ISSN 1004-9037,CODEN SCYCE4
Journal of Data Acquisition and Processing Vol. 39,No. 5,Sep. 2024,pp. 1163－1181
DOI:10. 16337/j. 1004-9037. 2024. 05. 009

http://sjcj. nuaa. edu. cn
E-mail:sjcj@nuaa. edu. cn
Tel/Fax:＋86-025-84892742

面向时空轨迹流的共同运动模式分布式挖掘算法

余舒鹏，吴春雨，赵　斌，吉根林

(南京师范大学计算机与电子信息学院/人工智能学院,南京 210023)

摘　要:从轨迹流中挖掘共同运动模式指在同一时间内发现具有相同运动行为的移动对象群体,在交通物流、疫情防控等方面具有重要意义。然而,现有研究面对大规模轨迹流数据难以做到快速响应。因此,本文首先提出了基于滑动窗口的分布式时空轨迹流共同运动模式挖掘算法,使用滑动窗口计算模型代替快照计算模型,利用增量式更新代替重新计算,使算法更适用于无界且快速到达的轨迹流数据,在效率和有效性方面呈现更好的性能。其次,针对分布式流处理系统中由于负载不均导致性能下降问题,提出了自适应多级动态数据分发策略,该策略能够适应轨迹流数据的动态变化,实时监测系统负载情况并根据负载不均的程度做出适当调整。最后,基于分布式流处理平台Flink实现了上述功能,并通过真实数据集的实验证明本文提出的算法比基准方法具有更快的响应速度和更低的延迟。

关键词:时空轨迹流;共同运动模式;分布式系统;滑动窗口;负载均衡

中图分类号:TP391　　**文献标志码**:A

Distributed Mining Algorithm for Co-movement Patterns in Spatio-Temporal Trajectory Streams

YU Shupeng, WU Chunyu, ZHAO Bin, JI Genlin

(School of Computer and Electronic Information/School of Artificial Intelligence, Nanjing Normal University, Nanjing 210023, China)

Abstract: Mining co-movement patterns from trajectory streams refers to discovering groups of moving objects with same behaviors at the same time, which is essential for transportation logistics, epidemic prevention and control and so on. However, the existing research faces difficulties in responding quickly to large-scale trajectory data streams. Therefore, this paper proposes a novel distributed sliding window algorithm for mining co-movement patterns from spatio-temporal trajectory streams. The algorithm employs a sliding window computing model instead of a snapshot computing model, and utilizes incremental updates instead of re-computing, making it more suitable for handling unbounded and rapidly arriving trajectory data streams. The proposed algorithm demonstrates superior performance in terms of efficiency and effectiveness. Secondly, to address the issue of load imbalance in distributed stream processing systems, this paper proposes an adaptive multi-level dynamic data partitioning strategy. This strategy can adapt to the dynamic changes in trajectory stream data, continuously monitor the system load in real-time, and make appropriate adjustments based on the degree of load imbalance. Finally, this paper implements the above functions on the Flink distributed big data processing platform and uses real data sets for experiments. Comprehensive empirical study demonstrates that the proposed algorithm has faster

收稿日期:2023-07-28;**修订日期**:2023-11-03

response speed and lower delay than the baseline method.

Key words: spatio-temporal trajectory streams; co-movement patterns; distributed system; sliding window; load balance

引　言

全球定位技术和无线通信技术的快速发展以及移动智能终端的广泛使用,使得持续获取海量移动对象位置数据成为可能。当前,移动互联网技术已经深入到人类生活工作的方方面面,海量位置数据可以被广泛、不间断地采集,汇聚形成具有时变性、持续性、快速到达的时空轨迹流数据。对轨迹流进行分析与挖掘可以实时监测、持续跟踪、及时发现移动对象的运动行为及其活动规律,可被广泛应用于交通物流、疫情防控和旅行推荐等多个领域。

学术界在流式处理和分布式计算方面对共同运动模式挖掘开展了深入的研究。共同运动模式是综合了Flock[1]、Convoy[2]、Swarm[3]、Platoon[4]和Traveling Companions[5]等经典伴随型运动模式的通用模型,用于发现具有相同运动特征的移动对象群体。Chen等[6]用Flink计算平台对大规模的轨迹流数据进行分布式处理,提出了共同运动模式的分布式实时挖掘框架,获得广泛认可。之后在线共同运动模式分布式挖掘系统CoMing[7]和MaSEC[8]被相继提出,将共同运动模式挖掘应用到现实场景中,但现有工作在流式处理和分布式计算方面依然存在改进的空间。

现有研究工作在流式处理的计算模型设计和分布式计算中的数据分发方法上仍存在不足,导致模式挖掘效率较低。一方面,计算模型是影响轨迹流数据处理性能的重要因素。现有工作[5-11]大多采用快照的计算模型,该模型能够将无界数据流转化为多份有限数据集,便于计算机处理且方法简单。但在计算过程中产生了大量重复计算,并不适用于快速到达的轨迹流数据。另一方面,数据分发方法在分布式方案中是事关性能的重要考量。现有工作大多采用静态数据分发策略[6-10],策略简单、易于实现。但由于轨迹流数据的时空分布变化系统难以维持负载均衡,导致执行效率下降。

设计高效的时空轨迹流分布式共同运动模式挖掘算法颇具挑战。首先,在保证快速响应、低延迟要求下应对大规模轨迹流的分析与挖掘是一个难点。通常共同运动模式挖掘算法的时间复杂度较高,加之要处理大规模数据,所以性能表现不理想。轨迹流数据无界和快速到达的特性会造成因无法及时处理数据而形成的拥塞,因此快速处理滑动窗口中轨迹流数据是本文面临的挑战之一。其次,如何平衡调整数据分发后系统负载均衡程度和调整代价也是一个难点。当分布式系统出现负载不均时,通过调整数据分发使其恢复均衡状态不可避免地会产生调整代价。调整代价通常与系统均衡程度成正比,均衡程度越高,所需调整代价越大。因此寻找到两者之间平衡点是本文面临的另一挑战。

针对以上挑战,本文主要工作如下:(1) 提出了基于滑动窗口的分布式时空轨迹流共同运动模式挖掘算法,通过增量式更新而非重新计算提高挖掘算法的效率;(2) 针对各计算节点的负载不均问题,提出了自适应多级动态数据分发策略,该策略能够根据负载不均程度分级自动调整,既能使系统恢复均衡状态又不会产生较高调整代价;(3) 基于分布式流处理平台Flink实现了上述算法,并通过真实数据集验证了本文所提算法的有效性与效率。

1　相关工作

轨迹数据的共同运动模式挖掘具有较高的研究和应用价值,目前共同运动模式挖掘面向的轨迹数据形式主要分为历史轨迹和轨迹流两种。

1.1 面向历史轨迹数据的共同运动模式挖掘

关于面向历史轨迹的共同运动模式挖掘,学术界开展了广泛而深入的研究。2005年Kalnis等[7]提出“移动簇”,表示多个移动对象在一段时间内距离靠近且一起移动,这是对伴随模式的早期定义形式。随后,Benkert等[12]在“移动簇”概念的基础上提出Flock模式。Flock模式要求一定时间内,若干个移动对象在满足一定半径阈值的圆形空间内一起移动。2006年,Gudmundsson等[8]研究持续时间最长的Flock模式,分别提出了精确和近似的挖掘算法。Jeung等[2]针对Flock模式无法检测指定圆形范围外的对象,提出了Convoy模式,该模式要求群体对象间密度相连,共同移动连续且足够长的时间戳。2016年Yeoman等[13]专门研究“去中心化空间计算”场景下的Convoy模式挖掘方法,特别用于发现移动通信网络中的动态群体。之后Orakzai等[14]提出$k/2$-hop算法,进一步提升了Convoy模式挖掘算法性能。针对以上模式都无法应对时间不连续的情况,Li等[3]提出了Swarm模式,允许群体中的移动对象存在短暂的分离。而Swarm模式放弃了时间维度连续性的要求,又过于松弛,容易引入噪声干扰,因此Li等[4]提出了Platoon模式,要求移动对象在一起的时间长度须达到阈值,而且在一起的连续时间长度也须满足阈值要求。Helmi等[15]对于更为灵活共同运动模式,提出了多尺度的频繁共同模式挖掘算法,研究一组移动对象在不同空间尺度上频繁、重复出现的共同运动规律。以上研究均是在单机环境中。2015年于自强等[10]提出了两种在多个数据流中发现频繁共现模式的有效方法。Orakzai等[16]设计了分布式策略,将相同时间戳的数据散列到同一个节点独立处理,多个节点共同处理全部数据。Fan等[17]提出了一个综合多种伴随模式的通用模型,称为共同运动模式,该模式通过配置参数可以表示各种伴随模式(如Flock、Convoy、Swarm、Platoon等)。张敬伟等[18]针对分布式策略中存在大量松散连接的问题,提出了两阶段伴随模式挖掘框架。通过数据预处理优化、伴随优化等方法使伴随模式挖掘具有更好的性能。Tritsarolis等[19]提出在线预测共同运动模式的问题并提出共同运动模式相似性度量,用于将预测与实际聚类结果比对。

目前针对历史轨迹数据的共同运动模式挖掘技术日趋成熟,但由于挖掘结果过于滞后,难以应用于真实场景。基于以上考虑,许多学者提出了针对轨迹流的实时共同运动模式挖掘方案。

