引文格式:GONG Jun, KE Shengnan, ZHU Qing, et al. An Efficient Trajectory Data Index Integrating R-tree, Hash and B*-tree[J]. Acta Geodaetica et Cartographica Sinica, 2015, 44(5): 570-577. (龚俊, 柯胜男, 朱庆, 等. 一种集成 R 树、哈希表和 B* 树的高效轨迹数据索引方法[J]. 测绘学报, 2015, 44(5): 570-577.) DOI: 10.11947/j.AGCS.2015.20130520

一种集成 R 树、哈希表和 B* 树的高效轨迹数据索引方法

龚 俊[1], 柯胜男[1], 朱 庆[2], 张叶廷[3]

1. 江西师范大学软件学院, 江西 南昌 330022; 2. 西南交通大学地球科学与环境工程学院, 四川 成都 610031; 3. 武汉大学测绘遥感信息工程国家重点实验室, 湖北 武汉 430079

An Efficient Trajectory Data Index Integrating R-tree, Hash and B*-tree

GONG Jun[1], KE Shengnan[1], ZHU Qing[2], ZHANG Yeting[3]

1. School of Software, Jiangxi Normal University, Nanchang 330022, China; 2. Faculty of Geosciences and Environmental Engineering, Southwest Jiaotong University, Chengdu 610031, China; 3. State Key Laboratory of Information Engineering in Surveying Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China

Abstract: To take into account all of efficiency and query capability, this paper presents a new trajectory data index named HBSTR-tree. In HBSTR-tree, trajectory sample points are collectively stored into trajectory nodes sequentially. Hash table is adopted to index the most recent trajectory nodes of mobile targets, and trajectory nodes will not be inserted into spatio-temporal R-tree until full, which can enhance generation performance in this way. Meantime, one-dimensional index of trajectory nodes in the form of B*-tree is built. Therefore, HBSTR-tree can satisfy both spatio-temporal query and target trajectory query. In order to improve search efficiency, a new criterion for spatio-temporal R-tree and one new node-selection sub-algorithm are put forward, which further optimize insertion algorithm of spatio-temporal R-tree. Furthermore, a database storage scheme for spatio-temporal R-tree is also brought up. Experimental results prove that HBSTR-tree outperforms current methods in several aspects such as generation efficiency, query performance and supported query types, and then supports real-time updates and efficient accesses of huge trajectory database.

Key words: trajectory; spatio-temporal index; R-tree; B*-tree; storage

Foundation support: The National Natural Science Foundation of China(No.41261086); The National High-tech Research and Development Program of China(863 Program)(No.2012AA121401)

摘 要:为兼顾时空索引方法的空间利用率、时间效率和查询种类,提出了一种新的轨迹数据索引方法——HBSTR 树。其基本思想是:轨迹采样点以轨迹节点的形式成组集中管理,哈希表用于维护移动目标的最新轨迹节点,轨迹节点满后作为叶节点插入时空 R 树,另外采用 B* 树对轨迹节点构建一维索引,既有利于提升索引创建效率,又同时满足时空条件搜索和特定目标轨迹搜索等多种查询类型。为提升时空查询效率,提出了新的时空 R 树评价指标和节点选择子算法改进时空 R 树插入算法,同时提出了一种时空 R 树的数据库存储方案。试验结果表明,HBSTR 树在创建效率、查询效率和支持查询类型等方面综合性能优于现有方法,支持大规模实时轨迹数据库的动态更新和高效访问。

关键词:轨迹;时空索引;R 树;B* 树;存储

中图分类号:P208 **文献标识码**:A **文章编号**:1001-1595(2015)05-0570-08

基金项目:国家自然科学基金(41261086); 国家 863 计划(2012AA121401)

1 引 言

全球各国重视和推动广域室内外高精度定位导航基础设施建设,定位精度和可靠性将在城市环境内得以显著提升,自由的市场环境将使移动定位服务应用发生井喷式发展。同时,Web 2.0

时代，社会大众既是信息使用者也是提供者，随着网络隐私安全问题的技术及制度破解，专业数据库服务器将不间断收集具有潜在价值的巨量轨迹数据。迅猛发展的移动计算、无线传输技术和广泛部署的室内外定位设备对时空数据管理能力也提出全新需求，多源异构、时空异变、巨量增长、语义复杂成为时空数据库的标志性特征，对现有数据库索引方法提出严峻挑战[1-2]。移动对象位置频繁更新且规模空前，致使索引结构剧烈变化[3-4]。轨迹数据索引还要支持复杂的社会应用，如行为模式分析、智能交通决策等[5-7]。因此，轨迹数据索引方法除了关注大数据对查询效率的影响，还要解决实时更新的代价问题，并要支持丰富的轨迹语义。

