计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(5):1−14 [doi: 10.15888/j.cnki.csa.009517]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

大规模时空轨迹数据连接查询效率优化实践①

丁强龙[1], 叶惠珠[2], 袁弘强[3], 李志新[1]

[1](昆明市公安局 科技信息化支队, 昆明 650217)
[2](昆明文理学院, 昆明 650221)
[3](昆明市公安局 情报指挥中心, 昆明 650217)

摘　要: 本文提出一种低集群计算资源条件下, 大规模轨迹类数据同时空关系的快速连接查询算法 DPCP-CROSS-JOIN. 该算法通过对轨迹数据时间字段进行分段交叉编码和位置网格化等方式对连续的轨迹数据离散化, 并以日期和网格区域编码进行两级分区存储. 通过交叉“等值”连接查询, 实现时空连接查询的三级索引、四级加速, 将$n \cdot n$对象间同时空关系连接查询时间复杂度从 O(n^2) 降为 O($n$log$n$). 在 Hadoop 集群上使用 Hive 和 TEZ 等进行大规模轨迹数据连接查询时能将连接查询效率最高提升到 30.66 倍. 该算法以时间段编码作为关联条件, 巧妙绕开连接过程中复杂表达式的实时计算, 以“等值”替代复杂表达式计算连接, 提高 MapReduce 任务并行度, 提升集群存储和计算资源利用率. 在面对仅使用一般优化已几乎无法完成的, 更大规模类似任务, 仍能在数分钟内完成. 实验表明, 该算法具有高效和稳定等特性, 尤其适用低“算力”资源条件下大规模轨迹数据的同时空关系连接查询. 此方法还可作为时空轨迹伴随查找, 对象间关系亲密度判定等的原子算法, 可广泛应用于维护国家安全、社会治安秩序, 预防和打击犯罪, 辅助城乡规划统筹等领域.

关键词: 轨迹数据; 三级时空索引; 复杂表达式连接查询; 交叉编码; 同时空; 低算力条件

引用格式: 丁强龙,叶惠珠,袁弘强,李志新.大规模时空轨迹数据连接查询效率优化实践.计算机系统应用,2024,33(5):1–14. http://www.c-s-a.org.cn/1003-3254/9517.html

Practice of Improving Join Query Efficiency for Large Scale Spatiotemporal Trajectory Data

DING Qiang-Long[1], YE Hui-Zhu[2], YUAN Hong-Qiang[3], LI Zhi-Xin[1]

[1](Detachment of Science and Information Technology, Kunming Public Security Bureau, Kunming 650217, China)
[2](The College of Arts and Sciences·Kunming, Kunming 650221, China)
[3](Intelligence and Command Center, Kunming Public Security Bureau, Kunming 650217, China)

Abstract: This study proposes an algorithm named DPCP-CROSS-JOIN for fast co-spatiotemporal relationship join queries of large-scale trajectory data in insufficient cluster computing resource environments. The proposed algorithm discretizes continuous trajectory data by segmenting and cross-coding the temporal fields of trajectory data and conducting spatiality gridded coding and then stores the data in two-level partitions using date and grid region coding. It achieves 3-level indexing and 4-level acceleration for spatiotemporal join queries through cross “equivalent” join queries. As a result, the time complexity of the co-spatiotemporal relationship join queries among $n \cdot n$ objects is reduced from O(n^2) to O($n$log$n$). It can improve the efficiency of join queries by up to 30.66 times when Hive and TEZ are used on a Hadoop cluster for join queries of large-scale trajectory data. This algorithm uses time-slice and gridding coding as the join condition, thereby cleverly bypassing the real-time calculation of complex expressions during the join process. Moreover, complex expression calculation join is replaced with “equivalent” join to improve the parallelism of

① 基金项目: 公安部应用创新计划 (2019YYCXYNST019)
收稿时间: 2023-11-15; 修改时间: 2023-12-20, 2024-01-10; 采用时间: 2024-01-23; csa 在线出版时间: 2024-04-07
CNKI 网络首发时间: 2024-04-10

© 中国科学院软件研究所 http://www.c-s-a.org.cn

MapReduce tasks and enhance the utilization rates of cluster storage and computing resources. Similar tasks of larger scales of trajectory data that are almost impossible to accomplish using general optimization methods can still be completed by the proposed algorithm within a few minutes. The experimental results suggest that the proposed algorithm is efficient and stable, and it is especially suitable for the co-spatiotemporal relationship join queries of large-scale trajectory data under insufficient computing resource conditions. It can also be used as an atomic algorithm for searching accompanying spatiotemporal trajectories and determining the intimacy of relationships among objects. It can be widely applied in fields such as national security and social order maintenance, crime prevention and combat, and urban and rural planning support.

Key words: trajectory data; 3-level spatio-temporal index; complex expression join query; cross coding; co spatio-temporal; low computing power condition

视频结构化、移动互联、位置服务、物联网等技术的发展, 产生了大量的时空轨迹数据, 仅一个地 (市) 级公安部门一年所采集的这类数据其规模就可能达到 PB 级别, 形成了大量可用于数据挖掘和关系计算的“算料”. 如何高效挖掘和利用这类数据是当前研究的热点和难点[1,2]. 由于受限于资金规模, 市、县级公安部门“算力”有限, 以本单位为例, 全局数十名数据治理和分析师可使用的计算服务器仅 30 多台. 在这种条件下, 不可能将全部“算力”都投入到时空轨迹数据的计算中, 使用常见的计算方法难以“全量”计算分析这些大规模的轨迹数据. 因此, 通过高效“算法”弥补“算力”的“不足”, 研究一种适合市、县级公安机关的, 用于低集群资源环境下的高效时空关系查询和计算方法, 有着十分重要的现实意义和实践价值. 通过挖掘这些巨量轨迹数据内蕴含的特定时空关系, 例如通过挖掘人与人之间同时空关系, 可用于判断人员间一定时间段内的轨迹伴随关系, 以及一定时间内的关系圈、朋友圈等, 在各类行政、刑事案件的扩线, 违法犯罪的延伸打击等方面都有很大的业务价值, 在赋能公共安全、国家安全等领域有十分重要的意义和实践价值.

各级公安机关收集的轨迹数据常被存储于 Hive 或其他类似的分布式数据仓库里. 以 Hive 为例: 其是一款基于 Hadoop 生态 (包括 HDFS、Yarn、MapReduce、TEZ、Spark 等组件和引擎) 的分布式数据仓库, 其自 2.2 版起增加了对复杂表达式连接查询的支持, 可通过复杂表达式连接查询得到时空轨迹间的时空关系. 这对大量的时空轨迹类数据的挖掘提供了又一选项, 但直接使用 Hive 复杂表达式连接查询, 由于需实时计算连接条件后并根据计算结果才能完成连接, 存在 Map 阶段的排序结果在关联过程中得不到利用, TEZ 等计算引擎的内置优化失效等问题, 导致连接查询过程中出现集群资源得不到有效利用, 连接查询需要耗费大量的时间等问题. 因此, 完成 $n\cdot n$ 对象的同时空关系连接查询工作, 其时间复杂度往往是 $O(n^2)$[3], 效率低下. 例如: 完成 4 千万条左右轨迹数据“全量”同时空关系的连接查询任务, 时间消耗一般都在 1 h 以上.

若不采取可行优化策略, 在没有大规模服务器集群资源的低集群计算资源 (算力) 场景下, 通常难以在可接受的时间范围内完成所需的时空关系的计算. 本文的实战场景仅 30 多台服务器 (每台划分为 2 个节点), 可用于连接查询的只有 70 多个节点. 这种条件下, 若按照 Hive 提供的复杂表达式进行连接查询, 完成对一天上亿条轨迹数据之间时空关系的连接查询, 实测当数据规模在 3 千万–4 千万时, 时间消耗基本都超过 1 h, 而当数据超过某个阈值时 (6 千万条), 还存在得不到计算结果的情况. 因此, 提升时空轨迹类数据的连接查询效率, 采用更高效的方法优化或替代复杂表达式连接查询, 对提升时空轨迹类数据时空关系挖掘效率和提升计算稳定性有重要意义.

本文对轨迹数据的采集时间进行分段交叉编码, 并按照“时空”二级分区存储等预处理. 基于预处理结果, 通过用“等值”连接查询替代复杂表达式实时计算连接查询的方式, 避免关联条件的实时计算, 减少连接条件的比对次数, 解决计算引擎的内置优化失效等问题. 所提出的融合“时空二级分区+时间段交叉编码”的快速连接查询方法, 其做法: 首先把连续的时空数据通过时空编码进行离散化, 相当于提前计算好连接查询的关联条件, 把原本需要在连接查询过程中进行 $n\cdot n$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

次实时连接查询条件计算, 优化为了在数据预处理阶段提前进行 n 次计算; 其次, 利用时间段编码以“等值”连接查询替代复杂表达式连接查询的方式, 避免了计算引擎的内置优化失效问题; 第三, 通过交叉的时间段编码进行“斜向”交叉“等值”连接, 解决因时间段“等值”连接出现的连续的时间段编码间的间隙问题. 通过这些步骤, 可实现不同对象间同时空关系的快速连接查询且能保证不丢失时空关系. 采用文献所提出的方法进行优化后, 完成一天轨迹数据同时空关系的连接查询任务, 时间消耗从原来的 1 h 以上, 减少到了 3–5 min, 而且也更加的稳定.

