基于 MapReduce 快速 *k*NN Join 方法

戴 健[1),2)] 丁治明[2)]

[1)](中国科学院大学 北京 100049)

[2)](中国科学院软件研究所基础软件国家工程研究中心 北京 100190)

摘 要 *k*NN 连接是空间数据库领域里一个基本而又重要的问题,被广泛地应用于多个其他领域.它对提高众多实际应用的性能有着重要意义.随着目前参加 *k*NN 连接的数据集的增大和要求的响应时间的缩短(尤其在一些应急环境中),作者实际上对 *k*NN 连接的效率要求更高.然而,目前的方法大多基于单个进程或者单台机器,并不具有很好的伸缩性.为了解决这个问题,作者引入了 map-reduce 框架来运行 *k*NN join 并提出了两种新的方法:基于 map-reduce 的分布式网格概略化 *k*NN join(DSGMP-J)和基于 map-reduce 的 voronoi diagram 下*k*NN join(VDMP-J).并把它们和最新的方法 H-BNLJ 进行了实验对比.实验结果证明了作者提出的 DSGMP-J 和 VDMP-J 方法具有较优的伸缩性.

关键词 *k*NN 连接;大数据;MapReduce

中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2015.00099

MapReduce Based Fast *k*NN Join

DAI Jian[1),2)] DING Zhing-Ming[2)]

[1)](*University of Chinese Academy of Sciences*, *Beijing* 100049)

[2)](*National Fundamental Software Research Center*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)

Abstract *k*NN Join is a basic and important operation which is widely used in many fields. Hence, it plays a significant role in improving the efficiency of the applications in those fields. Nowadays, with the rapid increase of data size and the requirement for shorter response time (especially in some emergency environments), people actually ask for a more efficient way to conduct *k*NN Join. However, conventional *k*NN join operation is mostly running on single computer and/or single process at present, which cannot provide enough scalability. To address this problem, we incorporate the map-reduce framework into the running of *k*NN join and propose two novel methods: distributed sketched grid based *k*NN Join using map-reduce (DSGMP-J) and voronoi diagram based *k*NN Join using map-reduce (VDMP-J). And compare them with a state-of-the-art method: hadoop block nested loop join (H-BNLJ). The experiment results prove that the DSGMP-J and the VDMP-J outperform the H-BNLJ in scalability.

Keywords *k*NN join; big data; MapReduce

收稿日期:2013-10-31;最终修改稿收到日期:2014-07-18.本课题得到国家自然科学基金重大研究计划·重点项目(2012.1~2014.12)"面向非常规突发事件主动感知与应急指挥的物联网技术与系统"(91124001)、国家"八六三"高技术研究发展计划"智慧城市"重大专项之子课题(2013.1~2015.12)"面向城市动态运行管理的大规模数据智能检索技术"(2013AA01A603)、中国科学院"感知中国"先导专项·重点课题(2012.1~2016.12)"面向物理信息感知的传感器时空数据管理与海云服务合成引擎研究"(XDA06020600)、中国科学院战略性科技先导专项课题(XDA06010600)资助.**戴 健**,男,1983 年生,博士研究生,主要研究方向为大规模时空数据管理及数据挖掘.E-mail:daijiancn@126.com.**丁治明**,男,1966 年生,博士,研究员,博士生导师,主要研究领域为数据库与知识库系统、时态与空间数据库、物联网与云计算数据管理.

1 引 言

对很多应用而言，K 最近邻连接（kNN join）是一个非常基本和关键的操作. 这些应用可能来自于知识发现，数据挖掘和空间数据库等.

近些年随着移动应用的广泛发展，相应地，kNN join 具有下面两个趋势：一方面，参与 kNN 连接的数据集越来越大；另一方面，需要 kNN 返回结果的响应时间越来越短.

传统地，很多 K 最近邻连接的算法是在单个进程或者单台机器上执行. 一般要求是用其中一个集合 S 的每个元素扫描一次另一个集合 R 的全部元素，这样，很容易导致计算复杂度为 $O(|S|\cdot|R|)$. 虽然这样的算法在单进程或者单 CPU 上运行很好，然而，随着 K 最近邻连接的规模的增大和要求相应时间的缩短，这些算法不能很好的伸缩，不能满足海量动态数据场景下应用的需求.

归根到底，为了能够快速响应大规模的 K 最近邻连接，我们需要一个具有很好伸缩性，能够并行的集群来进行 K 最近邻连接的计算. MapReduce 体系结构的设计采用了 divide-conquer 的方法，这是一个简单但功能强大的并行和分布式计算架构. 同时，MapReduce 充分考虑到可扩展性. 近年来，MapReduce 得到了来自工业界和学术界的众多支持. 在工业界，Google 公司内部，通过大规模集群和 MapReduce 软件，每天有超过 20 PB 的数据得到处理，每个月处理的数据量超过 400 PB. 在数据分析的基础上，Google 提供了围绕互联网搜索的一系列服务（包括地图服务、定向广告服务等）. 如此大规模的在线数据管理和分析，是传统的关系数据管理技术所无法完成的. 与此同时，在学术界，一些数据库相关的重要国际会议上近年来也越来越多的出现使用 MapReduce 来进行大数据处理的方法. MapReduce 已经成为了一个充满生命力的大数据处理框架.

出于这些考虑，本文中我们用 MapReduce 贯穿了所涉及到的 3 个 kNN 连接处理方法. 首先，我们介绍了局部暴力方法，即 H-BNLJ（Hadoop Block Nested Loop Join）方法，并把它作为了我们的 baseline 方法. 然而，由于 2 次的 MapReduce 和过多的传输代价，随着数据集规模的增大，通讯代价显著增加，H-BNLJ 的伸缩性并不好. 然后，我们思考并尝试了使用基于栅格的数据划分方式，同时使用分布式栅格索引对局部数据进行索引，提出了 DSGMR-J（Distributed Sketched Grid Join）方法. 实验证明，这个栅格的划分方法加上分布式索引的方式可以加速 kNN 连接的处理过程. 但是，由于数据的分布可能呈现出除均匀分布外的多种形式，因此，栅格式划分并不能保证 kNN 连接结果的局部性. 换句话说，在大多数情况下，我们仍然需要 2 次的 MapReduce 才能得到 kNN 的结果. 最后，我们设计了一种基于 Voronoi Diagram 的数据划分方式，在此划分基础上，我们证明了合适的参数选择可以提供一个很好的近似 kNN 连接结果. 从而，通过 1 次 MapReduce 可以得到 kNN 近似连接结果.

