软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(8):1805−1815 [doi: 10.3724/SP.J.1001.2011.03904]

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

面向多核多线程的移动对象连续 *K* 近邻查询*

赵 亮[1,2+], 景 宁[1], 陈 荦[1], 廖 巍[3], 钟志农[1]

[1](国防科学技术大学 电子科学与工程学院,湖南 长沙 410073)

[2](空军装备研究院 通信所,北京 100085)

[3](海军工程大学 电子工程学院,湖北 武汉 430033)

Continuous *K* Nearest Neighbor Queries over Moving Objects Based on Multi-Core and Multi-Threading

ZHAO Liang[1,2+], JING Ning[1], CHEN Luo[1], LIAO Wei[3], ZHONG Zhi-Nong[1]

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

[2](Institute of Communication, Equipment Academy of Air Force, Beijing 100085, China)

[3](College of Electronic Engineering, Navy University of Engineering, Wuhan 430033, China)

+ Corresponding author: E-mail: liangzhao2010@gmail.com

Zhao L, Jing N, Chen L, Liao W, Zhong ZN. Continuous *K* nearest neighbor queries over moving objects based on multi-core and multi-threading. *Journal of Software*, 2011,22(8):1805−1815. http://www.jos.org.cn/1000-9825/3904.htm

Abstract: To solve the problem of multiple continuous *K* nearest neighbor (KNN) queries over moving objects, considering the development of multi-core and multi-threading technologies, a two-stage framework is proposed for Multi-Threading Processing of Multiple Continuous KNN Queries (MPMCQ). This includes a preprocessing stage and a query execution stage to carry out the data updating task and the query execution task separately. In each of the stages, techniques are designed to optimize the cache access hit ratio and improve the parallelism through multi-threading. A query grouping technique in the query execution stage is proposed to improve the data temporal locality when accessing the memory. Thus, the cache hit ratio can be guaranteed. A KNN query algorithm is given based on the MPMCQ framework and the grid index for moving objects. Extensive experiments are carried out to verify that by adopting the multi-threading and the cache optimization technologies, the proposed framework implements a much superior performance than other famous algorithms; moreover, it maintains excellent performance scalability when executed under different multi-core CPUs.

Key words: moving object; continuous KNN query; multi-core and multi-threading; cache optimization; query grouping

摘 要: 针对移动对象的多用户连续 *K* 近邻查询处理问题,结合多核多线程技术的发展,提出了一种基于多线程的两阶段多用户连续 *K* 近邻查询处理框架.将查询处理分为查询预处理阶段和查询执行阶段,分别执行数据更新任

* 基金项目: 国家自然科学基金(40801160, 60902036); 国家高技术研究发展计划(863)(2008AA12A211); 中国博士后科学基金(20080431384)

收稿时间: 2009-08-17; 修改时间: 2010-03-04; 定稿时间: 2010-07-06

务和查询处理任务.每个阶段都设计了优化 cache 访问命中率,并利用多线程技术提高多用户连续查询处理并行性的方法及数据结构.提出了一种查询执行阶段的查询分组技术,利用查询之间的相关性提高了算法执行时内存访问的时间局部性.基于查询处理框架和移动对象内存格网索引结构提出了 *K* 近邻查询处理算法.充分的实验结果表明,采用了多线程和 cache 优化技术的连续查询处理框架与其他算法相比,在性能上具有较大优势,并且在不同核心数目的 CPU 平台下具有较好的性能扩展性.

关键词: 移动对象;连续 *K* 近邻查询;多核多线程;cache 优化;查询分组

中图法分类号: TP311 **文献标识码:** A

随着无线通信、计算技术、GPS 空间定位等技术的快速发展以及众多具有定位功能的无线手持和车载设备的大量普及,在许多应用,如交通调度、救援服务及位置服务等领域,往往需要对大量的空间移动对象进行监控和管理.在移动对象管理技术研究领域,当前的研究热点逐步从最初的索引技术、单查询处理技术转移到多用户查询上,其中,对多用户连续 *K* 近邻(*K* nearest neighbor,简称 KNN)查询技术的研究尤为突出[1−6].

连续 *K* 近邻查询是指从提交查询时刻开始,不断地给出随着查询位置或者移动对象位置信息变化的 *K* 近邻查询结果.多用户连续 *K* 近邻查询则具有以下特点[3]:需要处理多个长时间运行的查询;需要频繁计算并保持最新的查询结果;通常基于内存来处理大量的移动对象或查询的更新;目标是最小化 CPU 的执行时间或客户端服务器之间的通信代价,而不是单查询中通常的优化目标磁盘 I/O 代价.根据算法优化目标的不同,可以将多用户连续 *K* 近邻查询的研究成果分为两类:(1) 以优化算法的 CPU 执行时间为目标:Xiong 等人提出了 SEA-CNN 算法[1],通过引入查询搜索区域并利用共享查询执行思想进行多用户连续 *K* 近邻查询批更新处理;Yu 等人提出的 YPK-CNN 算法[2]采用移动对象层次格网索引和查询索引以提高连续 *K* 近邻查询更新性能;Mouratidis 等人提出的 CPM 算法[3]利用概念空间划分技术以支持多用户连续 *K* 近邻查询处理;(2) 以优化客户端和服务器通信代价为目标:Hu 等人[4]首次引入了通信代价这一优化目标,并详细讨论了计算移动对象安全区域的算法以减少位置信息的更新;Hseuh 等人[5]进一步假设客户端具有一定的计算能力,通过维护位置信息表来减少更新.除了以上研究之外,文献[6]研究了道路网中的移动对象多用户 *K* 近邻查询问题,通过利用空间网络的相关属性和移动对象运动受限这一性质,减少连续查询的重复计算.总的来说,上述多用户连续 *K* 近邻查询处理技术多采用基于内存的查询处理框架,利用格网索引对移动对象进行存储和管理,通过研究不同的查询搜索策略减少对移动对象集的重复访问,并采用单线程的共享查询执行机制,提高多用户连续 *K* 近邻查询处理的整体性能.