以该方法计算的不同对象间 (例如人员、车辆、电子设备等)“同时空”关系为基础, 可将复杂的同行伴随关系、对象间亲疏远近关系的判定等算法转化为简单的聚合统计查询就可以实现的问题. 例如: 通过该方法计算出的存在同时空关系的 m 辆电动自行车, 当其在一定时间内同时空点位大于某个数字, 且时空顺序相同, 基本就可作轨迹伴随的候选集. 又比如取半年内, 人脸白天同时空或晚上同时空天数大于某个数字的人, 就能够快速地筛选出相关人员的同事、家人等关系人员. 算法的计算结果, 可广泛服务于维护国家安全、维护社会治安、预防和打击电信网络诈骗犯罪、统筹城乡规划等领域的情报线索挖掘.

本文第 1 节介绍国内外类似研究的相关工作. 第 2 节介绍使用的样本数据、集群环境以及对数据进行的加工和转换工作, 包括数据分区、空间网格化、空间网格区域分区计算、时间交叉编码等. 第 3 节给出基于时空二级分区融合时间段交叉编码的时空关系快速连接查询优化策略及算法的主要步骤, 并与两种当前普遍采用的算法进行对比, 验证文章所提出方法的必要性和可行性. 第 4 节对 3 种不同算法进行实验, 验证了该方法对连接查询的加速效果. 第 5 节对实验结果数据进行分析, 并对 DPCP-CROSSS-JOIN 等算法时间复杂度进行简要分析. 第 6 节总结展望.

1 相关工作

目前经有大量对时空大数据的研究, 甚至出现了量子计算用于挖掘时空的关系等方面的研究[1], 相关研究数量非常多, 各种解决方案也相继被提出. 主要包括通过优化 Hadoop 生态各组件性能提升效率, 利用双向连接等方法提升查询效率, 采用分区或分桶等建立时空索引提升效率以及结合业务特征针对性的治理数据和优化提升效率等几类.

1.1 优化 Hadoop 相关配置、算法及执行策略

赵彦荣等人[4]通过设计多副本一致性哈希算法以及通过 Hash Map Join 并行连接查询提升 Hive 连接查询效率. 王华进等人[5]基于 ORC 元数据的 key 频率分布估计方法和相应的负载均衡 key 划分方法提升 Hive 连接查询效率. 马东等人[6]通过 Hive 中大小表关联的优化方法, 利用其中大表的索引特性, 降低传输和分析的数据量, 进而提升大小表关联分析的效率, 解决大表存在索引的时效率低下的问题. 吴锦坤等人[7]、Kulkarni 等人[8]通过优化 Hadoop 各组件和计算引擎配置参数, 防止数据倾斜等方法提升 Hadoop 集群计算效率. 郑灵逸等人[9]通过增加任务并行度和建立中间表组合等方法优化查询. Margoor 等人[10]针对连接查询数据快速膨胀问题, 提出一种贪婪的连接重新排序算法, 在没有统计数据的情况下提升连接查询效率.

1.2 利用分区、分桶和索引提升查询效率

齐恒等人[11]使用时间分区的排序数组和空间分区四叉树数据结构, 局部三维 R 树索引, 实现了数据的时空局部性和负载平衡, 达到时空伴随轨迹的快速查询. Sahal 等人[12] 提出一种基于索引的查询优化方法. Costa 等人[13]通过分区、分桶等方法组织数据, 并验证了这些方法在查询性能提升方面的效果. Arpitha 等人[14]分析优化器、查询等价规则、索引、成本估计和 SQL 表达式隐含依赖关系, 在查询优化中的重要性和作用, 构建查询优化器, 能够以最短的执行时间或响应时间生成评估计划. Xia 等人[15]提出了基于 MapReduce 的并行频繁模式增长 (MR-PFP) 算法, 在 Hadoop 平台上使用大规模出租车轨迹和海量小文件并行处理策略来分析出租车运营的时空特征. 房俊等人[16]针对 HBase 无法直接建立时空索引问题, 基于 HBase 行键创建时空索引, 并通过 Geohash 编码对数据进行降维处理, 实现区域时空数据查询性能的提升. Zhao 等人[17]通过轨迹数据库网格索引和分区两种技术解决 top-k 类似性搜索问题. Jin 等人[18]通过降维策略与 WR 树索引以加快搜索速度, 提出优化方法来提高链接的准确性和鲁棒性. Qin 等人[19]利用时空 K-近邻 (KNN) 算法和贝叶斯方法, 提出一种基于不完全和完整轨迹的链路 TTD 估计框架. 通过 KNN 算法来估计不完整轨迹的虚拟链路旅行时间, 改进粒子滤波器和吉布斯采样等方法, 提

© 中国科学院软件研究所 http://www.c-s-a.org.cn

高 KNN 查找的准确性. 王晨旭等人[20]采用时空二级分桶、多级索引、时间槽索引以及通过建立起“全局索引+动态网格范围计数索引+三维 R 树索引”多级索引等方法实现相似时空轨迹数据的查找, 解决了指定的常数条轨迹的相似轨迹快速查询问题, 适用于 $k \cdot n$ 的大规模数据查找或其他较小数据规模场景.

1.3 利用双向连接等方法提升查询效率

Dwivedi 等人[21]采用双向连接技术提升查询效率, 使用 TPC-H 基准数据集进行了测试, 数据查询效率得到一定程度的提升. Kadari 等人[22]通过多路空间连接的二进制分割方法和条分割技术, 实现了更好的总体周转时间.

1.4 根据业务特征治理数据提升查询效率

何文婷等人[23]公开了一种根据外表关联列的取值集进行分区优化, 通过重复利用各分区的中间结果的方式提升查询效率. 陈喜洲[24]提供了一种提升时序数据快速查询的方法, 通过时间戳偏移, 新增原始记录的时间偏移记录, 实现通过“等值”连接查询解决“非等值”连接查询需求. 该方法在面对需增加偏移数据较多时, 会导致数据的快速膨胀 (例如假设业务发生的前后 1 min 都认为是时间相关的, 使用该算法将导致原始数据 120 倍的膨胀), 增加连接查询的比对次数, 从而影响 SQL 连接查询效率, 业务应用场景比较有限.

1.5 现有研究的重点和不足以及改进方法

这些研究几乎都是围绕着提升时空关系数据的查询或查找的效率问题进行的研究. 如前文所述, 目前主要的几类研究.

第 1 类: 赵彦荣等人[4]、王华进等人[5]、马东等人[6]、吴锦坤等人[7]、Kulkarni 等人[8]、郑灵逸等人[9]以及 Margoor 等人[10]主要是针对 Hadoop 各组件特性进行优化, 例如使用 Map Join 提高效率, 防止数据倾斜等方法提升 Hadoop 集群计算效率.

第 2 类: 齐恒等人[11]、Sahal 等人[12]、Costa 等人[13]、Arpitha 等人[14]、Xia 等人[15]、房俊等人[16]、Zhao 等人[17]、Jin 等人[18]、Qin 等人[19]、王晨旭等人[20]的研究聚焦通过建立各种形式的索引, 提升与给定轨迹相似轨迹的查找效率, 对于解决类似 K-近邻查找有很好的借鉴作用.

第 3 类: Dwivedi 等人[21]、Kadari 等人[22]采用双向连接、多路空间连接这些方法来提升 Hive 连接查询的效率.

第 4 类: 何文婷等人[23]、陈喜洲[24]实现通过“等值”连接查询解决“非等值”连接查询需求.

当前已有的这些研究, 第 1 类主要存在优化完全依赖于大数据组件, 可优化空间有限的问题; 第 2 类主要解决 $k \cdot n$ (k 为常数) 的快速查询问题; 第 3 类仅是一些一般性的优化方法; 第 4 类受制于特定业务场景, 而且会存在连接时数据的快速膨胀的问题, 以及没有解决“间隙”问题. 已有的这些研究对于本文要需要解决的 $n \cdot n$ 对象间“全量”时空关系不能很好适用, 尤其是不能直接利用 Hive SQL 实现轨迹数据的空间关系快速连接查询.

鉴于此, 本文在第 1 类优化的基础上, 参考第 2、4 类优化方法, 着眼于解决时空轨迹类数据连接查询过程中集群资源利用率低, 连接查询过程中数据膨胀过快和查询时间消耗过多等几个方面问题, 以期实现时空轨迹关系的快速连接查询和挖掘. 主要做法包括: 融合“时空二级分区 + 时间段交叉编码”, 把连续的时空数据进行离散化, 通过用“等值”连接查询替代复杂表达式连接查询, 通过交叉编码避免时空数据进行离散化后连接查询产生的“间隙”等. 使用本方法, 可实现 $n \cdot n$ 对象间的时空关系, 例如同时空关系的快速计算, 可将 $n \cdot n$ 对象间同时空关系连接查询时间复杂度从 $O(n^2)$ 降为 $O(n\log n)$.

2 数据结构与数据预处理

为实现低“算力”条件下同时空关系的快速连接查询, 对时空轨迹数据进行时间段交叉编码和二级分区存储等预处理. 用于对比实验的数据是 X 市各类传感器 2022 年 6 月–2023 年 5 月采集的约 63 亿条, 约 230 GB 的时空轨迹数据 t_ost_dp, 数据格式及主要字段见表 1.

表 1　时空数据原始数据 (t_ost_dp)

字段名	字段描述	字段类型	备注
oid	对象ID	string	–
lon	出现位置经度	double	–
lat	出现位置纬度	double	–
t	被采时间	timestamp	精度秒
dp	日期分区字段	string	分区字段

提取采集时间中的日期部分, 并按照日期 (天) *dp* 进行分区存储. 其余字段主要包括对象编号 (*oid*), 经度 (*lon*), 维度 (*lat*), 采集时间 (*t*) 等字段, 数据总规模 230 GB, 记录总数达 63 亿条. 实验过程中详细记录连

© 中国科学院软件研究所 http://www.c-s-a.org.cn

接查询过程中的任务划分、执行查询的任务数、集群资源占用以及连接查询时间消耗等过程数据.

