面向层次编制移动对象的混合特征索引方法*

张 巨+, 肖予钦, 景 宁, 陈宏盛

(国防科学技术大学 电子科学与工程学院,湖南 长沙 410073)

Towards the Hybrid Feature Indexing of Hierarchically Organized Mobile Objects

ZHANG Ju+, XIAO Yu-Qin, JING Ning, CHEN Hong-Sheng

(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573480, E-mail: juzhang@linuxaid.com.cn, http://www.nudt.edu.cn

Received 2003-01-13; Accepted 2003-05-27

Zhang J, Xiao YQ, Jing N, Chen HS. Towards the hybrid feature indexing of hierarchically organized mobile objects. *Journal of Software*, 2004,15(3):371~378.
http://www.jos.org.cn/1000-9825/15/371.htm

Abstract: With the rapid advances of wireless communication and positioning techniques, tracking the positions of mobile objects is becoming increasingly feasible and necessary. Traditional spatial index structures are not suitable for indexing mobile objects because of numerous updating operations. A coordinates-organization hybrid-feature indexing structure called C2OR-Tree is firstly introduced in this paper to index the current positions of the hierarchically organized mobile objects. Based on C2OR-Tree, a notable activated-insertion and deferred-deletion (AIDD) algorithm is presented to deal with the bulk updates of the objects coordinates. In view of the local reconstruction characteristic of the C2OR-Tree, AIDD algorithm gives an efficient implemention of the bulk updates with the integration of the insert processing of the updated objects and the earmark processing of the updated regions. Experiment shows that C2OR-Tree can preserve its satisfying query response capability even after many times of AIDD operations.

Key words: R-Tree; mobile object; indexing; bulk updating

摘 要: 随着无线通信和定位技术的发展,移动对象的追踪已经变得越来越可行和必需.传统空间索引结构因无法适应大量的更新操作而不能应用于移动对象的存储与检索.针对具有层次化编制特征的移动对象集,首先给出了一种实现坐标-编制混合特征索引的 C2OR-Tree 方法.在 C2OR-Tree 的基础上,提出了称为"主动插入-延迟删除(AIDD)"技术的移动对象位置更新批处理算法.AIDD 算法充分利用了 C2OR-Tree 在更新批处理时的局部重构特性,通过在新坐标下对象插入过程中结合更新区域标记过程的思想给出了 C2OR-Tree 更新批处理的高效实现.实验结果显示,采用 AIDD 技术的 C2OR-Tree 不仅具有高效的位置更新批处理性能,而且在多次更新后

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA131010 (国家高技术研究发展计划(863)); the National Key Laboratory Foundation of China under Grant No.51447060101KG0101 (国防科技重点实验室基金)

作者简介: 张巨(1974－),男,吉林松原人,博士,主要研究领域为数据库技术,计算机系统安全技术;肖予钦(1975－),女,博士生,主要研究领域为空间数据库技术;景宁(1963－),男,教授,博士生导师,主要研究领域为地理信息系统与数据库技术;陈宏盛(1958－),男,副教授,主要研究领域为人工智能,数据库技术.

仍能保持令人满意的查询性能.

关键词: R-树;移动对象;索引;更新批处理

中图法分类号: TP311 **文献标识码**: A

无论是车辆监控、供应链管理、数字战场、移动电子商务等实际应用还是战场仿真、网络游戏等虚拟环境,都需要解决大量移动对象的高效检索问题.移动对象索引是计算资源受限的移动指挥系统和单兵数字化设备实现战场态势中动态(如对象的生灭)过程和运动(如对象的机动)过程的高效展示、存储和分析的有效途径.

众所周知,作战单元(BU)的编制属性具有分层的隶属关系.通常,BU 间编制属性上的亲缘远近与空间位置关系具有很强的相关性.将 BU 的层次隶属关系融合到空间索引中不仅可以实现以对象编号为条件的高效检索,而且还可以提供编制关系与空间关系组合约束查询的高效实现.基于上述考虑,我们参照数据挖掘领域中的混合(hybrid)特征索引技术[1],采用在二维坐标空间(x,y)基础上添加一个编制维 O 的方法构造出了一个坐标-编制混合特征空间(C^2O 空间).我们称 C^2O 空间上的 R-Tree[2]索引为 C^2OR-Tree.如果对象的更新批次是按照编制隶属关系划分的,那么 C^2OR-Tree 上的更新批处理完全可以通过 C^2OR-Tree 的局部重构实现,主动插入-延迟删除(activated insertion & deferred deletion,简称 AIDD)算法就是基于这样的思想提出的.