1.2 面向轨迹流数据的共同运动模式挖掘

针对轨迹流的共同运动模式挖掘,学术界也有许多研究。Vieira等[11]将Flock模式应用在轨迹流上,提出了剪枝算法减少冗余的候选簇,并设计了网格聚类算法以节省聚类时间。Tang等[5]提出了基于增量式算法挖掘伴游模式的方法,该方法仅维护移动对象的伙伴关系而不需要获取其具体空间坐标信息,因而具有较高的计算效率。Bhushan等[20]采用滑动窗口处理流式轨迹数据,提出基于图的增量算法挖掘Swarm模式。对于轨迹流中的共同运动模式挖掘,Chen等[6]用Flink计算平台对大规模的流式轨迹数据进行分布式处理,提出了共同运动模式的实时挖掘方法。Gao等[7]提出了一个实时挖掘共同运动模式的演示系统,该系统从轨迹流中及时挖掘共同运动模式。最近,Tritsarolis等[8]演示了一个群体移动模式发现系统MaSEC,用于从流式船舶位置数据中在线发现共同运动模式。在国内,朱美玲等[9]赋予Platoon模式新的定义,提出基于车牌识别数据流的Platoon模式挖掘算法,针对数据流的特点进行性能上的优化,使得车辆通过摄像头时能及时发现车辆Platoon模式。于自强等[10]提出面向大规模数据流的频繁共同运动模式分布式挖掘算法。该算法将每个数据流划分成若干个segment片段,构建出可以部署在分布式计算平台上的多层挖掘模型,并利用多计算节点对大规模数据流进行并行处理,从而达到实时发现频繁共同运动模式的效果。

尽管有部分研究工作涉及到对轨迹流的共同运动模式挖掘,但是这些工作大多使用快照计算模型,难以解决轨迹流数据量大及快速到达与模式挖掘代价高昂的矛盾。此外现有分布式解决方案[21-24]的重点大多集中在算法设计,很少关注轨迹流数据的分区策略,使得轨迹流数据挖掘算法效率难以突

破瓶颈。

2 问题定义

本节主要介绍共同运动模式的问题定义，表1列出了文中所使用的符号。

表1 符号定义

Table 1 Symbol definition

符号	定义	符号	定义
O	移动对象集合	$W.t$/s	滑动窗口开始时刻
T	时间序列	M	移动对象最小规模
S	时空轨迹流	K/min	最小持续时长
o_i	在时刻t_i到达的移动对象	L/min	最小局部连续时长
p_i	移动对象o_i在时刻t_i的位置	G/min	片段间最大间隔时长
η	滑动窗口大小/min	CP(M,K,L,G)	共同运动模式
θ	滑动步长/min	t_{buffer}	缓冲时间
W	连续窗口序列	N	计算节点数量
w	长度为η的窗口有限子序列		

给定移动对象集合$O=\{o_1, o_2, \cdots, o_n\}$和时间序列$T=\{t_1, t_2, \cdots, t_{i-1}, t_i, t_{i+1}, \cdots\}$，$t_i$表示时刻。

定义1(轨迹流) 时空轨迹流是指持续到达的由多个移动对象位置记录组成的时间序列，记为$S=\{(o_1, p_1, t_1), (o_2, p_2, t_2), \cdots, (o_i, p_i, t_i), \cdots\}$。其中，$(o_i, p_i, t_i)$表示某一移动对象在时刻$t_i$的位置记录，$t_i \in T$，$o_i$表示在时刻$t_i$到达的移动对象，$o_i \in O$，$p_i$表示移动对象$o_i$在时刻$t_i$的位置信息，$p_i=(x_i, y_i)$，$x_i, y_i \in \mathbf{R}^2$。

在轨迹流的运动模式挖掘中，最新的运动规律更受关注。滑动窗口模型能够很好地处理最近到达的轨迹流数据。

定义2(滑动窗口) 给定滑动窗口大小η和滑动步长θ，滑动窗口w是轨迹流S在时间间隔长度η的一段连续的有限子序列，滑动窗口开始时刻为$W.t$。在不引起歧义的情况下，下文将滑动窗口简称为窗口。

将轨迹流S的某个窗口记为w_i，w_i向前滑动即为其连续的最新窗口w_{i+1}，其中$w_{i+1}.t=w_i.t+\theta$。一个连续的窗口序列W满足：$\forall 1 \leqslant i < |W|, W[i+1].t = W[i].t + \theta$，$W[i]$表示序列$W$中的第$i$个元素。连续的窗口序列成为窗口片段，简称片段。根据以上定义，下文给出L-连续和G-连通的概念。L-连续控制片段的长度，G-连通控制连续片段之间的间隙长度。

定义3(L-连续) 给定$|W| \geqslant L$的片段W，序列$W'=\cup W$为L-连续。

定义4(G-连通) 任意相邻窗口之间的时间间隙最多为G个滑动步长，则序列W'是G-连通的，即$\forall 1 \leqslant i \leqslant |W|-1, W[i+1].t - W[i].t \leqslant G \times \theta$，其中$W[i]$表示集合$W$中的第$i$个元素。

下面给出基于滑动窗口的共同运动模式形式化定义。共同运动模式描述一组一起运动的移动对象，且同时满足5个约束条件：(1)邻近性：定义“一起运动”；(2)显著性：控制一起运动的移动对象的数量；(3)持续时间：控制移动对象一起运动的时间长度；(4)L-连续；(5)G-连通放松了“持续时间”的连续性。具体来说，移动对象一起运动的整个时间段不一定是严格连续的，在连续的片段之间允许存在间隙。因此，“L-连续”和“G-连通”分别控制每个连续片段的长度和两个连续片段之间的间隙。

定义5(共同运动模式) 给定时空轨迹流S，如果存在一个窗口序列W，满足以下5个约束条件，则

S的子集O是一个共同运动模式CP(M,K,L,G):(1)邻近性:在W的每个窗口中,O的移动位置属于同一簇;(2)显著性:$|O|\geqslant M$;(3)持续时间:$|W|\geqslant K$;(4)连续性:W为L-连续;(5)连通性:W为G-连通。

基于快照模型计算邻近性约束时,假设相同时刻报告每个移动对象的位置,每个快照中的移动对象在时间上严格对齐。本文在计算基于滑动窗口的邻近性约束时,引入缓冲时间t_{buffer}进行了时间上的放松,允许一定的"时空错位",对时间段t_{buffer}内不同时刻的移动对象计算近邻约束。考虑到移动对象运动具有时空局部性,短时间内的移动范围有限,本文方法具有合理性。为了提供近邻性约束的具体定义,本文选择依赖于基于密度的聚类进行定义。

定义6(分布式实时共同运动模式挖掘) 给定定义共同运动模式的参数M、K、L和G,计算节点个数N,分布式实时共同运动模式挖掘是在窗口序列$W=\{w_1,w_2,\cdots,w_n\}$中使用N个计算节点,找到所有共同运动模式,其中$w_n.t$为当前时间。

3 基于滑动窗口的时空轨迹流共同运动模式挖掘

共同运动模式挖掘首先需要使用聚类操作追踪移动对象的空间邻近性,聚类操作对于轨迹数据的模式挖掘具有十分重要的作用,但聚类操作也占据了模式挖掘整个过程的大量时间。大规模数据流源源不断、快速到达的特性要求共同运动模式挖掘算法必须快速高效。然而,计算共同运动模式需要对海量移动对象进行范围查询以及指数级的枚举操作,导致算法性能低下。面对共同运动模式挖掘的高计算复杂度,学术界现有研究工作取得了重要成果,主要包括CEPR[25]、ViStream[26]、CooMine[27]和ICPE[6]等。这些研究提出了有效的共同运动模式实时挖掘算法,但在计算模型选取方面略有疏忽,计算效率存在改进空间。现有研究大多采用快照作为计算模型,然而由于快照之间的计算是相互独立的,相邻快照之间因移动对象短时间内的邻近性具有相似性等因素,存在大量重复计算,使得算法性能下降。

基于滑动窗口的时空轨迹流共同运动挖掘算法框架包含3个阶段:基于滑动窗口的数据流分片、面向轨迹流的增量式聚类和基于位运算的模式枚举。其中面向轨迹流的增量式聚类是本文讨论的重点。

3.1 基于滑动窗口的数据流分片

使用窗口技术对轨迹流分片,分批次地处理实时到达的轨迹流数据,从而实时计算和分析。考虑到移动对象时空位置变化连续、短时间内的邻近性具有相似性的现实因素,本文采用窗口技术对无限的数据流从时间维度划分片段,记录移动对象的位置更新,也利于后续处理阶段进行增量式计算。

3.2 面向轨迹流的增量式聚类

聚类操作对于轨迹数据的模式挖掘具有十分重要的作用,占据模式挖掘整个过程的大量时间。现有研究工作基于快照模型进行模式定义,在每个快照上进行聚类,根据簇关系约束移动对象的空间邻近性。由于快照之间的计算是相互独立的,以致产生了大量的重复计算。此外,由于轨迹流采样频繁,其数据量和快照数量随时间不断增长,对这些数据进行聚类需要十分庞大的计算代价。

本文选择滑动窗口模型对轨迹流进行增量式聚类。固定大小的滑动窗口对最新到达的数据进行处理,模型本身与轨迹流采样率无关。同时,移动对象时空位置变化连续,短时间内的近邻性具有相似性,滑动窗口模型能够及时更新簇关系,无需对每个快照进行独立的聚类操作,提高了算法整体的效率。使用滑动窗口模型进行聚类操作时,直觉的方案是以窗口为单位,通过计算同一个滑动窗口内的位置记录得出满足空间邻近的移动对象集合。但是,与快照中移动对象的位置记录严格时间对齐不同,窗口中存在多个时间戳的位置记录,甚至同一个移动对象的多个记录,造成以滑动窗口为单位计算空间邻近性的混乱。因此,引入缓冲时间t_{buffer}进行时间上的放松,允许一定的"时空错位",对时间段t_{buffer}内不同时间戳的位置记录计算近邻约束。当t_{buffer}内出现同一个移动对象的多个记录时,最新时间