目前,随着内存与处理器中 cache 之间的速度差距越来越大,处理器等待 cache 从内存读取数据造成的延迟正成为数据库系统的主要瓶颈,尤其在多核处理器的多级存储体系结构下如何提高处理器中最下一级 cache 的数据缓存命中率,已经成为高性能数据库领域的研究热点[7].另外,传统单线程查询执行模式难以充分利用多核处理器的并行计算优势,因此,研究面向多核多线程的多用户连续 *K* 近邻查询处理框架和算法已成为目前多核计算环境下的必然趋势.

本文研究如何将多核多线程技术引入移动对象多用户连续 *K* 近邻查询处理问题,希望从以下两个方面提高连续 *K* 近邻查询处理性能:(1) 使用多线程并行处理技术;(2) 设计相关方法提高 cache 访问命中率.为此,重点提出了一个基于多线程的多用户连续 *K* 近邻查询处理框架,设计了查询分组技术提高 cache 访问命中率,并提出了 *K* 近邻查询处理算法.

本文第 1 节~第 3 节分别对应于上述关键技术.第 4 节为实验分析.第 5 节进行总结与展望.

1 基于多线程的多用户连续 *K* 近邻查询的处理框架(multi-threading processing of multiple continuous queries,简称 MPMCQ)

已有的多用户连续 *K* 近邻查询处理算法通常在内存中对移动对象进行格网索引,周期性地更新移动对象数据和查询的位置信息,并周期性地给出查询计算结果,通过维护每个查询的影响区域和重用中间结果来减少周期性计算中访问格网索引中单元格的次数,既没有考虑到在多核体系结构下如何使用多线程来提高算法的

性能,也没有考虑如何通过提高内存中数据的空间局部性和时间局部性,进而改善 cache 访问命中率.

基于上述认识,本文提出了一种基于多线程的多用户连续查询处理框架.与已有的移动对象连续查询处理技术相比,该框架采用多线程技术周期性地重复计算所有查询结果,力求利用多核处理器的并行处理能力提高连续查询处理的性能.如图 1 所示,MPMCQ 框架将多用户连续查询处理分解为查询预处理(preprocessing)和查询执行(execution)两个阶段,在每一阶段中同时考虑如何提高 cache 访问命中率和采用多线程提高多用户连续查询的并行性.只有当 Preprocessing 阶段处理完成后,才开始执行 Execution 阶段.

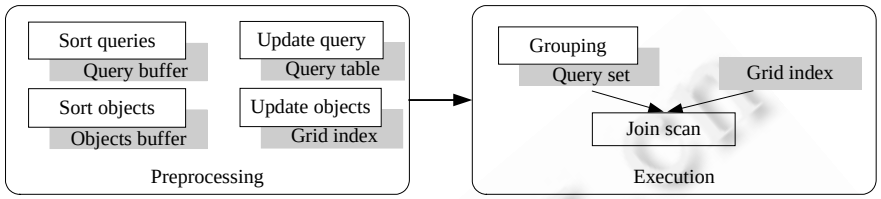


Fig.1 Framework of MPMCQ

图 1 MPMCQ 框架

下面对各个阶段包含的子处理和数据结构进行详细说明.

1.1 预处理阶段(preprocessing)

在移动对象连续查询处理问题中,移动对象位置信息和查询的位置信息都是不断变化的.因此,需要对这些信息在查询处理之前进行更新,保证结果的即时性和正确性.即预处理阶段的目的是在周期性的查询计算时刻首先更新移动对象数据和查询信息,为查询执行阶段准备数据.

图 2 为 MPMCQ 中预处理阶段的示意图,其中包含两个主要过程:更新查询表和更新移动对象格网索引.图 2 上半部分显示了更新查询表的过程,首先对两个查询计算时刻之间发生变化的查询信息进行排序,将排序结果放到查询缓存(query buffer)中;而后,线程调度器通过均匀划分所需处理的数据,调度多个更新线程,将查询更新到查询表中.移动对象更新的过程与之类似.

