第54卷第3期 南京航空航天大学学报 Vol. 54 No. 3
2022年6月 Journal of Nanjing University of Aeronautics & Astronautics Jun. 2022

DOI:10.16356/j.1005-2615.2022.03.020

基于轨迹大数据时空分布的索引与查询方法

李征宇[1,2],赵卓峰[1,2]

(1.北方工业大学信息学院,北京 100144;
2.大规模流数据集成与分析技术北京市重点实验室(北方工业大学),北京 100144)

摘要: 由于移动对象自身行为特征和整体规律的不同,使得其产生的轨迹数据具有较大的时空分布不均特点,从而影响轨迹数据索引和查询的效率。针对现有轨迹数据索引方法很少考虑轨迹数据分布不均特性的情况,提出了一种基于历史数据预分区的时空索引方法,其借助轨迹数据时空维度上分布的相似性,首先在空间上根据数据分布情况对Geohash编码进行预分区,进而建立轨迹数据的索引结构和基于HBase的存储模型,并利用该索引结构设计了基于Geohash分区的查询分解算法。基于真实出租车轨迹数据集的实验表明,相较于均匀划分的扩展的HGrid方法与混合编码的ST-hash方法,本文提出的索引结构及其查询方法可以有效提升海量具有不均匀特征轨迹数据的时空查询性能,并且可以在保证查询结果准确性的同时,最大限度地减少子查询的数量。

关键词: 轨迹数据;时空索引;HBase;时空范围查询

中图分类号: TP311 **文献标志码:** A **文章编号:** 1005-2615(2022)03-0528-09

Index and Query Method Based on Spatial-Temporal Distribution of Trajectory Big Data

LI Zhengyu[1,2], *ZHAO Zhuofeng*[1,2]

(1. School of Information, North China University of Technolgy, Beijing 100144, China; 2. Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data (North China University of Technolgy), Beijing 100144, China)

Abstract: Due to the diversified behavioral characteristics and regular pattern of moving objects, the trajectory data generated by these objects shows obvious uneven distribution feature in time and space, which may lead to worse performance for trajectory data indexing and querying. However, the existing trajectory data indexing methods rarely consider this problem. In this paper, a temporal-spatial distribution based indexing and querying method is proposed. In the method, Geohash code is introduced and pre-partitioned spatially by utilizing the temporal-spatial similarity of the trajectory data distribution. Then, we use the pre-partitioned Geohash code, partition number and the trajectory data timestamp to compose the index structure. With this index structure, a storage model based on HBase and a query algorithm based on Geohash partition are designed respectively. The empirical study using real trajectory dataset shows that the method improves the spatiotemporal query performance of trajectory data by comparing with the Extend_HGrid and ST-hash methods, and effectively reduces the number of sub-queries during query.

Key words: trajectory data; temporal-spatial index; HBase; temporal-spatial range query

随着定位技术的广泛应用,轨迹数据成为近年来一类典型的大数据。轨迹数据涵盖的领域非常广泛,其中包括人类出行轨迹、自然现象移动轨迹和动物迁徙轨迹等[1]。海量高质的轨迹数据中

基金项目: 北京市自然科学基金(4202021)。
收稿日期: 2021-10-10;**修订日期:** 2022-01-23
通信作者: 赵卓峰,男,研究员,E-mail:edzhao@ncut.edu.cn。

引用格式: 李征宇,赵卓峰. 基于轨迹大数据时空分布的索引与查询方法[J]. 南京航空航天大学学报,2022,54(3):528-536. LI Zhengyu, ZHAO Zhuofeng. Index and query method based on spatial-temporal distribution of trajectory big data [J]. Journal of Nanjing University of Aeronautics & Astronautics, 2022, 54(3):528-536.

蕴含着丰富的信息，通过对其进行深入的分析与研究，可以掌握人类活动和迁移的规律，分析交通、大气环境的移动特征[2]，其结果可以应用到城市规划[3]、路线推荐[4]和城市热点区域发掘[5]等方面，推动其他各个领域的发展。以Hadoop为代表的大数据技术被用来支持轨迹数据的存储和查询，然而直接使用Hadoop等大数据技术将轨迹数据存储于分布式集群中，只解决了海量轨迹数据的存储问题，并不能满足对海量轨迹数据的高效查询需求。例如：只是简单将数据存储在分布式文件系统上，或是不做任何处理地将数据存储于NOSQL数据库中，做局部的查询与分析工作就意味着对整体存储做一次全扫描，耗时长、资源占用率高等问题就会凸显，从而导致整个任务流程效率低下。因此，基于分布式存储系统与NOSQL数据库，建立高效的索引结构与查询策略来提高存储和查询效率，是当前存储与查询轨迹数据的主流方法。传统小规模轨迹数据的索引策略主要分为基于R-Tree方法与基于网格方法[2]。基于R-Tree的方法随着存储轨迹数据时间跨度的增长与数据量的增加，不同的3D框之间重叠变得频繁，导致查询效率下降，不适合在分布式大数据环境下进行海量轨迹数据的存储和查询；基于网格的方法是将空间划分成均匀的网格并加上时间维，查询时将查询任务转换成覆盖不同网格的子查询。它将所有的网格等价看待，没有充分考虑时空轨迹数据在时空间分布上不均匀的特点。本文设计并实现了一种基于历史数据预分区的轨迹数据索引结构，用于加速海量不均匀轨迹数据的轨迹查询，在非关系型数据库HBase中设计了存储模型并设计了基于此索引的查询优化算法。索引结构利用Geohash编码，借助了编码的空间邻近性，首先在历史数据集上随机抽取轨迹数据用于构建不均匀的分区，根据Geohash数量分布情况对Geohash编码进行合并，并生成Geohash的倒排索引，使得Geohash编码与分区一一对应。在此基础上，本文设计了基于HBase的轨迹存储模型以及轨迹数据查询算法——首先在倒排索引中检索查询范围覆盖的分区情况，再依次遍历分区，在分区内完成子查询的分割操作，最终完成时空轨迹数据的查询任务。

