引文格式:李柞霖,向隆刚,余列冰,等.面向大规模空间数据流的分布式连接查询方法[J].武汉大学学报(信息科学版),2025,50(5):1017-1028.DOI:10.13203/j.whugis20230040

Citation:LI Zuolin,XIANG Longgang,YU Liebing,et al.Distributed Join Query Method for Large-Scale Spatial Data Streams[J].Geomatics and Information Science of Wuhan University,2025,50(5):1017-1028.DOI:10.13203/j.whugis20230040

面向大规模空间数据流的分布式连接查询方法

李柞霖[1] 向隆刚[1] 余列冰[1] 吴华意[1] 关雪峰[1]

1 武汉大学测绘遥感信息工程全国重点实验室,湖北 武汉,430079

摘 要:空间连接查询是处理和分析空间数据的基础操作之一。随着空间数据的爆发式增长,针对海量空间数据的连接查询技术备受瞩目。面向大规模历史数据的空间连接查询已被广泛研究,而受限于数据流的高速接入率与连接的实时性需求,目前面向数据流的分布式空间连接查询仍充满挑战。为此,面向大规模空间数据流设计了一种分布式连接查询处理框架。首先形式化定义了针对空间数据流连接查询处理问题,按照参与连接的数据形态细分为"流-表"和"流-流"两类连接,然后分析了分布式连接场景的共性问题,设计了全局网格分区-局部空间索引的两层处理框架,以支持空间流的分布式连接。在此基础上,针对不同连接场景分别设计了适应的连接策略:对于"流-表"连接,提出了基于两级R-tree拓扑关系判断优化算法;对于"流-流"连接,设计了一种顾及分区边界的数据冗余路由算法,以保证分区边界数据的正确连接。此外,针对间隔时间语义的缓存需求,提出了兼顾状态管理与检索效率的BinR-tree结构。大量实验结果表明,所提出的空间数据连接方法具有良好的线性加速比,且相对于基线方法,连接查询效率得到了显著提升。

关键词:空间数据流;空间连接;流连接;分布式;空间索引

中图分类号:P208 **文献标识码**:A **收稿日期**:2023-08-02

DOI:10.13203/j.whugis20230040 **文章编号**:1671-8860(2025)05-1017-12

Distributed Join Query Method for Large-Scale Spatial Data Streams

LI Zuolin[1] *XIANG Longgang*[1] *YU Liebing*[1] *WU Huayi*[1] *GUAN Xuefeng*[1]

1 State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China

Abstract: **Objectives**: Spatial join query is one of the basic operations for processing and analyzing spatial data. With the explosive growth of spatial data, join query technology for massive spatial data has attracted much attention. Although the existing research work has explored the join query of spatial data streams, it is still in its infancy. The processing of join query processing of spatial data streams is still insufficient in terms of systematization and universality. It is urgent to explore how to define general spatial data streams connect query problems and provide systematic solutions. **Methods**: After fully excavating the common real-time spatial data processing problems behind various practical application scenarios, this paper refers to the processing theory and methods of spatial data in the batch processing field, and considers the characteristics of long-term continuous operation of stream processing, and formalizes the problem of spatial data stream connection. Two types of connection operators are defined, including "stream-table" connection and "stream-stream" connection. On this basis, a two-layer data index structure of global grid partition and local spatial index is proposed to support spatial stream supports distributed connection. For the "stream-table" connection, this paper proposes two physical realization methods of spatial dimension tables, and designs a two-level R-tree-based topology relationship judgment optimization for the memory-based storage algorithm. For "stream-stream" connection, We propose a partition boundary data redundancy algorithm to correctly implement cross-partition joins of partition boundary data. In addition, for the caching requirements of interval time semantics, a BinR-tree result

基金项目:湖北珞珈实验室专项基金(220100010)。

第一作者:李柞霖,硕士,研究方向为时空大数据流式计算。lynnlee1229@whu.edu.cn

通信作者:向隆刚,博士,教授。geoxlg@whu.edu.cn

that takes into account both management and retrieval efficiency is proposed. **Results:** (1) The "stream-table" spatial connection implemented in this paper can achieve a throughput of more than 60 000 under a single degree of parallelism, and the average overall delay is about 90 milliseconds. Compared with the native method, the average throughput is 9 times Boost, latency is reduced by an average of 20 milliseconds. (2) Same as the "stream-table" connection, the performance of the "stream-stream" connection method in this paper improves steadily with the increase of parallelism, and has good horizontal scalability. (3) In the experimental comparison between window connection and interval connection, with the window increases, the connection latency increases, much higher than the stream interval connection. This part of the experiment also shows that the stream interval connection without bounded time semantics is more suitable for stream processing logic. (4) Mesh division has little effect on throughput, and its main function is to balance the load and ensure the spatial proximity of data. (5) When the bin size is smaller than the window size, the query will span multiple BinR-tree, which will affect the query efficiency. Properly increasing the bin size will improve the query efficiency; when the bin size is greater than or equal to the time interval, the query efficiency will not be greatly improved. **Conclusions:** A large number of experimental results show that the spatial data connection method proposed in this paper has a good linear speedup ratio, and compared with the baseline method, the connection query efficiency has been significantly improved.

Key words: spatial data streams; spatial join; stream join; distributed; spatial index

随着传感器技术的发展、物联网的建设以及云计算的普及,海量的空间数据以数据流的形式源源不断地传送到数据中心,交通出行、移动社交等越来越多地理信息相关的行业需要对这些数据流进行持续的实时处理,从而为复杂的时空问题提供低延迟的解决方案。近年来,随着空间数据的爆发式增长,对海量数据的查询处理技术备受瞩目[1-2]。空间连接查询是处理和分析空间数据的基础操作之一,在如交通流量分析、区域监控、人车匹配等诸多场景中具有重要应用价值[2-6]。

定义1 空间数据流。空间数据流表示无界的空间数据集合,即空间数据可以持续生成,无限增长,用以描述空间对象的实时状态,可以形式化定义为由空间对象o及其数据生产时间戳t组成的二元组$<o,t>$的集合。

$$S=\{<o_i,t_i>|i=1,2,\cdots\} \tag{1}$$

针对空间数据流的连接查询处理受到了人们越来越多的关注。与针对大规模历史数据表的空间连接查询不同,面向空间数据流的连接查询更具有挑战。传统的数据库大都采用索引嵌套循环连接(indexed nested loop join, INLJ)算法[7]实现两个表的空间连接。然而,空间数据流高速接入且无限增长的特点,使其难以直接使用INLJ算法;在流场景下,连接中某个流中的记录到达时,另一个流中满足连接条件的记录可能尚未到达,因而必须记录数据流的状态,从而在适当的时机触发计算;此外,还必须维护和清理数据流的过期状态,防止其无限增长。

面向大规模历史空间数据的连接查询处理技术伴随着分布式批处理引擎(distributed batch processing engine,DBPE)的不断发展而日趋成熟。依据所依赖的底层计算引擎,基于DBPE的空间连接查询实现主要分为两类。一类是基于MapReduce计算引擎的扩展,如Hadoop-GIS[8]、SpatialHadoop[9]等。其中,Hadoop-GIS是Apache Hadoop之上第一个关于空间数据处理的扩展,它将Apache Hadoop视为黑盒并完全依赖其底层架构,在此基础上提供了基于均匀网格的数据分区策略,在各个分区内支持构建R*-tree索引加速计算;SpatialHadoop对Apache Hadoop的源码进行了扩展,使其从存储和计算两个方面原生支持空间数据,从而获得了比Hadoop-GIS更好的性能。另一类是基于Apache Spark的扩展,如SpatialSpark[10]、GeoSpark[11-12]、Simba[13]、LocationSpark[14]等。SpatialSpark和GeoSpark在Spark之上提供了空间数据分区策略,并支持多种类型的空间查询。而Simba和LocationSpark则更关注对SQL的扩展。现有研究大多支持点、线、矩形以及多边形等多种空间数据类型,并在此基础上支持多种类型的空间连接,如距离连接、拓扑连接、K-最近邻连接等[15-16]。然而,受限于DBPE的执行模式,它们的高吞吐以高延迟为代价。对于持续接入的空间数据,这些框架难以进行低延迟的连接查询处理。