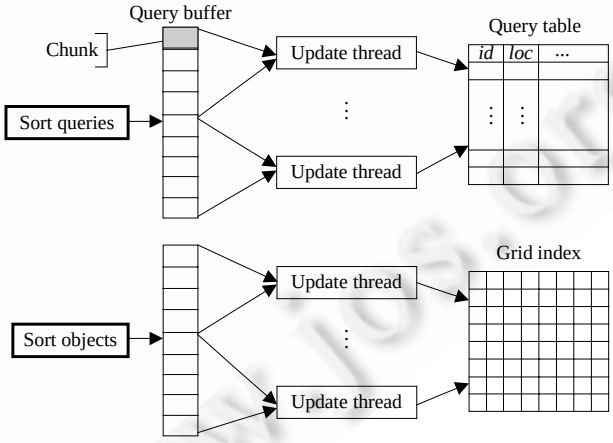


Fig.2 Detailed explanations to the preprocessing stage

图 2 预处理阶段详细说明

需要特殊说明的是:(1) 根据查询的空间位置,利用空间填充曲线[8]对其进行排序,使得空间上位置相邻的查询在 buffer 中存储时也保持相邻,目的是减少更新线程更新查询表和格网索引时的互斥访问,同时提高查询处理阶段多线程处理的 cache 访问命中率(第 2 节图 4 有详细解释);而对移动对象进行排序的目的是提高多线

程索引更新时访问索引数据的时间局部性(即如果某一存储位置的数据被访问,则其可能被再次访问)和空间局部性(即如果某一存储位置的数据被访问,则其相邻位置的数据也可能被访问).(2) 图 2 中的查询缓存(query buffer)采用文献[9]提出的 Parallel Buffer 进行内存访问和管理,整块内存分成若干个 Chunk,每个线程以 Chunk 为单位读/写内存,可以大大降低互斥访问代价.(3) 预处理阶段包含的数据结构主要有:

- 查询表(query table),基于内存的线性结构,用以存放用户提交的连续查询.记录形式为⟨*QID*,*Qloc*,*k*,*Range*,*InfReg*⟩.其中,*QID* 为连续查询 *q* 的唯一标识;*Qloc* 为当前更新时刻查询 *q* 的位置;*k* 为查询 *q* 的近邻个数;*Range* 为查询 *q* 的搜索半径;*InfReg* 为查询 *q* 的影响区域(即上一周期的查询结果所在的单元格),形式为格网单元格标识的集合.
- 格网索引(grid index),基于内存的格网结构.整个移动对象运动的空间划分为 $\delta\times\delta$ 个单元格,将移动对象根据其空间位置映射到单元格中.每一个单元格可表示为⟨*CellId*,*ObjectSet*⟩,其中,*CellId* 为单元格的标识,*ObjectSet* 为单元格中移动对象的集合;*ObjectSet* 中每一个移动对象表示为⟨*OID*,*Oloc*⟩,*OID* 为移动对象标识,*Oloc* 为移动对象当前的二维空间位置信息.
- 查询缓冲区(query buffer),基于内存的 Parallel Buffer 线性结构,用于缓存更新的连续查询.记录形式为⟨*OID*,(x_{old},y_{old}),(x_{new},y_{new})⟩,其中,*QID* 为连续查询 *q* 标识,(x_{old},y_{old})和(x_{new},y_{new})分别表示查询 *q* 更新前和更新后的位置.
- 移动对象缓冲区(object buffer),基于内存的 Parallel Buffer 线性结构,用于缓存更新的移动对象.记录形式为⟨*OID*,(x_{old},y_{old}),(x_{new},y_{new})⟩,其中,*OID* 为移动对象标识,(x_{old},y_{old})和(x_{new},y_{new})分别表示移动对象更新前和更新后的位置.

1.2 查询处理阶段(execution)

查询处理阶段的目的是在获得前一个阶段准备的数据后,在已经更新了的查询数据和移动对象格网索引间进行连接扫描(即调用 *K* 近邻查询处理算法),获得多查询的结果.

图 3 为 MPMCQ 中查询处理阶段的示意图,其中包含两个主要过程:对查询进行分组和调用查询处理算法.空间位置相邻的查询在调用查询处理算法时会扫描很多相同的移动对象格网索引中的单元格,将空间位置相邻的查询分组后,有利于提高查询算法调用时访问格网索引数据的时间局部性.分组后,线程调度器调用多个查询线程在格网索引和每个查询分组之间进行连接扫描,最终得出多用户查询的结果.查询处理阶段包含的数据结构主要有:查询工作集(query set),基于内存的 Parallel Buffer 线性结构,用于存放查询分组集合;每一个查询分组用一块 chunk 存储,其记录形式为⟨*CID*,(*QID*,*Qloc*,*k*,*InfReg*,*Results*,*Range*),…⟩,其中,*CID* 为该内存块 chunk 的标识;每个形为(*QID*,*Qloc*,*k*,*InfReg*,*Results*,*Range*)的元组存放了该工作集中的一个连续查询信息,*Results* 用以存放查询 *q* 的结果集,其他属性含义与查询表中查询的对应属性一致.