1　相关工作

现有的轨迹数据索引按构造方法也可分为两类：数据驱动方法与空间驱动方法[6]。

数据驱动方法主要是围绕以数据为中心，随着数据导入的过程动态的生成索引结构。普遍是基于R树方法的变体或拓展，例如基于轨迹建立的3D-Rtree方法[7]，通过构建时空R树的方法建立索引，将查询问题转化为三维查询立方体，但是这种方法随着数据量的增加，查询效率急剧下降。尽管后来提出的ST-R-tree和TB-tree[8]方法提出了解决此问题的新思路，但是随着存储轨迹数据时间的增加，索引框之间重叠的问题依然存在。文献[9]提出的Rt-Tree索引方式和文献[10]提出的HR-Tree和H＋R-Tree索引方法使用的都是多版本R树方法。对于给定的时空查询，这类索引首先判断查询窗口覆盖哪些时间段，然后从这些时间段对应的每个空间索引中检索查询范围相交的轨迹点。基于R树构建的索引结构相对复杂，R树的插入与节点分裂代价相对较大，需要动态地调整索引结构。

另一类空间驱动方法的思路是先将地理空间分为网格的形式，然后将落入每个网格的内容再依时间进行索引，达到同时索引时间维和空间维的目的。编码方式主要以常规网格编码的方式实现，按行、按列将空间有序地划分。随着空间填充曲线领域的发展，2种可以保持空间邻近性的空间填充方法应用比较广泛，即Z曲线[11]与Hilbert曲线[12]。Hughes等介绍了一种时空数据存储与检索引擎Geomesa[13]，其索引部分就是利用Z曲线进行Base32编码得到的Geohash实现，可以用于索引时空轨迹数据。JUST[14]是京东城市时空数据引擎，该引擎在Geomesa的基础上提出了Trajmesa[15]存储引擎，拓展了Z2T和XZ2T两种索引方式，解决时间与空间维度不匹配的问题。ST-Hash索引方法[16]将经度、纬度、时间3种属性进行统一编码，使地理位置邻近、时间相近的数据，物理存储的位置相近。相比较于Z曲线，Hilbert曲线映射步骤较多相对复杂，但数据的空间聚集性比Z曲线更优。文献[17-18]使用Hilbert曲线对空间进行编码，同时再加入时间属性构造索引。

数据驱动与空间驱动索引方法是从索引的形式上考虑的，面对海量轨迹数据的存储与查询需求，数据驱动的索引方法通常并不高效[19-20]。由于轨迹数据通常与路网、城市热点区域以及由此带来的移动对象出行特征密切相关，所以海量的轨迹数据通常聚集在城市热点区域以及路网范围内，但是空间驱动索引方法是将空间中的所有区域进行等分，这就造成了有的空间区域中数据量庞大，有的空间区域中数据量较少甚至没有数据，使得有的子查询需要耗时较长，有的子查询中并没有数据。

针对这种轨迹数据的分布不均匀问题，现有解决方法主要依靠数据的动态划分进行解决。文献[21]中提出了1种基于ST-hash的LPST-Hash的索引方案，设置了Regin分裂的策略，当Region中数据量达到了Region分裂阈值，使Region以某一个键值作为分界线分裂，同时，Region对应的索引树也应该和Region一样一分为二。文献[22]中，基于Geohash和R树，提出了1种2层时空索引-GRIST，通过改进原有的Geohash的编码及映射算法，设计了根据数据分布自适应分裂的方法，利用R-tree存储不均匀的Geohash网格上的数据。动态划分法虽然可以有效地解决数据分布不均匀的问题，但是由于数据分区是随着数据导入时动态生成的，分区过程会花费较长时间，同时已经生成的索引结构也要动态的更改；可能会存在数据存储完成但数据依然分布不均匀的问题。本文的方法利用了历史数据得到轨迹数据的时空分布规律，再利用规律构建分区的方法，解决不均匀轨迹数据的存储与查询问题。

2 基于历史数据预分区的索引方法

轨迹数据Traj由一系列轨迹点构成，可以表示为集合Tra=$\{P_1,P_2,\cdots,P_n\}$，轨迹点$P_i=(\mathrm{lon}_i,\mathrm{lat}_i,t_i)$，$0\leqslant i\leqslant n$，其中经度用lon表示，纬度用lat表示，$t$表示此坐标位置的时间。Traj可以表示在三维坐标系中，其中两维表示为地理空间-经度与纬度，第三维用于表示时间。

