软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(10):2631−2643 [doi: 10.13328/j.cnki.jos.004809]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

双流模式下高吞吐量移动对象范围查询算法*

薛忠斌[1,2], 周 烜[1,2], 王 珊[1,2]

[1](教育部数据工程与知识工程重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

通讯作者: 周烜, E-mail: xzhou@ruc.edu.cn

摘 要: 随着位置感知移动设备的出现及通信技术和GPS系统的不断发展,基于位置的查询在数据库领域得到了广泛的关注.研究了基于快照的空间范围查询,即,查询在某个时间段位于某个查询范围内的移动对象.范围查询是其他空间查询的基础,例如KNN查询和反KNN查询等,很容易在范围查询的基础上得到.国内外的研究者针对移动对象的范围查询问题提出了一系列的算法,然而这些算法要么关注于解决移动对象的快速更新问题,要么关注于解决范围查询的快速处理问题.在大数据的背景下,查询和更新大量涌入时,不仅要求查询算法有较快的响应速度,还要求它们能够实现较高的吞吐量,但已有算法不能很好地解决高吞吐量的问题.针对移动对象更新数据流和查询数据流,提出一种基于内存的高吞吐量移动对象范围查询算法——双向流连接(DSJ)算法.双向流连接算法采用基于快照的模式,通过在每个快照中重新构建索引的方式,以避免复杂的索引维护操作,充分发挥了硬件的性能;通过每次执行一组查询的方式,增加了数据的局部性,提高了算法的效率;在执行过程中,通过使用SIMD技术以加速查询处理过程.基于以上几点,双向流连接算法能够确保整个系统具有很高的吞吐量.在基于德国路网生成的数据集上对算法进行了测试,实验结果表明,双向流连接算法具有很好的性能表现.

关键词: 大数据;时空数据库;移动对象;范围查询;主存

中图法分类号: TP311

中文引用格式: 薛忠斌,周烜,王珊.双流模式下高吞吐量移动对象范围查询算法.软件学报,2015,26(10):2631−2643. http://www.jos.org.cn/1000-9825/4809.htm

英文引用格式: Xue ZB, Zhou X, Wang S. Throughput oriented range query algorithm for moving objects in dual stream mode. Ruan Jian Xue Bao/Journal of Software, 2015,26(10):2631−2643 (in Chinese). http://www.jos.org.cn/1000-9825/4809.htm

Throughput Oriented Range Query Algorithm for Moving Objects in Dual Stream Mode

XUE Zhong-Bin[1,2], ZHOU Xuan[1,2], WANG Shan[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

Abstract: With the development of location-aware mobile devices, communication technologies and GPS systems, location based queries have become an important research issue in the area of database. This paper studies the problem of snapshot based spatial range query which searches for the moving objects within a specific query range in a specific time interval. Range query is the building block of other types of spatial queries, such as *k* nearest neighbor query and reverse *k* nearest neighbor query. A series of algorithms have been proposed to process range queries of moving objects. However, these algorithms are either designed for fast response time or high update performance. They are not purposely designed for the situation of big data where throughput is more important as both queries and updates arrive at a very high rate. For the query stream and object update stream, a high throughput main memory algorithm—Dual Stream Join algorithm is proposed for moving object range query. DSJ uses a snapshot approach. In each snapshot, DSJ builds a new

* 基金项目: 国家自然科学基金(61272138); 中国人民大学科学研究基金(中央高校基本科研业务费专项资金资助)(10XNI018)
收稿时间: 2014-06-07; 修改时间: 2014-07-30, 2014-12-09; 定稿时间: 2014-12-17

index structure based on the update of the moving objects, which avoids maintaining a sophisticate structures and gives full play to the performance of the hardware. DSJ executes a batch queries at each run, which increases the data locality and improves the efficiency of the algorithm. DSJ also employs the SIMD technology to accelerate the query processing and makes sure that the system has high throughput. A comprehensive performance evaluation of the proposed techniques is conducted using the German network generated data. The results show that DSJ is highly efficient.

Key words: big data; spatial temporal database; moving object; range query; main memory

随着全球定位系统、无线网络等基础设施的飞速发展以及大量手持和车载无线通信定位设备的普及,基于位置的服务(location based service,简称 LBS)逐渐兴起并得到广泛应用.基于位置的服务是指移动终端利用各种定位技术获得当前位置信息,然后通过无线网络得到某项服务[1].基于位置的服务在各领域中得到广泛应用,例如:在社会生活中,通过位置服务公众用户可以定位当前位置、获取导航信息等;行业用户可以开展智能交通控制、污染物的扩散监测、飓风的移动路径及影响范围监测等;军事领域中,通过位置服务可以开展作战单元的监控与追踪管理、海上或陆地大型移动目标的跟踪与监视等.

在早期的研究中,研究者主要专注于解决移动对象的快速更新和查询实时响应问题[2−6].已有的解决方案需要构建复杂的索引结构,并随后续更新的到来不断维护索引结构.在这种原则下,产生了大量复杂的索引结构.而在以大内存和多核处理器为特点的新硬件环境下去维护复杂的索引结构,将不能充分发挥硬件的性能[4].

通过对许多基于位置服务的调研,研究者发现:在很多大数据应用中,系统的吞吐量成为关键因素.优化单个查询的处理时间,不能成为提升整个系统吞吐量的关键.对于大部分应用而言,单个查询的响应时间只要达到秒级就可以满足需求,无需刻意追求单个查询的响应时间.对于典型的应用,例如用户通过手机打车应用,大部分用户只要能在几秒钟内得到响应就能满足要求,对于更快的查询响应时间,用户的服务体验没有明显变化.对于整个系统,当大量查询涌入时,查询的排队等待时间将会影响用户的响应时间.提高系统的吞吐量,减少用户的排队等待时间,对增加用户的服务体验至关重要.

在本文中,我们针对移动对象的更新数据流和查询数据流,提出了一种新的基于内存的双向流连接算法.它充分利用了当前大内存、多核等新硬件特性,结合缓存敏感性思想、SIMD 指令集等,解决了在大数据环境下移动对象的更新以及当范围查询大量涌入时,范围查询响应的实时性和系统吞吐量最大化问题.