在第 2 节，我们形式化定义了本文要解决的问题 kNN Join；第 3 节，我们介绍本文所涉及到的相关技术；第 4 节详细给出我们为了解决快速大规模 kNN 连接所采用的 3 个方法；这 3 个方法的对比分析在第 5 节通过实验结果进行展示；最后，我们在第 6 节总结全文.

2 问题定义

形式化地，设两个定义在 $\Re^d$ 上的数据集 R 和 S. 这两个集合中的每个记录 $\forall r\in R(\forall s\in S)$ 都是一个 d 维空间的点. 本文中，为了简单起见并不失一般性，我们考察 $d=2$ 的情况；同时，两个点之间的距离使用它们的欧式距离来表示 $d(r,s)$. 这样 $knn(r,S)$ 返回 k 个属于 S 集合的最近邻 r 的点.

我们定义 $knnJ(R,S)$ 为 R 和 S 的连接：

$$knnJ(R,S)=\{(r,knn(r,S))|\text{for all } r\in R\}.$$

3 相关技术介绍

3.1 MapReduce

作为一种广受关注和欢迎的编程模型，MapReduce 已经被成功地用在众多大数据集的并行处理程序中[1-6]. 这些并行程序甚至可以很好地扩展到上千台普通机器上. Hadoop 是一个流行的开源 MapReduce 实现. 它运行在 Hadoop 文件系统（HDFS）：一种分布式的存储系统之上. 在 HDFS 中，一个大文件被拆分成很多大小固定的块（不能整除的情况下会有一块碎片）并在计算机之间进行分配. 而 MapReduce 是一种可以直接对这些文件进行操作的框架. 一般地，MapReduce 包含两个由用户自定义的函数，即 map 函数和 reduce 函数. Map 函数用于把输入分块对应地转化为中间结果，而

reduce 函数用于把中间结果进行提取，并进行归纳合并计算得到最终的结果.

MapReduce 技术是一种简洁的并行计算模型，它在系统层面解决了扩展性、容错性等问题，通过接受用户编写的 Map 函数和 Reduce 函数，自动地在可伸缩的大规模集群上并行执行，从而可以处理和分析大规模的数据. 形式化地，

$$\text{Map}:(k_1,v_1)\rightarrow(k_2,v_2)^*$$

$$\text{Reduce}:(k_2,v_2)^*\rightarrow v_3^*$$

3.2　DSTR-Tree

从抽象模型的角度来看，移动对象的时空轨迹对应于 $X\times Y\times T$ 空间的一条曲线. 在移动对象数据库中往往需要对轨迹曲线进行离散化才能进行进一步操作和处理. 网络受限移动对象的动态概略化轨迹 R 树索引(Dynamic Sketched-Trajectory R-Tree for Network constrained Moving Objects，DSTR-Tree)就是一种采用概率化方法对空间轨迹进行处理的灵活方法.

这里，设移动对象的时空轨迹对应于时空空间 $I_x\times I_y\times I_t$，其中，$I_x=[x_0,x_1]$，$I_y=[y_0,y_1]$，并且 $I_t=[t_0,T]$(T 代表不断增长的时间). DSTR-Tree 将此三维空间的投影空间 $I_x\times I_y$ 栅格化为 $n\times m$ 个栅格，其中 $\zeta_x=\dfrac{x_1-x_0}{n}$，$\zeta_y=\dfrac{y_1-y_0}{m}$. 由于 I_t 的不断增长，它被划分为等间距 Δt 的时间段，$\zeta_t=\Delta t$.

DSTR-Tree 是一个非常灵活的移动对象轨迹索引. 它的灵活性体现在两个方面：(1) 它在 $I_x\times I_y$ 的投影为栅格，而栅格被认为是一种非常高效的二维空间索引结构，并被经常应用于各种实际应用中；(2) 它的概率化算法可以有效地把空间轨迹转化成栅格内的线段集合，避免了路网匹配中不能把轨迹匹配到路上的个别情况，从而快速进行处理一些查询(如最近邻查询).

3.3　沃罗诺伊图

沃罗诺伊图(Voronoi Diagram，也称作 Dirichlet tessellation，狄利克雷镶嵌)是由俄国数学家 Georgy Fedoseevich Voronoi 建立的空间分割算法. 最初来源于笛卡尔用凸域分割空间的思想.

Voronoi 图是根据目标的最邻近原则而对空间目标所在的整个空间的一种剖分，其中的每一个目标均与其最近的邻近区域即 Voronoi 区域相连. 通常地，一个点目标的 Voronoi 区域可以定义为由距该点目标比其他所有目标距离近的目标构成：

$$p_i^v=\{x\mid d(x,p_i)\leq d(x,p_j),p_i,p_j\in P,i\neq j\},$$

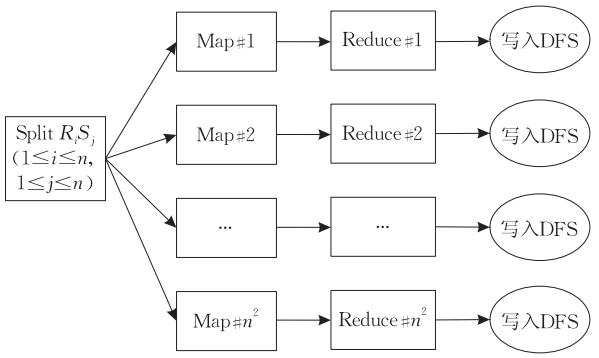
其中，p_i^v 是考察点 p_i 的 voronoi 区域，P 为待考察点集，x 为空间中任意一点，d 为距离函数.

对组成 Voronoi 图的每一个 Voronoi 区域而言，它不仅获取了考察点的空间邻域，反映了它与周围考察点的空间关系，而且 Voronoi 区域对于不同的空间关系是敏感的，因此相应地，有限的 Voronoi 内部区域起到了一个无限的外部区域的类似作用，而 Voronoi 的范围却比目标的补空间大大缩小，易于我们的 kNN 连接操作.

