一种层次结构化 P2P 网络中的负载均衡方法

张宇翔[1),2)] 张宏科[1)]

[1)](北京交通大学电子信息工程学院下一代互联网互联设备国家工程实验室 北京 100044)

[2)](中国民航大学计算机科学技术学院 天津 300300)

摘 要 相对于扁平结构化 P2P 网络,层次结构化 P2P 网络可利用稳定、高性能的超级节点提高 P2P 网络在动态环境下的性能.然而,超级节点的负载不均是层次结构化 P2P 网络面临的基本问题之一.对此,作者提出一种超级节点的负载均衡方法,通过分离超级节点负责的关键字空间和负责的叶子节点空间来为均衡负载提供条件,通过采用"力矩平衡原理"来实现兼顾均衡超级节点负责的叶子节点空间和查询请求负载.实验结果表明:在节点承载容量服从 Zipf 分布和查找请求服从正态分布或 Pareto 分布的环境下,负载均衡方法可使超级节点的负载达到较好的均衡,实现了用较少的超级节点承担较大的负载总量.

关键词 分布式散列表;Chord;层次结构化 P2P 网络;负载均衡

中图法分类号 TP393 **DOI 号**: 10.3724/SP.J.1016.2010.01580

A Load Balancing Method in Superlayer of Hierarchical DHT-Based P2P Network

ZHANG Yu-Xiang[1),2)] ZHANG Hong-Ke[1)]

[1)](*National Engineering Laboratory on Next Generation Internet Interconnection Devices, School of Electronics and Information Engineering, Beijing Jiaotong University, Beijing* 100044)

[2)](*College of Computer Science, Civil Aviation University of China, Tianjin* 300300)

Abstract Compared to flat DHT-based Peer-to-Peer (P2P) networks, hierarchical DHT-based P2P networks can use some stable and powerful peers (called superpeers) to achieve efficient lookup under highly dynamic environments. However, a crucial problem faced by all these networks is the load imbalance among superpeers. This paper proposes a novel load balancing algorithm, in which each superpeer, besides being responsible for the Chord identifier interval from its predecessor to it, maintains the linear leaf-peer interval within which its leaf peers fall. On this basis, "moment balance equation" is applied to finding a tradeoff point between balancing linear leaf-peer interval and balancing request loads for a superpeer. Analysis and simulation results show that the method can balance the load among superpeers in proportion to their capacity under Zipf capacities distribution and Gaussian or Pareto requests distribution.

Keywords DHT; Chord; hierarchical DHT-based P2P network; load balancing

1 引 言

在基于分布式散列表(Distributed Hash Table, DHT)的 P2P 网络(简称结构化 P2P 网络或 DHT 网络)中,所有的节点不论它们的计算能力和稳定性如何都承担着同样的功能角色,即在 Hash 空间上负责关键字的注册和查找,其中比较典型的系统有

收稿日期:2010-04-18;最终修改稿收到日期:2010-08-08.本课题得到国家自然科学基金(60833002,60776807)、北京市自然科学基金(4091003)和中央高校基本科研业务费专项资金(ZXH2010D016,ZXH2009A006)资助.**张宇翔**,男,1975 年生,博士研究生,副教授,主要研究方向为分布式计算、P2P 网络.E-mail: yxzhcn@sina.com.cn. **张宏科**,男,1957 年生,博士,教授,博士生导师,主要研究领域为新一代互联网理论体系.

Chord[1]、KAD[2]等. 然而，已有研究表明：(1) P2P 网络中节点的计算能力(包括存储空间、带宽及 CPU 性能等)具有很大差异性[3]；(2) 节点的稳定性也存在很大差异，在 P2P 网络中节点可随时、任意地加入或离开网络[4-6]. 随着 P2P 网络规模的增长，弱节点(指或计算能力差，或动态变化剧烈，或计算能力差且动态变化剧烈的节点)严重地制约了 P2P 网络的性能[4]. 为了克服该问题，研究者提出了许多层次 P2P 网络，其中典型的系统有 Chord²[7]、HIERAS[8]、Structured superpeers[9]和 Canon in G Major[10]等. 层次 P2P 网络的主要优点是：稳定的高性能节点被选出作为超级节点，在超级节点之间构建的上层 DHT 网络的维护代价和查找跳数相对于扁平的 DHT 网络均更小，且能有效应对 P2P 网络中的 churn 问题(churn 问题是指节点频繁地加入或离开对 P2P 网络性能造成的严重影响). 有关层次 P2P 网络的其它优点在文献[6-7]中有更具体的讨论.

层次 DHT 网络从逻辑上将节点划分为上、下两层. 下层的节点通常称为普通节点或叶子节点，所有的节点均先参与至下层，然后从普通节点中选出稳定的高性能节点作为超级节点，并按照 DHT 协议组织超级节点形成超级节点层(即上层). 每个超级节点除了承担上层 DHT 协议赋予的任务之外，还要管理一组叶子节点，并负责代表它的叶子节点实现关键字注册和查找. 层次 DHT 网络的组织结构有多种，它们的优缺点各不相同，相关介绍见第 2.1 节. 本文仅讨论第 2.1 节中图 3 所示结构的层次 DHT 网络，提出的负载均衡方法也仅针对该层次 DHT 网络，它的组织结构特点是：所有的节点在下层按照 DHT 协议组织，选出的超级节点在上层也按照 DHT 协议组织，最具代表性的系统是 Chord²，该结构的主要优点是当超级节点失效时它负责的叶子节点不会孤立，它的缺点是超级节点的负载不均问题较难解决.

在层次 DHT 网络中引起超级节点负载不均的主要原因有以下两个：(1) 因 Hash 函数导致的超级节点负责的 Hash 空间相差很大；(2) 因冷关键字和热关键字导致的查询请求量相差很大. 后者是导致超级节点负载不均直接的显性因素，而前者是导致负载不均的隐性因素，因为从长远来看，大的 Hash 空间蕴涵着较多的查询请求，而小的 Hash 空间蕴涵着较少的查询请求. 鉴于此，负载均衡的目标是均衡超级节点负责的 Hash 空间和负责的关键字查询请求量. 在此前的层次 DHT 网络中，超级节点负责的关键字空间和叶子节点空间是重叠的，统称为 Hash 空间，而本文为了方便实现超级节点的负载均衡特将超级节点负责的这两个空间分离开，故负载均衡的目标是均衡超级节点负责的叶子节点空间和负责的关键字查询请求量. 然而，均衡叶子节点空间与均衡查询请求量的是相互制约的，即不能同时实现均衡叶子节点空间和均衡查询请求量，因此需要对二者进行折中处理. 为此，本文引入了力学中的“力矩平衡原理”来实现该目标.

