一种基于 CHORD 系统的多维相似查询处理方法

A CHORD – Based Multidimensional Similarity Query Processing Method

李智玲 （中国人民公安大学 计算机物证 北京 100038）

摘要：如何在对等计算环境下处理多维空间中的相似查询是目前学术界的研究热点之一。本文通过基于代表点的多维空间划分策略，提出了一种对等计算环境下的分布式多维索引技术，实现了一种基于 CHORD 系统的多维相似查询处理。仿真实验证明了本文提出的方法的有效性。

关键词：对等计算 空间划分 索引 相似查询

1 概述

在 CHORD[3] 系统中，每个节点具有一个标识，所有节点按照标识的值依次连接组成一个环状网络，即从一个标识从 0 到 N－1 的环上。每个节点都使用前驱指针和后继指针来维护这种环状网络结构，并使用路由指针表来路由查询。具体地，每个节点 v 都维护了一个有 logN 个表项的路由指针表。节点 v 的第 i 个表项指向顺时针方向上距离节点 v 最近的节点 s 且它们之间的距离至少为 2^i-1，即 $s = successor(v+2^i-1)$，其中 i 的取值是从 1 到 logN。节点 s 被称为节点 v 的第 i 个指针。另一方面，在 CHORD 系统中，数据空间中的数据点也被映射到相同的标识空间，即从 0 到 N－1。每个数据点都被映射到该数据点标识所指的后继节点上，从而在数据与节点之间建立了一种一一映射关系。

本文提出的方法是基于 CHORD 系统的。为了将代表点与单元格映射到节点上，需要将代表点线性化到一维连续空间上。理论上，空间填充曲线（space filling curve）[2]，如 Hilbert 曲线，可以将这 N 个代表点映射到一维连续空间上。由于 $N << 2^d$，所以空间填充曲线无法线性化代表点。此外，随着数据维度的增长，空间填充曲线无法保持原有数据空间与变换后的线性空间之间近似性。也就是说，在原有数据空间中相互邻近的数据对象，在经过线性变换的数据空间中，彼此将不再邻近。

本文采用一种非常简单的方法，将代表点线性化到一维连续空间：每个代表点用自己在序列中的出现次序作为自己的标识，即第一个产生的代表点的标识为“1”，第二个产生的代表点的标识为“2”，依次类推。图 1 给出了 8 个由伪随机算法生成的代表点，以及根据它们在序列中出现的次序为它们分配的标识。这样，通过将代表点线性化到一维连续空间上，整个高维数据空间也就相应地被映射到了一维连续空间上。这是因为，每个代表点都对应了一个单元格，因而每个代表点就代表了所有落在与该代表点对应的单元格以内的数据对象。

在代表点被线性化之后，根据代表点的标识，就可以将这 N 个代表点连接在一起，构成一个环状网络。与 CHORD 网络一样，通过将节点映射到环上，每个节点就将负责一段连续的区间，所有属于该区间的代表点都将被分配给管理该区间的节点。这样，所有属于与代表点对应的单元格中的数据对象也就一起被分配给代表点所属的节点。这种方法具备两个特点：

（1）虽然每个节点负责一段连续的标识区间，但从数据空间的角度看，每个节点上的单元格空间并不遵循固定的形状。

（2）所有节点上的所有单元格的体积总和正好等于整个数据空间的体积，而且任意两个节点负责的单元格在空间上不存在任何重叠。

由上可知，基于 CHORD 系统，本文提出了一种灵活的数据分配方法：通过调整单元格的体积（粒度）和改变节点在网络中的位置，系统就可以随意地调整每个节点上的单元格数目，达到调整系统负载状况的目标。图 1 展示了如何将 8 个代表点的数据分配到一个 CHORD 系统上。