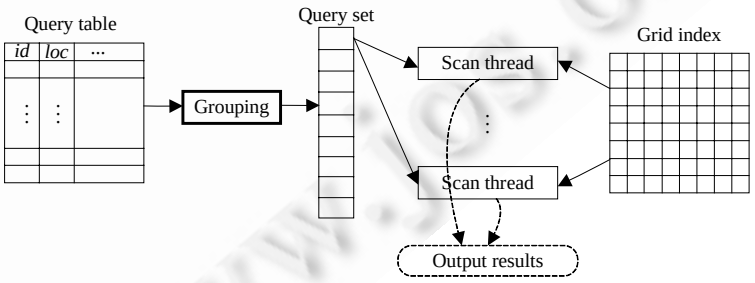


Fig.3 Detailed explanations to the query execution stage

图 3 查询执行阶段详细说明

2 查询分组技术

MPMCQ 框架在预处理阶段对查询利用空间填充曲线(hilbert curve)[8]进行了空间重排序,并将其存入查询

表中.Hilbert Curve 能够很好地保持查询位置的空间临近性,使得空间位置相邻的查询在存储时也是相邻的.如图 4 所示,如果按照查询更新的顺序,应该是从 q_1 到 q_n 的存储顺序,经过 Hilbert Curve 排序后,则按照空间位置从 $q_2,q_1,\ldots,q_n,q_k$ 的顺序存储.这种处理使得在调用查询处理算法时,算法遍历格网索引中的移动对象时能够保持较好的时间局部性.即处理 q_n 时访问的单元格中的移动对象在处理 q_k 时能够继续被访问,因而提高了 cache 访问命中率.

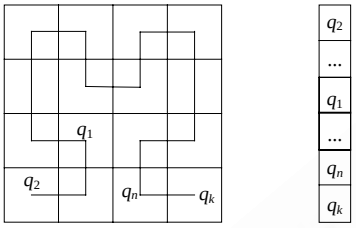


Fig.4 Sorting the queries by Hilbert curve

图 4 Hilbert 曲线对查询排序

但是,这种简单的处理并不能很好地利用查询之间的相关性,从而进一步地提高格网中移动对象访问时的时间局部性.如图 5 所示,给定连续 K 近邻查询 q 和 q',图 5(a)阴影区域所示为 q 在计算 K 近邻时所需要访问的 7 个格网单元;在同一时刻用户提交连续 K 近邻查询 q',如图 5(b)所示,斜线阴影区域为 q'的影响区域,即 q'需要搜索访问 5 个格网单元.q 和 q'的影响区域是部分重叠的,意味着处理 q 和 q'时很可能需要搜索同样的单元格.如果将具有重叠搜索区域较多的查询分为一组并以多线程处理时,重用数据非常多,cache 访问冲突较少,进而能够提高 cache 访问命中率.

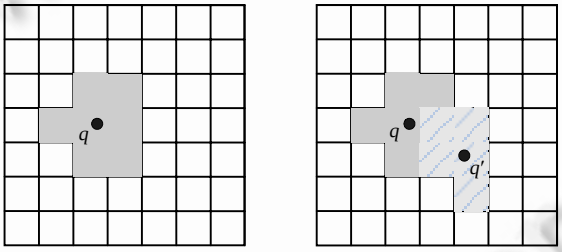


Fig.5 Influence region of CKNN query

图 5 连续 K 近邻查询影响区域

采用模式识别技术中经典的聚类算法——ISODATA 算法[10]对查询进行分组.与大多数聚类方法以欧式距离为相似性测度不同,在本文的查询分组中,定义查询相似性函数 $s(q_m,q_n)$为查询 q_m 和 q_n 的影响区域重叠部分的格网单元格数.具体来说,$s(q_m,q_n)$可表示为

$$s(q_m,q_n)=|q_m.InfReg\cap q_n.InfReg|.$$

接下来讨论算法中 7 个初始参数设置.表 1 列出了聚类算法中参数设置使用的符号及其含义.

(1) 预期的组个数 N_G、初始分组中心个数 N_c、每组中允许的最少查询个数 N_q

查询分组是为了提高访问移动对象索引时,对索引中移动对象数据的 cache 访问命中率.由于多核处理器一般共享最下一级高速缓存,最多将 C/T_o 个移动对象装入 cache 中.每个 K 近邻查询至少需要访问 K 个移动对象数据才能获取正确的查询结果.因此,每组中最大的查询个数为 $C/(T_o\times K)$.从而设置:

$N_G=Sum_q\times(T_o\times K)/C, N_c=N_G, N_q=0.5\times C/(T_o\times K)$.

Table 1 Symbols and their meanings used in the clustering algorithm

表 1 聚类算法中使用的符号及其含义

Symbols	Meanings
C	L2 cache size
T_o	Size of each moving object tuple
$\delta\times\delta$	Total cell numbers
Sum_q	Total query numbers

(2) 组内查询分布的标准差下限θ_s和两组中心间的相似性上限θ_D

对于第 j 个查询分组来说,其组内查询分布的标准差用下式进行计算:

$$\sigma_j=\left[\frac{1}{N_j}\sum_{j=1}^{N_j}S(q_{mj},q_j)\right]^{\frac{1}{2}},$$

其中,N_j表示第 j 个分组的查询个数,q_j表示其组心.σ_j的值表示组的聚类程度.标准差下限θ_s表示如果$\sigma_j<\theta_s$,则该组需要分裂为两组.

