doi: 10.3969/j.issn.1005-3026.2016.11.005

QGrid:一种空间移动对象并行索引结构

李传文, 车庆首, 谷 峪, 邓庆绪
(东北大学 信息科学与工程学院, 辽宁 沈阳 110819)

摘 要: 为提高空间移动对象数据更新效率和查询准确率,提出了一种空间移动对象并行索引结构. 利用主索引和辅助索引支持对空间对象进行基于范围的查询和基于对象标识的查询,还通过查询索引将更新操作和可能受其影响的查询操作相连接,在满足并行操作时间片语义的同时,避免了传统方法进行范围查询时对查询范围内相关对象及相关索引结构全部进行锁定的需求. 实验结果表明:高负载环境下,该索引结构不但能保证查询准确率,其处理能力也明显优于传统索引结构. 该索引通过提高系统并行度,使同一范围内的更新和查询操作可以并行执行,提升了系统整体运行效率.

关 键 词: 空间移动对象;空间数据索引;范围查询;时间片查询;并行更新
中图分类号: TP 311.13 **文献标志码**: A **文章编号**: 1005-3026(2016)11-1541-06

QGrid: A Parallel Indexing Structure for Spatial Moving Objects

LI Chuan-wen, CHE Qing-shou, GU Yu, DENG Qing-xu
(School of Information Science & Engineering, Northeastern University, Shenyang 110819, China. Corresponding author: LI Chuan-wen, E-mail: lichuanwen@ise.neu.edu.cn)

Abstract: In order to improve updating efficiency and querying accuracy for spatial moving object data, a parallel indexing structure for spatial moving objects was proposed. A main index and an auxiliary index were contained in the proposed structure, which were used for supporting range based and identity based spatial object query operations, respectively. A query index was also utilized which hooks updating operations to querying operations that may be influenced. Thus locking relevent spatial objects and indexing structures could be avoided when range query operations are processed. At the same time, it also support timeslice semantics for parallel operations. Experimental results show that, under high working load, the structure can not only guarantee querying accuracy, the throughput is also obviously higher than that of the existing methods. The index improves the degree of system parallelism, makes it possible for object updating and querying operations in same ranges be processed in parallel, and therefore improves the overall efficiency of the system.

Key words: spatial moving objects; spatial data indexing; range query; timeslice query; parallel updating

对空间移动对象数据进行频繁的更新和查询是很多基于位置服务(location based service, LBS)应用的基本操作[1-2]. 以目前流行的打车软件为例,应用系统需要实时更新车辆频繁提交的位置信息,同时还需要根据打车用户的位置进行范围查询,按距离由近到远的顺序找到相关车辆. 为满足这种大数据量的空间数据更新和查询要求,高效的空间移动对象存储和索引结构以及相应的更新和查询算法成为空间数据库领域研究的重点[3].

传统数据库系统由于其底层大都基于B+树[4]等结构,仅在索引线性数据时较为高效,而空间位置信息一般是二维甚至高维数据. 另外,传统数据库一般只索引不常变化的静态数据项,而

收稿日期: 2015-07-19
基金项目: 国家自然科学基金资助项目(61300021); 中央高校基本科研业务费专项资金资助项目(N140404008).
作者简介: 李传文(1982-),男,山东烟台人,东北大学讲师,博士; 邓庆绪(1970-),男,河南南阳人,东北大学教授,博士生导师.

空间对象的更新操作绝大多数是对参与索引的空间对象位置信息进行修改. 因此,传统数据库系统难以直接用于存储空间移动数据.

现有的针对空间移动数据的索引结构主要分为支持时间片语义和支持鲜度语义两大类[5]. 不同语义的主要区别在于更新及查询操作的可串行性不同,以及因此造成的操作执行效率和结果准确率的不同. 支持时间片语义的索引结构一般满足严格的可串行性,结果准确率较高,但操作执行效率相对较低,而支持鲜度语义的则恰恰相反. 如何在保证结果准确率的同时提高操作执行效率,是目前空间移动数据索引问题的一个难点[6-7].

鉴于此,本文提出一种支持时间片语义的空间对象并行索引结构,通过将满足时间片语义的时刻设定为查询结束时刻,避免了传统方法在查询开始时必须对查询范围内的对象和相关索引结构加锁的需求,使查询期间同一范围内的更新操作可以并行执行,从而提高了系统的整体运行效率.

1 QGrid 索引

1.1 问题定义

本文研究对平面空间内大量移动对象的位置进行索引以支持范围查询的数据结构. 每个对象被视为平面上的一个点,其在移动过程中定期向服务器发送信息,内容包括对象标识(id)及当前位置(x,y). 每个查询包含两个点$(x_{\min},y_{\min})$和$(x_{\max},y_{\max})$,代表矩形查询框的左下和右上两点.

为提高系统执行效率,空间对象的更新和查询操作被尽量以并行方式执行,其理想情况是实现事务的可串行性(serializable),即多个操作被并行执行得到的结果和这些操作串行执行得到的结果是完全一致的[8]. 时间片语义体现了这种可串行性.

定义 1 (时间片语义, timeslice semantics)查询返回某一时间点上在查询框内的所有对象.

现有索引结构中,定义 1 中这一时间点往往被默认为查询开始时刻. 由于需要在查询开始时将查询框涉及的对象进行锁定以保证可串行性,系统的执行效率被大大降低[9]. 目前有多种通过减弱对事务可串行性的要求以提高系统执行效率的语义,其中最先进的是 Darius 等提出的鲜度语义[10].

定义 2 (鲜度语义, freshness semantics)记查询框为 q,查询开始时刻为 t_s,结束时刻为 t_e,记对象 o 在 t_s 和 t_e 时刻的位置分别为 $o_{\text{pos}}^{\bullet}$和 o_{pos}°,假定任何对象在$[t_s,t_e]$时间段内最多只更新一次,则任何对象 o 与查询结果 R 的关系满足:

$$\left.\begin{aligned} &o\in R, o_{\text{pos}}^{\bullet}\in q\wedge o_{\text{pos}}^{\circ}\in q;\\ &o\notin R, o_{\text{pos}}^{\bullet}\notin q\wedge o_{\text{pos}}^{\circ}\notin q. \end{aligned}\right\} \tag{1}$$

由于一次查询的执行时间(t_e,t_s)远短于同一对象两次更新之间的时间间隔,因此上述定义中对于对象更新次数的假定是合理的.

虽然鲜度语义可提供较高的更新和查询效率,但由式(1)可以看出,这种语义下查询结果并不精确:如果某对象在查询过程中移出或移入查询框(即 $o_{\text{pos}}^{\bullet}\in q\wedge o_{\text{pos}}^{\circ}\notin q$ 或 $o_{\text{pos}}^{\bullet}\notin q\wedge o_{\text{pos}}^{\circ}\in q$),那么这个对象是否被包含在查询结果中并不确定.

1.2 索引结构

本索引结构的目标是以接近鲜度语义的执行效率提供时间片语义的结果准确率. 其核心思想是:①通过放弃提供查询到达时刻的系统快照,从而避免对查询区域的锁定;②通过对查询执行过程中可能影响查询结果的更新操作进行记录,从而在查询结束时刻得到符合时间片语义的结果. 由于不需要锁定查询框内的对象,该索引与基于传统时间片语义的索引结构相比具有较高的更新效率.

要达到上述目标,索引结构需满足下列要求:查找空间区域内的对象;通过标识找到对象;查找空间区域内正在执行的查询操作. 因此,本文提出一种面向查询的栅格索引结构(query oriented grid index, QGrid),由 3 部分构成:主索引、辅助索引和查询索引.

被 QGrid 索引的目标数据有 2 种:空间对象和查询操作. 记空间对象集为 $O=\{o_1,\cdots,o_n\}$,其中任一对象 $o_i=\{\text{id},x,y,t\}$,id 为 o_i 的唯一标识,x,y 为位置,t 为最近一次更新时间. 正在处理的查询操作集 $Q=\{q_1,\cdots,q_e\}$,其中任一查询请求 $q_j=\{x_{\min},y_{\min},x_{\max},y_{\max},l\}$,前四项定义了一个矩形查询框,$l$ 为保存与 q_j 相关的更新操作的链表.

主索引和辅助索引采用不同的方式对同一组空间对象 O 进行索引,而查询索引对所有正在执行的查询请求集合 Q 进行索引.

定义 3 (主索引, primary index)主索引采用 Hilbert 曲线 $H=\{h_1,\cdots,h_m\}$ 填充整个空间,记曲线上相邻节点 h_i 与 h_{i+1}之间的距离为 r,每个节点保存以 h_i 为中心、边长为 r 的正方形区域内所有空间对象.

以图 1 为例，左边被 Hilbert 曲线[11]填充的部分为整个空间，取出图中圆圈所示部分放大（放大部分为图中与之相连的大圆），图中标示的节点 h_i 保存的空间区域放大为图中右边方形区域. 其中每个圆点代表一个空间对象，圆点覆盖的区域为该对象可能存在的区域. 给定距离阈值 δ 及对象的最大移动速度 $v_{\max}$，令 t_n 为当前时间，则圆点的圆心为 $(o.x, o.y)$，半径为 $v_{\max}(t_n - o.t) < \delta$.

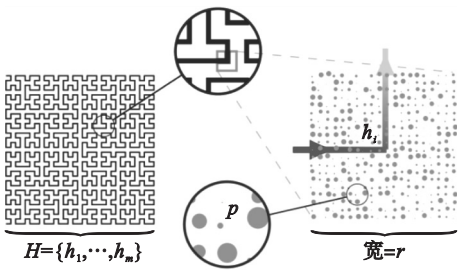

图 1　主索引结构示例
Fig. 1　Example of the main index

主索引的实现采用 B + 树[4]，其中存储 m 个指针，每个指针 p_i 指向一个链表 L_i，该链表包含属于节点 h_i 的所有空间对象. 在 B + 树中，指针 $p_1,\cdots,p_m$ 按照其对应的节点 $h_1,\cdots,h_m$ 在 H 中的顺序排列.

定义 4　（辅助索引，auxiliary index）辅助索引采用哈希表 S 根据 id 值对所有空间对象进行索引.

主索引与辅助索引都是对同一组空间对象 O 索引，然而它们的功能不同：主索引是为了加快基于空间位置的查询，提供对范围查询操作的支持，而辅助索引是为了加快基于对象 id 的查询，提供对对象更新操作的支持.

定义 5　（查询索引，query index）查询索引采用 R * 树，R 将所有正在处理的查询请求 Q 根据各自的查询框进行索引.

查询索引的功能是协助更新操作快速定位可能受其影响的查询，并将更新操作记录到相关查询中. 由于存在偏差 δ，查询时需要对查询框沿 x 轴和 y 轴分别向两端扩大（缩小）δ 以查询所有可能（肯定）在原始查询框内的对象.

2　QGrid 算法

空间移动对象涉及两种基本操作：更新操作和查询操作. 本节介绍 QGrid 索引结构对这两种操作提供支持的算法.

2.1　空间对象更新算法

算法 1 空间对象更新 update_object
输入：更新 $u = \{\mathrm{id}, x, y, t\}$，$H, S, R$
输出：无输出，操作完成后空间对象及其索引被更新

```
1. hook_queries(o,u,R)
2. o = find_object_by_id(id,S)
3. C_old = find_cell(o.x,o.y,H)
4. C_new = find_cell(u.x,u.y,H)
5. if(C_new == C_old))
6.       update_ object_position(o,u)
7. else
8.       delete_from_cell(C_old,o)
9.       add_to_cell(C_new,creste_ object(u))
```

如算法 1 所示，空间对象更新操作按位置变化情况分为本地和非本地两类. 如果更新为本地操作，即对象更新前后属于同一个空间节点，则只需修改对象位置；而如果更新为非本地操作，则需要先将对象从原节点中删除，再将其添加到目标节点中. 注意算法 1 第 8，9 行删除和加入对象时并不需要锁定相关的索引结构.

算法 1 中对象位置更新前，需先将更新操作记录到可能受其影响的查询中，即算法 1 中第 1 行 hook_queries 操作. 基本流程如算法 2 所示.

算法 2 记录更新操作 hook_queries
输入：o, u, R
输出：无，操作完成后 u 与受其影响的查询连结在一起

```
1. l_o = find_enclosing_queries(o.x,o.y,R)
2. l_u = find_enclosing_queries(u.x,u.y,R)
3. L_e = xor(L_u,L_o)
4. For esch(q in l_e) add_to_list(q.l,u)
```

如果一个查询操作受到某更新操作的影响，则说明此更新操作对应的对象移入或者移出此查询的查询框. 因此，先在 R 中分别找到对象更新前后各自所处的查询框列表 L_o 和 L_u，然后对这 2 个列表取异或，得到的结果列表 L_e 即包含所有可能受此次更新影响的查询操作.

2.2　空间对象查询算法

算法 3 空间对象查询 query_objects
输入：查询 q, H, S, R
输出：查询结束时 q 查询框中包含的所有空间对象列表 l

```
1. insert_query_to_rtree(q,R)
2. l_c = find_overlapping_cells(q,H)
```

```
3. l_o = Φ
4. for each(c in l_c)
5.     for each(o in c)
6.         if(is_enclosed(o.x, o.y, q))
7.             add_to_list(l_o, o)
8. for each(u in q.l)
9.     if(is_enclosed(u.x, u.y, q))
10.        add_to_list(l_o, find_object_by_id(u.id, S))
11.    else
12.        del_from_list(l_o, find_object_by_id(u.id, S))
13. delete_query_from_rtree(q, R)
```

如算法 3 所示,为保证查询结果的正确性,需对两类空间对象进行判断. 首先是主索引 H 中与查询框重叠的节点内保存的对象(第 2 ~ 7 行),判断这些对象是否在查询框内,如果是则加入到结果集. 在判断过程中,由于可能的并行更新操作,已加入到结果集的对象可能移出了查询框,也可能有新的对象移入了查询框. 因此,还需要对在查询过程中执行了并行的更新操作而可能影响到查询结果的对象进行判断(第 8 ~ 12 行). 这些对象都是在对象更新的过程中由 hook_queries 操作加入到 $q.l$ 中的.

分析算法 3 可知,算法的输出结果为即将运行第 13 行前的时刻 q 查询框内的所有对象,即算法满足这一时刻的时间片语义.

3 仿真实验与结果分析

本节将 QGrid 索引结构与目前最先进的基于鲜度语义的 PGrid 索引结构[10]以及基于时间片语义的 TwinGrid 索引结构[12]进行对比.

实验仿真环境为 Redhat Linux 9 系统, Intel Xeon E5 - 2620 v3 六核 CPU × 2,16 GB 内存;仿真程序由 C + + 实现并由 g + + 5.1 编译;空间对象数据由基于 Brinkhoff[13]算法的开源移动对象生成工具 MOTO^2 生成. 实验参数如表 1 所示.

图 2 给出了处理能力随线程数变化的实验结果,其中处理能力由索引结构在 1 s 内能够执行的更新或查询操作工作量表示. 由图可看出,在线程数不大于 12 时,3 种索引结构的处理能力都随着线程数的增大而逐渐增大,其中 QGrid 索引结构的处理能力明显高于其他两种索引结构. 这是因为 QGrid 索引结构采用了对查询进行索引从而避免锁定查询区域的方式,相比另外两种索引结构采用的锁定查询区域的方式具有更高的并行度.

表 1 实验参数

Table 1 Experiment parameters

参数	实验值	默认值
空间对象数 $\times 10^{-6}$	5,10,20,40	10
更新操作总次数 $\times 10^{-6}$	100	—
线程数	1,2,4,6,8,10,12,14,16	12
(更新/查询比) $\times 10^{-3}$	0.25,0.5,1,2,4,8,16	1
总面积/km^2	800 × 800	—
查询面积/km^2	1,2,4,8,16,32	4

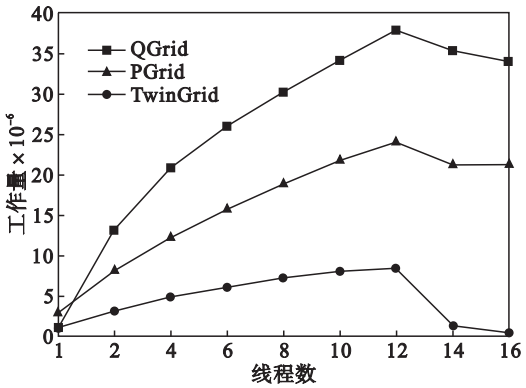

图 2 线程数对处理能力的影响

Fig. 2 Effect of thread numbers on throughput

从图中还可以看出,3 种索引结构的处理能力在线程数大于 12 后都呈下降趋势. 这是因为实验平台采用了两片各有 6 个核心的 CPU,即具有 12 个实际的硬件线程. 当实验程序调用的软件线程数超过了硬件能够直接支持的线程数后,线程之间相互抢占,反而降低了总处理能力. 以下实验中将设定线程数为 12.

图 3 给出了处理能力随空间对象总数变化的

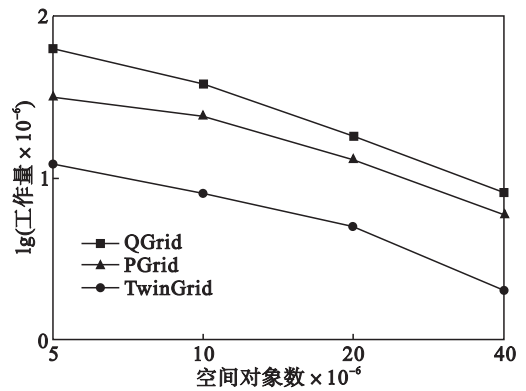

图 3 空间对象数对处理能力的影响

Fig. 3 Effect of spatial object numbers on throughput

实验结果. 由图可见,随着空间对象总数的增加,3 种索引结构的处理能力都逐渐下降. 这是由于总对象数的增加使保存在同一索引节点中的对象数增加,因而每次更新或查询时需要遍历更多的空间对象. 在 3 种索引中,QGrid 索引的处理能力始终高于其他两种索引结构.

查询面积对处理能力的影响由图 4 给出. 随着查询面积的增加,3 种索引的处理能力都有下降. 在大部分情况下,QGrid 的处理能力都高于其他两种. 图中 TwinGrid 索引结构在查询面积增加时处理能力下降较快,因为 TwinGrid 索引结构在查询时对查询区域的锁定粒度较大,造成大量更新操作处于等待状态. QGrid 索引结构在查询面积过大时处理能力稍弱于 PGrid 索引结构,这是因为此时 QGrid 的查询索引中存储的查询框之间重叠面积加大,造成执行更新操作时在查询索引中搜索的时间变长.

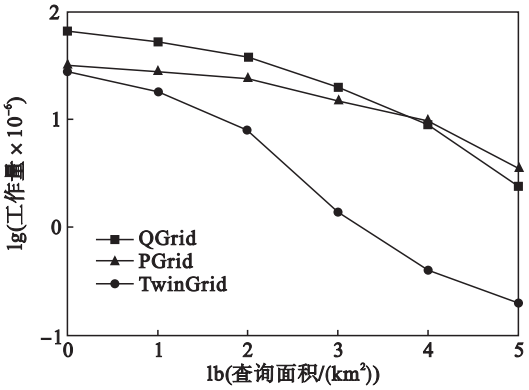

图 4　查询面积对处理能力的影响

Fig. 4　Effect of query size range on throughput

观察图 5 可发现,随着更新/查询比的增加,3 种索引的处理能力都有增加,但 QGrid 索引的处理能力始终优于其他两种索引结构. 随着更新/查询比的增加,PGrid 索引结构处理能力的增势放缓. 这说明PGrid索引结构处理查询操作占用的资源相对较少,进而说明 QGrid 索引处理能力高于 PGrid 的主要原因在于其更加高效的更新操作.

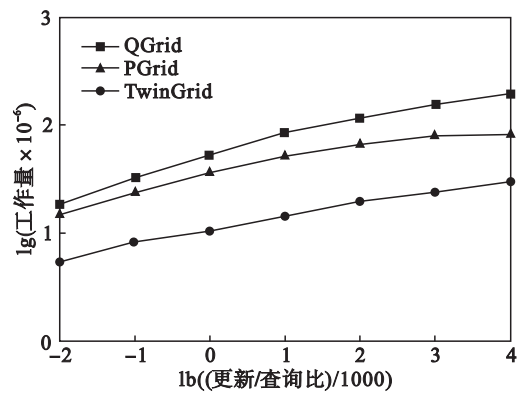

图 5　更新/查询比对处理能力的影响

Fig. 5　Effect of update/query ratio on throughput

上述实验结果表明,由于避免了更新和查询时对空间对象和索引结构的锁定,QGrid 索引结构可以更充分地发挥并行处理的优势,在大部分情况下其处理能力都要高于现有的索引结构.

4　结　　论

1）通过设计查询索引,将可能影响到查询的更新操作进行记录,并在查询执行过程中判断与之并行执行的更新操作对其是否有影响,可以避免锁定查询框内进行并行更新的空间对象,从而提高系统的并行度.

2）对同一组空间对象,采用主索引和辅助索引两种方式对其进行索引,可同时提高范围查询和基于对象标识查询的执行效率.

3）在空间对象索引系统运行过程中,将线程数设置为与 CPU 硬件核心数相同时,系统处理能力最高.

参考文献：

[1] Li C W, Gu Y, Qi J, et al. Processing moving kNN queries using influential neighbor sets[J]. *Proceedings of Very Large Database*, 2014, 8(2): 113 – 124.

[2] Li C W, Gu Y, Qi J, et al. A safe region based approach to moving KNN queries in obstructed space[J]. *Knowledge and Information Systems*, 2014, 45(2): 417 – 451.

[3] Samet H, Sankaranarayanan J, Auerbach M. Indexing methods for moving object databases: games and other applications [C] // Proceedings of the SIGMOD International Conference on Management of Data. New York: ACM, 2013: 169 – 180.

[4] Jensen C S, Lin D, Ooi B C. Query and update efficient B + – tree based indexing of moving objects [C] // Proceedings of Very Large Databases. Toronto: VLDB Endowment, 2004: 768 – 779.

[5] Nguyen-Dinh L V, Aref W G, Mokbel M. Spatio-temporal access methods: part 2 (2003 – 2010) [J]. *IEEE Data Engineering Bulletin*, 2010, 33(2): 46 – 55.

[6] Ward P G, He Z, Zhang R, et al. Real-time continuous intersection joins over large sets of moving objects using graphic processing units [J]. *The Very Large Database Journal*, 2014, 23(6): 965 – 985.

[7] Ray S, Blanco R, Goel A K. Supporting location-based services in a main-memory database [C] // Proceedings of Mobile Data Management. Brisbane: IEEE, 2014: 3 – 12.

[8] Gray J, Reuter A. Transaction processing: concepts and techniques [M]. San Francisco: Morgan Kaufmann Publishers, 1993: 373 – 445.

(下转第 1550 页)