面向时空数据流的移动对象空间索引构建

杨良怀,沈东海,范玉雷,高 楠
(浙江工业大学计算机科学与技术学院,浙江杭州 310023)

摘 要: 本文针对时空数据流提出了一种基于时间窗口数据排序和批量装载的移动对象空间索引构建方法HSTRCL.该方法用固定长度的时间窗口将连续的时空数据流进行切分,每当一个时间窗口完成数据缓存,采用优化的索引批量装载技术,从传统的构建流程中尽可能分离出耗时的数据划分和排序操作,将数据流的接收及其他构建操作并行执行,避免不必要的加锁同步开销,加快索引的构建效率;同时,采用基于Hash和STR的主、辅索引构建技术,满足高性能且多样化的查询需求.另外,为进一步提高对象查询性能,引入聚合技术划分对象,提出了一种基于时间窗口对象聚合和批量装载的移动对象空间索引构建方法OAHSTRCL,对象查询时间约为HSTRCL的65%,但对空间查询性能会有一定程度的影响.通过理论分析和多种实验验证了所提方法的有效性.

关键词: 时空数据流;移动对象;空间索引;R树;对象聚合
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2021)05-0992-09
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200300

A Moving Object Spatial Index for Spatio-Temporal Data Stream

YANG Liang-huai,SHEN Dong-hai,FAN Yu-lei,GAO Nan
(*School of Computer Science and Technology,Zhejiang University of Technology,Hangzhou,Zhejiang* 310023,*China*)

Abstract: In light of the characteristics of spatio-temporal data stream,we propose a moving object spatial index construction method called HSTRCL,which is based on time window data sorting and bulk loading. It segments the continuous spatio-temporal data stream with fixed-length time windows; after finishing caching the data of a time window,by combining parallel processing and optimized bulk loading technology,we isolate as much as possible the time-consuming work of data partitioning and sorting operations from the traditional build process,and parallize them with the reception of data streams and other build operations. Furthermore,we avoid unnecessary locking synchronization overhead. And all these techniques improve the efficiency of index construction. In addition,to meet the performance and diverse query requirements,we also adopt the primary-auxiliary index construction technology based on Hash and STR. To further improve the performance in the object query scenario,we invent another moving object spatial index construction method OAHSTRCL via time window object aggregation and bulk loading,where objects are divided more finely,and the object query time required is about 65% of HSTRCL,though it will affect the performance of spatial query to some extent. Theoretical analysis and experiments have demonstrated the effectiveness of our proposed methods.

Key words: spatio-temporal data stream;moving object;spatial index;R-tree;object aggregation

1 引言

随着物联网技术的快速发展,传感器、RFID、GPS等相关技术不断成熟,在现实生活中的应用也越来越广泛[1,2].采集具有时间和空间特性的数据变得便捷,它们形成时空数据流.同时,时空数据流通常会呈现出实时性、突发性、易失性、无限性等特征,其处理具有挑战性.为了提供准确快速的响应服务,需要对数据进行有效的组织和管理,高效索引构建是核心任务之一[3].

在时空间索引领域,R树[4]的应用尤为广泛.为提高R树的构建效率,衍生出基于R树的批量加载技术,其中以STR算法[5]的综合效果最佳.针对移动对象的索引方法大致可分为三大类.其中基于维度扩展的方法在传统的空间索引结构中加入时间信息,从而增强索引对时空数据的支持,3D-R树[6]是这种思想的一个天然实现.TB树[7]保证每一个叶节点中只包含属于同

收稿日期:2020-03-25;修回日期:2020-07-19;责任编辑:孙瑶
基金项目:国家自然科学基金(No.61702456);绍兴柯桥浙工大创新研究院项目(No.2019KQT002)

一条轨迹的轨迹段,非常适合移动对象历史轨迹数据查询.但也因此损失了大量空间信息,不适合传统范围查询.

基于重叠和多版本的方法为每个时间段内的空间数据维护一套索引结构,之后通过组合来表示整条轨迹.HR 树[8]就为每个时间戳建立一棵空间 R 树.MV3R 树[9]使用上下两棵树分别用于处理针对时间点和较长时间范围的查询.然而这一类方法的通病是每次创建新版本的成本都极高.

基于空间划分的方法先将数据对象散列划分到对应空间中,进而在每个子空间中为数据建立时间索引.SETI[10]将静态空间区域进行非重叠分区,使用六边形代替传统矩形作为索引目标的外接框.CSE[11]可以看成是 SETI 的扩展,在每个空间网格中针对不同的数据特征采用不同的索引结构.

在分布式时空间索引系统领域,李斌等[12]提出了面向 HBase 的二级索引,通过将索引列值聚集在对应索引表的行键上,来提高大数据场景下的查询效率,但索引更新开销大.Wang 等[13]提出了一种数据分区策略和基于模板的索引结构,可在分布式场景下实现快速的数据流提取和低延迟查询,但在网络带宽和负载均衡上仍需要优化.

王智广等[14]针对交通轨迹数据,提出了一种基于时空距离聚类的数据项构造方法,以缓解传统方法中因树过高及节点重复过多导致的查询性能下降问题.但该方法的建树效率较低,不适用于实时场景.赵馨逸等[15]提出了 GRIST 索引结构,其在空间上采用了基于 Geohash 编码的自适应分裂方法,在时间上为一维 R 树增加了节点合并策略以减少其规模,但目前只适用于历史数据,尚不支持对实时数据和未来数据进行索引.

综上,目前由 STR 算法构建的 R 树综合查询性能最佳,然因其构建时延高而无法直接应用到时空数据流场景.业界已有的大量时空间数据库和系统通常缺少针对时空数据流场景的额外设计,且普遍专注单一查询类型.对此,本文结合时空数据流的特性,提出了面向时间窗口和批量装载的高性能移动对象空间索引构建方法,既能够近实时地完成索引构建任务,又可以提供多样化的查询服务.

本文的主要贡献如下:提出了面向时空数据流的移动对象空间索引构建方法 HSTRCL 以及 OAHSTRCL;对所提两种方法的构建效率进行了理论分析;通过实验验证了所提方法.

2　数据流索引

本文采用时间窗口分片技术来处理时空间数据流.对于起始时间为 t_i 的第 i 个时间窗口 W_i,用二元组 $<t_i, W_i>$ 来表示,其中 $W_i = \{ <t_{ik}, id_{ik}, lon_{ik}, lat_{ik}, v_{ik}> | k = 1,2,\cdots, |W_i| \}$,其中 $<t_{ik}, id_{ik}, lon_{ik}, lat_{ik}, v_{ik}>$ 是一个流元组.t_{ik} 是该流元组对应的时间戳,id_{ik} 是产生该流元组的移动对象的唯一标识,lon_{ik}、lat_{ik} 分别是流元组的经、纬度坐标,v_{ik} 表示流元组的“值”,$|W_i|$ 表示该时间窗口内的流元组总数,在上下文不引起混淆的情况下也简单用 W_i 表示.