本文主要有以下贡献:

(1) 本文针对移动对象的更新数据流和查询数据流,提出一种高吞吐量的范围查询算法——双向流连接(dual stream join,简称 DSJ)算法.双向流连接算法主要应对查询动态变化的场景,但对于查询固定的场景,算法也能很好地应用.DSJ 算法充分运用大内存的特点,设计时也全面考虑了内存算法设计的缓存敏感性(cache conscious)等特性;

(2) 双向流连接算法采用新快照内重新构建索引的方式,避免了复杂的索引维护操作,充分发挥了硬件的性能.传统算法中,通过构建一个复杂的索引结构,然后基于更新对索引进行维护.在以大内存和多核处理器为特点的新硬件环境下,对索引的复杂更新操作将不能充分发挥硬件的性能;

(3) 双向流连接算法在运行时,通过一次执行多个查询的方式,增加了数据的局部性,提高了系统的性能.算法在执行过程中,通过一次处理多个查询的方式,实现了查询间的并行.多个查询在处理过程中,首先按照位置聚集到对应的单元格中,然后在对应的单元格执行连接操作得到结果.通过上面的两步操作,实现了查询内的并行.因此,我们设计的内存算法具有很好的空间局部性,实现了查询内和查询间的并行,提高了查询的响应时间和系统的吞吐量;

(4) 通过 SIMD 指令集、数据预取机制(prefetching)和列存储运算(column store)方式的结合,提出了 SPC 移动对象范围查询判断方法.双向流连接算法在执行时,通过一系列的划分,使连接操作变为计算密集型操作.在连接操作中,判断移动对象是否为范围查询的结果会占用大量的时间,通过使用 SPC 移动对象范围查询判断方法,消除了执行过程中的分值预测,提高了连接操作执行的效率;

(5) 在基于德国实际路网生成的数据集中,通过与已有算法进行对比,实验验证了双向流连接算法的有

效性.

本文第 1 节介绍针对移动对象范围查询展开的相关工作,重点描述支持并行处理的主存索引算法.第 2 节对文中涉及的一些预备知识进行描述.第 3 节提出基于内存的并行范围查询算法——双向流连接算法.第 4 节通过实验验证我们提出的算法的有效性.最后,第 5 节对全文进行总结.

1 相关工作

针对基于位置服务的应用,国内外学者进行了大量研究,提出了一系列的算法[2−9].例如:基于磁盘的 TPR-tree[7,8]及其变种 TPR*-tree[9]等索引,在时空数据库中得到了广泛的使用.这类索引使用时间参数化最小包围框(minimum bounding rectangle,简称 MBR)来包含移动对象,其索引结构与 R-tree 类似.在 TPR-tree 中,MBR 延伸包围框会随着时间越来越大,致使空间区域出现大量重叠的现象,从而导致性能下降.因此,需要在适当的时候对 TPR-tree 进行重构,以提高查询性能.TPR-tree 中删除、插入及更新等动态操作采用 R*-tree 标准算法,但使用时间参数的度量准则.TPR*-tree 中对 TPR-tree 的动态操作算法进行了修改,通过维护一个优先队列,保证插入路径选择的全局最优,以确保 TPR*-tree 中 MBR 的紧凑性,从而提高查询性能.

基于位置的服务要求具有较快的查询响应时间,随着内存容积的不断增大,一系列基于内存的算法被提了出来[2−6].我们关注于基于内存的、支持并行处理的范围查询算法[4−6].

MOVIE 算法[4]为基于快照的内存算法,MOVIE 算法采用电影中时间帧的概念,每个快照称为一个时间帧.MOVIE 算法充分运用当前硬件的特性,在每个时间帧中,通过把前一个时间帧的索引和移动对象更新合并,重新构建一个新的索引.MOVIE 算法中构建的索引结构为只读结构,用于应对到来的查询.对象的更新被缓存到更新 buffer 中.在 MOVIE 算法中,用线性 kd-trie[10]作为索引结构,用空间填充曲线,例如 Z-order 曲线[11,12]或者希尔伯特曲线[13],来标示每个节点.在 MOVIE 算法中,创建新索引时需要把已有的索引和更新 buffer 进行合并,耗费了大量时间.

TwinGrid 算法[5]主要应对更新密集型的工作负载.TwinGrid 算法中,通过使用数据快照的方式处理到来的查询.TwinGrid 算法中采用两个 Grid 结构:一个用于处理移动对象的更新,另一个处理查询.每隔一定的时间,TwinGrid 算法调用系统的 memcpy 命令,把更新 Grid 中的数据复制到查询 Grid 中,以更新数据快照.通过使用两个 Grid 结构,避免了执行过程中移动对象更新和查询的读写冲突,提高了算法的效率.通过实验显示,TwinGrid 算法的性能优于 MOVIE 算法.

基于 TwinGrid 算法,PGrid[6]算法继而被提出.PGrid 算法为 TwinGrid 算法的改进版.在 PGrid 算法中,只包含一个 Grid 结构,查询和更新在同一个 Grid 结构中进行.在 PGrid 中,通过使用改进的锁协议来应对读写冲突问题.实验结果显示,PGrid 的查询和更新性能优于 TwinGrid.

上述提到的索引结构都采用了每次处理一个查询(one-by-one)的方式.当大量查询到来,查询需要排队等待响应时,排队等待的时间会对查询的整体响应时间产生影响.为了缩短查询时排队等待的时间,在本文中,我们采用每次执行一组查询(group-by-group)的方式.在执行过程中,对组内的查询首先执行聚集操作,然后通过 join 操作得到结果,实现了查询间和查询内的并行.在这种方式下,查询执行时增加了数据的局部性,减少了高速缓存缺失(cache miss),使单个查询的响应时间和系统的吞吐率都有明显的提升,能够很好地应对大量查询涌入时的场景.

在已有的工作中,与我们当前工作最相关的是 SINA[14].SINA 主要解决移动对象的持续性范围查询问题.SINA 中,通过把移动对象和范围查询执行空间连接来得到结果.在执行过程中,对查询结果进行增量维护,基于对象的更新对查询结果进行更新.SINA 算法和我们当前的工作相比有以下几点不同:

(1) 应对场景不同:SINA 算法主要应对对象更新较快、查询变化不大的场景;我们的算法主要解决在大数据环境下查询动态变化的场景,对于查询变化不大的场景,我们的算法也能很好地应对;

(2) 索引构建方式不同:SINA 中采用首先创建索引,然后随着更新的到来不断对索引进行维护的方式;我们的算法中充分发挥新硬件的性能,在每个快照中重新构建新的索引结构,避免了复杂的索引维护

操作;

(3) 解决的查询不同:SINA 中主要解决持续性范围查询;我们的算法应对基于快照的范围查询;

(4) 所基于的存储介质不同:SINA 为基于磁盘的算法,执行过程中的主要瓶颈为磁盘的 I/O;我们的算法中,数据全部放入内存,算法设计时采用了一些高速缓存敏感的优化策略.