本文的主要工作集中在以下几个方面：(1) 设计将超级节点负责的关键字空间和负责的叶子节点空间分离的层次 DHT 系统，称其为 2Chord. 所有节点在下层由一个双向环结构组织起来，超级节点在上层由标准的 Chord 环结构组织起来. 每个超级节点除了负责环形的关键字区间之外，还负责一个线性的叶子节点区间. (2) 提出了适合于 2Chord 系统的关键字查找算法. (3) 提出了优先负载均衡、兼顾超级节点选择的负载均衡算法，以及优先超级节点选择、兼顾负载均衡的负载均衡算法. (4) 将两个负载均衡算法分别应用于 2Chord 和 2Chord′系统中. 2Chord′系统是指超级节点负责的关键字空间和叶子节点空间重叠的层次 DHT 系统，在此之前的绝大多数层次 DHT 网络都采用此种结构，引入该结构的目的是为了开展比对实验. (5) 给出仿真实验对结合不同负载均衡算法的各个系统的负载均衡性进行了对比，节点的承载容量服从 Zipf 分布且固定不变，查找请求量分别由正态分布和 Pareto 分布生成，变化的查询请求量作为输入用于分析系统的负载均衡性.

实验结果表明，在查询请求量极不平衡(服从 Pareto 分布)的环境下，2Chord 结合优先负载均衡、兼顾超级节点选择的负载均衡方法可使得超级节点获得较好的负载均衡. 同时还使得超级节点的数目较少，少的超级节点意味着小的查找跳数和小的网络维护代价.

第 2 节介绍相关工作；第 3 节提出 2Chord 系统，包括 2Chord 的组织结构、关键字查找算法和结合超级节点选择的负载均衡方法；第 4 节给出节点的动态维护机制；第 5 节给出仿真实验并对系统的负载均衡性能进行比对分析；第 6 节总结全文.

2 相关工作

2.1 层次 DHT 网络的组织结构

至今为止，研究者提出了许多层次 DHT 网络[6-10]. 这些网络从逻辑上将节点划分为上下两层. 超级节点位于上层，通常按照 DHT 协议组织. 在大多数层次 DHT 网络中，超级层按照照标准的 Chord 协议组织[6-8,10]，而在 Structured super-peers[9] 中超级层则按照全图(full graph)结构组织. 下层的组织结构可分为两大：一类是任意两个超级节点管理的叶子节点之间是不相交的. 文献[11]根据超级节点负责的组内叶子节点的不同组织结构将该大类结构细分为三小类，如图 1 和图 2 所示的结构和组内叶子节点全连接的结构，图中 *SP* 和 *P* 分别表示超级节点和叶子节点，在图 1 中每个叶子节点只与其超级节点连接，在图 2 中组内叶子节点按照 DHT 协议相互连接，文献[11]对上述 3 个结构的层次 DHT 网络的性能进行了详细分析和对比. 第二类是不同超级节点管理的叶子节点按照结构化的逻辑拓扑结构连接在一起，如图 3 所示的组织结构，最具代表性的系统为 Chord[2]，本文介绍的 2Chord 也属于此类. 相对于第一类层次 DHT 网络，第二类可有效应对当超级节点失效时它负责的叶节点孤立问题，但由于该结构对叶子节点的约束力很强，故很难有效地平衡超级节点的负载；相对于第二类，第一类由于该结构对叶子节点的约束力较弱，故相对容易平衡超级节点的负载，但会导致当超级节点失效时它负责的叶节点出现孤立. 本文仅针对第二类层次结构的负载均衡进行讨论.

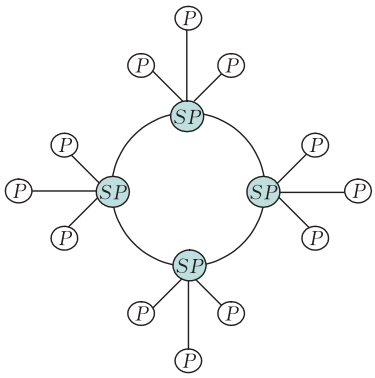


图 1 组内单连接的层次结构

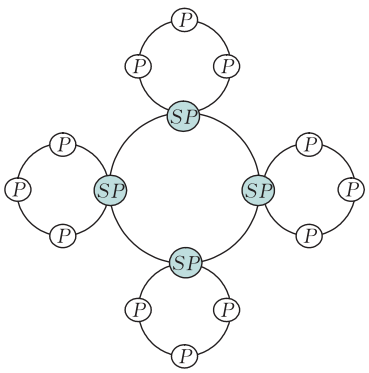


图 2 组内按 DHT 组织的层次结构

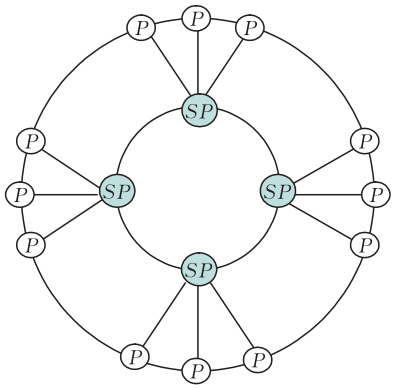


图 3 叶节点按 DHT 统一组织的层次结构

2.2 层次 DHT 网络的负载均衡

理想情况下的负载均衡是指节点根据它的承载容量按比例承担负载. 目前研究者提出了许多有关扁平 DHT 网络的负载均衡方法，而有关层次 DHT 网络的负载均衡方法还很少，文献[11-12]指出了层次 DHT 网络的负载均衡的重要性.

对于扁平 DHT 网络，较多的负载均衡方法采用虚拟服务器(virtual server). 最初由文献[13]提出，后来文献[14-19]对其进行了不断扩展和完善. 该方法的基本思想是通过虚拟服务器重新划分节点承载的 Hash 空间及关键字，其实质是通过节点频繁地加入和退出网络来达到各节点负载均衡. 该方法增加了网络的波动性，并导致网络维护代价增大.

