基于网络延迟的P2P路由算法的研究*)

王传殊　王意洁

(国防科技大学计算机学院并行与分布处理国家重点实验室　长沙 410073)

摘　要　近年来,P2P计算应用已经超过Web应用而成为占用互联网带宽最多的网络应用。针对目前P2P系统中采用的随机选择邻居节点的方法会降低路由效率以及增大网络开销方面的问题,在分析Chord方法特点的基础上,提出一种改进的Chord构建算法DeChord。从逻辑上相邻的点在物理上也相邻这一原则出发,DeChord采用Chord数据定位算法;利用全局网络定位系统计算节点坐标并以此为依据计算节点间的物理距离,节点加入时充分考虑节点之间的逻辑距离与物理距离的一致性,系统节点总是选择距离自己物理距离较近的节点作为邻居节点;DeChord算法使得节点的路由表的信息能得到及时的更新。DeChord中的邻居节点选择方式可以降低消息路由过程中每一跳的网络延时,从而降低整个消息路由的开销。模拟实验表明,利用该算法建立的P2P系统能大幅度降低数据定位的延时。

关键词　对等计算,Chord,节点坐标,数据定位

Research of Network-delay-based P2P Routing Algorithm

WANG Chuan-Shu　WANG Yi-Jie

(National Key Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073)

Abstract　In recent years, most network bandwidth has been consumed by Peer-to-Peer (P2P) computing applications rather than WEB applications. However, peers in P2P systems choose logical neighbors randomly without any knowledge about underlying physical topology at present, which can decrease greatly the efficiency of message routing and consume more bandwidth. According to the principle that the neighbor peer in logical is also the physical neighbor, an improved Chord construct algorithm DeChord, is proposed. In DeChord, Global Network Position is utilized to compute the coordinate of peers, and then the distance between nodes is computed based on these coordinates. The consistency between logical distance and physical distance is considered when nodes join the system and the closer peers in the physical network are taken as neighbor peers. DeChord makes the information of routing table updated in time. DeChord can decrease the latency of every hop, therefore the latency of total message routing is decreased. Simulation results indicate that the performance of data location in the system that is constructed by DeChord can be significantly improved.

Keywords　Peer-to-Peer computing, Chord, Node coordinate, Data location

1　引言

对等(Peer-to-Peer,P2P)计算是近年来兴起的一种重要网络计算技术。与传统的C/S计算不同,P2P计算中的各结点是地位平等的对等体,通过直接交换来利用计算、存储、信息和带宽等资源。目前P2P计算在很多领域都有着大量的研究和应用。P2P资源定位技术是P2P计算中的基础性关键技术。

P2P系统拓扑结构可以划分为集中式和分布式:集中式拓扑的P2P系统中,一个(或少数几个)中央服务器作为目录服务器来协调其它各个节点之间的交互,搜索效率高,但存在单点失效问题,如Napster[1]。分布式结构解决了集中式结构中节点之间交换信息要通过中心服务器的问题,系统中节点都是平等的,每个节点即使服务器也是客户端,系统中没有任何服务器,节点之间直接交换信息,具有良好的扩展性,如Gnutella[2]、FreeNet[3]、Chord[4]、CAN[5]、Pastry[6]。

分布式结构中的每条路由路径都是由源节点到目标节点的一系列的跳点组成,其中每对跳点之间的网络延迟直接影响路由效率。但是目前的P2P应用存在到逻辑路径和物理路径之间不一致的问题,即逻辑中相邻的两个节点之间的物理距离可能很大。一致性问题的存在使得单纯依据逻辑距离进行节点路由时,选中的逻辑上距离最进的节点实际上可能是物理上距离较大的节点,从而导致很大的网络延迟,严重影响了路由效率。

本文从P2P应用存在到逻辑路径和物理路径之间不一致出发,在Chord的基础上提出一种基于物理坐标计算节点距离的P2P算法DeChord(Network-Delay-based on Chord)。节点选择物理上近距离的节点构建路由表(finger表),并在路由过程中选择近距离的节点进行路由,能大幅度降低系统的定位延迟。

2　Chord算法

Chord中每个关键字和节点分别用m位的二进制字符串表示,称为关键值(key value)和节点标识(node ID),Chord通过一致性哈希技术(consistent hashing)[7]得到这些唯一的二进制字符串。