基于时间窗口的索引构建如图 1 所示.其中 W_1、W_2、…、W_k 表示接收时空间流元组的时间窗口,T_w 是时间窗口的时长;I_1、I_2、…、I_k 表示构建索引的时间区间,T_{delay} 表示构建索引所需时间,即“构建时延”.T_{wait} 表示在当前时间窗口对应索引构建完成后,直至下一个时间窗口的索引开始构建的时间间隔,即“构建间隔”.显然若 T_{wait} 小于零,则表示下一个时间窗口到达后,需要先等待当前时间窗口完成索引构建.在此状态下,等待时间将不断累加,因此整个构建方法需要保证 T_{wait} 大于等于零.

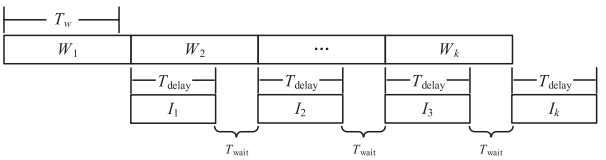

图1　接收时间窗口数据流元组构建索引的过程

所建移动对象空间索引的结构如图 2 所示.其中空间 R 树的每个节点 RNode 包含节点类型、节点层数、MBR 信息、指向父节点、子节点以及实际时空间流元组的指针数组;对象哈希索引的每个 *key* 值对应一个该时间窗口内移动对象的唯一标识,*value* 值为该移动对象在 R 树主索引中将会对应的叶节点信息集合.除此之外,最底层是某个时间窗口中的原始流元组,形式为 $<t, id, lon, lat, v>$,而中间层是一个经过排序处理的过渡数组.

2.1　HSTRCL

HSTRCL 是基于时间窗口数据排序和批量装载的移动对象空间索引构建方法,其具体构建过程如图 3 所示,共可分为 $P_1 \sim P_6$ 六个阶段.

(1) P_1-水平排序:先将时间窗口按照等时间长度进行分片,其中 $T_{M1\text{-slice}}$ 表示每个时间分片所对应的时长.在每个时间分片完成缓存后,立即以水平维度为排序依据,采用快速排序法对分片内的数据进行排序,$T_{M1\text{-sort}}$ 表示分片的排序时间,显然 $T_{M1\text{-slice}}$ 需要大于等于 $T_{M1\text{-sort}}$.在对当前时间分片内的数据进行排序的同时,可以继续接收下一个时间分片内的数据.当整个时间窗口结束时,将得到多个有序的时间分片,此时可以对它们进行归并排序,其中 $T_{M1\text{-merge}}$ 表示归并排序阶段耗时.

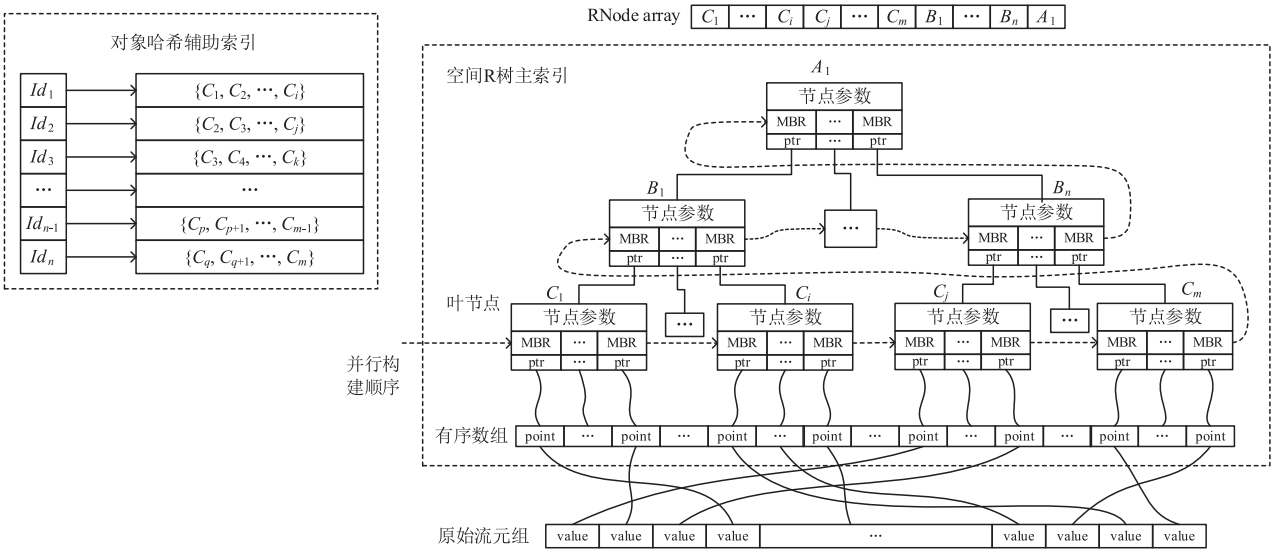

图2 移动对象空间索引结构示意图

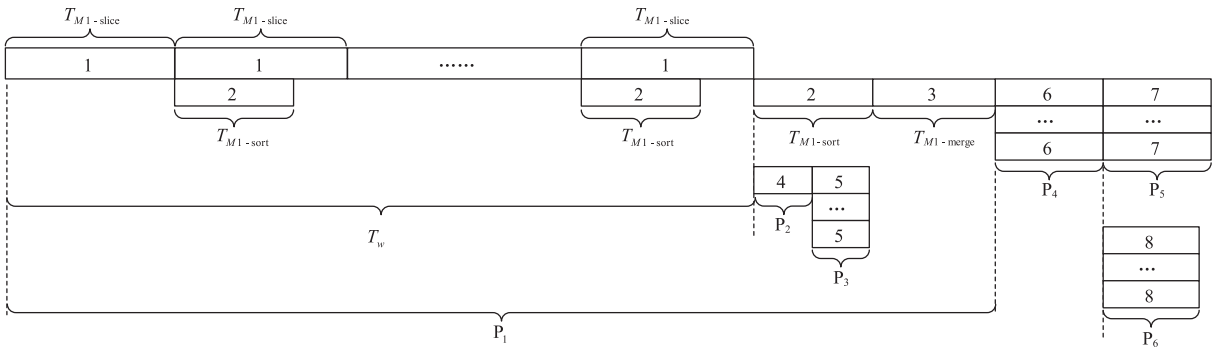

图3 HSTRCL构建过程

(2) P_2-结构参数计算:待时间窗口全部结束后开始计算结构参数,具体参数符号见表 1. 此过程可与最后一个数据分片的水平排序及所有分片的归并排序并行执行. 具体参数计算方式如下:

$$B^{H-1} < W \leqslant B^H \tag{1}$$

$$N_{\text{leaf}} = m^H = \left\lceil \frac{W}{B} \right\rceil \tag{2}$$

$$m^{k-1} = \left\lceil \frac{m^k}{B} \right\rceil \tag{3}$$

$$N_{n\text{-child}} = \begin{cases} B, & \text{if } m^k \% B = 0 \\ m^k \% B, & \text{if } m^k \% B \neq 0 \end{cases} \tag{4}$$

$$N_{l\text{-child}} = \begin{cases} B, & \text{if } W \% B = 0 \\ W \% B, & \text{if } W \% B \neq 0 \end{cases} \tag{5}$$

$$N_{n\text{-leaf}} = m^1 + m^2 + \cdots + m^{H-1} = \sum_{i=1}^{H-1} m^i \tag{6}$$

$$N_{\text{all}} = N_{n\text{-leaf}} + N_{\text{leaf}} = m^1 + m^2 + \cdots + m^H = \sum_{i=1}^{H} m^i \tag{7}$$