戳的位置记录参与计算。

面向轨迹流的增量式聚类与传统DBSCAN算法一样,将位置记录分为核心点、边界点和噪声点。当聚类完成时,核心点和边界点将被分配一个簇标识cid。与DBSCAN不同的是,本文算法的思想是通过计算进入和离开窗口的位置记录对上一次聚类结果的影响来更新本次聚类结果。$N_\varepsilon(p)$表示以位置记录p为中心、距离阈值ε内的一组数据点,$N_\varepsilon(p)$中位置记录的数量用$n_\varepsilon(p)$表示,并且始终保持最新。当窗口向前滑动θ步长时,新的轨迹数据进入窗口,而一些现有的轨迹数据离开窗口。追踪滑动窗口内这些发生变化的轨迹数据及其ε邻域受到的影响,并通过更新$n_\varepsilon(p)$以及类别标签$l(p)$即可保持簇的状态及时更新。显然,面向轨迹流的增量式聚类的主要任务是对当前窗口中的每一个点p重新计算$n_\varepsilon(p)$和$l(p)$,具体分为收集和聚类两个步骤,收集步骤找到所有对簇状态产生影响的位置记录,聚类步骤判断位置记录对簇分裂和合并的影响,更新簇状态。

3.2.1 收集

收集阶段的主要任务是发现对簇状态有影响的位置记录。轨迹流中变化的位置点对簇造成的影响最直接,移动对象的位置变化可能会改变位置点的类型,影响位置点间的密度可达关系,从而造成簇的变化。当窗口滑动时,离开窗口的核心点可能会造成簇的消减(包括分裂、缩小和消失),进入窗口的核心点可能会造成簇的增长(包括合并、扩大和生成)。用Δ_{in}表示进入窗口的位置记录集合,Δ_{out}表示离开窗口的位置记录集合,W_{curr}表示当前窗口中的位置记录集合,W_{prev}表示上一个窗口中的位置记录集合。为了便于表述,本文将离开窗口的核心点和进入窗口的核心点分别定义为前核和新核[28]。

定义7(前核) 给定上一个窗口中的核心点p,如果p在当前窗口中消失($p\in\Delta_{\text{out}}$)或变为非核心点($p\in W_{\text{prev}}\cap W_{\text{curr}}$),则称$p$为前核,记为ex_core。

定义8(新核) 给定当前窗口中的核心点p,如果p未出现在上一个窗口($p\in\Delta_{\text{in}}$)或为非核心点($p\in W_{\text{prev}}\cap W_{\text{curr}}$),则称$p$为新核,记为neo_core。

前核和新核是影响簇状态的关键因素。由于窗口中变化的位置记录p,改变了Δ_{in}和Δ_{out}所有ε邻域内位置记录的$n_\varepsilon(p)$,促使前核和新核改变了$l(p)$。因而,在收集阶段,算法对Δ_{in}和Δ_{out}中的每个位置记录p逐个扫描,重新计算$N_\epsilon(p)$中每个位置记录q的$n_\varepsilon(q)$值并更新类别标签$l(q)$。在收集步骤结束时,当前窗口中的每个数据点都将有一个最新的n_ε值。最后,根据更新后的类别标签值计算前核集合和新核集合。为了提高ε邻域内位置记录的搜索效率,使用R树作为空间索引来维护窗口内保存的位置记录,当轨迹数据进入或离开窗口时,需要在R树中插入或删除结点。伪代码如算法1所示。

算法1 对簇状态产生影响的位置记录的收集

输入:进入窗口的轨迹点集Δ_{in},离开窗口的轨迹点集Δ_{out},当前窗口的$N_\epsilon(p)$。

输出:新核集合{neo_cores},前核集合{ex_cores},Δ_{out}中的前核C_{out}。

①$C_{\text{out}}\leftarrow\varnothing$;

②for p in Δ_{out} // 检查离开窗口的每一个轨迹点,如果是核心点,则加入C_{out},否则直接删除

③ if $l(p)=$ core

④ $C_{\text{out}}\leftarrow C_{\text{out}}\cup\{p\}$;

⑤ else delete p from the R-tree index;

⑥ for q in $N_\epsilon(p)$ // 重新计算$N_\epsilon(p)$中每个位置记录q的$n_\varepsilon(q)$值并更新类别标签$l(q)$

⑦ if $l(q)\neq$ deleted

⑧ $n_\epsilon(q)--$;

⑨ $l(p)\leftarrow$deleted, $n_\epsilon(p)\leftarrow 0$;

⑩end for

⑪for p in Δ_{in} // 检查进入窗口的每一个轨迹点,根据更新后的类别标签值计算前核集合和新核集合

⑫ insert p into the R-tree index;

⑬ $l(p)\leftarrow$unclassified, $n_\epsilon(p)\leftarrow 1$;

⑭ for q in $N_\epsilon(p)$

⑮ if $l(q)\neq$deleted

⑯ $n_\epsilon(q)++$; $n_\epsilon(p)++$;

⑰ end for

⑱end for

⑲compute the sets {ex_cores} and {neo_cores};

⑳return ({ex_cores}, {neo_cores}, C_{out})

3.2.2 聚类

收集阶段得到了前核和新核,进一步计算它们对簇的影响范围和影响结果,能够加快簇演化的处理。前核会造成簇的缩小、分裂和消散。窗口中核心点的消失必然会导致簇的消减。除了前核本身对簇的影响,前核邻域内的位置记录也会对簇产生影响。具体而言,如果某前核邻域内的核心点因失去这一邻点而成为边界点或噪声点,则会出现更多的前核心点,若簇中所有核心点均因此成为前核心点,则该簇将会消散;如果簇中仅部分核心点转变为前核心点,则这些前核心点会切断核心点之间的密度可达路径,当簇中两个核心点之间的密度可达路径被完全切断,则会发生簇的分裂,这是判断簇分裂的关键。

事实上,通过检查窗口内的所有前核以及前核之间的可达关系能够加速簇的分裂。由于前核对簇的影响范围是局部的,通过估计前核的影响边界能够最小化前核的范围搜索空间,优化算法效率。考虑到前核之间的密度可达关系,前核是簇对应图的子图,这个子图对簇中剩余部分连通性的影响可以一起处理。下面对前核的可达关系和前核的影响边界给出两个定义。

定义9(再可达) 给定一对前核$(p,q)\in W_{prev}$,如果p直接密度可达q,则p直接再可达q。一般来说,如果在p和q之间存在一条直接再可达的前核链,那么p再可达q,记为$p \overset{-}{\rightarrow} q$。

对于一个前核p,p再可达的前核集合记作$R^-(p)$,表示为$R^-(p)=\{q\in W_{prev} \mid p \overset{-}{\rightarrow} q\}$。

定义10(前核最小外接核心点集) 给定一个前核p以及$R^-(p)$,与$R^-(p)$的某个前核直接密度可达的、且在W_{curr}和W_{prev}中都为核心点的位置记录集合被称为$R^-(p)$的前核最小外接核心点集$M^-(p)$,表示为$M^-(p)=\{q\mid((q\in W_{prev}\cap W_{curr})\wedge q\text{为核心})\wedge\text{对于部分}r\in R^-(p), q\in N_\epsilon(r)\}$。

前核p的最小外接核心点集$M^-(p)$能够确定由它们引起的簇演化的类型。如果$M^-(p)$对应的子图连通,则其余位置记录仍然在同一个簇中,原始簇缩小;如果$M^-(p)$对应的子图变为多个连通分量,则其余位置记录对应分解为多个簇;如果$M^-(p)$为空,则原始簇中的核心点都不存在,簇消失。

相似地,新核带来簇的扩大、合并和生成。窗口中新核的出现造成簇的扩大。同时,新核的出现改变其邻域内轨迹位置p的$n_\epsilon(p)$值。如果簇中的非核心点p因$n_\epsilon(p)$值增长变为核心点,可能带来新簇的生成;如果新出现的核心点建立了两个簇之间的密度可达路径,则带来簇的生成。检查窗口内的所有新核以及新核之间的可达关系能够加速簇的生成。本文将新核之间的可达关系定义为新生可达,在此基础上定义新核的最小外接核心点集来表达新核对簇的影响边界,从而避免对所有新核的遍历,加速簇状态的更新过程。

定义 11(新生可达) 给定一对新核$(p,q)\in W_{\text{curr}}$,如果p直接密度可达q,则p直接新生可达q。一般来说,如果在p和q之间存在一条直接新生可达的新核链,那么p新生可达q,记为$p\overset{+}{\rightarrow}q$。

对于一个新核p,p新生可达的新核集合记作$R^+(p)$,表示为$R^+(p)=\{q\in W_{\text{curr}}\mid p\overset{+}{\rightarrow}q\}$。

定义 12(新核最小外接核心点集) 给定一个新核p以及$R^+(p)$,与$R^+(p)$的某个新核直接密度可达的、且在W_{curr}和W_{prev}中都为核心点的位置记录集合被称为$R^+(p)$的新核最小外接核心点集$M^+(p)$,表示为$M^+(p)=\{q\mid((q\in W_{\text{prev}}\cap W_{\text{curr}})\wedge q\text{为核心})\wedge\text{对于部分}r\in R^+(p),q\in N_\varepsilon(r)\}$。

新核p的最小外接核心点集$M^+(p)$也能够确定由它们引起的簇演化的类型。如果$M^+(p)$为空,则一个仅有$R^+(p)$中的新核构成的新簇将会出现;如果$M^+(p)$中的所有核心都属于同一个簇,那么$R^+(p)$中的所有新核都将被添加进该簇,进而导致了原始簇的成长。如果$M^+(p)$中的核心分属多个簇$C_1,C_2,\cdots,C_k$,这些核心将会与$R^+(p)$中的新核合为一个单独的簇。聚类阶段实现方式的伪代码如算法2所示。

算法 2 基于滑动窗口的增量式聚类

输入:前核{ex_cores},新核{neo_cores},Δ_{out}中的前核C_{out}

输出:簇{clusters}。

①$E\leftarrow${ex_cores};

②while $E\neq\varnothing$ // 遍历每一个前核

③ Compute $R^-(p)$ and $M^-(p)$ for $p\in E$;

