ATTI:负载关注的查询自适应轨迹索引

孟祥旭,王晓东,周兴铭
(国防科学技术大学计算机学院并行与分布处理重点实验室,湖南长沙 410073)

摘 要: 当前基于空间切分的轨迹索引不能实现时空同步,在负载和查询范围动态变化时性能显著下降.本文提出负载自适应的时空八叉树,实现轨迹索引的时空同步;进而扩充单棵时空八叉树数据结构形成虚拟森林,优化现有基于查询范围均值的单树索引,以适应时空查询范围的动态变化.实验表明,该索引可将时空范围查询延迟降低50%以上.

关键词: 自适应;空间索引;时空范围查询;八叉树
中图分类号: TP311.13 **文献标识码:** A **文章编号:** 0372-2112 (2013)04-0625-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.04.001

ATTI:Workload-Aware Query Adaptive OcTree Based Trajectory Index

MENG Xiang-xu, WANG Xiao-dong, ZHOU Xing-ming
(*Key Lab for Parallel and Distributed Processing, National University of Defense and Technology, Changsha, Hunan* 410073, *China*)

Abstract: Currently, temporal and spatial factors are not considered together in current history trajectory indices, and the performance of indexes is sensitive to query size. This paper realizes adaptive octree based Trajectory clustering index(ATTI), which uses octree index spatial and temporal factors together and implements the query workload adaptive mechanism by virtual octree forest. The results show that ATTI can reduce half of range query process delay.

Key words: adaptation; spatial index range query; spatio-temporal; octree

1 引言

具有定位功能移动设备的普及使得收集大规模位置轨迹成为可能.MIT的CarTel工程[1],微软亚洲研究院的GeoLife工程[2],都自2007年起收集用户的真实轨迹数据,并且研究轨迹索引[3]、背景地图匹配[4]、相似性挖掘等一系列问题,开发了智能导游系统、驾驶智能指引、旅游热点搜索等应用.用户轨迹作为众多基于位置服务(LBS)的基础资源之一,需要支持多种用户的频繁检索.

北京有上万辆出租车安装了GPS设备,每天产生上百G的轨迹数据.当警方需要查看在18日9:00-12:00之间经过西直门立交桥的所有车辆的轨迹时,由于数据量过大使得查找非常困难.因此,按照位置和时间对轨迹进行索引,并根据索引安排数据在磁盘的存储以提升查询效率是非常必须的.检索轨迹的最常用方法是时空范围查询,如上例所述,返回 t_1-t_2 时间段内,某个空间区域内的所有轨迹.当轨迹规模较大时,时空范围查询的结果将占据较大的磁盘空间.因此,如何高效的从众多的轨迹数据中查找出所需的子轨迹,并且在最小的时延内将数据从磁盘返回用户是一个急需解决的问题.本文的主要工作是设计并实现高效、自适应的轨迹数据存储及索引机制,提升处理轨迹时空范围查询的效率.

2 相关工作及问题提出

2.1 相关工作

在数据库领域提升查询效率的主要手段是建立索引,轨迹数据的高效查询也不可例外的借助索引技术.轨迹数据既包含移动对象的空间特性,也包含对象移动的时间特性,是一种典型的时空数据.因此,轨迹索引技术是时空索引的一种.

时空数据索引可以根据索引对象状态,分为当前(未来)位置索引和历史位置索引.根据移动对象运行的区域是否受到限制,分为自由空间索引和受限空间索引.轨迹数据索引属于自由空间的历史位置数据索引.表1列举了部分该类索引.

收稿日期:2011-12-13;修回日期:2012-11-08
基金项目:国家自然科学基金(No.61170260)