(3) P_3-骨架构建:由结构参数计算阶段可确定所需构建 R 树的轮廓结构、各节点在树中的位置与父子关系,因此 R 树骨架的构建可与 R 树的值填充进行分离. 骨架构建阶段由图 3 中标记为 5 的矩形表示,多个标记为 5 的矩形表示多任务并发,下同. 在该过程中叶节点尚未开始装填流元组,因此将所构建的 R 树结构称为骨架.

表 1 *R* 树结构参数表

参数	解释
B	每个节点的最大容量
W	时间窗口内的流元组总数
H	R 树的层数
$N_{n\text{-leaf}}$	R 树非叶节点个数
N_{leaf}	R 树叶节点个数
N_{all}	R 树总节点数
m^k	R 树第 k 层节点总数
$N_{n\text{-child}}$	单个非叶节点的子节点数
$N_{l\text{-child}}$	单个叶节点包含流元组数

(4) P_4-分组垂直排序:本阶段包含两个过程. 首先是流元组水平分组. 通过水平排序阶段得到长度为 L

的水平维度有序的流元组数组;在结构参数计算阶段确定 R 树节点的最大容量为 B,因此按照 STR 算法把所有流元组顺序划分到 $\sqrt{L/B}$个分组中. 其次是分组内垂直排序. 在每个分组内以数据的垂直维度为依据进行快速排序. 根据上述水平分组的过程可知,每个分组间相互独立,因此本过程可并发执行.

此外为了达到更好的划分效果,可令相邻两个分组依次按照升降序进行排序,尽可能让不同分组内下标邻近的流元组在空间上也邻近,以提高后续构建 R 树的查询性能.

(5)P_5-主索引批量装载:本阶段在逻辑上按自左至右、自底向上的顺序操作每个节点,也包含两个过程. 首先是叶节点流元组填充. 经过 $P_1 \sim P_4$阶段,不同的流元组分组已头尾相接顺序存储,因此仅需依次将每 B 个流元组装填到一个叶节点中. 其次是节点 MBR 更新. 待流元组填充完毕,先计算叶节点 MBR,之后顺序计算所有非叶节点 MBR.

(6)P_6-辅助索引批量装载:先在每个分组内遍历流元组并逐步构建哈希索引,*key* 值为流元组对应移动对象唯一标识,*value* 值为流元组对应叶节点编号集合. 之后再归并各个分组,即汇总不同分组内相同移动对象对应叶节点编号集合.

算法 1　HSTRCL

输入:流元组缓存数组 *SArray*;时间窗口时长 T_W;分片时长 $T_{M1\text{-slice}}$;最大子节点数 B

输出:R 树节点数组 *RNodeArray*;移动对象哈希表 *HashTable*

1. 初始化分片信息结构数组 *PosSlices*,*PosSlices*[i]表示第 i 个时间分片($i \geqslant 1$),包含 *start* 和 *end* 两个属性分别用于标记分片在 *SArray* 中的起始和结束位置;
2. $count = 1$,*PosSlices*[*count*]. *start* = *SArray*. *cur*;
3. 每隔 $T_{M1\text{-slice}}$时间执行:
4. 　*PosSlices*[*count*]. *end* = *SArray*. *cur*;
5. 　对分片 *PosSlices*[*count*]执行水平排序;
6. 　$count = count + 1$;
7. 　*PosSlices*[*count*]. *start* = *SArray*. *cur* + 1;
8. 每隔 T_W时间执行:
9. 　归并所有分片得有序流元组数组 *SortedArray*;
10. 　计算结构参数;
11. 　初始化 *RNodeArray* 并完成节点间链接关系;
12. 　对 *SortedArray* 执行水平分组并垂直排序;
13. 　读取 *SortedArray* 装填 *RNodeArray* 并计算各节点 MBR;
14. 　读取 *SortedArray* 构建各分组内哈希索引并汇总至 *HashTable*;
15. *Return RNodeArray*,*HashTable*;

2.2　OAHSTRCL

OAHSTRCL 是基于时间窗口对象聚合和批量装载的移动对象空间索引构建方法,目的是进一步强化对象查询性能. 在 HSTRCL 中只能得到包含所需对象的叶节点集合,之后仍需对其进行遍历. 而 OAHSTRCL 在 HSTRCL 的每个水平分组内,先根据移动对象唯一标识聚合数据,形成若干对象聚合单元 OAC. 然后基于全体 OAC 集合继续构建索引,整个过程如图 4 所示.

(1)水平排序:与 HSTRCL 对应阶段相同.

(2)分组垂直排序:与 HSTRCL 对应阶段的主要区别在于每个分组内的垂直排序过程. 先在每个分组内遍历流元组,根据其移动对象唯一标识进行聚合,形成若干 OAC. 当遍历结束可得到一个 OAC 集合,且每个 OAC 中包含的流元组都具有相同的移动对象唯一标识. 之后再以 OAC 为排序对象,OAC 中心点的纬度坐标为排序依据进行垂直排序.

(3)结构参数计算:与 HSTRCL 对应阶段相同,只需将原式中的 W 变更为分组垂直排序阶段得到的 OAC 总数 N_{OAC}.

(4)骨架构建:与 HSTRCL 对应阶段相同.

(5)主索引批量装载:与 HSTRCL 对应阶段相同,但读取的数据变更为全体 OAC 集合.

(6)辅助索引批量装载:先在每个分组内遍历 OAC,*key* 值为 OAC 的唯一标识,*value* 值为 OAC 对应叶节点信息集合,包括叶节点编号、该 OAC 包含的流元组个数以及该 OAC 在叶节点中起始位置的偏移量. 之后可与 HSTRCL 类似汇总所有分组.