本文索引的生成利用了Geohash编码。Geohash是一种用于处理空间数据的高效的索引方法，通过对地理信息的编码，将二维的经纬度降维成一维字符串，Geohash编码代表的并不是1个点，而是1个矩形区域。对其加入时间维可以用于索引时空轨迹数据，在使用划分得均匀的立方体存储数据时，查询问题可转化为对查询立方体所覆盖的Geohash立方体进行遍历，产生大量的子查询任务。如图1所示，对于不均匀的数据分布，这些查询任务可能相差悬殊(每个Geohash立方体中的数据可能相差悬殊)；对于查询范围远大于单位立方体的查询需求时，查询会转换成为大量的子查询，影响查询性能。

区别于只利用Geohash编码构建的时空数据索引方法，本文索引方法的核心是利用历史轨迹数据提前得到轨迹数据的分布规律，在索引构建的过程利用此规律，将Geohash编码分区按历史数据分布进行合并，即通过阈值的方法对其进行合并，这样在查询时本文方法的子查询数量将会小于只利用Geohash编码的索引方法，并在查询时对分区子查询进行分割，避免扩大查询区域。通过这2种方式，有效地减少子查询个数，提升查询效率，并适当的平衡了任务规模。本文索引的构建过程如图2所示。

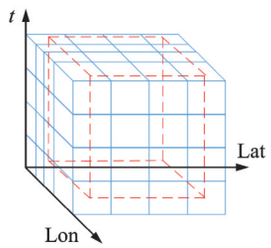

图1 Geohash立方体查询示例

Fig.1 Example of Geohash cube query

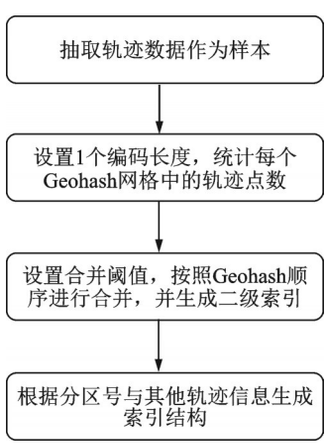

图2 索引构建过程

Fig.2 Process of building index

2.1 基于历史数据的分区生成

索引设计的首要步骤是利用历史数据对Geohash编码进行分区，所以历史数据的数据分布情况对索引的设计至关重要。图3中分别给出了北京市出租车轨迹数据集中10月8日与10月9日两日的数据分布情况，通过观察发现，两日的数据分布情况基本相似，即轨迹数据总体分布总是在路网附近，由此可以利用数据在空间上的分布特征，构建基于历史数据的分区，使其服务于时空轨迹数据的存储与检索。

图3从宏观的角度展示出在不同日期上的数据空间分布情况。若要利用历史轨迹数据提炼出信息，还需要判断不同日期的Geohash上所包含的数据量是否相似，图4给出了不同日期Geohash上轨迹点数据量的部分数据，其中$(\mathrm{Geohash})_{\mathrm{D}}$是将Geohash编码转换成十进制，Num表示的是每日在Geohash上数据的量，图4中绘制了10月8日～10月12日的部分数据情况，其中Avg是这几日数据

图 3　两日数据分布对比

Fig.3　Comparison of two-day data distribution

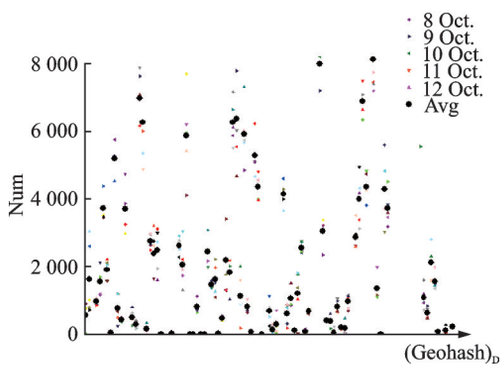

图 4　5 天 Geohash 数据分布与均值

Fig.4　Five-day Geohash distribution and Avg

的平均值，可以观察到这 5 天的每个 Geohash 中的数据量波动不大，属于同一数量级，所以使用了 Geohash 的平均值表示每个 Geohash 网格的数据特征。

利用设置分区合并阈值的方法完成分区的合并，即合并后所生成的 Geohash 组，其所包含的数据和应大于设置的阈值。合并顺序按照 Geohash 生成顺序，如果有的网格中没有数据，则跳过这一网格；如果将当前 Geohash 网格中的数据加入后，分区中的总数据量大于所设定的阈值，即分区结束，下一条数据开始新的分区，图 5 举例展示了分区的划分情况，其中 01、02、03、04 为 4 个分区，每个分区中包含一个或多个 Geohash 区域。

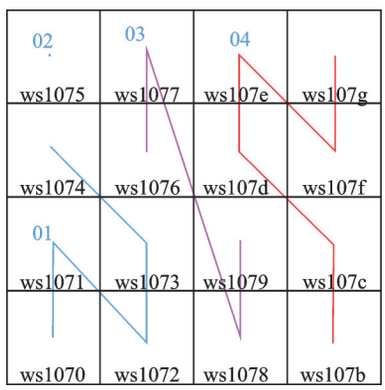

图 5　分区划分示例

Fig.5　Example of dividing partition

表 1 中给出了分区划分结果示例，表结构体现出了分区号与 Geohash 组的映射关系，1 个分区中包含 1 个或多个同一编码长度的 Geohash 编码，每个 Geohash 编码都对应 1 个分区号。分区阈值设定为 5 000，其中 Partition Id 代表的是分区的自增序号，Partition Number 表示的是分区内的数据量，Included Geohash 表示的是分区内包含的 Geohash 编码。

