据集中所有的 FCP,即查全率和查准率应为 100%.由于 CooMine 算法的目标也是准确发现给定数据流中所有的 FCP[1],其查全率和查准率均为 100%.表 1 比较了 FCP-DM 算法和 CooMine 算法在同一数据集上所发现的 FCP 数量.当参数θ和 k 取值一定时,两种算法在该实验数据集上发现的 FCP 数量相同,从而验证了 FCP-DM 算法的查全率和查准率均为 100%.

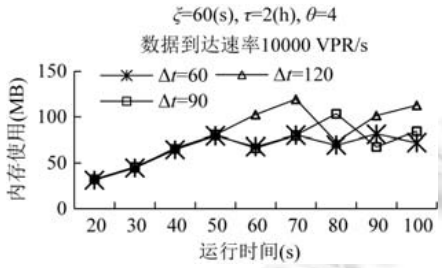

Fig.13 Memory consumption w.r.t. Δt

图 13 参数 Δt 对于内存使用的影响

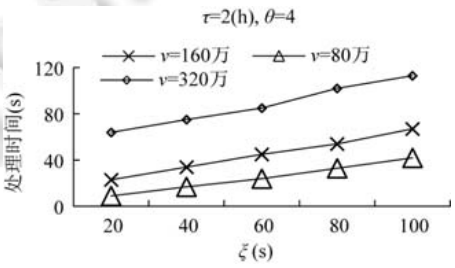

Fig.14 Processing time w.r.t. ξ

图 14 ξ对处理时间的影响

Table 1 Comparison of FCP-DM and CooMine w.r.t. the number of discovered FCP

表 1 FCP-DM 和 CooMine 挖掘结果数量比较

FCP-DM				CooMine			
	k=2	k=3	k=4		k=2	k=3	k=4
θ=3	953	54	7	θ=3	953	54	7
θ=4	182	7	0	θ=4	182	7	0
θ=5	32	2	0	θ=5	32	2	0

5.2.3 FCP-DM 挖掘结果评价

图 15 和图 16 主要测试数据规模和参数θ对 FCP-DM 算法挖掘结果的影响,其中,k 表示 FCP 的长度,ξ、τ和θ的含义见定义 2 和定义 3.该组实验中,一个 FCP 代表一组频繁伴随车辆.图 15 所示实验结果表明,当 FCP 定义中相关参数(ξ、τ、θ)的值固定时,随着数据规模的扩大,FCP-DM 发现的 FCP 数量逐渐增多;当数据规模一定时,FCP 的长度(k 值)越大,FCP 的数量越少,也就是说,用户指定一组频繁伴随车辆中的车辆数目越大,现实中这样的车辆组合越少,越符合我们的直观认识.FCP 定义中参数θ将对 FCP 的数量产生较大影响,图 16 所示实验结果表明,随着θ值的增大,FCP 数量变小,也就是说,如果用户要求一组 FCP 伴随出现的数据流越多,那么这样的 FCP 数量越少,实验结果同样符合预期.目前,FCP-DM 算法已被应用至山东省某地市智能交通综合管控平台,用于快速发现频繁伴随车辆.图 17 和图 18 给出了 FCP-DM 算法在该交通管控平台发现频繁伴随车辆的部分结果展示.其中,地图中的红色箭头是车辆伴随行驶方向,图片右侧是发现的频繁伴随车辆组合.

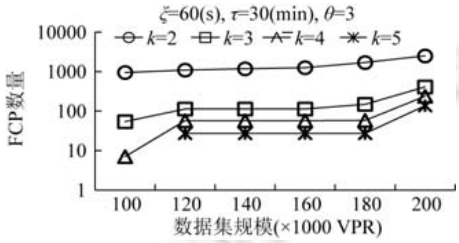

Fig.15 Number of FCP w.r.t. scale of data

图 15 数据规模对 FCP 数量的影响

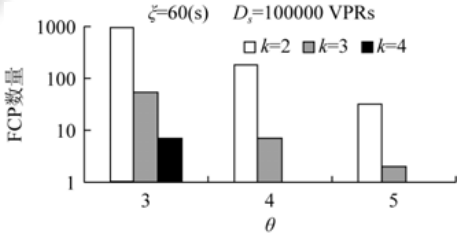

Fig.16 Number of FCP w.r.t. θ

图 16 参数θ对 FCP 数量的影响

Fig.17 Demonstration of applying FCP-DM (1)

图 17 FCP-DM 实际应用展示(1)

Fig.18 Demonstration of applying FCP-DM (2)

图 18 FCP-DM 实际应用展示(2)

6 总 结

本文研究了海量多数据流 FCP 挖掘问题,提出了 FCP-DM——一个可部署在分布式流数据处理平台的分布式挖掘方法.在该方法中,本文重点研究分布式环境下多计算节点协同挖掘大规模数据流所形成的 FCP 时面临的负载均衡问题、数据分布式存储背景下的 FCP 生成问题以及连续数据流的 FCP 增量挖掘问题,最后通过大量实验对 FCP-DM 方法的各项性能和挖掘结果进行充分验证,并给出了算法在实际应用中的效果展示.后续工作将继续研究如何对发现的大量 FCP 进行排序优化,从而将更符合用户偏好的 FCP 排在挖掘结果的前面.