Chord通过如下的方法将关键字存储在对应的节点中:关键值k存储在一个节点ID等于k的节点上,如果节点ID

*)基金项目:国家重点基础研究发展规划(973)(2002CB312105);高等学校全国优秀博士学位论文作者专项资金项目(200141)。王传殊　硕士生,研究方向为网络计算;王意洁　教授,博士生师,CCF高级会员,主要研究方向为网络计算、数据库技术、移动计算等。

等于 k 的节点不存在，则存储在关键值大于 k 的第一个节点。存储关键值 k 的节点称为关键值 k 的后继节点，k 的后继节点就是 k 在顺时针的方向上的第一个节点。节点按照节点值从 1 到 2^m 按照顺时针方向构成一个环形的拓扑结构，这个环称为 Chord 环(Chord Ring)[8]，图 1 是一个 $m=3$ 的 Chord 环，环中有三个节点：1，3，8。关键值 1 的后继节点为 1，2 的后继节点为 3，6 的后继节点为 8。

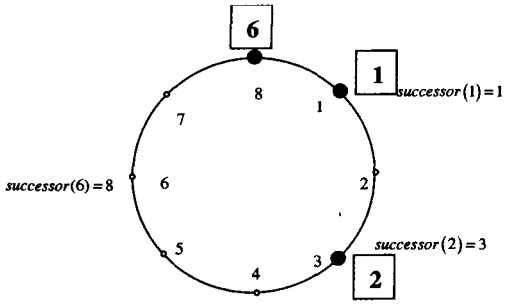

图 1　Chord 环体系结构和关键值分配示意图

Chord 中的每个节点维护一个 finger 表，存储关键字和其后继节点的信息，如表 1 所示。定位关键值 *key* 时，每个节点通过查询 finger 表，将搜索请求发送到离 *key* 最近的节点上，直至定位到 *key* 所在的节点。节点数为 N 的系统在稳定状态下，Chord 方法的节点度数为 $O(\log n)$，查询操作的消息开销为 $O(\log N)$，动态维护开销为 $O(\log^2 N)$。

表 1　DeChord 中节点路由表结构

域　名	定　义
finger[k]. start	$(n+2^{k-1}) \bmod 2^m$　$1 \leqslant k \leqslant m$
finger[k]. node	finger[k]. start 的后继节点
successor	本节点后继
predecessor	本节点前继

Chord 具有以下特点：分布性——Chord 是一个完全分布式的系统，具有很好的鲁棒性，也更适合松散结构的 P2P 应用；可扩展性——Chord 的查询负载是与节点数的对数成比例的，因此 Chord 不需要进行任何参数的调整便可适用于大规模系统；负载平衡——Chord 利用哈希函数把关键值分布在节点上，提供一定程度上的负载平衡；有效性——Chord 通过自动调整其路由表来反映节点加入的成功或者失败，只要下层网络不发生错误，管理某个关键值的节点总是能够被找到，从而保证了 Chord 系统的有效性。

传统 Chord 算法只考虑节点路由过程中的逻辑路径而忽略了物理路径。但是在逻辑上 ID 相邻的两个节点而在物理路径上可能并不相邻，这样节点在路由过程中由于物理距离的因素可能导致很大的网络延迟，而影响查找效率。DeChord 算法正是针对该问题进行改进。

3　全局网络定位系统 GNP

全局网络定位系统(Global Network Position，简称 GNP)[9]把网络构建成一个几何模型，并且把网络中的每个网络节点作为几何模型中的一个点，任何两个节点之间的网络距离由几何空间中两个节点之间的距离决定[10]，网络节点之间的距离能被终端同步的计算出来，这种基于坐标的方法非常适合于 P2P 应用[11,12]。

GNP 中节点坐标的计算过程为：① 首先在选定的几何空间计算地标(Landmark)节点的坐标；② 加入系统的节点获取地标节点，以地标节点为参考计算自己的坐标。

假设 Internet 被建模成一个特别的几何空间 S，定义其中节点 H 的坐标为 C，在坐标上使用的距离函数为 f^S，两个节点 H_1 和 H_2 之间计算出来的距离为 $f(C^S_{H1}, C^S_{H2})$ 或者 $d^S_{H1,H2}$。