受到历史空间数据连接查询处理的启发,以及分布式流处理引擎(distributed streaming processing engine, DSPE)的日臻成熟,空间数据流

的连接查询处理技术也得到了一定的发展。Zhang等[17]率先基于Apache Storm实现了轨迹数据流的距离连接,这一工作在空间数据流的低延迟连接查询处理方面具有重要的借鉴意义,但其仅支持基于轨迹点距离的连接;MobyDick[18]构建在Apache Flink之上提供了空间流连接支持,但其研究侧重于对空间数据流处理提供类SQL支持,缺少针对性的分区和索引策略;GeoFlink[19-20]同样基于Apache Flink支持空间数据流的距离连接,不过其完全采用有界空间数据的查询语义,在流处理场景下使用范围有限。

尽管现有研究工作已经对空间数据流的连接查询进行了一定的探索,但是仍处于起步阶段,空间数据流连接查询处理在系统性和普适性方面仍存在不足,亟须探索如何定义通用的空间数据流连接查询问题,并提供系统化的解决方案。为此,本文在充分挖掘各类实际应用场景背后共性的即时空间数据处理问题后,借鉴批处理领域空间数据的处理理论与方法,考虑流处理长期持续运行的特点,针对空间数据流连接问题形式化定义了两类连接算子,即“流-表”连接和“流-流”连接;在此基础上,提出了一种全局网格分区及局部空间索引的两层数据索引结构,以支持空间数据流分布式连接,对于“流-表”连接,提出两种空间维度表的物理实现方式,并针对其中基于内存的存储方式设计了一种基于两级R-tree的拓扑关系判断优化算法;对于“流-流”连接,提出一种分区边界数据冗余算法,以正确实现分区边界数据的跨分区连接。此外,针对间隔时间语义的缓存需求,提出了兼顾管理与检索效率的BinR-tree结构。

1　问题定义

本文研究面向空间数据流的连接查询方法,首先给出本文所指空间连接查询的形式化定义。

定义2　空间连接查询。给定空间数据集合O_1、O_2,对于$\forall o_1\in O_1$,找到所有$o_2\in O_2$,满足$r(o_1,o_2)$并输出连接结果$j(o_1,o_2)$。其中,函数r用于判断两个对象之间的空间关系是否符合连接条件,除支持九交模型所定义的所有拓扑关系[21]外,本文增加了距离关系(包括距离包含于、距离包含),相应地,定义r_δ表示考虑距离阈值δ的空间关系判断函数。连接函数j用于对连接对象进行定制化的合并操作。空间连接的形式化定义如下:

$$\mathrm{Join}(\boldsymbol{O}_1,\boldsymbol{O}_2,\delta)=\{j(o_1,o_2)\mid o_1\in\boldsymbol{O}_1,o_2\in\boldsymbol{O}_2,r_\delta(o_1,o_2)=\mathrm{true}\} \tag{2}$$

$$r_\delta(o_1,o_2)=\begin{cases}r(o_1,o_2)\wedge\mathrm{dis}(o_1,o_2)<\delta,\delta>0\\ r(o_1,o_2),\delta\leqslant 0\end{cases} \tag{3}$$

式中,dis函数用于计算空间对象间的距离。

根据参与连接对象O_1、O_2的数据形态分类,本文基于实际应用场景抽象了两类空间数据流连接问题,包括用于空间数据流与空间维度表的“流-表”连接,以及用于“流-流”的空间窗口连接和空间间隔连接。其中,“流-表”连接用于将单空间数据流关联静态空间表信息,例如将用户移动轨迹与行政区划连接以追踪其所经过的行政单元;“流-流”连接适用于连接对象均为空间数据流的场景,例如在交通领域中常需要把行人轨迹数据流与网约车轨迹数据流进行空间连接,从而为行人确定附近的网约车。下面给出相关的形式化定义。

1.1　“流-表”空间连接

维度数据是由独立的、不重叠的数据元素组成的数据集,主要有过滤、分组和标签3个方面的功能[22]。在空间数据流处理中,同样有为流几何关联维度信息的需求。为此,本文将维度数据的概念引入空间数据处理领域,将带有空间属性的维度数据称为空间维度数据,其形式化定义如下:

定义3　空间维度数据。空间维度数据是由独立的、不重叠的且带有空间属性的数据元素组成的数据集。空间维度数据中的单个数据元素称为维度几何。

基于空间维度数据的定义,本文把空间数据流与空间维度表基于拓扑关系的实时连接称为“流-表”空间连接,其形式化定义如下:

定义4　“流-表”空间连接。给定空间数据流S,空间维度表D,对$\forall s\in S$,找到所有$d\in D$,满足$r_\delta(s,d)$并输出连接结果$j(s,d)$。

1.2　“流-流”空间连接

1.2.1　“流-流”空间窗口连接

本文所定义的“流-流”空间窗口连接面向两个高速变化的空间数据流,将落在同一时间窗口内的两个窗口按照指定拓扑关系进行连接,其相关形式化定义如下:

定义5　窗口快照。如图1所示,给定空间数据流S,起始时间T,窗口大小W,窗口滑动ε,S中落在任意时间窗口$[T+i\varepsilon,T+i\varepsilon+W]$范围内的所有空间数据形成空间数据流$S$的一个窗口快照。

定义6　“流-流”空间窗口连接。给定两个空

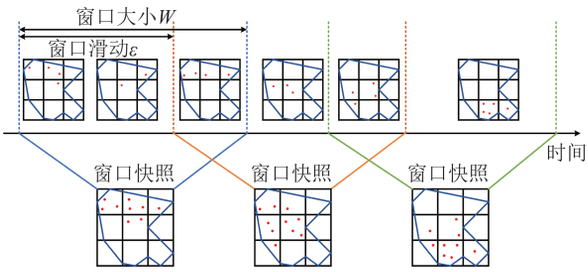

图1 窗口快照示意图

Fig. 1 Window Snapshot

间数据流S_1和S_2，对于分别属于S_1和S_2且落在同一窗口内的窗口快照S_{1w}和S_{2w}，“流-流”空间窗口连接是对$\forall s_1 \in S_{1w}$，找到所有$s_2 \in S_{2w}$，满足$r_\delta(s_1, s_2)$，输出$j(s_1, s_2)$。

1.2.2 “流-流”空间间隔连接

基于窗口的空间流连接只能对当前窗口内数据进行连接，无法处理相对向前、向后时间数据。为此，本文定义“流-流”空间间隔连接以拓展空间流连接的连接时间语义，其形式化定义如下：

定义7 “流-流”空间间隔连接。给定两个空间数据流S_1和S_2、上界（upper bound，UB）时间间隔T_{ub}和下界（lower bound，LB）时间间隔T_{lb}，对$\forall s_1 \in S_1$，$\forall s_2 \in S_2$，设其时间戳分别为t_1、t_2。“流-流”空间间隔连接是对$\forall s_1 \in S_1$，找到所有$s_2 \in S_2$，且$t_1 + T_{lb} \leqslant t_2 \leqslant t_1 + T_{ub}$，满足$r_\delta(s_1, s_2)$，输出$j(s_1, s_2)$。