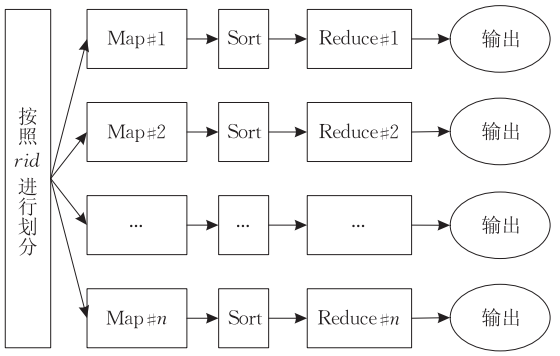
4　快速 kNN Join

4.1　Baseline：H-BNLJ

文献[7]提出了一种 H-BNLJ(Hadoop Block Nested Loop Join)的方法，如图 1 所示. 本文把它作为我们的基准对比方法. 这里，H-BNLJ 是一种直接的局部暴力解决 kNN 连接的算法，它利用了 MapReduce的嵌套循环算法. 基本思想是把待连接的两个集合 R 和 S 分割成大小相等的 n 块，这里可以通过线性扫描的方法来进行，每个块中分别含有 $\dfrac{|R|}{n}$(或 $\dfrac{|S|}{n}$)个元素. 然后，在 Map 阶段，每个连接块包含一个来自于 R 的分割块和一个来自 S 的分割



(a) 第1次MapReduce



(b) 第2次MapReduce

图 1　H-BNLJ 框架

块(也就是总共有 n^2 个连接块). 在 Reduce 阶段,采用了 n^2 个 reducer 来处理每个 mapper 生成的中间结果. 每个 reducer 在本地嵌套执行局部 R 和 S 的 kNN 连接,也就是对每个局部块中的 S 通过嵌套循环找到在局部块中 R 的 kNN. 所有来自 reducer 的结果写入(n^2)DFS 文件中.

然而,注意到在上面的 MapReduce 过程中,每个记录 $r\in R$ 出现在一个$\frac{|R|}{n}$块中,而它被复制到了 n 个与它相连接的连接块中(与每个来自$\frac{|S|}{n}$组成一个连接块). 在 reduce 阶段,只有本地的 kNN 被发现出来. 因此,为了得到全局的 kNN 连接结果,还需要对每个 $r\in R$ 通过第 1 次 MapReduce 所产生的 $n\times k$ 个候选结果进行筛选. 也就是要求对形如 $(rid,sid,d(rid,sid))$ 的元组进行排序,其中,rid 为 $r\in R$ 中 r 的 id,sid 为 $s\in S$ 中 s 的 id,$d(rid,sid)$为 r 和 s 的距离. 这样,第 2 次 MapReduce 在 map 阶段,需要把第一次 MapReduce 的结果按照 rid 作为划分划分主键(partitioning key). 同时,第 2 次 MapReduce 的 reducer 遍历$(rid,list(sid,d(r,s)))$,对其中的 $list(sid,d(r,s))$进行按照 $d(r,s)$的升序排序,然后为每个 rid 得出 top-k 的结果.

4.2 DSGMR-J

H-BNLJ 本质上作为一种暴力解法,虽然使用了两次 MapReduce,但并不是一种高效的方法. 首先,它划分的依据是线性扫描,如果原本的 R 和 S 并没有排序(二维的排序诸如 Z-Value 等),那么由于不能保证 R 中元素的全局 kNN 结果和它被分配到 n^2 个连接块中的某一个(对称地,对 S 也一样),它在第一步 MapReduce 得到的结果可能和最后的 kNN 连接结果偏离很大. 其次,由于没有采取索引的方法,将会导致在 $|R|$ 或者 $|S|$ 很大的时候,H-BNLJ 不能有效地从外存(DFS)数据加载内存中.

通常来说,索引的本质是排序和内容摘要. 为了弥补 H-BNLJ 的不足,我们引入分布式概略化网格索引(Distributed Sketched Grid,DSG)来对数据进行划分和索引,DSG 是 DSTR-Tree[8] 的变种. DSTR-Tree 原本是一种用于移动对象轨迹概略化索引的树形结构,在本文中,我们使用它来进行空间区域的划分和子区域数据的索引.

采用 DSG 而不是简单基于简单栅格的索引的原因在于两个方面. 一方面,DSTR-Tree[8-10] 很容易将我们的方法扩展到路网空间进行 kNN 连接. 另一方面,DSTR-Tree 很容易将我们的应用范围扩展到移动对象(Moving Object)的连续 kNN 查询. 这是我们的下一步工作,超出了本文的内容,但是本文仍将采用 DSG 来进行索引.

为了对数据集 R 和 S 进行划分,首先需要对 R 和 S 所在空间范围进行栅格化划分. 设 R 所在的空间区间为$[R_{x_{\min}},R_{x_{\max}}]\times[R_{y_{\min}},R_{y_{\max}}]$,$S$ 所在的空间区域为$[S_{x_{\min}},S_{x_{\max}}]\times[S_{y_{\min}},S_{y_{\max}}]$. 在进行栅格化处理时,将相应地需要对空间$[\min(R_{x_{\min}},S_{x_{\min}}),\max(R_{x_{\max}},S_{x_{\max}})]\times[\min(R_{y_{\min}},S_{y_{\min}}),\max(R_{y_{\max}},S_{y_{\max}})]$进行等大小 $m\times m$ 划分,如图 2 所示.

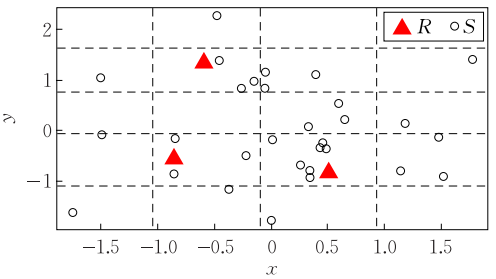


图 2 DSG 分割($m=5$)

算法 1. DSG 划分(DSG partition).

输入:R,S,m

输出:DSG 索引及相应数据块

1. $x_{\max}=R_{x_{\max}}>S_{x_{\max}}\ ?\ R_{x_{\max}}:S_{x_{\max}}$
2. $x_{\min}=R_{x_{\min}}<S_{x_{\min}}\ ?\ R_{x_{\min}}:S_{x_{\min}}$
3. $y_{\max}=R_{y_{\max}}>S_{y_{\max}}\ ?\ R_{y_{\max}}:S_{y_{\max}}$
4. $x_{\max}=R_{x_{\max}}>S_{x_{\max}}\ ?\ R_{x_{\max}}:S_{x_{\max}}$
5. $\zeta_x=\frac{x_{\max}-x_{\min}}{m}$
6. $\zeta_y=\frac{y_{\max}-y_{\min}}{m}$
7. $Interval_i=[x_{\min}+\zeta_x(i-1),x_{\min}+\zeta_x\times i]\times[y_{\min}+\zeta_y(i-1),y_{\min}+\zeta_y\times i]$
8. distribute the corresponding data and index

算法 1 描述了根据 R,S 的变化范围生成的 m^2 个栅格的过程. 其中每个栅格的 x 和 y 变化范围为 $interval_x$ 和 $interval_y$. 每个 R 或 S 中的元素根据它的 x 和 y 坐标确定所属的栅格. 对每个栅格而言,我们分布地构建了对应于此栅格的分布式索引 DSG. 这样,每个 reducer 可以快速地通过本地 DSG 索引来发现本地 kNN 从而避免了嵌套循环.