现有支持轨迹数据的时空索引方法主要分为 3 种类型：①基于版本的索引方法，即每个时间版本维护一套空间索引结构，目的是为节省存储共享版本间未变节点，代表方法包括 HR-Tree 和 MV3R-tree 等，这类方法采用先时间后空间的策略，优点是时刻查询效率高，然而轨迹更新频繁，要为每个更新时刻创建版本，使得维护成本极高，即使采用节点共享策略，仍将耗费大量存储空间，且时空区间查询效率低下；②空间划分方法，即采用格网或四叉树将采样点散列划分到对应空间内，进而为空间中的对象采样点构建时间索引，代表方法包括 SETI 和 CSE 等，其关键点是对地理空间进行一维编码，即将二维问题转换为一维问题，再将一维地理编码＋时间构建为一维复合索引（如 B* 树），这类方法采用先空间后时间的策略，优点是采用地理哈希思想，空间区域划分固定，构建索引和查询效率较高，缺点是仅适合点的场合，非点对象落于空间划分边界上将导致重复索引，同时要求预设空间范围，索引结构不均衡易致效率下降；③扩展时间维的多维索引方法，即在传统空间索引结构（如 R 树）中增加时间维，代表方法包括 STR 树和 TB 树等，这类方法采用空间和时间等同地位策略，然而 TB 树及变种更重视时间序列特性和显式轨迹描述，这类方法属于 R 树变种，可随数据分布动态调整索引结构，时空查询效率较好，缺点是索引创建效率易恶化[8-12]。

随着应用的综合性要求日益突出，全时段的移动对象数据模型及其索引方法被深入研究，混合索引方法成为一种发展趋势[13-15]。本文提出一种轨迹索引方法，它是一种联合哈希表和 B 树的时空 R 树方法（Hash & B*-tree combined spatio-temporal R-tree，HBSTR-tree）。

2　算法思路和索引结构

索引设计既要了解数据特点，还要明确查询需求，索引方法通常受到创建效率、存储利用率、查询效率、查询种类、缓存机制等需求的影响[16]。本文旨在为实时增量轨迹数据库提供索引支持，采用集成 R 树、哈希表和 B* 树的混合索引方法 HBSTR-tree，图 1 是 HBSTR-tree 的原理和框架示意图。3 种子索引结构的作用分别为：

（1）R 树是主体索引结构，用于实现时空范围查询；

（2）哈希表是辅助结构，维护移动对象的最新轨迹节点，用于成组插入采样点；

（3）B* 树是次要索引结构，轨迹节点的一维索引，用于实现目标对象的轨迹查询。

采样点包含对象标识符、时间戳和空间位置等数据项，由于传感器遮挡或者不稳定等因素，采样点空间位置数据可能出现粗差，一般采用某些数据清洗方法过滤粗差数据[17]。单个移动对象的采样点数目数以万计，分散管理采样点不利于查找轨迹，对存储利用率和创建效率均会产生负面影响。本文将连续采样点成组集中存储于轨迹节点，轨迹节点仅存储单个对象的连续采样点。现实应用中相邻采样点有可能间隔很长时间，如果相邻采样点的时间间隔超过设定阈值 T_{thres}，则重新采用新节点存储后续采样点。

哈希表用于管理所有对象的最新轨迹节点，便于通过对象标识符查找对象的最新轨迹节点，当一个轨迹节点满时，插入 R 树，创建新的轨迹节点接收新采样点。这样，以节点形式成组插入时空 R 树的方式，可大幅度提升索引创建效率。

HBSTR-tree 中，R 树是 $N+1$ 维（N 是空间维数，1 指时间维）的时空 R 树，R 树节点最小包围盒 MBR（minimal bounding rectangle）是其孩子集合的时空坐标轴最小范围，时间参考采用 1970 年以来的绝对秒数作为基准。上文轨迹节点作为 R 树的叶节点，采用一种新的节点插入算法（第 3 节介绍）将其索引项插入叶节点层的上一层中，利用节点选择和节点分裂子算法优化时空 R 树结构。时空 R 树支持多种查询类型，如搜索某时空范围内的对象集合、对象轨迹，或者某时刻某空间范围内的对象集合、对象位置，或者某时刻某空间点的最近邻对象等。时空 R 树搜索目标

对象在某时间段内的轨迹并不高效。为解决该问题,本文方法采用轨迹节点的对象标识符 OID 和起始时间 tTimeStart 组成一维关键码(OID+tTimeStart)构建轨迹节点的 B* 树索引,借助B* 树的一维查询能力,高效定位某对象在某时刻的轨迹节点,利用 B* 树兄弟节点间的双向指针进行轨迹追溯。轨迹节点通常包含近百个连续采样点,相对于直接采样点的一维索引结构,本文方法节省存储空间 90%以上。

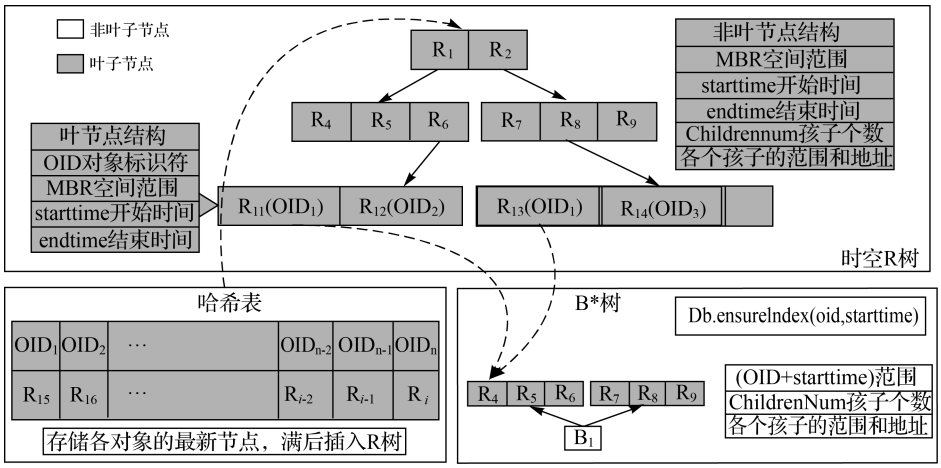

图 1 HBSTR 树的原理和框架

Fig.1 Principle and framework of HBSTR-tree

3 算法流程和步骤

本文针对轨迹查询需求改进时空 R 树算法,本节将予以详细介绍。由于哈希表和 B* 树是经典算法,不再赘述。

3.1 时空 R 树评价指标

鉴于多方面综合考虑,既要鼓励 MBR 在空间上形成规则块状,也要保证时间区分度,参考文献[18],本文提出时空评价值的定义作为时空 R 树评价指标。

时空评价值(N 维空间+1 维时间)EVAL:节点在 N 个空间坐标轴的区间(S_1,S_2,…,S_N)平均值的 N 次方与时间坐标轴的区间(T)的乘积

$$\mathrm{EVAL}\left(\frac{\sum_{i=1}^{N} s_i}{N}\right)^N \times T \tag{1}$$

选择该评价指标的原因来源于平均值不等式(以三维为例)

对任意 X、Y、$Z>0$,总有 $\left(\frac{X+Y+Z}{3}\right)^3 \geqslant XYZ$,当且仅当 $X=Y=Z$ 时,等式成立。

假设 XYZ 的值一定时,当 $X=Y=Z$ 时,X、Y、Z 的平均值 3 次方最小,即三维矩形体积一定时,当为立方体时,$\left(\frac{X+Y+Z}{3}\right)^3$ 最小。因此,将 $\left(\frac{\sum_{i=1}^{N} s_i}{N}\right)^N$ 引入 MBR 评价标准 EVAL,有助于控制节点的空间范围,即其值越小,MBR 的空间形态越趋近规则块状体,此外,T 越小,EVAL 也相应越小,引入 T 有助于控制节点的时间范围,控制 EVAL 值有助于优化时空 R 树节点的区分度。

3.2 HBSTR 树生成算法

为满足外存索引要求,R 树中的每个节点对应一个外存数据页,根据操作系统和数据库的基本参数设置数据页尺寸(如 2K 或 3K),而数据页尺寸也决定了节点中孩子数目,由于叶节点和非叶节点结构不同,叶节点中的轨迹元组数目和非叶节点中孩子节点数目不相等。同时,依据 R 树的特点,为保证节点的层号终生不变,将叶节点层号设为 0,其他层依次向上加 1,如叶节点的父节点层号为 1,因为 R 树层数增加时,发生在根节点分裂时,此时生成新的根节点,其层号为原根节点层号加 1。

下面给出 HBSTR 树生成算法的步骤描述。节点分裂子算法采用传统方法,节点选择子算法与传统方法有所不同,本文重点给出它的详细步骤,节点选择和节点分裂中均采用上文所述的时空范围评价指标。

首先简单介绍算法描述中涉及的元素。待插入采样点 Tuple{OID, Time, Pos, ROWID,

etc}，其中，OID 是采样点所属的对象标识符；Time 是采样点获取时刻；Pos 是采样点地理位置；ROWID 是采样点记录的外存指针。HBSTR 索引结构，包括哈希表 Hash（OID 映射至轨迹节点 TNode）、B* 树 Btree（索引键值是对象标识符 OID＋轨迹节点开始时间 tStartTime）和时空轨迹 R 树 Rtree。算法中 TNode 是对应于 OID 的轨迹节点，用于存储某移动对象的连续轨迹点，达到规定上限后插入时空轨迹 R 树成为叶节点，其结构为{OID，tStartTime，tEndTime，MBR，etc}，其中，OID 是对象标识符；tStartTime 是开始时间；tEndTime 是终止时间；MBR 是空间范围。

算法 1 描述：HBSTR 树的动态插入算法。

算法输入：待插入采样点元组 Tuple，HBSTR 已有索引结构，连续轨迹结束时间阈值 T_{thres}，节点缓存清除时间阈值 T_{cnode}。

算法输出：更新后的索引结构如下：

（1）通过 Hash 查找 OID 对应的轨迹节点 TNode。如果 Time 相对 TNode 结束时间 tEndTime 超出时间阈值 T_{thres}，进入步骤（2）；否则，将 Tuple 插入 TNode，并更新 TNode，如果 TNode 已满，进入步骤（2），否则退出算法。

（2）利用节点选择子算法 1.1 为 TNode 选择适合插入的第 1 层节点 Father，插入 TNode，如果导致 Father 上溢，则采用节点分裂子算法将 Father 一分为二，如果导致 Father 的父节点上溢，采用节点分裂算法递归处理，直至根节点。

（3）将 TNode 的 OID 和 tStartTime 的组合键加入 Btree 增加索引项。

（4）生成新的轨迹节点，将 Tuple 插入其中，并替代 Hash 中 OID 对应的 TNode。

（5）获取 Rtree 的主存 Cache 中的节点数目 NodesNum，如果 NodesNum 超出阈值，则从根节点开始从缓存中清除 T_{cnode} 未访问的节点。

（6）算法终止。

本文时空 R 树节点选择子算法与传统方法有所区别，其思想是，在 R 树中搜索完全包含待插节点时空范围的最下层节点集合，从集合中选择一个节点，此时该节点的子孙节点均不可能包含待插节点，则以该节点为根节点从上而下寻找最适合插入待插节点的第一层节点。本算法的优点在于可以避免节点重叠对选择结果的不良影响，采用非递归思想设计算法，保证良好的时间效率。节点选择子算法是本算法的核心，采用伪码方式描述算法流程，见下文。

算法：节点选择子算法（选出适合插入 TNode 的第 1 层节点，注意，叶节点处于第 0 层）

算法输入：待插入的轨迹节点 TNode，时空 R 树根节点 Root

算法输出：适合插入 TNode 的第 1 层节点

1. NodeSet.Clear()：清空节点集合

2. MinLevelID←Root.LevelID＋1：变量 MinLevelID 用于记录已搜索的符合条件最底层节点的层号，赋初值为树深即大于任意节点的层号

3. Father←Root：变量 Father 赋初值为根节点

4. SeqArray[Father.LevelID]←0;：数组 SeqArray 第 Father.LevelID 个值赋值为 0，该数组用于记录当前节点 Node 在父节点和祖先节点中的序号

5. While Father ！= NULL Do

6. Node ← Father.IthChild（SeqArray[Father.LevelID]）：Father 第 i 个孩子作为当前节点

7. IfNode.Contain(TNode)＝True Then：Node 包含 TNode 时

8. IfNode.LevelID＝MinLevelIDThen：Node 层号等于 MinLevelID 时

9. NodeSet.Add(Node)：将 Node 加入 NodeSet

10. Else IfNode.LevelID〈MinLevelIDThen：Node 层号小于 MinLevelID 时

11. NodeSet.Clear()：清空 NodeSet

12. NodeSet.Add(Node)：将 Node 加入 NodeSet

13. MinLevelID←Node.LevelID：MinLevelID 赋值为 Node 层号

14. End If

15. IfNode.LevelID〉1 Then：Node 层号大于 1 时，将进入下层节点

16. Father←Node：Father 赋值为 Node

17. SeqArray[Father.LevelID]←0：给序号数组中的对应元素赋值为 0

18. Continue Loop：重新开始新一轮循环

19. End If

20. End If

21. IfNode.LevelID＝1 Or Node.Contain(TNode)＝False Then：假如 Node 已经处于第 1 层，即叶节点的父节点层或者 Node 不包含 Leaf

22. SeqArray[Father.LevelID]←SeqArray[Father.LevelID]＋1：序号加 1，继续判断下一个孩子节点

23. WhileSeqArray[Father.LevelID]＝ Father.NumChildrenDo：如果所有孩子节点已遍历完毕

24. Father←Father.ParentNode：回退至上层节点

25. If Father＝NULL Then：如果根节点所有孩子遍历完毕

26. Break Loop：退出循环

27. End If

28. SeqArray[Father.LevelID]←SeqArray[Father.LevelID]+1：某节点已遍历，开始遍历其右兄弟节点

29. End While

30. End If

31. End While

32. If NodeSet.IsEmpty()=True Then：如果NodeSet为空，表示树中不存在包含TNode的节点

33. NewRoot←Root：将根节点Root赋给变量NewRoot

34. Else IfMinLevelID=1 Then：如果第一层中存在满足条件的节点

35. NewRoot←LargestNode(NodeSet)：选择其中时空范围最大的节点作为NewRoot

36. Else

37. 从NodeSet全部节点的所有孩子节点集合Set中选择包含TNode后时空范围变化最小的节点作为NewRoot

38. End If

39. 以NewRoot为根节点，采用传统节点选择算法从上而下搜索第1层节点作为输出结果

3.3 时空索引缓存机制

实时应用中，轨迹索引创建过程通常是按照数据元组的有效时间顺序实施，一般来说当前时间是查询焦点。最近被访问的数据近期被再访问的几率较大，如果缓存中持有这些数据，可减少外存访问次数。时空数据库的元组记录数目数以亿计，中等规模数据库的时空索引数据量就超过GB字节，不允许在内存中构建全部索引然后持久化至数据库中。同时，树型结构通常在堆内存中动态分配节点空间，全部维持在缓存中极易导致系统性能不稳定。

为此，本文设计一个基于LRU算法（近期最少使用算法）的时空索引缓存方法，其原理是，设定一个时间阈值T，每间隔一定时间清理时空索引树缓存，从缓存中释放从当前时间算起已有T时间没有被访问的节点数据，后面需要可再根据父节点中的子节点ROWID访问外存，图2描述了本索引缓存机制。为此，节点结构定义一个变量T_{LRU}用于记录节点最近被访问时间，节点一经访问，其T_{LRU}即被更新为当前时刻。由于树状索引的操作通常以根节点为入口，满足条件则向下进入子节点，不满足则剪枝不再深入，访问子节点的同时，必定也访问父节点，保证父节点的T_{LRU}不早于子节点。因此，如果父节点过时，以其为根节点的子树中所有节点均过时，可从索引缓存中释放子树。本文采用C++智能指针管理R树索引中的缓存节点指针，释放父节点时其子节点的引用计数自动减一，自上而下安全释放所有子孙节点。

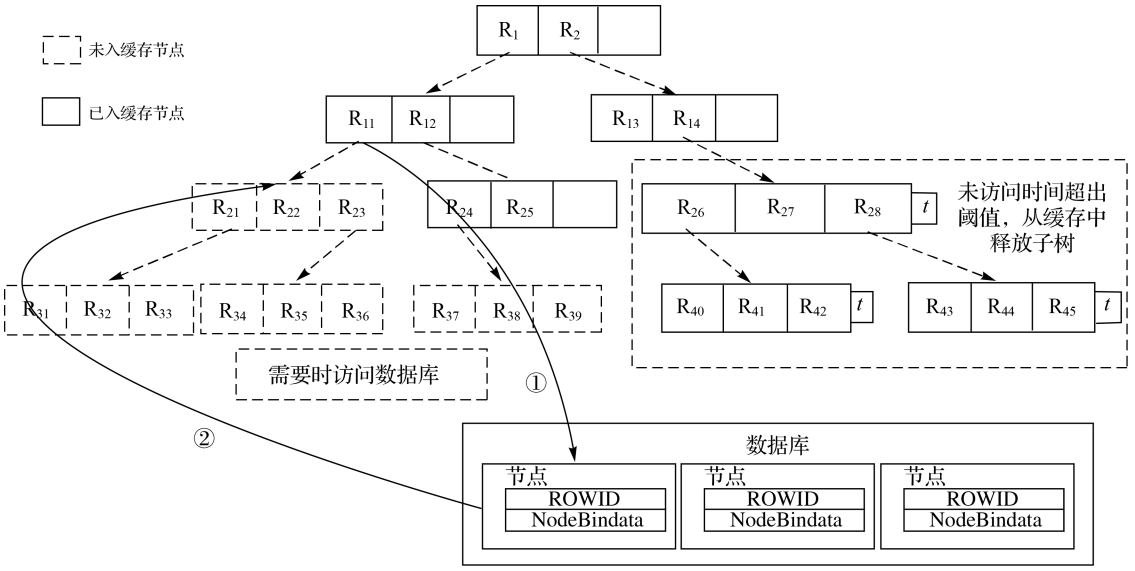

图2 时空R树索引缓存机制

Fig.2 Cache mechanism for spatio-temporal R-tree

4 时空R树索引存储设计方案

呈指数增加的移动应用用户使得移动对象数据管理方法需要云存储支持[19]。本文选择采用NoSQL数据库MongoDB作为实现HBSTR树的存储介质。MongoDB的组织结构是数据库-数

据集-文档-元素,数据集类似于关系数据库中的表,文档类似于记录,元素类似于字段。MongoDB 数据集内的文档即记录不要求结构严格相同,适合存储结构不固定的数据内容集。

本节将重点介绍时空 R 树的数据库存储方案。时空 R 树索引结构存储于独立数据集中,为便于识别,时空索引数据集的名称是对应时空数据集的名称后缀加.STI_Index,如数据集 DataSet 的时空 R 树索引数据集为 DataSet.STI_Index。时空 R 树索引元信息包括数据库名称、数据集名称、索引数据集名称、空间维度、时空 R 树扇出参数、节点数目以及根节点的 ROWID 号。根节点的 ROWID 号是根节点文档的唯一标识符,通过它从数据库中读取根节点数据,进而读取 R 树中任何节点的数据。MongoDB 的每条文档都有一个 ROWID,如果用户不指定,系统自动根据服务器、进程和当前时间等信息创建类型为 12 个字节的数组,它是 MongoDB 实现分布式存储的重要基础。为了方便查找时空索引元信息,元信息被集中存储于用户指定 ROWID(如"999999999999")的一条文档中。除元信息外,单个时空 R 树节点也被作为文档存入数据集,节点分为叶节点和非叶节点,叶节点存储元组集合信息,非叶节点存储孩子节点集合信息,二者结构不同,为存储方便,本文将节点数据和元信息组织为二进制块即 BinData 类型元素存入文档。叶节点另外记录对象标识符和连续轨迹采样点,非叶节点则记录孩子节点信息,如孩子节点的时空范围等。R 树中的父节点保存孩子节点的外存地址(ROWID)和 MBR 等作为指针项,访问父节点即可知道其孩子节点是否满足粗查要求。本文详细存储方案可参照文献[20]。

5　试验方法与结果分析

本节从索引创建和时空查询处理等方面综合测试 HBSTR 树,由于 TB* 树是当前综合性能最优的轨迹索引,本文选择 TB* 树与 HBSTR 树进行分析对比。为了试验对比的公正性和科学性,TB* 树中的叶节点仅在充满后插入树中。

5.1　试验环境设置

HBSTR 树的数据页尺寸设置为 3 kB,时空 R 树叶节点和非叶节点的最大扇出参数分别为 80 和 40,非叶节点的最小扇出参数是最大扇出参数的 40%,即为 16。除叶节点的双向链表指针外,TB* 树具有相同于时空 R 树的结构,因此 TB* 树的扇出参数可设为相同。空间坐标采用双精度浮点型数据类型,时间坐标采用 64 位整型数据类型。试验运行软硬件是:64 位 Window 7 操作系统,64 位 MongoDB 数据库,Intel Core I7-3770M 3.40 GHz 中央处理器,8 GB 主存和 1 TB 外存(7200 rpm,64 MB 高速缓存)。

本文采用 Brinkhoff 的时空数据生成器生成不同规模的数据样本,该生成器被广泛用于时空数据库的性能测试[21]。表 1 是试验采用的 4 份不同规模数据集的描述,这些数据集基于德国 Oldenburg 市的真实路网数据生成,它们具有相同时空区间,空间范围$[x_{\min},y_{\min},x_{\max},y_{\max}]$均为[281,3935,23 854,30 851],而移动对象数目和轨迹点数目存在差异。

表 1　试验样本数据集描述

Tab.1　Description of experimental datasets

名称	时间戳数目	对象数目	轨迹点数目
O5000K	1000	14 000	5 271 991
O10000K	1000	26 000	10 579 838
O20000K	1000	48 000	20 845 388
O40000K	1000	90 000	40 819 855

5.2　索引创建的时间性能和存储效率

对 TB* 树和 HBSTR 树的创建效率进行对比试验,二者均采用经典 R 树的平方级分裂策略。表 2 是索引创建的试验结果,其中包括时间开销和存储代价。以 O5000K 数据样本为例,创建 TB* 树花费 439 s,而 HBSTR 树花费 454 s,上述时间开销均涵盖样本数据访问和索引数据持久化的开销。

假设插入操作为原子操作,TB* 树和 HBSTR 树的创建算法的时间复杂度均为 $O(N/M)$,其中,N 是轨迹点数目,M 是叶节点的最大扇出参数。R 树插入操作是费时过程,其中包括节点选择和节点分裂两个环节。HBSTR 树中的节点选择操作比 TB* 树复杂,时间复杂度为 $O(\log_M N)$。除 HBSTR 树的时空 R 树评价因子外,节点分裂过程和经典 R 树相同,因此时间复杂度为 $O(A^2)$,其中 A 是非叶节点的最大扇出参数。创建 HBSTR 树的节点选择和节点分裂操作次数略微高于 TB* 树,因此 HBSTR 树的时间开销更大。需要指出的是,节点选择操作次数等于叶节点数目,节点分裂操作次数等于非叶节点数目。

表 2 索引创建试验结果(以 O5000K 数据样本为例)

Tab.2 Experimental results of index generation (dataset O5000K)

索引	时间/s	节点数目	节点选择次数	节点分裂次数	存储开销/(MB)
TB* 树	439	76 652	73 734	2918	217
HBSTR 树	454	76 814	73 734	3080	221(216+5)

评价索引性能中一项容易被忽略的因素是索引结构数据量。HBSTR 树有一个内存子索引(Hash)和两个外存子索引(R 树和 B* 树),因此其存储代价为 R 树和 B* 树二者之和。以 O5000K 数据样本为例,HBSTR 树的数据量是 221 MB,其中两个子索引分别为 216 MB 和 5 MB。由于非叶节点的最小扇出参数为 16,R 树节点中的绝大多数超过 94% 是叶节点。鉴于 HBSTR 树的原理,HBSTR 树中时空 R 树大部分叶节点的存储空间能够被完全利用。子索引 B* 树仅索引时空 R 树叶节点,其索引键是叶节点的 OID/timestamp,因此,其存储代价远小于 R 树。为显式保存轨迹,TB* 树另外维护叶节点间的双向链表结构,需要额外的存储空间。相对于 675 MB 的原始数据样本,TB* 树和 HBSTR 树均得以明显压缩。

5.3 时空范围查询处理

查询时空范围内的轨迹点是轨迹数据库中最常见的操作,本节对 TB* 树和 HBSTR 树进行时空范围查询处理性能对比和分析。本节采用 3 组范围查询条件(Q1—Q3)进行试验测试,每组包含 100 个随机查询窗口条件,3 组查询条件分别是有效时空范围的轴向区间的 1%、2%和 4%,并将 3 组查询条件均实施于不同规模的合成数据集(O5000K-O40000K)。

将 100 次窗口查询处理数据平均值作为试验结果,图 3 是不同数据规模的范围查询效率对比。在 Q1—Q3 等 3 组查询条件上,HBSTR 树相对于 TB* 树具有明显的查询优势。TB* 树和 HBSTR 树有相同的叶节点集合,因此,查询命中的叶节点集合是相同的。

查询范围越大,叶节点命中几率越大,占全部被访问节点的百分比更高。同时,叶节点显著多于非叶节点,因此,当查询条件范围变大时,访问节点数目差异进一步增加,但是相对差距缩小。

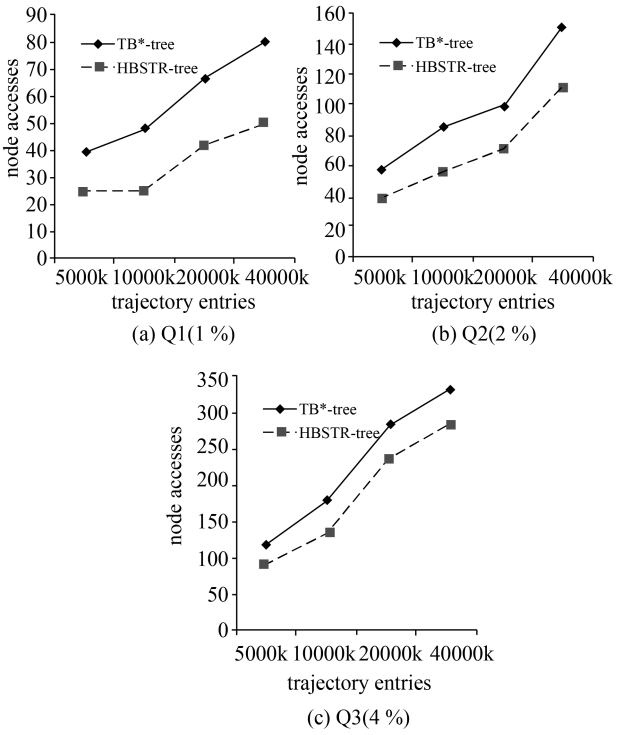

图 3 时空范围查询效率对比

Fig.3 Performance comparison in spatio-temporal range queries

5.4 目标对象轨迹查询

另一个常见操作是查询某时间区间内指定目标的移动轨迹。分别从 4 个数据集合中随机选择 100 个移动对象,同时,随机选择移动对象生命周期的 10%作为轨迹试验的查询条件。表 3 是上述 4 组数据集的试验结果,轨迹查询操作中节点访问数目的平均值作为试验结果。

HBSTR 树中,B* 树子索引直接定位满足查询条件的首叶节点,然后向后扫描和逐一输出满足条件的叶节点,以致避免多余节点访问。相反,TB* 树从根节点搜索整个树结构。在首个目标叶节点命中后,叶节点间的双向链表被用于搜索满足条件的叶节点。显然,上述查询过程并不稳定可靠,有时可能会遍历全树。由于一个对象的 10%连续轨迹点存在于 1~2 个叶节点,HBSTR 树仅需访问 1~2 个叶节点,另外需要访问 B* 树子索引,其节点访问也要考虑在内。假如 B* 树节点中最大关键码数目是 100,O5000K、O10000K、O20000K 和 O40000K 的 B* 树层数均为 3。在一次查询操作中,B* 树每层仅一个节点被访问。

表 3　轨迹查询处理的节点访问数目

Tab.3　Node accesses in trajectory queries

索引类型	节点访问数目			
	O5000K	O10000K	O20000K	O40000K
TB* 树	52	77	122	178
HBSTR 树	5(2+3)	5(2+3)	5(2+3)	5(2+3)

6　结　论

本文提出的轨迹索引方法充分利用 R 树、B* 树和哈希表等索引的优点，支持时空范围查询和目标对象轨迹查询等多种查询方式，在索引创建效率上和 TB* 树相近，在时空查询效率上明显优于 TB* 树，满足大规模轨迹数据库的实时更新管理要求，适应时空查询处理的多样化需求。同时提出了一种数据库存储管理方案，可以支持云存储业务需求。大规模实时轨迹数据管理是智能交通和社交网络等焦点应用的支撑技术，后续工作还要针对复杂地学应用增加时空过程模拟与分析能力。

参考文献：

[1] YE Li. Research on Data Queries and Processing Techniques in Moving Objects Databases[D]. Chengdu: University of Electronic Science and Technology of China, 2011.(叶李. 移动对象数据库查询及处理技术研究[D]. 成都：电子科技大学，2011.)

[2] YIN Zhangcai, LI Lin. Research of Spatiotemporal Indexing Mechanism Based on Snapshot-increment[J]. Acta Geodaetica et Cartographica Sinica, 2005, 34(3): 257-261, 282.(尹章才，李霖. 基于快照-增量的时空索引机制研究[J]. 测绘学报，2005，34(3)：257-261，282.)

[3] LIAO Wei. Indexing and Query Processing on Moving Objects in Location-based Services[D]. Changsha: National University of Defense Technology, 2007.(廖巍. 面向位置服务的移动对象索引与查询处理技术研究[D]. 长沙：国防科学技术大学，2007.)

[4] HAO Zhongxiao. New Theories of Spatio-temporal Database[M]. Beijing: Science Press, 2011.(郝忠孝. 时空数据库新理论[M]. 北京：科学出版社，2011.)

[5] LI Qingquan, HUANG Lian. A Map Matching Algorithm for GPS Tracking Data[J]. Acta Geodaetica et Cartographica Sinica, 2010, 39(2): 207-212.(李清泉，黄练. 基于 GPS 轨迹数据的地图匹配算法[J]. 测绘学报，2010，39(2)：207-212.)

[6] XU Lin, LI Qingquan, YANG Bisheng. A Moving Object Index Structure and Near Neighbor Query Method Based on Road Network[J]. Acta Geodaetica et Cartographica Sinica, 2010, 39(3): 316-321, 327.(许林，李清泉，杨必胜. 一种基于道路网的移动对象的位置索引与邻近查询方法[J]. 测绘学报，2010，39(3)：316-321，327.)

[7] PARENT C, SPACCAPIETRA S, RENSO C. Semantic Trajectories Modeling and Analysis[J]. ACM Computing Surveys, 2013, 45(4): 42: 1-42: 32.

[8] ZHENG Y, ZHOU X. Computing with Spatial Trajectories [M]. New York: Springer-Verlag, 2011.

[9] FRENTZOS E. Trajectory Data Management in Moving Object Databases[D]. Piraeus: University of Piraeus, 2008.

[10] MOKBEL M, GHANEM T, AREF W. Spatio-temporal Access Methods[J]. IEEE Data Engineering Bulletin, 2003, 26: 40-49.

[11] NGUYEN-DINH L, AREF W, MOKBEL M. Spatio-temporal Access Methods: Part2 (2003-2010)[J]. IEEE Data Engineering Bulletin, 2010, 33: 46-55.

[12] CHAKKA V, EVERSPAUGH A, PATEL J. Indexing Large Trajectory Data Sets with SETI[C]// Proceedings of the 2003 CIDR Conference. Asilomar: [s.n.], 2003.

[13] MA Linbing, ZHANG Xinchang. Research on Full-period Query Oriented Moving Objects Spatio-temporal Data Model[J]. Acta Geodaetica et Cartographica Sinica, 2008, 37(2): 207-211, 222.(马林兵，张新长. 面向全时段查询的移动对象时空数据模型研究[J]. 测绘学报，2008，37(2)：207-211，222.)

[14] GUO Jing, LIU Guangjun, GUO Lei, et al. A Whole-time Index Design Based on 3D+-TPR-tree for Moving Point Targets[J]. Acta Geodaetica et Cartographica Sinica, 2006, 35(3): 267-272.(郭晶，刘广军，郭磊，等. 基于 3D+-TPR-tree 的点目标全时段移动索引设计[J]. 测绘学报，2006，35(3)：267-272.)

[15] GONG Jun, KE Shengnan, ZHU Qing, et al. An Efficient Management Method for Point Cloud Data Based on Octree and 3D R-tree[J]. Acta Geodaetica et Cartographica Sinica, 2012, 41(4): 597-604.(龚俊，柯胜男，朱庆，等. 一种八叉树和三维 R 树集成的激光点云数据管理方法[J]. 测绘学报，2012，41(4)：597-604.)

[16] PFOSER D, JENSEN C, THEODORIDIS Y. Novel Approaches to the Indexing of Moving Object Trajectories [C]// Proceedings of the International Conference on Very Large Data Bases. Cairo: [s.n.], 2000.

[17] AGGARWAL C. Managing and Mining Sensor Data[M]. New York: Springer Publishing, 2013.

[18] GONG Jun, ZHU Qing, ZHANG Yeting, et al. An Efficient 3D R-tree Extension Method Concerned with Levels of Detail[J]. Acta Geodaetica et Cartographica Sinica, 2011, 40(2): 249-255.(龚俊，朱庆，张叶廷，等. 顾及多细节层次的三维 R 树索引扩展方法[J]. 测绘学报，2011，40(2)：249-255.)

[19] LEE D, LIANG S. Geopot: A Cloud-based Geolocation Data Service for Mobile Applications[J]. International Journal of Geographical Information Science, 2011, 25(8): 1283-1301.

[20] KE S, GONG J, LI S, et al. A Hybrid Spatio-temporal Data Indexing Method for Trajectory Databases[J]. Sensors, 2014, 14(7): 12990-13005.

[21] BRINKHOFF T. A Framework for Generating Network-based Moving Objects[J]. GeoInformatica, 2002, 6(2): 153-180.

(责任编辑：陈品馨)

收稿日期：2013-12-17

修回日期：2014-10-27

第一作者简介：龚俊(1978—)，男，博士，教授，研究方向为时空数据库。

First author: GONG Jun(1978—), male, PhD, professor, majors in spatio-temporal database.

E-mail: gongjunbox@163.com