为方便描述，下文称“流-流”连接中，S_1为空间驱动流，S_2为空间被驱动流。

2 空间数据流连接查询算法

本文研究开展的技术路线如图2所示。首先，对于输入空间数据进行适应分布式连接的索引与管理，对空间流数据建立全局网格分区-局部空间索引两级索引，对空间维表根据其物理存储形式进行对应的管理。在此基础上，对各连接算子进行针对性优化。“流-表”空间连接基于广播表的思想实现，并进一步对拓扑判断进行优化；对于“流-流”空间连接，本文顾及数据分区、状态管理等问题，将嵌套循环连接算法推广至空间流处理。

本文提出的查询方法针对空间数据流和分布式环境的特点进行了分析，设计了适用于空间流连接算子的共用索引与管理框架。在此基础上，针对不同算子进行了适应性的调整与进一步优化。

2.1 空间流索引

面向大规模空间数据的分布式连接查询实现一般包含两层索引。第一层用于全局数据分区，以便在分布式环境下对数据进行划分，这一

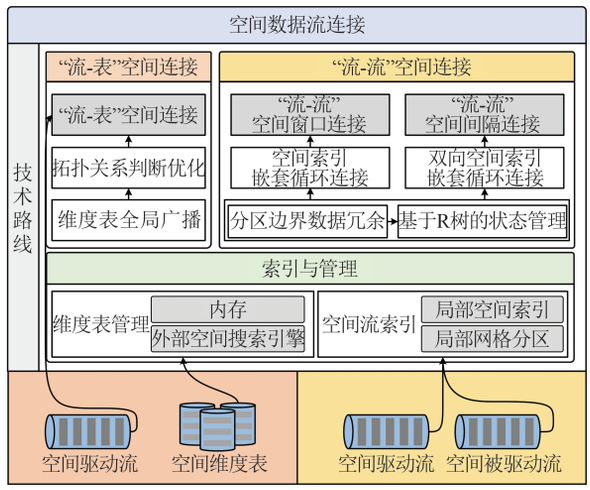

图2 空间数据流连接技术路线

Fig. 2 Technical Routes of the Spatial Data Streams Join

层索引既要实现数据的均匀划分，又要保持每个节点内数据的空间邻近性，以尽可能减少处理时各个计算节点间的通信成本；第二层作为局部索引在每个节点上加速计算。空间数据索引可大致分为两类：(1)网格状索引，如均匀网格、Z-curve、Hilbert等；(2)树状索引，如R-tree、Quadtree等[3,23]。

相较于网格状索引，树状索引的查询效率更高，基于DBPE的方法一般采用两层树状索引的结构。然而，树状索引的维护成本较高，持续到达的空间数据流可能带来频繁的索引重构。网格状索引在给定数据边界的情况下结构固定且不需要维护，因此，本文选择使用基于规则格网的全局分区索引，在各分布式节点使用局部树状索引加速空间查询。

2.2 “流-表”空间连接

“流-表”空间连接的计算图如图3所示，空间维度表以广播的形式发往每个连接算子，故无需对空间数据流构建索引。连接算子读取来自多个输入分区的空间数据流，根据输入条件立即对每个到达的几何对象进行判断，符合连接条件的几何对象与对应维度表连接。

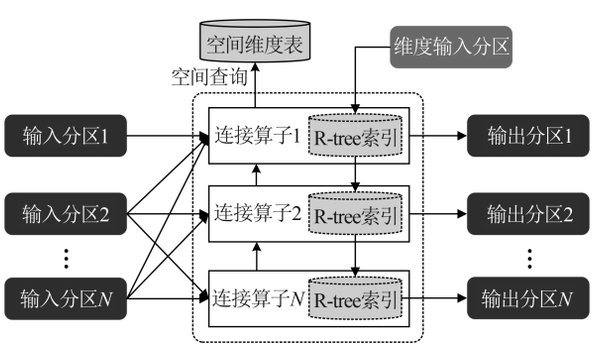

图3 “流-表”空间连接计算图

Fig. 3 “Stream-Table” Spatial Join Calculation Graph

连接算子在初始化阶段可选择是否为本文提供两种空间维度表的物理实现方式:一是内存存储,将维度表作为连接算子的本地状态;二是把空间检索和拓扑关系判断全权交给支持高并发、低延迟空间检索的外部引擎,如GeoMesa[24]。虽然在实现上相对简单,但是由于需要大量访问外部存储,在效率上不如前者。

在流表连接流程中,最关键的问题是连接算子如何高效判断流几何与查询几何之间的拓扑关系。为此,本文提供了两个层面的优化,一是在查询几何数量较多的情况下对其建立R-tree索引,以加速空间搜索效率;二是对常见的拓扑关系判断进行了特殊的优化。本文以最为常见的点与多边形之间的包含关系的判断为例进行优化介绍。

射线法常用于判断点是否包含于多边形内部[25]。现实场景中常存在复杂多边形,如未经简化的行政区划多边形,其边数通常在数千到数万不等。针对此类复杂多边形,本文提出了一种基于R-tree优化的射线法。该算法将多边形的各条边视为独立的几何对象,并为其构建R-tree索引。

如图4所示,A、B分别为点在多边形外部和内部的情况,传统射线法需要将以A、B为起点的射线与多边形所有边进行相交判断,而借助索引可快速定位可能与射线相交的边。以点A为例,首先在R-tree中查询到其包围盒为R_2,则只需将以A为起点的射线与R_2中所有可能相交边(③、④)进行相交判断,即可确定该点与多边形的位置关系,该方法可将时间复杂度从$O(n)$降低到$O(\log n)$。基于R-tree优化的射线法的算法流程如下:

算法1:基于R-tree优化的射线法

输入:空间点P、多边形R;

输出:点是否在多边形内判读结果result。

① R-tree $rt \leftarrow \varnothing$
② for each edge e in R do
③ rt.insert(e)
④ line ← Ray(P)
⑤ result ←false
⑥ for each edge e in rt.query(line) do
⑦ if Cross(e,line) do
⑧ result ← !result

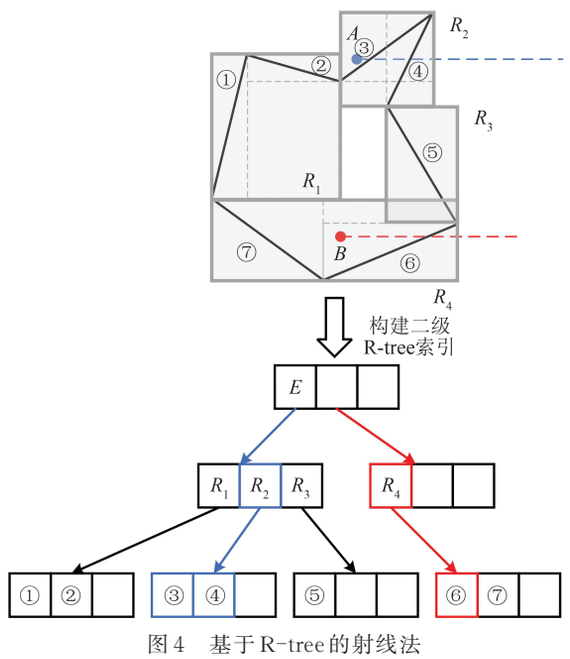

图4　基于R-tree的射线法

Fig. 4　R-tree Based Ray Method

2.3　“流-流”空间连接

2.3.1　“流-流”空间窗口连接

“流-流”空间窗口连接的计算图如图5所示。

网格分配算子从输入分区中读取几何对象并为其生成网格索引,之后根据网格索引对流几何进行重新分发,根据用户指定的窗口大小划分为窗口快照后进行连接处理。本文借鉴了INLJ算法,将其调整后用于空间数据的连接,称之为空间索引嵌套循环连接(spatial index nested loop join,SINLJ)算法。