为绕开 Hive SQL 复杂表达式的 $n \cdot n$ 关联条件的实时计算, 首先对数据作系列的预处理, 包括对数据格式化, 提取采集时间中的日期, 对空间位置网格化编码, 计算空间网格区域分区值, 计算时间编码以及其交叉编码等预处理; 其次将预处理的数据以“日期”和“空间网格区域编码”进行二级分区存储, 供后续用, 主要步骤如下.

步骤 1. 将当前日期分区前一天最后一个 Δt 以及后一天的第 1 个 Δt 插入当前日期分区, 填补每个日期分区连接计算存在的缝隙, 实现在同一个日期分区内计算时空关系时不至于丢失前一天和后一天与之相关的关系.

步骤 2. 根据网格编码计算网格区域编码, 作为二级分区字段.

步骤 3. 对每条记录的采集时间进行编码, 生成 3 个时间编码字段和 3 个扩展时间编码字段.

步骤 4. 将时间编码后的数据按照时间以日期 (天) 分区, 空间按网格区域分区, 进行二重分区进行存储, 表名: t_ost_cc.

数据的预处理流程包括提取采集日期, 进行日期分区间隙填充, 计算空间位置网格编码, 计算空间位置网格区域编码, 进行时间段交叉编码以及根据预处理的结果按时空两级分区对数据进行存储等过程, 如图 1 所示.

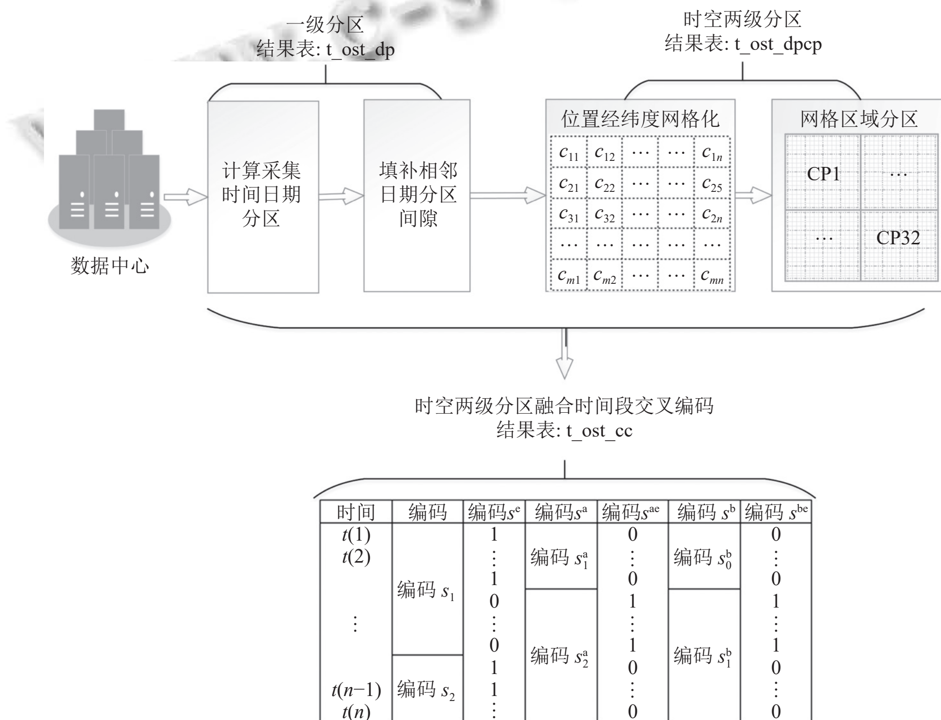

图 1 时空数据预处理主要流程

2.1 数据空间位置编码

空间网格: 采用 Geohash 对所有位置经纬度进行编码, 该编码是 Gustavo Niemeyer 于 2008 年发明的公共领域地理编码系统方法, 是国际上较为通用的一种空间位置网格化方法. 本文计算精度保留前 7 位, 每个编码对应网格覆盖约 153 m×152 m 范围, 将位置纬度对应 Geohash 二进制数据并进行 Base32 编码, 使用的编码字符与十进制数据对应关系见表 2, 得到该经纬度对应的网格编码串, 例如经纬度位置 (25.06, 102.71) 编码后为: wk3n91u, 标记为 c, Geohash 计算公式见式 (1), 目前该算法已有 Java、Python 等不同语言版本实现.

$$c(lon, lat) = GH(lon, lat, \text{Base32}) \tag{1}$$

表 2 Geohash Base32 编码字符与十进制数据对应关系

十进制	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
十进制	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Base32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2.2 对象间同时空关系定义

此研究的对象间同时空关系, 特指两对象 (人员、车辆、电子设备等) 间, 当对象 a 在 t 时刻出现在某空间网格 c 内, 另一对象 b 在其前、后一定时间 $[\Delta t-t, \Delta t+t]$ 段内也出现在同一空间网格区域 c 内, a 与 b 同时空定义见式 (2).

$$F(a,b)=E(a,t,c)\land E(b,t',c) \tag{2}$$

其中, $\Delta t-t\leqslant t'\leqslant t+\Delta t$; $E(a, t, c)$ 表示 a 对象在 t 时刻出现在 c 空间网格内.

2.3 日期分区间隙填充

因时空轨迹类数据规模巨大, 在进行时空数据连接查询时, 使用循环选取每个日期分区数据并进行关联的方式进行. 这种方式存在正在进行连接的当前日期分区数据与前一日期分区及后一日期分区时间连续, 但被割裂的问题. 因此将前一日期分区最后一个 Δt 以及后一日期分区的第 1 个 Δt 对应的数据插入当前日期分区, 填补每个日期分区连接计算存在的缝隙. 具体见算法 1.

算法 1. DP_SP_FILL

输入: t_ost_dp.
输出: t_ost_dp # 更新的 t_ost_dp.

---

1. D←Query t_ost_dp's partitions names
2. Foreach d in D do
3. $tab1\leftarrow\sigma_{(dp=(d-1))\land(\mathrm{abs}(t-FT(d))\leqslant\Delta t)}\rho_{tab1}$(t_ost_dp)
4. Store $tab1$ to t_ost_dp(d) #将 $tab1$ 插入 Hive 仓库 t_ost_dp 的 $dp=d$ 分区中
5. $tab2\leftarrow\sigma_{(dp=(d+1))\land(\mathrm{abs}(t-FT(d+1))\leqslant\Delta t)}\rho_{tab2}$(t_ost_dp)
6. Store $tab2$ to t_ost_dp(d) #将 $tab2$ 插入 Hive 仓库 t_ost_dp 的 $dp=d$ 分区中
7. End

算法 1 中, σ、ρ、$\land$ 等符号系关为运算里面的选择、重命名以及条件合取; $FT(d)$ 表示取时间 d 对应当天的最后时刻对应的时间, 精确到秒, 例如当 d 取 20230502 时, 无论 t 取 2023 年 05 月 02 日什么时刻, $FT(d)$ 都取 2023 年 05 月 02 日 00 时 00 分 00 秒, $FT(d+1)$ 都取 2023 年 05 月 03 日 00 时 00 分 00 秒; $abs(value)$ 表示对 $value$ 取绝对值.

2.4 位置信息网格化及分区

空间网格区域码: 把网格编号 c 转换为十进制, 再进行 mod 32 运算, 目的是把一个日期分区的数据再相对均匀的划分为常数个 (本研究划分为 32 个, 可视集群算力设置合理数值) 不同的空间网格区域, 记 cp, 式 (3), 即把同一日期分区数据再约等分为 32 个空间网格区域, 区域码主要作为分区字段存储.

$$cp(c)=\left(\sum\nolimits_{i=1}^{L}(D(c_i)(32)^{i-1})\right)\%32 \tag{3}$$

其中, L 表示使用的 Geohash 编码的长度, $D(c_i)$ 表示把 Geohash 编码从右到左第 i 个字符转换为对应的十进制数, 例如 wk3n91u 的第 7 位字符"w"对应的十进制数是 28, 第 6 位字符"k"对应的是 18, 以此类推. 采用长度为 7 的 Geohash 编码时, 计算"wk3n91u"的 cp 编码可对照表 2 (Geohash Base32 编码字符与十进制数据对应关系表) 查找到 w、k、3、n、9、1、u 各自对应的十进制数字 28、18、3、20、9、1 和 26, 通过式 (3), 即: cp(wk3n91u) = $(28\times32^6+18\times32^5+3\times32^4+20\times32^3+9\times32^2+1\times32+26)\ \%\ 32=26$, 计算出具体的网格区域分区 cp 值. 根据计算规则可知, 一个网格编码对应若干经纬度, 一个空间网格区域包含若干空间网格, 空间网格化及按网格区域存储的过程见算法 2.

算法 2. DPCP_ETL

输入: t_ost_dp.
输出: t_ost_dpcp # 将 t_ost_dp 按时空二级分区存入 t_ost_dpcp.

---

1. Begin
2. $tab1\leftarrow\rho_{tab1}(oid,lon,lat,t,dp,c,cp)(\prod\nolimits_{oid,lon,lat,t,dp,c(lon,lat),cp(c)}$(t_ost_dp))
3. Store $tab1$ to t_ost_dpcp(dp, cp) #将 $tab1$ 存入 Hive 仓库 t_ost_dpcp 表的 dp, cp 二级分区中
4. End

算法 2 中, c 按照经纬度网格化式 (1) 计算得出; cp 按照网格区域码式 (3) 计算得出.