表 1 历史位置数据索引概览

3DRTree[5](1998)	(R 树系列)R 树的 3D 扩展.
MV3RTree[6](2001)	(R 树系列)融合了多版本 R 树和 3DR 树.即每个时间段建立一棵 R 形成多版本 R 树,并基于多版本 R 树的叶节点建立节点共享的 3DR 树.
SETI[7](2003)	(空间网格 + R * 树)基于空间网格对轨迹进行划分,在每个子网格内建立 R * 树.
BBXTree[8](2005)	(时间段 + B 树)移动轨迹按照时间切分,并采用空间填充曲线表示对象的空间位置,建立 B + 树森林.
CSE-tree[9](2008)	(空间网格 + B + 树)对空间进行网格切分,网格中的子轨迹依据终止时间建立时间索引.基于更新操作集中在距离当前时间较近的时间段这一观察,在临近时间段建立 B + 树索引,而对于距离当前时间较远的轨迹段采用动态数组索引.
PIST[10](2008)	(空间网格 + B 树)对空间进行最优化网格切分,在子空间内建立时间索引.
QuadTree[1](2010)	(空间四叉树 + 时间索引)基于空间划分建立一棵负载关注的不均衡四分树.每个空间切分单元中建立时间索引.

根据索引建立方法的不同,自由空间历史索引也可以分为两大类:基于空间切分的索引和 R 树类空间索引.R 树类索引的主要不足是:(1)数据插入时,可能导致节点分裂,更新开销较大;(2)范围查询效率受到数据分布的严重影响,当空间数据密度较高时,各个节点的边界矩形(MBR)重叠严重,导致查询效率降低;而当数据较稀疏时,中间节点边界矩形(MBR)的"死空间"比例升高.当 R 树向时间维度扩展时,形成三维 R 树(3DR 树[5],MV3R 树[6]),各个节点的 MBR 重叠度和"死空间"比例更高,严重影响了查询效率.

轨迹数据具有较高的更新频率,且分布极其不均.如表 1 所示,自 MV3DR 树以来最新的历史轨迹索引均未采用 R 树及其扩展方法,而是转向空间切分方案.文献[1,7]的实验结果表明其各自所提出索引的性能均优于 R 树类索引.

2.2 问题提出

基于空间切分的索引创建开销较低[11],且在处理范围查询时性能优于 R 树类索引,但是仍旧面临如下挑战:

(1)对于时、空等多个维度,如何进行同步索引;

(2)能否根据空间数据的动态变化自适应的调整空间切分粒度;

(3)查询开销同空间切分紧密相关,当查询范围变化较大时,单一的切分难以保证处理所有查询时性能最优;

(4)同查询区域相交的切分区域包含的部分数据点并不属于查询结果,二次筛选带来额外开销.

值得注意的是:现有的空间切分索引均为嵌套索引(Nested index or Multiple-level index,或称为多级索引[12])——首先对时间维(或者空间维)建立一级索引,而后在空间维(或者时间维)建立二级索引.嵌套索引均未实现时空的同步索引,导致时空上相近的轨迹对象并未存储在临近的磁盘空间,检索时磁盘的寻道时延增加.3DR 树或者 3 维网格切分可以同时对空间、时间维度进行索引,理论上,它们应该具有更高的时空范围查询处理效率.

综上所述,如果能够利用 3 维索引的空、时聚集效果,并且避免其构建、更新和查询的高开销,将会有效提升时空范围查询的效率.幸运的是,轨迹数据不同于具有固定形态的 3D 对象,能够当作离散的三维点,根据需求进行切分来提升查询和存储效率.

3 时空范围查询开销数学模型

与一个时空范围查询对应的三维(时间、空间)体积为 S_Q,轨迹的时空密度表示为 D,每个轨迹点占据 k 字节.则查询结果占据的磁盘页数量如下:

$$N_{\text{page}} = \left\lceil \frac{|S_Q * D| * k}{PageSize} \right\rceil \tag{1}$$

根据磁盘的工作原理[12],查询的时间开销主要由两部分组成,第一部分为查找结果页面地址所需的时间.该时延受到多种因素的影响,主要包括是否建立索引,内存索引还是磁盘索引,索引的大小,以及查询算法的计算复杂度,处理器的计算能力等.本文设计的索引同文献[1]一样,使用内存索引——即内存足够大可以保存整个索引.因此,索引访问不涉及磁盘的 I/O 操作,仅同索引内存中数据结构和查询处理算法有关,很难定量描述.同时,时空范围查询结果数据量较大,读取这些数据需要多次 I/O 操作,开销高于内存访问.本文主要优化比例较大的磁盘 I/O 时间.即:

$$C_{\text{IO}} = (T_{\text{transfer}} + A * T_{\text{move}}) \left\lceil \frac{|S_Q * D| * k}{PageSize} \right\rceil \tag{2}$$

其中 T_{transfer}表示传输一个数据块所用时间,T_{move}表示磁头移动过一个数据块所用时间,A 表示每两个数据块之间间隔的平均数据块.如式(2)所示,I/O 开销主要受磁盘页面分布情况以及数量决定.

