

R-树中的频繁更新问题研究

潘锦基^{1,2} 刘景发¹ 马廷淮¹

摘要

在更新密集型应用中,索引结构容易成为系统性能的瓶颈,频繁更新是移动对象索引中的关键问题,由于更新性能很差,R-树不适用于更新密集型应用.改善 R-树的更新性能是一个挑战性的问题.分析了 R-树在更新中存在的问题,并对相关的解决方法进行了详细的分类与讨论.最后给出了有待解决的问题及进一步研究的方向.

关键词

移动对象索引;R-树;频繁更新;延迟更新

中图分类号 TP392

文献标志码 A

0 引言

Introduction

在移动对象数据库应用中需要管理大量的移动对象,同时连续移动对象的位置不断变化,而移动对象位置的改变会引起索引结构的动态更新.因此,索引移动对象不仅要考虑查询操作的效率,还必须重点考虑索引结构的更新代价问题.

移动对象数据库索引技术的关键是同时高效地支持查询和更新操作.R-树^[1]及其变体在空间索引结构中占据主导地位,但由于传统的空间索引的研究主要考虑静态数据,只关注高效的查询处理,R-树的更新性能却很差,不能直接用于频繁更新场合^[2].

近年来,国内外学者对 R-树中的更新问题进行了广泛的研究,在减少对象更新次数、改进对象更新算法、调整对象更新策略等方面都有了大量的研究成果.然而,当前的索引技术仍不能满足实际应用的需求,成为阻碍移动对象数据库应用的关键因素之一.

1 R-树及其存在的问题

R-tree and its existing problems

作为经典的空间索引结构,R-树的主要目标是支持高效的查询处理,而没有考虑对象的更新效率问题.

1.1 R-树

R-树^[1]是 B-树在多维空间的扩展,也是一棵高度平衡树,其插入算法基于最小外包矩形 (Minimum Bounding Rectangle, MBR) 的最小扩张准则,即以 MBR 递归地对数据空间按照面积规则进行划分,删除过程与插入过程相反.图 1 是一棵简单的 R-树示意图.

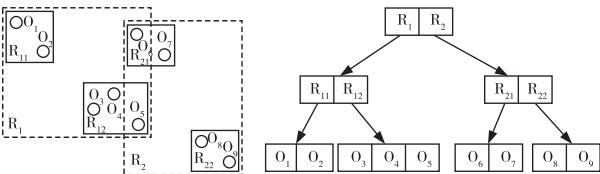


图 1 R-树结构示意图

Fig. 1 Structure of the R-tree

收稿日期 2009-11-20

资助项目 南京信息工程大学科研基金 (2007-0033)

作者简介

潘锦基,男,讲师,博士生,主要研究时空数据库及数据挖掘. jinjian@163.com

1 南京信息工程大学 计算机与软件学院,南京,210044

2 南京航空航天大学 信息科学与技术学院,南京,210016

R-树有 2 个关键参数:结点尺寸 (大小);插入和删除操作的顺

序. 结点大小一般选择磁盘块的倍数,其结构主要由溢出结点的分裂决定. 在 R-树中基于 1 种假设:查询是均匀分布的,即 1 个结点将来被某一查询访问的概率正比于该结点覆盖的面积. 为了提高查询性能, R-树在分裂时试图达到一种平衡,即分裂后的 2 个结点大致包含相同数量的孩子结点,这样可以使得相应 MBR 所覆盖的面积最小,从而提高查询性能.

1.2 R-树中的频繁更新问题

标准 R-树采用删除-重插机制来实现移动对象位置的更新,频繁更新会导致 R-树性能急剧下降.

从更新角度来看,R-树的缺点如下:

- 1) R-树直接存储对象的位置信息,但由于连续运动对象的位置会不断变化,导致频繁更新.
- 2) R-树结点的 MBR 允许重叠,因此,在查找旧索引项时需要遍历多条路径.
- 3) 为了提高查询性能,R-树要求 MBR 的边界尽可能紧凑,这会导致高昂的更新代价,因为边界上的对象很容易频繁地出入该 MBR,而每次删除或插入操作都可以引起合并与分裂操作.

2 R-树中的频繁更新问题

Frequent updates in R-trees

目前,已有很多的方法用于解决 R-树中的频繁更新问题. 概括起来,解决频繁更新问题的主要思路是:尽量减少更新次数;设法提高更新操作的效率. 如表 1 所示.