2.5 时间段交叉编码

将采集时间 t, 转换为对应的 Unix 时间戳 (从 1970 年 1 月 1 日 00 时 00 分 00 秒所经过的秒数, 每过 1 s, 数值加 1), 记为: $u=UT(t)$. 为能通过等值连接解决复杂表达式连接, 需用长度为 $2\Delta t$ 的滑动条对采集时间进行划分和分段编码.

$$s=\lceil(u+1+p)/2\Delta t\rceil \tag{4}$$

使用式 (4), p 值取 0, 计算出时间戳 u 对应的时间段编码 s, 其中"⌈⌉"表示对该符号内数值进行向上取整, 其目的是使用 $2\Delta t$ 为长度的滑块顺序号对时间段进行编码.

$$s^e=(\lceil(ut+1+p)/\Delta t\rceil)\%2 \tag{5}$$

再按照式 (5), p 值取 0, 计算出 s 的扩展码 s^e, 其目

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的是把同一个时间段编码对应的采集时间落在第 1 个 Δt 范围的对应记录的 s^{e} 赋值“1”, 落在第 2 个 Δt 范围对应记录的 s^{e} 赋值“0”, 即把每个编码对应的时间又划分为前一个 Δt 和后一个 Δt 两段, 扩展码用于交叉连接的辅助判断.

类似地, 通过式 (4), p 值取 Δt, 计算出 u 的交叉时间段编码 s^{a}; 以及按照式 (5), p 值取 Δt, 计算其对应的扩展编码 s^{ae}.

同样通过式 (4), p 值取 $-\Delta t$, 计算出 u 的另一个交叉时间段编码 s^{b}; 以及按照式 (5), p 值取 $-\Delta t$, 计算其对应的扩展编码 s^{be}. 交叉编码及其扩展编码组合使用, 可用于解决相邻的两个不同时间段连接查询存在的缝隙问题, 详见算法 3.

算法 3. DP_SP_TSEC

输入: t_ost_dpcp.
输出: t_ost_cc # 将计算结果按时空二级分区存入 t_ost_cc.

1. Begin
2. $tab1 \leftarrow \Pi_{oid,lon,lat,t,dp,c,cp,s,s^{e},s^{a},s^{ae},s^{b},s^{be}}$(t_ost_dpcp) # s, s^{e}, s^{a}, s^{ae}, s^{b}, s^{be} 分别按照式 (4) 和式 (5) 计算得出.
3. Store $tab1$ to t_ost_cc(dp, cp) # 将 $tab1$ 存入 Hive 仓库 t_ost_cc 的 dp, cp 二级分区中
4. End

2.6 时空分区及网格化数据

按照前述方法计算 t_ost_dpcp 中每条轨迹中采集时间对应的时间段编码 s、s^{a}、s^{b} 以及对应的扩展编码 s^{e}、s^{ae}、s^{be}, 计算经纬度位置的 Geohash 网格编码 c, 网格区域分区值 cp, 计算结果存入 t_ost_cc, 见算法 3.

3 优化策略及相关算法

如第 2 节所述, 存在复杂表达式的 Hive SQL 连接查询, 全部数据连接完成至少需要 $n \cdot n$ 次连接条件的实时计算, 需要的计算次数多、耗时长, 连接条件计算成为整个连接查询过程的瓶颈, 难以在低集群计算资源 (算力) 环境下实现高效时空关系连接查询. 为实现低“算力”环境下时空关系的高效连接查询, 需要一是避免连接条件的实时计算, 减少关联条件计算次数; 二是针对存在复杂表达式的连接查询导致的 Hadoop 相关组件优化失效的问题进行优化. 利用数据预处理过程对采集时间的分段编码, 以时间的分段编码而不是时间范围计算作为连接条件可避免连接查询关联条件的实时计算, 同时 “等值”关联代替复杂表达式计算关联又能够避免引擎失效等问题. 为便于标识, 文中所提融合时空二级分区存储和时间段交叉编码的连接查询算法简称 DPCP-CROSS-JOIN. 算法以第 2 节的数据预处理结果 t_ost_cc 为基础, 连接查询主要骤如下.

步骤 1. 通过 Java 等编程语言开发工具, 生成 SQL, 循环提交每个日期分区对应的数据进行连接查询.

步骤 2. 每次只进行一个日期分区数据的连接查询, 待连接表分别标记为 $tab1$、$tab2$, 每次连接查询首先判断空间网格区域分区是否相等, 再比较空间网格是否相等, 然后比较 $tab1$ 与 $tab2$ 时间段编码 s 是否相等, 选取相等的进行连接.

步骤 3. 依然对选定的日期分区数据连接查询, 前几个比对条件同步骤 2, 但是连接查询关联的最后一个条件由比较 $tab1$、$tab2$ 的 s 字段是否相等变为分别比较 $tab1$ 的 s 与 $tab2$ 的 s^{a}、s^{b} 以及 $tab1$ 的扩展编码 s^{e} 与 $tab2$ 的 s^{ae}、s^{be} 是否相等, 选取相等的分别进行连接.

步骤 4. 对 $tab1$ 与 $tab2$ 的连接结果进行过滤, 以编码 s、s^{a}、s^{b} 以及 s^{e}、s^{ae}、s^{be} 等值连接的结果, 连接结果中存在部分关联上的对象时间差大于 Δt, 故需要进行过滤处理.

步骤 5. 按照日期和网格区域分区进行二级分区存储连接查询结果.

主要步骤说明: 步骤 2–步骤 5 以循环和并行方式提交运行, 其中使用步骤 1 提供的工具的目的是为了提高任务的并行度, 步骤 2 通过使用时间段编码实现等值连接查询避免每次连接时实时计算连接条件, 同时避免了 MapReduce、TEZ 等引擎优化失效问题, 从而达到提升连接查询效率的目的, 对应算法见 DPCP-CROSS-JOIN 算法 (第 3.1 节).

步骤 3 通过比较 $tab1$ 的 s 与 $tab2$ 的 s^{a}、s^{b} 以及 $tab1$ 的扩展编码 s^{e} 与 $tab2$ 的 s^{ae}、s^{be} 是否相等, 选取相等的分别进行连接, 主要解决时间离散化后“等值”关联带来的间隙问题, 例如假设在任意一个对象前后 5 s 出现在同一空间网格 c 为存在同时空关系, 按照数据预处理的方法第 0–9 s 将被编码为 1, 记 $s(1)$, 第 10–19 s 编码为 2, 记 $s(2)$, $o1$ 在第 9 s 出现即时间分段编码 $s(1)$ 与 $o2$ 在 11 s 即时间分段编码 $s(2)$ 出现, 明明符合同时空关系, 但因为使用编码 s 作为等值关联条件, 漏算该同时空关系. 通过 $tab1.s = tab2.s^{b}$ 且 $tab1.s^{e} = tab2.s^{be}$ (斜向连接) 作为关联条件进行再次连接查询, 可解决上述漏算问题. 类似地, 通

© 中国科学院软件研究所 http://www.c-s-a.org.cn

过 $tab1.s = tab2.s^{a}$ 且 $tab1.s^{e} = tab2.s^{ae}$(交叉连接) 解决 $o2$ 与 $o1$ 的时空关系漏算问题.

为对比算法的优化效果, 验证过程中也对当前主流的以时空进行二级分区的 DPCP-JOIN 算法和仅以日期进行分区的连接算法 DP-JOIN 也进行了实现 (第 3.2 节).

3.1 时间编码等值连接查询算法

该算法简称 DPCP-CROSS-JOIN, 通过循环日期分区集合 D, 取出每个分区 d 对应的数据, 分别存放于 $tab1$, $tab2$ 临时表中. 对 $tab1$, $tab2$ 进行连接查询, 连接条件为:

(1) $tab1$, $tab2$ 对应记录的空间网格区域分区相同;

(2) $tab1$, $tab2$ 对应记录的空间网格编号相同;

(3) $tab1$, $tab2$ 对应记录的时间段编码相同.

连接完成后, 对连接结果进行过滤, 条件: $(ts \leftarrow |u-u2|) <= \Delta t$.

通过这些步骤得到 $tab1$, $tab2$ 表中每条记录对应其时间段编码的同时空关系, 见图 2, 结果记: γ_1.

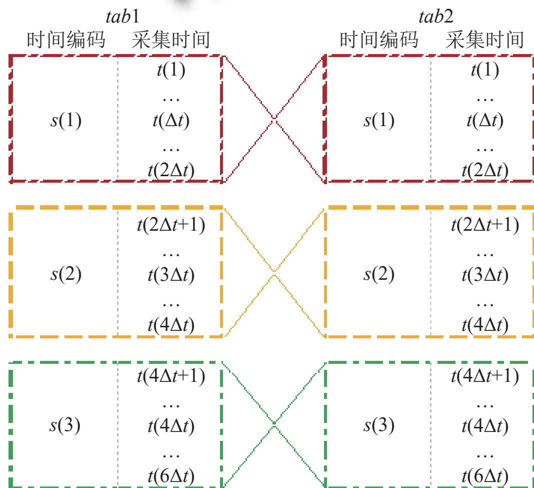

图 2　数据时空分区及时间段连接查询过程

按此方法, 存在:

(1) $tab1$ 表时间编码 $s(1)$ 的后半段即后一个 Δt 与 $tab2$ 表时间编码 $s(2)$ 的前半段即前一个 Δt 的一些同时空关系存在漏算;

(2) $tab2$ 表时间编码 $s(2)$ 的前半段即前一个 Δt 与 $tab1$ 表时间编码 $s(1)$ 的后半段即后一个 Δt 一些同时空关系存在漏算问题.