4 ATTI 的负载均衡机制(时空八叉树)

本文基于查询开销模型,采取动态的多维同步切分策略,建立负载关注的时空八叉树聚集索引.

4.1 动态空间切分

如图 1 所示,将一个空间区域(整个体积表示为

Square)切分为8个相等的子区域,而后将各个子区域迭代切分,形成多级八叉树.其中核心的问题是如何根据轨迹数据的动态变化调整树的结构,保证处理时空范围查询时磁盘的I/O次数最少.

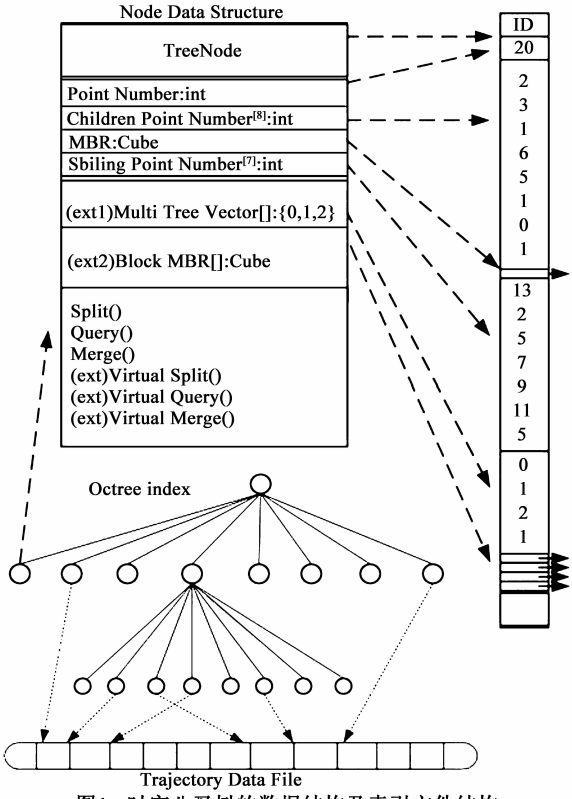

图1　时空八叉树的数据结构及索引文件结构

首先考虑最简单的场景:树的各个分支深度相等——网格切分.空间切分的最小子区域称为一个切分单元(*cell*).首先分析在这样一个均匀网格中,一个时空范围查询的磁盘I/O开销.每个网格单元可以表示为一个三维时空区域,用 r_x, r_y, r_t 分别表示在三个维度的间隔.假设轨迹数据在该区域的密度为 D,对于任意一个查询使用 qr_x, qr_y, qr_t 表示在每个维度上的查询间隔.T_{transfer}和 T_{move}为常数值,分别表示为 α 和β,则I/O开销为:

$$C_q = \sum_{cells} P(q \cap cell)(\alpha + \beta A)\left\lceil \frac{k * r_x * r_y * r_t * D}{PageSize} \right\rceil \tag{3}$$

其中,$P(q \cap cell)$为查询区域同 *cell* 的相交概率如式(4):

$$P(q \cap cell) = \frac{(qr_x + r_x) * (qr_y + r_y) * (qr_t + r_t)}{Square} \tag{4}$$

将式(4)带入式(3)得出:

$$C_q = \sum_{cells} \left(\frac{(qr_x + r_x) * (qr_y + r_y) * (qr_t + r_t)}{Square}(\alpha + A * \beta)\right.$$
$$\left.\left\lceil \frac{k * r_x * r_y * r_t * D}{PageSize} \right\rceil\right) \tag{5}$$

由式(5)可知,开销主要由查询范围以及网格单元大小,和数据分布密度 D 决定.首先讨论如何确定最优的网格大小.假设查询已经确定,则最优的网格单元为使查询开销最小的状态 .因此,最优的网格切分同轨迹数据密度 D 紧密相关.然而,轨迹的分布并不均匀(系统负载),更重要的是移动对象的轨迹更新频繁,新轨迹点的加入将动态的影响数据的分布.因此,最优的网格切分也将动态变化.