这里,使用本地 DSG 索引来发现本地 kNN 是一个 filter-refine 的过程. 显然地,如果当前栅格中的元素 $|R_{\mathrm{DSG}(i)}|\geq k$ 或者 $|S_{\mathrm{DSG}(i)}|\geq k$(我们总是栅格化含有元素较多的那个集合),并且查询点位于栅格的中央附近的时候(即 $\left|x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)\right|\leq\delta$

并且$\left|y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)\right|\leq\delta$），那么所有的$R_{\mathrm{DSG}(i)}$或者$S_{\mathrm{DSG}(i)}$均分别为对方的$k$NN. 当且仅当$|R_{\mathrm{DSG}(i)}|<k$或者$|S_{\mathrm{DSG}(i)}|<k$时候，在本地的$k$NN才无法发现所有的$k$个元素，从而需要进行扩展.

这里，我们定义$\left|x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)\right|\leq\delta$并且$\left|y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)\right|\leq\delta$为点$(x,y)$的$grid_i(\delta)$邻域，其中$i$为栅格的编号. 当点不在$grid_i(\delta)$邻域的时候，我们这里需要对当前栅格进行扩展. 我们的扩展方式有4种：左上扩展，左下扩展，右上扩展和右下扩展.

具体地，当$x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)<-\delta$并且$y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)>\delta$时，我们进行左上扩展，合并原有栅格和相邻左上，上和左边的栅格形成一个新的栅格；当$x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)>\delta$并且$y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)>\delta$时，我们进行右上扩展，合并原有栅格和相邻右上，上和右边的栅格形成一个新的栅格；当$x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)>\delta$并且$y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)<-\delta$时，我们进行右下扩展，合并原有栅格和相邻右上，上和右边的栅格形成一个新的栅格；当$x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)<-\delta$并且$y-\left(y_{\min}+\zeta_y\times i-\frac{\zeta_y}{2}\right)<-\delta$时，我们进行左下扩展，合并原有栅格和相邻右上，上和右边的栅格形成一个新的栅格.

定理 1.　当$\delta>\max\left(\frac{\zeta_y}{4},\frac{\zeta_x}{4}\right)$时候，根据当前点的坐标，我们可以通过一次相应的扩展调整后得到，$(x,y)\in grid_{i'}(\delta)$.

证明.　以左上扩展为例，我们有

$$x-\left(x_{\min}+\zeta_x\times i-\frac{\zeta_x}{2}\right)<-\delta,$$

即$x-x_{\min}-\zeta_x\times i+\frac{\zeta_x}{2}<-\delta$，

也就是$x-x_{\min}-\zeta_x\times i+\frac{\zeta_x}{2}+2\delta<\delta$，

因为$\delta>\max\left(\frac{\zeta_y}{4},\frac{\zeta_x}{4}\right)$，

所以$x-(x_{\min}+\zeta_x\times(i-1))<x-x_{\min}-\zeta_x\times i+\frac{\zeta_x}{2}+2\delta<\delta$，

得到$|x-(x_{\min}+\zeta_x\times(i-1))|<\delta$，

同样地，$|y-(y_{\min}+\zeta_y\times(i+1))|<\delta$.

综上，当$\delta>\max\left(\frac{\zeta_y}{4},\frac{\zeta_x}{4}\right)$时候，通过左上扩展将得到$(x,y)\in grid_{i'}(\delta)$.

同法，其余3种扩展也可以证明.　　证毕.

然而，即使在扩展后可能仍然无法得到全部的kNN结果集，这时我们仍然需要两遍MapReduce过程来进行kNN连接的结果计算，但是这里的kNN连接的集合为$\overline{R'}$，其中R'为已经完成kNN连接的属于集合R的子集，$\overline{R'}$为R的补集（这里设$|R|<|S|$）.

我们注意到在不预先知道数据集R和S的数据分布的情况下，在某一个$m\times m$子区域m_i内的R和S的数据可能很多，然而在另外一个子区域m_j的数据可能很少，即$|m_i|\gg|m_j|$. 这种不均衡的情况会大大影响我们的两阶段MapReduce过程. 由于存在平凡的划分$|m_j|$，第一次MapReduce中map的结果将会有很多空值. 换句话说，极端的情况下，由于R和S的数据分布为$\forall r\in R, r\notin grid(i)$，DSGMP-J将会退化成H-BNLJ.

4.3　VDMR-J

对应我们的kNN Join而言，DSGMP-J的不足之处在于，它均匀地划分了空间区域. 换句话说，在(XYT)的投影空间(XY)上，DSGMP-J并没有考虑R或者S的原始分布情况.

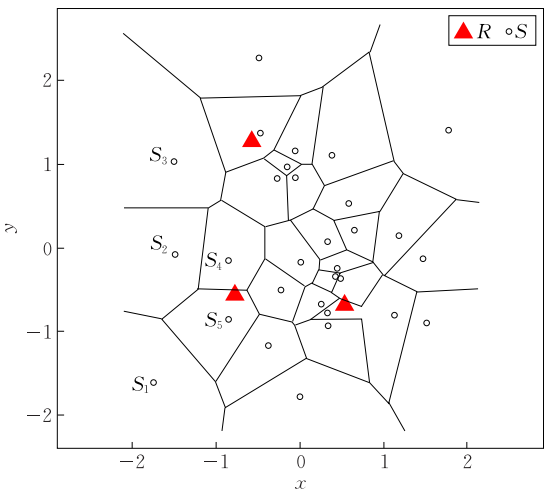


图 3　Voronoi 分割

由于Voronoi Diagram[11-12]充分考虑了空间的相互位置关系，基于Voronoi Diagram，对于kNN

Join 问题，我们可以通过单遍 MapReduce 来近似完成. 我们把方法命名为 VDMR-J(Voronoi Diagram based kNN Join using MapReduce). 为了说明我们的近似方法，我们需要证明下面的两个定理及两个相关推论.