④ $ncc\leftarrow$MS_BFS($M^-(p)$); // 计算前核最小外接核心点集$M^-(p)$连通分量

⑤ if $ncc>1$ //大于1时分裂

⑥ a cluster splits;

⑦ else a cluster shrinks or dissipates

⑧ $E\leftarrow E-R^-(p)$;

⑨end while

⑩Remove C_{out} from the R-tree index;

⑪$N\leftarrow${neo_cores};

⑫for p in N // 遍历每一个新核

⑬ if $M^+(p)$ is disconnected // 如果新核最小外接核心点集$M^+(p)$分属多个簇,则合并

⑭ clusters merge;

⑮ else a cluster grows or emerges

⑯ $N\leftarrow N-R^+(p)$;

⑰end for

⑱return clusters

3.3 基于位运算的模式枚举

聚类操作能够发现满足共同运动模式中群体规模和空间邻近的位置记录集合,而时间连续性需要模式枚举来进一步验证。本文采用共同运动模式的最新研究工作[6]中的模式枚举方法。模式枚举时,首先对每个窗口内的簇枚举所有可能的移动对象进行组合,然后找到每个组合有效的时间序列。具体来说,根据簇的集合初始化所有可能的模式P,其中$|P|\geqslant M$;然后验证每个模式P是否有效,即验证每个模式P的窗口序列W是否满足时间连续性的参数K、L和G。

3.4 算法复杂度分析

时间复杂度:在增量式聚类过程中,正如文献[6]所述,本文算法将轨迹流数据分为若干窗口,然后对每个窗口并行执行DBSCAN,最后收集结果。本文DBSCAN方法的时间复杂度为$O(n)$,其中n为每个窗口中位置发生变化的轨迹点数目,与集中范围连接的$O(n^2)$成本相比更具优势。ICPE方法的时间复杂度为$O(m)$,其中m为每个快照中轨迹点的数量,虽然在数量级上相同,但显然$m>n$。模式枚举过程采用文献[6]中方法,时间复杂度与其相同。

空间复杂度:在增量式聚类过程中,需要保存所有形成簇的轨迹点,并且需要保存$\eta\left(\eta=\left(\left\lceil\frac{K}{L}\right\rceil-1\right)(G-1)+K+L-1\right)$[17]个窗口,空间复杂度为$O(n*\eta)$,其中$n$为形成簇的轨迹点数量。模式枚举过程的空间复杂度为$O\left(n\frac{G+L}{L}\right)$[6]。

4 自适应多级动态数据分发策略

分布式方案中数据倾斜是普遍存在的问题之一。具体来说,轨迹流数据被划分为多个窗口并行计算,由于不同窗口内轨迹点和簇数量的不同,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。这是因为每个共同运动模式的生成需要多个计算单元协同合作,同步完成多个阶段的计算任务。负载不均时,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,从而影响整个系统的挖掘效率。

对负载不均问题的研究,已经取得了一些重要成果[29-32],其解决思路是利用全局索引进行数据分发,控制各计算单元的负载。但是,这些工作只是针对空间文本数据流的查询,并不适用于面向时空轨迹流的共同运动模式挖掘研究中的负载不均问题。现有的分布式共同运动模式研究重点大多在于算法设计,很少关注到负载不均的问题。

除了数据分发阶段会导致负载不均外,由于轨迹流动态变化的特征,负载调整阶段也是影响负载不均的重要环节。数据流有无限性的特性,导致数据流处理任务被长期连续执行。在不同的时间,数据本身的分布偏差变化会影响系统整体负载,静态数据分配策略难以保持节点的平衡负载。因此,需要设计灵活的负载调整策略,同时平衡负载调整本身的代价,适应数据流新的分布特征。

为了解决上述问题,本文研究了共同运动模式分布式挖掘算法中的数据分发方案,其总体架构如图1所示。随着轨迹流数据的到达,首先会根据成本模型计算出每个数据块的负载并对所有数据块组

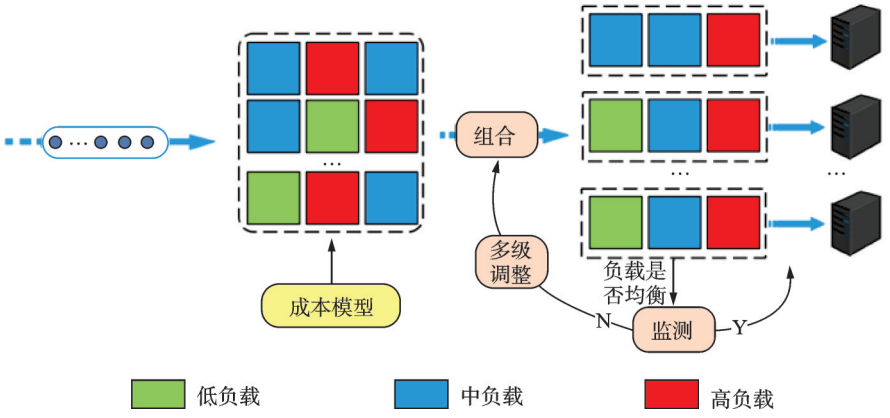


图1 自适应多级动态数据分发策略

Fig.1 Partitioning strategy for adaptive multi-level dynamic data

合分发给各计算节点,使得各计算节点负载相差尽可能少。同时系统会对当前分区方法实时监测,当发生负载不均时根据具体情况调整分区方法。将对数据块的分发操作抽象为数集分区问题,使用Karmarkar-Karp差分算法[33-34]辅助数据划分,保证各个数据分发单位中负载之和的差距尽可能地小。自适应多级动态数据分发策略会实时监测系统的负载情况,根据负载情况做出适当调整。下面将详细介绍成本模型设计方法和负载调整的具体过程。

4.1 数据分发

在分布式流处理系统中,数据倾斜导致的负载不均是普遍存在的问题,但是现有共同运动模式挖掘工作对此考虑略有疏忽。计算节点并行执行多个子任务,节点的整体计算时间由计算量最大的子任务决定。这种方式尽管保证了节点局部聚类的正确性,但未能规避轨迹数据空间分布不均导致节点负载差距过大的问题,抑制了整体算法的性能。

通过分析数据分发的操作细节,本文将共同运动模式挖掘算法中两次数据分发的优化目标抽象为同一个问题。在模式挖掘的分布式处理中,尽管节点(子任务)之间的任务处理是并行的,但是划分至同一子任务的多个数据块将被串行处理,且每个数据块的任务独立进行。基于此,本文将数据块重新组合,同一组合中的所有数据块称为分区。每个分区的数据属于同一子任务,子任务依次处理分区内部的若干个数据块,每个计算节点并行运行子任务。不难发现,数据分发的难点在于保证各个分区中的负载之和差距尽可能地小,因此可以将这两次数据分发过程抽象为共同的数集分区问题。下面给出数集划分问题的定义。

定义13(数集划分问题) 给定正整数集$S=\{s_1,s_2,\cdots,s_y\}$,要划分的子集数量x,目标是将S划分为x个子集$A_1,A_2,\cdots,A_x$,满足x个子集中整数总和的最大值和最小值之差最小,形式化表示为$D(A_1,A_2,\cdots,A_x)=\max\limits_i\left\{\sum\limits_{x\in A_i}x\right\}-\min\limits_i\left\{\sum\limits_{x\in A_i}x\right\}$。其中$s_i$表示某个数据块的负载大小,子集数量$x$代表计算节点个数,子集$A_i$代表分区。

数集划分问题被Richard Karp证明是NP完全问题[35],精确求解的方法需要极高的时间复杂度,难以满足数据流处理的实时要求。为了降低计算代价,本文使用近似算法Karmarkar-Karp差分算法来解决数集划分问题。该算法的思想是将较大数与较小数组合来对冲差距,使得组合之间均匀分布。

Karmarkar-Karp差分算法空间复杂度为$O(n)$,但最糟糕情况下时间复杂度达到$O(2^n)$。由此可见,得到较好的分区组合方式所需的代价极大,而轨迹流数据的动态变化又使得分区组合方式难以一劳永逸。因此,在调整数据分发时,应尽可能减少调整频率或缩小调整范围。

4.2 成本模型

计算成本模型是衡量节点中负载的关键指标。成本模型的设计原则是必须考量在处理过程中影响处理代价的主要因素[29-32]。

4.2.1 聚类的成本模型

共同运动模式挖掘的聚类计算以网格为单位,对于每个开始时刻为t的窗口中的分区$p_i(\text{cell})$,影响计算代价的因素主要与网格中的位置记录数量和总窗口数$|W|$相关。分区$p_i(\text{cell})$的计算代价形式化表示为$\text{Cost}(p_i(\text{cell}))$,则

$$\begin{aligned}\text{Cost}(p_i(\text{cell}))&=|w\cap w'|(|\Delta_{\text{out}}|+|\Delta_{\text{in}}|)(|W|-1)=\\&|w\cap w'|(|w|+|w'|-2|w\cap w'|)(|W|-1)\end{aligned}\tag{1}$$

4.2.2 模式枚举的成本模型

根据成本模型的设计原则,模式枚举过程为对于时刻t的每个分区$P_t(o)$,首先列举所有可能的轨迹点组合,然后找到每个组合的有效时间序列。具体来说,首先初始化所有可能的模式$O\subseteq P_t(o)\cup\{o\}$,其中$|O|\geqslant M$,为简单起见,分区$P_t(o)$上的模式枚举被删除了,因为$o$是公共轨迹。在模式验证阶段,对于每一个模式$O$,首先初始化$T$为$\{t\}$。如果下一次时刻$t'$时$O$也存在于$P_{t'}(o)$中,则$T=T\cup\{t'\}$。从算法过程中可以发现影响处理代价的因素有簇的大小以及M(决定需要枚举的模式数量)、K、L和G(决定需要枚举的窗口数量)的大小。