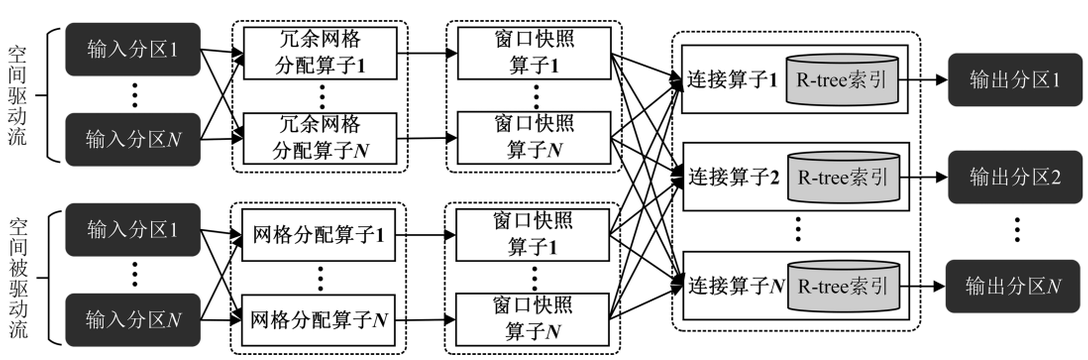

图5　“流-流”空间窗口连接计算图

Fig. 5　“Stream-Stream” Spatial Window Join Calculation Graph

SINLJ算法的流程如下：

算法2：空间索引嵌套循环连接算法

输入：窗口快照S_{1w}、S_{2w}和空间关系判断函数r_∂；

输出：符合连接条件的子对$j(s_1,s_2)$。

① R-tree $rt\leftarrow\varnothing$
② for each s_2 in S_{2w} do
③ rt.insert(s_2)
④ for each s_1 in S_{1w} do
⑤ for each s_2 in rt.query(s_1) do
⑥ if $r_\partial(s_1,s_2)$ do
⑦ output $j(s_1,s_2)$

首先为所有来自被驱动流的几何对象创建R-tree索引，然后遍历所有来自驱动流的几何对象，在遍历过程中通过R-tree索引检索出被驱动流中与之匹配的候选几何，最后判断与候选几何之间的拓扑关系是否符合需求，将满足要求的对象连接后输出。

在空间连接中，落在不同网格内的几何对象很可能会出现跨分区连接的现象。为了解决这一问题，本文设计了一种分区边界数据冗余算法。具体而言，对于驱动流中的几何对象，若其为非点几何对象则为其分配所覆盖的所有分区编号，如图6左图中的多边形A所对应的网格编号为g_4和g_5，若其为点对象且为距离连接，那么为其分区ε邻域所覆盖的所有分区编号，如图6右图中的点B所对应的网格编号为g_4、g_5、g_7和g_8。分区边界冗余网格分配算法的具体流程如下：

算法3：分区边界冗余网格分配算法

输入：几何对象P、距离阈值ε（仅用于点的距离连接）、网格边长l；

输出：网格编号(i,j)。

① box$\leftarrow$GetBoundingBox(P,ε)
② min $X\leftarrow$box.min Lng/l
③ max $X\leftarrow$box.max Lng/l
④ min $Y\leftarrow$box.min Lat/l
⑤ max $Y\leftarrow$box.max Lat/l
⑥ for i from min X to max X do
⑦ for j from min Y to max Y do
⑧ output (i,j)

需要注意的是，在“流-流”空间窗口连接中，仅对驱动流使用分区边界冗余的网格分配算法，而被驱动流采用常规的网格分配算法，否则将出现重复的连接结果。另外，由于冗余网格分配算法的使用，“流-流”空间窗口连接仅支持驱动流包含非点几何对象。为了使连接更加高效，在创建作业时推荐将数据量较小的空间数据流作为驱动流。

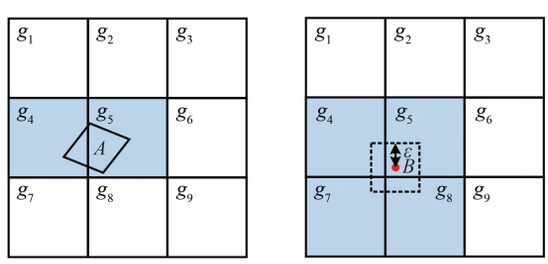

图6 驱动流几何网格分配案例

Fig. 6 Driving Stream Geometry Mesh Assignment Example

图7是一个基于SINLJ算法的“流-流”空间窗口连接示例，在该示例中，两个空间数据流均由空间点组成，窗口快照内点的空间分布如图右上角矩形虚线框内所示。依据SINLJ算法，窗口

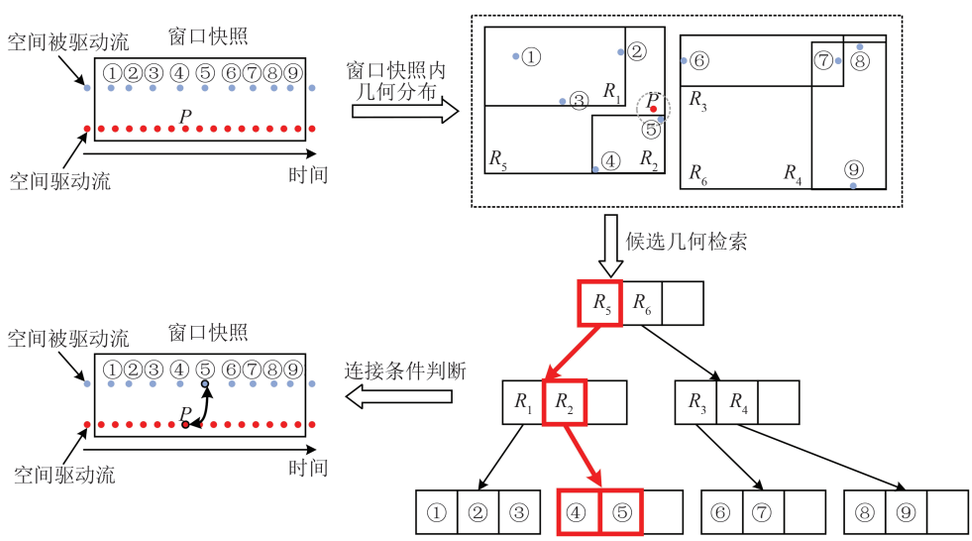

图7 “流-流”空间窗口连接示例

Fig. 7 “Stream-Stream” Spatial Window Join Example

快照内属于被驱动流的点将被插入到R-tree索引中。对于窗口快照内属于驱动流的每个点,通过R-tree索引检索出候选的点,仅以空间点P为例,可在R-tree索引中通过路径$R_5 \rightarrow R_2$快速确定与P关联的候选几何对象④、⑤,进一步距离判断⑤符合连接条件,连接输出。

2.3.2　“流-流”空间间隔连接

如图8所示,相比于“流-流”窗口连接,“流-流”空间间隔连接舍弃了有界的窗口时间语义,在连接处理时更加灵活,但也带来了新的问题。每个空间数据流中的几何对象到达时,均无法确定另一个数据集中落在指定时间范围内的数据是否已经全部到达,难以直接采用SINLJ算法进行连接。为此,本文在SINLJ的基础之上提出了双向空间索引嵌套循环连接(bidirectional spatial index nested loop join,BSINLJ)算法。其算法流程如下:

算法4:BSINLJ算法

输入:空间数据流$\boldsymbol{S}_1$、$\boldsymbol{S}_2$,以及上界时间T_{ub}、下界时间T_{lb}、空间关系判断函数r_δ;