① 地标节点的作用是：提供一系列的参考坐标值，以便被 S 中的其它节点所使用。地标节点首先测量彼此间的网络延时，然后计算节点间实际测量的距离与坐标距离的差值，当差值达到最小时获取节点坐标，即下面函数值最小：

$$f_{obj}(c_{L_1}, \cdots, c_{L_N}) = \sum_{i,j \in \{1,\cdots,N\} | i>j} \varepsilon(d_{L_iL_j}, \hat{d}_{L_iL_j}) \tag{1}$$

$$\varepsilon(d_{H_1H_2}, \hat{d}_{H_1H_2}) = \left(\frac{d_{H_1H_2} - \hat{d}_{H_1H_2}}{d_{H_1H_2}}\right)^2 \tag{2}$$

其中，$\hat{d}_{L_iL_j}$ 为 L_i 节点和 L_j 节点间实际测量距离，$d_{L_iL_j}$ 为节点 L_i 和 L_j 间坐标距离，$f_{obj}(c_{L_1}, \cdots, c_{L_N})$ 为节点集 $(c_{L_1}, \cdots, c_{L_N})$ 中各对节点间坐标距离与测量距离之间相对误差之和，$\varepsilon(d_{H_1H_2}, \hat{d}_{H_1H_2})$ 为 H_1 节点和 H_2 节点间坐标距离与测量的距离之间的相对误差。

对于一个 D 维的空间，我们至少使用 $D+1$ 个地标节点，否则不可能计算出唯一的节点坐标值。实验表明地标节点在 16 个以上且坐标空间达到 7 维时能达到足够高的精度，而在这个基础上再增加地标节点数量和空间维数，其计算结果对精度的提高非常小。

② 获取普通节点的坐标。每个普通节点 H 使用 ICMP 包测量它到 N 个地标节点的 RTT 值，取每条路径上多次测量值中的最小值作为它的距离值，地标节点只需简单的回复 ICMP 包。使用这 N 个距离值 d_{HLi}，节点 H 可以计算自身的坐标值 C^S_H，使得在测量出的距离和计算出的距离间的总体误差最小，即如下函数取值最小：

$$f_{obj}(c_{L_1}, \cdots, c_{L_N}) = \sum_{i \in \{1,\cdots,N\}|} \varepsilon(d_{L_iL_j}, \hat{d}_{L_iL_j}) \tag{3}$$

4　基于网络延迟的 P2P 路由算法 DeChord

DeChord 的思想是：把网络构建成为 N 维坐标的几何坐标模型，网络中的每个节点都映射到几何空间中的一个点，当有一个节点加入 P2P 网络中时，根据 GNP 计算出自己的节点坐标值；节点之间的距离可以由节点的坐标值根据几何空间中点到点之间的距离计算公式计算得出；每个节点都维护一张 finger 表，finger 表中的节点是动态变化的，当有节点加入或者退出时，需要更新相关节点的 finger 表；更新节点的 finger 表时，淘汰节点到该节点的 finger 表表项的后继节点距离较远的节点，并定期触发稳定算法保证节点信息得到及时更新。

DeChord 算法将网络延迟引入 Chord 算法，节点综合考虑节点 ID 和网络延迟来构建 finger 表，使得节点的 finger 表中每个表项的后继节点都是距离该节点物理距离较近的节点。DeChord 算法主要由如下几个部分组成：关键字的查询；节点的加入；稳定和失效处理。

4.1　关键字的查询

DeChord 中，定位关键值 k 时，每个节点都通过查询 finger 表，将搜索请求发送到离 k 最近的节点上，直至定位到 k 所在的节点，算法描述如图 2 所示。其中，n 网络中的节点，Value(n)表示 n 的节点 ID，successor(n)表示 n 的后继节点，m 为 finger 表中表项数。

```
1.  flag ← 0;
2.  while ( flag = 0 ) do
3.  if  Value(n) = k
4.  then  return n and flag ← 1;
5.  else if  k ∈ (Value(n), Value(successor(n)))
6.  then  return successor(n) and flag = 1;
7.  else  检查 n 的指针表 n.fingerList;
8.  for  i = 1 to m
9.  if  k ∈ [finger[i].start, finger[i].node]
10. then  return finger[i].node and flag = 1;
11. else  n = finger[m].node;
```

