带宽自适应的 P2P 网络路由协议*

胡进锋+, 黎 明, 郑纬民, 汪东升

(清华大学 计算机科学与技术系 高性能计算研究所,北京 100084)

A Self-Adaptive Peer-to-Peer Routing Protocol

HU Jin-Feng+, LI Ming, ZHENG Wei-Min, WANG Dong-Sheng

(Institute of High Performance Computing, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: +86-10-62524913, E-mail: hujinfeng00@mails.tsinghua.edu.cn, http://hpc.cs.tsinghua.edu.cn

Received 2003-11-03; Accepted 2004-07-29

Hu JF, Li M, Zheng WM, Wang DS. A self-adaptive peer-to-peer routing protocol. *Journal of Software*, 2005,16(5):991−999. DOI: 10.1360/jos160991

Abstract: This paper presents a novel peer-to-peer structure overlay network SmartBoa. Compared to previous protocols, SmartBoa nodes have routing tables with different sizes, which are determined by the local nodes individually. It is ensured that the bandwidth cost of a node is proportional to its routing table size. Therefore, from the view of the whole system, all the allowable bandwidth are fully utilized to improve the routing efficiency. SmartBoa does not increase the capacity requirement for nodes when the system expands, so it can achieve higher scalability than the one-hop protocol. Furthermore, SmartBoa nodes can adjust its level at runtime, and thereby can warm up gradually when starting. This avoids the long-time initiation which is an important problem in one-hop overlay. In a word, SmartBoa is a general structure overlay network that can be deployed in any environments, not matter what the system size is, how dynamic the nodes are, and what the node-capacity distribution is like.

Key words: peer-to-peer system; structured overlay network; routing protocol; heterogeneous; distributed system

摘 要: 提出一种普适于各种系统环境和网络规模的结构化 P2P 网络协议 SmartBoa.与已有的结构化 P2P 路由协议(如 Pastry,Chord 等)相比,SmartBoa 结点并不维护同样大小的路由表,而是各结点根据自身的带宽能力决定其路由表的大小(最强的结点可能记录全部结点的指针,最弱的结点可能只记录其中不足 1%的一小部分),算法保证路由表大小正比于维护开销,充分利用所有结点的可用带宽,使路由效率达到最优;另一方面,SmartBoa 并不因为系统规模的增大而增加对结点带宽的要求,因此与全连通的 one-hop overlay 相比,SmartBoa 可以获得更好的可扩展性;再者,SmartBoa 结点根据系统环境的变化动态地调节自身级别,并且可以通过逐渐调高级别的慢

* Supported by the National Natural Science Foundation of China under Grant No.60433040 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.G2001AA111010 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032702 (国家重点基础研究发展规划(973))

作者简介: **胡进锋**(1978−),男,安徽马鞍山人,博士,主要研究领域为 P2P 系统,广域存储服务;**黎明**(1979−),男,硕士,主要研究领域为 P2P 系统,广域存储服务;**郑纬民**(1946−),男,教授,博士生导师,主要研究领域为并行计算,分布式系统;**汪东升**(1966−),男,教授,博士生导师,主要研究领域为分布式系统,CPU 设计.

启动方式来克服 one-hop overlay 的启动时间过长的缺陷.总之,SmartBoa 是一种可以运行于任何环境,不受限于系统规模的大小、结点能力的强弱、强弱结点的比例、结点出入的频率,并通过动态调节保证路由效率的 P2P 路由协议,适用于各种广域分布式系统.

关键词: P2P 系统;结构化覆盖网;路由协议;异构性;分布式系统

中图法分类号: TP393 **文献标识码**: A

P2P 系统以飞快的速度发展成为 Internet 中最重要的应用系统之一,当前,P2P 协议产生的 Internet 流量已经超过 http 协议.P2P 系统的体系结构也发生了由 Napster 的集中查询到自由连接(unstructured overlay network),再到 Kazaa 的偏向于强结点的自由连接的逐步演变.