2 预备知识

2.1 问题定义

考虑与我们的生活相关的基于位置服务的应用,例如用户通过手机打车的应用,手机用户和车辆作为移动对象,周期性地向中央服务器报告位置信息,移动对象的位置信息与时间相关.服务器基于用户和车辆的位置向用户发送服务信息.这类应用要求通过基于快照的查询得到结果,我们给出相应的形式化描述.

定义 1(移动对象). 移动对象用三元组(OID,$\langle X,Y\rangle$)表示,其中,OID 为移动对象的标识;$\langle X,Y\rangle$表示移动对象在二维欧式空间中的坐标,移动对象的位置随时间不断发生变化.所有的移动对象存储在对象表(object table)中.

移动对象位置的不断变化要求索引结构简单且容易维护.我们用 Grid 结构来索引移动对象.

定义 2(Grid 索引). 对所要查询的区域通过大小相同的单元格进行划分,每个单元格被称为一个 Cell,所有的单元格构成一个 Grid 索引结构.

定义 3(范围查询). 范围查询用五元组(QID,$\langle X_{low},Y_{low}\rangle$,$\langle X_{high},Y_{high}\rangle$)表示,其中,$QID$ 为查询的标识,$\langle X_{low},Y_{low}\rangle$和$\langle X_{high},Y_{high}\rangle$为查询区域的左下角和右上角空间坐标.

当单元格被查询完全覆盖时,单元格中的所有对象为查询的结果;当单元格被查询部分覆盖时,需要在单元格中进行检查,以确定里面的对象是否为查询结果.对于持续性范围查询和 KNN 查询等,很容易从范围查询中得到结果[4].

定义 4(系统吞吐量). 系统在单位时间内能够响应的查询数量.

移动对象在空间内自由移动,位置信息持续更新,形成一个移动对象更新数据流.范围查询持续不断地到来,形成一个查询数据流.我们主要考虑在内存多核环境下对于对象更新数据流和范围查询数据流,既能够保证查询响应的实时性,同时又确保系统保持高的吞吐量.

2.2 Grid中单元格编码方法

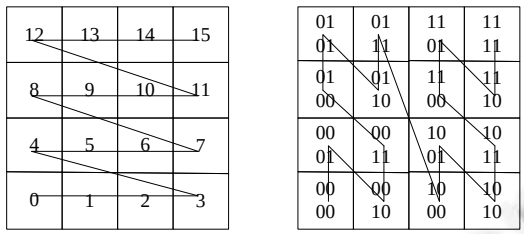
在对 Grid 中的单元格进行编码时,常用的方法有:

- 常规编码方法,即,按照从左到右的方式对单元格进行编号,如图 1(a)所示;
- 用 Z-Curve 曲线或者希尔伯特曲线对单元格进行编号,图 1(b)为 Z-Curve 编码方法.当用 Z-Curve 对 Grid 中的单元格进行编码时,对于单个范围查询,在执行时具有很好的空间局部性.

例如,假设范围查询 Q 涉及到图 1(a)所示中 Grid 左下角的 4 个单元格,对于常规编码方法,即为编号为 0,1,4,5 的单元格,由于 4 个单元格在编码上不连续,不能一起读入高速缓存,在执行查询时,导致高速缓存缺失增加,影响查询效率;对于 Z-Curve 编码方法,查询所涉及的单元格为 0000,0001,0010,0011.4 个单元格在划分上连续,单元格对应的数据可以一起读入高速缓存,在执行查询时,减少高速缓存缺失,从而提高查询效率.

当多个查询到来时,Z-Curve 编码的空间局部性优势将不存在.例如:同样对于图 1 所示的 Grid 结构,假设在单元格 1100 中存在 2 个范围查询 Q_1 和 Q_2,Q_1 查询涉及右上角的 4 个单元格,即 1100,1101,1110,1111;Q_2 查询涉及单元格 1100,0110,0011,1001.执行 Q_1 查询时,把涉及到的 4 个单元格中的数据读入高速缓存进行查询;当执行 Q_2 查询时,除了单元格 1100 中的数据外,另外 3 个单元格中的数据都不在高速缓存中,将导致高速缓存缺失,影响查询效率.对于常规划分方法,也存在同样的问题.

通过上面的分析,当多个查询并行执行时,两种单元格编码方式没有明显的性能差异.相对于 Z-Curve 编码方式,常规编码方法运算量少,使用简单.在本文,我们采用了常规方法对单元格进行编码.



(a) 常规编码方法 (b) Z-Curve编码方法

Fig.1 Cell coding methods

图 1 单元格编码方法

3 双向流连接算法

3.1 双向流连接算法概述

双向流连接算法通过利用当前大内存和多核等新硬件特性,解决了当查询数据流和移动对象更新流大量涌入时,单个查询实时响应和系统吞吐量最大化问题.双向流连接算法在执行过程中采用了基于快照的方式,在每个快照中,基于对象的更新列表,重新构建一个新的索引结构,避免了传统方式中对索引的复杂更新操作,充分发挥了硬件的性能.

算法 1. Dual Stream Join (DSJ).

Input: *OT*, Object Table; *QT*, Query Table; *TF*, Transform Function; *GO*, Grid for Object; *GQ*, Grid for Query;

Output: *R*, Results.

```
1   R=NULL
2   For each object in OT
3     GO.cell=TF(object)
4     copy object to GO.cell
5   For each query in QT
6     GQ.cell=TF(query)
7     copy query to each of the GQ.cell that it intersected
8   For cell in GO and GQ with the same number
9     R=join(GO.object,GQ.query)
10  return R;
```

双向流连接算法分为 3 个阶段:第 1 阶段为索引移动对象阶段,创建一个 Grid 索引结构,把对象表中的移动对象通过转换函数用 Grid 结构进行索引;第 2 阶段为索引查询阶段,用 Grid 结构索引所有的查询;经过前两个阶段,移动对象和查询分别按照所在的单元格进行聚集.在对应的单元格中,通过把移动对象和查询执行连接操作,得到查询结果.