定理 2. 凸多边形外的点到该凸多边形各顶点的距离之和大于凸多边形内(上)的点到该凸多边形各顶点的距离之和.

证明. 不失一般性，设 $q_1,\cdots,q_n$ 是凸多边形 V 的 n 个顶点，$d(r,V)$ 表示点 r 到凸多边形 V 的各个顶点的距离之和，且 $d(r,V)=\sum_{i=1}^{n}d(r,q_i)$. 设 p 是 V 外一点，我们可以分两种情况进行讨论：

(1) 如果 p 在 V 的某一条边旁. 设 p 在边 $q_iq_j(i\neq j)$ 旁，$d(p,V)=|pq_1|+\cdots+|pq_n|$. 则在凸多边形 V 内，必然存在一点 p'，有 $|p'q_1|<|pq_1|,\cdots,|p'q_n|<|pq_n|$，使得 $d(p',V)=|p'q_1|+\cdots+|p'q_n|<d(p,V)$.

(2) 如果 p 在 V 的某一个顶点旁. 设 p 在边 q_i 旁，则有 $|q_iq_1|<|pq_1|,\cdots,|q_iq_n|<|pq_n|$. 使得 $d(q_i,V)=|q_iq_1|+\cdots+|q_iq_n|<d(p,V)$.

综上可知命题正确. 证毕.

推论 1. 设 kNN Join 的两个点集分别为 $R=\{r_1,r_2,\cdots,r_m\}$ 和 $S=\{s_1,s_2,\cdots,s_m\}$，其中 R 的凸壳边界为凸多边形 C_R(相应地，S 的为 C_S)，其顶点分别为 R 和 S 中的点，同时，记 S 所在的 Voronoi 图记作 $\bigcup_{i=1}^{n}VD_i(S)$(设 $|R|<|S|$)，则对 R_j 所对在的 $VD_i(S)$，我们有 R_j 到 $VD_i(S)$ 各顶点距离之和小于任何 R_j 到 $\overline{VD_i(S)}$ 各顶点距离之和.

证明. $VD_i(S)$ 和以当前 $VD_i(S)$ 的点为定点的凸多边形一一对应.

由于 R_j 在 $VD_i(S)$ 内部，由定理 2 可知，R_j 到 $VD_i(S)$ 各顶点距离之和小于任何 R_j 到 $\overline{VD_i(S)}$ 各顶点距离之和. 证毕.

定理 3. 设集合 $S'=\{S'_1,S'_2,\cdots,S'_N\}\subset S$，而且 kNN$(R_j)\subset S'$，我们从这些点中按照一定的概率 $p=\frac{1}{\varepsilon^2N}$ 随机均匀抽样 $\hat{N}$ 次，这样我们可以得到集合 $\hat{S}'=\{\hat{S}'_1,\hat{S}'_2,\cdots,\hat{S}'_{\hat{N}}\}$(其中 $\varepsilon\in[0,1]$). 定义 $r(N,\psi)=\sum_{i=1}^{N}[d(S'_i,R_j)<\psi]$，其中，$[d_i<\psi]=1$，如果满足 $d_i<\psi$. 定义 $s(\hat{N},\psi)=\sum_{i=1}^{\hat{N}}[d(\hat{S}'_i,R_j)<\psi]$，其中，相应地，$[d_i<\psi]=1$，如果满足 $d_i<\psi$. 那么，$\hat{r}(\hat{N},\psi)=\frac{1}{p}s(\hat{N},\psi)$ 是 $r(N,\psi)$ 的无偏估计，且标准差 $sd\leqslant\varepsilon N$，$\forall\varepsilon\in[0,1]$.

证明. 首先，S' 的存在性，我们通过构造的方法可以获得. 由定理 2 和推理 2.1 我们可以知道，通过扩展 R_j 所在的 $VD(S)$，我们可以不断得到离 R_j 较近的点集，从而直到 kNN$(R_j)\subset\bigcup_{i=1}^{n'}VD_i(S)$. 以这些 $S_j\in\bigcup_{i=1}^{n'}VD_i(S)$ 作为 S' 中的点，我们必然可以得到 $S'=\{S'_1,S'_2,\cdots,S'_N\}\subset S$，而且 kNN$(R_i)\subset S'$.

其次，我们定义 N 个独立同分布的随机变量，他们分别为 $X_1,X_2,\cdots,X_N$，其中 X_i 以概率 p 的可能性为 1，1$-p$ 为 0. 我们把这些随机变量和相应的 $[d_i<\psi]=1$ 对应起来，就有

$$s(\hat{N},\psi)=\sum_{i=1}^{\hat{N}}[d(\hat{S}'_i,R_j)<\psi]=\sum_{i=1}^{r(N,\psi)}X_{l_i}.$$

根据定义，X_i 是概率为 $p=\frac{1}{\varepsilon^2N}$ 的 Bernoulli 实验. 这样，$s(\hat{N},\psi)$ 就是一个可以表示为 $B(r(N,\psi),p)$ 的二项分布. 根据二项分布的性质，我们可以得到

$$E[\hat{r}(\hat{N},\psi)]=E\left[\frac{1}{p}s(\hat{N},\psi)\right]=\frac{1}{p}E[s(\hat{N},\psi)]=r(N,\psi)$$

并且

$$\begin{aligned}Var[\hat{r}(\hat{N},\psi)]&=Var\left[\frac{s(\hat{N},\psi)}{p}\right]\\&=\frac{1}{p^2}Var[s(\hat{N},\psi)]=\frac{1}{p^2}r(N,\psi)p(1-p)\\&<\frac{N}{p}=(\varepsilon N)^2.\end{aligned}$$

证毕.

如果我们使用随机采样的 $s\in\hat{N}$ 的 $\hat{r}(\hat{N},\psi)$ 来估计 $s\in N$ 的 $r(N,\psi)$，则有下面的推论.

推论 2. 通过 $s=\underbrace{\operatorname{argmin}}_{s\in\hat{N}}|\hat{r}(\hat{N},\psi)-r(N,\psi)|$ 得到的 s 满足 $Pr[|r(N,\psi)-r|\leqslant\varepsilon N]\geqslant1-e^{-2/\varepsilon}$.