输出:符合连接条件的子对$j(s_1,s_2)$。

① State $st_1 \leftarrow \varnothing$
② State $st_2 \leftarrow \varnothing$
③ for each s_1 in $\boldsymbol{S}_1$ do
④　　for each s_2 in st_2.query(s_1) do
⑤　　if $Ts_1 + T_{lb} \leqslant T(s_2) \leqslant Ts_1 + T_{ub}$ and $r_\delta(s_1,s_2)$ do
⑥　　output $j(s_1,s_2)$
⑦　　st_1.insert(s_1)
⑧ for each s_2 in $\boldsymbol{S}_2$ do
⑨　　for each s_1 in st_1.query(s_2) do
⑩　　if $Ts_2 - T_{ub} \leqslant T(s_1) \leqslant Ts_2 - T_{lb}$ and $r_\delta(s_1,s_2)$ do
⑪　　output $j(s_1,s_2)$
⑫　　st_2.insert(s_2)

首先创建两个状态st_1与st_2分别用于存储来自两个空间数据流的几何对象,对于每个到来的属于$\boldsymbol{S}_1$的几何对象$s_1 g_1$,检索st_2中$[Ts_1+T_{lb}, Ts_1+T_{ub}]$时间间隔内的候选数据,然后筛选出符合空间连接条件的几何对象,连接输出,最后将s_1插入st_1中以记录状态。$\boldsymbol{S}_2$中的几何对象做对称的操作。

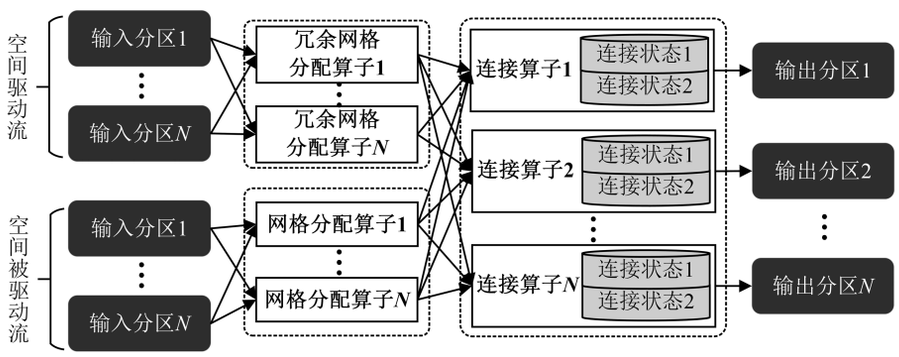

图8　“流-流”空间间隔连接计算图

Fig. 8　“Stream-Stream” Spatial Interval Join Calculation Graph

BSINLJ算法中状态的管理对于其正确执行至关重要。<时间戳,数据列表>是常用的存储结构,只需要判断时间戳是否过期。然而,该结构查询时需要对所有数据遍历计算,对于计算开销大的空间数据有较大优化空间。沿用R-tree是自然的优化思路,但是单R-tree结构数据删除较耗时,<时间戳,R-tree>结构又面临R-tree频繁建立的问题。为此,本文基于数据分箱思想提出了BinR-tree状态管理方法,即对于空间数据流按照其时间戳分箱后,再建立R-tree索引,将数据按照<BinID,R-tree>形式存储,BinID表示箱编号,如图9所示。该方法既保留了主流存储结构状态管理高效的优点,又提升了空间数据查询效率。

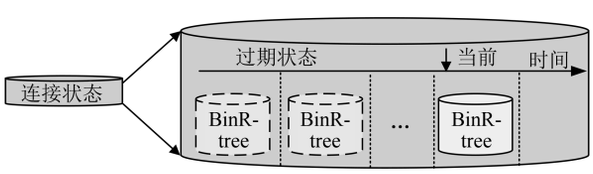

图9　BinR-tree结构

Fig. 9　Structure of BinR-tree

对应地,算法4中的查询和插入操作都需要根据事件时间(timestamp)和预设的分箱大小(Binsize)计算BinID,公式如下:

$$\mathrm{BinID} = \frac{\mathrm{timestamp}}{\mathrm{Binsize}} \tag{4}$$

连接状态存储于内存,随着时间的推移,过期状态的堆积不仅会影响查询性能,一旦超过限

制还会造成内存溢出错误。本文的状态清理方案为在建立BinR-tree的同时设定其对应的清除触发器,按照下式计算触发时间:

$$\max\{t_{\text{binmax}}, t_{\text{binmax}}+T_{\text{rub}}\} \tag{5}$$

式中,t_{binmax}是箱的时间上界;T_{rub}指相对上界(relative upper bound,RUB)时间。对于S_1中元素,相对时间上界为T_{ub};对于S_2中元素,其相对时间上界为$-T_{\text{lb}}$。

图10是一个基于BSINLJ算法的示例,在该示例中,S_1由多边形组成,S_2由空间点组成,其中每个几何对象对应的时间戳如图中表格内所示,连接的上界时间为30 min,下界时间为15 min,分箱间隔为1 h;连接条件为多边形包含空间点。下面基于该示例进一步阐述算法细节。

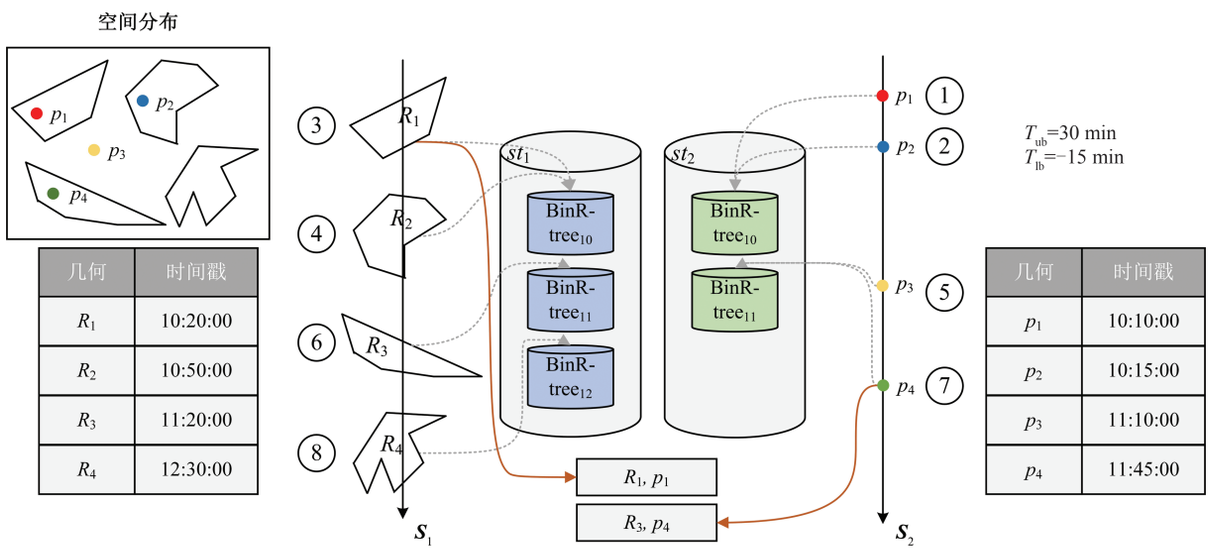

图10 “流-流”空间间隔连接示例

Fig. 10 “Stream-Stream” Spatial Interval Join Example

1)元素行为。元素到达后首先检索对方流中状态是否有可连接对象,如果有,则连接,并将其插入本流的状态中。

2)时间间隔基准。查询的时间间隔以S_1中的元素为基准,S_2中的时间间隔需要对称变换为$[-T_{\text{ub}}, -T_{\text{lb}}]$。示例中,$S_1$中10:20:00到达的$R_1$需要在$st_2$中检索[10:05:00,10:50:00]的数据,而S_2中10:10:00到达的p_1需要在st_1检索的时间范围是[9:40:00,10:25:00]。

3)分箱语义。本文状态管理基于分箱语义,S_1和S_2的状态可能存放于多个BinR-tree中,BinR-tree的建立与删除也以数据驱动。以p_3为例,其到达后,首先需要检索st_1中[10:40:00,11:25:00]的数据,这些数据分布在BinR-tree$_{10}$和BinR-tree$_{11}$中,分别检索;检索与连接完成后,需要将p_3记录于st_2,其目标BinR-tree$_{11}$不存在,建立并注册其清理时间为12:15:00。