4.2　时空八叉树索引的数据结构

基本思路:如图1左图所示,系统动态的进行空间划分,数据密集的区域切分次数相对较多,数据稀疏区域的切分次数相对较少.该不等深的空间划分对应于一棵时空八叉树.当有新的轨迹加入时,每个切分单元都需要动态的判断如何调整:(1)继续切分,将该单元切分为8个新的子单元;(2)将该单元同另外7个兄弟切分单元合并.判断的依据是查询处理的I/O开销是否变小.

如图1的"Node data Structure"子图所示,每个八叉树节点保存对应的切分区域内包含的GPS点的数目(成员变量 PointNumber).同时计算若该区域进一步切分,八个子区域内各包含的GPS点的数目,保存在数组 ChildrenPointNumber 中.在成员数组 bilingPointNumber 中保存七个兄弟节点包含的GPS点的数目.

根据式(4)可知,给定一个查询,即可计算出每个切分单元同它相交的概率,并进一步计算该网格单元处理该查询时的I/O时间开销:

$$C_{cell}(q) = \frac{(qr_x + r_x) * (qr_y + r_y) * (qr_t + r_t)}{Square}(\alpha + A * \beta)$$
$$\left\lceil \frac{k * \sum_{i=1}^{8} P_i}{PageSize} \right\rceil \tag{6}$$

其中 $\sum_{i=1}^{8} P_i$ 表示该区域中的8个子区域的轨迹点数目的和,即 PointNumber 的值.而 P_i 则表示该区域中的点落在第 i 个子区域的数目.当有新轨迹点插入时,每个叶节点都计算若进一步分国八个子区域时,处理同一个查询的开销值.即对假设的八个子区域的查询开销求和:

$$VirtualC_{\text{cell}}(q) =$$
$$\sum_{i=1}^{8} \left(\frac{(qr_x + r_x/2) * (qr_y + r_y/2) * (qr_t + r_t/2)}{Square}\right. \tag{7}$$
$$\left.(\alpha + A * \beta)\left\lceil \frac{k * P_i}{PageSize} \right\rceil\right)$$

因此,为了保证查询的开销最小,只有当

$VirtualC_{\text{cell}}(q) < C_{\text{cell}}(q)$时才进一步执行切分(split)操作.同理,随着轨迹的加入也会导致切分后该单元查询开销高于切分前,这时则需要合并切分区域.合并判断需要计算若一个切分区域同它的七个兄弟区域合并后的查询开销,并将该值同当前的八个区域的开销和进行比较.兄弟节点合并后的开销可以用如下公式计算:

$$ParentC_{\text{cell}}(q) = \frac{(qr_x + 2r_x) * (qr_y + 2r_y) * (qr_t + 2r_t)}{Square} (\alpha + A * \beta) \left\lceil \frac{k * \sum_{i=1}^{8} PSible_i}{PageSize} \right\rceil \tag{8}$$

在未合并前八个节点的查询开销和为:

$$LevelC_{\text{cell}}(q) = \sum_{i=1}^{8} \frac{(qr_x + r_x) * (qr_y + r_y) * (qr_t + r_t)}{Square} (\alpha + A * \beta) \left\lceil \frac{k * PSible_i}{PageSize} \right\rceil \tag{9}$$

其中,$PSible_i$ 表示兄弟节点.则当 $ParentC_{\text{cell}}(q) < LevelC_{\text{cell}}(q)$时进行合并.

如上方法能够根据轨迹点加入时查询开销的变化,对时、空区域进行动态切分,实现了负载的自适应.当前,均假设查询的大小已知,然而现实中并不可能预知所有用户的查询大小.下一节讨论如何实现查询的自适应,生成能够为所有的查询提供最低处理时延的索引.

5 ATTI 的查询自适应机制(虚拟森林)

5.1 查询范围对索引性能的影响

负载自适应八叉树可以根据负载的变化动态的调整树的结构,但是调整依靠的开销公式中查询大小是用一个固定的数值表示的[1,10].这个均值并不能够有效的表示所有查询.解决查询自适应问题的方案应该:保持多棵索引树的查询性能优势,同时降低多棵树的存储和维护开销.