表 1 解决 R-树更新问题的方法

Table 1 Solutions for frequent updates in R-trees

方法	减少更新次数		提高更新效率		
	位置 预测	“容忍” 更新	算法 改善	批量 装载	延迟 更新
典型代表	TPR-树	LUR-树 CT-R 树	R ^R -树 TPR [*] -树 LUR-树	GBI	RUM-树 TPROM-树 R [*] -树

2.1 减少更新操作的次数

减少更新操作次数主要包括位置预测和“容忍”更新 2 种策略.

2.1.1 位置预测方法

处理频繁更新的 1 种常用方法是采用线性函数来表示移动对象的位置^[3],即数据库中并不直接存储移动对象的位置信息,而是保存其运动特性(主要是当前位置和速度参数),通过保存的参数来预测移动对象将来一段时间的位置(称为预测轨迹),该方

法又称为参数化方法. 使用预测轨迹,可以在满足一定精度的前提下,保证服务器中存储的对象位置与实际位置在相当长的时间内保持一致,从而大大减少了更新的次数. TPR-树^[4]就是采用这种建模方法. 在 TPR-树中引入时参范围矩形(Time-Parameterized Bounding Rectangle, TPBR)的概念,移动点的位置采用参考位置和速度向量来表示. 当分裂结点时,TPR-树要同时考虑移动点的位置及其速度.

为了解决 TPR-树不能有效支持对象标识查询缺陷,赵卿松等^[5]提出采用双索引结构的 BiR-树,即构建 1 棵以对象标识符(Oid)为关键字的 B⁺-树作为辅助索引,用于记录对象在 TPR-树中的位置,原有的 TPR-树结构和相应算法不变.

此外,TPR-树是以移动对象在更新时刻的瞬时速度为参考来判断是否发送更新信息或预测将来位置的,而实际应用中,对象的移动速度也会频繁改变^[6],这将导致索引结构的频繁更新,或查询结果与移动对象实际位置差别太大. 为了解决这个问题,学者们已经提出了很多的预测方法,如采用时间序列预测方法^[7]、基于灰色理论的预测方法^[8]、数据挖掘方法^[9]等.

然而预测方法只能解决一部分频繁更新问题,其缺点也很明显:预测的精度受到先前知识(对象的历史运动模式)的影响,而这些知识在很多情况下无法获得;对象在很多(如自由运动)情况下运动复杂,预测方法往往失效.

2.1.2 “容忍”更新

索引结构内在的对数据改变的容忍能力可以用来改善更新性能,R-树本身也具有一定的容忍改变的能力. 在更新时,如果对象的新位置没有移出原来的 MBR(即还处于同一个叶结点内),这时只要修改对应叶结点中的位置信息,不需要查找旧索引项,也不存在结点的分裂或合并操作.

因此,可以探索适当改变原索引结构的设计来增加容忍改变的能力,同时权衡可能增加的查询开销. 这方面的研究成果主要包括 LUR-树^[10]、CT-R 树^[11]等.

LUR-树^[10]的基本思想是只有当对象的新位置已经移出了原来对应的 MBR 时才更新索引;如果对象的新位置没有移出原来的 MBR(即还处于同一个叶结点内),这时只要修改对应叶结点中对象的位置信息即可. 为了进一步改善性能,LUR-树引入了扩展 MBR(Extended MBR, EMBR)的概念. 如果对象 p

的新位置只是稍稍移出原来的 MBR, 这时不用更新整个索引结构, 而是通过适当扩大对象 P 所在叶结点的 MBR, 使其包含 P 的新位置. 这样可以明显减少更新次数, 从而改善索引结构的总体性能.

CT-R 树^[11] 利用对象的历史更新信息来创建 CT-R 树, 以便于处理将来的更新. CT-R 树基于 R-树结构, 但是其 MBR 不仅仅由索引对象的当前值决定, 而是根据数据值改变的特性来定义. 这样可以有效地减少那些跨越边界的更新次数, 并能最大化延迟更新的发生. 由于将来的更新(或查询)不能准确预测, 因此文中假定过去的行为能指示将来的事件.

通过以上分析可以看出, “容忍”更新的实质是利用更新的局部性(Locality)特点, 以牺牲一部分查询性能为代价, 来明显改善更新性能, 从而提高索引的总体性能. 但该方法仅适用于索引那些在大部分时间内是缓慢变化的对象(类静止对象), 不适用于快速移动对象.