通过 $tab1$ 表时间编码 $s(1)$ 的后半段即后一个 Δt 与 $tab2$ 表时间编码 $s(2)$ 的前一个 Δt 进行交叉连接查询解决第 1 个漏算问题, 类似地解决第 2 个问题, 见图 3.

连接查询条件: 使用 $s = s_{2}^{a}$, 并且 $s =_{2}^{ae}$ 条件进行连接查询, 结果记: γ_2; 再使用 $s = s_{2}^{b}$, 并且 $s^{e} = s_{2}^{be}$ 条件进行连接查询, 结果记: γ_3, 详见算法 4.

算法 4. DPCP-CROSS-JOIN

输入: t_ost_cc.

输出: $\gamma_1, \gamma_2, \gamma_3$.

1. D←Query t_ost_cc's partitions names
2. $tab1 \leftarrow \sigma_{dp=d}\rho_{tab1}$(t_ost_cc)
3. $tab2 \leftarrow \sigma_{dp=d}\rho_{tab2(oid2,c2,tab2,u2,s_2,s_2^{e},s_2^{a},s_2^{ae},s_2^{b},s_2^{be},cp2)}$(t_ost_cc)
4. Foreach d in D do
5. 　# 并发提交 $\gamma_1, \gamma_2, \gamma_3$ 的连接查询及存储
6. 　Begin Concurrency execute(part 1, part 2, part 3) part 1:
7. 　　$\gamma_1 \leftarrow \sigma_{ts \leqslant \Delta t}\left(\Pi_{oid,c,t,oid2,tab2,ts \leftarrow |u-u2|}\left(\underset{(cp=cp2)\wedge(c=c2)\wedge(s=s_2)}{tab1 \text{ JOIN } tab2}\right)\right)$
8. 　　Store γ_1 #将连接查询结果 γ_1 存入 Hive 数据仓库
9. 　End Concurrency part 1
10. 　Begin Concurrency execute(part 1, part 2, part 3) part 2:
11. 　　$\gamma_2 \leftarrow \sigma_{ts \leqslant \Delta t}\left(\Pi_{oid,c,t,oid2,tab2,ts \leftarrow |u-u2|}\left(\underset{(cp=cp2)\wedge(c=c2)\wedge(s=s_2^{a})\wedge(s^{e}=s_2^{ae})}{tab1 \text{ JOIN } tab2}\right)\right)$
12. 　　Store γ_2 #将连接查询结果 γ_2 存入 Hive 数据仓库
13. 　End Concurrency part 2
14. 　Begin Concurrency execute(part 1, part 2, part 3) part 3:
15. 　　$\gamma_3 \leftarrow \sigma_{ts \leqslant \Delta t}\left(\Pi_{oid,c,t,oid2,tab2,ts \leftarrow |u-u2|}\left(\underset{(cp=cp2)\wedge(c=c2)\wedge(s=s_2^{b})\wedge(s^{e}=s_2^{be})}{tab1 \text{ JOIN } tab2}\right)\right)$
16. 　　Store γ_3 #将连接查询结果 γ_3 存入 Hive 数据仓库
17. 　End Concurrency part 3
18. End

t	ut	s	s^{e}	s^{a}	s^{ae}	s^{b}	s^{be}
2023/5/1 0:00:00	1682899201	168289921	1	168289921	0	168289920	0
2023/5/1 0:00:01	1682899202	168289921	1	168289921	0	168289920	0
2023/5/1 0:00:02	1682899203	168289921	1	168289921	0	168289920	0
2023/5/1 0:00:03	1682899204	168289921	1	168289921	0	168289920	0
2023/5/1 0:00:04	1682899205	168289921	1	168289921	0	168289920	0
2023/5/1 0:00:05	1682899206	168289921	0	168289922	1	168289921	1
2023/5/1 0:00:06	1682899207	168289921	0	168289922	1	168289921	1
2023/5/1 0:00:07	1682899208	168289921	0	168289922	1	168289921	1
2023/5/1 0:00:08	1682899209	168289921	0	168289922	1	168289921	1
2023/5/1 0:00:09	1682899210	168289921	0	168289922	1	168289921	1
2023/5/1 0:00:10	1682899211	168289922	1	168289922	0	168289921	0
2023/5/1 0:00:11	1682899212	168289922	1	168289922	0	168289921	0
2023/5/1 0:00:12	1682899213	168289922	1	168289922	0	168289921	0
2023/5/1 0:00:13	1682899214	168289922	1	168289922	0	168289921	0
2023/5/1 0:00:14	1682899215	168289922	1	168289922	0	168289921	0
2023/5/1 0:00:15	1682899216	168289922	0	168289923	1	168289922	1
2023/5/1 0:00:16	1682899217	168289922	0	168289923	1	168289922	1
2023/5/1 0:00:17	1682899218	168289922	0	168289923	1	168289922	1
2023/5/1 0:00:18	1682899219	168289922	0	168289923	1	168289922	1
2023/5/1 0:00:20	1682899220	168289922	0	168289923	1	168289922	1

交叉连接

斜向连接

图 3　数据时空分区及时间段交叉连接查询过程

3.2 常规两级索引优化连接查询算法

常规两级索引查询优化算法 (简称 DPCP-JOIN), 通过循环日期分区集合 D, 取出每个日期分区 d 对应

© 中国科学院软件研究所　http://www.c-s-a.org.cn

www.c-s-a.org.cn

的数据, 分别存放于 *tab*1, *tab*2 临时表中. 对 *tab*1, *tab*2 进行连接查询, *tab*1, *tab*2 表字段命名规则同算法 4, 连接条件如下:

(1) 空间网格区域分区 *cp* 与 *cp*2 相同;

(2) 空间网格编号 *c* 与 *c*2 相同;

(3) *tab*1, *tab*2 每条记录的采集时间 *u* 与 *u*2 差的绝对值 *ts* 小于等于Δt.

为了便于后续比较, 连接查询结果标记为 β.

仅按日期分析的查询优化算法 (简称 DP-JOIN), 通过循环计算数据表日期分区集合 *D*, 取每个日期分区 *d* 对应的数据, 分别存放于 *tab*1, *tab*2 临时表中, 使用该算法对 *tab*1, *tab*2 进行连接查询, 连接条件如下:

(1) 空间网格编号 *c* 与 *c*2 相同;

(2) *tab*1 待关联记录与 *tab*2 每条记录的采集时间 *u* 与 *u*2 差的绝对值 *ts* 小于等于Δt, 进行 *tab*1 表每一条记录的关联需要进行 *n* 次实时计算.

连接后 *tab*1 对应的 *oid*, *c*, *t* 等字段名称保持不变, *tab*2(*oid*, *c*, *t*) 字段分别重命名为 (*oid*2, *c*2, *tab*2), 为了便于后续比较, 连接查询结果标记为 α.

4 实验及数据

为比较 DPCP-CROSS-JOIN 算法的在低“算力”条件下的性能优势, 使用 2023 年 5 月中 31 天的轨迹数据在腾讯“私有云”上分别对 3 个算法进行了 SQL 实现和实验, 并记录了主要的过程数据. 为简化实验过程, 连接查询采用同一轨迹数据表的自连接, 故 *tab*1 和 *tab*2 的记录集相同, 记录数为 *n*.

DPCP-CROSS-JOIN 算法的主要过程包括, 首先通过 Java 编写的并行批处理程序循环提交每个日期分区“一天”的轨迹数据连接查询任务, 任务依算法 4 实现, 计算出 γ_1、γ_2 和 γ_3. 因为 γ_1、γ_2 和 γ_3 逻辑上可分, 计算结果可合并, 故其计算通过并行方式提交至服务集群. 然后, 将 γ_1、γ_2 和 γ_3 按照日期和网格区域分区进行二级分区并写入连接结果表.

DPCP-JOIN 算法和 DP-JOIN 比较简单, 直接通过工具循环提交每一天的轨迹数据, 并分别按照第 3.2 节所述的关联方式完成连接查询, 并把结果 β、α 存入对应的表中.

为便于比对优化效果, 在查询过程中, 详细记录了每个日期分区对应的轨迹数据连接查询过程中的任务划分、执行查询的任务数、集群资源占用、时间消耗等数据. 实验环境、工具和数据预处理, 实验过程及记录详见第 4.1–4.6 节.

4.1 实验软硬件环境

此实验环境包含 75 个节点, 每个节点含 24 个 VCores 和 40 GB 内存. 分配的实验租户内存 1.8 TB, HDFS 存储容量 200 TB. 在进行本实验前, 已对 Hadoop 后台各类参数进行了反复优化和调整. Hive 采用的版本是 3.1.2, 相关大数据组件及版本见表 3.

4.2 并行提交工具

为实现自动及并行按日期分区提交连接查询 SQL 语句, 使用 Java 语言开发程序, JDK 版本 17.0.1. 工具的主要处理流程见图 4. 本文的用到连接查询语句, 都使用模板进行定义, 涉及需要循环执行的日期分区信息通过占位符进行表达, 执行过程中依次取出每个日期分区参数列表中的内容替换占位符内容, 然后启动进程将连接查询语句提交至 HiveServer2 端执行.