算法1 森林中真实树节点Insert算法

```
Input:GPSPoint:Point,Octree-Node:Node
Ouput:Whether insert success
Variables:Whether split:SplitIndicator
Whether merge:MergeIndicator
Index of Virtual tree:index
Costs of index-th virtual tree according equation 6,7,8,9:
Cost[index],Virtualcost[index],ParentCost[index],
LevelCost[index]
1:SplitIndicator=false;
2:MergeIndicator=true;
3:if(Node is leaf node)then
4:    add Point into Node;
5:    for all index do
6:       compute Cost[index],VirtualCost[index],
       ParentCost[index], LevelCost[index];
7:       if(VirtualCost[index]<Cost[index])then
8:          SplitIndicator=ture;
9:       end if
10:      if(ParentCost[index]>LevelCost[index])then
11:         MergeIndicator=false
12:      end if
13:   end for
14:   if(SplitIndicator is true)then
15:      call Split-Method to create 8 NewNode;
16:      for all index do
17:         set MultiTreeVector[index]of NewNode to 0;
18:      end for
19:      get all points of NODE to do reinsert-method;
20:   end if
21:   if(MergeIndicator is true)then
22:      add all the points belong to Node and its 7 SiblingNode
         to her ParentNode;
23:      set ChildrenPointer of ParentNode to null;
24:      for all index do
25:         if(MultiTreeVector[index]≠0)then
26:            set MultiTreeVector[index]=0;
27:         end if
28:         set MultiTreeVector[index]=0;
29:      end for
30:      end if
31:      return ture;
32:else
33:      get SubNode to which Point belongs;
34:      call RealInsert-Method of SubNode;
35:end if
```

5.2 虚拟索引森林

基本思路:每一棵树都是通过空间切分形成的,任何切分较深的子空间形成的分支必定包含该切分较浅的情况下生成的树的分支.因此,可以通过在多棵树之间共享节点,降低树的维护和存储开销.用户只需要设定最小查询范围和最大查询范围,以及需要建立的索引树的个数.系统通过用户设定的 3 个参数计算出一个组查询值,组成向量.并根据每个查询建立一棵虚拟索引树(这些树并不真正存在,因此称为虚拟索引树),形成一个查询森林.为了降低维护和存储开销各个树共享中间节点,形成一个“虚拟”的索引树森林.在系统启动时,用户设定最小查询和最大查询(可以是绝对数值,也可以是初始区域的百分比).同时设定虚拟查询森林的规模,即包含的查询树的个数.分析式(6)、(7)、(8)、(9)可知最小或者最大查询范围对应的查询树,不一定每个分支都是最深的.因此很难确定使用哪个查询范围才能建立一棵可以包含其他所有树的节点的索引树.为了生成一棵包含所有虚拟树节点的查询树,需要修改第 4 节中区域分割和合并的标准,利用整个查询向量元素的值判断一个空间区域是否合并或者分割:

(1)只要有一个虚拟树需要分割,则对空间进行分割;

(2)只有当所有的虚拟树都需要合并时,才进行合并.

很明显,基于该策略生成的查询树,必定是在每个子区域都切分最深的.该空间切分对应的查询树称为真实树,使用该树指导数据的存储.而其他的查询范围生成的虚拟查询树为虚拟树,只处理查询,不对数据进行聚集.

每次有数据插入时,在调用真实查询树的插入算法时,同时也调用每个虚拟树的插入算法.因为虚拟算法的开销计算需要使用真实树的信息,因此必须首先调用真实查询树的插入方法,而后调用虚拟树的插入算法.当某个虚拟树需要执行插入算法时,实体树的相应节点已经执行了分割操作,并创建了 8 个新的子节点.因此,虚拟的分割和合并只对虚拟向量进行操作,不涉及真实的节点生成和撤销(真实树的深度一定大于等于虚拟树).

5.3 磁盘页面索引的构建

当真实树生成后,一个切分单元包含的数据可能需要跨越多个磁盘页面存储.在索引中记录每个数据块存储数据的时空区域,可以消除加载和处理无关页面带来的开销.因此,在查询树的叶节点的数据结构中添加 BlockMBR 数组(图 1 中已添加)保存页面的时空范围描述信息.

