一种基于对等网络的云资源定位算法*

李 璞[a]，陈世平[b]，李剑锋[c]
（上海理工大学 a. 光电信息与计算机工程学院；b. 网络与信息中心办公室；c. 经济与管理学院，上海 200093）

摘 要：提出用分布式哈希表（DHT）为每台云服务器产生一个唯一的节点编号，该编号作为网络拓扑结构、检索信息存储和信息查询共同的标志符，从而形成一个适合分布式计算的结构化 P2P 覆盖网。设计了新的拓扑和路由协议来解决云资源的常数跳定位问题。仿真实验表明，经典的 P2P 算法平均查找跳数与网络规模成正相关，无法依据云计算的实际需要人为地控制查找跳数；该算法的平均查找跳数与网络规模无关，随着网络规模的增大而趋向于设定值，可以解决云资源的常数跳定位问题。

关键词：云资源；云对等网络；拓扑；路由；资源定位；平均查找跳数
中图分类号：TP393 **文献标志码**：A **文章编号**：1001-3695(2013)02-0570-04
doi:10.3969/j.issn.1001-3695.2013.02.070

Cloud resources locating algorithm based on peer-to-peer network

LI Pu[a], CHEN Shi-ping[b], LI Jian-feng[c]
(*a. School of Optical-Electrical & Computer Engineering, b. Network & Information Center Office, c. Business School, University of Shanghai for Science & Technology, Shanghai* 200093, *China*)

Abstract: This paper proposed a method that using distributed hash table (DHT) to generate a unique node ID for each cloud server. The ID represented the common identifier for the network topology, the retrieval information storage and query. At last built a structure P2P overlay network for distributed computing. This paper designed a new topology and routing protocol to solve the cloud resources' constant jump locating problem. Simulation results show the classic P2P algorithms' average search hops is positively correlated to the network size, and it cannot control the average search hops according to the actual needs of cloud computing. However the proposed algorithm's average search hops has nothing to do with the network size, and it tends to the set value as the network size increases, therefore it can resolve the constant jump locating problem.

Key words: cloud resources; cloud peer-to-peer network; topology; routing; resources locating; average search hops

0 引言

如何将网络上提供云服务的大量物理资源统一管理是云计算应用的一个基本问题。如果使用一个中央节点来收集其他所有节点的实时状态信息并处理所有的用户查询，则会使中央节点成为云计算大规模应用的瓶颈，因此有必要研究如何建立分布式的云资源信息存储及索引系统。用结构化全分布式 P2P 网络来建立云资源信息存储及索引系统能够自适应节点的加入与退出，具备良好的可扩展性、健壮性，且节点 ID 分配均匀，还可以提供资源的精确发现。这种用来管控云计算资源的对等网络称为云对等网络。

云对等网络把云计算和对等网络结合起来构建一个广域的云资源管理、互补及交换的平台，使大量的计算、存储、带宽、多媒体和信息资源通过对等网络连接成一个整体的云。对于用户的计算资源需求，云对等网络需要快速地对云资源进行定位。举一个例子，假设某用户需要一台虚拟机，对虚拟机要求如下：处理速度 1 000 MIPS，内存 4 GB，硬盘 100 GB。经查询比较得到满足要求节点的 ID，云对等网络就可以依据节点 ID 定位该云节点。云对等网络的拓扑结构特点与传统对等网络相比有很大不同：云对等网络是由云服务器（一般是小型机、x86 服务器或一些其他的专业云机器）构成的，且提供云计算的云服务器的状态较为稳定，不会频繁地上下线，故其具有稳定的拓扑结构。过去对等网络研究的重点往往是尽量减少每个节点的邻居数，这样一来，当节点频繁上下线时，它们不需要动态生成太多的邻居。云对等网络拓扑结构的研究重点不应该是减少生成邻居的开销，而应该是减少每个用户查询被转发的跳数。本文的思路是建立多层次对等网络并增加网络中节点的邻居数来减少查询跳数。

1 相关工作

1.1 经典结构化 P2P 路由算法比较

当前经典的结构化 P2P 路由算法有 Chord[1]、Pastry[2]、CAN[3]、Viceroy[4]和 Koorde[5]。在规模为 N 个节点的网络中它们的性能比较如表 1 所示（表中 d 为拓扑结构图的维数，$N = d \times 2d$）。

收稿日期：2012-06-24；**修回日期**：2012-08-08 **基金项目**：国家自然科学基金资助项目(61170277)；上海市教委科研创新重点资助项目(12zz137)

作者简介：李璞(1986-)，男，江苏淮安人，硕士研究生，主要研究方向为计算机网络、云计算(534046717@qq.com)；陈世平(1964-)，男，江西南昌人，教授，博导，主要研究方向为计算机网络、分布式计算、云计算；李剑锋(1977-)，男，黑龙江哈尔滨人，讲师，博士研究生，主要研究方向为云计算、供应链管理.