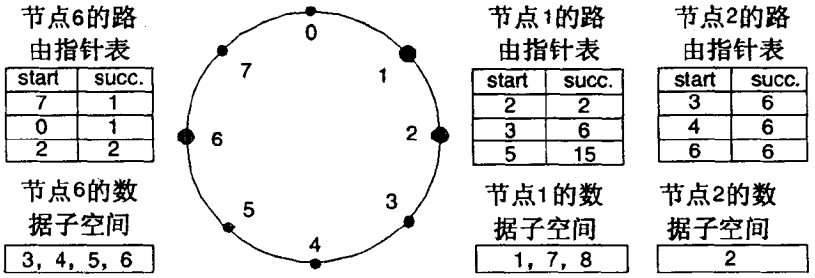

图 1　一个 CHORD 系统

1.1 全局索引

为了实现 P2P 环境下的相似查询，每个节点必须维护一棵由 N 个代表点构造而成的多维索引树，称为“全局索引”，具有以下两个用途：

(1) 为节点分配数据对象。当一个代表点被分配给一个节点之后，所有落在与该代表点对应的单元格中的数据对象都将被分配给该节点。给定任意一个数据对象，节点使用全局索引来确定该数据对象的“最近邻代表点”，即该数据对象属于与哪个代表点对应的单元格。以代表点的标识为搜索键，采用 CHORD 路由协议，就可以定位到该数据对象所属的节点，完成数据对象的分配操作。

(2) 确定相似查询需要访问的代表点。给定一个相似查询，搜索全局索引，就可确定哪些代表点与该相似查询的搜索区域相交。换言之，凡是与相似查询的搜索区域相交的单元格都有可能包含结果，这些单元格都必须被查询所访问。以需要被访问的单元格对应的代表点为搜索键，就可以将查询路由给这些代表点所属的节点，实现相似查询处理(见下节)。

本文采用势点树(vp－tree)。构造一棵势点树的时间复杂度是 O(N×logN)。给定一个查询点，在势点树上执行最近邻查询(定义见下节)的时间复杂度是 O(logN)，其中 N 是代表点的个数。由于这 N 个代表点是在节点加入 P2P 系统时生成的且固定不变，所以势点树只需构建一次。此外，由于代表点的个数与 P2P 系统最多可容纳的节点数成正比，所以每个节点因维护索引树而引起的存储代价并不会造成节点上存储资源的匮乏。

1.2 K－最近邻查询处理算法

定义 1：设数据空间是一个 d 维的单位超立方体，表示为 $\Omega=[0,1]^d$。给定一个数据对象集合 $DB\subseteq\Omega$，则一个查询点 $q\in\Omega$ 的最近邻满足条件：

$$nn(q)=\{o\in DB \mid \forall o'\in DB: dist(o,q)<dist(o',q)\}$$

其中，dist 表示欧几里德距离函数，o 和 o' 表示数据对象。

给定一个查询点，最近邻查询就是找到距离该查询点最近的那个数据对象，而 k－最近邻查询就是要找到距离该查询点最近的 k 个数据对象。为了找到一个查询点 q 的 k 个最近邻，需要确定距离 q 最近的第 k 个数据点与 q 之间的距离 r，然后以 r 为搜索半径得到 k 个最近邻。由于无法事先确定这个搜索半径 r，所以现有的方法都以迭代的方式，通过不断扩大搜索半径来实现 k－最近邻查询。本文也采用这种思想。假设节点 v 发出一个查询 q，则 k－最近邻查询处理算法如下：

(1) 节点 v 首先设定初始搜索半径 r，初始值可以参考 k－最近邻查询的历史记录。

(2) 节点 v 执行区域查询 <q,r>，同时维护一个 k－最近邻中间结果集，用于保存 k 个可能的全局最近邻结果。

① 节点 v 在势点树上执行区域查询 <q,r>，并将所有与搜索区域相交的单元格 cid 记录在一个列表中。

② 对于每个还没有被区域查询 <q,r> 访问到的单元格 cid，节点 v 判断该 cid 是否属于后继节点。如果该 cid 属于节点 v 的后继节点，则后继节点在本地执行区域搜索，并将查询结果返回给节点 v。否则，CHORD 系统的 find_successor(cid)[3] 方法被用于定位该 cid 的所有者，并由该节点在本地执行区域查询。