2.2 提高更新效率

提高更新效率的方法包括改进基本算法、批量装载和延迟更新方法等.

2.2.1 改进基本算法

可以通过修改基本索引结构中的插入、删除算法来提高算法的效率. 如 R^* -树^[12] 在结构上与 R-树完全相同, 但在对象插入路径的选择和结点分裂算法上做了改进, 并在插入的过程中进行“强制重新插入”, 以获得动态的结构重组. 尽管构造过程时间开销有所增加, 但 R^* -树以增加少量的构造时间开销换取了更高的查找性能和空间利用率. TPR * -树^[13] 则通过改善 TPR-树中的插入和删除算法, 进一步减少查询开销.

为了解决 R-树的频繁更新问题, Kwon 等^[10] 提出了自底向上更新方法的思路, 但文中并没有具体实现. Lee 等^[14] 在 LUR-树^[10] 的基础上采用了自底向上的更新方法, 该方法避免了自顶向下方法中潜在的遍历操作, 并利用了连续更新的局部特点, 然而, 为了能够直接访问索引的最底层, 该方法需要辅助的数据结构, 这就增加了空间开销, 且该辅助结构是基于磁盘的, 因此, 每次更新至少需要 3 次 I/O 操作; 另外, 该方法对索引对象的分布和运动情况敏感, 随着对象分布变得不均匀或运动速度的提高, 更新代价显著增加.

HTPR-树^[15] 中采用了一种扩展的自底向上动态更新(Extended Bottom-Up Update, EBUU)算法来提

高更新操作的效率. 针对 TPR-树及 TPR * -树在实际应用中动态更新性能低下且随着时间变化索引查询性能急遽下降的问题, 廖巍等^[16] 又在 HTPR-树的基础上, 综合考虑移动对象在速度域和空间域中的分布, 提出了一种基于速度分布的移动对象混合索引 HVTTPR 树. 首先在速度域中对移动对象集进行规则划分, 根据速度矢量大小将移动对象映射到不同的速度桶, 每个速度桶中移动对象具有相近的速度矢量; 然后对每个速度桶中的移动对象利用 TPR 树进行索引. HVTTPR 树能明显减少更新次数, 同时采用 EBUU 算法以提高其更新效率, 具有很好的动态更新性能和并发性, 但速度桶数目对 HVTTPR 树索引更新性能及查询性能影响显著, 文中并没有给出确定速度桶数量的具体方案.

2.2.2 批量装载和延迟更新

延迟更新(Lazy-Update)方法实质上是批量装载技术(Bulk Loading Techniques)的进一步发展. Choubey 等^[17] 在 R-树基础上提出了批量插入(Bulk-Insertion)策略, 主要思想是一次性在已有索引结构上执行多个插入操作, 从而提高更新效率. 文献[18-19] 则分别在 R-树和 TPR 树上实现了批量装载技术.

对象的更新操作可以分为删除和插入 2 步操作, 对应的, 延迟更新也包括延迟删除(Lazy Deletion)和延迟插入(Lazy Insertion)2 个方面. 延迟删除策略的基本思想是在接收到对象更新信息时, 立即写入磁盘(完成插入操作); 如果相应对象的旧索引项位于不同的磁盘页, 则不会立即删除. 这时往往需要一个内存缓冲区来存储这些暂未删除的索引项信息, 以便于区分新、旧索引项. 这些旧索引项信息直到满足一定条件时(如缓冲区已满)才以某种方式来删除. RUM-树^[20] 和 TPROM-树^[21] 分别在 R-树和 TPR-树上实现了延迟删除策略来提高更新效率.

为了克服现有支持频繁更新的索引树性能大都深受处理器缓存失效的影响, 苏亮等^[22] 提出了一种基于双 Memo 的量化 R^* 索引树 QDM-树, 并给出了相应的插入、删除、更新和范围查询算法.

延迟插入策略的基本思想是更新信息先存放在缓冲区, 然后根据某种机制(如当缓冲区满的时候)将缓冲的更新批量写入磁盘, 同时删除对象的旧索引项. 这样就避免了过多的 I/O 操作, 降低每次更新的平均 I/O 代价. 同样的, 这种方法也需要一种辅助的索引结构(对对象 ID 进行索引)来快速定位旧索

引项.