表 1　分区划分的结果示例

Table 1　Result of dividing partition

Partition Id	Partition number	Included Geohash
0	5 811	wx407v wx407w wx407x wx40e8 wx40eb wx40qj wx40qn wx40qq wx40qr
1	5 412	wx42fz wx42g1 wx42g4 wx42g5 wx42g7 wx42gd
2	5 125	wx430h wx430k wx430m wx430q
⋮	⋮	⋮

2.2　辅助二级索引

索引结构的主要设计思想体现在通过设置阈值，对 Geohash 进行分区，将包含数据量少的 Geohash 分区进行合并，从而减少查询生成的子查询数量。对于给定的经纬度坐标 $P=(\mathrm{lon},\mathrm{lat})$，通过 Geohash 算法可以很容易地得到 Geohash 编码，但由于 Partition Id 与 Geohash 编码可能是一对一关系，也可能是一对多的关系，所以根据如表 1 所示的数据结构，不能直接根据 Geohash 编码得到 Partition Id，所以需要 1 个 Geohash 与 Partition Id 对应的倒排索引作为二级索引，来服务于数据的存储与查询。

二级索引的构建如表 2 结构所示。利用 Geohash 编码与分区号进行对应，通过 Geohash 编码值可以直接获取分区号，在数据存储时，可以通过二级索引快速地查询到 Geohash 编码所对应的分区号；同理，在查询时，通过查询范围的解析得到

表 2　二级索引结构

Table2　Secondary index structure

Geohash Code	Partition Id
wx43m5	15
wx43m6	15
wx43m7	15
wx43m9	16

查询范围所覆盖的Geohash的编码，再通过二级索引可以检索到这些Geohash编码属于哪个分区。

3 基于HBase的轨迹数据索引存储模型

与许多轨迹管理系统类似，如文献[15]中提出的垂直存储模式，本文方法也是利用数据库中1行存取1个轨迹点数据，每1个轨迹点数据包括如下几个属性。

(1)分区号：利用空间属性与轨迹数据分布生成的自增序号。

(2)空间属性：包括本轨迹点的经度、纬度。

(3)时间属性：轨迹点的产生时间。

(4)其他属性：移动对象速度等其他描述轨迹点的信息。

数据存储模型的设计对提升查询效率有至关重要的作用，由于HBase中不支持设置二级索引结构，HBase中的行键即为数据表的直接索引，所以将轨迹数据索引直接设计为行键。HBase的存储模型如表3所示，其中TR表示时间尺度，本文处设计为1天。T表示尺度内的具体时间。本文的设计既保证了属于同一分区的Geohash在物理位置上邻近，也保证了分区内相邻时间T的轨迹点在物理上也相对邻近。

表3 数据存储模型

Table 3 Data storage model

行键	对象Id	经度	纬度	速度
Partition Id+TR+Geohash+T+Oid	Oid	Lon	Lat	Speed

图6描述了数据存入HBase数据库的具体流程，其中需要的参数包括在上文中构建的不均匀网格分区的二级索引Geohash Map，即Geohash与Partition Id组成的倒排索引，用于获取当前轨迹点的Geohash所在的分区；轨迹数据为需要入库的原始数据，从中可以分离出当前轨迹点的详细信息，如经度、纬度、时间和速度等；Geohash编码长度应与前文设置的Geohash编码长度保持一致，存储的数据才能在查询时与查询任务相匹配。本文利用Hash Map结构存储二级索引，通过Key便可直接查找到对应的value值，故本算法的时间复杂度为$o(n)$，在空间消耗方面主要受到Geohash编码长度影响，编码长度越长，其所代表的空间区域就越小，在一定的空间范围内，包含的编码就会更多。

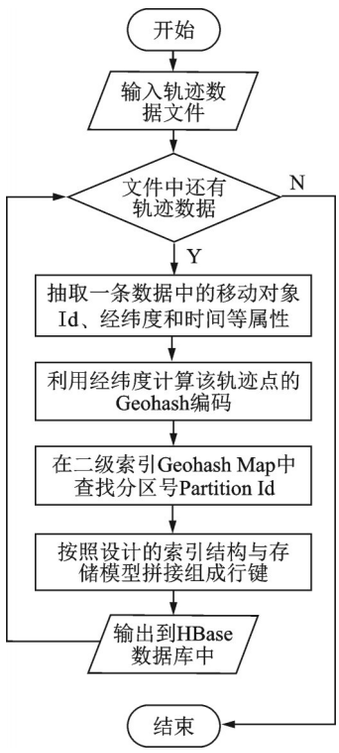

图6 轨迹数据入库流程

Fig.6 Process of importing trajectory data into HBase

4 轨迹数据时空查询

本文提出的索引方法主要服务于轨迹数据的时空查询，定义时空查询为Q_{st}，给定空间范围$[Geo_{start}, Geo_{end}]$，其中$Geo_x=(lon_x, lat_x)$；给定时间范围定义为$[T_{start}, T_{end}]$，对于给定时空范围条件，查询出的符合条件的轨迹点的集合定义为Result，$Result=Q_{st}(Geo_{start}, Geo_{end}, T_{start}, T_{end})$，不同于对HGrid[6]拓展的索引方式，本文提出的索引方法在时空查询时需要设计1个额外的查询算法来支持。由上文可知，此时已经获得了不均匀轨迹数据的Geohash分区信息，并根据分区信息将数据集中的轨迹数据依据要求存入了HBase数据库。分区的目的就是优化查询效率，扩展的HGrid方法先是利用空间维进行查询，查询哪些Geohash编码被查询条件覆盖，再依次遍历每1个Geohash，利用时间维与精确的位置信息条件进行筛选。由于引入了分区的概念，本文方法使用查询条件向分区进行映射，得到查询范围所对应的分区，再在分区内划分查询范围具体覆盖的区域，在数据分布不均匀时，生成的子查询数量会少于前者。