但是,由于自由连接的随意性,使得数据查询必须依靠广播(flooding)来完成,耗费大量的网络开销,因此,系统的可扩展性受到严重限制.于是,近年来提出的结构化覆盖网(structured overlay network)以及基于结构化覆盖网的分布式哈希表(distributed hash table,简称 DHT)算法成为研究领域的热点.在 DHT 中,每个结点有其唯一标识 nodeId,一般为 128 位数;需要存储的数据将其标识(文件名、ObjectId 等)通过固定的哈希算法(如 SHA-1)同样得到一个 128 位数,规定该数据被存放在 nodeId 在数字空间离该数最近的结点上.

结构化覆盖网负责任意两个结点之间的路由.在结构化覆盖网中,每个结点存储一张路由表,记录指向某些其他结点的指针(即 nodeId+ip+port),消息的路由过程也就是结点在路由表中查找离目标结点更近的结点从而将消息不断转发直至到达目标结点的过程,消息每从一个结点转发到下一个结点称为一跳(hop).

已有的结构化覆盖网分为两类.第 1 类算法中每个结点只记录一小部分其他结点的指针,算法保证消息路由在一定的跳数内完成,典型的算法如 Tapestry[1],Pastry[2],CAN[3],Chord[4]等.以 Pastry 为例,典型的 Pastry 的路由表含有 $\left(2^b-1\right)\cdot\log_{2^b}N$ 项,消息路由在 $\log_{2^b}N$ 跳数内完成(其中 N 为总结点数,b 为系统常数,一般取 4).这类算法的特点在于系统开销小(每秒仅收发几个消息),并且相对于 unstructured overlay network 路由效率有大幅度提高,但是与结点之间的直接消息通信相比,仍然慢 4~9 倍[5].

第 2 类结构化覆盖网算法中每个结点记录所有其他结点的指针(one-hop overlay),使得绝大多数(99%以上)消息通信可以直接完成[6].这种算法下的路由非常高效,但是存在几个基本问题:一是带宽要求大,对于 100 000 结点的系统,带宽要求为弱结点 3.3kbps,中间结点 6.5kbps,强结点 300kbps,且要求约 100 个强结点存在,若结点数上升到 1 000 000,则强结点的上行带宽要求达到 70Mbps;二是可扩展性差,结点的带宽开销为 $O(N)$量级;三是启动时间长,在 1 000 000 结点的系统中,弱结点在初始化时下载全部路由表需要约 20 分钟时间.

为了提高第 1 类算法的效率,解决第 2 类算法面临的基本问题,本文给出一种新的结构化覆盖网 SmartBoa.与已有算法不同的是,SmartBoa 遵循一条新的设计思路:每个结点根据自己的带宽约束决定其路由表的大小,尽可能地记录更多的指针.显然,路由表越大,路由效率也将越高.这样,弱结点只记录一小部分指针,能力越强的结点记录的指针越多,最强的结点可以记录全部指针.结点独立决定各自路由表的大小,不受其他条件约束,因此 SmartBoa 可以看成是一种普适的结构化覆盖网,不论系统中结点能力如何、结点数量如何、结点变化频率如何.从宏观上看,当系统规模大、结点动态性强、结点能力相对较弱时(如 Kazaa 等 P2P 文件共享系统),SmartBoa 中每个结点只记录一小部分其他结点的指针,类似于 Pastry;当系统规模不很大、结点比较稳定、能力相对较强时(如 Grid 系统),SmartBoa 中每个结点都记录所有结点的指针,类似于 one-hop overlay.