1 相关工作

就我们所知,LUR-Tree[3]是针对移动对象当前位置索引的惟一公开成果,其基本思想是只有对象离开其原来所处的叶结点时,才选择"删除-重插回根结点"、"删除-重插回父结点"或"扩充叶结点 MBR"实现该对象的位置更新.LUR-Tree 实现的基本前提是必须能够根据〈OID,New_Position〉元组确定对象原来所处的叶子结点.为此,LUR-Tree 在 R-Tree 基础上又引入了一个从属的 Direct-Link 结构.Direct-Link 是以 OID 为关键字的散列或 B-Tree 结构,它可以实现以标识符为条件的对象查找.LUR-Tree 对高离散、慢运动的移动对象索引具有一定的参考意义,但它无法保证 R-Tree 结构和 Direct-Link 结构的双重聚簇,因而不适用于坐标更新批处理.

J.V. Bercken等人[4]总结了R-Tree批装载(bulk loading,简称BL)技术的研究成果,并给出了两类通用的BL算法.I. Kamel[5],R. Choubey[6]和N. Roussopoulos[7]等人研究了R-Tree的增量插入批处理问题.L. Arge等人[8]提出的 Buffered R-Tree(BR-Tree)算法可以支持装载、插入和删除操作中任意类型的批处理,BR-Tree通过为结点分配buffer存储区的方法将R-Tree的深度优先搜索策略转换为分阶段的深度-宽度轮换优先搜索策略.直观地,可以采用基于BR-Tree的"批删除-批插入"两阶段方法实现索引对象位置更新的批处理.但是,这种两阶段方法存在以下弊端:BR-Tree的 $\lfloor \log_B M/4B \rfloor$ 深度子树全载入过程必然会读入一些无用结点;BR-Tree的延迟更新过程结束后,buffer中很可能还存有大量的数据有待进一步处理,额外的扫尾工作可能对算法的整体性能造成严重的损害;BR-Tree在最坏情况下只能够使用可用内存的1/B;BR-Tree适合批处理那些欲更新的数据量远远超出可用内存容量的情况,如果每个更新批次的数据量完全可以被内存所容纳,则为索引结点关联磁盘缓冲区的方法显然是低效的.

2 面向层次编制移动对象的混合特征索引

2.1 C^2O空间的构造方法

多维特征空间索引实际上就是根据一组预设的规则对这些对象进行逐级的聚类划分.给定一组特征指标,在它们所构成的特征空间上实现聚类划分将涉及以下几个问题.特征的相关性:良定义的特征指标必须是正交的;特征的度量:计算程序中,任何一类判断问题从根本上说都是">=<"问题,而">=<"只能用来处理间隔尺度和部分有序尺度;特征对聚类划分的影响:对于不同的应用目标,每个特征对聚类划分的"贡献"也可能不同,所以同一种分布在不同目标函数下的聚类划分可能会有很大差异.

在数字战场应用中,作战单元的编制属性与空间坐标是"天然"正交的,但对象的编制属性并不是可度量属性,所以首先要考虑的问题就是如何将对象的编制属性合理地映射到一个一维坐标轴 O 上.因为有限集合总是

可列的,所以最直接的映射方法就是按照某个准则将对象按编制属性排序,排序后的对象序号就可以作为该对象在 O 上的映射.为了体现对象在编制属性上的聚类关系,并且保证编制属性到实数的映射不致太复杂,我们采用“分层编码换算、层间加权保聚合”的方法给出了一种编制属性到实数的映射方法(如图 1 所示).编码阶段:按照编制的自然层次 l 赋予对象 x 一个分级编号(类似于部队番号)$H_l(x)$,并记录层次 l 上的最大扇出 MAX_l;变换阶段:利用公式 $f(x)=\sum_l \frac{H_l(x)}{MAX_l}\zeta^l$ 将对象 x 的编制属性换算为实数 $f(x)$,其中 ζ 为一个大于 1 的权值.ζ 越大,对象集在 O 上呈现的聚合特征就越明显.R-Tree 通常以 MBB 空间范围最小化为判定准则,因此编制属性对聚类划分的“贡献”随 ζ 的增长而增长(后面的实验会进一步分析 ζ 的取值问题).最后,需要对 C^2O 进行归一化处理,以便对象的编制属性与空间坐标对聚类划分的“贡献”的全局比例为 1:1.