表 3 大数据组件及编程环境

主要组件	版本	主要功能
HDFS	3.2.1	分布式文件系统: 支持高吞吐量、高度容错性的海量数据分布式文件系统
MapReduce	3.2.1	分布式计算框架
Hive	3.1.2	基于Hadoop的数据仓库工具: 用于将SQL语句转换为MapReduce任务运行
ZooKeeper	3.6.2	分布式协调服务: 提供分布式应用提供一致性服务, 提供配置维护、名字服务、分布式同步、组服务等功能
Yarn	3.2.1	提供统一的资源管理和调度服务
TEZ	0.10.1	TEZ是Apache开源的支持DAG作业计算框架, 将Map和Reduce两个操作进一步拆分提升优化MapReduce框架性能
Java	1.17	用于开发工具生成SQL语句并控制其提交方式, 记录主要执行过程

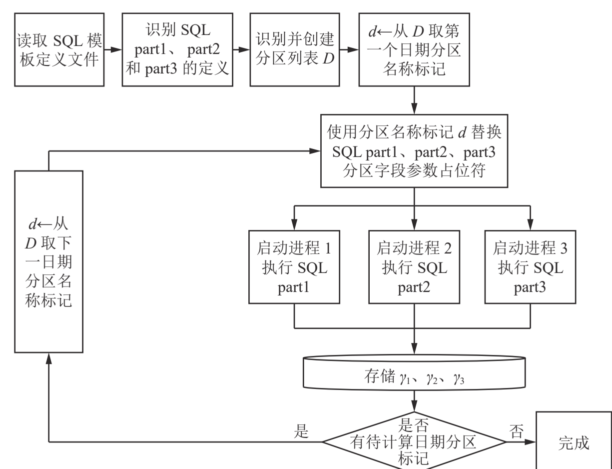

图 4 工具并行处理流程

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4.3 仅按日期分区存储连接查询

DP-JOIN 算法的第 7 个步骤对应的主要连接查询语句见 SQL 01.

SQL 01

```
SELECT t1.c, t1.oid, t1.t, t2.oid AS oid2, t2.t AS t2, t1.u – t2.u AS ts, t1.dp FROM (SELECT * FROM t_ost_dp WHERE dp = [:d]) AS t1 JOIN (SELECT * FROM t_ost_dp WHERE dp = [:d]) AS t2 ON t1.c = t2.c AND abs(t1.u – t2.u) <= 5
```

SQL 01 提交到 Hive 服务后, 连接查询共创建 Map1 和 Map5 这 2 类 Map, 平均预分配任务数大约都是 2.9 个; 创建 Reducer 2 、Reducer 3 和 Reducer 4 这 4 类 Reducer, 平均预分配任务分别为 55.1、1 009 和 4.7 个.

通过实验: 20230501–20230531 每个日期分区连接查询时间消耗分别是: 4 507 s, 4 721 s, 3 920 s, 4 877 s, 4237 s, 3929 s, 6559 s, 4607 s, 9 182 s, 5230 s, 4222 s, 4348 s, 3868 s, 3429 s, 2960 s, 4596 s, 4038 s, 3816 s, 5221 s, 5406 s, 3907 s, 6718 s, 4701 s, 10457 s, 6303 s, 6664 s, 4935 s, 4036 s, 4862 s, 10138 s, 4733 s. 累计耗时 161 127 s (约 44.76 h), 平均每天数据的连接查询消耗 5 197.65 s (约 1.44 h).

使用 DP-JOIN 算法执行连接查询, 由于耗时过多, 因未在一次实验完成 31 个日期分区数据的连接查询, 故实验依日期分区先后顺序进行了 3 次 (对应集群资源消耗用蓝色实线、紫色线段、黄色点线框定), 任务执行期间集群 CPU 和内存使用及消耗是紧密正相关的, 具体资源消耗情况见图 5.

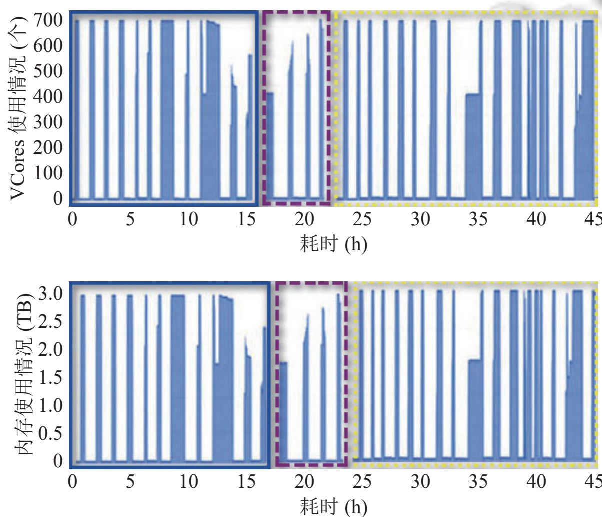

图 5 DP-JOIN 连接查询集群资源消耗

4.4 两级分区存储连接查询

DPCP-JOIN 算法的第 7 个步骤对应的主要连接查询语句见 SQL 02.

SQL 02

```
SELECT t1.c, t1.oid, t1.t, t2.oid AS oid2, t2.t AS t2, t1.u – t2.u AS ts, t1.dp FROM (SELECT * FROM t_ost_dpcp WHERE dp = [:d]) AS t1 JOIN (SELECT * FROM t_ost_dpcp WHERE dp = [:d]) AS t2 ON cp = t2.cp AND t1.c = t2.c AND abs(t1.u – t2.u) <= 5
```

SQL 02 提交到 Hive 及 Hadoop 后, 连接查询共创建 Map1 和 Map5 这 2 类 Map, 平均预分配任务数都是 32 个; 创建 Reducer 2 、Reducer 3、Reducer 4 这 4 类 Reducer, 平均预分配任务分别为 216.2 个、1 009 个和 145.7 个. 20230501–20230531 每个日期分区数据连接查询时间消耗分别是: 1 570 s, 1 144 s, 1 100 s, 1 504 s, 1999 s, 1786 s, 1224 s, 1568 s, 1742 s, 1482 s, 1364 s, 1674 s, 1 148 s, 1 161 s, 838 s, 1370 s, 1 140 s, 1 404 s, 2308 s, 2002 s, 1548 s, 2735 s, 2635 s, 3230 s, 2677 s, 2472 s, 1896 s, 1414 s, 1 814 s, 2 958 s, 1 490 s. 累计耗时 54 397 s (约 15.11 h), 平均执行一天数据的连接查询消耗 1 754.74 s (约 0.49 h). 使用 DPCP-JOIN 算法执行连接查询期间集群 CPU 和内存使用及消耗同样是紧密正相关的, 具体资源消耗情况见图 6.

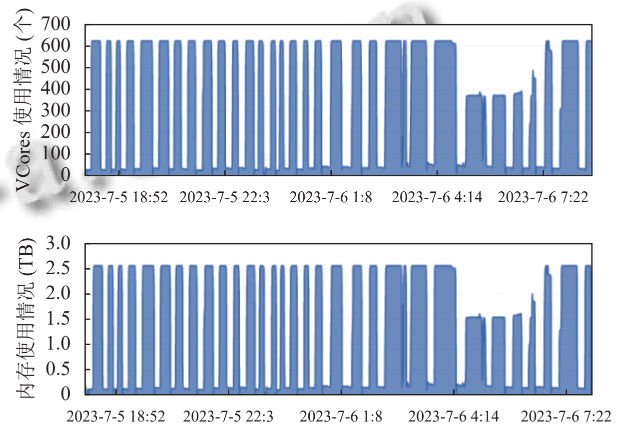

图 6 DPCP-JOIN 连接查询集群资源消耗

4.5 三级索引四级加速连接查询

DPCP-CROSS-JOIN 算法对应的交叉连接查询 SQL 语句共分 3 个部分, 对应的连接查询语句包括 SQL 03 part1、SQL 03 part2 和 SQL 03 part3, 即 DPCP-CROSS-JOIN 算法中的第 9、13、17 这 3 行, 以并发方式提交执行. 实验期间集群资源相对充裕, part1、part2 和 part3 并发执行 SQL 实现了并行执行.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

SQL 03 part1

```
SELECT t1.c, t1.oid, t1.t, t2.oid AS oid2, t2.t AS t2, t1.u – t2.u AS ts, t1.dp FROM (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t1 JOIN (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t2 ON cp = t2.cp AND t1.c = t2.c AND t1.s = t2.s WHERE abs(t1.u – t2.u) <= 5
```

SQL 03 part2

```
SELECT t1.c, t1.oid, t1.t, t2.oid AS oid2, t2.t AS t2, t1.u – t2.u AS ts, t1.dp FROM (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t1 JOIN (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t2 ON cp = t2.cp AND t1.c = t2.c AND t1.s = t2.sa AND t1.se = t2.sae WHERE abs(t1.u – t2.u) <= 5
```

SQL 03 part3

```
SELECT t1.c, t1.oid, t1.t, t2.oid AS oid2, t2.t AS t2, t1.u – t2.u AS ts, t1.dp FROM (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t1 JOIN (SELECT * FROM ots_dpcp_cc WHERE dp = [:d]) AS t2 ON cp = t2.cp AND t1.c = t2.c AND t1.s = t2.sb AND t1.se = t2.sbe WHERE abs(t1.u – t2.u) <= 5
```

SQL 03 的 3 个部分的连接查询语句以并行方式提交到 Hive 服务, 每个日期分区连接查询平均集群资源分配 Map 和 Reducer 数量见表 4.