如图7所示，01、02、03、04为分区号，例如分区01={ws1070, ws1071, ws1072, ws1073, ws1074}，QueryWindow1和QueryWindow2分别为查询窗口。对于QueryWindow1来说，QueryWindow1覆盖的Geohash编码同属于01分区，其中包括

ws1070、ws1071 和 ws1074，由于 ws1070 和 ws1071 相邻，所以 QueryWindow1 可以生成 2 个子查询，即 01[ws1070，ws1071]和 01[ws1074]，相较于扩展的 HGrid 方法，在总体查询数据不变的情况下，子查询个数由 3 个减少为 2 个；对于 QueryWindow2，即整个区域都为查询范围，依据本文设计的查询方法 QueryWindow2 可生成 4 个子查询，为 01[ws1070，ws1071，ws1072，ws1073，ws1074]、02[w s1075]、03[ws1076，ws1077，ws1078，ws1079]和 04 [ws107b，ws107c，ws107d，ws107e，ws107f，ws107g]，相较于扩展的 HGrid 方法，在总扫描数据不变的情况下，子查询个数由 16 个减少为 4 个，查询效率可以得到显著提升。图 8 给出了轨迹时空查询的详细分解与执行过程，在时间复杂度方面，由于需要对每 1 个 partition 进行遍历，同时每个 partition 中还包含多个 Geohash 编码，查询方法的时间复杂度为 $o(n^2)$，空间复杂度同数据入库时相同，取决于 Geohash 编码长度。

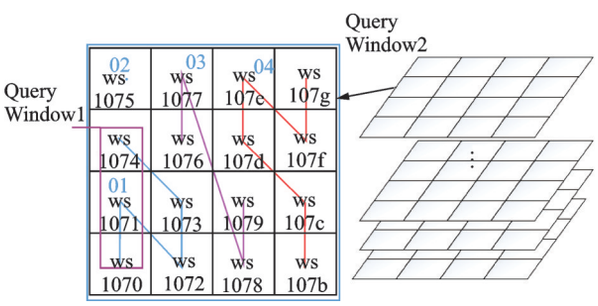

图 7　查询算法示意图

Fig.7　Example of query algorithm

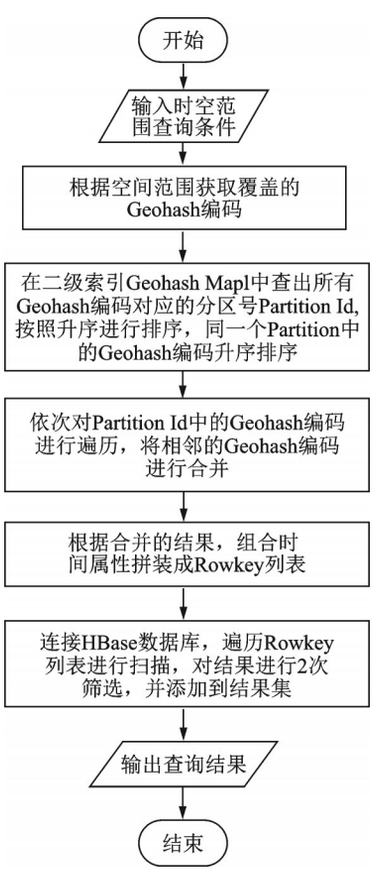

图 8　轨迹数据时空查询流程

Fig.8　Spatio-temporal query process of trajectory data

5　实验评价

为了测试本文提出的索引方法的查询响应情况，基于 NoSQL 的 HBase 数据库实现了数据持久化，借助 HBase Java api 实现了数据的分布式查询算法完成数据查询。

5.1　实验准备

实验环境采用 Hadoop 完全分布式集群环境，集群由 3 个节点组成，其中 1 个 Master 节点部署 NameNode，2 个 Slave 节点部署 DataNode，软件环境：节点系统为 centos7，集群环境安装有 Hadoop-2.7.1、HBase-1.3.6 和 Zookeeper-3.4.14；硬件环境：每个节点拥有 16 GB 内存、2 TB 硬盘和两颗 Intel E5-2620 0 @ 2.00 GHzCPU。

本文实验的轨迹数据采用浮动车轨迹数据，来源于北京市 2012 年 10 月份的出租车真实数据。实验采用 3 种不同规模的数据集，分别为 2 500 万级、5 000 万级和 1 亿级，其中 2 500 万级为 2012 年 10 月 15 日 1 天的数据，5 000 万级是 2012 年 10 月 15、16 日 2 天的数据，1 亿级是 2012 年 10 月 15 日～2012 年 10 月 19 日的数据。