对第 i 和第 j 个分组来说,其组心间相似性用下式计算:

$$d_{ij}=S(q_i,q_j),$$

其值表示两组的相似性程度.两组中心间的相似性上限θ_D表示,如果 $d_{ij}>\theta_D$,则第 i 和 j 组需要合并为一组.

假设移动对象在运动空间中均匀分布,每个单元格平均包含 N_o/δ^2 个移动对象,则对每个 K 近邻查询来说,至少访问 $K\times\delta^2/N_o$ 个单元格才能获得正确结果.θ_s越小且θ_D越大,则分组速度越快,但分组效果较差.为此设置:

$$\theta_s=\sqrt{\frac{1}{8}\cdot\frac{K\cdot\delta^2}{N_o}},\ \theta_D=\frac{1}{4}\cdot\frac{K\cdot\delta^2}{N_o}.$$

(3) 每次迭代中可以合并的组的最多对数 L 和允许的最多迭代次数 I

在聚类算法中,这两个参数主要用来控制聚类算法的运行时间.当然,迭代次数越多分组效果越好.由于 ISODATA 算法本身计算较多,希望能够快速完成聚类.为此,设置最多对数 L 为 1;设置最大迭代次数 I 为 3.

通过实验发现,将迭代次数控制较低时,也即聚类算法耗时较少时,能够带来较好的框架执行性能.

3 *K* 近邻查询处理算法

MPMCQ 框架在查询处理阶段调用多个线程执行 K 近邻查询处理算法得到每个查询的结果.其主要思想是:针对预处理阶段生成的查询分组(query group)任务队列,算法从队列中拾取查询分组分配给线程池中的空闲工作线程进行处理,查询分组中的多个查询在多线程之间均匀分配,即每个线程处理一个查询分组的子集 QG_{sub}.

K 近邻查询处理算法的基础是首先对移动对象进行格网索引.假设移动对象的运动空间为单位正方形区域,将该区域划分为$\delta\times\delta$个单元格,则第 i 行第 j 列的单元格 c_{ij} 包含横坐标范围为$[i\cdot\delta,(i+1)\cdot\delta]$且纵坐标范围为$[j\cdot\delta,(j+1)\cdot\delta]$的移动对象.反之,坐标为$(x,y)$的移动对象存储在第 $i=\lfloor x/\delta\rfloor$行、第 $j=\lfloor y/\delta\rfloor$列的单元格中.表 2 对下面的描述中常用的符号和函数进行了说明.

Table 2 K 近邻查询算法中的符号和函数说明

表 2 Symbols and functions used in the KNN query algorithm

Symbols	Descriptions
$dist(p,q)$	The Euclidean distance between p and q
Results	q's result list
InfReg	q's influence region
Range	The distance between the kth nearest neighbor and q
$mindist(c,q)$	The minimal distance between cell c and q

令 *Results* 为当前查询 q 的 K 近邻结果集,*Range* 为当前第 K 个最近邻距 q 的距离.如果某一单元格 c 距 q 的距离 min*dist*(c,q)≥*Range*,则该单元格 c 肯定不包含查询 q 的 K 近邻,从而不用遍历 c 中的移动对象.因此,一种自然的算法是根据单元格 c 到查询 q 的最短距离,由小到大排序.对于序列中的单元格,遍历其中的移动对象,计算 *dist*(p,q)并更新 *Results* 和 *Range*,直到序列中某一单元格距 q 的距离大于 *Range* 为止.这种最直观的方法需要计算所有单元格与查询之间的最短距离并排序,效率很低.由于采用多线程来进行查询处理,因此没有像 CPM 算法那样考虑计算结果的重用减少查询的重复计算,而是对每个查询都周期性地重新计算查询结果.因此,设计了一种相对于 CPM 算法来说,具有简单数据结构的 K 近邻的算法.

基于移动对象格网索引的 K 近邻查询处理算法可用图 6 进行阐释.以计算查询 q 的 3 近邻为例,说明算法的运算过程.在遍历格网索引时,首先搜索查询 q 所在单元格,如果当前单元格不足 3 个移动对象,则搜索以所在单元格为中心的周围的其他 8 个单元格,此时,算法找到了 3 个最近邻;然后,在与以当前结果集中的最大距离为半径圆相交的单元格中继续搜索找到正确的第 3 个近邻,即图中 p_1,p_2,p_4 为实际的查询结果.

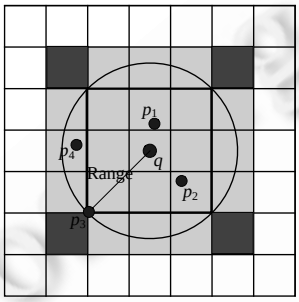


Fig.6 Explanations of the KNN query algorithm

图 6 K 近邻查询算法说明

在查询执行阶段,将多个查询均匀划分给多个线程执行.对每个查询来说,具体的算法如下:

Algorithm. KNN Query Algorithm.

Input: Query q_i, Index *GI*.

Output: q_i.*Results*.