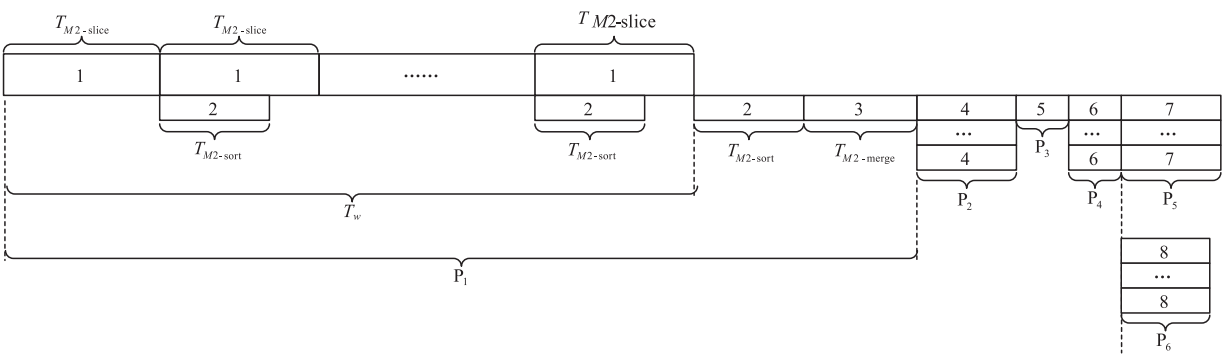

图4　OAHSTRCL构建过程

3 索引构建时间代价估算

本节对所提两种方法中涉及的多个过程进行理论分析,所涉及的符号汇总于表2,且与时间相关参数均以 s 为单位. 对于现代 CPU,可近似认为 $T_{RM}=T_{WM}=T_M$,且 $T_{\text{CPU}} \ll T_{RM}$. 因此将 T_{RM} 和 T_{WM} 统一为 T_M,同时忽略 T_{CPU}.

表2 符号参数表

参数名	参数含义
T_W	单个时间窗口的时间长度
V_S	时空数据流的实时到达流速(单位:条/s)
T_{CPU}	CPU 单次计算时间
$T_{RM}=T_M$	一次读内存的时间
$T_{WM}=T_M$	一次写内存的时间
B	R 树的节点容量
H_{M1}	HSTRCL 中 R 树的高度
H_{M2}	OAHSTRCL 中 R 树的高度
$T_{M1\text{-slice}}$	HSTRCL 中单个时间分片的时间长度
$N_{M1\text{-slice}}$	HSTRCL 中时间窗口内的时间分片数量
$t_{M1\text{-sort}}$	HSTRCL 中单个时间分片内的水平排序耗时
$t_{M1\text{-merge}}$	HSTRCL 中所有时间分片的归并排序耗时
$t_{M1\text{-para}}$	HSTRCL 中结构参数计算耗时
$t_{M1\text{-skltn}}$	HSTRCL 中搭建 R 树骨架的耗时
$t_{M1\text{-vsort}}$	HSTRCL 中分组垂直排序耗时
$t_{M1\text{-bulkLd-R}}$	HSTRCL 中批量装载 R 树索引耗时
$t_{M1\text{-bulkLd-O}}$	HSTRCL 中批量装载对象哈希索引耗时
$t_{M1\text{-delay}}$	HSTRCL 中移动对象空间索引的构建时延
$T_{M2\text{-slice}}$	OAHSTRCL 中单个时间分片的时间长度
$N_{M2\text{-slice}}$	OAHSTRCL 中时间窗口内的时间分片数量
$t_{M2\text{-sort}}$	OAHSTRCL 中单个时间分片内的水平排序耗时
$t_{M2\text{-merge}}$	OAHSTRCL 中所有时间分片的归并排序耗时
$t_{M2\text{-para}}$	OAHSTRCL 中结构参数计算耗时
$t_{M2\text{-skltn}}$	OAHSTRCL 中搭建 R 树骨架的耗时
$t_{M2\text{-vsort}}$	OAHSTRCL 中分组垂直排序耗时
$t_{M2\text{-bulkLd-R}}$	OAHSTRCL 中批量装载 R 树索引耗时
$t_{M2\text{-bulkLd-O}}$	OAHSTRCL 中批量装载对象哈希索引耗时
$t_{M2\text{-delay}}$	OAHSTRCL 中移动对象空间索引的构建时延
$N_{\text{OAC-gmax}}$	OAHSTRCL 中单个水平分组内的最大 OAC 个数
N_{OAC}	OAHSTRCL 中分组垂直排序得到的 OAC 总个数
k	系统可提供并行执行的核数,简称并行度

3.1 HSTRCL 构建时延分析

在 HSTRCL 中,需要满足 $T_{M1\text{-slice}} \geqslant t_{M1\text{-sort}}$,数据排序过程均采用复杂度为 $O(n\log n)$ 的快速排序. 假设 V_S 均匀,则一个分片内的元组数为分片时长与流速的乘积 $T_{M1\text{-slice}} \times V_S$,由此可得 $t_{M1\text{-sort}}$ 为

$$t_{M1\text{-sort}}=7\times T_M\times(T_{M1\text{-slice}}\times V_S)\times\log(T_{M1\text{-slice}}\times V_S) \tag{8}$$

其中 $7\times T_M$ 为快速排序法在单次循环内比较并交换数据所需时间[16]. 结合 $T_W=T_{M1\text{-slice}}\times N_{M1\text{-slice}}$ 可得

$$t_{M1\text{-sort}}=7\times T_M\times\frac{T_W\times V_S}{N_{M1\text{-slice}}}\times\log\left(\frac{T_W\times V_S}{N_{M1\text{-slice}}}\right) \tag{9}$$

由式(8)(9)、$T_{M1\text{-slice}} \geqslant t_{M1\text{-sort}}$ 及 $T_W=T_{M1\text{-slice}}\times N_{M1\text{-slice}}$ 可得

$$N_{M1\text{-slice}} \geqslant \frac{T_W\times V_S}{2^{(7\times T_M\times V_S)^{-1}}} \tag{10}$$

结论 1 由式(10)可知,$N_{M1\text{-slice}}$ 存在下限.

在 HSTRCL 中,首先需要完成分片的水平排序与归并,之后进行流元组的分组垂直排序. 同时,结构参数计算与构建 R 树骨架也需要按顺序执行. 上述两组操作可并行执行,最终取两组中耗时最长者. 批量装载 R 树索引和对象索引的过程与上述同理. 由此可得 $t_{M1\text{-delay}}$ 为

$$\begin{aligned}t_{M1\text{-delay}}=&\max\{t_{M1\text{-sort}}+t_{M1\text{-merge}}+t_{M1\text{-vsort}},t_{M1\text{-para}}+t_{M1\text{-skltn}}\}\\&+\max\{t_{M1\text{-bulkLd-R}},t_{M1\text{-bulkLd-O}}\}\end{aligned} \tag{11}$$

其中 $t_{M1\text{-merge}}$ 按最坏情况下的估算公式为

$$t_{M1\text{-merge}}=2\times T_M\times N_{M1\text{-slice}}\times T_W\times V_S \tag{12}$$