表 1　典型结构化 P2P 路由算法比较

名称	拓扑结构	路由表大小	路由复杂度
Chord	Ring	$O(\log N)$	$O(\log N)$
Pastry	Plaxton trees	$O(\log N)$	$O(\log N)$
CAN	d-dim. torus	$O(d)$	$O(d\times N1/d)$
Viceroy	butterfly	$O(1)$	$O(\log N)$
Koorde/D2B	De Bruijn	$O(1)$	$O(\log N)$

Chord 基于一维的环形空间,每个节点维护的路由表大小为 $O(\log N)$,路由的复杂度为 $O(\log N)$。Pastry 路由算法中每个节点维护的路由表大小及系统的路由步长和 Chord 数量级相同,其拓扑结构为 Plaxton 树。CAN 路由算法的拓扑结构为 d 维环形空间,与前两种路由算法相比,路由表数量级更小,为 $O(d)$,但路由复杂度更大,为 $O(d\times N1/d)$。Viceroy 和 Koorde 算法的路由表大小和路由复杂度在同一个数量级,分别为 $O(1)$和 $O(\log N)$,但它们的拓扑结构不同,这两种算法较前三种算法在性能上有较大的改进,路由复杂度虽在同一个数量级 $O(\log N)$,但是路由表大小是常量级的,与网络的规模无关,这使得网络中的通信量大大减少。

但是这些传统网络仍不能满足云网络常数转发跳数的要求。

1.2　基于超节点的路由算法比较

一些结构化覆盖网的研究者认为 $O(\log N)$跳仍然效率不高,而现在的终端节点有能力维护更多的指针,从而获取更高的路由效率。他们提出了一些在一跳或两跳之内即可完成消息路由的结构化覆盖网协议,包括 Kelips[6]、one-hop[7]、Twins 等。

Kelips 将节点分为 k 组,组内节点全连接,通过 Gossip 方式进行更新,这样维护开销增大很多,但是路由可以在第二跳完成。One-hop 中每个超节点维护全局路由状态,记录其他所有节点的信息,这样路由可以在一跳内完成,查找速度快,延时小。这些协议与其他协议的根本不同之处在于,它们使用组播或广播方式进行指针维护,即节点加入或退出时将消息广播给所有需要知道该消息的节点。为了保证路由在很少的跳数内完成,节点需要维护非常大量的指针,而且该方法使得路由表的开销较大,为 $O(N)$或 $O(N^{1/2})$。

当网络规模非常大时,上述的协议就会因为路由表开销太大而无法应用。

2　云对等网络的资源定位算法

云对等网络把网络拓扑结构、检索信息存储和信息查询三者通过一个共同的标志符空间有机地结合起来,利用特殊的覆盖网络拓扑,通过设计高效的路由算法来解决常数跳定位问题。

2.1　基于 DHT 的云网络

在云对等网络通信中使用 DHT(distributed hash table,分布式散列表)为实际网络建立一层结构化的覆盖层,即一个逻辑层。节点加入 DHT 网络时需要对其哈希产生一个标志节点的编号,从而决定它在 DHT 网络中的位置,以及它在逻辑网络中的邻居表(路由表)。每个节点要维护一些云资源索引信息,即(key,value)对。其中,key 为目标节点的编号,value 则是存储在目标节点的索引信息,包括 CPU、内存、带宽等。

传统对等网络是由用户终端直接构成的,用户会频繁上线与下线,因此它的拓扑结构是不稳定的。故在算法设计中,减少动态拓扑构建的开销也就成为一个重点。由于云服务器长期稳定地滞留在网上,传统算法为动态拓扑构建而做的以损失查询性能为代价的一些折中往往是不必要的。根据云节点的物理属性计算节点编号,节点 ID = H(节点物理属性),H 为分布式散列表。因为这些节点相对稳定地滞留在网上,所以整个对等网络的拓扑也相对稳定。