给定共同运动模式的条件CP(M,K,L,G),时刻t分区$P_t(o)$可能的模式数量为$C_{|P_t(o)|}^{M-1}+C_{|P_t(o)|}^{M}+\cdots+C_{|P_t(o)|}^{|P_t(o)|}$,对于每个可能的模式,需要使用$\eta\left(\eta=\left(\left\lceil\frac{K}{L}\right\rceil-1\right)(G-1)+K+L-1\right)$[17]个窗口验证。时刻$t$分区$P_t(o)$的负载表示为$\mathrm{Cost}(P_t(o))$,则

$$\mathrm{Cost}(P_t(o))=\eta\left(C_{|P_t(o)|}^{M-1}+C_{|P_t(o)|}^{M}+\cdots+C_{|P_t(o)|}^{|P_t(o)|}\right)=\eta*\left(2^{|P_t(o)|}-2^{M-2}\right)\tag{2}$$

4.3 负载调整

在算法运行的初始阶段,优化的数据分发方案能够确保负载均衡。然而,新到达的轨迹流属性分布会随着时间的推移而变化。分配给计算节点的工作量随之改变,这可能会导致负载失衡。为了避免算法整体性能下降,需要重新平衡负载。

在分布式查询研究中,负载迁移是重新平衡负载的重要因素。这是由于查询对象分散存储在各个计算节点中,节点处理查询任务依赖本地数据。当系统负载失衡时,除了需要通过改变指针来调整查询的流向外,还需要调整节点本地的查询对象数据,甚至需要中断系统,带来额外的网络通信和系统开销。分布式查询问题中负载调整需要平衡迁移成本和负载调整后的收益。

聚类和枚举操作无需依赖节点中存储的数据而进行计算,因而分布式共同运动模式挖掘算法中的负载调整无需考虑迁移代价,只需调整数据分发方案,改变轨迹数据的流向,从而均衡负载。但是,Karmarkar-Karp差分算法的时间复杂度较高,频繁调整数据分发策略会带来额外的调整代价。为了应对不同程度的负载失衡,并以最小的调整代价恢复负载均衡,本文采用多级调整策略以重新平衡负载。3个层次的调整策略分别为分区拆分和合并、分区重新分配和分区重构,其调整能力和调整成本依次增加。

本文算法通过定期地收集每个分区的统计信息并计算节点负载来监测负载失衡是否发生。用$w_{\max}$表示负载值最大的节点,$w_{\min}$表示负载值最小的节点,L_w表示节点w的负载值。给定负载比阈值μ,如果$L_{w_{\max}}/L_{w_{\min}}>\mu$,则节点间出现了负载失衡,此时需要进行负载调整。负载调整应满足以下两个要求:(1)负载调整后,节点间能恢复负载均衡;(2)调整成本小,能够高效地进行负载调整。

为了满足这些要求,本文针对不同的负载失衡程度自适应地采用调整策略。以聚类操作为例,用$p_{\max}$表示最大负载值节点$w_{\max}$中的最大分区数据,$L_{p_{\max}}$为其负载值。用$g_{\max}$表示所有节点中负载值最大的网格,$L_{g_{\max}}$为其负载值。用$L_\Delta=L_{w_{\max}}-L_{w_{\min}}$表示负载失衡的程度。三级负载调整概述如下:

(1)当节点间出现负载失衡时,并且$L_\Delta<2L_{p_{\max}}$,调整方式为分区拆分和合并,由于$L_\Delta<2L_{p_{\max}}$,只需将$w_{\max}$中$p_{\max}$的部分负载转移到$w_{\min}$来调整负载。

(2)当$L_\Delta>2L_{p_{\max}}$时,采用分区再分配调整。此时,仅拆分一个分区无法保证负载均衡,因此需要对更多的分区重新分配来调整负载。

(3)当网格严重过载、甚至超过某个节点的负载时,譬如$L_{g_{\max}}>\mu L_{w_{\min}}$,以上两种策略都无法恢复节

点间的负载均衡,需要再次对当前窗口中的所有位置记录使用Karmarkar-Karp差分算法构建分区,优化分发策略。

在大多数情况下,前两种方式可以满足平衡负载的要求,第3种策略则针对极端失衡情况。此外,这3种方法都不需要中断系统。在操作过程中,不会涉及任何本地数据,只需更改最新窗口中数据块对应的分区指针。下文将详细介绍分区拆分和合并、分区再分配和分区重构。

分区拆分和合并策略通过拆分负载大的分区和合并负载小的分区来恢复负载平衡。当$L_{\Delta}<2L_{p_{\max}}$时,分区数据$p_{\max}$被分为两个新的分区,每个分区的负载接近$L_{\Delta}/2$,并在新到达的窗口中将其中一个新分区数据指定给$w_{\min}$。调整后两个节点的整体计算时间趋于接近。重复以上操作,直到节点负载恢复平衡。此外,遍历所有分区,合并负载同时小于$L_{\Delta}/2$的相邻分区的位置记录数据。不难发现,拆分和合并操作的调整代价很低,在调整过程中只简单地改变部分轨迹数据的指向。

分区再分配策略通过对部分节点中的分区数据运行Karmarkar-Karp差分算法来恢复负载平衡。当$L_{\Delta}>2L_{p_{\max}}$时,至少有$L_{p_{\max}}$的负载对应的位置记录需要转移,只分割一个分区数据不能保证恢复平衡。此时,首先对$w_{\max}$和$w_{\min}$中的分区使用Karmarkar-Karp差分算法重分配。由于处理的数据规模较小,这一阶段Karmarkar-Karp差分算法的时间开销也较少。如果分区重新分配后两个节点的负载仍然不平衡,就对其他的部分分区采用拆分和合并策略,然后重新分区再分配,直至负载均衡。虽然分区再分配的调整方法对部分分区数据使用了Karmarkar-Karp算法,但是控制了数据规模,即使在最糟糕的情况下,其调整代价和调整能力也具有性价比。

分区重构策略通过对当前窗口中的所有位置记录重新计算最优的数据分发方法来恢复节点间的平衡。在大多数情况下,如果网格大小合适,则分区调整时数据块的大小也合适,前两种策略可以满足负载调整的要求。但不排除极端情况下,若干网格的负载大于其他计算节点,或者轨迹流数据的空间分布急剧变化,以至于网格大小不再适合。此时,必须根据位置记录新的分布统计情况来调整网格的大小,并再次使用Karmarkar-Karp差分算法来计算数据分发方法。此时,负载调整的调整代价较大。

总而言之,节点的负载情况将被定期监测。一旦发现负载失衡,不同的调整策略将根据整体负载的不同自适应地执行。即使出现极端情况,算法也有应对的解决方案。

5 实验与分析

5.1 实验条件

5.1.1 实验环境

实验的硬件配置为由21个节点组成的集群,其中1个节点作为管理节点,其余的节点作为计算节点。管理节点为Intel(R) Xeon(R) CPU E5645处理器,主频2.4 GHz,六核,内存容量为24 GB。20个计算节点为Intel(R) Xeon(R) CPU E5620处理器,主频2.4 GHz,四核,内存容量为16 GB。所有节点的操作系统为Centos7.8,并配置jdk1.8,Hadoop3.1.3,Zookeeper3.5.7。Flink作为一个真正的流引擎,与其他流处理平台如Storm、Spark Streaming相比延迟低,吞吐量高,本文选择Flink1.12.0作为实验平台。实验程序采用Java程序设计语言进行编写,并通过Maven3.6.1进行打包上传到Flink集群。

5.1.2 数据集

(1)GeoLife[36-37]:该数据集保存了182名用户在2008年4月—2012年8月的旅行记录。对于每个用户,定期收集GPS信息。

(2)北京市高德出租车数据集(Taxi):该数据集统计了北京市12 697辆出租车在2012年11月1日—2012年11月30日产生的时空轨迹数据。

对两个数据集都进行预处理,首先对移动对象进行重新编号,使编号连续并由1开始,并使用线性插值对缺失数据进行填充。对两个真实数据集预处理后的结果如表2所示。由于GeoLife数据集较稀疏,本文利用其进行有效性实验;Taxi数据规模较大,本文将利用Taxi数据集进行效率实验和可伸缩性实验。

表2 数据详情

Table 2 Data details

数据集	轨迹数	轨迹点数
GeoLife	18 670	44 189 853
Taxi	121 468	248 284 500

5.1.3 参数设置

为了全面评估本文算法对共同运动模式的发现能力及挖掘框架的性能,对设置的各个参数进行了实验。表3列出了所有需要评估的参数,其中粗体为默认值。

表3 参数及默认值

Table 3 Parameters and default values

参数	含义	参数值
M/个	移动对象最小规模	2,4,**6**,8,10
K/min	最小持续时长	3,6,**9**,12,15
L/min	最小局部连续时长	2,3,**4**,5,6
G/min	片段间最大间隔时长	2,3,**4**,5,6
(θ/η)/%	窗口滑动步长/窗口大小	16,33,**50**,66,83
ε/m	邻域半径	100,150,**200**,250,300
minPts/个	核心点邻域半径内样本数	5,10,**15**,20,25
O_r/%	数据集规模	20,40,**60**,80,100
N/个	计算节点数	1,5,**10**,15,20

5.2 结果对比及分析

由于本文提出的分布式算法涉及多处优化,为了便于观察,本文在以下实验对比与分析中对所使用到的算法进行简化表示,第3节中描述方法简称为SWA,第4节中方法简称为SWA-LB。本文使用的比较方法为ICPE[6]和SPARE[17],其中ICPE是首个在轨迹流上进行实时分布式共同运动模式挖掘的研究,是目前最先进的方法,与本文研究内容最相似;SPARE是在历史轨迹上最先进的分布式共同运动模式挖掘算法,原算法在Spark平台中实现,本文将其扩展到Flink中。