在水平分组阶段,计算可知每个分组中包含 $\sqrt{T_W\times V_S\times B}$ 个流元组. 假定全体水平分组根据并行度 k 均匀分配并处理,则 $t_{M1\text{-vsort}}$ 可近似为

$$t_{M1\text{-vsort}}=7\times T_M\times T_W\times V_S\times\log\left(\sqrt{T_W\times V_S\times B}\right)\times\frac{1}{k} \tag{13}$$

$t_{M1\text{-para}}$ 由于只涉及 CPU 操作可忽略不计. 创建 R 树骨架的过程是由最底层开始逐层向上,可得 $t_{M1\text{-skltn}}$ 为

$$\begin{aligned}t_{M1\text{-skltn}}=&\frac{6\times T_M\times T_W\times V_S}{B}+T_M\times T_W\times V_S\\&+\frac{6\times T_M\times T_W\times V_S}{B^2}+\frac{T_M\times T_W\times V_S}{B}\\&+\frac{6\times T_M\times T_W\times V_S}{B^3}+\frac{T_M\times T_W\times V_S}{B^2}+\cdots\\&+\frac{6\times T_M\times T_W\times V_S}{B^{H_{M1}}}+\frac{T_M\times T_W\times V_S}{B^{H_{M1}-1}}\\=&\frac{6\times T_M\times T_W\times V_S\times(B^{H_{M1}}-1)}{(B^{H_{M1}+1}-B^{H_{M1}})}\\&+\frac{T_M\times T_W\times V_S\times(B^{H_{M1}}-1)}{B^{H_{M1}}-B^{H_{M1}-1}}\end{aligned} \tag{14}$$

其中 $6\times T_M$ 为 R 树节点 6 个参数赋值的写内存时间.

$t_{M1\text{-bulkLd-R}}$ 的计算式为

$$\begin{aligned}t_{M1\text{-bulkLd-R}}=&\left(2\times T_M\times T_W\times V_S+\frac{2\times T_M\times T_W\times V_S}{B}+\cdots\right.\\&\left.+\frac{2\times T_M\times T_W\times V_S}{B^{H_{M1}-1}}\right)\times\frac{1}{k}\\=&\frac{2\times T_M\times T_W\times V_S\times(B^{H_{M1}}-1)}{(B^{H_{M1}}-B^{H_{M1}-1})}\times\frac{1}{k}\end{aligned} \tag{15}$$

类似地,$t_{M1\text{-bulkLd-O}}$ 的计算式为

$$t_{M1\text{-bulkLd-O}}=2\times T_M\times T_W\times V_S\times\frac{1}{k}+2\times T_M\times\sqrt{\frac{T_W\times V_S}{B}}\times N_{M1\text{-obj}}\tag{16}$$

由式(9)、式(12)~(14)可知,$t_{M1\text{-sort}}+t_{M1\text{-merge}}+t_{M1\text{-vsort}}>t_{M1\text{-para}}+t_{M1\text{-skltn}}$,由式(15)(16)结合 $B\gg 1$ 可知,$t_{M1\text{-bulkLd-R}}<t_{M1\text{-bulkLd-O}}$,因此 $t_{M1\text{-delay}}$ 的计算式为

$$\begin{aligned}t_{M1\text{-delay}}&=t_{M1\text{-sort}}+t_{M1\text{-merge}}+t_{M1\text{-vsort}}+t_{M1\text{-bulkLd-O}}\\&=7\times T_M\times\frac{T_W\times V_S}{N_{M1\text{-slice}}}\times\log\left(\frac{T_W\times V_S}{N_{M1\text{-slice}}}\right)+2\times T_M\times N_{M1\text{-slice}}\times T_W\times V_S+7\times T_M\times T_W\times V_S\times\log\left(\sqrt{T_W\times V_S\times B}\right)\times\frac{1}{k}\\&\quad+2\times T_M\times T_W\times V_S\times\frac{1}{k}+2\times T_M\times\sqrt{\frac{T_W\times V_S}{B}}\times N_{M1\text{-obj}}\end{aligned}\tag{17}$$

3.2 OAHSTRCL 构建时延分析

在 OAHSTRCL 中,同理有 $T_{M2\text{-slice}}\geqslant t_{M2\text{-sort}}$,且两者的水平排序阶段类似,借鉴式(8)~(10)可知

$$N_{M2\text{-slice}}\geqslant\frac{T_W\times V_S}{2^{(7\times T_M\times V_S)^{-1}}}\tag{18}$$

结论 2　由式(18)可知,$N_{M2\text{-slice}}$ 存在下限.

类似在 HSTRCL 中的分析,可得 $t_{M2\text{-delay}}$ 为

$$t_{M2\text{-delay}}=t_{M2\text{-sort}}+t_{M2\text{-merge}}+t_{M2\text{-vsort}}+t_{M2\text{-para}}+t_{M2\text{-skltn}}+\max\{t_{M2\text{-bulkLd-R}},t_{M2\text{-bulkLd-O}}\}\tag{19}$$

其中 $t_{M2\text{-sort}}$ 和 $t_{M2\text{-merge}}$ 的计算方式类似式(9)和式(12):

$$t_{M2\text{-sort}}=7\times T_M\times\frac{T_W\times V_S}{N_{M2\text{-slice}}}\times\log\left(\frac{T_W\times V_S}{N_{M2\text{-slice}}}\right)\tag{20}$$

$$t_{M2\text{-merge}}=2\times T_M\times N_{M2\text{-slice}}\times T_W\times V_S\tag{21}$$

对于 $t_{M2\text{-vsort}}$ 按最坏情况下的估算公式为

$$t_{M2\text{-vsort}}=\left[2\times T_M\times\sqrt{T_W\times V_S\times B}+7\times T_M\times N_{\text{OAC-gmax}}\times\log(N_{\text{OAC-gmax}})\right]\times\sqrt{\frac{T_W\times V_S}{B}}\times\frac{1}{k}\tag{22}$$

同样地 $t_{M2\text{-para}}$ 可忽略,$t_{M2\text{-skltn}}$、$t_{M2\text{-bulkLd-R}}$ 及 $t_{M2\text{-bulkLd-O}}$ 的计算方式与 HSTRCL 的对应参数相似,但数据总量变为 N_{OAC},计算式为

$$t_{M2\text{-skltn}}=\frac{6\times T_M\times N_{\text{OAC}}\times(B^{H_{M2}}-1)}{(B^{H_{M2}+1}-B^{H_{M2}})}+\frac{T_M\times N_{\text{OAC}}\times(B^{H_{M2}}-1)}{B^{H_{M2}}-B^{H_{M2}-1}}\tag{23}$$

$$t_{M2\text{-bulkLd-R}}=\frac{2\times T_M\times N_{\text{OAC}}\times(B^{H_{M2}}-1)}{(B^{H_{M2}}-B^{H_{M2}-1})}\times\frac{1}{k}\tag{24}$$

$$t_{M2\text{-bulkLd-O}}=2\times T_M\times N_{\text{OAC}}\times\frac{1}{k}+2\times T_M\times\sqrt{\frac{T_W\times V_S}{B}}\times N_{\text{OAC}}\tag{25}$$