图 2　DeChord 中关键字查找算法

4.2　节点加入

在动态 P2P 网络中，节点可能随时加入，DeChord 需要把关键字正确的映射到相应的节点上，因此在 DeChord 中，每个关键字 k 都正确的保存在它的后继节点 successor(k)上。为了实现快速、正确的查找，同时也要保证 finger 表的信息正确。

假设新加入的节点为 n，我们按照如下具体步骤完成新节点的加入：

4.2.1　*为新节点分配 ID 并根据 GNP 计算节点坐标，初始化 finger 表和前置节点*

新节点加入 DeChord 时，先根据哈希算法给节点分配唯一 ID，并根据 GNP 计算出节点坐标。新节点 n 通过系统中的一个已知的节点 n' 来找到自己的前置节点并初始化 finger 表。如果严格的寻找每一个节点的后继节点即 successor，那么完成 finger 表的 m 个入口的计算要花费的时间为 $O(m * \log N)$。为了减少时间开销，在进行查询的时候 DeChord 会检查一个区间的入口和下一个区间的入口是否相同。这样就可以将需要检查的节点数减少为 $O(\log N)$。算法描述如图 3 所示。

```
1.  n 为加入系统的新节点 and n' 为系统中
    已经存在的任意一个节点;
2.  从 NodeHashTable 中为 n 分配一个节点 ID,
    Value(n) = Hash(Key);
3.  GNP 计算出 n 的节点坐标值 n.Coordinate;
4.  初始化自身 finger 表, successor(n) = n';
5.  if  (Value(n) < finger[i+1].start < finger[i].node)
6.  then  finger[i+1].node = finger[i].node;
7.  else  定位并更新 finger[i+1].start 的后继节点即
    successor(finger[i+1].start);
```

图 3　新节点加入算法

4.2.2　*更改其他相关节点的 finger 表*

当有新节点加入后必然影响其他节点的 finger 表，DeChord 使用这样的机制：当有新节点加入 finger 表时，更新原 finger 表中的一个节点。我们综合考虑节点 ID 和网络节点距离两方面的因素，替换 finger 表的表项中后继节点到该节点距离较远的点，节点距离根据 GNP 计算的节点坐标值按照公式(4)计算得到。算法描述如图 4 所示。

$$Distance(n',n)=\sqrt{\sum_{i=1}^{d}(X_i-Y_i)^2} \tag{4}$$

其中，d 为坐标空间的维数，X_i，Y_i 分别是两节点 n'，n 的第 i 维坐标值。

```
1.  新节点 n 调用定位过程定位其前继节点并向其
    前继节点发送更新 finger 表的消息;
2.  节点 n' 接收到更新 finger 表第 i 项的消息;
3.  根据 n' 和 n 的节点坐标 n'.Coordinate 和 n.Coordinate
    计算两节点距离 Distance(n', n)
4.  if  Value(finger[i].node) > Value(n')
5.  then if  Value(finger[i].node) > Value(n)
6.  then if  Distance(n', finger[i].node) > Distance(n', n)
7.  then  finger[i].node = n;
8.  向前继发送更新 finger 表第 i 项的消息;
9.  else  向前继发送更新 finger 表第 i 项的消息;
10. else 第 i 项更新完毕;
11. else  if n 是环中最后一个节点
12. then  if  Distance(n', finger[i].node) > Distance(n', n)
13. then  if  Value(n) > Value(n')
14. then  finger[i].node = n;
15. 向前继发送更新 finger 表的消息;
16. else  向前继发送更新 finger 表的消息;
17. else 更新完毕;
18. else  if  Value(finger[i].node) > Value(n)
19. then  if  Distance(n', finger[i].node) > Distance(n', n)
20. then  finger[i].node = n;
21. 向前继发送更新 finger 的消息;
22. else  向前继发送更新 finger 的消息;
23. else 更新完毕;
```