扁平 DHT 网络的负载均衡方法不适合用于层次 DHT 网络，主要原因是：在解决层次 DHT 网络负载不均时，除了可将重负载节点的负载设法转移至轻负载节点之外，还可以通过增加适当的超级节点来解决负载不均，因此解决负载不均需要伴随设计合适的超级节点选择方式. Zoels 等人[12] 提出了针对层次 DHT 网络的负载均衡方法，该方法专门为图 1 所示的组内单连接的层次 DHT 网络设计，该方法的基本思想是：监测超级节点的负载；当普通节点请求加入时，负责将普通节点连接至轻负载的超级节点. 该方法不适合用于图 3 所示的上下两层都为 DHT 组织结构的层次 DHT 网络，因为在图 3 所示的结构中叶子节点的所归属的超级节点相对固

定，即叶子节点不能像在图 1 所示的结构中随意选择所归属的超级节点. 本文正是拟解决图 3 所示的上下两层都为 DHT 组织结构的层次 DHT 网络的负载不均问题.

3　2Chord 及负载均衡方法

3.1　2Chord 体系结构

图 4 给出了 2Chord 系统的基本体系结构，图中标识空间的规模为 128. 所有的节点都作为普通节点在下层按照双向环状结构组织（下层也可按照 Chord 结构组织，如在 Chord^2 中下层即为 Chord 结构；双向环状结构是 Chord 结构的简化，只是它的查找跳数大于 Chord 的，不影响讨论层次 DHT 网络的超级节点的负载均衡）. 从普通节点中选出承载容量大的节点作为超级节点，超级节点按照 Chord 结构组织. 在 2Chord 的上层，任意超级节点 u 负责的关键字空间 $R_k(u)$ 为 Chord 协议规定的标准环形空间，即 $R_k(u)=(w,\ u]$（设 w 是 u 在上层 Chord 环上的直接前驱）；它负责的叶子节点空间 $R_p(u)$ 为线性空间，即 $R_p(u)=[a,\ b]$（$0\le a\le u\le b$）. 而在此前的层次 DHT 网络中，每个超级节点负责的关键字空间和叶子节点空间是重叠的，即 $R_k(u)=R_p(u)=(w,u]$. 将超级节点关键字空间和叶子节点空间分离的最直接好处是便于均衡超级节点负责的叶子节点空间，更详细的讨论见第 3.4 节.

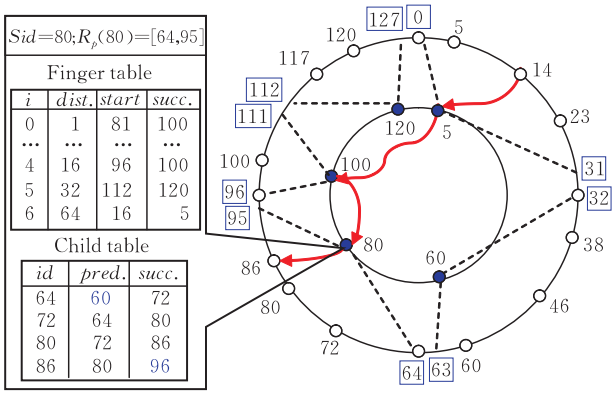


图 4　2Chord 的基本结构及查询示例（节点 14 查找关键字 84）

3.2　查找算法

在 2Chord 中，任意超级节点 u 维护 3 个信息表：(1) Chord 协议中的 finger 表（finger table）；(2) 管理的叶子节点所属的线性区间 $R_p(u)=[a,b]$（$0\le a\le u\le b$）；(3) 管理的叶子节点列表（child table）. 要求 u 的所有叶子节点的标识必须落入区间 $R_p(u)$ 内. 设 l 是 u 的第 i（$1\le i\le l_u$，l_u 是 u 的叶子节点的数目）个孩子，为了方便后面算法描述将其记作 $u.child[i].id$. 在孩子列表中，第 i 个条目包括 u 的叶子节点 l、l 的直接前驱 $l.predecessor$ 和直接后继 $l.successor$. 图 4 右侧给出了超级节点 80（$Sid=80$）维护的信息表. 节点 l 请求它的超级节点 u 查找关键字 id 的负责节点 $successor(id)$ 的步骤如下：

1. l 将关键字查询请求直接发送给它的超级节点 u；

2. 如果 $id\in[u.child[1].predecessor, u.child[l_u].successor]$，则 u 在其孩子列表中查询 id 的负责节点 $successor(id)$，并将查询结果返回给节点 l；

3. 否则，根据 Chord 协议的查询算法，超级节点 u 在上层 Chord 结构中查找 id 的负责节点 v；

3.1. 如果 $id\in[v.child[1].predecessor, v.child[l_v].id]$，则 v 在它的孩子列表中查询 id 的负责节点 $successor(id)$，并将查询结果返回给节点 l；

3.2. 否则，v 将该查询请求发送给它的直接后继超级节点 w，w 在它的孩子列表中查询 id 的负责节点 $successor(id)$，并将查询结果返回给节点 l.

例如，在图 4 中节点 14 查询关键字 84 的负责节点 86 的查询过程如下：

节点 14 将查询请求直接发送给它的超级节点 5. 因关键字 84 并没有落在环形区间 120（它是节点 0 的直接前驱）和 32（它是节点 23 的直接后继）之间，故超级节点 5 在上层按照 Chord 协议的查找算法找到关键字 84 的负责超级节点 100. 因关键字 84 并没有落在环形区间 86（它是节点 96 的直接前驱）和 100（它为 $100.child[l_{100}].id$）之间，故超级节点 100 将查找请求发送给它的直接后继超级节点 80. 超级节点 80 完成查询请求将查询结果（关键字 84 的负责节点 86）直接返回给查询发出节点 14.

就查找性能和网络维护代价而言，层次 DHT 网络明显优于扁平 DHT 网络. 设 S 和 N 分别为超级节点和普通节点的数目，超级层和普通节点层的查找跳数分别为 $O(\log S)$ 和 $O(\log N)$，因 S 通常明显小于 N，故层次 DHT 网络的查找性能明显优于扁平 DHT 网络的. 另外，因超级节点通常较普通节点稳定得多，加之较少的超级节点数目，故层次 DHT 网络的维护代价也明显低于扁平 DHT 网络的.