③ 根据得到的 k 个最近邻结果，节点 v 更新 k－最近邻中间结果集。如果发生了更新操作，则返回“假”。否则，返回“真”。

(3) 如果搜索结束标志为“假”，则意味着节点 v 还没有得到全局的 k 个最近邻结果，需将搜索区域 r 扩大为 r＋Δr，并跳转到第 2 步，继续执行 k－最近邻查询。否则，节点 v 已经找到了全局的 k 个最近邻结果，结束 k－最近邻查询。

1.3 节点的加入与退出

当一个节点加入或退出 P2P 系统后,该节点将执行相应的更新操作,以保持数据分配和系统路由状态的正确性。当一个节点 v 加入系统后,节点 v 首先初始化节点 v 的前驱节点和路由指针表,更新相关节点的前驱节点、后继节点和路由指针表,确保节点 v 被正确地加入到 P2P 网络中。这与节点加入 CHORD 系统时的操作相同;然后,节点 v 从它的后继节点 u 接管所有属于区间,(v. successor,v),的单元格,即所有属于区间,(v. successor,v),的代表点,以及属于这些代表点所对应的单元格中的所有数据对象都将被转移给节点 v。

当一个节点 v 退出 P2P 系统后,系统执行与上述操作中的第一步,将属于节点 v 上的所有代表点和数据对象都将被转移给它的后继节点 u,以保持数据分配和系统路由状态的正确性。

此外,当一个节点加入系统时,它不能简单地通过散列自身的 IP 地址来确定它在 P2P 系统中的位置(节点标识);否则,这将很容易导致系统负载状态的不均衡性。对于一个加入系统的节点而言,如果它能够作为当前系统中一个负载较重的节点的前驱节点而加入系统,那么它就可以从这个负载较重的节点上分担一部分负载,实现系统负载均衡的目标(见下节)。

1.4 负载均衡

当一个节点 v 加入系统时,让节点 v 从系统中负载最重的节点上分担一部分负载,达到均衡系统负载的目的。如果当前系统中负载最重的是节点 u,那么节点 v 将作为节点 u 的前驱节点而加入系统。根据 CHORD 协议,节点 u 的一部分负载将被转移给节点 v。这样,系统负载的不均衡性就得到了缓解。在节点加入系统时才调整系统负载不仅可以有效地均衡系统的负载均衡状态,还能保证系统开销较低(开销是指用于保证指针表的正确性而必须执行的更新操作)。

实现上述策略的关键点在于如何确定当前系统中负载最重的节点。由于 P2P 系统缺乏集中式控制,所以无法知道哪个节点的负载最重。因此,当一个节点加入系统时,它只需找到一个负载较重的节点,并作为该节点的前驱节点而加入系统即可。为了找到一个负载较重的节点,新加入系统的节点只需随机地访问一些节点,比较这些节点的负载状况并选取负载最重的节点即可。

另一方面,当系统状态相对稳定且又处于负载不均衡的情况下,没有理由等待新节点的加入来均衡系统的负载状况。上述的负载均衡策略无法解决这种情况,需要考虑如何使系统中的节点依据自身的负载状况,通过主动地调整自身在网络中的位置,达到均衡系统负载的目标。

(1) 当一个节点过载时,该节点将一部分负载(代表点与相应的数据对象)转移给它的前驱节点和后继节点。如果后继(前驱)节点在获得了转移给它的负载后,自身的负载状态变为过载,那么该后继(前驱)节点将把过多的负载转移给它的后继(前驱)节点,依次类推。

(2) 当一个节点轻载时,该节点上的所有代表点和数据对象都将被转移给它的后继节点,并被强迫退出 P2P 系统;然后,通过让该节点重新加入 P2P 系统,实现均衡系统负载的目标。

2 性能评价

为了验证相似查询处理方法的性能,实现了一个用 JAVA SDK 1.4.2 编写的 P2P 模拟器。所有仿真实验都是在一个具有 Intel Xeon 2GHz 和 512MB 主存的 PC 上运行完成的。实验使用的数据集遵循 Zipfian 分布($\alpha=0.8$),包含 100,000 个数据对象。伪随机算法被用于生成代表点。