为便于比较,设单个水平分组内的原始流元组个数与 $N_{\text{OAC-gmax}}$ 之比为 q_1,全体原始流元组个数与 N_{OAC} 之比为 q_2,即

$$\begin{cases}N_{\text{OAC-gmax}}=\dfrac{\sqrt{T_W\times V_S\times B}}{q_1}\\[2ex]N_{\text{OAC}}=\dfrac{T_W\times V_S}{q_2}\end{cases}\tag{26}$$

则 $t_{M2\text{-delay}}$ 的计算式为

$$\begin{aligned}t_{M2\text{-delay}}&=t_{M2\text{-sort}}+t_{M2\text{-merge}}+t_{M2\text{-vsort}}+t_{M2\text{-skltn}}+t_{M2\text{-bulkLd-O}}\\&=7\times T_M\times\frac{T_W\times V_S}{N_{M2\text{-slice}}}\times\log\left(\frac{T_W\times V_S}{N_{M2\text{-slice}}}\right)+2\times T_M\times N_{M2\text{-slice}}\times T_W\times V_S+\left[2\times T_M\times T_W\times V_S+\frac{7\times T_M\times T_W\times V_S}{q_1}\times\log\left(\frac{\sqrt{T_W\times V_S\times B}}{q_1}\right)\right]\times\frac{1}{k}\\&\quad+\left[\frac{6\times T_M\times T_W\times V_S\times(B^{H_{M2}}-1)}{(B^{H_{M2}+1}-B^{H_{M2}})}+\frac{T_M\times T_W\times V_S\times(B^{H_{M2}}-1)}{B^{H_{M2}}-B^{H_{M2}-1}}\right]\times\frac{1}{q_2}\\&\quad+2\times T_M\times T_W\times V_S\times\frac{1}{k\times q_2}+2\times T_M\times\sqrt{\frac{T_W\times V_S}{B}}\times\frac{T_W\times V_S}{q_2}\end{aligned}\tag{27}$$

假设时间窗口内的数据流元组在空间上的分布大致均匀,且通常情况下绝大多数的移动对象都会对应多条流元组数据,则有 $q_1\approx q_2=q\gg 1$. 当 $N_{M1\text{-slice}}=N_{M2\text{-slice}}=N$ 时,结合 $B>k$,$t_{M1\text{-delay}}$ 与 $t_{M2\text{-delay}}$ 之差可近似计算为

$$t_{M1\text{-delay}}-t_{M2\text{-delay}}\approx 7\times T_M\times T_W\times V_S\times\left[\frac{(q-1)\times\log\sqrt{T_W\times V_S\times B}}{k\times q}-1\right]>0\tag{28}$$

结论 3　由式(28)可知,当 $N_{M1\text{-slice}}=N_{M2\text{-slice}}$ 时,可得 $t_{M1\text{-delay}}>t_{M2\text{-delay}}$. 即在索引构建时延上,OAHSTRCL 优于 HSTRCL.

4　实验评价

4.1　实验环境

实验平台采用 Intel(R) Core(TM) i7-4790 四核处理器,3.30GHz 主频,DDR3 1600 MHz 16GB 内存,1TB 7200RPM 硬盘,Ubuntu 16.04 系统.

实验采用 C ++ 实现两种方法,采用 BerlinMOD[17] 生成仿真的时空间数据集,并逐步将数据缓存至内存,再根据一定的流速从内存中读取来模拟时空数据流.

4.2　线程数目对构建性能的影响

设实验数据量为 1 亿,时间窗口大小为 60s,时间

窗口分片数固定为10. 观察指标为$t_{M1\text{-delay}}$和$t_{M2\text{-delay}}$、时间窗口最后一个数据分片的水平排序与归并所有分片的总时间($t_{M1\text{-sort}}+t_{M1\text{-merge}}$)和($t_{M2\text{-sort}}+t_{M2\text{-merge}}$)、分组垂直排序时间$t_{M1\text{-vsort}}$和$t_{M2\text{-vsort}}$,以及索引批量装载时间$t_{M1\text{-bulkLd}}$和$t_{M2\text{-bulkLd}}$. 结果见图5和图6.

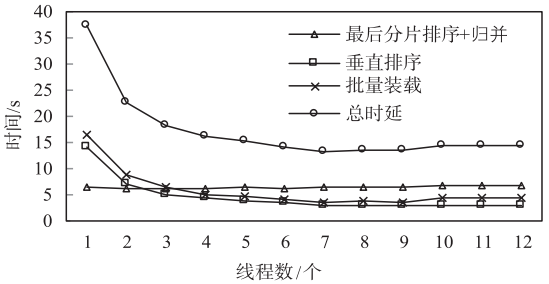

图5 HSTRCL中线程数与构建时延的关系

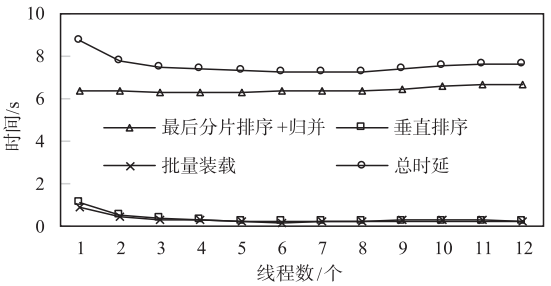

图6 OAHSTRCL中线程数与构建时延的关系

由图可知$t_{M1\text{-vsort}}$、$t_{M2\text{-vsort}}$、$t_{M1\text{-bulkLd}}$和$t_{M2\text{-bulkLd}}$会快速减少,后因CPU等硬件瓶颈将趋于平稳. 两种方法的性能均在7线程时接近最优;($t_{M1\text{-sort}}+t_{M1\text{-merge}}$)和($t_{M2\text{-sort}}+t_{M2\text{-merge}}$)所对应阶段采用单线程处理,因此始终保持平稳.

OAHSTRCL因采用对象聚合处理,其在分组垂直排序和索引批量装载时所需处理的数据都会大大减少,即对应阶段所需时间也会减少.

保持其他参数不变,分析不同数据量下构建时延随线程数的变化情况,结果见图7和图8. 由图可知,两种方法在不同数据量下的构建时延都会先快速减少后趋于平稳.