1. *TempCellList*=∅
2. Search around the cell that q_i resides until more than k objects are found
3. Add the corresponding cells to *TempCellList*
4. q_i.*Range*=the longest distance from the objects in *TempCellList* to q_i
5. Add cells to *TempCellList* that are fully or partly overlapped with the circle centered at q_i with radius *Range*
6. Compute the real KNNs of q_i in *TempCellList* according to distance comparing, add them to the query result list q_i.*Results*, update q_i.*InfReg* according to the *TempCellList*

算法分 4 个步骤,对每一个查询 q_i,算法第 1 步需要找到围绕 q_i 的 K 个移动对象,并将这些对象所在单元格加入临时列表中(第 2 行、第 3 行);第 2 步计算临时列表中所有移动对象到查询 q_i 的最远距离(第 4 行);第 3 步将与以 q_i 为圆心、最远距离为半径圆相交的格网加入临时列表(第 5 行);最后,从临时列表中计算查询 q_i 的 K 近邻并更新查询的影响区域(第 6 行).

4 实验结果与分析

利用大量并发的连续 K 近邻查询来评估 MPMCQ 框架的性能.首先介绍实验采用的硬件环境:采用了 3 种不同的 CPU 平台:单核 CPU 为 Intel Celeron 3.2GHz、双核 CPU 为 Intel Pentium Dual E2200 2.2GHz, 1MB L2 高速缓存、四核 CPU 为 Intel Core 2 Quad Q8200, 4MB L2 高速缓存.内存均为 1GB.操作系统均采用 Windows XP

SP2.使用 Visual Studio 2005 开发环境,采用 C++语言实现了本文提出的算法.

使用随机生成的数据来进行性能测试.移动对象在运动空间为 1×1 的单位正方形内均匀分布.移动对象和查询数据的更新周期为 10s,两次查询执行的间隔为 30s.具体的参数设置见表 3.

Table 3 Experimental parameters

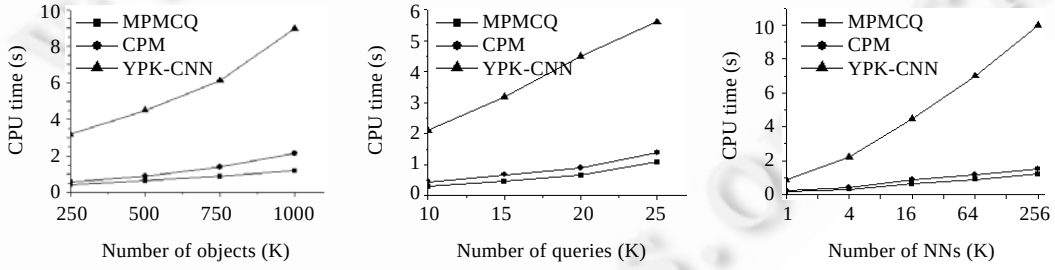
表 3 实验参数

Parameters	Default value	Optional values
Numbers of moving objects (*N*)	500K	250K, 500K, 750K, 1 000K
Numbers of queries (*Q*)	20K	10K, 15K, 20K, 25K
Value of *k*	16	1, 4, 16, 64, 256

4.1 MPMCQ框架与其他算法的比较

实验在双核 CPU 平台上比较 YPK-CNN,CPM,MPMCQ 这 3 种计算框架的性能.由于 3 种机制都是基于内存的算法,因此比较的性能指标采用执行一个处理周期(包括数据更新和查询处理)的 CPU 时间.其中,MPMCQ 框架中的多线程操作设置为两个线程,采用运行 10 个周期求平均的方法得出性能指标;YPK-CNN 和 CPM 算法均采用单线程处理方式.格网索引设置为 1000×1000 大小.

如图 7 所示为各种算法的计算时间随着移动对象数目、查询数目、近邻个数的变化而变化的情况.从图中可以看出,MPMCQ 框架和 CPM[3]算法在 3 种情况下的性能指标均优于 YPK-CNN[2]算法.这是由于 YPK-CNN 算法当移动对象更新后位于查询影响区域之内时立即对查询进行更新,极大地降低了查询更新性能.CPM 算法则优化了查询更新算法,在查询或移动对象更新时只需访问最少的格网.随着数据规模的增加,MPMCQ 框架中多线程和 cache 优化的性能优势逐步体现得比较明显.能够看出,MPMCQ 框架的性能明显优于 CPM 算法.这是因为相对于其执行代价来说,无论是排序优化还是查询分组优化都能带来更大的性能提升.事实上,CPM 算法的共享查询特性使其具有非常好的查询性能,且其性能随着数据量的变化并不会明显地恶化.而 MPMCQ 框架中的 cache 访问优化技术和多线程技术的使用,一方面提高了内存数据访问时的数据空间局部性和时间局部性,保证了 cache 访问命中率;另一方面,多线程的使用切实利用了多核 CPU 的并行计算能力.在如图 7 所示的 3 种不同情况下采用最大数据量时,MPMCQ 框架相对于 CPM 算法的性能提升分别为 48%,21%,17%.