5.2　实验设计

本部分设计了 3 个实验。实验 1 对比了本文提出的索引与其他索引在子查询生成速度与子查询生成个数的异同，并在不同数据规模、不同的查询范围下，比较了本文提出的索引方法与其他索引方法的查询响应时间，还比较了内存消耗。由于在预分区时使用了历史数据，实验 2 探究了使用不同规模的历史数据对查询响应时间的影响。实验 3 探究了不同的分区阈值对查询响应时间与导入时间的影响。本文简要介绍 2 种对比的索引方法，其中对 HGrid[6] 拓展时间维度的索引方法是空间驱动类索引方法中的典型方法；ST-hash[16] 索引方法是对经度、纬度和时间维进行混合编码，Geomesa 中的 Z3 索引使用了此方法，本文实现的每一维度都使用 14 bit 进行编码，再对 3 个维度进行混合编码，最后使用 Base64 编码方法将 bit 数组编码成 7 位字符串，ST-hash[16] 生成实例如图 9 所示。

5.2.1　索引方法比较

随机设定某空间范围 0.3°×0.3°，时间范围为 10 月 15 日，本节实验首先探究不同索引方式的子查询生成时间以及子查询生成个数，如图 10 所

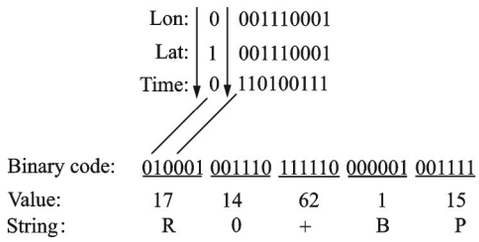

图9 ST-hash编码方式

Fig.9 Example of ST-hash encoding

示。在给定的时间和空间范围下,由于需要在二级索引上查找分区,所以本文方法子查询生成时间最长,由于完成了对子查询按照分区的合并,所以子查询个数最少。本文构建索引方法与其他2种方法相同,构建的都是聚集索引。构建索引的过程就是将数据按照设计好的模型存入数据库中,3种方法都是以轨迹点的方式进行存储,所以构建时间与对比的两种方法大致相同,占用的空间情况也类似。

图11在不同的数据规模与不同空间范围下完

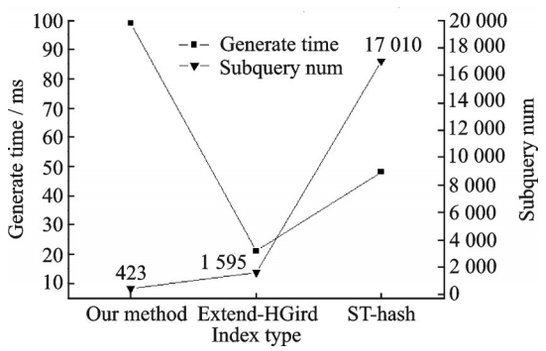

图10 不同索引间对比

Fig.10 Comparison between different indexes

成了实验,其中的分区阈值选为5 000,实验数据规模从2 500万级扩展到1亿级,空间查询范围由0.1°×0.1°扩展到0.3°×0.3°,时间范围与上文保持一致,进行了3次重复实验将平均值绘制在图中。由图中的结果分析可知,本文提出的索引结构配合查询优化算法在不同的数据规模下均有较好的时间响应;在导入不同数据量级的轨迹数据时,查询响应时间变化不大。

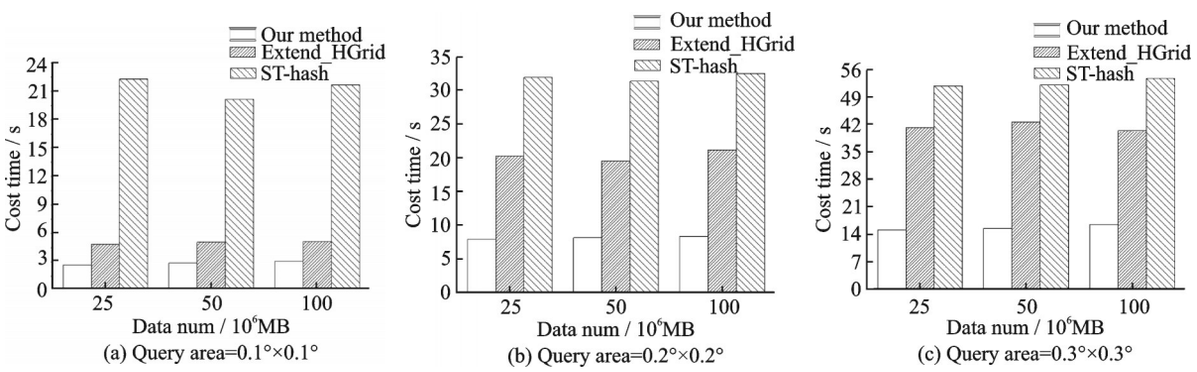

图11 不同数据规模与空间范围下的查询响应时间

Fig.11 Time performance of different data size and spatial scope

图12对比了本文方法与另外2种方法的内存代价,实验数据规模采用2 500万级,分区阈值与时空查询范围与上文保持一致时,内存代价比较的是执行查询时集群占用的内存资源最大值情况,在集群中搭建了Ganglia分布式监控系统监控集群的