3 实验结果与分析

3.1 实验设置

本文所使用的物理机配备Intel(R) Xeon(R) Gold 6240处理器,其基本频率为2.60 GHz,最大睿频频率可达3.90 GHz,内部含36个逻辑内核。内存总量为256 GB,速率为2 933 MT/s。为了模拟分布式环境,本文在物理机上创建了3个虚拟机用于实验。每个虚拟机分配8个逻辑中央处理器内核,64 GB内存,使用Kafka模拟空间数据流,虚拟机集群所配备的主要组件及对应版本分别为JDK 1.8.0_261、Apache Flink 1.13.3、Apache Kafka 2.6.0。

本文采用两组不同规模的数据进行实验评估:(1)T-Drive[26]数据集,该数据集是由10 357辆出租车从2008年2月2日—8日在北京市范围内产生的轨迹,共17 662 984条记录;(2)深圳(Shenzhen)轨迹数据集,该数据集使用深圳市某年连续7天的出租车轨迹数据,与T-Drive数据集同时间跨度,共425 205 439条记录。

本文采用吞吐量和延迟来评价连接查询的性能。吞吐量是指单位时间内系统所处理的数据量,延迟是指数据记录输入和输出流处理引擎的时间差,本文对上述指标分别独立测试以保证准确性。在进行吞吐量测试时,在输入端Kafka中积累一定数据之后再启动作业执行连接查询,以避免由于输入端数据不足而导致的Flink作业

闲置;在延迟测试时,首先启动空间流查询作业,模拟输入流,小批量写入数据以减少延迟测试误差。本文针对空间数据流的特性做了一系列查询优化,为了验证其有效性,以直接使用Flink原生接口的流查询算法(下文称Flink原生方法)作为基线方法。具体来说,由于Flink未提供空间数据分区与索引支持,对于"流-表"连接,遍历维表数据实现连接;对于"流-流"连接,采用随机分区(为保证连接结果不遗漏,将其中一条流广播处理),各分区内遍历所有候选数据进行连接。

3.2　性能评估与分析

3.2.1　"流-表"空间连接性能评估

本文将数据集所在地区的行政区划地理多边形作为查询几何,标记数据流中点的所属行政区。由图11可以看出,本文所实现的"流-表"空间连接在单并行度下可以达到每秒6万以上的吞吐量,总体延迟平均在90 ms左右,同Flink原生方法相比,吞吐量平均有9倍的提升,延迟平均降低20 ms,对于大规模数据查询性能提升更明显。

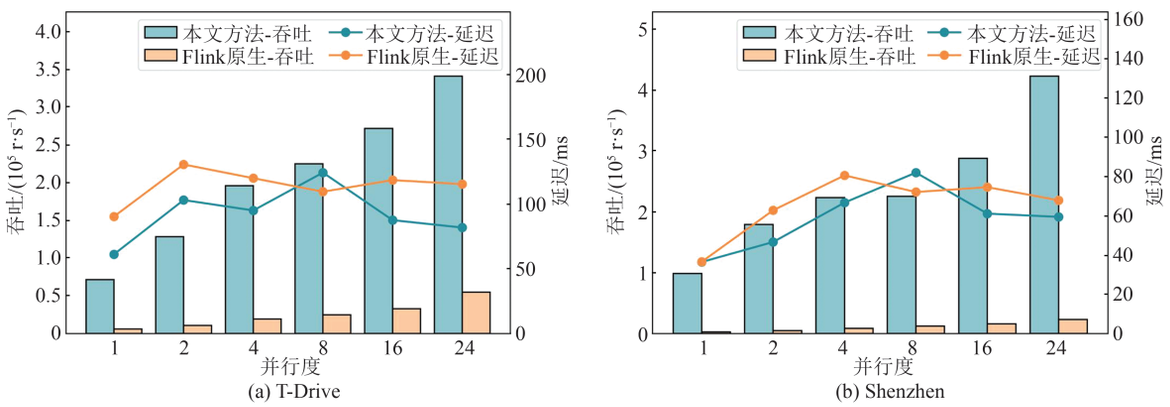

图11　"流-表"空间连接实验结果

Fig. 11　Experimental Results of "Stream-Table" Spatial Join

3.2.2　"流-流"空间连接性能评估

"流-流"连接实验将数据集模拟2次,并视其为两个不同空间数据流,连接查询条件是距离小于1 km。分别从并行度、窗口(时间间隔)大小、网格划分进行实验,对于间隔连接,额外进行分箱大小实验。

1)并行度实验。如图12所示,与"流-表"连接相同,本文"流-流"连接方法随着并行度增加性能平稳提升,具有良好的横向拓展性。本文方法查询效率较Flink原生方法有所提升,且间隔连接的性能提升比窗口连接明显,这是因为间隔连接不存在窗口语义下流数据的等待行为。

2)窗口(时间间隔)实验。如图13所示,窗口(时间间隔)大小对"流-流"连接的吞吐量影响不大,吞吐量呈现上升趋势,主要是由于查询时间范围扩大导致参与计算的元素增多。而延迟方面,窗口连接需要根据事件时间的推进,在特定时间点触发计算,随着窗口的增大,连接延迟提升,远高于流间隔连接。该实验也说明脱离了有界时间语义的流间隔连接更加适合流处理逻辑。

3)网格划分实验。由于Flink原生方法无网格剖分实现,本实验仅进行数据规模对照。如图14所示,网格划分n是指将实验区域剖分为$n\times n$个网格,其主要作用是均衡负载和保证数据的空间邻近性。结果显示,大部分实验组调整网格划分对于性能影响不大。但是对于T-Drive数据集,当网格划分较多后,通信时间开销和边界冗余的物理开销均会增大,反而会增加其延迟(图14(a)),说明网格划分应当根据集群配置和数据分布确定,不是越多越好。

4)分箱大小实验。分箱实验统一在5 min时间间隔下进行,实验结果见图15。对于T-Drive数据集(图15(a)),当箱大小小于窗口大小时,查询会跨多个BinR-tree,影响查询效率,此时增大箱大小可以提升查询效率;当箱大小大于等于时间间隔后,对查询效率的提升不大。对于深圳轨迹数据集,由于其单位时间内数据量更大,各BinR-tree的建立开销小于其查询效率提升缩减的开销,在实验参数范围内性能变化不大。同时,在实验过程中也发现,若箱大小设置过大,会带来大量过期数据的堆积,甚至因为内存占用过大导致流作业执行失败,将状态存储至外部系统是提高容错性的方法。