(a) Effects of moving objects' amount to the performance
(a) 移动对象数目对性能的影响

(b) Effects of queries' amount to the performance
(b) 查询数目对性能的影响

(c) Effects of *k* value to the performance
(c) *K* 值对性能的影响

Fig.7 Effects of data amount to the query performance of different algorithms

图 7 数据量的变化对各种算法性能的影响

4.2 MPMCQ框架在不同处理器平台下的性能

本节测试 MPMCQ 框架在不同处理器平台下的性能.在 MPMCQ 框架中的移动对象更新、查询更新和 *K* 近邻查询处理 3 个操作设置为多线程执行,以上 3 个操作是一种顺序执行的关系,在框架中依次执行.同样,采用运行 10 个周期求平均的方法得出性能指标.

首先,使用默认参数测试不同处理器平台下,MPMCQ 框架的性能随着算法中线程数设置的不同而变化的

情况.如图 8 所示,在单核平台上,多线程执行并没有带来性能的提升.相反,随着线程数目的增加,算法的执行时间略微有所增长.这主要是因为单核平台上的多线程执行并不是真正意义上的并行执行,线程之间的频繁交互导致性能降低;在双核平台上,多线程执行在性能上具有比单线程明显的优势,当线程数大于 2(核的数目)且为奇数时,由于处理器核心之间负载不均衡,导致性能下降.但随着线程数的增加,单个线程数据量变小,负载不均衡的程度也随之降低,性能有所回升.但总的来说,在线程数等于核心数时,多线程执行性能最高;在四核平台上,得到了与双核平台类似的结果.

其次,使用默认的查询 16 近邻,测试在双核和四核平台下,分别设置 MPMCQ 框架中执行线程为 2 和 4 时,MPMCQ 框架性能随数据量的变化而变化的情况.如图 9 所示,在不同数据量的情况下,四核平台具有更好的查询性能,体现了 MPMCQ 框架较好的扩展性.

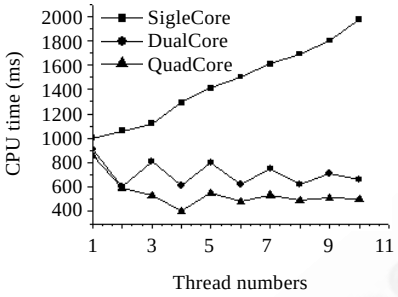


Fig.8 Effects of thread numbers to the performance

图 8 线程数对性能的影响

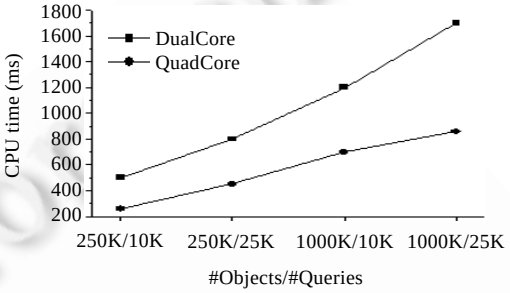


Fig.9 Effects of data amount to the performance under different platforms

图 9 不同平台下数据量对性能的影响

4.3 Cache访问性能优化

本节在双核平台上测试采用 cache 优化前后 MPMCQ 框架的执行性能,以验证 MPMCQ 框架中提出的排序优化和查询分组优化对提高 cache 命中率的作用及对整个框架执行性能的影响.采用 Intel VTune V8.0.014 版本记录 cache 缺失次数.

图 10 和图 11 分别显示了随着移动对象数据量的变化,采用 cache 优化前后框架的执行性能和平均每个查询的 cache 缺失次数.在实验中仍使用多线程进行更新和查询处理操作.从结果来看,采用 cache 优化前后的性能平均提高了 23.3%,且能够明显降低 cache 缺失次数.

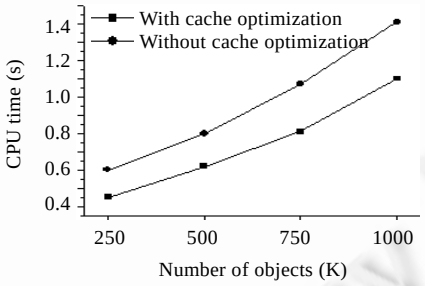


Fig.10 Performance before and after cache optimization

图 10 采用 cache 优化前后框架的性能

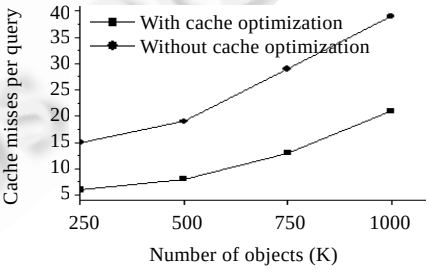


Fig.11 Cache misses per query

图 11 平均每个查询的 cache 缺失

4.4 聚类参数对框架性能的影响

第 2 节通过分析给出了聚类算法中除了每次迭代中可以合并的组的最多对数 L 和允许的最多迭代次数 I 这两个参数以外,其他 5 个参数的计算公式.本节测试 L 和 I 对框架性能的影响.

使用的默认数据量为 50 万个移动对象和 2 万个查询.其他参数是:L2 cache 为 1MB,一个移动对象元组为 12 字节.此时,根据计算公式可计算出预期组个数和初始分组中心数等于 4,每组最少的查询个数为 2 730.而组内查询分布的标准差下限和两组中心间的相似性上限分别为 2 和 8.