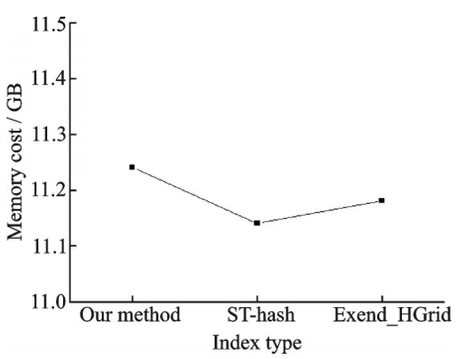

图12 不同索引结构的内存代价

Fig.12 Memory cost of different index structures

内存使用情况。从图中可以得出,由于本文方法需要在内存中预先缓存分区结果,所以相比较于其他2种方法,在内存占用资源稍多;在I/O代价方面,由于3种数据全部存储在HBase数据库中,I/O代价可以等价为不同索引的查询方法对HBase数据库的查询次数。由图10可知,子查询个数为ST-hash>Extend_HGrid>Our method,子查询数量越多,查询方法访问HBase数据库的次数就越多,I/O代价越高,所以本文的I/O代价最小。

5.2.2 不同规模历史数据对查询响应时间的影响

由于使用历史数据完成了预分区的生成,本节实验探究了使用不同规模的历史数据生成的索引结构对查询响应时间的影响。实验分别选取了1天、2天、4天和8天的历史数据,利用各自生成的预分区结果分别将轨迹数据导入HBase数据库。

选取的导入数据为2012年10月18日2 500万量级的数据，空间查询范围与上文保持一致，时间范围为10月18日，查询响应时间如图13所示。随着选用的历史数据的天数增加，查询响应时间变化不大，说明生成预分区的历史数据天数对查询响应时间影响不大，同时也从侧面验证了轨迹数据分布不均匀的特性在相同时间尺度下的相似性。

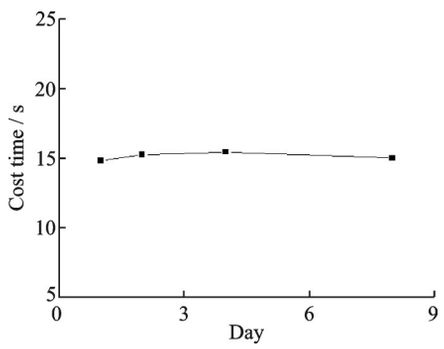

图13　不同规模历史数据预分区结果的查询响应时间

Fig.13　Time performance of different scale historical data pre-partition

5.2.3　不同的分区阈值对查询的影响

本节实验的合并阈值范围设置从1 000变化到20 000，其中包括1 000、3 000、5 000、10 000和20 000，测试数据采用的是2012年10月15日2 500万量级的数据，随机设定空间查询范围设置为[115.877 951，39.830 862 116.305 688，39.608 864]，时间范围为10月15日，数据的导入以及查询情况如图14所示。从图中可以分析出，轨迹数据的导入速度与阈值设定的大小无关。对于不同阈值下的查询操作，查询响应时间呈现出先减少再稳定的趋势，其主要原因是当查询阈值较小时，查询范围所覆盖的分区数较多，使响应时间变慢，随着分区阈值的增大，查询范围覆盖的分区逐渐减少，响应速度变快，但随着分区阈值的继续增大，查询范围覆盖的分区总数虽然减小，但是查询方法还会对分区内Geohash编码区域进行合并，所以子查询数量变化不大，查询响应时间趋于稳定。

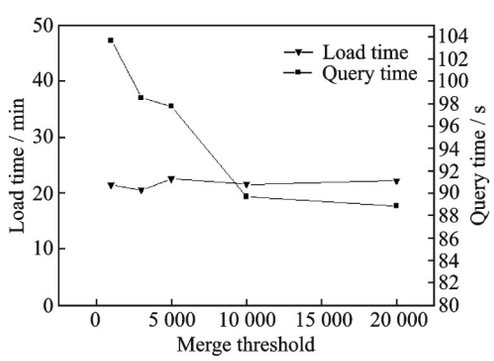

图14　不同分区阈值的导入和查询响应时间

Fig.14　Results of different threshold experiments

6　结　　论

针对海量的时空轨迹数据分布不均匀的特性，本文提出了一种索引方法，并详细地介绍了该索引的构建过程和与其相匹配的查询算法。方法首先通过设计阈值的方法对Geohash编码进行合并生成分区，设计了基于HBase的轨迹数据索引存储模型，优化了查询时分区内的子查询，有效地减少的子查询数量。通过实验证明，本文的索引方法在时空查询方面优于ST-hash方法与扩展HGrid的方法。下一步还将在以下几个方面改进，首先是支持更多种类的轨迹查询，如KNN查询、路径查询等；其次是由于构建的分区依赖于历史抽样数据，本文的抽样数据与实验数据为工作日数据，需要探究在非工作日时，构建的分区还是否有效；再者就是轨迹数据分布不均匀的特性是随时间的变化而变化的，需要探究生成分区的有效周期，是否需要动态变化、变化频率以及变化带来的开销等问题。

参考文献：

[1] 许佳捷，郑凯，池明旻，等. 轨迹大数据：数据、应用与技术现状[J]. 通信学报，2015，36(12)：97-105.
XU Jiajie，ZHENG Kai，CHI Mingmin，et al. Trajectory big data：Data，applications and techniques[J]. Journal on Communications，2015，36(12)：97-105.