为了评估SWA和SWA-LB的有效性,本文分别比较了SWA和SWA-LB的结果集与ICPE的结果集,有效率定义为

$$F=\frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}} \tag{3}$$

式中:precision为查准率;recall为召回率,该指标广泛应用于数据挖掘问题中[1]。

算法效率通过比较算法总执行时间和延时说明,其中延时指平均窗口(或快照)的处理时间。具体计算方式为

$$\text{延时}=\frac{\text{模式挖掘总时间}}{\text{总窗口数(或总快照数)}} \tag{4}$$

5.2.1 有效性实验

表4展示了在数据集GeoLife上SWA和SWA-LB有效性随数据集规模改变的变化情况。显然,随着数据集规模的增大,SWA和SWA-LB

表4 SWA和SWA-LB算法有效性对比

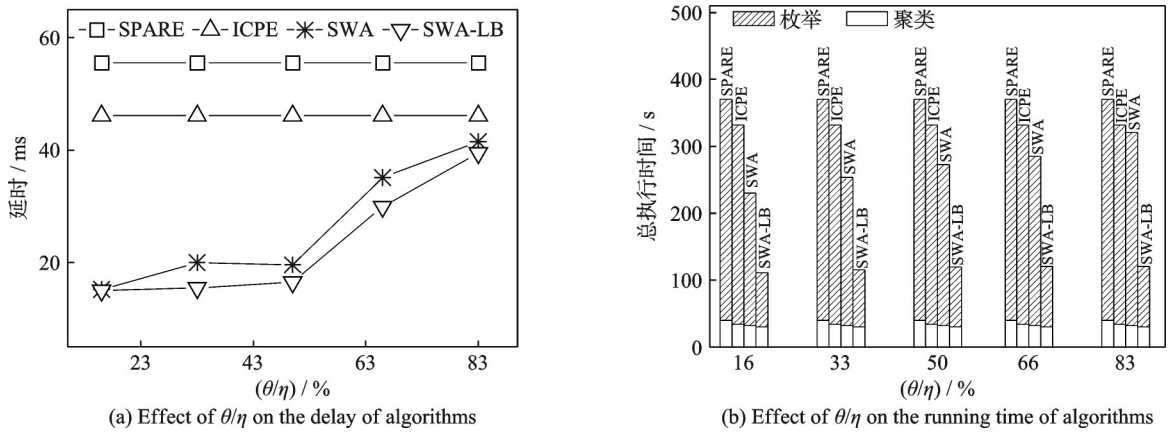
Table 4 Effectiveness comparison of SWA and SWA-LB algorithms %

数据集规模	20	40	60	80	100
SWA	98.1	96.2	95.5	95.0	93.3
SWA-LB	98.2	96.1	97.2	95.3	94.5

的有效性能够保持稳定。由于滑动窗口的计算模型在计算时对位置记录的时间对齐进行了一定的宽松，造成连续时间的长度略有减小。实验结果显示的差错率在可接受的范围内，有效率依然保持在93%以上。

5.2.2 效率实验

图2给出了窗口滑动步长/窗口大小(θ/η)变化对挖掘算法效率的影响，由于SPARE和ICPE算法不使用窗口，本文默认其$\theta/\eta=1$。不难发现，对于SWA算法和SWA-LB算法而言，当θ/η值越接近1时，算法效率越接近ICPE结果。同时，θ/η越小时效率会有所提升，但当其低于一定比例时，对算法效率提升效果降低。对于Taxi数据集而言，当$\theta/\eta=50\%$时算法效率达到最高。

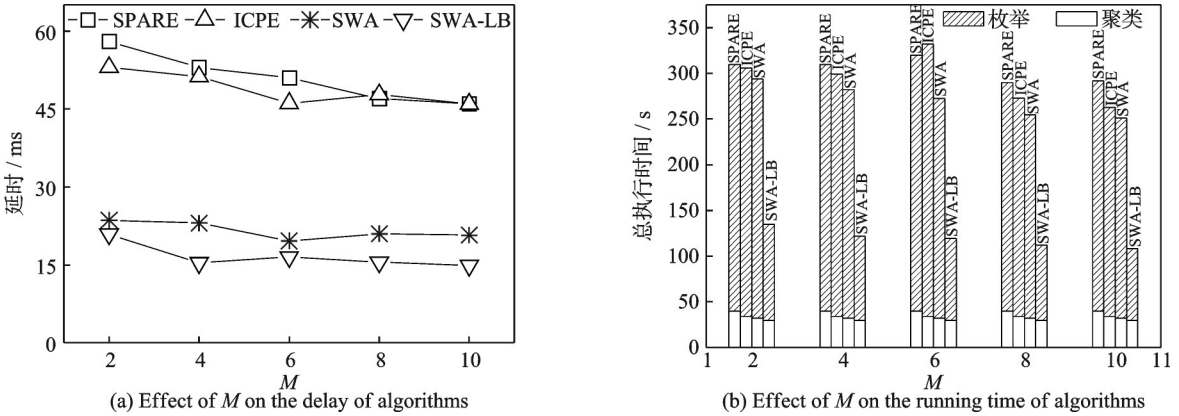


(a) Effect of θ/η on the delay of algorithms　(b) Effect of θ/η on the running time of algorithms

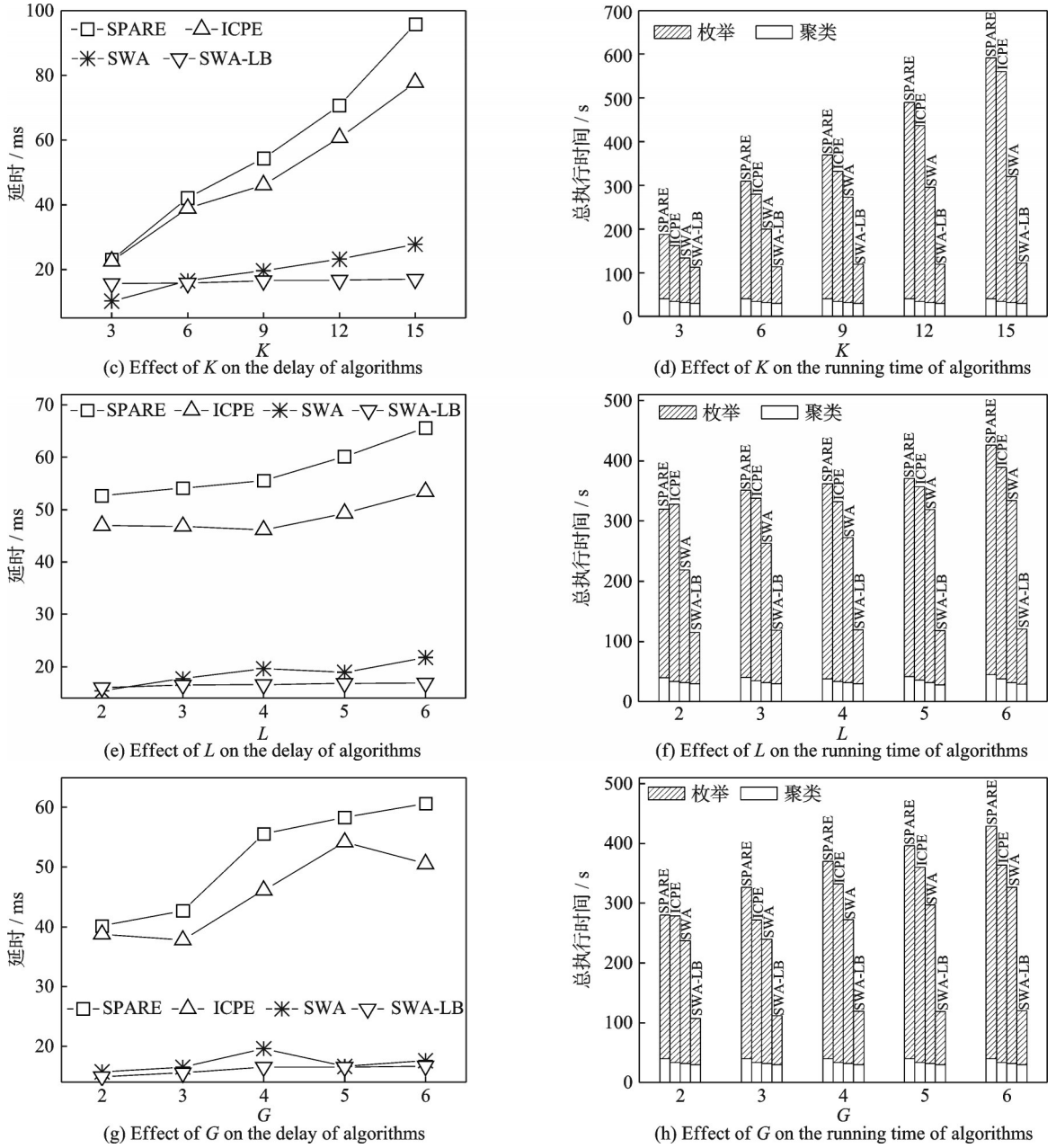
图2 θ/η值变化对算法效率的影响

Fig.2 Effect of varying θ/η value on the efficiency of algorithms

图3给出了共同运动模式挖掘中4个参数M、K、L、G的变化对模式发现效率的影响。图3(a,b)给出了SWA和SWA-LB对共同运动模式发现效率随M值的变化。不难发现，随着M的增加，SPARE、ICPE、SWA和SWA-LB的延时都有不同程度的减低，这是因为M值越大，满足要求的簇数量就越少，模式枚举阶段中需要枚举的移动对象数量就越少。在Taxi中SWA和SWA-LB都有较好的性能表现，相比于SPARE和ICPE方法延时降低了60%以上。图3(c～h)分别给出了SWA和SWA-LB对共同运动模式发现效率随参数K、L和G值的变化。随着K、L和G的增加，算法的总执行时间和延时也在增加，这是因为上述3个参数都会影响在枚举过程中需要检查的窗口(或快照)数量，



(a) Effect of M on the delay of algorithms　(b) Effect of M on the running time of algorithms