3.3　负载均衡与超级节点选择

在 2Chord 中，当超级节点的负载达到它的承载容量的上限时，它将从其的叶子节点中选出承载容量大的节点作为新超级节点加入超级层，新超级节点分担相应的负载，以达到减轻重负载节点的负载. 超级节点的理想选择标准是选出承载容量最大

的节点作为超级节点.然而,很难做到既实现超级层负载均衡又保证每次选出承载容量最大的节点作为新超级节点.也即,当尽力均衡超级层的负载时,就很难保证新选出的超级节点的承载容量最大;而当每次选出承载容量最大的节点作为新超级节点时,就很难实现均衡超级层的负载.这里给出两种解决方案:(1)优先实现负载均衡,兼顾选择承载容量尽可能大的节点作为超级节点;(2)优先选择承载容量大的节点作为超级节点,兼顾实现负载均衡.

方案 1:优先负载均衡,兼顾超级节点选择

设超级节点 u 负载的叶子节点集合为 $R'_p(u)=\{p_1,p_2,\cdots,p_e\}$ (e 是叶子节点的数目),叶子节点 p_i 的查询负载为 $rp(p_i)$,它的承载容量为 $c(p_i)$.超级节点 u 需从它的叶子节点中选出新超级节点 v.优先负载均衡、兼顾超级节点选择对于超级节点 u 意味着:先在 u 负责的叶子节点区间 $R_p(u)=[a,b]$ 上寻找合适的负载均衡分割点 m,然后以 m 为分界点在没有 u 的区间选择新的超级节点 v.

该方案将力学中的"力矩平衡原理"用于解决超级层的负载不均问题."力矩平衡原理"可有效地对均衡叶子节点空间和均衡查询请求量进行折中.负载均衡分割点 m 的计算公式如下:

$$\sum_{i=1}^{e}(m-p_i)rp(p_i)=0,$$

其中 $m-p_i$ 为分割点 m 到节点 p_i 的距离(见图 5).解方程可得:

$$m=\lfloor \sum_{i=1}^{e}rp(p_i)p_i/\sum_{i=1}^{e}rp(p_i)\rfloor.$$

其中符号 $\lfloor\ \rfloor$ 为取下整,即 $\lfloor x\rfloor=\max\{n\in Z|n\leq x\}$.

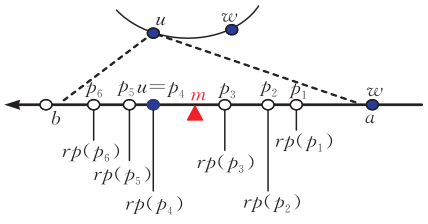


图 5 方案 1 中负载均衡分割点 m 计算示例

从超级节点 u 的叶子节点中选出新超级节点 v 并在 u,v 之间重新分配关键字空间和叶子节点空间的算法见算法 1.

算法 1. Select_Superpeer_1(u,$[a,b]$).

```
1. 在区间[a,b]上计算负载均衡分割点 m;
2. if (m==u)        //这种情况意味着 u 负责的节点
                    //只有自己
3.    return {超级节点选择失败};
4. else //2
5.    if (u∈[m,b]) // u 落在区间[m,b]上
6.       在区间[a,m]上选择处理容量大的节点 p;
                    //计算区间[a,m]所有节点的负载总和
7.       Lp=Calculate_Total_Loads([a,m]);
                    //计算区间[m,b]所有节点的负载总和
8.       Lu=Calculate_Total_Loads([m,b]);
                    //p 的承载超过它的容量或 u 的承载超
                    //过它的容量
9.       if (c(p)<Lp ‖ c(u)<Lu)
                    //在区间[a,m]上重新选择承载能力
                    //强的节点
10.         Select_Superpeer_1(p,[a,m]);
11.      else //9
12.         v=p;   //p 为选出的新超级节点,记为 v
                    //重新分配两个超级节点负责的关键字
                    //区间和叶子节点区间,并将其返回
13.         return {v; Rk(v)=(w,v]; Rp(v)=[a,m];
                    Rk(u)=(v,u]; Rp(u)=[m,b];}
14.   else //5,u 落在区间[a,m]上
15.      在区间[m,b]上选择处理容量大的节点 p;
16.      Lp=Calculate_Total_Loads([m,b]);
17.      Lu=Calculate_Total_Loads([a,m]);
18.      if (c(p)<Lp ‖ c(u)<Lu)
19.         Select_Superpeer_1(p,[m,b]);
20.      else //18
21          v=p;
22          return {v; Rk(v)=(u,v]; Rp(v)=[m,b];
                    Rk(u)=(w,u]; Rp(u)=[a,m];}
```

方案 1 可有效均衡超级层的负载,这一优点在随后的仿真实验中会得到验证.它的缺点是不能保证新超级节点 v 的承载容量是 u 的所有叶子节点中最大的,也即承载容量最大的节点可能没有被选出来作为新超级节点.例如,当 u 落在区间 $[m,b]$ 上时,算法 1 规定新超级节点要在区间 $[a,m]$ 上选出,如果承载容量最大的节点落在区间 $[m,b]$ 上时,那么新选出的节点的承载容量一定非最大.为了克服该缺点提出了方案 2.

方案 2:优先超级节点选择,兼顾负载均衡

该方案对于超级节点 u 意味着:首先在 u 负责的叶子节点区间 $R_p(u)$ 上选出新的超级节点 v,然后将区间 (u,v) 或 (v,u) 上的普通节点根据 u、v 的负载均衡情况分别分配给两者,也即在区间 (u,v) 或 (v,u) 上寻找合适的负载均衡分割点 m(见图 6).从超级节点 u 的叶子节点中选出新超级节点 v 并在 u、v 之间重新分配关键字空间和叶子节点空间的算法见算法 2.

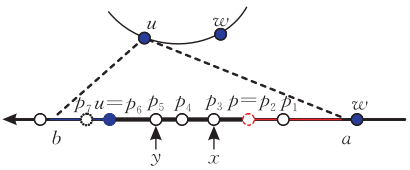


图 6　方案 2 中区间(p,u)上的节点分配示例

在算法 2 中，当承载容量大的节点 p（不妨设 $p<u$）选出之后，分别计算区间$[a,p]$和$[u,b]$上的普通节点的负载总量，如果它们分别小于 p 和 u 的承载容量，接下来需将区间(p,u)上的普通节点根据负载均衡目标分别分配给 u 和 p，具体分配方法见算法 3. 在该算法中用到节点容量利用率度量指标，定义如下.

定义 1（节点容量利用率）. 节点容量利用率定义为节点的负载除以它的承载容量，简称为节点利用率，那么节点 u 的利用率定义为 $ul(u)=rp(u)/c(u)$.