Biveinis 等^[23]设计的 R^R -树同时采用了延迟删除和延迟插入策略. 该树包括 1 棵常驻主存 R -树和 1 棵传统的基于磁盘的 R -树, 其中内存 R -树(又称为操作缓冲树)用于缓存将要执行的操作(删除和插入). R^R -树有效地发挥了大内存的优势, 但同时对于 2 棵树的维护势必增加算法的复杂性, 不利于大规模数据的并发控制.

通过以上分析可以看出, 延迟更新方法的实质是尽量使多个操作(删除或插入)共用 I/O 操作, 从而提高更新效率. 但由于同一个对象在索引结构中可能存在多个数据项, 其中只有一个是最新的(也就是有效的), 每次查询操作需要加以确认, 从而在一定程度上降低了查询效率.

2.3 有效利用主存资源

实际上, 在上述一些解决频繁更新的方法(如自底向上更新、批量装载、延迟更新等)中已经涉及主存资源的利用. 然而, 现有的利用主存的方法都无法充分利用可获得的主存资源, 有些对主存容量有一个最低要求. 此外, 普通的使用主存的方法(LRU Cache)在提高更新速度方面的效果不是很好^[23].

Biveinis 等^[24]在提出 R^R -树^[23]的基础上, 对主存的高效利用问题进行了探讨, 但没有给出具体的方法.

3 总结与展望

Summary and forecast

本文分析了 R -树在更新中存在的问题, 并对相关的解决方法进行了分类与介绍. 当前大部分解决方法都有其适用范围和局限性, 并不能完全满足更新密集型应用的需求. 索引结构依然是此类系统性能的瓶颈.

下一步的工作将继续研究解决频繁更新问题的方法, 主要有以下几个方面:

1) 探索有效的位置预测方法来尽量提高预测精度, 从而减少更新次数;

2) 深入研究延迟更新方法(特别是延迟插入策略), 充分利用可获得的主存资源, 使对象的更新操作尽量在内存中完成, 提高更新操作的效率;

3) 研究索引结构对工作环境变化的适应能力及其局限性, 针对不同的环境变化采用相应的技术, 使得索引结构能根据移动对象的空间分布、速度分布及变化情况、系统工作负载等的变化而自动调整,

提高索引结构的自适应能力.

参考文献

References

- [1] Guttman A. R-Trees: A dynamic index structure for spatial searching[C]//Proc of the ACM SIGMOD International Conference on Management of Data. Boston: ACM Press, 1984: 47-57
- [2] Lin B, Su J. Handling frequent updates of moving objects[C]//Proc. of 2005 ACM CIKM International Conference on Information and Knowledge Management. Bremen: ACM Press, 2005: 493-500
- [3] Wolfson O, Xu B, Chamberlain S, et al. Moving objects databases: Issues and solutions[C]//Proc of the 10th International Conference on Scientific and Statistical Database Management (SSD-BM). Capri: IEEE Computer Society, 1998: 111-122
- [4] Šaltenis S, Jensen C S, Leutenegger S T, et al. Indexing the positions of continuously moving objects[C]//Proc of the ACM SIGMOD International Conference on Management of Data. Dallas: ACM Press, 2000: 331-342
- [5] 赵卿松, 卢炎生. 移动对象位置预测的索引方法[J]. 计算机科学, 2006, 33(8): 170-172
ZHAO Qingsong, LU Yansheng. An access method for prediction of moving objects' location[J]. Computer Science, 2006, 33(8): 170-172
- [6] Sun J, Paradias D, Tao Y, et al. Querying about the past, the present, and the future in spatio-temporal databases[C]//Proc of the 20th International Conference on Data Engineering (ICDE). Boston: IEEE Computer Society, 2004: 202-213
- [7] Xu B, Wolfson O. Time-series prediction with applications to traffic and moving objects database[C]//Proc of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access. San Diego: ACM Press, 2003: 56-60
- [8] Xiao Y, Zhang H, Wang H. Location prediction for tracking moving objects based on grey theory[C]//Proc of the 4th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Haikou: IEEE Computer Society, 2007: 390-394
- [9] Morzy M. Mining frequent trajectories of moving objects for location prediction[C]//Proc of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition. Leipzig: Springer-Verlag, 2007: 667-680
- [10] Kwon D, Lee S, Lee S. Indexing the current positions of moving objects using the lazy update R-tree[C]//Proc of the 3rd International Conference on Mobile Data Management (MDM). Singapore: IEEE Computer Society, 2002: 113-120
- [11] Cheng R, Xia Y, Prabhakar S, et al. Change tolerant indexing for constantly evolving data[C]//Proc of the 21st International Conference on Data Engineering (ICDE). Tokyo: IEEE Computer Society, 2005: 391-402
- [12] Beckmann N, Kriegel H-P, Schneider R, et al. The R^* -tree: an efficient and robust access method for points and rectangles[C]//Proc of ACM SIGMOD International Conference on Management of Data. Atlantic City: ACM Press, 1990: 322-331
- [13] Tao Y, Papadias D, Sun J. The TPR * -tree: an optimized spatio-temporal access method for predictive queries[C]//Proc of the 29th International Conference on VLDB. Berlin: Morgan Kaufmann, 2003: 790-801
- [14] Lee M L, Hsu W, Jensen C S, et al. Supporting frequent updates in R-trees: a bottom-up approach[C]//Proc of the 29th International Conference on VLDB. Berlin: Morgan Kaufmann, 2003: 608-619
- [15] 廖巍, 熊伟, 景宁, 等. 支持频繁更新的移动对象混合索引方法[J]. 计算机研究与发展, 2006, 43(5): 888-893
LIAO Wei, XIONG Wei, JING Ning, et al. Hybrid indexing of