证明. 具体地，推理 3.1 的目标是证明使用 $\hat{r}(\hat{N},\psi)$ 估计出来的 kNN 结果集和真实 kNN 结果集的差值以大于 $1-e^{-2/\varepsilon}$ 的概率小于 εN.

$$\begin{aligned}Pr[|r(N,\psi)-r|>\varepsilon N]&=(1-p)^{2\varepsilon N}=\left(1-\frac{1}{\varepsilon^2N}\right)^{2\varepsilon N}\\&<e^{(-1/\varepsilon^2)2\varepsilon}=e^{-2/\varepsilon},\end{aligned}$$

也就是 $Pr[|r(N,\psi)-r|\leqslant\varepsilon N]\geqslant1-e^{-2/\varepsilon}$. 证毕.

Pre-processing. 使用 Fortune's algorithm 生成相应的具有标签的 voronoi diagram. 这里，如果 $|R|<|S|$，则生成 S 相应的 voronoi diagram，反之，

则生成 R 相应的 voronoi diagram. 需要说明的是，一方面 Fortune 算法是一个增量扫描线算法，这样有 2 个好处，一个是可以实现对大规模数据的一边加载一边编号，不需要一次性加载所有数据到内存中，另一个是 Fortune 算法的时间复杂度是 $O(n\log n)$，保证了预处理的时间是可以接受的；另一方面，我们使用了改进的 Fortune 算法，在扫描的同时按照扫描顺序加上了标签(L)，$L\in N$ 是一个从 1 开始的递增自然数.

Partition. 对生成的 voronoi diagram 的按照 L 值的大小，进行 N 分割($N>2k$)，也就是每 N 个元素放入一个分割块中，从而得到 $\frac{\max(|R|,|S|)}{N}$ 个分割块.

Map. 假设我们对如图 3 所示的数据集进行 3-NN Join 操作，我们可以采用 2 个 mapper 和 1 个 reducer. 首先，每个 mapper(mapper_i)读取它相应的分割(split_i). 所有的在 split_i 的元素被放入一个 hash map 中，其中每个元素的形式为$\langle rid, VN(r)\rangle$，$rid$ 为 R 中元素的 id，$VN(r)\subset VD_i(S)$，$\bigcup_{i=1}^{n} VD_i(S)=S$ 为 S 的第 i 个包含 r 的 voronoi diagram 分割. 如果 $\forall r\in R$，我们都有 $|VN(r)|\geq k$，其中，$VNk(r)$ 为 r 的 k 邻域. 根据定理 2，我们有 $k\text{NN}(r)=VNk(r)$. 因此，对多个 R 中元素可以同时求解它们的 kNN.

Reduce. 得到 mapper 的$\langle rid, VN(r)\rangle$作为输入，然后进行 $VN(r)$作为搜索空间进行 kNN join. 由定理 3 可知，在进行N分割后的$s=\underbrace{\arg\min}_{s\in\hat{N}}|\hat{r}(\hat{N},\psi)-r(N,\psi)|$在 N 足够大的时候，是以较大的概率逼近于真实 kNN 连接结果的. 因此，我们只要通过 reducer 找出 $VN(r)$的 kNN 结果即可近似地表示为真实 kNN 连接结果.

需要注意 Map 函数开始产生输出结果时，并不是简单地把结果直接写到磁盘. 每个 map 任务都有一个环形内存缓冲区. 默认情况下，输出会被先写到环形内存缓冲区中，当缓冲内容达到指定比率(一般默认为 0.8)时，map 任务会把内容溢出写到磁盘中(而不是 DFS 中). 这样，map 任务如果还有输出结果，那么它需要进行阻塞直到溢写过程结束.

在我们的 kNN 连接中，如果数据集非常大(即 $|R|$，$|S|$都非常大)的时候，我们通过配置，采用了两个优化方法. 首先，由于数据集非常大，溢写文件的合并是频繁发生的. 我们需要根据具体数据集的大小来调整溢写文件的合并值(例如，在 hadoop 中就是 io.sort.factor)来进行调优. 其次，当溢写频繁发生时，我们需要对 map 的中间输出结果进行压缩来减少频繁溢写带来的代价. 相应地，在 hadoop 中，我们需要设置 mapred.compress.map.output 为 true，并定义或者重写相应的压缩方法.

在我们的 kNN 连接中，当 k 非常大的时候，显然 map 的输出结果非常大，我们可以在 map 的基础上定义它的优化函数 combiner 来限制可用的 reduce 输入，从而减少对可用带宽的消耗，保证我们程序的健壮性，防止频繁的 map 任务重分配. 当 $|R|$，$|S|$都非常大，但是 $\frac{|S|}{|R|}$ 并不大的时候(例如 $\frac{|S|}{|R|}\leq 10^4$)，在有限的机器节点的情况下，我们可以通过配置 combiner 合并的溢写次数来进行调优. 相应地，在 hadoop 里，我们需要配置 min.num.spills.for.combine 属性的值，并且此值必须大于等于 3. 在本文我们的方法主要适用于 $|k|\leq\frac{|S|}{|R|}$ when $|R|<|S|$的情况.

5　实验分析

为了验证我们方法的有效性和高效性，证明我们的方法能够胜任快速大规模的 kNN 连接操作. 我们在不同的实验环境下进行了详细的实验及分析，包括不同的数据集，不同的节点数目，不同的连接参数设置. 在相同的实验条件下将本文提出的方法和 baseline 方法进行了对比分析.

5.1　实验条件

实验使用平台配置：Dell PowerEdge R720，处理器类型 Intel® Xeon® CPU E5-26200@2.00GHz，网卡 6 个，内存 60907 MB，硬盘 9.09 TB. 通过 VMware vSphere 5 Enterprise Plus 进行管理. 在虚拟机上我们使用的 CentOS 5.9 的 64 位系统，使用的 Hadoop 版本是 0.20.2，DFS 的块大小是 64 MB.

实验中使用的数据集来源于两个数据源，一个数据源是我们随机生成的数据集 D_1，另外一个数据源是北京市出租车的数据源 D_2. D_1 中我们随机生成了 200 000 000 个二维点，然后随机抽取了 20 000 个点作为 R 中元素，其余作为了 S 中的元素. D_2 中包含了北京市的 60258 辆出租车 7 天中的运动轨迹数据，我们从中选取了 260 000 条数据记录，并在其中随机选择了 10 000 个点作为 R 中的元素，其余作为了 S 中的元素.

整个系统的配置图如图 4 所示.