2.2　拓扑模型设计

根据实际需要将网络中的全部节点分层,每层再划分若干组。每个分组中实节点最大数目为 $M-1$,$N[i]$表示普通节点,$S[i]$表示超节点,$V[i]$表示虚节点。每个分组的超节点可以有多个,一个典型的拓扑模型示例如图 1 所示。

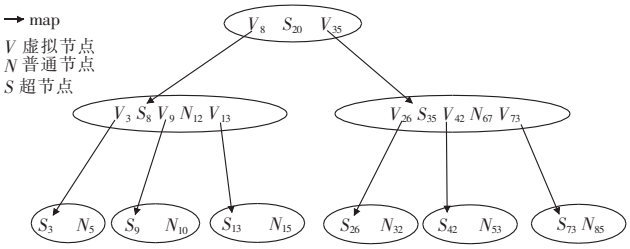

图 1　一种拓扑模型示例

定义 1　顶层仅有一个分组且分组中节点数为[1,$M-1$]。

定义 2　除顶层外,其他每层的各分组中节点数为[$M/2-1$,$M-1$]。

定义 3　节点分为超节点和普通节点,超节点负责路由信息维护、节点查找和网络关系维护。普通节点保存自己的前驱和后继节点的路由信息并把自己的路由信息及时同步到本组超节点。

定义 4　分组中每个虚拟节点的功能由下一层子组中对应的超节点执行。

定义 5　各分组中节点编号为 $V[1]$、$N[1]$、$V[2]$、$N[2]$、…、$N[M-1]$、$V[M]$,其中 $N[i]<N[i+1]$,$V[1]$为节点编号小于 $N[1]$的下一层分组中超节点的虚拟,$V[M]$为节点编号大于 $N[M-1]$的下一层分组中超节点的虚拟,其他 $V[i]$为节点编号属于($N[i-1]$, $N[i]$)的下一层分组中超节点的虚拟,$S[i]$表示本组内的超节点。

2.3　分组大小的计算

由拓扑模型定义可知,分组大小决定了超节点路由表的大小。假设云对等网络有 n 个节点,每个分组的实节点数为 m,共有 h 层,每层最少实节点的数目为(自顶向下):

$$1,\lfloor m/2\rfloor\times 2,(\lfloor m/2\rfloor+1)\times 2\times\lfloor m/2\rfloor,(\lfloor m/2\rfloor+1)^2\times 2\times\lfloor m/2\rfloor,(\lfloor m/2\rfloor+1)^3\times 2\times\lfloor m/2\rfloor,\cdots$$

将其求和,得出 m、n 和 h 之间的关系:

$$\frac{(\lfloor m/2\rfloor+1)[1-(\lfloor m/2\rfloor+1)^{h-2}}{1-(\lfloor m/2\rfloor+1)}\times 2\times\lfloor m/2\rfloor+2\times\lfloor m/2\rfloor+1=2\times(\lfloor m/2\rfloor+1)^{h-1}-1=n$$

$$\xRightarrow{令 t=\lfloor m/2\rfloor+1} h=\log_t((n+1)/2)+1$$

故对一个具有 n 个节点云对等网络,其层数 h 至多为

$$\log_t((n+1)/2)+1\quad 其中\ t=\lfloor m/2\rfloor+1 \tag{1}$$

在云对等网络中,目标是使查询转发的跳数不大于常数 C,由于云对等网络的规模相对稳定,根据上述结论,$h\leqslant\log_t(\frac{n+1}{2})+1\leqslant C$,则有:

$$\log_t(\frac{n+1}{2})+1\leqslant C\Rightarrow t\geqslant(\frac{n+1}{2})^{\frac{1}{c-1}}\xRightarrow{式(1)}\lfloor m/2\rfloor+1\geqslant(\frac{n+1}{2})^{\frac{1}{c-1}}\Rightarrow m>2\times\lfloor(\frac{n+1}{2})^{\frac{1}{c-1}}\rfloor-2$$

为维护较小的路由表,m 通常取临界值,即

$$m = 2 \times \lfloor (\frac{n+1}{2})^{\frac{1}{c-1}} \rfloor - 1 \quad (2)$$

根据式(2)可知路由表大小满足云对等网络应用的要求。

2.4 路由算法

超节点路由表中应该包括所在组全部实节点和虚节点的路由信息。路由信息包括节点编号、IP 地址、提供服务的端口号等。若超节点不在顶层组中,路由表中还应该有路由到顶层超节点的路由信息。当网络规模非常大时,每个分组中可以由多个超节点用以负载均衡。