为了衡量系统性能,采用了两个评价指标:覆盖度和负载均衡。覆盖是指被查询访问的节点总数。节点上的数据对象个数被用来评价系统的负载均衡状况。为了评价系统的可扩展性,在每组实验中变换两个系统参数的取值:网络规模和数据维度。对于每组实验,网络规模的大小是从 1000 个节点变化到 8000 个节点,而数据维度是从 2 维变化到 10 维。每组实验结果都是 500 个随机产生的相似查询的结果的统计平均值。

2.1 参数 c 对系统性能的影响

这组实验运行在 2000 个节点的 CHORD 系统上,数据集的维度为 10。参数 c 的取值范围是从 4 到 40。对于 c 的每个取值,测试了系统的负载均衡状况和 k-最近邻查询处理的覆盖度,其中参数 k 的取值为 10。

图 2 展示了负载均衡策略对系统负载均衡状态的影响。从实验结果可以看到,随着 c 的增大,整个系统的负载均衡也就越接近于理想情况。因此,适当的取

一个较大的 c 值，可以有效地均衡系统的负载状况。注意到，任何一个实际的 P2P 系统都无法达到理想的

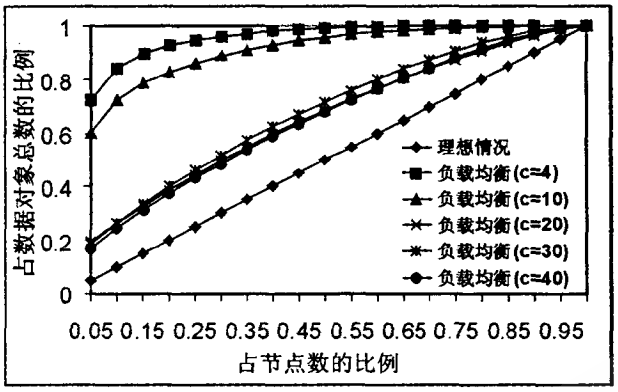

图 2　倾斜数据集上的负载均衡

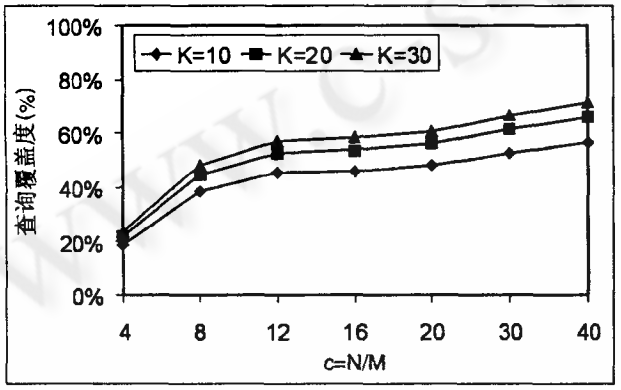

图 3　查询覆盖度

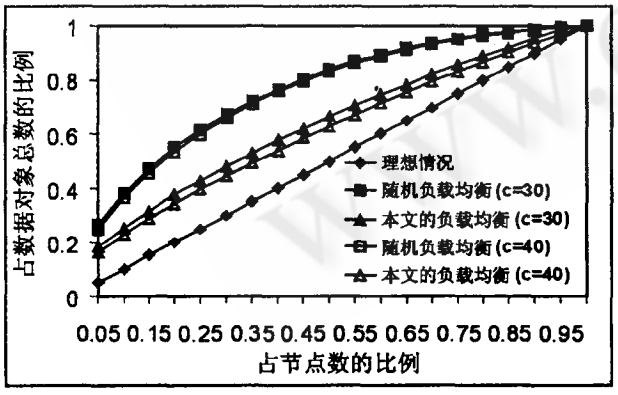

图 4　10 维倾斜数据集上的负载均衡状况

负载均衡状态。图 3 给出了参数 c 对查询覆盖度的影响。实验结果表明，随着 c 的增大，查询覆盖度会随之增大。这是因为，随着 c 的增大，被划分出的单元格数目($N=c\times M$)相应地增多了，而随着单元格数目的增多，包含需要被查询访问的单元格的节点数目也就相应地增多。虽然一个较小的 c 值可以降低查询覆盖度，但却增加了系统负载的不均衡性。因此，需要在系统的查询处理代价与负载均衡状况之间达到一个折衷：在保证系统负载均衡的前提下，尽可能地降低系统的查询处理代价。下面将评估负载均衡策略的有效性。