实现 SmartBoa 的关键是解决路由表维护问题,必须保证维护路由表的系统开销正比于路由表的大小,只有这样,结点才能通过调节路由表的大小来控制自身的系统开销,以保证系统开销在带宽可以承受的范围之内.维护路由表的基本方法有两种,一是定期探测(probe)路由表中的所有指针,像 Pastry 中那样;二是结点发生变化时,通过“发现-组播”机制告知其他结点,像 one-hop overlay 那样.显然,第 1 种方式直接保证了维护开销正比于路由表大小,但是只有在路由表不大时才可行,一旦路由表达到一定规模,其开销将无法承受(例如,对于含 10 000 个表项的路由表,若每 50s 探测一次,则每秒需发送 200 个消息).因此,只能采用组播的方式进行路由表维护.

SmartBoa 通过给结点设定级别,规定路由表内容(即路由表中包含哪些指针)与 nodeId 和级别的关系,使得当系统中某结点 M 发生变化时,另一结点 N 是否关心这一变化只决定于 M 的 nodeId、N 的 nodeId 以及 N 的级

别,而无须设置专用标志,这样就可以设计出合适的组播算法,保证 M 变化的消息只在关心这一变化的结点中传播,并且保证组播无冗余(同一个消息每个结点只收到一次).这样,路由表的大小就与维护开销(指下行带宽)成正比关系,结点可以根据自己的带宽能力估算自己能够维护的路由表大小,从而确定自己的级别.在实际运行过程中,能力不足或有余时,结点还可以对级别进行动态调整.

与已有算法相比,SmartBoa 由于充分利用了带宽,从而获得比 Pastry 更高的路由效率;由于系统规模增大时,结点自动降低级别,通过与路由效率的折衷,保证系统的正常运行,从而获得了比 one-hop overlay 更好的可扩展性;由于结点在启动时可以逐步调高级别(慢启动),从而克服了 one-hop overlay 的启动时间过长的缺陷;由于将级别记入结点指针,且通常路由表具有较大的规模(指针数上万),从而可以方便地向上层应用系统提供在当前网络中搜寻强(弱)结点的服务.

本文第 1 节描述 SmartBoa 算法框架,包括路由表的构成与路由方法.第 2 节讨论路由表的维护.第 3 节是算法中的其他问题.第 4 节讨论相关工作.第 5 节总结全文.

1 算法框架

与 Pastry 和 Chord 一样,SmartBoa 赋予每个结点一个唯一标识 nodeId,nodeId 为 128 位长,因此可以看成是每个结点在 0~2^{128}–1 的数字空间(看做是一个环,称“id 空间”,又称“nodeId 环”)中占据一个位置.NodeId 由结点的标识(如 ip、域名等)通过均匀的哈希算法(如 SHA-1)得到.消息发送的目标地址也表示为一个 128 位数,消息最终被转发至 nodeId 环中离目标地址最近的结点,称为“受理结点”.

与 Pastry 及 Chord 不同的是,SmartBoa 中每个结点另有“级别”(0,1,2,…)属性,用以标志其强弱(0 为最强,也称级别最高),级别越高的结点路由表越大,在路由过程中的“下一跳”也就离目标结点越近.结点的级别作为结点的基本属性同 ip、端口一起被记入指针.