[2] ZHENG Y. Trajectory data mining：An overview[J]. ACM Transactions on Intelligent Systems and Technology (TIST)，2015，6(3)：1-41.

[3] YUAN J，ZHENG Y，XIE X. Discovering regions of different functions in a city using human mobility and POIs[C]//Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining-KDD'12. Beijing，China：ACM Press，2012：186-194.

[4] ZHENG Y，XIE X，MA W Y. GeoLife：A collaborative social networking service among user，location and trajectory[J]. IEEE Data Eng Bull，2010，33(2)：32-39.

[5] 夏冬. 基于RFID电子车牌数据的城市热点区域发现[D]. 重庆：重庆大学，2018.
XIA Dong. Discovery of urban hot spots based on RFID electronic license plate data[D]. Chongqing：Chongqing University，2018.

[6] HAN D，STROULIA E. Hgrid：A data model for large geospatial data sets in HBase[C]//Proceedings of 2013 IEEE Sixth International Conference on Cloud Computing.[S.l.]：IEEE，2013：910-917.

[7] THEODORIDIS Y，VAZIRGIANNIS M，SELLIS T. Spatio-temporal indexing for large multimedia applications[C]//Proceedings of the Third IEEE Interna-

tional Conference on Multimedia Computing and Systems. [S.l.]:IEEE, 1996: 441-448.

[8] PFOSER D, JENSEN C S, THEODORIDIS Y. Novel approaches to the indexing of moving object trajectories[C]//Proceedings of International Conference on Very Large Data Bases. Cario, Egypt: the VLDB Endownmen, 2000: 395-406.

[9] XU X, HAN J, LU W. RT-tree: An improved R-tree indexing structure for temporal spatial databases [C]//Proceedings of The International Symposium on Spatial Data Handling (SDH). Zurich, Switzerland: Department of Geography University of Zurich, 2017: 1040-1049.

[10] TAO Y, PAPADIAS D. Efficient historical R-trees [C]//Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management(SSDBM 2001). [S.l.]:IEEE, 2001: 223-232.

[11] HAVERKORT H, VAN WALDERVEEN F. Locality and bounding-box quality of two-dimensional space-filling curves[C]//Proceedings of Algorithms—ESA 2008. Berlin, Heidelberg: Springer, 2008: 515-527.

[12] MOON B, JAGADISH H V, FALOUTSOS C, et al. Analysis of the clustering properties of the Hilbert space-filling curve[J]. IEEE Transactions on Knowledge and Data Engineering, 2001, 13(1): 124-141.

[13] HUGHES J N, ANNEX A, EICHELBERGER C N, et al. Geomesa: A distributed architecture for spatio-temporal fusion[C]//Proceedings of Geospatial Informatics, Fusion, and Motion Video Analytics V. [S.l.]: International Society for Optics and Photonics, 2015: 94730F.

[14] LI R, HE H, WANG R, et al. Just: Jd urban spatio-temporal data engine[C]//Proceedings of 2020 IEEE 36th International Conference on Data Engineering (ICDE). [S.l.]:IEEE, 2020: 1558-1569.

[15] LI R, HE H, WANG R, et al. Trajmesa: A distributed nosql storage engine for big trajectory data[C]// Proceedings of 2020 IEEE 36th International Conference on Data Engineering (ICDE). [S.l.]: IEEE, 2020: 2002-2005.

[16] GUAN X, BO C, LI Z, et al. ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database[C]//Proceedings of 2017 25th International Conference on Geoinformatics. Buffalo, NY, USA:IEEE, 2017: 1-7.

[17] UDDIN R, RAVISHANKAR C V, TSOTRAS V J. Indexing moving object trajectories with Hilbert curves [C]//Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. New York, NY, USA: Association for Computing Machinery, 2018: 416-419.

[18] WU Y, CAO X, AN Z. A Spatiotemporal trajectory data index based on the hilbert curve code[J]. IOP Conference Series: Earth and Environmental Science, 2020, 502: 012005.

[19] CHEN S, OOI B C, TAN K L, et al. ST^{2}B-tree: A self-tunable spatio-temporal b$^+$-tree index for moving objects[C]//Proceedings of the 2008 ACM SIGMOD International Conference on Management of data. New York, NY, USA: Association for Computing Machinery, 2008: 29-42.

[20] GUO S, HUANG Z, JAGADISH H V, et al. Relaxed space bounding for moving objects: A case for the buddy tree[J]. ACM SIGMOD Record, 2006, 35(4): 24-29.

[21] 邱峰. 基于HBase的时空数据的存储与查询技术研究[D]. 西安:西安电子科技大学, 2019.
QIU Feng. Research on storage and query technology of spatio-temporal data based on HBase[D]. Xi'an:Xidian University, 2019

[22] 赵馨逸, 黄向东, 乔嘉林, 等. 基于不均匀空间划分和R树的时空索引[J]. 计算机研究与发展, 2019, 56(3): 666-676.
ZHAO Xinyi, HUANG Xiangdong, QIAO Jialin, et al. Spatiotemporal index based on uneven space partition and R-tree[J]. Journal of Computer Research and Development, 2019, 56(3): 666-676.

(编辑:刘彦东)