5.4 自适应查询算法

基于索引森林进行查询时,首先确定最合适的虚拟查询树,之后调用该树根节点的时空范围查询算法找出所有同查询区域相交的叶节点,并将查询区域同 BlockMBR 中的磁盘页面时空区域进行对比,筛选出所有相交的磁盘页面.

算法2　虚拟树节点的时空范围查询处理算法

```
Input:Virtual-Node:Node,Spatio-temporal region:Cube
Ouput:Disk pages:Vector
Variables:Index of Proper Virtual tree:S
1:if(Node.MultiTreeVector[S]==2)then
2:   for all(DiskPage belongs to Node)do
3:      if Region of DiskPage intersect with Cube then
4:         add DiskPage to Vector;
5:      end if
6:   end for
7:   return Vector;
8:else
9:   if(NodeMultiTreeVector[S]==1)then
10:     for all i=1:8 do
11:        if Region of ChildrenNode[i]intersect with Cube then
12:           call QueryProcessing-Method of ChildrenNode[i];
13:        end if
14:     end for
15:  end if
16:end if
```

最优虚拟树的选择方法:搜索查询向量中同查询区域最相近的元素,该元素对应的虚拟索引树即为最优合适索引树.本文采用在时间、空间三个维度的上的查询间隔的欧几里德距离进行最近元素查找,即两个查询区域的空间维度和时间维度间隔的平方和表示两个查询的距离.向量中同查询最相近的元素(index)作为虚拟查询的输入.虚拟森林调用第 index 棵虚拟树的查询函数,该棵虚拟查询树调用自己 top 节点的虚拟查询函数进行数据页的查找.每个节点的虚拟查询算法如下算法 2 所示.

6 性能分析

6.1 实验平台

为了测试自适应八叉树索引的性能,我们建立了基础的实验平台.

(1)软、硬件配置:4 核 Intel i3 处理器,主频 2.27GHz;物理内存 2G(Java 虚拟机内存设定为 1G);SATA-II 磁盘,转速 5400 转/分钟,磁盘页面大小为 4KB(4096Byte).为了避免系统缓存对结果的影响,在每个查询之后执行 linux 的 /proc/sys/vm/drop_caches 例程进行缓存清除操作.

(2)比较对象:Cartel 工程的两级索引[1],该索引首先基于空间切分建立四叉树,而后在每个树节点中建立时间索引.根据文献[1]的性能评测可知,其性能优于二维网格,基本的二维 R 树索引和文献[14]设计的分簇(ClustSplit)索引.同时优于其它已有的二级索引,包括 SETI 索引.而 SETI 优于 3DR 树索引[7].因此,本文工作不再同其它二级索引以及 R 树的时间维扩展类索引进行比较.

(3)数据集:真实数据为 Geolife 工程从 2007 年 ~ 2009 年采集的全部 GPS 轨迹数据.用户数目为 165,总条目为 2300 万条,占用磁盘空间 1.02G,空间范围主要为北京市及周边,个别用户活动区域涉及到整个欧洲和美洲.在测试时,空间和时间范围均进行了归一化处理.

①GeoLife 真实负载数据集:依次取全部数据集的前 2000 万条的 25%,50%,75%,100%,生成真实数据集:GeoLife-25%, GeoLife-50%, GeoLife-75%, GeoLife-100%.

②查询数据集:随机生成占初始空间区域 1%,5%,25%(长宽比不超过 4:1),占时间区间分别为 10%,23%,50% 的查询:query-1;query-5;query-25,每个查询数据集包含 100 个查询.同时生成时空区间占初始时空立方体 0.00001% 至 12.5% 之间的随机的 100 个查询,组成查询数据集:query-random.每次测试后,取每个查询数据集中 100 个查询的平均查询时延及 I/O 次数作为性能评价的指标.

6.2 自适应八叉树性能分析

该小节使用负载数据集GeoLife-25%和查询数据集

query-random 进行实验.