4　结　语

本文聚焦空间数据流的连接查询问题,首先基于对连接场景共性问题的分析,形式化定义了

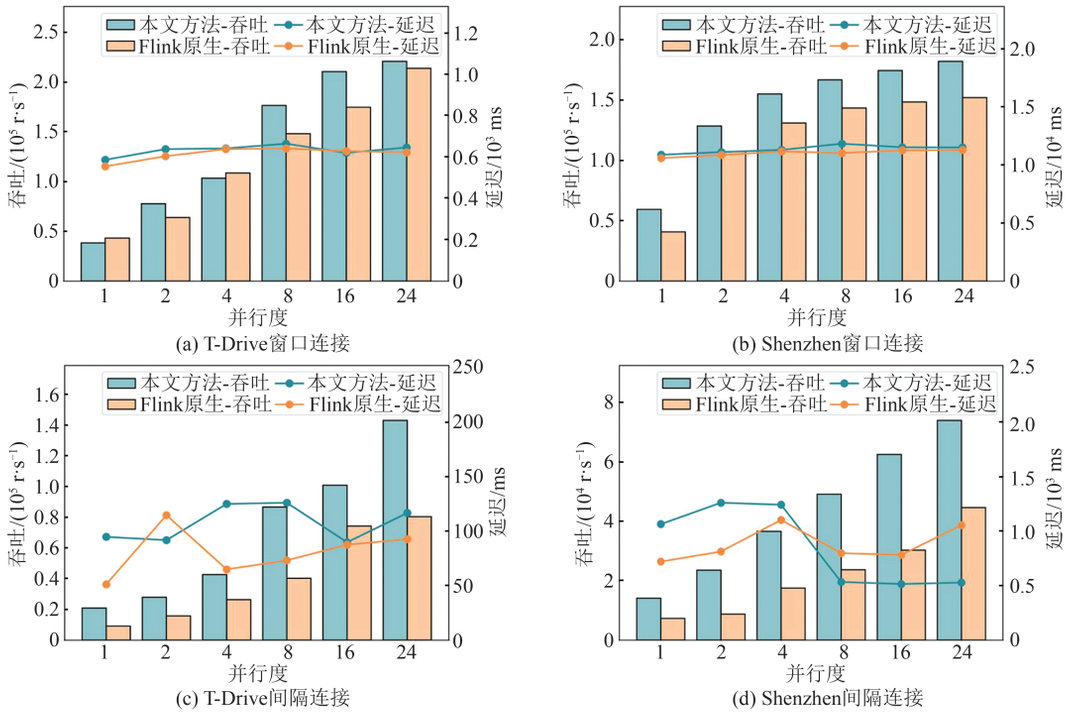

图12 并行度实验结果

Fig. 12 Parallelism Experiment Results

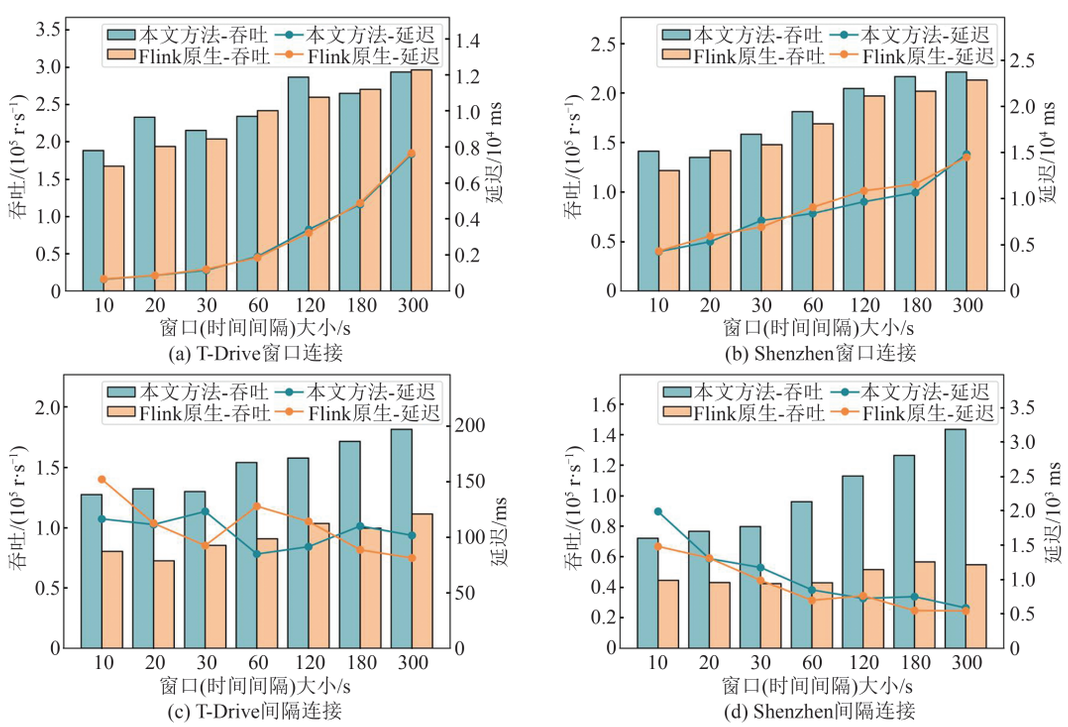

图13 窗口(时间间隔)实验结果

Fig. 13 Window (Interval) Experiment Results

针对空间数据流的各类连接查询处理问题,包括针对空间数据流和空间维度表的“流-表”空间连接,针对两条空间数据流的“流-流”空间窗口连接和“流-流”空间间隔连接。然后在此基础上,基于全局网格数据分区及局部空间索引的两层数据索引结构,针对不同连接场景分别设计了适应的连接策略:对于“流-表”连接,提出了基于两级R-tree的拓扑关系判断优化算法;对于“流-流”连接,设计了一种分区边界数据冗余算法,以保证分区边界数据的正确连接。此外,对于间隔时间语义的缓存需求,基于数据分箱思想提出了一种兼顾状态管理与检索效率的BinR-tree结构。最