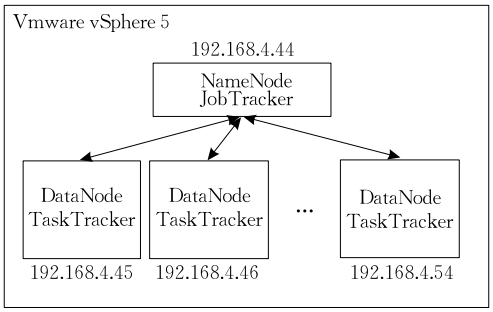


图 4 实验 MapReduce 平台结构示意图

5.2 实验结果与分析

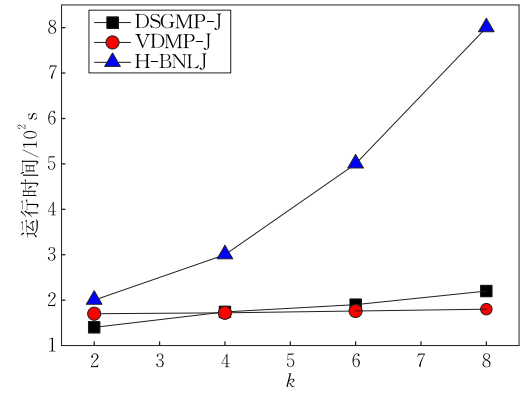
5.2.1 不同的数据集及 k 的影响

图 5(a)和(b)分别展示了我们在不同的数据集(D_1 和 D_2)不同的 k 取值($k=\{2,4,6\}$)上运行 3 个方法(VDMP-J, H-BNLJ 和 DSGMP-J)的运行时间. 关于运行时间这里有两点说明,一是所有的运行时间均为多次实验的平均值;二是这里的运行时间包含建立索引和预处理的时间. 另外,本实验的目的在于考察不同的数据集和 k 对于运行时间的影响,因此,我们设置节点数目均为 6.

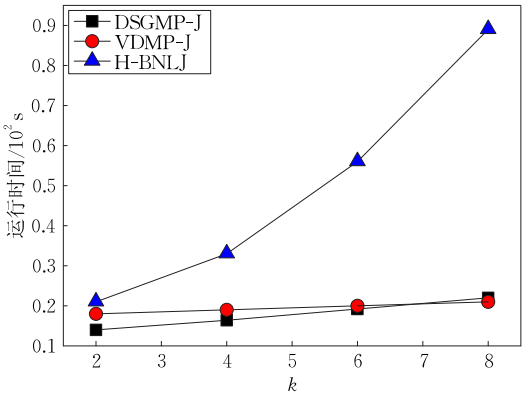
总体上,从图 5(a)和(b)中我们可以看出随着 k 的增加,3 个方法的运行时间都在增加. 其中,H-BNLJ 的时间增加最为迅速,VDMP-J 和 DSGMP-J 的增加较为缓慢. 这说明了 VDMP-J 和 DSGMP-J 方法的有效性,通过 DSG 索引划分和 VD 划分,和没有进行局部索引的暴力方法 H-BNLJ 相比,可以有效地减少运行时间. 另一方面,在 k 较小的时候,DSGMP-J 往往优于 VDMP-J,原因在于 DSGMP-J 建立索引的时间较短,而且在 k 较小的时候,DSGMP-J 的 filter-refine 可以有效获取 kNN 连接的结果. 往往并不需要进行第二次 MapReduce.

5.2.2 不同的节点数的影响

本实验的目的在于比较采用 MapReduce 框架的不同方法的伸缩性,即随着节点数目的增加运行时间的变化情况. 实验中,对于同一个数据集(D_1 和 D_2),我们设置了相同的 k 值($k=6$),并把节点数目(DataNode number)从 4 增加到了 6,最后进一步增加到了 10,分别运行了 VDMP-J, H-BNLJ 和 DSGMP-J,得到的实验结果如图 6(a)和(b)所示. 注意由于一次实验结果可能存在一定的随机性,这里,我们的运行时间均为平均时间. 而且,我们的时间单位为 10^2 s.

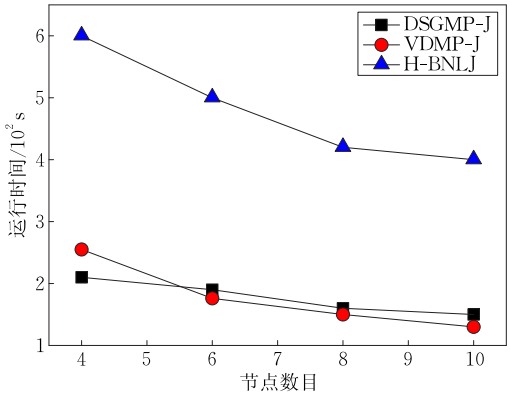


(a) 在 D_1 上的运行时间

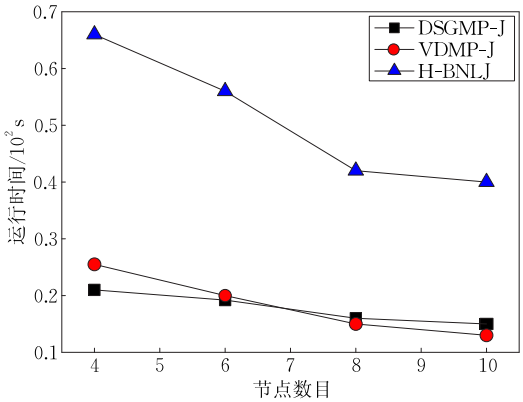


(b) 在 D_2 上的运行时间

图 5 不同数据集不同的 k 下 kNN Join 运行时间



(a) 在 D_1 上的运行时间



(b) 在 D_2 上的运行时间

图 6 不同节点数目下 kNN Join 运行时间

从图 6(a)可以看出，从总体上讲，一方面，随着节点数目的增加，3 个方法的运行时间都呈现下降趋势，这说明 MapReduce 框架本身保证了 3 种方法的伸缩性都比较好，但是需要指出的是随着节点数目的增多，同时带来的也有通讯代价的增多，因此下降率并不是线性的；另一方面，在运行时间上，H-BNLJ 的运行时间一直长于 DSGMP-J 的运行时间，并且 VDMP-J 的运行时间最少，这说明 VDMP-J 的执行效率最高. 但是需要指出 VDMP-J 是一种近似算法，就 DSGMP-J 和 H-BNLJ 而言，DSGMP-J 较优. 另外，需要说明的是，H-BNLJ 的下降较为缓慢，DSGMP-J 次之，而 VDMP-J 下降的最为迅速. 这里也说明了 DSGMP-J 采用栅格化的有效性和 VDMP-J 使用一遍 MapRedcue 的高效性. 图 6(b)给出了在数据集 D_2 上运行结果，同样验证了我们上面的两个结论：(1) MapReduce 可以保证方法具有较好的伸缩性；(2) 在近似的情况下，VDMP-J 较优，在精确的情况下，DSGMP-J 较优.

我们同时也通过实验分析了在不同 k 下VDMP-J 的精度，实验结果如图 7 所示. 可以看出相比较数据集 D_2，数据集 D_1 的 VDMP-J 具有较高的精度，这也同时验证了在 N 足够大的时候，VDMP-J 是以较大的概率逼近于真实 kNN 连接结果的.

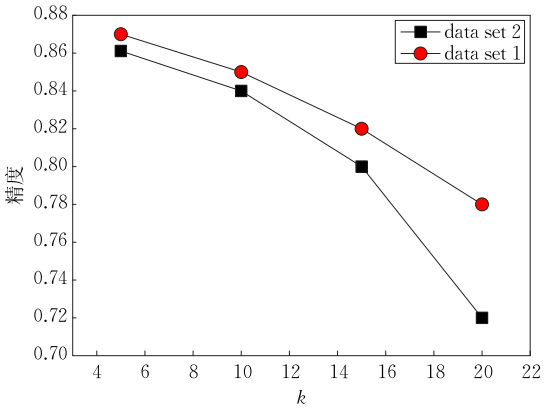


图 7　不同 k 下 VDMP-J 的精度

6　结　论

kNN 连接是空间数据库领域里一个基本而又重要的问题，它对提高众多实际应用的性能有着重要意义. 随着目前参加 kNN 连接的数据集的增大和要求的响应时间的缩短(尤其在一些应急环境中)，kNN 连接的效率要求越来越高. 传统的方法大多基于单个进程或者单台机器，并不具有很好的伸缩性.

我们发现：一方面，有效的过滤可以减少待比较的两个集合中元素的对数(candidate pair)；另一方面，在执行 map-reduce 的时候，良好的划分策略可以达到较好的负载均衡. 基于这两个观察，本文在 H-BNLJ 两次 MapReduce 框架的基础上，提出了精确的 DSGMP-J 和近似的 VDMP-J 方法，充分利用了局部索引和划分来求解 kNN 连接问题. 实验证明了我们提出的 DSGMP-J 和 VDMP-J 方法的有效性和具有较优的伸缩性.

参　考　文　献

[1] Candan K S, Nagarkar P, Nagendra M, Yu R. RanKloud: A scalable ranked query processing framework on Hadoop//Proceedings of the 14th International Conference on Extending Database Technology. Uppsala, Sweden, 2011: 574-577

[2] Okcan A, Riedewald M. Processing theta-joins using MapReduce//Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. Athens, Greece, 2011: 949-960

[3] Wang Jinbao, Wu Sai, Gao Hong, et al. Indexing multi-dimensional data in a cloud system//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, Indiana, USA, 2010: 591-602

[4] Yu Cui, Zhang Rui, Huang Yaochun, Xiong Hui. High-dimensional kNN joins with incremental updates. Geoinformatica, 2010, 14(1): 55-82

[5] Zhou X, Abel D J, Truffet D. Data partitioning for parallel spatial join processing. Geoinformatica, 1998, 2(2): 175-204

[6] Zhang Shubin, Han Jizhong, Liu Zhiyong, et al. Spatial queries evaluation with MapReduce//Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing. Lanzhou, Gansu, China, 2009: 287-292

[7] Zhang C, Li F, Jestes J. Efficient parallel kNN joins for large data in MapReduce//Proceedings of the 15th International Conference on Extending Database Technology. Berlin, Germany, 2012: 38-49

[8] Ding Zhi-Ming. An index structure for frequently updated network-constrained moving object trajectories. Chinese Journal of Computers, 2012, 35(7): 1448-1461(in Chinese)
(丁治明. 一种适合于频繁位置更新的网络受限移动对象轨迹索引. 计算机学报, 2012, 35(7): 1448-1461)

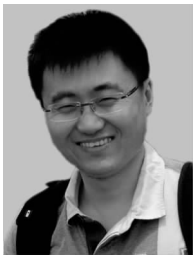
[9] Güting R H, de Almeida V T, Ding Z. Modeling and querying moving objects in networks. The VLDB Journal, 2006, 15(2): 165-190

[10] Yin Xiao-Lan, Ding Zhi-Ming, Li Jing, Moving object query k nearest neighbors in spatial network databases. Journal of Computer Research and Development, 2007, 44(Supplement): 55-60(in Chinese)

(殷晓岚，丁治明，李京. 移动对象在空间网络数据库上的 kNN 查询. 计算机研究与发展，2007，44(增刊)：55-60)

[11] Akdogan A, Demiryurek U, Banaei-Kashani F, Shahabi C. Voronoi-based geospatial query processing with MapReduce //Proceedings of the 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science. Indianapolis, Indiana, USA, 2010: 9-16

[12] Böhm C, Krebs F. The k-nearest neighbor join: Turbo charging the KDD process. Knowledge and Information Systems, 2004, 6(6): 728-749



DAI Jian, born in 1983, Ph. D. candidate. His current research interests include spatial keyword management and data mining.

DING Zhi-Ming, born in 1966, Ph. D., professor, Ph. D. supervisor. His main research interests include database systems, mobile computing, embedded systems, spatial temporal database/data mining, and information retrieval.

Background

The k Nearest Neighbor (kNN) join is a common and primitive spatial operation that has been widely used in many real-world applications, such as: finding the nearest friends in the WeChat app and delivering the goods to the nearest places in www. taobao. com. Meanwhile, it also broadly incorporated in many data mining and analytic tasks, such as: k-means and other similar clustering methods.

A lot of existing works related to kNN join have been conducted with the assumption that it is running on a single machine. However, with the coming of the age of big data, to store and join the two data sets in kNN join is impractical. We need figure out a new way to efficiently address the kNN join in big data environment.

Inspired by our observation that we can use the locality of the data to conduct kNN join while sharing nothing, we propose two novel methods (Distributed Sketched Grid based kNN Join, DSGMP-J and Voronoi Diagram based kNN Join, VDMP-J) and compare them with a state-of-the-art method (Hadoop Block Nested Loop Join, H-BNLJ).

Specifically, the DSGMP-J is a precise method and may degenerate to H-BNLJ in some extreme cases, and the VDMP-J is an approximate approach which only efficiently includes one MapReduce. The experiment results prove the effectiveness and the efficiency of our methods.

The work is supported by the grants from the National Natural Science Foundation of China (91124001, 60970030).