References:

[1] Yu ZQ, Yu XH, Liu Y, Li WZ, Pei J. Mining frequent co-occurrence patterns across multiple data streams. In: Proc. of the 19th Int'l Conf. on Extending Database Technology. Brussels, 2015. 73–84.

[2] Chang JH, Lee WS. Finding recent frequent itemsets adaptively over online data streams. In: Proc. of the 9th SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Washington, 2003. 487–492.

[3] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. of the 20th Int'l Conf. on Very Large Data Bases. Santiago de Chile, 1994,1215:487–499.

[4] Manku GS, Motwani R. Approximate frequency counts over data streams. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong, 2002. 346–357.

[5] Yu JX, Chong Z, Lu H, Zhou A. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In: Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto, 2004. 204–215.

[6] Leung CS, Khan QI. Dstree: A tree structure for the mining of frequent sets from data streams. In: Proc. of the 2006 IEEE Int'l Conf. on Data Mining. Hong Kong, 2006. 928–932.

[7] Li J, Maier D, Tufte K, *et al*. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. In: Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Baltimore, 2005. 39–44.

[8] Mozafari B, Thakkar H, Zaniolo C. Verifying and mining frequent patterns from large windows over data streams. In: Proc. of the 29th Int'l Conf. on Data Engineering. Cancun, 2008. 179–188.

[9] Karp RM, Papadimitriou CH, Shenker S. A simple algorithm for finding frequent elements in streams and bags. ACM Trans. on Database Systems, 2003,28(1):51–55.

[10] Chi Y, Wang H, Yu PS, Muntz R. Moment: Maintaining closed frequent itemsets over a stream sliding window. In: Proc. of the Int'l Conf. on Data Mining. Brighton, 2004. 59–66.

[11] Silva A, Antunes C. Multi-relational pattern mining over data streams. Data Mining and Knowledge Discovery, 2015,29(6): 1783–1814.

[12] Li HF, Zhang N, Zhu JM, Cao HH. Frequent itemset mining over time-sensitive streams. Chinese Journal of Computers, 2012,35(11):2283–2293 (in Chinese with English abstract).

[13] Guo J, Zhang P, Tan J. Mining frequent patterns across multiple data streams. In: Proc. of the 20th ACM Conf. on Information and Knowledge Management. Glasgow, 2011. 2325–2328.

[14] Mao YX, Chen TB, Shi BL. Efficient method for mining multiple-level and generalized association rules. Ruan Jian Xue Bao/ Journal of Software, 2011,22(12):2965−2980 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3907.htm [doi: 10.3724/SP.J.1001.2011.03907]

[15] Ao X, Luo P, Li C, *et al*. Online frequent episode mining. In: Proc. of the 31st Int'l Conf. on Data Engineering. Seoul, 2015. 891–902.

[16] Patnaik D, Laxman S, Chandramouli B, Ramakrishnan N. Efficient episode mining of dynamic event streams. In: Proc. of the 12th IEEE Int'l Conf. on Data Mining. Brussels, 2012. 605–614.

[17] Vanchinathan HP, Marfurt A, Robelin CA. Discovering valuable items from massive data. In: Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Melbourne, 2015. 1195–1204.

[18] Saber S, Reza A, Florent M. Fast parallel mining of maximally informative *k*-itemsets in big data. In: Proc. of the 15th IEEE Int'l Conf. on Data Mining. 2015. 350–368.

[19] Wang L, Yang B, Chen YH, Zhang XQ, Orchard JF. Improving neural-network classifiers using nearest neighbor partitioning. IEEE Trans. on Neural Networks and Learning Systems, 2017,28(10):2255–2267.

[20] Han SY, Chen YH, Tang GY. Fault diagnosis and fault-tolerant tracking control for discrete-time systems with faults and delays in actuator and measurement. Journal of the Franklin Institute, 2017,354(12):4719–4738.

[21] Neumeyer L, Robbins B, Nair A, Kesari A. S4: Distributed stream computing platform. In: Proc. of the 2010 IEEE Int'l Conf. on Data Mining Workshops. Washington, 2010. 170–177.

附中文参考文献:

[12] 李海峰,章宁,朱建明,曹怀虎.时间敏感数据流上的频繁项集挖掘算法.计算机学报,2012,35(11):2283–2293.

[14] 毛宇星,陈彤兵,施伯乐.一种高效的多层和概化关联规则挖掘方法.软件学报,2011,22(12):2965−2980. http://www.jos.org.cn/1000-9825/3907.htm [doi: 10.3724/SP.J.1001.2011.03907]

于自强(1984－),男,山东青岛人,博士,讲师,CCF 专业会员,主要研究领域为时空数据分布式查询,流数据分布式计算.

禹晓辉(1977－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为海量数据管理与分析.

董吉文(1984－),男,博士,教授,CCF 高级会员,主要研究领域为数字图像处理.

王琳(1984－),男,博士,副教授,CCF 专业会员,主要研究领域为机器学习,数据反向建模.