图 4　更新 *finger* 表的算法

```
1.  新节点 n 加入到节点 n' 的 finger 表中;
2.  predecessor = nil;
3.  n' 获得 n 的后继 successor = n'.find_successor(n);
4.  x = successor.predecessor;
5.  if  (x ∈ (n, successor))
6.  then  successor = x;
7.  successor 调用 notify(n) 过程利用本身节点值
    更新其他节点的前继;
8.  n.notify(n')
9.  if  ( predecessor is nil or n' ∈ (predecessor, n))
10. then  predecessor = n';
11. 调用节点定位过程 find_successor 查找 finger[i].start 的
    值更新 finger[i].node 的值( i > 1);
```

图 5　系统稳定算法的算法 *stabilize*

4.2.3　*关键字的转移*

节点加入的最后一步就是要更改由于新节点加入引起的关键字到节点的映射的变化，即关键字的转移。一些原本映射到新节点的后继节点的关键字可能要映射到这个新节点上，也就是把所有后继节点是 n 的关键字转移到新节点 n 上。

4.3　稳定和失效处理

在 DeChord 中，其稳定协议的基本要求就是要保持 successor 指针的及时更新，如果 successor 指针信息得不到及时

(下转第 97 页)

入的讨论，并在本文的写作过程中提出了许多宝贵的意见和建议，在此一并表示感谢。

参 考 文 献

1 Montgomery PL. Modular Multiplication Without Trial Division. Mathematics of Computation, 1985, 44(170): 519～521
2 Acar T, BS Kaliski Jr, C Ko C. Analyzing and Comparing Montgomery Multiplication Algorithms. IEEE Micro, 1995, 16(3): 26～33
3 Karatsuba A, Ofman Y. Multiplication of multidigit numbers on automata. Soviet Physics Doklady (English translation), 1963, 7(7): 595～596
4 Knuth DE. The Art of Comuter Programming. Vol. 2, Seminumerical Algorithms. 2nd edition. Addison-Wesley, 1981
5 Montgomery PL. Five, six, and seven-term Karatsuba-like formulae. IEEE Transaction on Computers, 2005, 54(3): 362～369
6 Schönhage A, Strassen V. Schnelle mutiplikation grosser zahlen. Computing, 1971, 7: 281～292
7 熊全淹. 线性代数. 第三版. 高等教育出版社, 1985. 164～172
8 GNU. GNU multiple precision arithmetic library. http://www.swox.com/gmp. 2005
9 Jebelean T. An algorithm for exact division. Journal of Symbolic Computation, 1993, 15: 169～180
10 Krandick W, Jebelean T. Bidirectional exact integer division. Journal of Symbolic Computation, 1996, 21: 441～455
11 Chung J, Hasan MA. http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-24.pdf, 2006

（上接第 43 页）

的更新，在进行关键字查找时必然影响查询的正确性并降低查找效率，因此 *successor* 对保证查询的正确性起着非常重要的作用。环中节点定期触发算法 stabilize（如图 5 所示）实现 *successor* 的及时更新，更改节点的 *successor* 的指针，更新 *finger* 表，既保证了查找的正确，又保证了查找的快速。

5 性能分析

我们在 CPU 为 AMD700MHz、内存为 256M 的机器上使用 SFS Chord 模拟器[13]进行了模拟测试。SFS Chord 模拟器的结构由两部分组成：协议实现部分（sim）和事件产生部分（traffic_gen）。协议实现部分用来实现各种 P2P 协议；事件产生部分由事件产生器产生各种事件，包括节点加入，文档插入，文档查找等事件。模拟实验中网络节点间的延时采用了 P2PSim[14]项目中测量的大约 2000 个网络 DNS 之间的 RTTs 矩阵，通过统计事件执行结果获取实验数据。

模拟实验主要验证 DeChord 相对于 Chord 所带来的查找效率的改进。在模拟过程中，提交相同数量的节点加入事件和文档查询事件，统计两种算法下每个文档的平均查询时间。这里文档的查询时间等于查询文档事件中路由路径所经过的节点之间的延时之和。