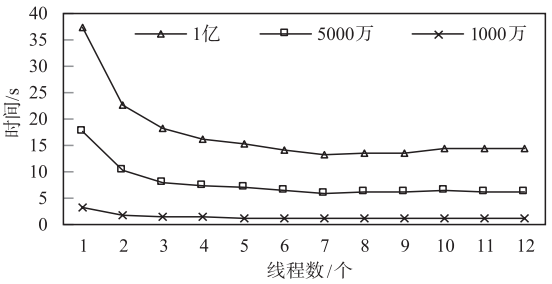

图7 HSTRCL中不同数据量下线程数与构建时延的关系

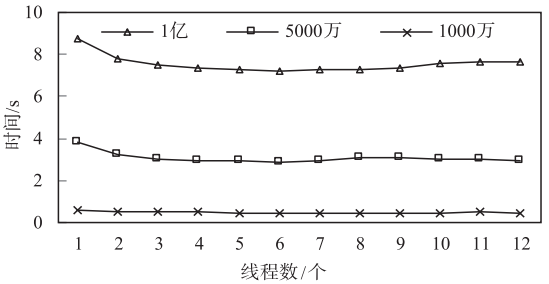

图8 OAHSTRCL中不同数据量下线程数与构建时延的关系

4.3 窗口分片数目对构建性能的影响

设实验数据量为1亿,时间窗口大小为60s,根据上一小节所得结果固定线程数为7,具体指标分别与图5和图6相同,结果见图9和图10. 由图可知,($t_{M1\text{-sort}}+t_{M1\text{-merge}}$)和($t_{M2\text{-sort}}+t_{M2\text{-merge}}$)开始会快速减少,之后反而有所增大. 因为分片数的增加固然会使分片内的排序时间减少,但分片归并的时间会增加更多,从而提高整体时间;$t_{M1\text{-vsort}}$、$t_{M2\text{-vsort}}$、$t_{M1\text{-bulkLd}}$、$t_{M2\text{-bulkLd}}$所对应阶段均与窗口分片数量均无关,因此基本保持不变.

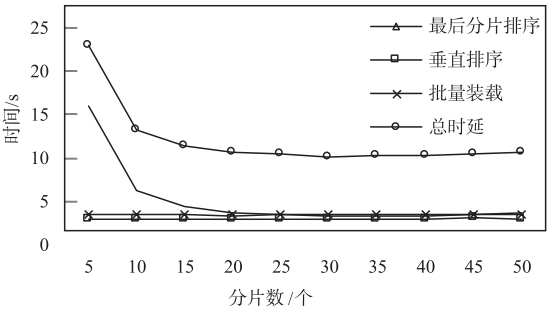

图9 HSTRCL中分片数与构建时延的关系

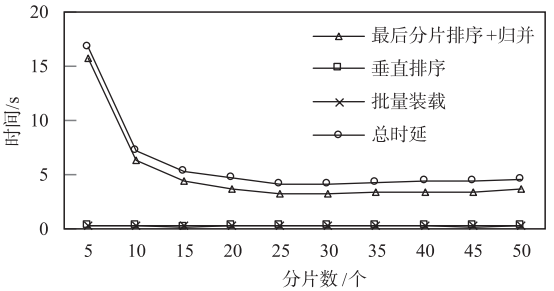

图10 OAHSTRCL中分片数与构建时延的关系

采用同样方式分析不同数据量下构建时延随分片数的变化情况,结果见图11和图12. 由图可知,两种方法在不同数据量下的构建时延都会先快速减少后略微增加.

4.4 方法所能承受的数据流最大流速

假设数据流是匀速到达的,则可以通过固定时间窗口大小,而改变窗口内流元组数量的方式,来模拟不同流速的数据流. 设线程数为1,分片数为10,时间窗口大小为10s,即每个分片时长为1s,结果如图13所示.

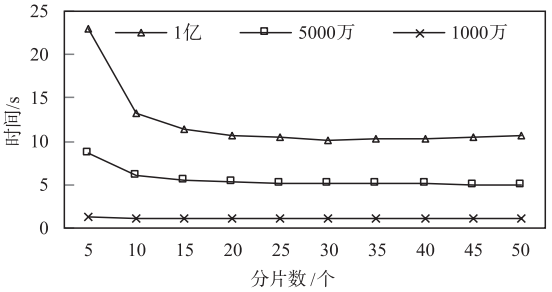

图11　HSTRCL中不同数据量下分片数与构建时延的关系

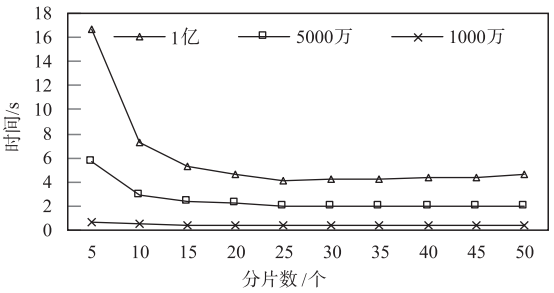

图12　OAHSTRCL中不同数据量下分片数与构建时延的关系

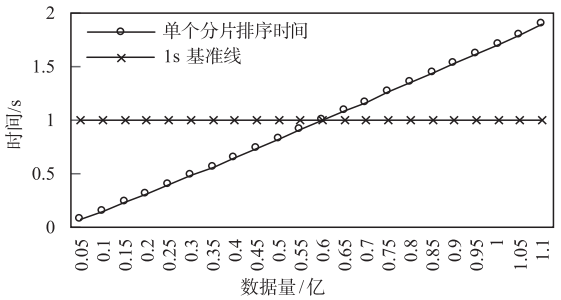

图13　单线程10分片下单个分片排序时间随数据量变化趋势

由图可知,排序时间折线在约0.6亿数据量时与分片时长基准线相交,即在本实验条件下,方法能承载的最大流速约为600万条/s.

4.5　方法的查询性能

设实验数据量为1亿,分别进行空间查询和对象查询,且在空间查询中每种窗口条件均实验100次. 两者均记录累计时间和,结果见图14和图15.

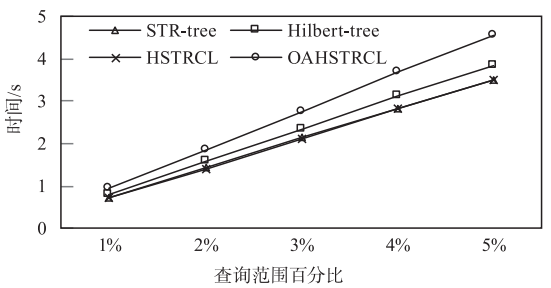

图14　不同方法的空间查询时间对比

由图可知,HSTRCL方法所得索引与高效空间索引STR树性能相近,但略优于Hilbert R树. OAHSTRCL方法所得索引与高效移动对象索引TB树性能相近,但优于3D-R树.

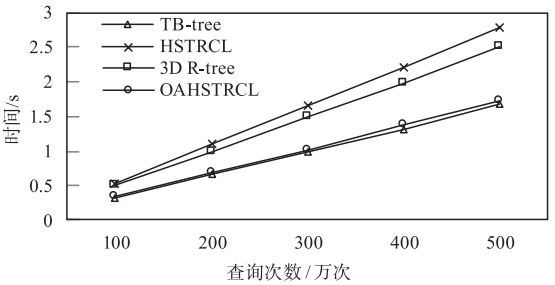

图15　不同方法的对象查询时间对比

虽然STR树和TB树在各自擅长查询领域内效果很好,但若两者交换查询领域,由于缺少对应索引,效率会明显降低,近似于蛮力搜索. 采用蛮力法在1亿数据量下执行查询操作,结果见图16.