另外，设区间(p,u)上的普通节点存储于数组 $DP[0,\cdots,k-1]$中，k 为节点数目. 算法 3 的基本思想如下：

比较 p 和 u 的利用率，如果 p 的利用率小于 u 的利用率，试将 p 左边的普通节点（在图 6 中从 p_3 起）分配给 p；否则，试将 u 右边的普通节点（在图 6 中从 p_5 起）分配给 u，直到(p,u)上的普通节点分配完为止. 在每次分配节点时，分配成功的条件是待加入节点的负载与当前超级节点的负载总和小于当前超级节点的承载容量.

算法 2.　Select_Superpeer_2(u,$[a,b]$).

```
1. for (i=0; i<e; i++)//e 是 u 叶子节点的数目
2.    从 u 的叶子节点中选出承载容量最大节点 p;
3.    if (p<u)
4.       Lp=calculate total loads([a,p]);
5.       Lu=calculate total loads([u,b]);
         //p 和 u 的负载都小于各自的承载容量
6.       if (Lp<c(p) && Lu<c(u))
         //将区间(u,p)区间上的负载按照
         //负载均衡分配给超级节点 u 和 p
7.       Distribute_Local_Peers((p,u));
8.    else //3
9.       Lp=calculate total loads([p,b]);
10.      Lu=calculate total loads([a,u]);
11.      if (Lp<c(p) && Lu<c(u))
12.      Distribute_Local_Peers((u,p));
```

算法 3.　Distribute_Local_Peers((p,u)).

```
1. x=0; y=k-1; //k 为区间(p,u)上普通节点数目
2. while (x≠y)
3.    if (ul(p)<ul(u))
4.       if (rp(DP[x])+Lp<c(p));
5.          x++; Lp=Lp+rp(DP[x]);
6.       else //4
7.          Lz=Calculate_Total_Loads([x,y]);
8.          if (Lz+Lu<c(u))
9.             v=p; m=x;
10.            return {v; Rk(v)=(w,v];
               Rp(v)=[a,m);Rk(u)=(v,u];
               Rp(u)=[m,b];}
11.         else //8
12.            return {超级节点选择失败};
13.   else //3
14.      if (rp(DP[y])+Lu<c(u));
15.         y--; Lu=Lu+rp(DP[y]);
16.      else //14
17.         Lz=Calculate_Total_Loads([x,y]);
18.         if (Lz+Lp<c(p))
19.            v=p; m=y;
20.            return {v; Rk(v)=(w,v];
               Rp(v)=[a,m]; Rk(u)=(v,u];
               Rp(u)=(m,b];}
21.         else //18
22.            return {超级节点选择失败};
```

算法 4.　Select_Superpeer_4(u,$(w,u]$).

```
1. for (i=0; i<e; i++)//e 是 u 叶子节点数目
2.    从 u 的叶子节点中选出承载容量最大节点 p;
3.    Lp=calculate total loads((w,p]);
4.    Lu=calculate total loads((p,u]);
5.    if (Lp<c(p)&&Lu<c(u))//负载小于承载容量
6.       v=p;
7.       return {v; Rk(v)=Rp(v)=(w,v];
                    Rk(u)=Rp(u)=(v,u];}
```

尽管方案 2 的初衷是确保将承载容量最大的节点选出，然而承载容量最大的节点未必能为它的超级节点分担适当的负载. 例如在图 6 中若绝大部分负载来源于区间$[u,b]$，然而承载容量最大的节点 p 并不能为其分担适当的负载，因此节点 p 本次并不会被选作新的超级节点. 它的另一个缺点是可均衡的负载为区间(u,v)或(v,u)上的负载，由于该区间较小，因此该方案很难达到好的负载均衡效果.

上述两个方案如遇特殊情况可能会出现超级节点选择失败，可借助文献[20]提出的节点负载统计视图，在超级层将热关键字相关信息缓存至负载轻的超级节点或者将其反向缓存至发出查询请求的超级节点.

3.4　2Chord 与 2Chord′的负载均衡对比

在 2Chord 中，超级节点负责的关键字空间和

叶子节点空间不相同.而在此前的层次 DHT 网络中每个超级节点负责的关键字空间和叶子节点空间是重叠的,即 $R_k(u)=R_p(u)=(w,u]$(设 w 是 u 在上层 Chord 环上的直接前驱).为了便于比对实验,本文将这类系统称为 2Chord′.

算法 1 可直接用于 2Chord′中实现负载均衡和超级节点选择,然而算法 2 并不能直接应用于 2Chord′中,需要对其简单修改,修改后的优先选择超级节点同时兼顾负载均衡算法见算法 4(见图 7).在 2Chord′结合算法 4 中,没有专门的负载均衡控制条件来选择超级节点,p 能成为新超级节点 v 的唯一条件是 p 和 u 的负载总量同时小于它们各自的承载容量.另外,2Chord′结合算法 4 可能会出现选出的新超级节点不能分担原超级节点的负载.如在图 7 中,当新超级节点 v 为 w 的直接后继或为 u 的直接前驱时,负载均衡无法实现,不妨设 v 为 w 的直接后继,那么 u 负责区间 $(v,u]$ 而 v 负责区间 $[v]$.然而这种极端的情况在 2Chord 中不会出现.

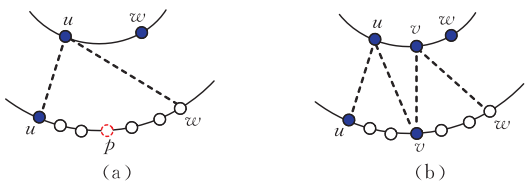


图 7 2Chord′中超级节点选择过程示例

表 1 2Chord 和 2Chord′的负载均衡对比表

	实施负载均衡的叶子节点空间	是否兼顾均衡 DHT 空间	优先负载均衡还是超级节点选择
2Chord 结合算法 1	$[a,b]$(见图 5)	是	负载均衡
2Chord 结合算法 2	(u,v)或(v,u)(见图 6)	否	超级节点选择
2Chord′结合算法 1	$(w,u]$(见图 7)	是	负载均衡
2Chord′结合算法 4	无(见图 7)	否	超级节点选择

表 1 从实施负载均衡的叶子节点空间、是否兼顾均衡 DHT 空间和优先负载均衡还是优先超级节点选择 3 个方面对 2Chord 和 2Chord′结合不同算法进行了总结.从上述分析初步可知,最好的负载均衡解决综合方案是 2Chord 结合算法 1,而最差的是 2Chord′结合算法 4.更具体定量的仿真实验比对详见第 5 节.