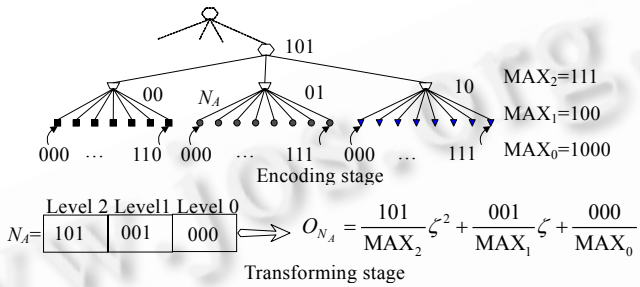

Fig.1 The encoding and transforming stage of organization attributes

图 1 编制属性的编码与变换阶段

2.2 基于buffer的AIDD算法

前面已经提到,在 C^2OR-Tree 上仅通过标识符(OId)就可以快速定位对象在索引树中所处的叶结点,所以在 C^2OR-Tree 上用“删除-重插入”替代位置更新操作并不存在技术上的障碍.如果对象的更新批次是按照编制隶属关系划分的,那么发生位置更新的对象将刚好占据编制轴 O 上的一个坐标区域($[O_{\min},O_{\max}]$).这个特征对于对象位置更新的批处理具有非常重要的意义,因为整个更新批处理过程完全被约束在窗口区域 UR(x:[0,1];y:[0,1];O:$[O_{\min},O_{\max}]$)内.如果对象的空间分布具有按编制聚合的特征,那么 C^2OR-Tree 的更新批处理过程也将呈现非常明显的局部重构特征.在高离散、慢运动的移动对象索引应用中,坐标更新前后的对象很可能处在 C^2OR-Tree 的同一个叶结点区域,这时的“删除-重插入”过程完全可以用单纯的坐标属性更新所替代.即使是无法用单纯的坐标属性更新实现的更新操作,用更有效的算法替代“删除-重插入”过程也是可能的.因为每次更新批处理过程都局限在编制轴的某个区域内,所以我们可以通过对这个更新区域作标记模拟实际的删除过程.但这个更新区域标记将因新坐标下对象的重插入而失效,故单纯的更新区域标记并不能满足更新批处理的需求.如果能够在重插入批处理过程所访问的路径上顺便执行更新区域标记和结点回收,则不仅可以保证更新区域的标记不再受对象重插入过程的影响,而且可以顺便回收那些已删除的叶子结点和子树.这就是主动插入-延迟删除(activated insertion & deferred deletion,简称 AIDD)算法的基本思想.

AIDD 算法的第 1 阶段是以新坐标下对象的插入批处理为驱动的下推进程,下推操作在执行插入批处理的同时,对沿途所访问的结点顺便执行更新区域标记和结点回收.具体地,AIDD 下推过程首先将那些欲插入到某个子结点 N_c 的新坐标下对象集 $S_N(N_c)$ 缓存到与 N_c 相关联的 N_c.buffer 中;在进一步处理 N_c 时,更新区域标记过程仅仅作用于那些 AI 过程不会访问到的子结点引用项(即 N_c 内存储的子结点指针和子树 MBB),从而保证这些标记不会受到 $S_N(N_c)$ 中对象重插入的影响;而那些 AI 过程需要访问的结点则像 N_c 一样做进一步的“递归”处理.即使 $S_N(N_c)$ 中的部分对象处在某个完全被更新区域所覆盖的子树 MBB 范围内,算法也将回收这个结点(及其子树),这是因为调整结点的代价(2×IO)要高于分配结点的代价(1×IO).下面我们将对 N_c.MBB(N_c.MBB 实际存储在 N_c 的父结点 FN_c 中)在编制轴 O 上的投影 $f_O(N_c.\text{MBB})$ 覆盖(contain)或交叠(overlap)更新区段$[O_{\min},O_{\max}]$两种情况的 AIDD 处理过程予以详细说明.

$f_O(N_c.\text{MBB})$交叠$[O_{\min},O_{\max}]$.

第 1 步,首先将那些 MBB 完全落在更新区域 UR 的子结点引用项从 N_c 摘除并放入回收链表 X-list 中;

第 2 步,利用公式 $\text{MBB}_i:=\text{MBB}_i-\text{UR}$ 调整 N_c 中剩余的子结点引用项(MBB_i 表示 N_c 中第 i 个引用项的 MBB),子结点引用项的调整对应于更新区域标记,可见,标记过程不需要调整 C^2OR-Tree 中结点引用项的数据结构;

第 3 步,利用标准 R-Tree 的插入启发式将 N_c.buffer 中待处理的对象逐一插入到新创建的、关联相应子结点的 buffer 链表中.

这里需要补充两点:第 1,由于 N_c.MBB 参数是从 FN_c 中得到的,它本身可能已经被先前的 AIDD 过程标记过,所以在执行第 1 步操作之前首先要利用公式 $\text{MBB}_i:=\text{MBB}_i\cap N_c.\text{MBB}$ 调整 N_c 中的子结点引用项并适时执行结点回收;第 2,引用项的删除会造成 N_c 下溢,但下溢可能是暂时的,因为与 N_c 的子结点相关联的 buffer 中还有一定数量的新坐标下对象有待进一步处理,因此我们暂时将 N_c 放入下溢结点集合 Set_{UF},Set_{UF} 的处理留待后面予以详细说明.图 2 给出了 Nc.MBB 交叠$[O_{\min},O_{\max}]$时 AIDD 处理前后的结点与索引树调整情况.