节点定位的具体流程如图 2 所示。

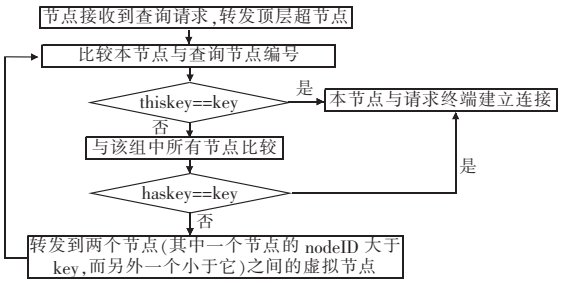

图 2 定位节点流程

任意节点在接到查询请求时,都将该查询请求转发给顶层组中的超节点。超节点在接收到请求时,判断查询节点的节点编号与自己的编号,若相同,则查找成功;若不同,则与该组中所有节点进行比较。假如在比较时发现有节点编号与之相同,则把该节点信息发送给资源请求节点。如果没有发现相同编号节点,则请求转发到两个实节点(其中一个节点的编号大于查询节点的编号,而另外一个小于它)之间的虚拟节点。

用图 1 描述节点 S_8 到节点 S_{73} 的路由。S_8 在接收到查找 S_{73} 节点的请求后,把此请求转发至网络顶层组的超节点 S_{20};而后 S_{20} 查询本分组的路由表后,再把请求转发给子分组的超节点 S_{35}。最后 S_{35} 把请求转发至目标节点 S_{73}。

2.5 节点的加入

假设需要在云对等网络中插入一个编号为 K 的节点,首先在网络中查找 K,若找到则直接返回(假设不处理相同编号节点的插入);否则查找操作必失败于最底层某分组上,然后将 K 插入该分组中。若该分组原来是非满(网络中原有的节点总数小于 $M-1$)的,则插入 K 后并未违反拓扑模型的定义,故插入 K 后即完成了插入操作;若该分组原为满,则 K 插入后违反拓扑模型定义 2,故须调整局部网络使其维持拓扑模型定义不变。

调整操作。将违反定义 2 的分组以中间位置上的节点 $N[M/2]$ 为划分点,该分组(不妨设为 * current)为 $(V_0, N_1, V_1, \cdots, N_{M-1}, V_M)$,$N_i$ 表示 $N[i]$,V_i 表示 $V[i]$,将"分裂"为两个节点:$(V_0, N_1, V_1, \cdots, N_{M/2-1}, V_{M/2-1})$,该分组仍是 * current;$(V_{M/2}, N_{M/2+1}, \cdots, N_{M-1}, V_M)$,该分组是新产生的分组 * new。将中间节点 $N_{M/2}$ 和新分组的 * new 一起插入到 * current 对应的上一层分组中。

其中 i 表示下标,当 $N_{M/2}$ 和新局部网络超节点的虚拟节点一起插入已满的上一层分组时,对应的上一层分组也要做分裂操作。最坏的情况是,各层的分组都是满的,此时,插入过程中的分裂操作一直向上传播到顶层。当顶层组分裂时,因为没有上一层,故需建立一个新的顶层组,此时网络层数加 1。图 3 为调整操作的一个示例。

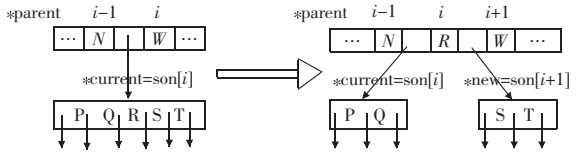

图 3 大小为 5 的组分裂示例

由空网络开始逐一加入云节点便可以生成一个完整的云对等网络。

2.6 节点的退出

假设需要在云对等网络中删除一个编号为 K 的节点,首先查找节点,然后进行删除操作。任一编号为 K 的前趋(后继)必是 K 的左子组(右子组)中最右(左)下的节点中最后(最前)一个节点。