1.1 路由表的构成

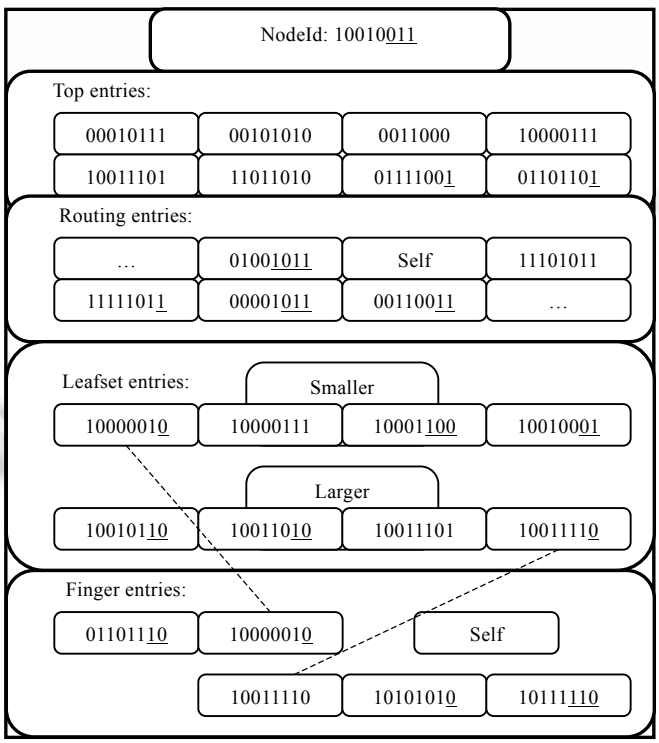

Fig.1 An example of SmartBoa node. Eigenstrings are underlined

图 1 SmartBoa 结点路由表示例.特征串用下划线标出以提示级别

SmartBoa 结点的路由表分为 4 个部分:routing entries,top entries,leafset entries 和 finger entries.图 1 是一个可能的路由表的示例,为方便描述,例中的 nodeId 设为 8 位长.图 2 描绘路由表中各部分指针所指向的结点与该结点本身在 nodeId 环上的相互位置关系.以下分别讲述 4 部分的组成.

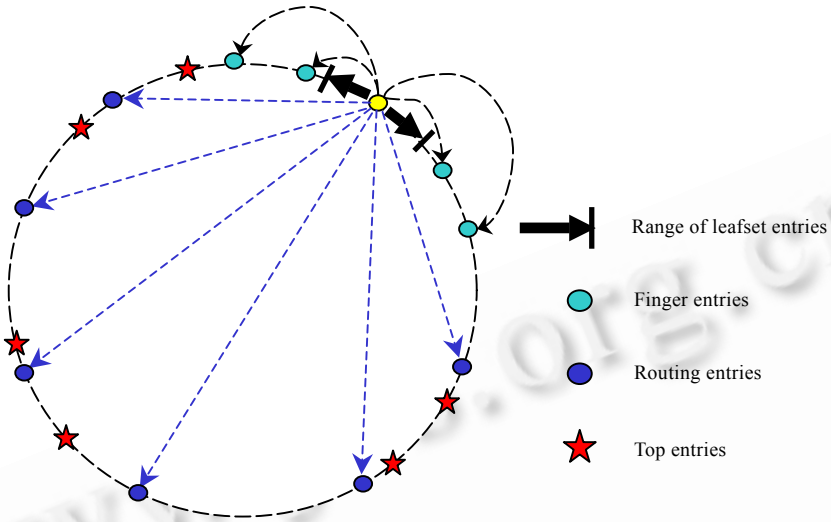

Fig.2 Relative positions of the entries in the four parts of a routing table
图 2 结点路由表中各部分指针所指向的结点在 nodeId 环上的位置

Routing entries.路由表的绝大部分被 routing entries 占据,routing entries 的大小直接决定了路由速度.

设结点 M(为方便叙述,本文中对结点的代称和它的 nodeId 混用,“结点 M”即意为该结点的 nodeId 为 M)的级别为 k,则它的 routing entries 中记录当前所有 nodeId 的末 k 位与 M 相同的结点(实际记录数据为对应结点的“nodeId+ip+端口+级别”)(如图 1 所示).由于 nodeId 在 id 空间均匀分布,一个级别为 k 的结点 routing entries 的大小约为 $N/2^k$,在 nodeId 环上相邻指针之间约有 2^k-1 个其他结点(如图 2 所示).称 M 的末 k 位结点为“特征串”,写作“α_M”.级别为 0 的结点的特征串为空串,文中空串用“$\varnothing$”表示.如图 1 中所示的是一个级别为 3 的结点的路由表,在显示 nodeId 时均用下划线将其特征串标出以提示结点级别,没有下划线标注的 nodeId 表示此结点的级别为 0.