图 2 为查询示例:

- 首先索引移动对象,对移动对象进行聚集操作.如图 2 所示,对象 O_1 和 O_k 都位于格网索引的单元格(2,2)中,但在对象表中位置不相邻,通过转换函数,把两个对象聚集在 Grid 索引的相邻位置;
- 然后处理查询,对查询表中的查询,通过转换函数在 Grid 索引中聚集.当查询涉及到多个单元格时,在查询所涉及的每个单元格中保存一个副本,避免了执行过程中跨单元格访问导致的高速缓存缺失.

通过上面的两步操作,完成对对象和查询的聚集,提高了数据的空间局部性,从而提高了高速缓存命中率.例如:当单元格(2,2)中的对象和查询执行 join 操作时,对应的移动对象和查询都位于对象表和查询表的相邻位置,可以一次读入高速缓存中,从而提高了高速缓存的命中率,进而提高了算法的效率.

双向流连接算法通过一次处理多个查询的方式实现了查询间的并行;通过先对查询聚集然后再执行连接的方式,实现了查询内的并行;对查询执行聚集操作,充分增加查询的空间局部性,减少了执行过程中的高速缓存缺失;对对应单元格中的对象和查询执行连接操作,能够快速得到查询结果.通过使用快照内重建索引的方式,充分发挥了内存大和多核的性能.

通过设计合理的内存存取模式和数据布局方式,使算法在执行过程中具有很好的数据空间局部性和高速缓存命中率,从而提高了算法的执行效率和系统的吞吐量.

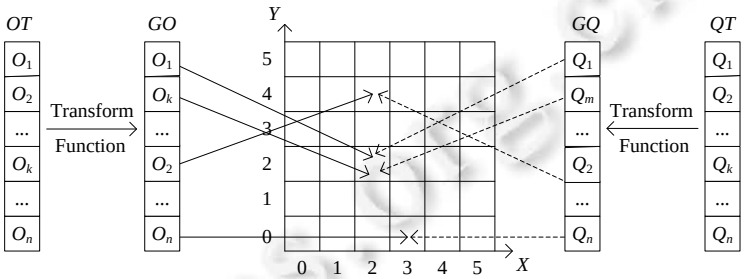


Fig.2 DSJ example

图 2 双向流连接算法示例

3.2 索引移动对象阶段

本文所设计的算法为内存算法,在内存中,频繁地使用指针结构将导致大量的高速缓存缺失.在我们的算法中,为了减少高速缓存的缺失,对移动对象采用了直接复制的方式,消除了指针结构.在新硬件的环境下,大内存已经成为主流的趋势,当前,在单个服务器中已经实现了 4TB 内存的配置.在我们的算法中,移动对象主要存储为三元组(OID,$\langle X,Y\rangle$),当把所有的移动对象放入内存后,对象在内存中所占容积达不到 TB 数量级,因此内存的容积不会是一个瓶颈.

在我们的算法中,Grid 中的单元格采用常规编码方式编码,移动对象到单元格的转换函数主要与单元格的长度 L 和 X 轴方向单元格的数量 M 相关,即,$cell=X/L+Y/L\times M$.每个移动对象通过其位置坐标(X,Y),可以计算出其所在的单元格.在 Grid 结构中,每个单元格为一个连续的数组结构.移动对象按照坐标信息找到对应单元格后,移动对象被存储在单元格所在的数组中.如图 3 所示,在对象表中有两个移动对象 $O_1(1,\langle 35,13\rangle)$和 $O_2(2,\langle 12,21\rangle)$.对于移动对象 O_1,通过转换函数 $X/10+Y/10\times 4$ 计算 35/10+13/10×4=7,即,O_1 将会存在单元格 7 中.采用相同的转换函数,移动对象 O_2 被存储在单元格 9 中.

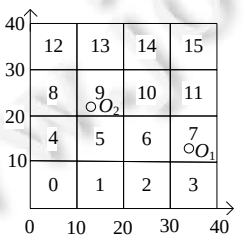


Fig.3 Index the moving object

图 3 索引移动对象

当对单元格进行编码后,移动对象按照位置信息通过转换函数放入对应的单元格中.通过转换函数,将移动对象由二维空间转换到一维空间,实现了降维.在当前的场景中,转换函数等价于哈希函数,每个单元格可以看作一个桶结构.即,对移动对象的索引可以理解为将移动对象通过一个哈希函数转换后放入对应的桶中.

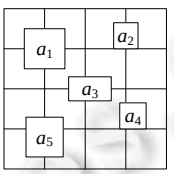
对于降维后移动对象的聚集操作,我们参考了文献[15]中的多核并行方案.执行过程中对数据进行两次顺

序扫描,采用数据分布直方图的方式,避免了使用锁机制,使操作可以充分地并行执行,消除了多线程执行过程中对象向单元格复制时的空间竞争问题,充分发挥了多核的性能.

3.3 索引查询阶段

本阶段主要把查询表中的查询映射到 Grid 结构中.在执行过程中,采用了与移动对象相同的转换函数和处理思路.

与移动对象不同,范围查询可能涉及多个单元格,在处理时,我们采用了文献[16]中的方法,对于查询涉及的每个单元格中都保留一个副本.通过保存副本的方法,消除了跨单元格访问造成的高速缓存缺失的问题,提高了数据的空间局部性.如图 4 所示,通过划分的方式,每个查询只需对比查询部分所覆盖的单元格中的移动对象,减少了查询对比的次数,提高了查询效率.



1	1	2	2
1	1,3	3	
5	3,5	3,4	4
5	5		

Fig.4 Partition (left) and assignment (right) for range query

图 4 范围查询的划分(左)和赋值(右)

在算法中,为了均衡单个查询的响应时间和系统的吞吐量,可以通过调整查询数量的方式实现:当关注于单个查询的响应时间时,通过减少查询的数量,以提高查询的响应时间;当关注于系统吞吐量时,通过增大查询数量,一次执行更多的查询,以提高系统的吞吐量.

3.4 连接执行阶段

通过前两步操作,查询和移动对象分别存放在对应的单元格中,各个单元格中的数据相互独立,可以多线程地并行执行.在执行过程中,采用 round robin 模式,每个线程处理一个单元格,对单元格中的对象和查询执行连接操作(join),最后把结果汇总.在单元格中连接运算,为计算密集型操作.