2.2　负载均衡策略的效果

第二组实验测试了负载均衡策略的有效性，基准测试采用随机地负载均衡策略。随机负载均衡策略是指，在一个节点加入系统时，该节点仅通过散列函数确定自己在系统中的节点标识，而不考虑系统中其他节点上的负载状况。所有实验运行在一个 2000 个节点的 CHORD 系统上，数据集的维度为 10。

图 4 给出了负载均衡策略在 10 维倾斜数据集上的负载均衡效果。从实验结果可以看到，本文提出的负载均衡策略的负载均衡效果远远好于随机负载均衡策略的负载均衡效果，而且非常接近理想情况下的负载均衡状况。这说明本文提出的负载均衡策略是有效的，能够适应数据的倾斜特征。另一方面，随着 c 的增大(从 30 变为 40)，系统的负载均衡状况分别改善了 2% 和 5%。这是因为，当 c 变大后，用于调整负载的单元格数量增多了，所以通过调整节点上的单元格数量能够有效地改善整个系统的负载均衡状况。这说明，无论数据分布如何，只要选择一个适当大的 c 值就可以确保整个系统的负载均衡状况。

从实验结果可以看到，在相同测试条件下，基于 CHORD 系统的最近邻查询处理的查询覆盖度小于 MUCK 方法的查询覆盖度。随着数据维度的增长，与搜索区域相交的单元格数目会相应地增多。因此，尽管单元格的数目 N 并没有增加，但查询覆盖度也会增加。随着网络规模的扩大，查询覆盖度减小了。这是因为，虽然对于一个固定的 K 值，与搜索区域相交的单元格数目会随着单元格数目 N 的增大而增多，但是其增长速度却远远小于与节点数目的增长。因此，查询覆盖度也就相应地降低了。这说明本文提出的方法可以很好的适应网络规模的增长。当确定了网络规模和维度之后，随着 K 值的增大，查询结果的数目增多了，

(下转第 44 页)

(上接第 40 页)

所以被查询访问的节点数目也就增多了。但是,对于每个 K 值而言,查询覆盖度的增长趋势的变化较小,这说明本文提出的方法并不会因用户所需查询结果的增多而快速地增长。

3 结束语

本文提出了一种基于 CHORD 系统的多维相似查询处理方法:通过基于代表点的数据空间划分技术,首先在代表点和子空间之间建立起对应关系;通过线性化代表点,将代表点(与其对应的数据子空间)映射到 P2P 系统中的节点上,在代表点、子空间和节点之间建立起一一映射关系;最后,通过搜索基于代表点的全局索引,以代表点的标识为搜索键,实现了 P2P 环境下的多维相似查询处理。

参考文献

1 Gaede V, Gunther O. Multidimensional Access Methods. ACM Computing Surveys, 1998, 30(2): 170 – 231.

2 Schmidt C, Parashar M. Flexible Information Discovery in Decentralized Distributed Systems. In Proc. of HDPC, 2003.

3 Stoica I, Moris R, Karger D, Kaashoek F, Balakrishnan H. Chord: A Scalable Peer – to – Peer Lookup Service for Internet Applications. In Proc. of ACM SIGCOMM, 2001. 149 – 160.

4 Ganesan P, Yang B, Garcia – Molina H. One Torus to Rule Them All: Multi – Dimensional Queries in P2P Systems. In Proc. of WebDB, 2004. 19 – 24.

5 Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A Scalable Content – Addressable Network. In Proc. of ACM SIGCOMM, 2001. 161 – 172.