显然,特征串相同的结点 routing entries 也相同,所有特征串为α的结点组成集合$\{\alpha\}$.

Top entries.它在组播协议中使用,这里只描述它的特征,其使用和维护将在第 2 节中介绍.为方便叙述,先给出如下 3 个定义:

定义 1. 如果结点 A 的特征串α_A是结点 B 的特征串α_B的后缀,且$\alpha_A\neq\alpha_B$,则 A 的 routing entries 必然包含 B 的 routing entries 中的所有项,此时称“A 强于 B”,也称“A 是 B 的超结点”,记作“$A>B$”.

定义 2. 如果当前系统中不存在任何 A 的超结点,则称 A 为一个“顶结点(top node)”.

定义 3. 若 $A>B$ 或$\alpha_A=\alpha_B$,且 A 为顶结点,则称“A 是 B 的顶结点”.

一般情况下,顶结点的集合即为$\{\varnothing\}$.但当系统规模大、结点能力不足时,系统中不存在级别为 0 的强结点,此时,顶结点分属于{“0”}和{“1”}两个集合,或者级别更低.

结点 M 的 top entries 中记录所有 M 的超结点中最强的 t 个结点(t 为系统参数,通常设为 8).若 M 的超结点不足 t 个,则不足部分留作空白.顶结点的 top entries 全为空.一般说来,top entries 中都是顶结点,但如果当前强于 M 的顶结点不足 t 个,则由级别稍低的超结点补足,但 top entries 中必然至少有一个顶结点.如图 1 所示,只有 6 个级别为 0 的顶结点时,余下两项记录级别为 1 的超结点.

当结点 M 加入、退出或者改变级别时,需将这一变化组播给所有 routing entries 中含有 M 的结点,组播算法的第 1 步就是将消息发给 M 的某个顶结点.组播算法具体细节见第 2 节.

Leafset entries.与 Pastry 中的 leaf set 一样,leafset entries 记录 nodeId 环离自己最近的 l 个结点,其中左边 $l/2$

个,右边 $l/2$ 个(在 nodeId 环上,nodeId 比自己小的方向称为“左边”,比自己大的称为“右边”),如图 2 所示.Leafset entries 使结点在 nodeId 环上紧密相连,确保系统连通,并保证路由总收敛到目标结点.l 一般取 16 或 32.

Finger entries.在一般情况下,routing entries 和 leafset entries 已经能取得很好的路由效率.但当系统规模很大、结点相对较弱时,路由第 1 跳(通过 routing entries)后仍离目标地址很远,如果之后都通过 leafset entries 路由,则跳数仍会很多.这时,需要 finger entries 进行加速.对于结点 M,考察其 routing entries 中的所有结点,在 nodeId 环上位于 M 右边第 1 个的结点记为 N,finger entries 中记录 M 和 N 之间的折半查找的指针,即(M+N)/2 的受理结点,(M+(M+N)/2)/2 的受理结点等等,直至与 leafset entries 重合.类似地,M 与 routing entries 中左边第 1 个结点之间的折半指针也记录在 finger entries 中.如图 2 所示.通常情况下,finger entries 中的表项很少,如图 1 所示.

1.2 路 由

路由使用最简单的贪婪算法.

对于目标地址为 X 的消息,结点 M 将其转发给路由表(包括全部 4 个部分)中离 X 最近的结点.如果自己即为离 X 最近的结点,则自己就是路由的终点.如果自己和另一个结点 N 离 X 距离相同,则 X 左边的结点为路由终点.

虽然这种算法非常简单,但已经可以取得很高的效率.

2 路由表的维护

2.1 Routing entries的维护

由于 P2P 系统中不断有新结点加入、已有结点退出,因此,结点的路由表必须得到及时的更新,以反映系统的最新状况.SmartBoa 中不同结点的路由表大小不同,在维护时就必须保证路由表小的结点维护开销也小,以控制其不超过结点的能力范围,否则 SmartBoa 的不定大小的路由表设计就没有意义.本节讲述 routing entries 如何维护,其他部分的维护在后面几节中讲述.