对于连接运算,我们首先采用了传统的算法——桶链连接(bucket-chaining-join)算法和嵌套循环连接(nested-loops-join)算法.桶链连接算法首先在文献[17]中被提出,主要思路为:把单个元组串联形成一个桶结构,然后将数组位置作为指针(而不是内存中实际的指针),通过这种方式提高内存算法的效率.桶链连接算法减少了连接执行过程中移动对象和查询的比较次数,但在构建桶时需要大量的运算.嵌套循环连接算法在执行过程中,需要把单元格中所有的查询和对象进行比较,大量额外的比较影响了算法的性能.

在二维空间中,判断一个移动对象(OID,$\langle X,Y\rangle$)是否为范围查询(QID,$\langle X_{\min},Y_{\min}\rangle$,$\langle X_{\max},Y_{\max}\rangle$)的结果,常用的方式为判断移动对象的坐标点是否大于等于范围查询最小值、小于等于范围查询最大值,即:

$$\text{if } ((X\geqslant X_{\min}) \,\&\&\, (X\leqslant X_{\max}) \,\&\&\, (Y\geqslant Y_{\min}) \,\&\&\, (Y\leqslant Y_{\max})) \tag{1}$$

但语句(1)在执行过程中由于逻辑与操作的特性,当某个表达式结果为 false 时,后续的表达式将不会继续执行.在这种情况下,会导致分支预测失败,致使性能大幅下降.

我们结合 SIMD 指令集合、数据预取机制和列存储运算的方式,提出了一种 SPC 移动对象范围查询判断运算方法.

算法 2. SPC.

Input: *OT*, Object Table, *QT*, Query Table;

Output: R_1, Results.

1 $Q\_x_{\min}$=*_mm_set_epi*32($Q_1X_{\min}$,$Q_2X_{\min}$,$Q_3X_{\min}$,$Q_4X_{\min}$)

2 $Q\_x_{\max}$=*_mm_set_epi*32($Q_1X_{\max}$,$Q_2X_{\max}$,$Q_3X_{\max}$,$Q_4X_{\max}$)

3 $Q\_y_{\min}$=*_mm_set_epi*32($Q_1Y_{\min}$,$Q_2Y_{\min}$,$Q_3Y_{\min}$,$Q_4Y_{\min}$)

4 $Q\_y_{\max}$=*_mm_set_epi*32($Q_1Y_{\max}$,$Q_2Y_{\max}$,$Q_3Y_{\max}$,$Q_4Y_{\max}$)

5 *O_x*=*_mm_set_epi*32(O_1X,O_1X,O_1X,O_1X)

6 *O_y*=*_mm_set_epi*32(O_1Y,O_1Y,O_1Y,O_1Y)

7 R_1=*_mm_sub_epi*32(*O_x*,$Q\_X_{\min}$)

8 R_2=*_mm_sub_epi*32(*O_y*,$Q\_Y_{\min}$)

9 R_3=*_mm_sub_epi*32($Q\_X_{\max}$,*O_x*)

10 R_4=*_mm_sub_epi*32($Q\_Y_{\max}$,*O_y*)

11 R_1=*_mm_and_si*128(R_1,R_2);

12 R_3=*_mm_and_si*128(R_3,R_4);

13 R_1=*_mm_and_si*128(R_1,R_3);

14 Return R_1

图 5 为 SPC 移动对象范围查询判断运算示例.

- 通过使用 SIMD 指令集,允许一定程度的并行,使多个操作同时执行;
- 使用 SIMD 指令时没有分支预测指令,消除了由于分支预测失败导致的性能下降.

如图 5 所示:在执行时,每次读取 4 个查询与一个对象进行比较,对应代码中的第 7 行~第 10 行操作.

- 从 4 个查询中,将查询的 $X_{\min}$ 值与移动对象 *X* 值进行比较;
- 对于查询中另外的 3 个变量,也与对象中对应的值进行比较;
- 最后,把 4 组运算的结果进行分析,得到运算的最终结果.

通过上述方式,提高了范围查询的效率.

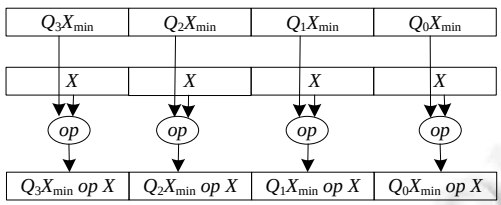


Fig.5 SPC calculation for moving objects range query

图 5 SPC 移动对象范围查询判断运算示例

对于嵌套循环连接算法,可以很好地和 SPC 运算结合,形成优化的嵌套循环连接算法;而对于桶链连接算法,由于算法在执行过程中,其数据通过一个桶结构进行存储,数据的存放不连续,数据预取机制在该算法中不起作用,不能与 SPC 运算相结合.

4 实验结果与分析

4.1 数据集

在实验中,我们使用了基于德国实际路网生成的数据集.实际数据集中包含了整个德国的路网,由 3.8 亿个节点和 4 亿个路段构成,覆盖 641km×864km 的面积.用开源的移动对象路径生成器 MOTO[2]生成数据.MOTO 是在数据生成器 Brinkhoff[18]的基础上形成的.MOTO 中采用了一个基于路网的对象布局方式,所有的移动对象都随机分布在一个给定的路网中,所设定的移动对象的最大车速 $S_{\max}$=60m/s=216km/h.数据生成器还根据现实中城市人口的分布进行了修改,一半的对象分布在德国的 5 个主要城市,因此能够确保更新最频繁的区域同时也是查询最多的区域.

图 6 为德国路网的数据分布图,移动对象用点表示,移动对象越集中的区域点越密集.通过图 6 可以发现,移

动对象明显集中在几个固定区域.

Fig.6 Data distribution of Germany road network

图 6 基于德国路网的数据分布图

- 实验设置

所有的算法用 C/C++实现,用 g++在最高的优化等级下进行编译.

实验运行在 32 核(4 Intel E5-2670@2.6GHz,物理核数为 16 核,采用了超线程技术)的计算机上,使用了 SUSEOS 11(64bit)系统,有 256G RAM,片上的内存被所有的线程共享(见表 1).