- moving objects with frequent updates[J]. Journal of Computer Research and Development, 2006, 43(5): 888-893
- [16] 廖巍,唐桂芬,景宁,等. 基于速度分布的移动对象混合索引方法[J]. 计算机学报, 2007, 30(4): 661-671
LIAO Wei, TANG Guifen, JING Ning, et al. Hybrid indexing of moving objects based on velocity distribution[J]. Chinese Journal of Computers, 2007, 30(4): 661-671
- [17] Choubey R, Chen L, Rundensteiner E A. GBI: A generalized R-tree bulk-insertion strategy [C] // Proc of the 6th International Symposium on Large Spatial Databases (SSD). Hong Kong: Springer, 1999: 91-108
- [18] Arge L, Hinrichs K, Vahrenhold J, et al. Efficient bulk operations on dynamic R-trees[J]. Algorithmica, 2002, 33(1): 104-128
- [19] Lin B, Su J. On bulk loading TPR-tree[C] // Proc of the 5th International Conference on Mobile Data Management (MDM). Berkeley: IEEE Computer Society, 2004: 114-124
- [20] Xiong X, Aref W G. R-Trees with update memos[C] // Proc of the 22nd International Conference on Data Engineering (ICDE). Atlanta: IEEE Computer Society, 2006: 22
- [21] Ding X, Lu Y, Ding X, et al. An efficient index for moving objects with frequent updates[C] // Proc of the 2007 International Conference on Wireless Communications, Networking and Mobile Computing. Shanghai: [s. n.], 2007: 5951-5954
- [22] 苏亮,王博,邹鹏,等. QDM-Tree: 支持数据流频繁更新的Cache 敏感索引[J]. 微电子学与计算机, 2008, 25(9): 193-195
SU Liang, WANG Bo, ZOU Peng, et al. QDM-tree: Cache conscious indexing for frequent updates over data streams[J]. Microelectronics & Computer, 2008, 25(9): 193-195
- [23] Biveinis L, Šaltenis S, Jensen C S. Main-memory operation buffering for efficient R-tree update[C] // Proc of the 33rd International Conference on VLDB. Vienna: ACM Press, 2007: 591-602
- [24] Biveinis L, Šaltenis S. Towards efficient main-memory use for optimum tree index update[C] // Proc of the 34th International Conference on VLDB. Auckland: ACM Press, 2008: 1617-1622

Research on frequent updates in R-trees

PAN Jinji^{1,2} LIU Jingfa¹ MA Tinghuai¹

1 School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044

2 School of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016

Abstract In the update-intensive applications, index structures can easily become performance bottlenecks. Frequent update is a key issue in indexing moving objects. Due to the poor update performance, R-trees are not practically applicable to update-intensive applications. Improving the R-trees' update performance is an important yet challenging issue. This work thoroughly analyzes the update problems in R-trees. And most existing solutions are classified and discussed in detail. Based on this, some problems remaining to be solved and future research directions are outlined.

Key words moving objects index; R-tree; frequent update; lazy update