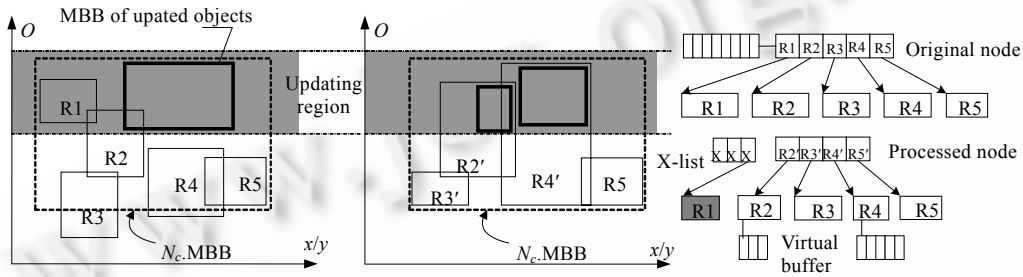

Fig.2 Case 1: N_c.MBB overlaps $[O_{\min},O_{\max}]$

图 2 情况 1:N_c.MBB 交叠 $[O_{\min},O_{\max}]$

$f_O(N_c.\text{MBB})$覆盖$[O_{\min},O_{\max}]$.覆盖与交叠在处理方法上的区别体现在那些 MBB 在编制轴上的投影仍然覆盖$[O_{\min},O_{\max}]$区域的子结点引用项上.对这些引用项,MBB_i-UR 操作将导致 MBB_i 分裂成两个不连通的区域,所以标记方法无法直接应用到这些引用项上.覆盖情况的具体处理方法如下:

第 1 步,首先也是要通过 $\text{MBB}_i:=\text{MBB}_i\cap N_c.\text{MBB}$ 调整 N_c 中的子结点引用项,并且回收那些引用项满足 $\text{MBB}_i\cap N_c.\text{MBB}-\text{UR}=\varnothing$的子树,如果子树回收导致 N_c 下溢,则将 N_c 放入 Set_{UF};

第 2 步,利用公式 $\text{MBB}_i:=\text{MBB}_i-\text{UR}$ 调整那些交叠$[O_{\min},O_{\max}]$区域的引用项;

第 3 步,利用标准 R-Tree 的插入启发式将 N_c.buffer 中待处理的对象逐一插入到新创建的、关联相应子结点的 buffer 链表中;

第 4 步,为第 2、3 步没有访问到的引用项(即 MBB 在编制轴上的投影仍然覆盖$[O_{\min},O_{\max}]$区域的引用项)所对应的子结点关联一个空 buffer,关联空 buffer 的目的是使结点能够被下一层次上的 AIDD 操作所访问并予以进一步处理.图 3 给出了 $f_O(N_c.\text{MBB})$覆盖$[O_{\min},O_{\max}]$时 AIDD 操作前后的结点与索引树调整情况.

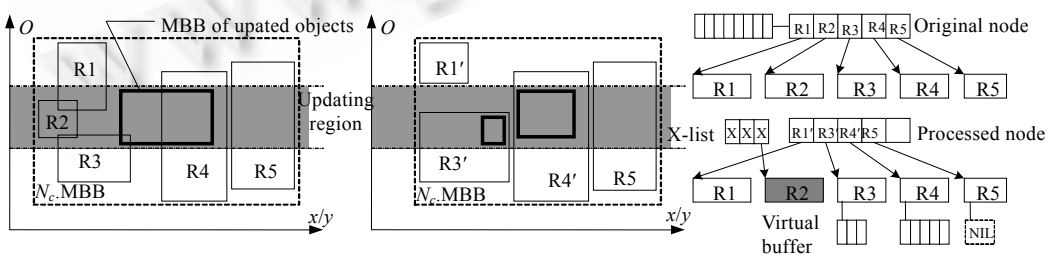

Fig.3 Case 2: N_c.MBB contains $[O_{\min},O_{\max}]$

图3 情况2:N_c.MBB覆盖$[O_{\min},O_{\max}]$

AIDD 的下推过程结束后,新坐标下对象就以 buffer 链表的形式与某些叶结点关联在一起了.对每一个带有 buffer 的叶结点 l,首先删除 l 中那些处在 l.MBB 以外或者 UR 以内的对象;然后将与 l 关联的 buffer 链表中的对

象逐个摘除并装入 l,直到 buffer 清空或 l 溢出为止.如果结点下溢,则将这个结点(的指针)放入 Set_{UF};如果结点上溢,则将这个结点(溢出的对象仍存储在与该结点相关联的 buffer 链表中)放入上溢结点集合 Set_{OF};而满足容量约束的结点则立即写回磁盘.

上溢所对应的结点分裂过程和下溢所对应的索引树收缩过程都可能向上传播甚至导致沿途的结点重组.如果利用传统的 R-Tree 算法对这些结点分别予以处理,那么下溢结点造成的索引树收缩可能又被其邻域内某个结点的上溢调整回来(反之亦然),为避免类似的情况发生,我们仍采用广度优先算法执行索引树的回溯重组.回溯重组过程分为下溢结点处理与上溢结点处理两个阶段(如图 4 所示).

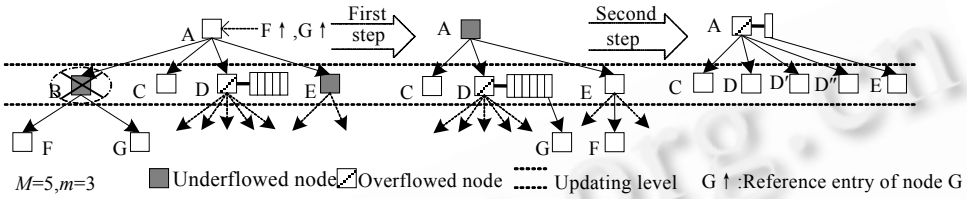

Fig.4 Retrospect stage of AIDD operations

图4 AIDD操作的回溯过程

第 1 阶段:对 Set_{UF} 中当前执行层次上的每一个结点 N_i,回溯算法首先将 N_i 从索引树上摘除,即删除 N_i 在其父结点 FN_i 中对应的引用项;然后按照标准 R-Tree 的插入启发式将 N_i 中的引用项以 FN_i 为入口逐一插回到 N_i 的兄弟结点中;最后将 N_i 的所有数据位都清“0”后放入 X-list,清“0”的目的是保证后面将讨论的空闲块回收算法能够区分 X-list 中的单结点删除项与下推操作阶段产生的子树删除项.这里需要强调两点:(1) 引用项重插入过程也必须像下推操作阶段一样适时根据 FN_i 中相应引用项的 MBB 调整重插入过程所访问的结点(即 N_i 的兄弟结点),因为重插入的数据项是 N_i 的子结点,所以 MBB 调整过程到达 N_i 的兄弟结点也就终止了;(2) 重插入过程可能导致 N_i 的兄弟结点下溢/上溢或原本处在下溢集合的结点重新填充到正常或上溢状态,产生下溢的结点将利用这里给出的方法作进一步的处理,重填充到正常状态的结点将从 Set_{UF} 删除并写回磁盘,而那些发生了上溢的结点则放入 Set_{OF} 等待进一步处理.

第 2 阶段:对 Set_{UF} 中当前执行层次上的每一个结点 N_i,回溯算法首先将 N_i 从索引树上摘除;然后以 N_i 中的子结点引用项和与 N_i 相关联的 buffer 中的数据项为索引对象执行 Quickload 算法[4]的叶结点装载过程(只需要去掉 Quickload 算法的最外层循环就可以了);最后,将 Quickload 装载过程产生的叶结点逐一插回到 N_i 的父结点 FN_i,如果插回过程造成 FN_i 上溢,则创建一个关联 FN_i 的 buffer 以存储剩余的叶结点.

如果当前层次中所有的下溢和上溢结点都处理完毕,那么算法将回溯到更高的一个层次继续执行,上述回溯过程一直继续下去,直到 Set_{OF}/Set_{UF} 中不再有待处理的结点为止.至此,AIDD 算法的执行过程就结束了.

AIDD 过程导致了索引树中部分结点下子树的 MBB 与其在父结点中对应的引用项的 MBB 不一致,这种不一致来源于两个方面:首先,那些被更新区域标记过程标记却又没有被 AI 过程访问的结点引用项将导致编制维度上的不一致;其次,导致子树整体删除的更新区域标记过程并没有执行父结点 MBB 的递归重调整,所以有可能产生坐标维度上的不一致.查询过程必须经过适当的修改,以适应上述不一致情况.在窗口查询算法中,只要在查找算法的递归寻路过程中根据当前结点中记录的子结点 MBB 及时调整查询窗口(即求查询窗口与子结点 MBB 的交)就可以适应第 1 种不一致情况.对于第 2 种不一致情况,查询算法并不需要任何调整,但这种不一致将导致查询路径“虚警”,所以会在一定程度上影响查询性能.我们将在后面的实验中进一步探讨上述不一致情况对查询响应能力的影响.对于最近邻(NN)查询,一般并不关心编制轴上的临近特征,而是以空间上的近邻关系为目标,所以在以分枝界定(Branch&Bound)法实现空间最近邻查询时,我们仅以结点在空间坐标平面上的投影作为计算代价函数、下界函数和上界函数的依据,由于 C^2O 空间中 MBB 在 x-y 平面上的投影会产生比较严重的交叠,所以在 C^2OR-Tree 上实现坐标空间上的最近邻查询的效率不会很高.

2.3 AIDD实现中的其他技术问题

前面已经提到,为索引结点关联 buffer 是宽度优先搜索的基础.因为无法预知更新批处理过程中哪些结点需要 buffer 以及所需 buffer 的具体大小,所以我们在算法中采用动态关联的链表结构模拟了 buffer 功能.首先,对当前更新批次内的对象按照新坐标进行 Hilbert 排序,然后将排序结果保存到一个临时创建的文件 tmpFILE 中,Hilbert排序的目的是使得每次buffer清空过程所处理的对象在存储位置上尽量邻近,从而保证即使tmpFILE 的部分或全部被交换到外存,算法也能以最小的I/O代价将这些数据重新载入内存.结点与buffer的关联采用单链表实现,为了使 buffer 清空过程更接近深度优先算法的执行顺序,我们利用在链表头部同时记录链尾的方法保证了单链表上快速 FIFO 的实现.链表的数据项主体不是⟨OID,New_Position⟩元组本身,而是指向临时文件中对应元组的指针,从而使得需要经常访问和更新的链表结构消耗尽量少的内存空间.在前面介绍 Buffered R-Tree算法时我们已经指出,采用buffer方法实现的广度优先搜索无法给出*parent*(*v*)函数的合理实现.为了解决这个问题,我们在内存中保留了 AIDD 执行过程所访问的结点及其父结点的指针对⟨*v*↑,*parent*(*v*)↑⟩,从而形成了如图 4 的左半部分所示的一个反向树结构.因为回溯过程也需要对索引树予以分层处理,所以我们在反向树结构上附加了一个层次内结点链表以便快速查找这些指针对.综合上述考虑,图 5 中间部分的结点缓存数据项实际上应该是形如⟨*v*↑,*parent*(*v*)↑,bufferhead↑,buffertail↑,nextnode↑⟩的数据结构.以缓冲结构 D 为例,它实际上是5元组⟨v_D↑,v_E↑,b↑,g↑,B↑⟩.注意,图5所示的是AIDD下推阶段的内存数据结构.在回溯阶段,除了没有tmpFILE文件外,bufferhead↑与buffertail↑指针将被借用到上溢结点的溢出项管理上.由此可见,我们倾向于假设单次AIDD批处理过程不至于耗尽所有可用内存.

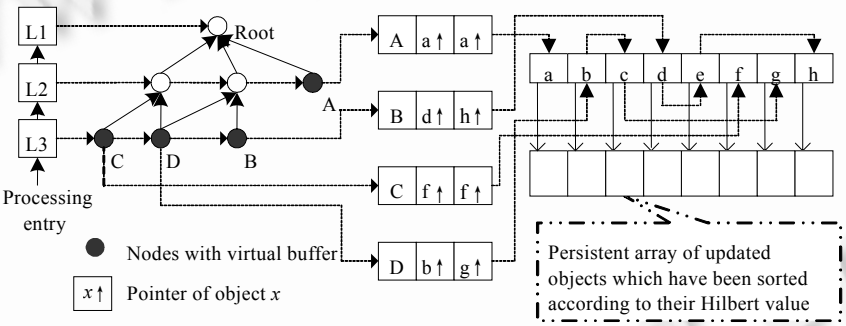

Fig.5 Memory management for AIDD processing

图 5 AIDD 过程中的内存管理

上溢结点的分裂需要申请存储空间,下溢结点的收缩和子树删除则释放部分存储空间.在实际应用环境中,周期性的 AIDD 伴随着大量的存储空间申请与释放过程,随着时间的推移,子树的收缩与删除将导致索引文件中有效数据的分布越来越零散,并且通过 append()系统调用为结点分裂补充存储空间的方法将导致索引文件的无节制增长.解决问题的惟一途径就是设计一个实现文件内空闲块回收与重利用的专用算法.我们选择FSIP[9](free space inventory page)作为索引文件内空闲块管理算法的基础,FSIP 的思想与文件系统超级块中的盘块占用位图非常相似,它被广泛应用于 Rdb,DB2,Starburst,EXODUS/Shore 等 DBMS 中.因为 FSIP 需要的存储空间非常小且经常被访问,所以 FSIP 页面通常是常驻内存的.在我们的程序中,索引文件内的空闲块回收工作由一个专门处理 X-list 的独立线程完成,X-list 中的数据项有 3 种类型:叶结点、回溯过程删除的目录结点和下推过程删除的子树.对于前两种类型,只需要将其在FSIP中的对应位清"0"并从X-list中删除即可;后一种类型结点的回收要复杂一些,因为程序不仅要释放这个结点,而且还要将该结点下的所有子孙结点(包括目录结点和叶结点)都释放掉.空闲块回收的对偶问题是闲置块的重用,A.D. Rijk[10]研究了最小偏移量优先分配、层序优先分配和叶结点聚簇分配等3种用于R*-Tree索引文件的空间分配策略,其结论是叶聚簇法有利于较大的窗口查询,而层序法则更利于宽度优先搜索和小窗口查询.在我们的应用中,宽度优先的 AIDD 算法是影响 C^2OR-Tree 性能的主要因素,所以我们选择层序优先的空闲块重分配策略.由于实现方法比较简单,这里不再赘述.

3 仿真测试与结果分析

我们的实验环境为:Mandrake Linux 8.1 操作系统、Ext2 文件系统(1kB 盘块)、10 000 转 SCSI-3 硬盘、80M SCSI-PCI 适配卡、Celeron733 处理器、133M PCI-X 总线.算法采用在 C++中嵌入 RDTSC(Pentium 系列 CPU 专用)汇编指令获得纳秒级精度的墙上时钟(wall time).以实际应用为背景,我们将编制属性分为 8 个层次,且规定结点具有 4~8 之间均匀分布的整数随机扇出.为了简化实现算法的复杂度,我们定义假设叶结点具有与中间结点相同的扇出,即用两个相同的点坐标描述一个对象的位置.为了产生比较贴近实际的实验数据,我们首先随机生成移动对象集的初始位置集合 S_{start} 和目标位置集合 S_{end},具体的生成方法是从编制树的根结点开始,按独立同边际二维正态分布逐级产生结点下子树的聚类中心,为了保证对象分布能够按编制聚合,生成过程中的标准差σ按一定的比例逐层递减.如果实验过程需要ω个更新周期,则对象的速度期望为其起点到终点向量的 $1/\omega$.我们再为这个速度期望附着一个期望为 $\vec{0}$ 的随机扰动,就可以得到一个比较贴近实际的运动场景.

在静态情况下,我们的测试结果显示:(1) 编制属性变换阶段的ζ在 1.5~2.0 之间取值所生成的 C^{2}OR-Tree 总可以达到比较好的聚集效果;(2) C^{2}OR-Tree 对空间窗口查询(即只有 x-y 坐标约束的窗口查询)的响应时间稳定在相同数据集上建立的 R-Tree 响应时间的 2~3 倍之间,这应该是由于 C^{2}OR-Tree 索引结点的 MBB 在空间坐标平面上的投影会产生一定程度的交叠和 C^{2}OR-Tree 索引中结点的扇出较 R-Tree 小等原因造成的;(3) 通过简单的分析可以得出,C^{2}OR-Tree 在静态情况下的窗口查询复杂度为 $O(Q\log_B N)$,即仍能保持较好的索引质量.下面我们着重给出动态情况下的比较实验结果和分析结论.

图 6 是在一组包含 854 370 个移动对象的空间数据集上执行 C^{2}OR-Tree 与 LUR-Tree 批量更新的响应能力比较实验结果.其中,ζ=1.5,σ的递减比例为 1.2,x%表示每个更新批次的更新比例.图 6 所示的 AIDD 算法具有很强的批处理能力.图 7 是在相同的数据集上执行多次 AIDD 操作后,C^{2}OR-Tree 窗口查询响应能力的测试结果.其中,结果曲线上的标注表示正方形空间查询窗口的大小.虽然 AIDD 操作对 C^{2}OR-Tree 的索引能力有一定的影响,但这种影响并不随着 AIDD 操作次数的增长而增长,而是趋向于一个相对稳定的状态,并且这个稳态下的响应能力也是可以接受的.这里,我们仅对空间窗口查询的响应能力进行了比较测试.C^{2}OR-Tree 在编制-坐标混合特征查询处理能力上的优势是比较明显的.

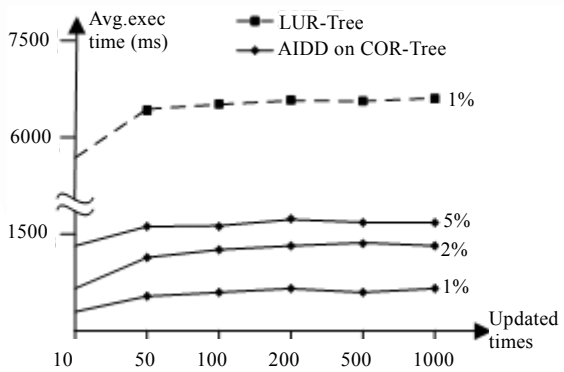

Fig.6 Bulk updating on C^{2}OR-Tree and LUR-Tree

图 6 C^{2}OR-Tree 与 LUR-Tree 的批处理

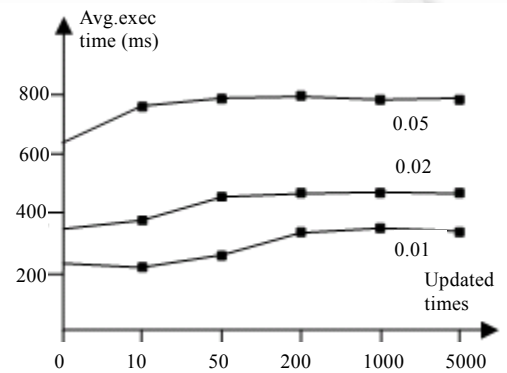

Fig.7 Affect of AIDDs on C^{2}OR-Tree

图 7 AIDD 对 C^{2}OR-Tree 的影响

References:

[1] Chakrabarti K, Mehrotra S. The hybrid tree: An index structure for high dimensional feature spaces. In: Proc. of the 15th Int'l Conf. on Data Engineering (ICDE). Sydney: IEEE Computer Society Press, 1999. 440~447.

[2] Guttman A. R-Trees: A dynamic index structure for spatial searching. SIGMOD Record, 1984,14(2):47~57.

[3] Kwon D, Lee S, Lee S. Indexing the current positions of moving objects using the lazy update r-tree. In: Proc. of the 3rd Int'l Conf. on Mobile Data Management (MDM). Singapore: IEEE Computer Society Press, 2002. 113~120.

[4] Bercken JV, Seeger S. An evaluation of generic bulk loading techniques. In: Apers PMG, *et al*. eds. In: Proc. of the 27th Int'l Conf. on Very Large Data Bases. Roma: IEEE Computer Society Press, 2001. 461~470.

[5] Kamel I, Khalil M, Kouramajian V. Bulk insertion in dynamic R-trees. In: Kraak M, *et al*. eds. Proc. of the 4th Int'l Symp. on Spatial Data Handling (SDH). Delft: Taylor and Francis, 1996,3B:31~42.

[6] Choubey R, Chen L, Rundensteiner EA. GBI: A generalized R-tree bulk-insertion strategy. In: Güting RH, *et al*. eds. Proc. of the 6th Int'l Symp. on Advances in Spatial Databases(SSD). Hong Kong: Springer-Verlag, 1999. 91~108.

[7] Roussopoulos N, Roussopoulos KY. Cubetree: Organization of and bulk incremental updates on the data cube. SIGMOD Record, 1996,25(2):259~270.

[8] Arge L, Hinrichsy KH, Vahrenhold J, Vitter JS. Efficient bulk operations on dynamic R-trees. Special Issue on Experimental Algorithmics in Algorithmica, 2002,33(1):104~128.

[9] McAuliffe ML, Carey MJ, Solomon MH. Towards effective and efficient free space management. SIGMOD Record, 1996,25(2):389~400.

[10] Rijk AD. Improving the R*-Tree storage allocation algorithm [MS. Thesis]. Netherlands: Delft Univ. of Tech., 2000.

第 14 届中国计算机学会网络与数据通信学术会议

征 文 通 知

中国计算机学会网络与数据通信专委会和开放系统专委会定于 2004 年 10 月在西安召开第 14 届中国计算机学会网络与数据通信学术会议,本届会议的主题是研究信息网络关键技术,营造安全可靠网络环境.

一、征文范围

开放系统及其互联技术;新一代网络结构与协议;网络智能化、网络管理;网络信息系统模型;网络计算与应用;网络环境下的信息安全;无线通信网络;电子商务系统以及光纤通信技术.

二、要求

1、欢迎围绕以上主题提交研究论文,字数一般不要超过 6000 字.录用的论文将在西北大学学报(增刊)(国内核心期刊)上发表,评选的优秀论文将推荐在国家权威期刊上发表.

2、稿件格式要求请查询网址:http://www.nwu.edu.cn/xsnh/index.htm

3、应征文章请寄两份打印稿,同时须用电子邮件的附件发来(或用软盘寄来)电子稿,电子稿件请用 WORD 文件格式(.doc 文件).

4、请随同稿件一起,用另纸写明文章题目,所属主题,作者姓名(最多 4 人),职务/职称,所属单位,详细通信地址,邮编,电话,E-mail 地址.

5、已经发表的论文请勿报送.如因一稿多投带来任何问题,责任由投稿者自负.

6、论文收寄地址:710069 西安市太白北路 229 号西北大学现代教育技术中心学术年会筹备组 宋峰 收

发送电子稿的邮件地址:cetc@nwu.edu.cn.

三、论文截止日期

2004 年 6 月 30 日(以寄出邮戳日期为准),会议组委会将于 2004 年 7 月 31 日前发出论文录用通知和参加会议邀请信.

四、联系方式

通信地址:710069 西安市太白北路 229 号 西北大学现代教育技术中心学术年会筹备组

电话:029-88302758 传真:029-88303857 电子邮件: cetc@nwu.edu.cn

联系人:宋峰 刘瑞献