Table 1 Workload configuration

表 1 实际数据集参数

Parameter	Values
Objects ($\times 10^6$)	5, 10, 20, 40
Updates ($\times 10^6$)	100
Monitored region (km^2)	Germany, 641×864
Range query size (km^2)	0.25, 0.5, 1, 2, 4, 8
Update/Query ratio	250:1,…,1000:1,…,10000:1

4.2 双向流连接算法性能表现

本部分实验主要用于显示双向流连接算法的性能.若无特别说明,所有实验都在 32 核中完成.

在算法执行过程中,单元格划分粒度会对结果产生影响.本实验测试了对 500 万移动对象同时执行 500 万范围查询时,不同单元格划分粒度对查询响应时间的影响.图 7 中:横轴为单元格的划分粒度,单元格边长从 20m~60m;纵轴为查询的响应时间.当单元格边长很小时,会产生更多的单元格,在算法中每个线程处理一个单元格的数据,将导致每个线程处理的单元格数量增加.同时,对于查询,当单元格小的时候,会有更多的查询跨多个单元格,增加了查询的数量,进一步增加了查询的响应时间.随着单元格长度的增加,查询的响应时间逐渐减少.当单元格的长度为 40m 时,响应时间最快;当单元格的长度超过 40m 后,随着单元格边长的增加,在每个单元格中会包含更多的对象,将会导致单个单元格处理的时间延长,进而增加了查询的响应时间.参照当前的实验结果,在后面的实验中,我们设置单元格的边长为 40m.

为了充分发挥硬件的性能,在每个新的时间戳中,双向流连接算法都基于移动对象的更新重新构建一个新的 Grid 索引.本实验中测试了在不同移动对象数量下双向流连接算法创建 Grid 索引所用的时间.图 8 中:横坐标为需要索引移动对象的数量,从 500 万~4 000 万;纵轴为创建索引所用的时间.由图 8 可以看到:随着移动对象数量的增加,索引创建时间呈线性增长趋势.

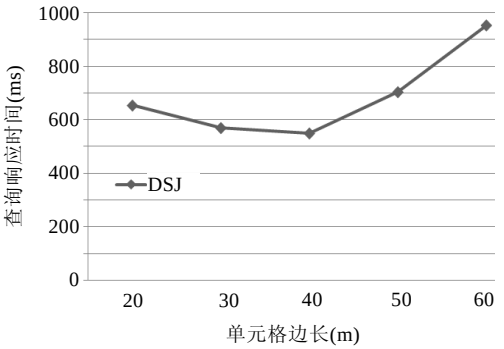


Fig.7 Query response time of varing grid size

图 7 不同单元格划分粒度下的查询响应时间

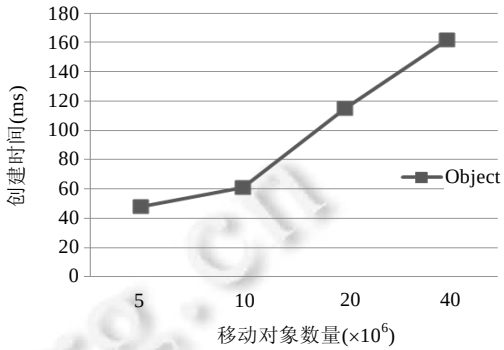


Fig.8 Index creation time of varing moving object

图 8 不同数量移动对象索引创建时间

当对移动对象构建完索引后,为了提高运算效率和系统的吞吐量,我们采用了每次执行多个查询的方式.在查询执行过程中,多个查询并行执行,增加了数据的局部性,提高了算法的效率.图 9 中显示不同数量的移动对象执行 500 万个查询时响应时间,横坐标表示移动对象的数量,纵坐标为查询的响应时间.由图 9 可以发现:随着数据量的增加,查询的响应时间呈线性增长.在不同移动对象的数据量下,所有查询的响应时间小于 2s,完全可以应对大部分移动场景下用户响应时间的要求.

图 10 显示了双向流连接算法对 500 万移动对象同时执行 500 万范围查询时,随线程数增加吞吐量的变化趋势.双向流连接算法在对对象和查询构建索引时,首先通过转换函数实现了降维,对于降维后的数据参考了文献[15]中的多核并行方案,对对象和查询按照单元格进行聚集.算法在执行过程中,为了减少高速缓存的缺失,对于对象和查询采用了直接复制的方式,消除了指针结构.对于查询,在每个查询所在单元格都保存一个副本.当执行连接操作时,通过 round robin 方式,每个线程处理一个单元格中的数据,各个线程独立执行,避免了线程间的竞争.通过图 10 可以看到:随线程数的增加,双向流连接算法的吞吐量呈线性增长;当超过系统的物理核数 16 时,算法的吞吐量仍缓慢增长.双向流连接算法具有很好的多核扩展性.

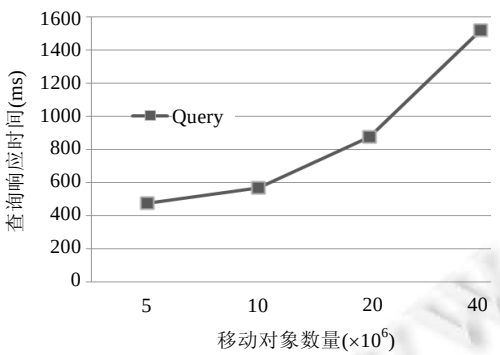


Fig.9 Query response time of varing moving object

图 9 不同移动对象数量下的查询响应时间

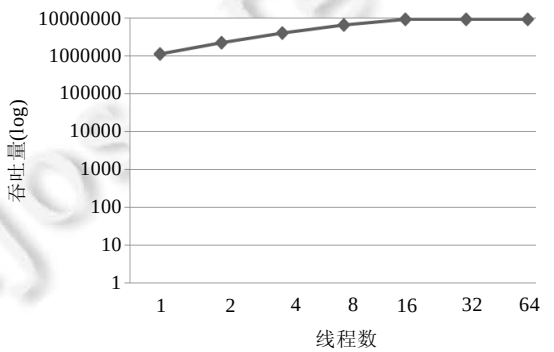


Fig.10 Performance of multi-core scalability

图 10 多核扩展性

图 11 显示了 SPC 移动对象判断策略相对于传统判断策略(如公式(1)所示)的优势.通过对比优化的嵌套循环连接算法和嵌套循环连接算法可以发现,采用 SPC 移动对象判断策略的优化的嵌套循环连接算法更有优势.在优化的嵌套循环连接算法中,执行运算的数据是连续存储的,减少了高速缓存缺失,同时可以充分发挥硬件预取机制这一特性.在运算中,SIMD 指令集结合列存储的运算方式,每次 4 个查询并行执行,提高了运算的效率.同时,通过 SIMD 方式使运算顺序执行,避免了分支预测失败现象的发生.对于嵌套循环连接算法,在执行过程中采用了公式(1)的方式,对查询和对象顺序进行判断,执行过程中,由于指令预测失败导致性能下降.对于

bucket-chaining-join 算法,运行过程中,前期索引构建耗费了大量时间,同时数据不连续,需要通过索引获取.在这种方式中,增加了高速缓存缺失,导致性能的下降.