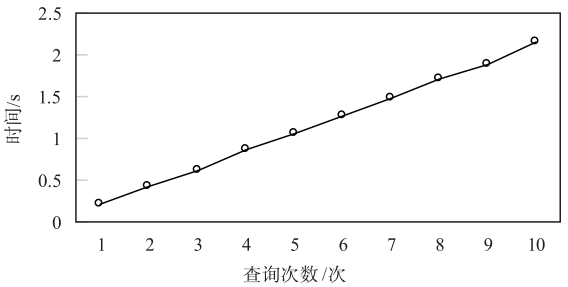

图16　蛮力法查询时间随查询次数变化趋势

由图可知,使用蛮力法在1s内仅足够进行约5次查询,效率远远无法和本文所提两种方法相比.

5　结语

本文面向时空数据流并针对其特点,利用并行处理的思想,将耗时的数据排序操作与其他操作进行分离,同时结合多种优化构建手段最大化并行装载的性能,有效提高构建效率. 基于上述思想,本文先后提出了HSTRCL与OAHSTRCL,两者各有适用场景. 今后将进一步研究适用于分布式场景时空流数据索引方案.

参考文献

[1] 章登义,李想. 一种基于密度网格索引的k-最近邻查询算法[J]. 电子学报,2017,45(2):376－383.
Zhang D Y,Li X. A k-nearest neighbor query algorithm for density grid-based index [J]. Acta Electronica Sinica,2017,45(2):376－383. (in Chinese)

[2] 李灿,钱江波,等. M2LSH:基于LSH的高维数据近似最近邻查找算法[J]. 电子学报,2017,45(6):1431－1442.
Li C,Qian J B,et al. M2LSH:An LSH based technique for approximate nearest neighbor searching on high dimensional data[J]. Acta Electronica Sinica,2017,45(6):1431－

1442. (in Chinese)
[3] 梁俊杰,李凤华,等. MapReduce 框架下的优化高维索引与 KNN 查询[J]. 电子学报,2016,44(8):1873 - 1880.
Liang J J,Li F H,et al. Optimized high-dimensional index and KNN query in mapreduce[J]. Acta Electronica Sinica,2016,44(8):1873 - 1880. (in Chinese)
[4] Guttman A. R-trees:A dynamic index structure for spatial searching[A]. ACM SIGMOD International Conference on Management of Data[C]. New York,USA:ACM,1984. 47 - 57.
[5] Leutenegger S T,Lopez M A,Edgington J. STR:A simple and efficient algorithm for R-tree packing[A]. Proceedings of the 13th International Conference on Data Engineering[C]. Piscataway,USA:IEEE,1997. 497 - 506.
[6] Theodoridis Y,Vazirgiannis M,Sellis T. Spatio-temporal indexing for large multimedia applications[A]. Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems[C]. Piscataway,USA:IEEE,1996. 441 - 448.
[7] Pfoser D,Jensen C S,Theodoridis Y. Novel approaches to the indexing of moving object trajectories[A]. VLDB[C]. San Francisco,USA:Morgan Kaufmann,2000. 395 - 406.
[8] Nascimento M A,Silva J R O. Towards historical R-trees[A]. Proceedings of the 1998 ACM Symposium on Applied Computing[C]. New York,USA:ACM,1998. 235 - 240.
[9] Tao Y,Papadias D. MV3R-Tree:A spatio-temporal access method for timestamp and interval queries[A]. VLDB[C]. San Francisco,USA:Morgan Kaufmann,2001. 431 - 440.
[10] Chakka V P,Everspaugh A C,Patel J M. Indexing large trajectory data sets with SETI[J]. Ann Arbor,2003,1001(48109 - 2122):12.
[11] Wang L,Zheng Y,Xie X,et al. A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval[A]. Proceedings of the 9th International Conference on Mobile Data Management (MDM'08)[C]. Piscataway,USA:IEEE,2008. 1 - 8.
[12] 李斌,郭景维,彭骞. 面向大数据存储的 HBase 二级索引设计[J]. 计算技术与自动化,2019,(2):23.
Li B,Guo J W,Peng Q. Design of Hbase secondary indexes for big data storage[J]. Computing Technology and Automation,2019,(2):23. (in Chinese)
[13] Wang L,Cai R,Fu T Z J,et al. Waterwheel:Realtime indexing and temporal range query processing over massive data streams[A]. Proceedings of IEEE 34th International Conference on Data Engineering (ICDE)[C]. Piscataway,USA:IEEE,2018. 269 - 280.
[14] 王智广,申思,鲁强. 一种用于交通轨迹数据的时空 R 树索引结构[J]. 内蒙古大学学报(自然科学版),2019,50(03):317 - 323.
Wang Z G,Shen S,Lu Q. An index structure about spatio-temporal R-tree for traffic trajectory data[J]. Journal of Inner Mongolia University (Natural Science Edition),2019,50(03):317 - 323. (in Chinese)
[15] 赵馨逸,黄向东,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 X Y,Huang X D,et al. A spatio-temporal index based on skew spatial coding and R-tree[J]. Journal of Computer Research and Development,2019,56(3):666 - 676. (in Chinese)
[16] Hoare C A R. Algorithm 64:Quicksort[J]. Communication of the ACM,1961,4(7):321.
[17] Christian D,Behr T ,Ralf H G. BerlinMOD:A benchmark for moving object databases[J]. VLDB Journal,2009,18(6):1335 - 1368.

作者简介

杨良怀 男,1967 年出生于浙江新昌. 毕业于北京大学,获博士学位. 浙江工业大学计算机学院教授,主要研究方向为数据库系统、数据挖掘. 在 VLDB J,J of Info and Soft Tech,J of Sys and Soft,《计算机研究与发展》、《软件学报》等期刊以及 VLDB,SIGKDD,CIKM,DASFAA 等数据库领域重要会议上发表学术论文 80 余篇.
E-mail:yanglh@zjut.edu.cn

沈东海 男,1995 年出生于浙江宁波. 浙江工业大学硕士研究生,主要研究方向为时空数据流处理.
E-mail:215936795@qq.com

范玉雷(通讯作者) 男,1984 年出生于内蒙古赤峰. 毕业于中国人民大学,获博士学位,浙江工业大学计算机学院讲师,主要研究方向为数据库管理系统、数据流管理系统和数据挖掘. 在 IEEE Data Eng Bull 等英文期刊和计算机学报等中文期刊以及 DASFAA 和 WAIM 等数据库领域国际会议发表学术论文 20 余篇.
E-mail:fyl815@zjut.edu.cn