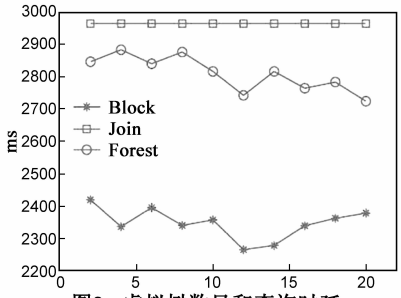

图2 虚拟树数目和查询时延

如图 2 所示,基于虚拟森林(Forest,圆点曲线)执行自适应查询处理的时延较之基于固定的时空八叉树(Join,方形曲线,查询规模为 10% 时生成的自适应八叉树)查询处理时延降低约 5% ~ 10% .但是,当虚拟树的个数从 2 增加到 20 时,查询时延的变化仅为 5% .这一结果表明,虚拟树个数对查询性能影响不大.我们认为主要原因是 ATTI 是基于内存的索引.另外,磁盘块索引的加入(Block,星形曲线)可以减少约百分之二十的查询时延.

6.3 性能对比实验

索引的创建时间及内存空间占用量:如表 2 所示,ATTI 索引数据占用空间并不比嵌入式索引节省.虽然,ATTI 并未带来索引空间的降低,其大小依旧是可以接受的.索引在建立时,平均一秒钟可以索引 2 万条数据(包括数据从磁盘读出的时间),对于每秒钟更新一次的 GPS 设备,一台普通的个人电脑可以为两万个用户更新的数据建立索引.两千万条记录(165 个用户两年的轨迹数据)的索引占用的空间仅为 50MB,可以长期驻存在内存中.

表 2 ATTI 索引大小(KB)和建立时间(ms)

Q / D	Query-1	Query-5	Query-25
	ATTI, Nested	ATTI, Nested	ATTI, Nested
25%	12.63(282), 14.07(445)	12.63(284), 13.27(445)	12.62(282), 12.91(452)
50%	23.64(589), 26.98(915)	23.63(580), 27.04(909)	23.63(580), 26.72(526)
75%	35.02(883), 40.72(799)	35.02(884), 41.17(800)	35.03(887), 41.05(807)
all	46.4(1180), 54.5(1078)	46.4(1181), 54.0(1091)	46.4(1194), 54.6(1083)

真实数据集下不同索引的查询处理性能对比:如图 3 的数据显示,可以发现在真实数据集下,时空联合索引带来的时延消减较之均匀分布的模拟数据更多.在处理范围较小的查询时,ATTI 时延仅为嵌套索引的百分之五.这一结果符合实际,因为当数据集的时间跨度较大时(GeoLife 的数据为两年时间),嵌套索引的第一级索引只划分空间,而数据在时间维度的临近性并没有得到很好的体现.因此,嵌套索引在处理时间间隔较小的查询时,较之同时考虑了时间和空间特性的联合索引性低是合理的.在真实数据负载下,负载均衡机制起到了很好的作用,将时延分别从均匀分布的 30%(query-1),50%(query-5),80%(query-25)降低到了真实数据的 5%,10% 和 35% .