若被删的编号为 K 的节点不在最底层,则用 K 的前趋(或后继)K' 取代 K,然后从子组中删去 K'。从最底层组 $*x$ 开始删去某编号为 K 节点的三种情形为:

a)若 $x \rightarrow \text{keynum} \geqslant M/2-1$,则只需删去 K 即可使删除操作结束。

b)若 $x \rightarrow \text{keynum} = M/2-1$,该组中的节点个数已是最小值,删除 K 及其右虚拟节点后会破坏拓扑定义 2。若 $*x$ 的左(或右)邻兄弟组 $*y$ 中的节点数目大于 $M/2-1$,则将 $*y$ 中的最大(或最小)节点上移至双亲组 * parent 中,而将 * parent中相应的节点下移至 $*x$ 中。显然这种移动使得双亲中节点数目不变;$*y$ 被移出一个节点,故其 keynum 减 1,因它原大于 $M/2-1$,故减少 1 个节点后 keynum 仍大于等于 $M/2-1$;而 $*x$中已移入一个节点,故删去 K 后 $*x$ 中仍有 $M/2-1$ 个节点。涉及移动节点的三个组均满足拓扑定义 2。请读者验证,上述操作后仍满足拓扑定义 1。移动完成后,删除过程亦结束。

c)若 $*x$ 及其相邻的左右组(也可能只有一个兄弟)中的节点数目均为最小值,则上述的移动操作就不奏效,此时须 $*x$ 和左或右组合并。不妨设 $*x$ 有右邻组 $*y$(对左邻兄弟的讨论与此类似),在 $*x$ 中删去 K 及其右子组后,将双亲组 * parent中介于 $*x$ 和 $*y$ 之间的节点 K 作为中间节点,将 $*x$ 和 $*y$ 中的节点一起合并为一个新的组来取代 $*x$ 和 $*y$。因为 $*x$ 和 $*y$ 原各有 $M/2-1$ 个关键字,从双亲中移入的 K' 抵消了从 $*x$ 中删除的 K,故新节点中恰有 $2\lceil M/2 \rfloor - 2 (\leqslant M-1)$ 个节点,没有破坏拓扑定义 2。但由于 K' 从双亲中移到新组后,相当于从 * parent 中删去了 K',若 parent→keynum 原大于 $M/2-1$,则删除操作到此结束;否则,同样要通过移动 * parent 的左右组中的节点或将 * parent 与其左右组合并的方法来维护拓扑定义 2。最坏情况下,合并操作会向上传播至顶层,当顶层只有一个节点时,合并操作将会使顶层组及其两个子组合并成一个新的组,从而使整个网络减少一层。

3 i 仿真实验

一般情况下,用平均查找跳数来衡量查询性能。本文算法的实验是在 Linux 环境下,使用 Java 编写模拟程序来评估。为体现本文算法的优越性,选择传统结构化对等网络具有代表性的经典算法 Chord 进行对比。Chord 算法实验是在 Linux 环境下,使用 P2P 仿真工具 PeerSim 进行仿真实验。

3.1 实验 1

第一次实验假设在路由表大小可维护的前提下,云对等网

络要求查找跳数不大于 5,依次模拟 10000、15000 ~ 65000 个节点的云对等网络,根据公式计算得到分组大小。

根据分组大小生成网络,由生成的网络进行模拟实验得到节点的平均查找跳数。在模拟实验中,随机选择节点对随机产生的节点编号 p 进行查找,反复进行 80 次,最终求得所有节点的平均查找长度,反复进行 10 次取平均值。

第二次实验假设在路由表大小可维护的前提下,云对等网络要求查找跳数不大于 6,其他条件及实验次数同第一次实验。两次实验结果数据如表 2 所示。

表 2　实验 1 仿真实验结果

网络规模	第一次实验		第二次实验	
	分组大小	平均查找跳数	分组大小	平均查找跳数
10 000	15	4.57	9	5.54
15 000	17	4.63	11	5.61
20 000	18	4.67	11	5.66
25 000	21	4.73	12	5.71
30 000	21	4.77	13	5.75
35 000	23	4.8	13	5.78
40 000	23	4.83	13	5.81
45 000	23	4.86	13	5.84
50 000	24	4.89	14	5.86
55 000	25	4.91	15	5.88
60 000	25	4.92	15	5.90
65 000	25	4.93	15	5.91

3.2　实验 2

在与实验 1 相同网络规模和相同实验环境下使用 PeerSim 仿真工具对 Chord 算法进行两次仿真实验,实验结果数据如表 3 所示。

表 3　实验 2 仿真实验结果

网络规模	第一次实验	第二次实验
	平均查找跳数	平均查找跳数
10 000	6.55	6.53
15 000	6.92	6.94
20 000	7.16	7.2
25 000	7.38	7.36
30 000	7.54	7.54
35 000	7.67	7.68
40 000	7.78	7.77
45 000	7.87	7.83
50 000	8	8
55 000	8.02	8.02
60 000	8.11	8.09
65 000	8.16	8.2