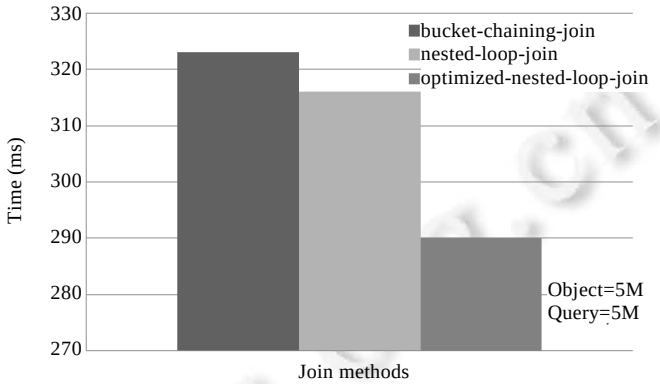


Fig.11 Performance of different join algorithms

图 11 不同连接策略下的性能表现

4.3 对比实验

下面,我们对双向流连接算法、PGrid 算法和 TwinGrid 算法的性能表现进行对比.

双向流连接算法主要用于应对大量查询涌入时,通过一次执行多个查询的方式以提高系统的吞吐量.图 12 中对比了在 500 万个移动对象下,查询从 500 到 500 万时,双向流连接算法、PGrid 算法和 TwinGrid 算法的性能表现.通过图 12 可以看出:当查询数量少时,TwinGrid 算法的查询响应时间最长.在 TwinGrid 算法中,查询和更新分别在两个 Grid 结构中,每隔一定的时间间隔,需要更新快照中的数据,把更新 Grid 中的数据复制到查询 Grid 中,复制操作占用了大量的时间.随着查询数量的增加,PGrid 算法的查询性能逐渐下降.PGrid 算法中,查询和更新在同一个 Grid 结构中,为了避免发生冲突,查询执行过程中采用了加锁机制,增加了查询的处理时间.对于 TwinGrid 算法,在执行过程中,更新和查询分别在两个 Grid 结构中,不存在冲突的问题.当查询数量增加时,其性能优势逐渐明显.对于双向流连接算法,性能表现是最优的;随着查询数量的增加,优势更加明显.双向流连接算法采用了每次执行一组查询的方法,在执行过程中,通过索引查询的方式增加了数据的局部性,实现了查询的并行,提高了算法的效率.

图 13 中显示了更改查询范围后,3 种算法的性能表现.图中横坐标为查询范围的变化,从 250m 到 8 000m;纵轴为系统的吞吐量.为了更好地显示吞吐量的变化趋势,对于纵坐标取对数形式表示.随着查询范围的扩大,需要涉及到更多的单元格和对象,对索引的性能会产生一定的影响.从图 13 可以看到:TwinGrid 算法和 PGrid 算法随着查询范围的扩大,吞吐量保持平稳.双向流连接算法随着查询范围的扩大,需要进行更多的比较,致使性能有所下降,但吞吐量仍比 TwinGrid 算法和 PGrid 算法高一个数量级.

图 14 中显示改变更新查询率,从 2000:1 到 1:4 时,PGrid 算法、TwinGrid 算法和双向流连接算法在执行过程中吞吐量的变化情况.随着查询数量的不断增加,需要进行更多的运算,3 种算法的吞吐量都会下降.通过图 14 可以看到:当查询增加时,PGrid 算法的吞吐量下降得更快.PGrid 算法中采用锁机制来保持数据的一致性,随着查询数量的增加,竞争更加激烈,导致性能有所下降.TwinGrid 算法在执行过程中,查询和更新分别在两个结构中,执行过程中不存在冲突,提高了查询的效率.双向流连接算法采用每次处理多个查询的方式,充分利用数据的局部性,提高了算法效率.与 TwinGrid 算法类似,双向流连接算法中对于更新也通过一个 buffer 进行存储,消除了查询和更新过程中的冲突.因此,在大数据量下,当查询和更新大量涌入时,双向流连接算法能够保持较高的系统吞吐量.