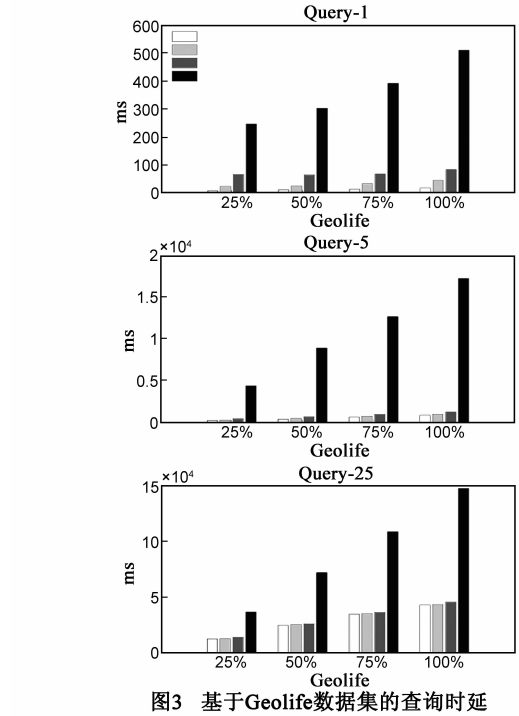

图3 基于Geolife数据集的查询时延

7 结论

当前,空间划分类索引均为嵌套索引,没有对时空维度进行同步索引.同时,空间划分类索引依旧面临负载动态变化,多用户查询的差异较大等挑战.ATTI 通过扩展二维空间切分索引到三维(加入时间维度)空间实现了时空的同步索引;基于查询开销模型动态调整空间的切分粒度实现了负载自适应;采用多树共享机制有效的解决了查询范围差异较大带来的性能下降问题.实验表明,负载关注的查询自适应时空同步索引是提升时空范围查询处理效率的有效手段.然而当数据规模较大,时空范围查询处理时延仍旧可能高达几十秒,超出了用户在线使用的容忍限度.因此,下一步我们将考虑设计海量轨迹数据的分布式存储和并行化处理平台,实现海量轨迹数据时空范围查询的实时响应.

参考文献

[1] Cudre-Mauroux, P, Wu, E, Madden, S. TrajStore: An adaptive storage system for very large trajectory data sets[A]. International Conference on Data Engineering (ICDE)[C]. Long

Beach, CA: IEEE, 2010. 109 – 120.

[2] Yu Zheng, Xing Xie, et al. GeoLife: A collaborative social networking service among user, location and trajectory[J]. IEEE Data(base) Engineering Bulletin-DEBU, 2010, 33(2): 32 – 39.

[3] Ying Cai, Toby Xu. Design, analysis, and implementation of a large-scale real-time location-based information sharing system [A]. MobiSys[C]. New York, NY, USA: ACM, 2008. 106 – 117.

[4] Yin Lou, Xing Xie, et al. Map-matching for low-sampling-rate GPS trajectories[A]. Proceedings of the 17th ACM SIGSPATIAL GIS[C]. New York, NY, USA: ACM, 2009. 352 – 361.

[5] Michael Vazirgiannis, Yannis Theodoridis, et al. Spatio-temporal composition and indexing for large multimedia applications [J]. Multimedia Systems-MMS, 1998, 6(4): 284 – 298.

[6] Yufei Tao, Dimitris Papadias. MV3R-Tree: A spatio-temporal access method for timestamp and interval queries[A]. Very Large Data Bases-VLDB[C]. Roma, Italy: Morgan Kaufmann, 2001. 431 – 440.

[7] V. Prasad Chakka, Adam Everspaugh, et al. Indexing large trajectory data sets with SETI[A]. Conference on Innovative Data Systems Research-CIDR[C]. Asilomar, CA, USA: ACM 2003, 35 – 43.

[8] Dan Lin, Christian S. Jensen. Effcient indexing of the historical, present, and future positions of moving[A]. Mobile Data Management(MDM)[C]. Ayia Napa, Cyprus: ACM, 2005. 59 – 66.

[9] Longhao Wang, Yu Zheng, et al. A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval[A]. Mobile Data Management(MDM)[C]. Beijing, China: IEEE Computer Society, 2008. 1 – 8.

[10] Viorica Botea, Daniel Mallett, et al. PIST: An efficient and practical indexing technique for historical spatio-temporal point data[J]. GEOINFORMATICA , 2008, 12(2): 143 – 168.

[11] 薛向阳,罗航哉,等. LIFT:一种用于高维数据的索引结构[J].电子学报,2001,29(2):191 – 195.
Xue Xiang-yang, Huo Hang-zai, et al. LIFT: An index structure for high dimensional data[J]. Acta Electronica Sinica, 2001, 29(2): 191 – 195. (in Chinese)

[12] H Garcia-Molina, JD Ullman, et al. Database System Implementation(Second Edition)[M]. Prentice Hall, 2009.

[13] Yannis Theodoridis, Jefferson R. O. Silva, et al. On the generation of spatiotemporal datasets[A]. Symposium on Large Spatial Databases(SSD)[C]. Hong Kong, China: Springer, 1999. 147 – 164.

[14] Slobodan Rasetic, Jörg Sander, et al. A trajectory splitting model for efficient spatio-temporal indexing[A]. Proceedings of the 31st VLDB[C]. Trondheim, Norway: ACM, 2005. 934 – 945.

作者简介

孟祥旭　男,1982 年出生,河北唐山人,博士生,主要研究领域为基于位置的服务系统.
E-mail: aiguokk@gmail.com

王晓东　男,1973 年出生,湖南长沙人,博士,教授,主要研究领域为移动计算,数据库系统,无线网络安全等.