3.3　实验结论

随着网络规模的增大,实验 1 中第一次实验的平均查找跳数趋向于极限 5,第二次实验最终趋向于极限 6,而对于传统的 Chord 算法,在两次实验中在相同网络规模下的平均查找跳数并没有明显变化。

从以上的两个实验结果对比可以得出,传统对等网络的平均查找跳数与网络规模直接相关,无法依据云环境的实际需要人为地控制查找跳数。本文算法的平均查找跳数与网络规模无关,随着网络规模的增大而趋向于设定值。

4　结束语

本文在分析了使用一个中央节点来收集和处理所有节点的实时信息无法满足云计算大规模应用的需要后,提出用结构化对等网络来建立分布式的云资源信息存储及索引系统。在对传统对等网络路由算法和基于超节点路由算法优缺点比较之后,提出用分布式散列表来标志云资源并设计一种新型拓扑来解决云资源的常数跳定位,利用该拓扑设计基于超节点的路由算法,在路由表可维护的前提下选择查找跳数的极限。最后仿真实验表明,该算法的路由表维护开销远远小于其他基于超节点的 P2P 算法,且能满足云计算大规模应用的需要,其查找跳数符合云资源的常数跳定位的需求。

参考文献:

[1] STOICA I, MORRIS R, KARGER D, *et al*. Chord: a scalable peer-to-peer lookup service for Internet applications [J]. IEEE/ACM Trans on Networking, 2003, 11(1): 17-32.

[2] ROWSTRON A, DRUSCHEL P. Pastry: scalable, distributed object location and routing for large scale peer-to-peer systems [C]//IFIP/ACM International Conference on Distributed Systems. Berlin: Springer-Verlag, 2001: 329-350.

[3] RATNASAMY S, FRANCIS P, HANDLEY M, *et al*. A scalable content-addressable metwork [J]. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 161-172.

[4] MALKHI D, NAOR M, RATAJCZAK D. Viceroy: a scalable and dynamic emulation of the butterfly [C]//Proc of the 21st Annual Symposium on Principles of Distributed Computing. New York: ACM Press, 2002: 183-192.

[5] KLEIS M, LUA E K, ZHOU Xiao-ming. Hierarchical peer-to-peer networks using lightweight super peer topologies [C]//Proc of the 10th Symposium on Computers and Communications. Washington DC: IEEE Computer Society, 2005: 143-148.

[6] GUPTA I, BIRMAN K, LNGA P, *et al*. Kelips: building an efficient and stable P2P DHT through increased memory and background overhead [C]//Proc of the 2nd International Workshop on Peer-to-Peer System. Berlin: Springer-Verlag, 2003: 160-169.

[7] GUPTA A, LISKOV B, RODRIGUSE R. One-hop lookups for peer-to-peer overlays [C]//Proc of the 1st Symposium on Networked Systems Design and Implementation. New York: ACM Press, 2004: 113-126.

[8] 王必晴,贺鹏. H-Chord:基于层次划分的 Chord 路由模型及算法实现[J]. 计算机工程与应用,2007,43(36):141-143.

[9] 段世惠,王劲林. 基于有限范围组播的 Chord 路由算法[J]. 计算机应用,2009,29(2):514-517.

[10] SHEN Hai-ying, XU Cheng-zhong, CHEN Gui-hai. Cycloid: a constant-degree and lookup-efficient P2P overlay network [C]//Proc of the 18th International Parallel and Distributed Processing Symposium. Washington DC: IEEE Computer Society, 2004: 1-10.

[11] XU Ke, SONG Mei-na, ZHANG Xiao-qi, *et al*. A cloud computing platform based on P2P [C]//Proc of IEEE International Symposium on IT in Medicine and Education. Washington DC: IEEE Computer Society, 2009: 427-432.

[12] ZHAO Peng, HUANG Ting-lei, LIU Cai-xia, *et al*. Research of P2P architecture based on cloud computing [C]//Proc of International Conference on Intelligent Computing and Integrated Systems. Washington DC: IEEE Computer Society, 2010: 652-655.

[13] SOTIRIADIS S, BESSIS N, ANTONOPOULOS N. Using self-led critical friend topology based on P2P chord algorithm for node localization within cloud communities [C]//Proc of International Conference on Intelligent Computing and Integrated Systems. Washington DC: IEEE Computer Society, 2011: 490-495.

[14] HUANG Li-can. Semantic P2P networks: future architecture of cloud computing [C]//Proc of the 2nd International Conference on Networking and Distributed Computing. Washington DC: IEEE Computer Society, 2011: 336-339.