表 4 Hadoop 集群平均资源分配 (DPCP-CROSS-JOIN)

集群	Map 1	Map 5	Reducer 2	Reducer 3	Reducer 4
$\gamma 1$	32	32	413.2	1 009	4.6
$\gamma 2$	32	32	406.7	1 009	3.9
$\gamma 3$	32	32	411.2	1 009	3.8
合计	96	96	1 231.1	3 027	12.3

使用 DPCP-CROSS-JOIN 算法执行连接查询耗时秒数见表 5, 期间集群 CPU 和内存使用及消耗见图 7. 并发执行完 SQL 03 的每个日期分区 d 的 3 个部分 part1、part2 和 part3 连接查询共耗时间 5 255 s (约 1.46 h), 仅为表 5 所示的每个日期分区 3 个部分耗时合计 14 809 s 的 37.3%, 系并行加速带来的优势, 平均执行一天数据的连接查询消耗 169.53 s (约 0.05 h).

表 5 DPCP_ CROSS-JOIN 连接查询耗时 (s)

日期分区	$\gamma 1$	$\gamma 2$	$\gamma 3$	合计
20230501	106	96	98	300
20230502	398	386	98	882
…	…	…	…	…
20230530	122	108	102	332
20230531	100	112	110	322
合计	5 251	4 587	4 251	14 089
平均	169.39	147.97	137.13	454.484

4.6 更大规模数据连接查询扩展实验

使用 DPCP-CROSS-JOIN 算法, 实验数据扩大 10 倍, 同样选取 31 个日期分区数据共计 (485 GB, 59.6 亿条) 数据进行连接查询. DP-JOIN 算法和 DPCP-JOIN 算法面对更大规模数据时, 仅执行 1 个日期分区数据的同时空连接查询时间消耗都超过 6 h, 难以满足业务需求, 故未进行进一步的实验. 使用 DPCP-CROSS-JOIN 算法完成 31 个日期分区数据的同时空连接查询共耗时 6 960 s, 计算 10 倍于第 4.5 节实验的数据仅多消耗 1 705 s, 多耗时 32.45%. 连接查询期间集群资源消耗图 8 所示.

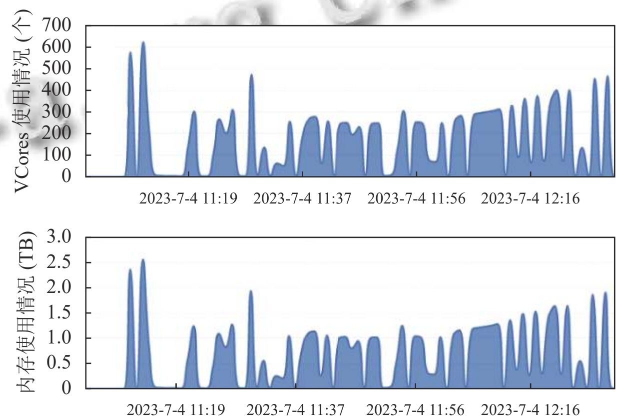

图 7 DPCP-CROSS-JOIN 连接查询集群资源消耗

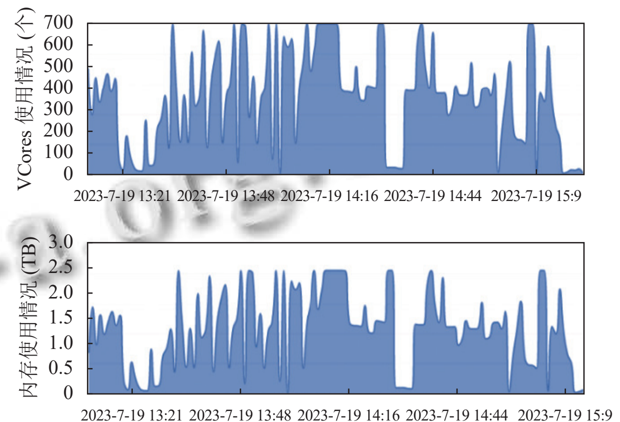

图 8 DPCP-CROSS-JOIN 更大规模连接查询资源消耗

5 实验数据分析

本节主要对 DPCP-CROSS-JOIN、DPCP-JOIN 和 DP-JOIN 实验过程中集群资源利用情况, 执行任务的 Map、Reduce 数量, 各算法执行相同任务的时间消耗情况进行对比分析. 实验结果表明: DPCP-CROSS-JOIN 算法相比 DP-JOIN 算法, 连接查询性能提升了 30.66 倍, 总体性能提升 21.38 倍, 相比 DPCP-JOIN 算法也有明显提升, 能更好适用于低“算力”条件下时空轨迹类

© 中国科学院软件研究所 http://www.c-s-a.org.cn

数据的快速连接查询. 第 4.6 节的扩展实验数据每个日期分区数据规模平均达 1.92 亿条, 在使用 DPCP-JOIN 和 DP-JOIN 在本文的实验环境下执行 6 h 以上都未计算出结果的情况下, DPCP-CROSS-JOIN 算法每个日期分区对应数据的连接查询仍然能在 5 min 以内完成, 进一步说明的算法的连接查询性能提升明显, 而且非常稳定.

5.1 通过分区索引优化的连接查询效率提升有限

显然通过连常规接查询计算$n\cdot n$对象是否存在同时空共需要进行 n^2 次时间差减法计算和 n^2 次两个时间差是否在指定范围内的判断, 故 DP-JOIN 算法时间复杂度为 $O(n^2)$.

相比较于 DP-JOIN 算法, DPCP-JOIN 算法通过日期分区 (*dp*)+空间位置网格区域分区 (*cp*) 二级分区, 降减少每次连接查询的加载的数据规模, 增加了并行的 Map 和 Reducer 数, 一定程度上减少每次连接查询比对和时间差的计算次数, 但是由于连接条件存在复杂表达式, 每次进行连接都要计算和所有记录的时间差, 导致 Hadoop 的 TEZ 等引擎无法利用 Map 阶段的排序结果, 故连接查询性能提升有限, 耗时仍然多. 通过分析 DPCP-JOIN 算法可知, 同一日期分区的 *tab*1 和 *tab*2 记录进行连接查询时, *tab*1 的每一条记录首先判定 *tab*2 表记录的空间网格区域分区相同是否相等, 然后再计算日期分区内对应的所有记录采集时间差后判断, 计算次数为 n^2/Size(*cp*), 查询效率有了一定提升, 但还是存在 TEZ 等引擎无法利用 Map 阶段的排序结果, 导致引擎优化失效的问题, 因此该算法的时间复杂度仍是 $O(n^2)$.

因为此次实验将空间网格进一步划分为了 32 个区域作为分区, 故 Size(*cp*) = 32, 故时间差的计算次数仅是 DP-JOIN 算法的 1/32. 实验结果表明, DPCP-JOIN 算法所采取的时空二重分区索引方法的时间消耗为 DP-JOIN 算法的 33.76%.

5.2 DPCP-CROSS-JOIN 连接查询效率高

DPCP-CROSS-JOIN 算法在 DPCP-JOIN 算法的基础上, 结合交叉编码, 在极大地减少每次连接查询的加载的数据规模的基础上, 通过时间编码进行“等值”连接, 可充分利用集群计算引擎 (如 MapReduce、TEZ) 的优化能力, 实现类似“日期分区+空间分区+交叉编码索引”三级加速的效果.

通过分析 DPCP-CROSS-JOIN 算法可知, 同一日期分区的 *tab*1 和 *tab*2 记录连接关联计算次数主要如下.

(1) 首先判断 *tab*1 记录与 *tab*2 表记录的空间网格区域分区相同是否相等, 按照等值连接可利用 Map 阶段排序结果的优势, 完成一条记录的连接需 log(Size(*cp*)) 次判断.

(2) 再在同一个空间网格区域内, 查找是有满足关联条件的网格, 平均判断次数为 log(*n*/Size(*cp*)) .

(3) 然后再判断时间编码是否相等, 由于已经将复杂表达式连接优化为了等值连接, 可以充分利用 Map 阶段的排序结果, 在有序表中查找条件可能符合的记录比对判断 (log*n*) 次. 因此完成一条记录 γ_1、γ_2 和 γ_3 共计需要比对 3log*n* 次, 比对完成 *n* 条需 (3*n*log*n*) 次判断.

(4) 最后连接结果根据 *u*1 与 *u*2 的时间差过滤, 共需比较 2*kn* 次, 包含 γ_1 的 *kn* 次计算及过滤, γ_2、γ_3 各 *kn*/2 次计算及过滤 (*kn* 为连接后的结果表记录数, 其中 *k* 远小于 *n*).

(5) 综合上述 4 项, 其主要比较次数为 *n*log(Size(*cp*)) + *n*log(*n*/Size(*cp*)) + 3*n*log*n* + 2*n*.

通过上述步骤分析可知该算法总的时间复杂度为 O(*n*log*n*). 在相同集群及相同租户资源分配环境下, DPCP-CROSS-JOIN 算法把 DPCP-JOIN 算法中需要通过复杂表达式计算才能进行的连接转换为交叉编码法的“等值”连接实现, 避免了连接条件之一的时间差绝对值的实时和反复计算. 计算 γ_1、γ_2 和 γ_3 的 3 个子算法相互独立, 故可通过并行提交相应 SQL, 进一步提高了资源利用率.

从表 6 可以看出, 集群为 DPCP-CROSS-JOIN 算法分配的 Map 1、Map 5、Reducer 2 和 Reducer 3 对应的任务要远多于 DP-JOIN 算法和 DPCP-JOIN 算法, 资源利用率得到了提高. 从图 8 可以看出, 任务执行期间, DPCP-CROSS-JOIN 算法所占用的集群资源相对较均衡, VCores 和内存占用都明显少于 DP-JOIN 算法和 DPCP-JOIN 算法.