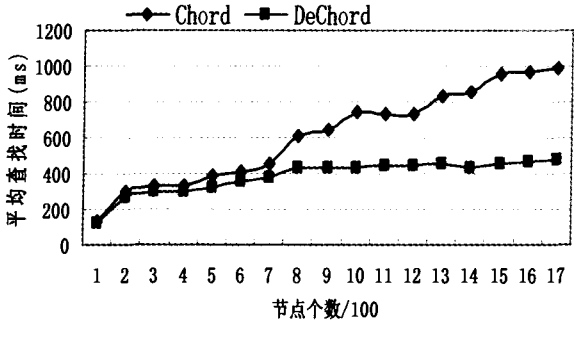

图 6 DeChord 与 Chord 的平均查找时间分布图

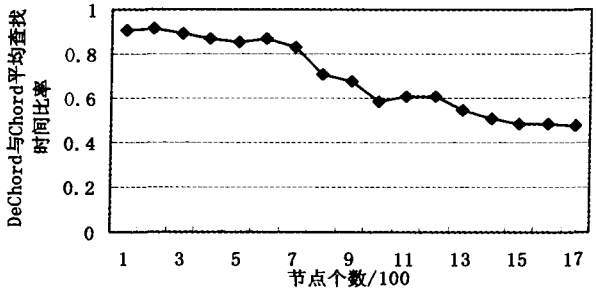

图 7 DeChord 与 Chord 的平均查找时间比率图

图 6 给出了 DeChord 与 Chord 在不同规模的环境下平均查找时间的分布情况，其中横坐标是节点个数，纵坐标是查找时间（ms）。从模拟试验结果可以看出来，网络节点数越多，DeChord 相对于 Chord 带来的改进就越明显。这是因为系统中加入的节点数越多，节点的 *finger* 表更新的次数就越多。由于每次更新都是淘汰 *finger* 表中距离较远的节点，导致 *fniger* 表中表项的后继节点中近距离节点占节点总数的比例越高，因此查找过程中节点都是选择距离较近的节点进行路由，减少了路由过程中的网络延时。

图 7 给出了 DeChord 与 Chord 的平均查找时间的比率图，其中横坐标代表网络节点个数，纵坐标代表 DeChord 与 Chord 的平均查找时间比率。从图中可以看出，比率曲线成一个逐渐下降的趋势，这也说明了与 Chord 相比，网络节点数越多，DeChord 改进的效果越明显，这也符合我们的算法设计目标。

结论 本文提出一种改进的 Chord 构建算法 DeChord，DeChord 考虑到节点之间的物理距离，系统节点总是选择距离自己物理距离较近的节点作为邻居节点，这种邻居节点的选择方式可以降低消息路由过程中每一跳的网络延时，从而降低了路由的总延时。模拟实验结果表明，与 Chord 相比，DeChord 算法能大幅度缩短数据定位延时，且网络节点越多，效果越明显。下一步的工作将针对节点失效处理进行进一步的研究，提出更优化的失效处理策略并进行模拟。

参 考 文 献

1 Napster. http://www.napster.com
2 Gnutella. http://gnutella.wego.com
3 FreeNet. http://freenet.sourceforge.net
4 Stoica I, Morris R, Karger D, Kaashoek M F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of SIGCOMM 2001, Aug. 2001
5 Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A Scalable Content-Addressable Network. In: Proceedings of SIGCOMM 2001, Aug. 2001
6 Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Available at: http://research.microsoft.com/antr/PAST/, 2001
7 FIPS 180-1. Secure Hash Standard. U. S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995
8 Dabek F, Kaashoek M F, Karger D, Morris R, Stoica I. Wide-area cooperative storage with CFS. SOSP'01, Banff, Canada, October 2001
9 Castro M, Druschel P, Hu Y C. Topology-aware routing in structured peer-to-peer overlay networks, Microsoft research technical report msr-tr-2002-82
10 Szymaniak M, Pierre G, van Steen M. Scalable Cooperative Latency Estimation
11 Ng T S E, Zhang Hui. A Network Positioning System for the Internet
12 Castro M, Druschel P, Hu Y C. Topology-aware routing in structured peer-to-peer overlay networks, Microsoft research technical report msr-tr-2002-82
13 SFS Chord simulator. http://pdos.lcs.mit.edu/cgi-bin/cvsweb.cgi/sfsnet/simulator
14 The p2psim project. http://www.pdos.lcs.mit.edu/p2p sim