(c) Effect of K on the delay of algorithms

(d) Effect of K on the running time of algorithms

(e) Effect of L on the delay of algorithms

(f) Effect of L on the running time of algorithms

(g) Effect of G on the delay of algorithms

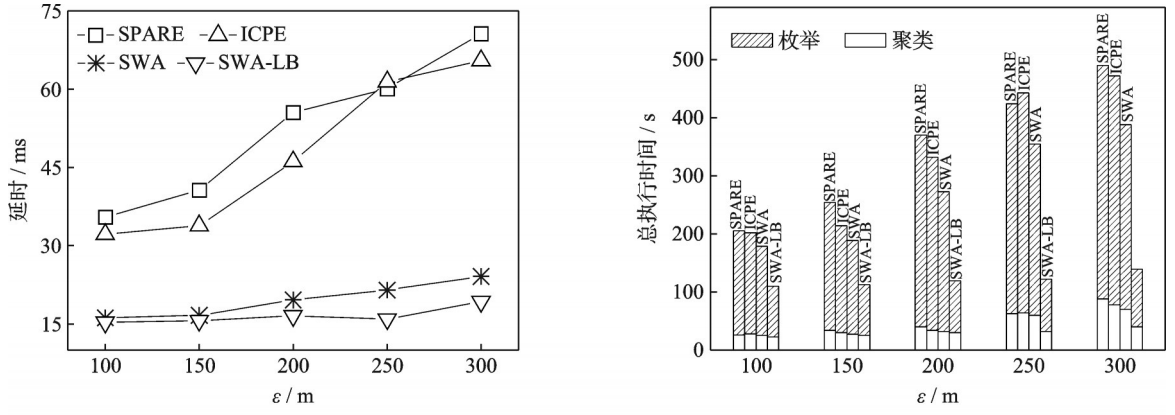
(h) Effect of G on the running time of algorithms

图3 M、K、L和G值变化对算法效率的影响

Fig.3 Effect of varying M, K, L, and G values on the efficiency of algorithms

值越大需要检查的窗口(或快照)数量越多。实验结果显示,本文算法极大地提高了共同运动模式挖掘算法效率,表明本文提出的滑动窗口策略和动态数据分发策略对于分布式共同运动模式挖掘效率的改进显著。

图4给出了邻域半径ε在从100～300 m变化时的算法性能表现。随着ε的增加,所有方法的总执行时间都在增加,这是因为搜索的邻域半径越大,包含的移动对象就越多,在聚类和模式枚举阶段中都需要更多的计算量。不难看出,SWA和SWA-LB的性能要强于SPARE和ICPE。

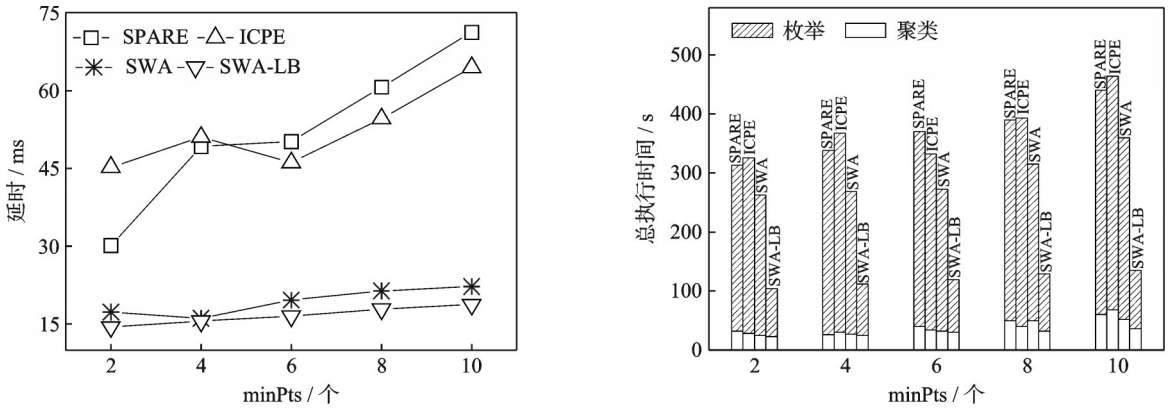


(a) Effect of ε on the delay of algorithms (b) Effect of ε on the running time of algorithms

图4 ε值变化对算法效率的影响

Fig.4 Effect of varying ε value on the efficiency of algorithms

图5给出了minPts变化对算法效率的影响,可以明显看出随着minPts的增加,SWA和SWA-LB的总执行时间在增加,但明显低于SPARE和ICPE方法。这是因为minPts越大聚类阶段得到的簇就越大,模式枚举阶段需要枚举的子集越多,SWA在聚类阶段较ICPE节省了时间,SWA-LB充分发挥了计算机集群的资源,节省了更多的时间。



(a) Effect of minPts on the delay of algorithms (b) Effect of minPts on the running time of algorithms

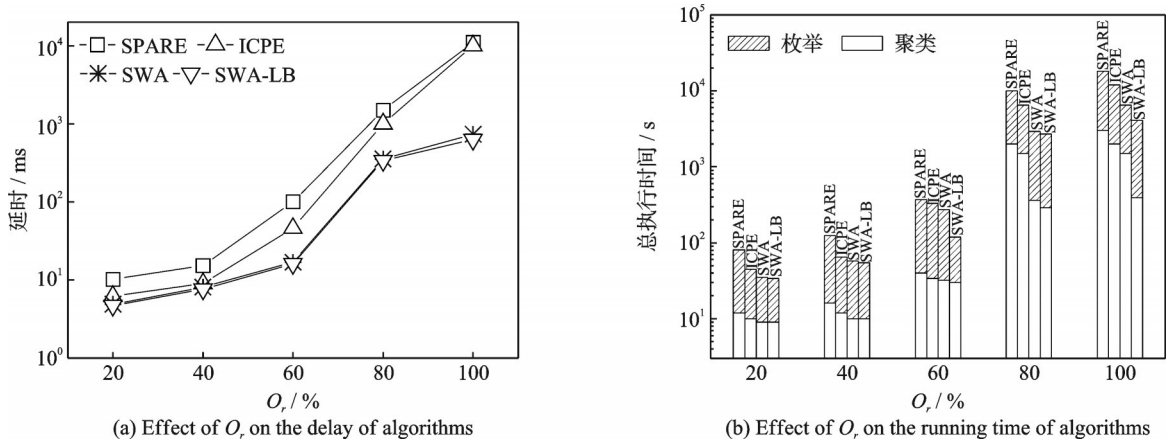
图5 minPts值变化对算法效率的影响

Fig.5 Effect of varying minPts value on the efficiency of algorithms

5.2.3 可伸缩性实验

可伸缩性实验通过改变数据集规模和计算节点数量观察算法性能。图6给出了移动对象比例O_r从20%到100%SWA和SWA-LB的总执行时间变化,可以看出,当移动对象比例O_r超过60%时ICPE的执行时间急剧上升,这是因为模式枚举的时间复杂度是$O(2^n)$。而SWA在聚类时节省部分时间,因此算法性能要优于ICPE,SWA-LB充分利用了分布式集群所有计算节点的资源,具有较好的性能和可伸缩性。

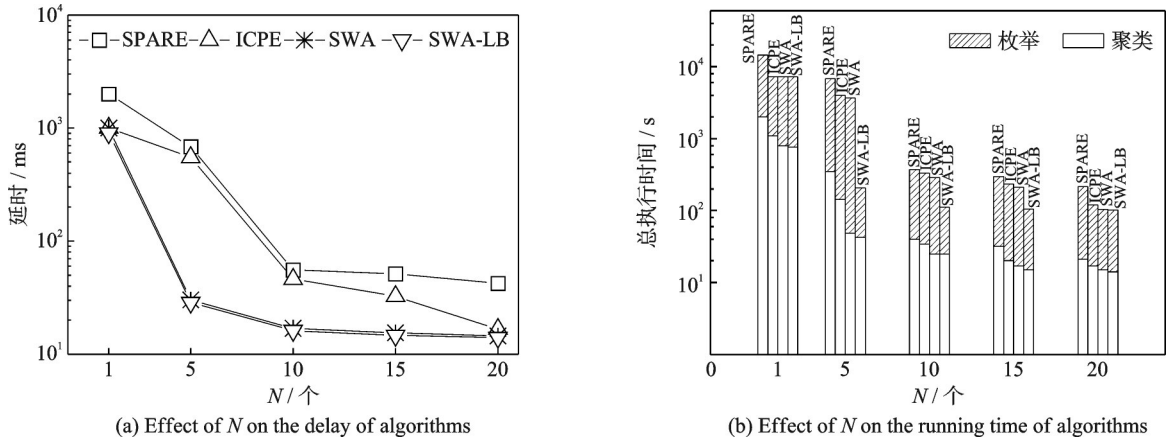
图7给出了随着计算节点数量改变3个方法的执行时间变化。可以看出,在单机环境下(计算节点数量为1),本文算法与基准方法执行效率相当。随着计算节点数量增加,算法总执行时间明显减少,本文算法展现出比ICPE更好的性能。不难发现,SWA算法在聚类阶段的执行效率有较大的提升,SWA-LB在聚类和枚举阶段都有较好的提升,这是因为SWA算法主要改进的是聚类阶段,SWA-LB对于两阶段都进行了改进,这也再次证明了本文算法的有效性。



(a) Effect of O_r on the delay of algorithms (b) Effect of O_r on the running time of algorithms

图6 O_r值变化对算法效率的影响

Fig.6 Effect of varying O_r value on the efficiency of algorithms



(a) Effect of N on the delay of algorithms (b) Effect of N on the running time of algorithms

图7 N值变化对算法效率的影响

Fig.7 Effect of varying N value on the efficiency of algorithms

6 结束语

本文主要研究面向时空轨迹流的共同运动模式分布式挖掘算法,基于现有挖掘框架,主要提出两点改进。首先,在数据处理的聚类阶段采用了基于滑动窗口的共同运动模式分布式挖掘算法,摒弃了常用的快照计算模型,利用增量式聚类代替重新聚类,更适应采样频繁的轨迹流数据;其次,针对分布式方案中不可回避的负载均衡问题提出了自适应多级动态数据分发策略,该策略能够实时监测系统负载情况,并在发生负载不均时做出适当调整,充分利用各计算节点资源。实验结果证明,SWA和SWA-LB都具有较高的算法效率和可伸缩性。尽管本文对云环境中面向时空轨迹流的共同运动模式挖掘算法进行了详细的研究与实现,并取得了一些进展,但在算法和系统中仍存在有待改进和优化的空间:(1)本文SWA算法主要优化了其中的聚类操作,但枚举操作仍然占据了大部分的耗时,下一步工作可以关注挖掘算法中枚举操作的性能优化,更及时地发现共同运动模式;(2)本文提出的SWA-LB算法仍存在负载调整粒度粗、自我调节能力差等问题,下一步工作可以利用轨迹流中的周期性规律,利用机器学习的技术建立负载预测模型,实现更智能的负载均衡策略。

参 考 文 献：

[1] KALNIS P, MAMOULIS N, BAKIRAS S. On discovering moving clusters in spatio-temporal data[C]//Proceedings of the 9th International Symposium on Spatial and Temporal Databases. Berlin: Springer, 2005: 364-381.