图 12 显示了在 L 和 I 变化情况下,框架性能的变化情况.可以看出,在 L 不变的情况下,迭代次数为 1 和 3 时性能相差不多;随着迭代次数的增加,聚类算法耗时快速增长,相对于其能够带来的 cache 优化效果来说,其优化作用越来越不明显.而在 I 不变的情况下,随着 L 的增加,性能略有下降.这是因为合并次数越多最终分组的数目可能就越少,cache 优化的效果就越不明显.最终,在前面几节的实验中选择将 L 和 I 分别设置为 1 和 3.

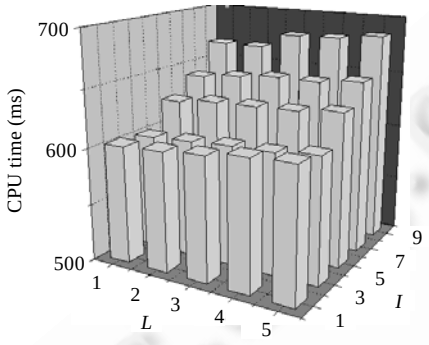


Fig.12 Effects of cluster parameters to the performance

图 12 聚类参数对性能的影响

5 总结与展望

面对多核体系结构给数据库领域带来的机遇和挑战,本文将多核多线程处理技术引入移动对象的查询处理,提出了针对多用户连续 K 近邻查询处理问题的多线程解决框架,设计了各种方法优化查询处理中的 cache 访问命中率,并以多线程的方式提高查询处理的并行性.实验结果表明,该框架的执行性能相对于已有的算法具有较大优势,并且在不同核心 CPU 平台下具有较好的性能扩展性.利用多核多线程技术和 cache 优化技术来优化数据库的查询是数据库领域新兴的研究热点,已有一些针对关系数据库中的查询优化工作.但是在移动对象数据库的多核多线程优化方面,还有很多值得深入研究的问题,包括基于内存的移动对象索引技术、缓存优化的单查询处理技术以及与本文研究内容相关的多用户并行查询优化技术等.

References:

[1] Xiong XP, Mokbel MF, Aref WG. SEA-CNN: Scalable processing of continuous *K*-nearest neighbor queries in spatio-temporal databases. In: Proc. of the 21st IEEE Int'l Conf. on Data Engineering. Tokyo: IEEE Computer Society, 2005. 643−654. [doi: 10.1109/ICDE.2005.128]

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

[3] Mouratidis K, Hadjieleftheriou M, Papadis D. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: Proc. of the 24th ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM Press, 2005. 634−645. [doi: 10.1145/1066157.1066230]

[4] Hu HB, Xu JL, Lee DL. A generic framework for monitoring continuous spatial queries over moving objects. In: Proc. of the 24th ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM Press, 2005. 479−490. [doi: 10.1145/1066157.1066212]

[5] Hsueh YL, Zimmermann R, Wang HJ, Ku WS. Partition-Based lazy updates for continuous queries over moving objects. In: Proc. of the 15th Int'l Symp. on Advances in Geographic Information Systems. Seattle: ACM Press, 2007. [doi: 10.1145/1341012.发 1341060]

[6] Mouratidis M, Yiu ML, Papadis D, Mamoulis N. Continuous nearest neighbor monitoring in road networks. In: Proc. of the 32nd Int'l Conf. on Very Large Data Bases. Seoul: ACM Press, 2006. 43−54.

[7] Qiao L, Raman V, Reiss F, Haas PJ, Lohman GM. Main memory scan sharing for multi-core CPUs. In: Proc. of the 34th Int'l Conf. on Very Large Data Bases. Auckland: ACM Press, 2008. 610−621. [doi: 10.1145/1453856.1453924]

[8] Shekhar S, Chawla S, Wrote; Xie KQ, Ma XJ, Yang DQ, *et al*., Trans. Spatial Databases A Tour. Beijing: China Machine Press, 2004 (in Chinese).

[9] Cieslewicz J, Ross KA, Giannakakis I. Parallel buffer for chip multiprocessors. In: Proc. of the 3rd Int'l Workshop on Data Management on New Hardware. Beijing: ACM Press, 2007. [doi: 10.1145/1363189.1363192]

[10] Sun JX, *et al*. Pattern Recognition. Changsha: Press of the National University of Defense Technology, 2002 (in Chinese).

附中文参考文献:

[8] Shekhar S, Chawla S,著;谢昆青,马修军,杨冬青,等,译.空间数据库.北京:机械工业出版社,2004.

[10] 孙即祥,等.现代模式识别.长沙:国防科技大学出版社,2002.

赵亮(1982－),男,山西五台人,博士,工程师,主要研究领域为移动对象数据库.

廖巍(1980－),男,博士,讲师,主要研究领域为时空数据库.

景宁(1963－),男,博士,教授,博士生导师,主要研究领域为地理信息系统,数据库技术.

钟志农(1975－),男,博士,副教授,主要研究领域为地理信息系统,数据库技术.

陈荦(1973－),男,博士,副教授,主要研究领域为空间数据库,地理信息系统.