4 节点动态维护机制

超级节点加入. 在 2Chord 中,当超级节点的负载达到它的承载容量的上限时,超级节点根据本文前面给出的超级节点选择算法从它的叶子节点中选出一个新的超级节点,并将相应的负载分配给新超级节点.设超级节点 u 从它的叶子节点中选出新超级节点 v,那么 v 加入超级层的步骤如下:

1. 初始化 v 在超级层上的直接前驱、直接后继和 Chord 协议中的 finger 表,其中 v 的直接前驱和直接后继可直接从 u 处获得,而 finger 表中的其它条目可由 Chord 协议的查找算法[1]完成.

2. 根据超级节点选择算法调整超级节点 u 和 v 负责的关键字空间和叶子节点空间,并调整位于叶子节点空间内的叶子节点.

3. 超级节点 v 通告它的叶子节点将它们的负责节点由 u 调整为 v.

其它超级节点的 finger 表的更新由 Chord 协议的稳定算法[1]完成.

普通节点加入. 普通节点加入先要获得一个启动节点,然后启动节点按照文中给出的查找算法查找负责待加入节点的超级节点,超级节点安排待加入节点成为它的叶子节点.

超级节点退出. 尽管超级节点比较稳定,但是它也会失效或有计划地离开 P2P 网络.超级节点退出会导致与该超级节点相连的连接失效,失效的连接包括超级节点之间的连接以及超级节点与其叶子节点之间的连接.超级层可以通过 Chord 稳定算法重新调整超级节点之间的相互连接,而失效超级节点负责的叶子节点区间必须合并至失效节点的直接后继超级节点或者直接前驱超级节点.设超级节点 v 失效,它的直接前驱和直接后继超级节点分别为 $predecessor(v)$ 和 $successor(v)$.叶子节点区间合并方法如下:

1. 如果 v 的标识在所有超级节点中是最大的,那么 v 负责的叶子节点区间需归至它的直接前驱.它的直接前驱负责的叶子节点区间调整为 $R_p(predecessor(v))=R_p(v)\cup R_p(predecessor(v))$,同时将 v 负责的叶子节点也调整给它的直接前驱;

2. 否则,v 负责的叶子节点区间归至它的直接后继,它的直接后继负责的叶子节点区间调整为 $R_p(successor(v))=R_p(v)\cup R_p(successor(v))$,同时将 v 负责的叶子节点也调整给它的直接后继.

如果合并后的超级节点的负载过重或超出它的承载容量,则可根据本文前面给出的超级节点选择算法选出新超级节点为其分担负载.

普通节点退出. 超级节点周期性地探测它的叶子节点,如果超级节点发现它的某个叶子节点退出,那么该超级节点将其叶子列表中的退出叶子节点删除,通告退出叶子节点的前后邻居叶子节点,并

要求它们更新各自的邻居叶子节点.

5　仿真实验及负载均衡分析

5.1　实验配置及负载均衡度量指标

设标识空间的规模为 1024，普通节点的数目 $n=1024$，即节点数目达到饱和. 假设系统运行达到稳定状态. 为了反映节点承载容量的异质性特征，使用 Zipf 分布产生节点的承载容量(在文献[19,21]中，该分布用于产生 P2P 节点的承载容量). 在本实验中，任意节点 i 的承载容量 $c_i=1000i^{-\beta}$，其中 $\beta=1.2$. 系统的承载总容量为 $C=c_1+c_2+\cdots+c_n$.

节点的负载由两个不同类型的随机分布产生：正态分布 $N(\mu,\sigma^2)$ 和 Pareto 分布(在文献[19]中，这两个分布用于产生节点的负载). 在本实验中，正态分布的均值 $\mu=[0.3:0.05:0.8]C/n$，它以平均承载容量 C/n 为参考值，其中[0.3:0.05:0.8]表示从 0.3～0.8 每隔步长 0.05 取一个值，并且包括两个端点，共取到 11 个值；它的标准差 $\sigma=0.1\mu$. 标准 Pareto 分布的密度函数为 $f(x)=ab^a/x^{a+1}$ ($a,b>0$; $b\leq x<\infty$)，其中 a 为形状参数，b 为规模参数，$b=\mu(a-1)/a$(可由 Pareto 分布的期望计算公式 $\mu=ab/(a-1)$[22]得到). 在本实验中 $a=1.5$，Pareto 分布的均值的取值与正态分布的相同，它的标准差是无穷大. 因 Pareto 分布是偏态的重尾分布，故它可以反映出网络波动严重(churn)情况下的负载分布情况[16]. 设节点 i 的负载为 f_i，那么系统的总负载 $F=f_1+f_2+\cdots+f_n$.

为了确保每个节点的承载容量大于它的负载，重新分配节点的承载容量. 节点 i 的承载容量重新赋值为 $c_i=c_i+5\max\{f_i\mid 1\leq i\leq n\}$，该变换为线性变化，它保留 Zipf 分布的性质.

为了衡量系统负载均衡的性能，引入 3 个度量指标，其中超级节点容量利用率见定义 1，其它两个定义如下.

定义 2(超级节点所占最小比例).　超级节点所占最小比例是指承载给定的系统总负载所需的最少超级节点数目与总节点数目的比例，即定义为 $\alpha_{\min}=\min(\alpha)$，其中 $\alpha=n_s/n$ 为超级节点数目 n_s 占所有节点数目 n 的比例. 为此规定：超级节点的负载在没有达到它的承载容量之前不能选择新的超级节点为其分担负载.

定义 3(超级层利用率).　超级层利用率是指负载总量与超级层的承载总容量的比值(F/C).

5.2　超级节点所占最小比例

在仿真过程中，节点的承载容量一次分配固定不变，而节点的负载是变量，对于每个固定的负载均值 μ，分别用两个分布各产生 5 组相应的负载数据，这些数据分别用于 2Chord 结合算法 1 和算法 2 以及 2Chord′结合算法 1 和算法 4 中. 下面各负载均衡度量指标的值没有特别说明都指均值.