[2] JEUNG H, YIU M L, ZHOU X, et al. Discovery of convoys in trajectory databases[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 1068-1080.

[3] LI Z, DING B, HAN J, et al. Swarm: Mining relaxed temporal moving object clusters[J]. Proceedings of the VLDB Endowment, 2010, 3(1): 723-734.

[4] LI Y, BAILEY J, KULIK L. Efficient mining of platoon patterns in trajectory databases[J]. Data & Knowledge Engineering, 2015, 100: 167-187.

[5] TANG L A, ZHENG Y, YUAN J, et al. On discovery of traveling companions from streaming trajectories[C]//Proceedings of the 28th IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE, 2012: 186-197.

[6] CHEN L, GAO Y, FANG Z, et al. Real-time distributed co-movement pattern detection on streaming trajectories[J]. Proceedings of the VLDB Endowment, 2019, 12(10): 1208-1220.

[7] FANG Z, GAO Y, PAN L, et al. CoMing: A real-time co-movement mining system for streaming trajectories[C]//Proceedings of the 2020 International Conference on Management of Data. New York: ACM, 2020: 2777-2780.

[8] TRITSAROLIS A, KONTOULIS Y, PELEKIS N, et al. MaSEC: Discovering anchorages and co-movement patterns on streaming vessel trajectories[C]//Proceedings of the 17th International Symposium on Spatial and Temporal Databases. Berlin: Springer, 2021: 170-173.

[9] 朱美玲, 刘晨, 王雄斌, 等. 基于车牌识别流数据的车辆伴随模式发现方法[J]. 软件学报, 2017, 28(6): 1498-1515.
ZHU Meiling, LIU Chen, WANG Xiongbin, et al. Approach to discover companion pattern based on anpr data stream[J]. Journal of Software, 2017, 28(6):1498-1515.

[10] 于自强, 禹晓辉, 董吉文, 等. 分布式多数据流频繁伴随模式挖掘[J]. 软件学报, 2019, 30(4): 1078-1093.
YU Ziqiang, YU Xiaohui, DONG Jiwen, et al. Distributed mining of frequent co-occurrence patterns across multiple data streams[J]. Journal of Software, 2019, 30(4): 1078-1093.

[11] VIEIRA M R, BAKALOV P, TSOTRAS V J. On-line discovery of flock patterns in spatio-temporal data[C]//Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York: ACM, 2009: 286-295.

[12] BENKERT M, GUDMUNDSSON J, HÜBNER F, et al. Reporting flock patterns[C]//Proceedings of the 14th European Symposium on Algorithms. Berlin: Springer, 2006: 660-671.

[13] YEOMAN J, DUCKHAM M. Decentralized detection and monitoring of convoy patterns[J]. International Journal of Geographical Information Science, 2016, 30(5): 993-1011.

[14] ORAKZAI F, CALDERS T, PEDERSEN T B. *k*/2-hop: Fast mining of convoy patterns with effective pruning[J]. Proceedings of the VLDB Endowment, 2019, 12(9): 948-960.

[15] HELMI S, KASHANI F B. Multiscale frequent co-movement pattern mining[C]//Proceedings of the 36th IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE, 2020: 829-840.

[16] ORAKZAI F, CALDERS T, PEDERSEN T B. Distributed convoy pattern mining[C]//Proceedings of the 17th IEEE International Conference on Mobile Data Management. Piscataway, NJ: IEEE, 2016: 122-131.

[17] FAN Q, ZHANG D, WU H, et al. A general and parallel platform for mining co-movement patterns over large-scale trajectories[J]. Proceedings of the VLDB Endowment, 2016, 10(4): 313-324.

[18] 张敬伟, 刘绍建, 杨青, 等. DMFUCP:大规模轨迹数据通用伴随模式分布式挖掘框架[J]. 计算机研究与发展, 2022, 59(3): 647-660.
ZHANG Jingwei, LIU Shaojian, YANG Qing. DMFUCP: A distributed mi-ning framework for universal companion patterns on large-scale trajectory data[J]. Journal of Computer Research and Development, 2022, 59(3): 647-660.

[19] TRITSAROLIS A, CHONDRODIMA E, TAMPAKIS P, et al. Predicting co-movement patterns in mobility data[J]. GeoInformatica, 2022,28(2): 1-23.

[20] BHUSHAN A, BELLUR U, SHARMA K, et al. Mining swarm patterns in sliding windows over moving object data streams [C]//Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York: ACM, 2017: 60:1-60:4.

[21] DING J, FANG J, CHAO P, et al. A distributed framework for online stream data clustering[C]//Proceedings of International

Conference on Algorithms and Architectures for Parallel Processing. Cham: Springer International Publishing, 2020: 190-204.
[22] LI R, WANG R, LIU J, et al. Distributed spatio-temporal *k* nearest neighbors join[C]//Proceedings of the 29th International Conference on Advances in Geographic Information Systems. New York: ACM, 2021: 435-445.
[23] QIAN T, SUN S, SHAN X, et al. Distributed-swarm: A real-time pattern detection model based on density clustering[J]. IEEE Access, 2022, 10: 59832-59842.
[24] GAO Y, FANG Z, XU J, et al. An efficient and distributed framework for real-time trajectory stream clustering[J]. IEEE Transactions on Knowledge and Data Engineering, 2023,36(5): 1-17.
[25] GU J, WANG J, ZANIOLO C. Ranking support for matched patterns over complex event streams: The CEPR system[C]//Proceedings of the 32nd IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE, 2016: 1354-1357.
[26] YANG D, GUO Z, XIE Z, et al. Interactive visual exploration of neighbor-based patterns in data streams[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2010: 1151-1154.
[27] YU Z, YU X, LIU Y, et al. Mining frequent co-occurrence patterns across multiple data streams[C]//Proceedings of the 18th International Conference on Extending Database Technology. Berlin: Springer, 2015: 73-84.
[28] KIM B, KOO K, KIM J, et al. DISC: Density-based incremental clustering by striding over streaming data[C]//Proceedings of the 37th IEEE International Conference on Data Engineering. Piscataway. NJ: IEEE, 2021: 828-839.
[29] YANG Z, ZHENG B, TONG C, et al. HASTE: A distributed system for hybrid and adaptive processing on streaming spatial-textual data[C]//Proceedings of the 30th ACM International Conference on Information and Knowledge Management. New York: ACM, 2021: 2363-2372.
[30] MAHMOOD A R, DAGHISTANI A, ALY A M, et al. Adaptive processing of spatial-keyword data over a distributed streaming cluster[C]//Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York: ACM, 2018: 219-228.
[31] CHEN Z, CONG G, ZHANG Z, et al. Distributed publish/subscribe query processing on the spatio-textual data stream[C]//Proceedings of the 33rd IEEE International Conference on Data Engineering. Piscataway, NJ: IEEE, 2017: 1095-1106.
[32] CHEN Y, CHEN Z, CONG G, et al. SSTD: A distributed system on streaming spatio-textual data[J]. Proceedings of the VLDB Endowment, 2020, 13(11): 2284-2296.
[33] KARMARKAR N, KARP R M. An efficient approximation scheme for the one-dimensional bin-packing problem[C]//Proceedings of the 23rd Symposium on Foundations of Computer Science. Piscataway, NJ: IEEE, 1982: 312-320.
[34] MITCHELL J E, TODD M J. Solving combinatorial optimization problems using Karmarkar's algorithm[J]. Mathematical Programming, 1992, 56: 245-284.
[35] KARP R M. Reducibility among combinatorial problems[C]//Proceedings of a Symposium on the Complexity of Computer Computations. Piscataway, NJ: IEEE, 1972: 85-103.
[36] ZHENG Yu, CHEN Yukun, XIE Xing, et al. Understanding mobility based on GPS data[C]//Proceedings of ACM Conference on Ubiquitous Computing. New York: ACM, 2008: 312-321.
[37] ZHENG Yu, XIE Xing, MA Weiying. GeoLife: A collaborative social networking service among user, location and trajectory [J]. IEEE Data Engineering Bulletin, 2010, 33(2): 32-39.

作者简介:

余舒鹏(1999-),男,硕士研究生,研究方向:大数据、数据挖掘,E-mail:yushupeng0226@163.com。

吴春雨(1998-),女,硕士研究生,研究方向:大数据、数据挖掘,E-mail:wcy212202043@163.com。

赵斌(1978-),**通信作者**,男,副教授,研究方向:数据挖掘、数据库及其应用,E-mail: zhaobin@njnu.edu.cn。

吉根林(1964-),男,教授,研究方向:数据挖掘及其应用,E-mail: glji@njnu.edu.cn。

(编辑:刘彦东)