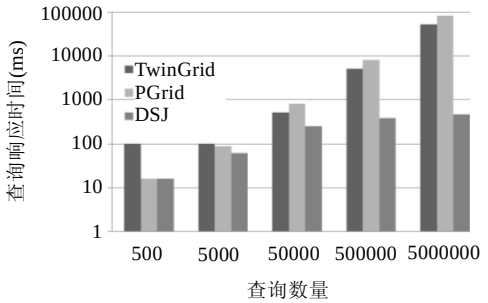


Fig.12 Performance of change queries

图 12 更改查询数量的性能表现

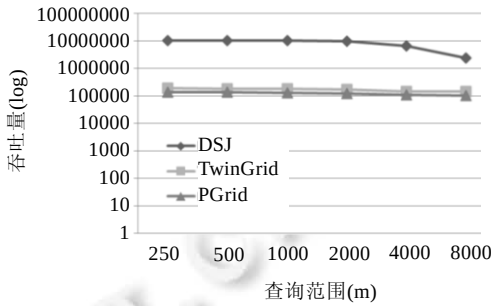


Fig.13 Performance of change query range size

图 13 更改查询范围的性能表现

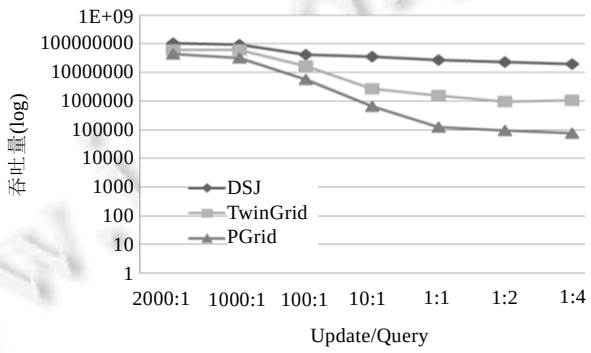


Fig.14 Performance of change query update rate

图 14 改变查询更新比率的性能表现

5 结束语

随着手机、平板电脑等设备的普及及网络通信技术的快速发展,基于位置的服务(location based service)在我们的生活中得到了广泛的普及和应用.国内外学者对基于位置的服务进行了大量的研究.已有研究中,主要关注移动对象的快速更新和查询的快速处理问题.在大数据环境下,当更新和查询大量涌入时,系统的吞吐量是更值得关注的问题.本文中,我们研究高吞吐率的移动对象范围查询算法.针对移动对象的更新数据流和查询数据流,结合当今新硬件的特性,我们提出了一种新的基于内存的移动对象范围查询算法——双向流连接算法.双向流连接算法在每个时间戳中重新构建一个新的索引结构,避免了对传统复杂索引结构的维护,充分发挥了新硬件特性;通过每次执行一组查询的方式,增加了执行过程中数据的局部性,提高了算法的效率.同时,结合 SIMD 指令集,提出了一个 SPC 移动对象范围判断运算方式,提高了运算过程中连接执行的效率.由以上 3 个方面可以看出,双向流连接算法提升了系统的吞吐量.实验结果也表明:双向流连接算法对于查询和更新快速涌入时能够很好地处理查询的响应时间,同时还能确保整个系统具有很高的吞吐量.

References:

[1] Zhou AY, Yang B, Jin CQ, Ma Q. Location-Based services: Architecture and progress. Chinese Journal of Computers, 2011,34(7):1155−1171 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01155]

[2] Dittrich J, Blunschi L, Salles MAV. Indexing moving objects using short-lived throwaway indexes. In: Proc. of the Advances in Spatial and Temporal Databases. Berlin, Heidelberg: Springer-Verlag, 2009. 189−207. [doi: 10.1007/978-3-642-02982-0_14]

[3] Yu XH, Pu KQ, Koudas N. Monitoring *k*-nearest neighbor queries over moving objects. In: Proc. of the 21st Int'l Conf. on Data Engineering. Toronto: IEEE, 2005. 631−642. [doi: 10.1109/ICDE.2005.92]

[4] Dittrich J, Blunschi L, Salles MAV. MOVIES: Indexing moving objects by shooting index images. Geo Informatica, 2011,15(4): 727−767. [doi: 10.1007/s10707-011-0122-y]

[5] Šidlauskas D, Ross KA, Jensen CS, Šaltenis S. Thread-Level parallel indexing of update intensive moving-object workloads. In: Proc. of the Advances in Spatial and Temporal Databases. Berlin, Heidelberg: Springer-Verlag, 2011. 186−204. [doi: 10.1007/978-3-642-22922-0_12]

[6] Šidlauskas D, Šaltenis S, Jensen CS. Parallel main-memory indexing for moving-object query and update workloads. In: Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM Press, 2012. 37−48. [doi: 10.1145/2213836.2213842]

[7] Zhang J, Zhu M, Papadias D, Tao Y, Lee DL. Location-Based spatial queries. In: Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 443−454. [doi: 10.1145/872757.872812]

[8] To QC, Dang TK, Kung J. OST-Tree: An access method for obfuscating spatio-temporal data in location based services. In: Proc. of the 4th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). Ho Chi Minh City: IEEE, 2011. 1−5. [doi: 10.1109/NTMS.2011.5720620]

[9] Tao Y, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In: Proc. of the 29th Int'l Conf. on Very Large Data Bases. Berlin: VLDB Endowment, 2003. 790−801.

[10] Harizopoulos S, Liang V, Abadi DJ, Madden S. Performance tradeoffs in read-optimized databases. In: Proc. of the 32nd Int'l Conf. on Very Large Data Bases. Seoul: VLDB Endowment, 2006. 487−498. [doi: 10.3724/SP.J.1016.2011.01155]

[11] Orenstein JA, Merrett TH. A class of data structures for associative searching. In: Proc. of the 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems. Boston: ACM Press, 1984. 181−190. [doi: 10.1145/588011.588037]

[12] Tropf H, Herzog H. Multidimensional range search in dynamically balanced trees. Angewandte Info, 1981,(2):71−77.

[13] Hilbert D. Ueber die stetige abbildung einer line auf ein flächenstück. Mathematische Annalen, 1891,8(3):459−460. [doi: 10.1007/978-3-662-25726-5_1]

[14] Mokbel MF, Xiong X, Aref WG. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In: Proc. of the 2004 ACM SIGMOD Int'l Conf. on Management of Data. Paris: ACM Press, 2004. 623−634. [doi: 10.1145/1007568.1007638]

[15] Balkesen C, Alonso G, Teubner J, Tamer özsu M. Main-Memory Hash joins on modern processor architectures. IEEE Trans. on Knowledge and Data Engineering (TKDE), 2015. [doi: 10.1109/TKDE.2014.2313874]

[16] Patel JM, De Witt DJ. Partition based spatial-merge join. ACM SIGMOD Record, 1996,25(2):259−270. [doi: 10.1145/235968.233338]

[17] Manegold S, Boncz P, Kersten M. Optimizing main-memory join on modern hardware. IEEE Trans. on Knowledge and Data Engineering, 2002,14(4):709−730. [doi: 10.1109/TKDE.2002.1019210]

[18] Brinkhoff T. A framework for generating network-based moving objects. Geo Informatica, 2002,6(2):153−180. [doi: 10.1023/A:1015231126594]

附中文参考文献:

[1] 周傲英,杨彬,金澈清,马强.基于位置的服务:架构与进展.计算机学报,2011,34(7):1155−1171. [doi: 10.3724/SP.J.1016.2011.01155]

薛忠斌(1985－),男,山东淄博人,博士,主要研究领域为内存数据库,时空数据库.

周烜(1979－),男,博士,副教授,主要研究领域为数据库.

王珊(1944－),女,教授,博士生导师,CCF会士,主要研究领域为高性能数据库新技术,内存数据库新技术,Video 数据库技术,数据仓库,大数据管理与分析技术.