图 8 给出了 2Chord 和 2Chord′中正态分布负载的超级节点所占最小比例，图 9 给出了 2Chord 和 2Chord′中 Pareto 分布负载的超级节点所占最小比例. 从图中可知：(1) 在给定负载总量下，超级节点所占最小比例由小到大分别是 2Chord 结合算法 1、2Chord 结合算法 2、2Chord′结合算法 1 和 2Chord′结合算法 4. 该数据意味着，如果承载相同的负载总量，2Chord 结合算法 1 需要很少的超级节点，而 2Chord′结合算法 4 需要很多的超级节点. (2) 无论是 2Chord 还是 2Chord′，算法 1 可获得更好的负载均衡效果，而算法 2 和算法 4 的负载均衡性较差(算法 4 仅为算法 2 的简单形式). (3) 无论是算法 1 还是算法 2，随着负载总量的增加，2Chord 的超级节点

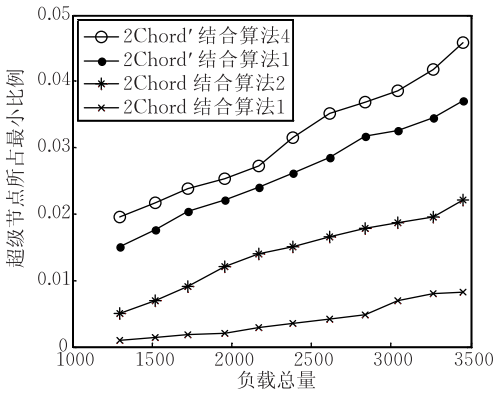


图 8　Chord 和 2Chord′中正态分布负载的超级节点所占最小比例

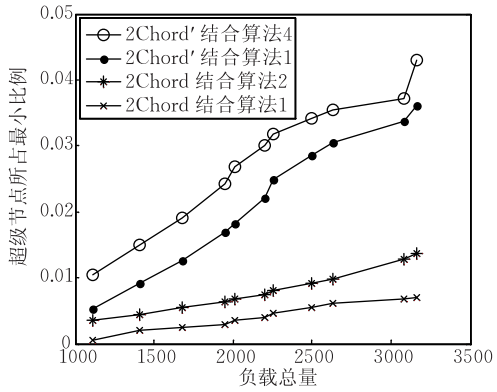


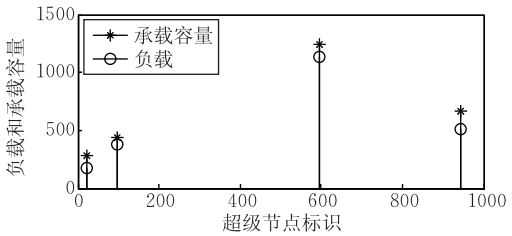
图 9　Chord 和 2Chord′中 Pareto 分布负载的超级节点所占最小比例

所占最小比例缓慢增长，而 2Chord′的超级节点所占最小比例明显快速增长. 上述分析基本上可以说明超级节点负载均衡效果由好至差分别为：2Chord 结合算法 1、2Chord 结合算法 2、2Chord′结合算法 1 和 2Chord′结合算法 4.

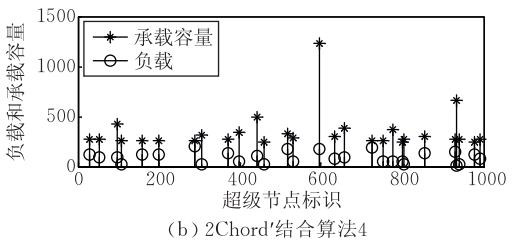
5.3 超级节点利用率

本节首先比对 2Chord 和 2Chord′的超级节点容量的利用率. 随机选取一组实验结果，如选取 2Chord 和 2Chord′承担同样的负载总量 2615.7（即均值 $\mu=0.6\times4.2399$），节点的负载由 Pareto 分布产生.

图 10 给出了在 Pareto 分布产生负载情况下 2Chord 结合算法 1（最好负载方案，见图 10 上面）和 2Chord′结合算法 4（最差负载方案，见图 10 下面）的超级节点容量利用率的直观图. 横坐标为节点标识（ID），纵坐标为负载和承载容量. 该图可以直观地反映出节点的承载容量和负载之间的差距. 由图 10 可知：(1) 在 2Chord 结合算法 1 中，超级节点容量的利用率很高，如超级节点（$ID=594$）的容量为 1228，它的负载为 1149，它的利用率约为 0.94；而在 2Chord′结合算法 4 中，超级节点的利用率很低，同样如超级节点（$ID=594$）的负载仅为 178，它的利用率约为 0.14. (2) 对于同样的负载总量，2Chord 结合算法 1 中超级节点数目为 4 个，而在 2Chord′结合算法 4 中超级节点数目为 28 个. 少的超级节点数目意味着小的查找跳数和低的网络维护代价.



(a) 2Chord结合算法1



(b) 2Chord′结合算法4

图 10 Pareto 分布负载的超级节点的负载和承载容量

图 11 给出了负载总量与超级层的平均利用率的变化关系. 从图 11 可知：(1) 2Chord 结合算法 1 和算法 2 的超级层的平均利用率分别约为 0.9 和 0.8，而 2Chord′结合算法 1 和算法 4 的超级层的平均利用率分别约为 0.4 和 0.2，由此可见 2Chord 和 2Chord′的超级层利用率之间的差距很大. (2) 无论是 2Chord 还是 2Chord′，Pareto 分布负载的超级层的平均利用率要低于正态分布的超级层的平均利用率，这验证了服从 Pareto 分布的负载对应请求量极不平衡.

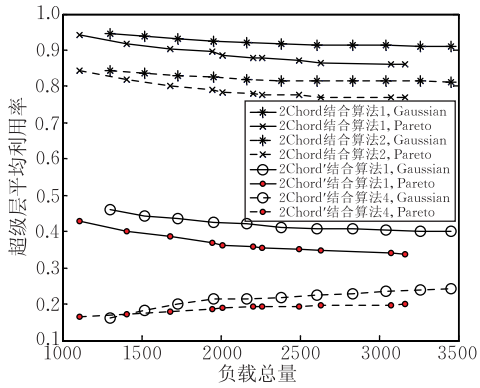


图 11 超级层平均利用率

6 结 论

本文针对叶子节点按照结构化的逻辑拓扑结构连接在一起的层次 DHT 网络（如图 3 所示的 DHT 网络），解决了它的超级节点的负载不均问题. 率先采用了分离超级节点负责的关键字空间和负责的叶子节点空间的技术（用于 2Chord 系统），该技术为均衡超级节点的负载提供了前提条件. 在此基础上，采用了优先均衡负载、兼顾超级节点选择的设计思想，将“力矩平衡原理”应用于解决均衡超级节点负责的叶子节点空间与均衡查询请求之间的折中（由算法 1 完成）. 在节点承载容量服从 Zipf 分布和查找请求服从 Pareto 分布（代表查询请求量极不平衡）的环境下，2Chord 结合算法 1 可使超级节点的负载达到较好的均衡，超级层的利用率约为 0.9，从而用较少的超级节点承担了较大的负载总量.

致 谢 非常感谢审稿人提出的修改意见，这些修改意见对提高论文水平有很大帮助！

参 考 文 献

[1] Stoica I, Morris R, Karger D, Kaashoek M F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications//Proceedings of the ACM SIGCOMM. San Diego, USA, 2001: 149-160

[2] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer in-

formation system based on the XOR metric//Proceedings of the International Workshop on Peer-to-Peer Systems. Cambridge, USA, 2002: 53-65

[3] Saroiu S, Gummadi P K, Gribble S D. A measurement study of peer-to-peer File sharing systems//Proceedings of the Multimedia Computing and Networking. San Jose, USA, 2002: 156-170

[4] Godfrey P B, Shenker S, Stoica I. Minimizing churn in distributed systems//Proceedings of the ACM SIGCOMM. Pisa, Italy, 2006: 147-158

[5] Krishnamurthy S, El-Ansary S, Aurell E, Haridi S. An analytical study of a structured overlay in the presence of dynamic membership. IEEE Transactions on Networking, 2008, 16(4): 814-825

[6] Garces-Erice L, Biersack E W, Ross K W, Felber P A, Urvoy-Keller G. Hierarchical peer-to-peer systems. Parallel Processing Letters, 2003, 13(4): 643-657

[7] Joung Y J, Wang J C. Chord^2: A two-layer Chord for reducing maintenance overhead via heterogeneity. Computer Networks, 2007, 51(3): 712-731

[8] Xu Z, Min R, Hu Y. HIERAS: A DHT based hierarchical P2P routing algorithm//Proceedings of the International Conference on Parallel Processing. Kaohsiung, Taiwan, China, 2003: 187-194

[9] Mizrak A T, Cheng Y, Kumar V, Savage S. Structured superpeers: Leveraging heterogeneity to provide constant time lookup//Proceedings of the IEEE Workshop on Internet Applications. San Jose, USA, 2003: 104-111

[10] Ganesan P, Gummadi K, Garcia-Molina H. Canon in G Major: Designing DHTs with hierarchical structure//Proceedings of the 24th International Conference on Distributed Computing Systems. Tokyo, Japan, 2004: 263-272

[11] Zöls S, Despotovic Z, Kellerer W. On hierarchical DHT systems-an analytical approach for optimal designs. Computer Communications, 2008, 31(3): 576-590

[12] Zöls S, Despotovic Z, Kellerer W. Load balancing in a hierarchical DHT-based P2P system//Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing. New York, USA, 2007: 353-361

[13] Rao A, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in structured P2P systems//Proceedings of the 2nd International Workshop Peer-to-Peer Systems. Berkeley, USA, 2003: 68-79

[14] Godfrey B, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in dynamic structured P2P systems//Proceedings of the IEEE INFOCOM. Hong Kong, China, 2004: 2253-2262

[15] Godfrey B, Stoica I. Heterogeneity and load balance in Distributed Hash Tables//Proceedings of the IEEE INFOCOM. Miami, USA, 2005: 596-606

[16] Ledlie J, Seltzer M. Distributed, secure load balancing with skew, heterogeneity, and churn//Proceedings of the IEEE INFOCOM. Miami, USA, 2005: 1419-1430

[17] Wang X, Zhang Y, Li X, Loguinov D. On zone-balancing of Peer-to-Peer networks: Analysis of random node join//Proceedings of the ACM SIGMETRICS. New York, USA, 2004: 211-222

[18] Chen C, Tsai KC. The server reassignment problem for load balancing in structured P2P systems. IEEE Transactions Parallel and Distributed Systems, 2008, 19(2): 234-245

[19] Zhu Y, Hu Y. Efficient, proximity-aware load balancing for DHT-based P2P systems. IEEE Transactions Parallel and Distributed Systems, 2005, 16(4): 349-361

[20] Xiong Wei, Xie Dong-Qing, Jiao Bing-Wang, Liu Jie. Self-adaptive load balancing method in structured P2P protocol. Journal of Software, 2009, 20(3): 660-670(in Chinese)
(熊伟，谢冬青，焦炳旺，刘洁. 一种结构化 P2P 协议中的自适应负载均衡方法. 软件学报，2009, 20(3): 660-670)

[21] Lu Q, Ratsnasamy S, Shenker S. Can heterogeneity make Gnutella scalable? //Proceedings of the 1st International Workshop on Peer-to-Peer Systems. Cambridge, USA, 2002: 94-103

[22] Arnold B C. Pareto Distribution. Fairland, Maryland: International Co-Operative Publishing House, 1983

ZHANG Yu-Xiang, born in 1975, Ph. D. candidate, associate professor. His current research interests include distributed computing and peer-to-peer networks.

ZHANG Hong-Ke, born in 1959, Ph. D., professor, Ph. D. supervisor. His current research interests focus on next generation network.

Background

This research is partly supported by the National Natural Science Foundation of China under grant Nos. 60833002, 60776807, the Beijing Natural Science Foundation under grant No. 4091003 and by the Special Fund of

Basic Scientific Research of Central Colleges under grant Nos. ZXH2010D016, ZXH2009A006.

In flat DHT-based Peer-to-Peer (P2P) networks, as the network size increases, weak peers will seriously limit the performance of DHT networks in a churn environment. To address this problem, several hierarchical DHT-based P2P networks have been proposed so far. However, a crucial problem faced by all these networks is the load imbalance among superpeers. These hierarchical DHT networks are mostly organized into two layers: a superlayer built by superpeers, and a leaf-peer layer built by all peers. The superlayer is mostly implemented using a DHT algorithm such as Chord Protocol. At the same time, each superpeer manages a group of leaf peers, and is responsible for delivering queries on behalf of the leaf peers in its group.

This paper presents a novel load balancing algorithm for two-tier DHT system with a connection leaf-peer layer, which we call 2Chord (2-teir Chord). In 2Chord, the basic idea of the load balancing algorithm is that each superpeer, besides being responsible for the circular identifier interval from its identifier to its predecessor's identifier (this is the Chord identifier interval), maintains the linear leaf-peer interval which is different from the former. On this basis, "moment balance equation" is applied to finding a tradeoff point between balancing linear leaf-peer interval and balancing request loads for a superpeer. Analysis and simulation results show that the method can balance the load among superpeers in proportion to their capacity.