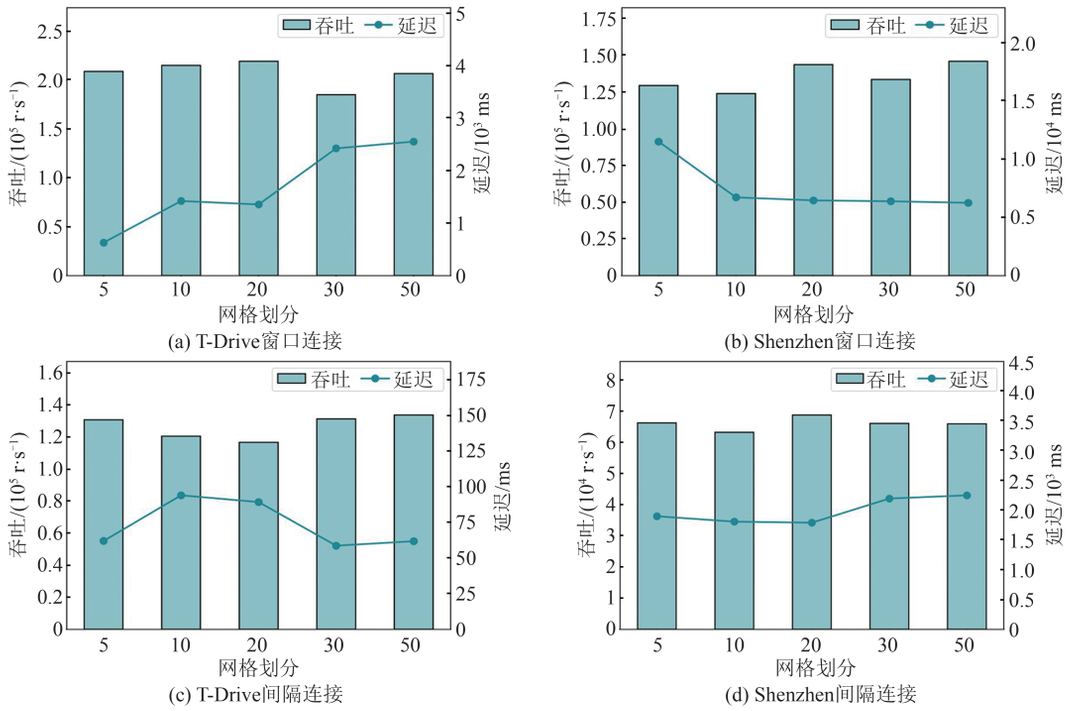

图 14　网格划分实验结果

Fig. 14　Grid Split Experiment Results

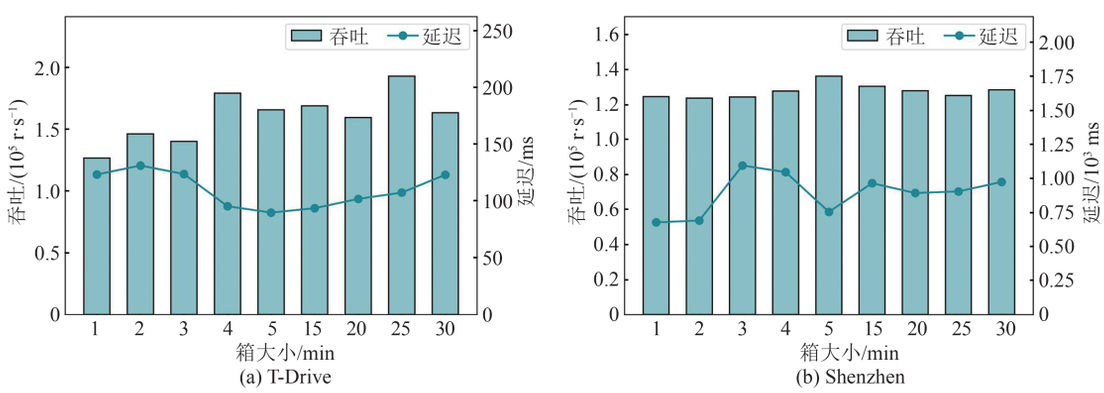

图 15　分箱大小实验结果

Fig. 15　Bin Size Experiment Results

后,在Apache Flink中实现了上述连接查询算法,并基于模拟数据流开展了大量性能测试。实验结果表明,本文所定义的连接处理适用于实时空间数据流处理场景,相较基线方法具有不同程度的性能提升。

在后续工作中,将进一步对空间数据流查询的优化方法进行研究,包括设计更低维护成本的空间流索引结构,设计更多细化场景的空间流连接算子(如多流连接算子)、对于更多拓扑判断规则的优化,以及提高大规模空间数据流查询处理的容错性。

参　考　文　献

[1] 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.
SUN Dawei, ZHANG Guangyan, ZHENG Weimin. Big Data Stream Computing: Technologies and Instances[J]. *Journal of Software*, 2014, 25(4): 839-862.

[2] 朱建军,宋迎春,胡俊,等. 测绘大数据时代数据处理理论面临的挑战与发展[J]. 武汉大学学报(信息科学版),2021,46(7):1025-1031.
ZHU Jianjun, SONG Yingchun, HU Jun, et al. Challenges and Development of Data Processing Theory in the Era of Surveying and Mapping Big Data[J]. *Geomatics and Information Science of Wuhan University*, 2021, 46(7): 1025-1031.

[3] 李军,刘举庆,赵学胜,等. 地理格网模型支持下的轨迹数据管理与分析框架:方法与应用[J]. 武汉大学学报(信息科学版),2021,46(5):640-649.
LI Jun, LIU Juqing, ZHAO Xuesheng, et al. Trajectory Data Management and Analysis Framework

Based on Geographical Grid Model: Method and Application[J]. *Geomatics and Information Science of Wuhan University*, 2021, 46(5): 640-649.

[4] 陈栋, 张翔, 陈能成. 智慧城市感知基站: 未来智慧城市的综合感知基础设施[J]. 武汉大学学报(信息科学版), 2022, 47(2): 159-180.
CHEN Dong, ZHANG Xiang, CHEN Nengcheng. Smart City Awareness Base Station: A Prospective Integrated Sensing Infrastructure for Future Cities [J]. *Geomatics and Information Science of Wuhan University*, 2022, 47(2): 159-180.

[5] 仇阿根. 基于分布式内存计算的空间数据近似查询处理方法[D]. 武汉:武汉大学, 2017.
QIU Agen. In-Memory Distributed Computing Based Approximate Query Processing on Spatial Data[D]. Wuhan:Wuhan University, 2017.

[6] 盛宇裕, 毕硕本, 范京津, 等. 运用交通运行状况指标分析交通热点时空模式[J]. 武汉大学学报(信息科学版), 2021, 46(5): 746-754.
SHENG Yuyu, BI Shuoben, FAN Jingjin, et al. Analyzing Spatiotemporal Patterns of Traffic Hotspots Using Traffic Operation Indicators[J]. *Geomatics and Information Science of Wuhan University*, 2021, 46(5): 746-754.

[7] TAYLOR R W, SHAPIRO L D. Join Processing in Database Systems with Large Main Memories [J]. *ACM Transactions on Database Systems*, 1986, 11(3): 239-264.

[8] AJI A, WANG F S, VO H, et al. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce[J]. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 2013, 6(11): 1009.

[9] ELDAWY A, MOKBEL M F. SpatialHadoop: A MapReduce Framework for Spatial Data [C]// IEEE 31st International Conference on Data Engineering, Seoul, Republic of Korea, 2015.

[10] YOU S M, ZHANG J T, GRUENWALD L. Large-Scale Spatial Join Query Processing in Cloud [C]//IEEE 31st International Conference on Data Engineering, Seoul, Republic of Korea, 2015.

[11] YU J, WU J, SARWAt M. GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data[C]//The 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems,Seattle,Washington D C,USA, 2015.

[12] YU J, ZHANG Z S, SARWAT M. Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond[J]. *GeoInformatica*, 2019, 23 (1): 37-78.

[13] XIE D, LI F F, YAO B, et al. Simba: Efficient In-Memory Spatial Analytics[C]//International Conference on Management of Data, San Francisco, California, USA, 2016.

[14] TANG M J, YU Y Y,MALLUHI Q M,et al. LocationSpark:A Distributed In-Memory Data Management System for Big Spatial Data[J]. *Proceedings of the VLDB Endowment*, 2016, 9(13): 1565-1568.

[15] PANDEY V, KIPF A, NEUMANN T, et al. How Good Are Modern Spatial Analytics Systems? [J]. *Proceedings of the VLDB Endowment*, 2018, 11: 1661-1673.

[16] SHAHVARANI A, JACOBSEN H A. Distributed Stream *K*NN Join[C]//International Conference on Management of Data, Virtual Event China, 2021.

[17] ZHANG F, ZHENG Y, XU D P, et al. Real-Time Spatial Queries for Moving Objects Using Storm Topology[J]. *ISPRS International Journal of Geo-Information*, 2016, 5(10): 178.

[18] GALIĆ Z, MEŠKOVIĆ E, OSMANOVIĆ D. Distributed Processing of Big Mobility Data as Spatio-temporal Data Streams[J]. *GeoInformatica*, 2017, 21(2): 263-291.

[19] SHAIKH S A, MARIAM K, KITAGAWA H, et al. GeoFlink: A Distributed and Scalable Framework for the Real-Time Processing of Spatial Streams[C]//The 29th ACM International Conference on Information & Knowledge Management, Virtual Event Ireland, 2020.

[20] SHAIKH S A, KITAGAWA H, MATONO A, et al. GeoFlink: An Efficient and Scalable Spatial Data Stream Management System [J]. *IEEE Access*, 2022, 10: 24909-24935.

[21] SHEKHAR S, XIONG H. Encyclopedia of GIS [M]. USA:Springer Publishing Company, 2007.

[22] KIMBALL R,ROSS M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling [M]. 3rd ed. New York: Wiley,1996.

[23] TIAN R J, ZHAI H W, ZHANG W S, et al. A Survey of Spatiotemporal Big Data Indexing Methods in Distributed Environment[J]. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2022, 15: 4132-4155.

[24] HUGHES J N, ANNEX A, EICHELBERGER C N, et al. GeoMesa: A Distributed Architecture for Spatio-temporal Fusion[C]//Geospatial Informatics, Fusion, and Motion Video Analytics V, Baltimore, Maryland, USA, 2015.

[25] FOLEY J D, VAN F D, VAN Dam A, et al. Computer Graphics Principles and Practice in C[M]. Massachusetts: Addison-Wesley Professional, 1996.

[26] YUAN J, ZHENG Y, ZHANG C Y, et al. T-Drive [C]//The 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, California, USA, 2010.