表 6 3 种算法对应连接查询集群资源分配对比表

算法	Map 1	Map 5	Reducer 2	Reducer 3	Reducer 4
DP-JOIN	2.9	2.9	55.1	1 009	4.7
DPCP-JOIN	32	32	216.2	1 009	145.7
DPCP-CROSS-JOIN	96	96	1 231.1	3 027	12.3

DP-JOIN、DPCP-JOIN 和 DPCP-CROSS-JOIN 算法业务目标相同, 执行耗时比较见统计表 7.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 7　3 种算法对应连接查询时间消耗对比表

算法	额外耗时 (s)	交叉耗时 (s)	连接耗时 (s)	总耗时 (s)	连接提升至 (倍)	总体提升至 (倍)
DP-JOIN	0	0	161127	161127	–	–
DPCP-JOIN	361	0	54397	54758	2.96	2.94
DPCP-CROSS-JOIN	390	1890	5255	7535	30.66	21.38

由表 7 数据可得到以下结论.

(1) DPCP-JOIN 相比 DP-JOIN, 其性能有所提升. 实验结果数据表明其连接查询性能提升了 2.96 倍, 总体性能提升 2.94 倍.

(2) DPCP-CROSS-JOIN 相比 DP-JOIN, 其性能提升明显. 实验结果数据表明其连接查询性能提升了 30.66 倍, 总体性能提升 21.38 倍.

从图 9 可以看出, 一是处理相同规模数据连接查询时, DPCP-CROSS-JOIN 算法时间消耗最小; 二是 DPCP-CROSS-JOIN 算法在每个日期分区, 即每天数据的连接查询时间消耗波动幅度不大, 整体运行稳定.

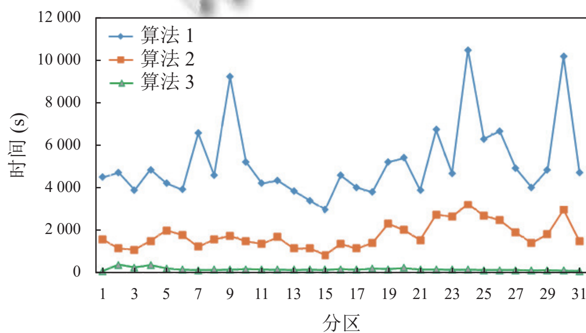

图 9　3 种算法各日期分区连接查询时间消耗对比

5.3　存在的问题

采用 DPCP-CROSS-JOIN 算法优化连接查询, 性能提升显著而且更稳定, 但也存在如下问题.

(1) 进行 DPCP-CROSS-JOIN 算法连接查询之前先要对数据进行交叉编码, 需消耗一定时间.

(2) 交叉编码后每条记录增加了 6 个编码相关字段, 会额外占用存储. 本次选择的实验数据 21.6 GB, 通过交叉编码后 47.4 GB, 存储空间增加了 1.19 倍.

(3) 该算法适用于 Hive 等行式存储的关系或者类似的其他关系数据库环境, 在图数库等非关系数据以及 HBase 等列式存储的数据库环境下适用性有待进一步验证.

6　结语

通过 DPCP-CROSS-JOIN 算法, 可实现低集群计算资源即: 低“算力”环境下大规模时空轨迹类数据同时空关系的高效连接查询, 可通过提高算法效率弥补市、县级公安机关当前存在的“算力”不足问题, 又能解决完成“全量”对象间同时空关系计算的现实需求. 通过该算法对时空类关系数据连接查询的优化, 可将 $n \cdot n$ 对象间同时空关系查询的时间复杂度从 $O(n^2)$ 降为 $O(n\log n)$. 在进行大规模数据的连接查询时, 性能稳定, 并能将连接查询效率最高提升至普通日期分区表优化查询的 30.66 倍. 在该算法实现连接查询过程中, 虽然需要增加一些额外的存储空间, 但总体性能最高提升至了 21.38 倍, 瑕不掩瑜. 这种方法可作为时空轨迹伴随查找, 不同对象间关系亲密度判定等算法的原子算法. 可广泛应用于维护国家安全、预防和打击犯罪等领域, 也可用于人员流量分析, 人员群落稳定性分析, 辅助区域城乡规划等领域. 当然本次研究主要针对的是 Hadoop 生态, Hive 数仓内的连接查询优化, 在其他数据库或者数据仓库环境里的必要性和适应性, 还需要更多学者进行更深入的研究和实践.

参考文献

1 高强, 张凤荔, 王瑞锦, 等. 轨迹大数据: 数据处理关键技术研究综述. 软件学报, 2017, 28(4): 959–992. [doi: 10.13328/j.cnki.jos.005143]

2 王家耀, 武芳, 郭建忠, 等. 时空大数据面临的挑战与机遇. 测绘科学, 2017, 42(7): 1–7. [doi: 10.16251/j.cnki.1009-2307.2017.07.001]

3 陈叶旺, 曹海露, 陈谊, 等. 面向大规模数据的 DBSCAN 加速算法综述. 计算机研究与发展, 2023, 60(9): 2028–2047.

4 赵彦荣, 王伟平, 孟丹, 等. 基于 Hadoop 的高效连接查询处理算法 CHMJ. 软件学报, 2012, 23(8): 2032–2041. [doi: 10.3724/SP.J.1001.2012.04124]

5 王华进, 黎建辉, 沈志宏, 等. 基于 ORC 元数据的 Hive Join 查询 Reducer 负载均衡方法. 计算机科学, 2018, 45(3): 160–166. [doi: 10.11896/j.issn.1002-137X.2018.03.025]

6 马东, 周帅锋, 郑伟, 等. 一种 Hive 中大小表关联的优化方法: 中国, 201710032231.4. 2021-10-19.

7 吴锦坤, 张金波, 张睿智, 等. 一种用于 Hive-SQL 执行效率的提升方法及系统: 中国, 202111611070. 2022-05-13.

8 Kulkarni A, Dharmadhikari S, Emmanuel M. Enhancing HiveQL engine using Map-Join-Reduce. Data Mining & Knowledge Engineering, 2013, 5(1): 9–12.

9 郑灵逸, 李擎. 一种基于 HiveSQL 的增加任务并行度与建立中间表组合的优化查询方法. 现代计算机, 2021, 27(36): 55–59. [doi: 10.3969/j.issn.1007-1423.2021.36.010]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

10 Margoor A, Bhosale M. Improving join reordering for large scale distributed computing. Proceedings of the 2020 IEEE International Conference on Big Data (Big Data). Atlanta: IEEE, 2020. 2812–2819. [doi: 10.1109/BigData50022.2020.9378281]
11 齐恒, 张志齐, 文瑞, 等. 大规模轨迹数据时空伴随者查询方法和系统: 中国, 202211362098.6. 2023-07-14.
12 Sahal R, Nihad M, Khafagy MH, *et al*. iHOME: Index-based join query optimization for limited big data storage. Journal of Grid Computing, 2018, 16(2): 345–380. [doi: 10.1007/s10723-018-9431-9]
13 Costa E, Costa C, Santos MY. Evaluating partitioning and bucketing strategies for Hive-based big data warehousing systems. Journal of Big Data, 2019, 6: 34. [doi: 10.1186/S40537-019-0196-1]
14 Arpitha P, Kumar PV. Efficient query optimization in data warehouse using indexing & partitioning techniques by limit clause. International Journal of Engineering Science and Computing, 2019, 9(1): 19561–19566.
15 Xia DW, Lu XN, Li HQ, *et al*. A MapReduce-based parallel frequent pattern growth algorithm for spatiotemporal association analysis of mobile trajectory big data. Complexity, 2018, 2018: Paper ID 2818251. [doi: 10.1155/2018/2818251]
16 房俊, 李冬, 郭会云, 等. 面向海量交通数据的 HBase 时空索引. 计算机应用, 2017, 37(2): 311–315. [doi: 10.11772/j.issn.1001-9081.2017.02.0311]
17 Zhao P, Rao WX, Zhang CX, *et al*. SST: Synchronized spatial-temporal trajectory similarity search. Geoinformatica, 2020, 24(4): 777–800. [doi: 10.1007/s10707-020-00405-y]
18 Jin FM, Hua W, Zhou T, *et al*. Trajectory-based spatiotemporal entity linking. IEEE Transactions on Knowledge and Data Engineering, 2022, 34(9): 4499–4513. [doi: 10.1109/TKDE.2020.3036633]
19 Qin WW, Zhang MF, Li W, *et al*. Spatiotemporal K-nearest neighbors algorithm and Bayesian approach for estimating urban link travel time distribution from sparse GPS trajectories. IEEE Intelligent Transportation Systems Magazine, 2023, 15(6): 152–176. [doi: 10.1109/MITS.2023.3296331]
20 王晨旭, 汪谨权, 杨鑫. 基于二级时空分桶的伴随轨迹查询. 计算机学报, 2024, 47(1): 131–147.
21 Dwivedi M, Agrawal D. Performance optimization of Hive by two-way join. International Journal of Modern Engineering and Research Technology, 2017, 4(4): 14–20.
22 Kadari P, Potluri A, Sristy NB, *et al*. Skew aware partitioning techniques for multi-way spatial join. Proceedings of the 7th International Conference on Mining Intelligence and Knowledge Exploration. Goa: Springer, 2019. 52–61.
23 何文婷, 程学旗, 郑天祺, 等. 一种非等值关联子查询的优化方法和系统: 中国, 201810097136.7. 2020-12-25.
24 陈喜洲. 一种基于业务特征优化 Hive 中两个大表不等值关联的方法. 广东通信技术, 2017, 37(11): 52–55. [doi: 10.3969/j.issn.1006-6403.2017.11.011]

(校对责编: 张重毅)

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn