# 一种加快 WebGIS 服务器响应速度的空间索引*

叶常春+, 罗金平, 周兴铭

(国防科学技术大学 计算机学院,湖南 长沙 410073)

# A Spatial Index to Improve the Response Speed of WebGIS Servers

YE Chang-Chun+, LUO Jin-Ping, ZHOU Xing-Ming

(Institute of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573664, E-mail: iamdouble@163.com, http://www.nudt.edu.cn

Received 2003-12-11; Accepted 2004-09-07

**Ye CC, Luo JP, Zhou XM. A spatial index to improve the response speed of WebGIS servers. *Journal of Software*, 2005,16(5):819−826.** DOI: 10.1360/jos160819

**Abstract**: WebGIS servers send digital maps to users. For each request-response round, the servers access map data in batches. The access has a feature called multiscale, that is, the map scales selected by users determine the map detail levels. The access method based on R-tree is not adaptive to the multiscale and batch feature. It has two problems: (1) the data records of features of the same level are not clustered in disks; (2) the granularity of data I/O is too small. So accessing map data for display is unefficient. This paper presents a novel spatial index called Multilevel R-tree, which can solve the two problems. The statistics from experiments show that for range queries, the access method based on multilevel R-tree is much more efficient than the one based on R-tree and can improve the response speed of WebGIS Servers.

**Key words**: WebGIS service; spatial database; access method; spatial index; multilevel R-tree

摘 要: WebGIS 服务器向用户提供电子地图浏览服务.每一个请求/响应回合,服务器端都进行着具有多尺度特性的成批式数据访问.多尺度特性是指地图比例尺决定着地图显示内容的详略.基于 R-tree 的数据访问方法与多尺度性和成批性不相适应,存在“同级要素弱簇聚”和“I/O 粒度偏小”两大问题,绘图数据访问效率不高.提出的多级 R-tree 能够解决上述两个问题.来自实验的统计数据表明,对于区域查询,基于多级 R-tree 的访问方法的效率明显高于基于 R-tree 索引的访问方法.使用多级 R-tree 能够有效地提高 WebGIS 服务器的响应速度.

关键词: WebGIS 服务;空间数据库;访问方法;空间索引;多级 R-tree

中图法分类号: TP391 文献标识码: A

现今,基于 WebGIS,越来越多的 Internet 用户通过 Web 浏览器阅读电子地图.WebGIS 服务器管理着大型地图数据库,响应用户请求,向浏览器传送背景地图和文字或符号标注.其中,背景地图的数据量占大比重.由于每一个请求/响应回合都有可能传送新的背景地图,所以背景地图发布效率是 WebGIS 应用的关键指标.本文提出

* 作者简介: 叶常春(1974−),男,浙江东阳人,博士,讲师,主要研究领域为空间数据库技术;罗金平(1970−),男,博士,副研究员,主要研究领域为计算机体系结构,GIS 应用;周兴铭(1938−),男,教授,博士生导师,中国科学院院士,主要研究领域为计算机体系结构,并行与分布数据库,移动计算.

一种能够有效提高 WebGIS 服务器发布地图速度的方法.

每次地图发布,服务器执行区域查询 $Q(w,s)$,数据访问过程分两步:第 1 步是索引检索,得到出现于比例尺 $s$ 的地图上并与查询窗口 $w$ 相交的一批要素(即空间对象).第 2 步是数据 I/O,把这些要素的数据记录从处在二级存储系统内的地图数据库取出并发布给浏览器.很明显,区域查询 $Q(w,s)$的效率决定着 WebGIS 服务器的响应速度.

使用空间索引是提高空间查询效率的必要手段.各种各样的空间索引已被用来组织地图数据库.它们基于以下假设:由于拥有有限的可用内存,软件系统不能够把空间索引完整地装载入内存.也就是说,只有索引树的一部分能够始终驻留内存.由于索引检索涉及磁盘 I/O,使得检索过程的时间开销在数据访问的总时间开销中占较大比重,所以以往提出的空间索引技术的核心目标是减少索引检索的时间开销.但技术的发展使得内存价格低廉.服务器乃至 PC 机能够配备大容量物理内存.因此,空间索引在应用系统运行期间完整地驻留内存是可行的.我们的实验数据(见第 4 节)表明,R-tree 索引树常驻内存时,查询检索时间开销不到数据 I/O 时间开销的二十分之一.我们估计,对于其他空间索引种类,查询检索时间开销所占比例同样很小.所以,提高数据访问效率的关键不再是减少索引检索时间开销,而是减少另一环节即数据 I/O 的时间开销.相应地,有必要摒弃为减少索引检索时间而采用的举措,进而采用能够减少数据 I/O 时间的举措.

本文的目标是减少区域查询 $Q(w,s)$的数据 I/O 时间开销.易知数据访问过程具有两个特性:一是成批性,指的是查询涉及大量的要素,结果集的数据量大;二是区域性,指的是所涉及的要素的全部或部分落在查询窗口 $w$ 内.另外,地图浏览应用具有多尺度性,即用户选用的地图比例尺 $s$ 决定着显示内容的详略.比例尺越大,包含的细节越多;比例尺变小,内容也变得简略.多尺度性意味着重要性等级相同的要素经常被一起使用.

在以往提出的索引中,大部分仅仅考虑了区域性,数据访问过程存在 I/O 粒度偏小和经常一起使用的同级要素的数据记录簇聚程度低两大不足,具体的原因分析见下一节.也有部分索引结合考虑了多尺度性,但依旧没有解决 I/O 粒度偏小的问题.本文提出的改进方法全面考虑了前述 3 个特性,能明显减少数据 I/O 的时间开销.

本文第 1 节首先分析指出基于普通空间索引的数据访问方法存在“I/O 粒度偏小”和“同级要素弱簇聚”两大问题,导致成批式数据访问效率低;然后考察结合了多尺度性的几种空间索引技术.第 2 节提出多级 R-tree 索引,它能够同时解决上述两大问题.第 3 节描述动态多级 R-tree 索引的插入、删除和检索算法.第 4 节给出性能评测结果,比较了分别基于 Hilbert R-tree 索引[1]、Reactive R-tree 索引和多级 R-tree 索引的数据访问方法执行成批式数据访问的性能.最后是总结.

## 1 问题描述和相关工作

### 1.1 问题描述

至今,几乎没有空间索引考虑到成批性,表现在区域查询 $Q(w,s)$的数据 I/O 期间对查询命中的要素数据记录逐一地实施读取操作而不是批量地读取.数据 I/O 粒度是全体要素数据记录的长度的平均值.这一 I/O 粒度偏小.例如,在上海市某区地图内,数据长度为数十字节的要素数据记录的数目是全部记录总数的 42.75%,而这些数据记录的数据总量只占全部数据量的 14.46%.读大批量数据时,粒度过小而 I/O 次数过多,限制了数据访问效率.

空间索引若没有考虑多尺度性,区域查询 $Q(w,s)$执行过程中将遇到“同级要素弱簇聚”这一问题.原因在于空间索引只保证空间上邻近的要素的数据记录之间彼此簇聚.而地图浏览应用具有多尺度特性,重要性等级相同的要素经常一起使用.它们的数据记录簇聚程度低,这无疑会减少数据 I/O 效率.下面考察几种结合考虑了多尺度性的空间索引.

### 1.2 相关工作

空间索引技术多种多样,其中 R-tree 是应用最为广泛的一种.R-tree 是拥有唯一根结点的索引树.每一个要素由它的最小包围矩形(minimal boundary rectangle,简称 MBR)代表.最小包围矩形是包围该要素且各边与坐标轴平行的面积最小的矩形.叶结点所包含的记录的结构是$(ID,Addr,MBR)$,$ID$ 是要素在数据库内的唯一编号;$Addr$ 是要素数据记录的存储地址;$MBR$ 是要素的最小包围矩形.叶结点的 $MBR$ 是包围全体要素的 $MBR$ 的最

小包围矩形.索引树的内部结点所包含的记录的结构是(*MBR*,*ptr*),*ptr* 是指向子结点的指针;*MBR* 是包围结点内全体子结点的 *MBR* 的最小包围矩形.R-tree 索引及各种变体[1−4]都通过专门定义的算法为地图数据集内的全体要素定义了一维序列.序列内相互靠近的要素在空间上彼此邻近.相应地,数据集内遵循此顺序放置要素数据记录.这样的顺序保证了空间邻近的要素簇聚存储,却没有保证同等级要素簇聚存储,不利于具有多尺度特性的地图浏览应用.

于是,人们提出了改进方法.一种简单的方法是按要素等级的不同把数据集也分成多个独立的子集合,分别为各子集合创建独立的索引树.假如数据集内的要素分成 $m$ 个等级,就有 $m$ 棵索引树.这显然是不受欢迎的.对此可作一点改进:增设一个连接 $m$ 个条目的链表把独立的索引树串联在一起.每个条目的结构是(next,*d*,root),next 指向链表的下一条目,root 指向等级为 *d* 的索引树的根结点.

Reactive R-tree[5,6]是另一种改进方法.做法是按要素等级的不同把其索引记录分布于索引树的不同层次.最底层放置等级最低(即 $m$ 级)的要素的索引记录;倒数第 2 层放置等级为 $m-1$ 的要素的索引记录,依次往上类推直至 1 级.Reactive R-tree 允许内部结点包含要素索引记录.$m-1$ 级至 1 级的要素索引记录分别挂接于底部各层的内部结点(最底层为 $m$ 级要素的索引记录组成的叶结点).在每一个结点层,全体矩形按照空间邻近关系排序.数据集内,要素数据记录遵照要素索引记录在索引树内的宽度优先顺序排列.这样能够保证同等级要素簇聚存储.Reactive R-tree 有两方面不足:一是索引树可能极度不平衡;二是若要素等级过多,索引树会很高.

文献[7]介绍了一种结合比例尺的索引技术.这项技术把比例尺处理成单独的一维,和二维(或三维)空间构成多维空间,而后构建多维空间索引.文献[8]提出的 3D R-tree 正是这项技术的具体应用.二维平面和比例尺构成三维空间.每一个要素用一个立方体代表.每个要素 *f* 对应一个比例尺范围[*ls*,*hs*],当地图比例尺 *s* 处在该范围内时,要素将出现于地图之中.比例尺范围[*ls*,*hs*]便是要素 *f* 在比例尺维度上的外延.这项技术要求比例尺分布范围不能过于狭窄,而且某些情形下查询效率不佳.

## 2 多级 R-tree

GIS 应用通常依据重要性、尺寸或其他属性对要素实施分级.下面假设要素集合一共被分成 $m$ 个等级.等级为 $k$ ($k$=1,2,…,$m$)的要素简称为第 $k$ 级要素;第 $k$ 级要素的索引记录简称为第 $k$ 级要素索引记录.

### 2.1 多级R-tree的结构

多级 R-tree 是一棵拥有唯一根结点的树.假设它的层次总数为 $n$,从根结点层往下,各层依次编号为 1,2,…,$n$.图 1 示例了 $m$=3,$n$=4 的多级 R-tree 树.图中,$N_i$($i$=1,2,…,7)为结点记号.$r$ 代表索引记录.数字为索引记录编号,其中被圆圈包围的数字是要素索引记录的编号,其余是结点索引记录的编号.所有结点中,$N_1$,$N_2$,$N_4$ 为内部结点(即只包含指向下层子结点的索引记录的结点),其余为叶结点(即只包含要素索引记录的结点).

多级 R-tree 把 $m$ 个等级的要素索引记录放置在索引树最下面的 $m$ 个层次中(这里,必有 $m<n$),每个层次对应一级.第 $n$ 层放置所有第 $m$ 级的要素索引记录,这些要素索引记录分组构成叶结点.第 $n-i$($i$=1,2,…,$m-1$)层则包含内部结点和叶结点.内部结点内的索引记录指向下层子结点(可能是内部结点也可能是叶结点),称为结点索引记录.叶结点由第 $m-i$ 级要素索引记录分组构成,称为第 $m-i$ 级叶结点.从第 $n-m$ 层往上直至第 1 层,各层都只包含内部结点.

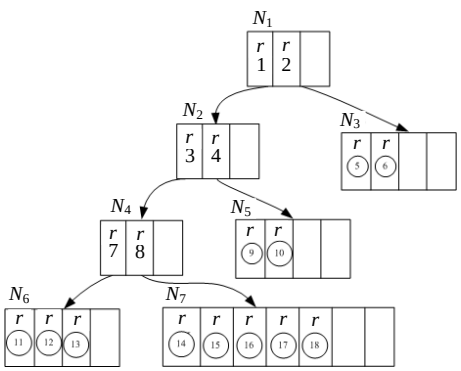

Fig.1 Multilevel R-tree
图 1 多级 R-tree

内部结点扇出系数值为一常数值(记为 $M$).叶结点的扇出系数是动态可变的.这是多级 R-tree 与众不同的地方,具体描述如下.

每一个结点层内,全体要素索引记录(若有的话)排序并分组组成叶结点;全体结点索引记录(若有的话)排序并分组组成内部结点.要素索引记录依据要素的空间邻近关系排序,如何分组见

下节描述.结点索引记录依据结点的 *MBR* 的空间邻近关系排序,分布于$\lceil C/M \rceil$个内部结点内.这里,$C$ 为本结点层内结点索引记录数目.结点索引记录的排列顺序决定了下层子结点从左到右的排列顺序.

在地图数据库内,要素数据记录的排列顺序是严格的.这一顺序与多级 R-tree 内全体要素索引记录的宽度优先顺序一致.这一顺序保证了同一个叶结点内部的索引记录各自指向的要素数据记录连续存储,将这些要素数据记录组成的集合称为要素簇.

与普通 R-tree 相比,多级 R-tree 的不同在于把高等级要素提到索引树的高层.至于要素索引记录的结构和结点索引记录的结构都无须做任何变化.

多级 R-tree 与 Reactive R-tree 有较多相似之处.两者主要不同在于:(1) 前者没有把要素索引记录直接插入内部节点,而是独立组成叶结点.(2) 前者动态设定叶结点的扇出系数.多级 R-tree 不是平衡树,而且层次可能会很多,不过由于索引树驻留内存,不影响使用性能.

### 2.2 叶结点扇出系数

多级 R-tree 的叶结点的扇出系数是动态设定的.这么做的目的是尽可能地使叶结点所指向的要素簇的数据总量接近磁盘块的大小.对于区域查询 $Q(w,s)$,要素簇将被用作数据 I/O 单元.于是,数据 I/O 粒度被优化,时间开销减少.

尽可能地,要素簇的数据总量满足两个条件(称为簇容量限定条件):

(1) 叶结点所指向的要素簇的数据总量不小于设定的常量 *n*ClusterMinSize;

(2) 叶结点所指向的要素簇的数据总量不大于设定的常量 *n*ClusterMaxSize.

条件(1)须优先得到满足.插入或删除算法根据限定条件确定叶结点索引条目数目.在每一次重组叶结点时,算法为要素簇设定最接近于磁盘 I/O 最佳粒度 *n*ClusterBestSize,这里,有 *n*ClusterMinSize<*n*ClusterBestSize<*n*ClusterMaxSize.

多级 R-tree 在插入或删除要素索引记录时动态设定叶结点扇出系数.叶结点所指向的要素簇的数据总量大于 *n*ClusterMaxSize 时发生上溢;小于 *n*ClusterMinSize 时发生下溢.一旦上溢或下溢,叶结点所在层次的全体叶结点实施重组.

动态设定叶结点扇出系数的做法与传统做法不同.基于“索引树不能常驻内存”这一假设,传统做法是把叶结点扇出系数设定为一常数值,该值通常使得叶结点(而不是它所指向的要素簇)的数据总量与磁盘块大小相等,旨在通过优化索引 I/O 粒度以高效存取索引文件.现在情形发生变化,索引树常驻内存既可行也值得.对驻留内存的索引树,动态设定叶结点扇出系数不会降低索引检索性能,却能够优化数据 I/O 粒度,减少 I/O 时间开销.

本文与文献[9]的主要差别在于,前者提出动态设定叶结点扇出系数,而后者则采用静态设定叶结点扇出系数的做法.动态设定利于充分优化数据 I/O 粒度.

## 3 多级 R-tree 操作算法描述

多级 R-tree 是 R-tree 索引的变种.由于空间邻近关系定义算法不同,R-tree 有不同变种.所有 R-tree 变种都能扩充为多级 R-tree.这一节的算法描述针对多级 Hilbert R-tree 索引(multi-level Hilbert R-tree,简称 MLHR).MLHR 是 Hilbert R-tree[1,3]的扩充,后者比其他 R-tree 变种具有更高的空间利用率和空间查询效率.下面采用与 C 语言类似的风格描述动态 MLHR 的插入、删除和查询算法.

算法描述使用了多个术语.

- 结点的等级.对于叶结点,全体索引记录的等级相同,该等级视为叶结点的等级.对于内部结点,各索引记录的等级设定为所指向的结点的等级;全体索引记录等级的最高者作为该内部结点的等级.
- Hilbert 值.每一个坐标点被赋予一个 Hilbert 值[1],该值是由 2 维或 3 维空间到 1 维空间的映射函数计算得到的一个整数值.要素的 *MBR* 的中心点的 Hilbert 值视为要素的 Hilbert 值.记内部结点或叶结点的 *MBR* 之中心点的 Hilbert 值视为该结点的 Hilbert 值.
- 结点的 LHV(largest Hilbert value)值.对于叶结点,其内部全体要素索引记录分别指向要素,这些要素的

Hilbert 值中最大者视为叶结点的 LHV 值[1].对于内部结点,其子结点的 LHV 数值中最大者视为其 LHV 值.

### 3.1 插　入

记待插入的索引记录为 NewIndRec,等级为 *n*Level.索引记录将插入等级为 *n*Level 的叶结点所在的层次.记位于该层次的叶结点集合为 InsertedLayerLeafSet(可能为空,此时派生新的层次);位于该层次的内部结点集合为 InsertedLayerIntlNodeSet;等级高于 *n*Level 的叶结点集合为 HigherLeafSet;待插入层之上的内部结点集合为 HigherIntlNodeSet.

#### 3.1.1 插入算法

Insert(NewIndRec,*n*Level)

输入:NewIndRec 为待插入的索引记录;*n*Level 为索引记录的等级.

算法描述:

1. [定位待插入层]　寻找叶结点等级等于 *n*Level 的结点层次.若没有找到,转步骤(4);
2. [选择待插入的叶结点]　InsertedLayerLeafSet 内的全体叶结点按照各自的 LHV 值从小到大排列,待插入的叶结点是序列中第 1 个 LHV 值大于待插入的索引记录所指向的要素的 Hilbert 值的叶结点.若所有叶结点的 LHV 值都小于该要素的 Hilbert 值,则序列尾部结点成为待插入结点.记待插入结点为 InsertedLeaf.
3. [插入新索引记录]　向叶结点 InsertedLeaf 插入 NewIndRec(存在两种情况,如下),而后转结束.

```
if (要素簇数据量满足限定条件){
    if (叶结点 InsertedLeaf 内无空条目){
        扩充叶结点的容量,增加 nIncrEntryNum(预设常量)个空条目;
    }
    把索引记录直接插入叶结点;
    修正叶结点的祖先结点内相关索引记录的 MBR 值和 LHV 值;
}else{
    去除插入层之上的全部内部结点(即集合 HigherIntlNodeSet 内的全部结点);
    把 InsertedLayerLeafSet 内的要素索引记录和待插入的要素索引记录 NewIndRec 组成要素索引记录集合 IndRecSet;
    调用 HandleOverflow(IndRecSet,InsertedLayerIntlNodeSet,HigherLeafSet)做“上溢处理”;
}
```

4. [插入新等级]　分配一个叶结点 Leaf,包含 *n*InitialEntryNum(预设常量)个空条目,把 NewIndRec 插入第 1 个空条目.叶结点 Leaf 单独组成待插入层叶结点集合 InsertedLayerLeafSet.
5. [生成待插入层的父结点集合]　令全体等级低于 *n*Level 的要素索引记录逐级往上生成的内部结点集合为待插入层内部结点集合 InsertedLayerIntlNodeSet.从 InsertedLayerIntlNodeSet 和 InsertedLayerLeafSet 出发,为每个结点设置一条结点索引记录,分组生成父结点集合 FartherSet.FartherSet 包含 $\lceil N/M \rceil$ 个内部结点,结点索引记录均匀分布于这些内部结点内.这里,$N$ 为结点索引记录总数,$M$ 为内部结点扇出系数.
6. [往上生长成完整的索引树]　调用 GrowUp(*n*Level,FartherSet,HigherLeafSet)往上生长得到完整的索引树.
7. 结束.

#### 3.1.2 上溢处理算法

HandleOverflow(IndRecSet,InsertedLayerIntlNodeSet,HigherLeafSet)

输入:IndRecSet 为待插入层要素索引记录和新要素索引记录组成的集合.待插入层要素索引记录来自 InsertedLayerLeafSet.InsertedLayerIntlNodeSet 为待插入层内部结点集合.HigherLeafSet 为待插入层

之上的叶结点集合.

输出:完整的索引树,根结点为 Root.

算法描述:

1. IndRecSet 内的要素索引记录按照要素的 Hilbert 值从小到大排列,从前到后分组,每当累计的要素簇数据总量第 1 次超过 *n*ClusterBestSize,启用新的叶结点.最后得到新的叶结点集合 NewLeafSet.
2. 从 InsertedLayerIntlNodeSet 和 NewLeafSet 出发,为每个结点设置一条结点索引记录,分组生成父结点集合 FartherSet.FartherSet 包含$\lceil N/M \rceil$个内部结点,结点索引记录均匀分布于这些内部结点内.这里,$N$为结点索引记录总数,$M$ 为内部结点扇出系数.
3. 调用 GrowUp(*n*Level,FartherSet,HigherLeafSet)往上生长得到完整的索引树.
4. 结束.

本算法的步骤 1 对同级要素索引记录重新分组,算法为要素簇设定最接近于磁盘 I/O 最佳粒度 *n*ClusterBestSize.

3.1.3 生长算法

GrowUp(*n*Level,IntlNodeSet,LeafSet)

输入:*n*Level 为要素等级.IntlNodeSet 为当前处理的结点层次的内部结点集合,可以为空集合.每个内部结点分别是一棵子树的根结点,子树包含的叶结点的等级低于 *n*Level.LeafSet 为等级等于或高于 *n*Level 的全体叶结点组成的集合,可以为空集合.

输出:完整的索引树,根结点为 Root.

算法描述:

5. 从叶结点集合 LeafSet 找出所有等级最低的叶结点,与和内部结点集合 IntlNodeSet 一起组成结点序列 NodeSet.
6. 分别为 NodeSet 内各结点生成索引记录,全部记录按照所指向的结点的 Hilbert 值从小到大排序,记序列为 IndRecSet.
7. 记 $N$ 为 IndRecSet 序列的长度,把序列内的索引记录均匀地分配到$\lceil N/M \rceil$个内部结点,令 IntlNodeSet 等于这些内部结点组成的集合.
8. 若 LeafSet 不为空集,从叶结点集合 LeafSet 去除所有等级最低的叶结点并转第 1 步.
9. (此时,LeafSet 为空集.)若 IntlNodeSet 只包含 1 个内部结点(即为根结点 Root),则转结束;否则令 NodeSet = IntlNodeSet,转第 2 步.
10. 结束.

### 3.2 删 除

删除算法的框架和插入算法的框架相同.两种算法的部分操作相逆,比如插入算法的上溢处理相逆于删除算法的下溢处理.下溢是指叶结点所指向的要素簇的数据总量小于下限阈值 *n*ClusterMinSize.此时,所涉及层次的叶结点将重组,过程和上溢处理相同.溢出处理后的索引树生长算法完全相同.限于篇幅,在此省略删除算法的描述.

### 3.3 查 询

对于区域查询 $Q(w,s)$,基于多级 R-tree 的访问过程是:首先,索引检索输出叶结点序列;然后采用叶结点指向的要素簇(而不是单个要素数据记录)作为 I/O 单元执行数据 I/O.对同样的区域查询,上述访问过程实施 I/O 次数少于以单个要素数据记录作为 I/O 单元时的次数,而且要素簇的数据量和二级存储系统的最佳 I/O 粒度相接近,使得数据访问性能最优.被读入的要素簇可能包含若干不落在目标区域内的要素数据记录.然而,由于同级要素依据空间邻近关系分簇存储,被读入的无效要素数据记录总数能够控制在小范围内.

查询操作的算法描述比较简单,限于篇幅,在此从略.

## 4 性能评测

### 4.1 评测结果

实验目的是比较分别使用基于 Hilbert R-tree,Reactive R-tree 和多级 Hilbert R-tree (MLHR)的数据访问方法执行成批式数据访问的效率.评测方法是:在相同的软硬件平台上,分别用 3 种访问方法执行同一组区域查询,测量 3 种访问方法所耗费的时间,时间少者效率高.空间查询样本是记录用户浏览地图过程所使用的区域查询序列所生成.实验选用面向导航应用的上海市区地图数据库.数据总量为 99.20Mb,包含 99.38 万条要素数据记录.实验选用桌面机,CPU 为赛扬 1.7G,物理内存总共 256M.测试中,3 种索引树同时驻留内存.

表 1 列出执行一组包含大量区域查询的查询序列后统计得到的多项时间开销及磁盘 I/O 统计数据.所列出的每项时间都是平均值.查询时间是指完成单次查询所耗费的平均时间.它由索引检索时间、数据 I/O 时间和其他时间组成.

**Table 1** Average time used and statistics of disk I/O

表 **1** 平均时间开销统计和磁盘 I/O 统计数据

| Access methods | Total time used by queries (ms) | Time used by searches in the index (ms) | Time used by data I/O (ms) | Average granularity of data I/O (bytes) | Counts of data I/O |
|---|---|---|---|---|---|
| Hilbert R-tree | 61.028 | 2.234 | 54.102 | 122.55 | 1 736 651 |
| Reactive R-tree | 51.419 | 1.516 | 45.396 | 122.55 | 1 736 651 |
| MLHR | 24.109 | 0.176 | 23.397 | 4 670.18 | 52 695 |

从表 1 能够得出以下结论:

- 基于 MLHR 的数据访问方法的效率明显高于其他两种方法的效率.这是通过优化 I/O 粒度和增加同级要素簇聚性得到的收益.
- 索引搜索时间不足数据 I/O 时间的二十分之一.因此,减少数据 I/O 时间是提高访问效率的关键.

使用基于 MLHR 的数据访问方法,WebGIS 服务器可以显著提高响应速度.图 2 展示了分别使用 3 种方法得到的空间查询吞吐率的对比.空间查询吞吐率代表单位时间内服务器能够处理用户请求的最大数目.

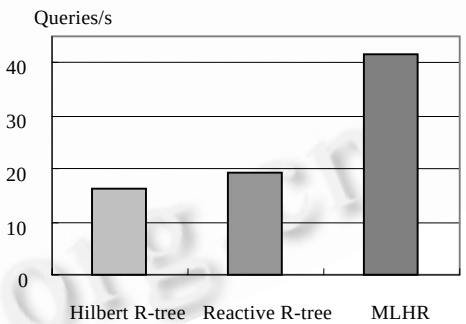


Fig.2 Query throughput

图 2 查询吞吐率

表 2 列出单次查询用于 I/O 的时间值.“全序列最大值”一栏列出的数值是各次查询用于 I/O 的时间的最大值.“浏览全地图操作”一栏给出浏览全地图操作(单次区域查询)用于 I/O 的时间量.从“浏览全地图操作”一栏可以清楚地看出数据簇聚和优化 I/O 粒度所带来的收益.其中,基于 Hilbert R-tree 的访问方法用于 I/O 的时间很长,正是同级要素弱簇聚造成的结果.

**Table 2** Time used by data I/O (ms)

表 **2** 数据 I/O 时间开销(ms)

| Access methods | Maximal time used by each query | Time used by accessing the whole map |
|---|---|---|
| Hilbert R-tree | 2 851.355 | 2 851.355 |
| Reactive R-tree | 413.481 | 304.368 |
| MLHR | 325.971 | 22.243 |

在 WebGIS 应用中,地图数据集的主体部分可视为静态的(仅偶有变动).只要把经常变化的少部分数据(若有的话)另作考虑,多级 R-tree 将是少有变化的.于是,多级 R-tree 的构造以及动态维护的开销对应用性能影响不大.因此,本文对它们不作考察.

需要指出的是,实验使用的地图数据库属于中型规模.我们估计,对于更大规模的地图数据库,基于多级 R-tree 的数据访问方法的优势将更加明显.因为版权等因素,我们无法获得规模更大的实际地图数据库,故未能

作更全面的测验.

### 4.2 性能分析

在 WebGIS 应用中,区域查询 $Q(w,s)$的效率是最值得关心的.从以上实验结果可以得出,基于多尺度 Hilbert R-tree 比基于 Hilbert R-tree 或 Reactive R-tree,区域查询效率更高.原因在于:(1) 以要素簇作为 I/O 单元,优化了数据 I/O 粒度,减少了磁盘操作次数.(2) 保证同级要素的数据记录之间的簇聚性,由于地图浏览的应用特性决定同级要素经常一起被使用,从而提高数据访问效率.

在空间利用率方面,因为叶结点的尺寸按需增长,多级 R-tree 几乎达到 100%.实验中我们发现,多尺度 Hilbert R-tree 的空间开销与静态 Hilbert R-tree[3]的空间开销相差不大.

多级 R-tree 的不足在于:(1) 要求应用系统内存空间较富裕.动态设定叶结点扇出系数这一做法要求索引树完整地驻留内存,加剧了内存开销.(2) 索引树不宜频繁地在线变动.过多变动既增加内存管理开销,又降低稳定性.对于必要的变动,可以施加于离线维护的拷贝,必要时切换使用.

## 5 结 论

每一个请求/响应回合,WebGIS 服务器都进行具有多尺度特性的成批式数据访问.基于普通的 R-tree 索引的数据访问方法因“同级要素弱簇聚”和“I/O 粒度偏小”使得成批式数据访问效率不高.基于多级 R-tree 的数据访问方法避免了上述两种现象,明显地提高了数据访问效率.实验数据证明了这一点.多级 R-tree 要求索引树完整地驻留内存.WebGIS 服务器拥有大容量内存,能够做到这一点.

**References**:

[1] Kamel I, Faloutsos C. Hilbert R-tree: An improved R-tree using fractals. In: Bocca JB, Jarke M, Zaniolo C, eds. Proc. of the 20th Int’l Conf. of VLDB. Santiago: Morgan Kaufmann Publishers, 1994. 500−509.

[2] Guttman A. R-trees: A dynamic index structure for spatial searching. In: Yormark B, ed. Proc. of the Int’l Conf. on Management of Data. Massachusetts: ACM Press, 1984. 47−57.

[3] Kamel I, Faloutsos C. On packing R-trees. In: Bhargava B, Finin T, Yesha Y, eds. Proc. of the 2nd Int’l Conf. on Information and Knowledge Management. Washington, DC: ACM Press, 1993. 490−499.

[4] Beckmann N, Kriegel HP, Schneider R, Seeger B. The r*-tree: An efficient and robust access method for points and rectangles. ACM SIGMOD Record, 1990,19(2):322−331.

[5] Oosterom PV. Reactive data structure for geographical information system [Ph.D. Thesis]. Leiden: University of Leiden, 1990.

[6] Edward PFC, Kevin KWC. On multi scale display of geometric objects. Data and Knowledge Engineering, 2002,40(1):91−119.

[7] Hörhammer M, Freeston M. Spatial indexing with a scale dimension. In: Güting RH, Papadias D, Lochovsky FH, eds. Proc. of the 6th Int’l Symp. on Advances in Spatial Databases. Hong Kong: Springer-Verlag, 1999. 52−71.

[8] Zhou S, Christopher BJ. Design and implementation of multi-scale databases. In: Jensen CS, Schneider M, Seeger B, Tsotras VJ, eds. Proc. of the 7th Int’l Symp. on Advances in Spatial and Temporal Databases. London: Springer-Verlag, 2001. 365−386.

[9] Ye CC, Zhou XM. A method for organizing map dataset to support multi-scale. Chinese Journal of Computers, 2004,27(7): 964−970 (in Chinese with English abstract).

附中文参考文献:

[9] 叶常春,周兴铭.一种支持多比例尺表示的地图数据组织方法.计算机学报,2004,27(7):964−970.