由于 P2P 结点在离开系统时不通知其他结点,所以,维护路由表的第 1 步的工作是发现结点离开的事件.

注意到 SmartBoa 结点集按照结点特征串的不同划分成多个子集,一个子集中的任一结点都知道该子集中的所有结点,也就是说,子集内的结点全连通.规定结点 M 总定期探测$\{\alpha_M\}$中 M 右边第 1 个结点(不妨设为 N),若发现 N 已离开,则报告给 M 的某个顶结点 A(由于 M 和 N 的特征串相同,A 也必然是 N 的顶结点),A 负责将 N 离开的消息组播给所有 routing entries 中含有 N 的结点.

类似地,结点在新加入系统或改变级别时,也需通知它的某个顶结点,以进行消息组播.这种通知顶结点某个结点发生变化的消息称作“变更汇报”消息.

下面介绍 SmartBoa 的核心:组播算法.

首先,在系统某时刻,一个关于结点 M 的更新消息需要被组播到的范围(称“目标结点集”)仅决定于 M,记 M 的 128 位从高到底依次为 $M_{128},M_{127},\ldots,M_3,M_2,M_1$,则目标结点集为集合 $\{\varnothing\},\{\text{“}M_1\text{”}\},\{\text{“}M_2M_1\text{”}\},\ldots,\{\text{“}M_{128}M_{127}\ldots M_3M_2M_1\text{”}\}$的并集,记为 T_M.

显然,$\{\varnothing\}$中结点的 routing entries 中记录有 T_M 中的全部结点,$\{\text{“}M_k\ldots M_1\text{”}\}$中结点的 routing entries 中必然记录了所有$\{\text{“}M_j\ldots M_1\text{”}\}(k<j)$中的结点.这样,在组播算法的设计中,只要保证在组播过程中不要求一个结点把消息发送给比自己级别高的另一个结点,消息即可从强结点向弱结点逐级传播,散播到 T_M 中的全部结点上.

算法的伪码描述如图 3 所示.图 4 是一个示例,这里假设 M=10010110,首先得到通知的是它的一个顶结点 T=00000000.消息的散播过程是:第 1 步,T 将消息转发给目标结点集中的一个 nodeId 末位与自己不同的结点 T'=10000011;第 2 步,T 和 T'分别把消息转发给 T_M 中的一个 nodeId 末位与自己相同,而倒数第 2 位与自己不同的结点;依次类推,第 n 步,所有已经得到该消息的结点将消息再转发给 T_M 中的一个 nodeId 后 n−1 位与自己相同,而倒数第 n 位与自己不同的结点.注意,算法规定每一步结点在 routing entries 中找寻满足条件的结点时,如果没有合适的结点(这说明 T_M 中没有这样的结点存在),则无须动作,如图 3 中的(3)所示;如果有多个合适的结点,则选取其中级别最高的一个,如图 3 中的(4)所示,这样就严格保证了消息总是从强结点向弱结点传播.

```
rcv_bcast(nodeId=M, step=s):                    //receive an event message of M at step s
(1)  Rs=getRelevant(routing_entries, M)         //get all the entries that needs to know the even for i:=s+1 to 128 do
(2)    Rn:=getSuffix(Rs,my_nodeId,I–1)          //get entries of Rs whose last i–1 bits are same, // but the ith bit is
       different with the local nodeId
(3)    if Rn=null then
           continue
       fi
(4)    P:=getHighestLevel(Rn)                   //get the entry that has the highest level
(5)    send_bcast(P,M,i)                        //send the message to node of the entry
       od
```

Fig.3 Pseudo codes of the multicast

图 3 组播算法的伪码表示

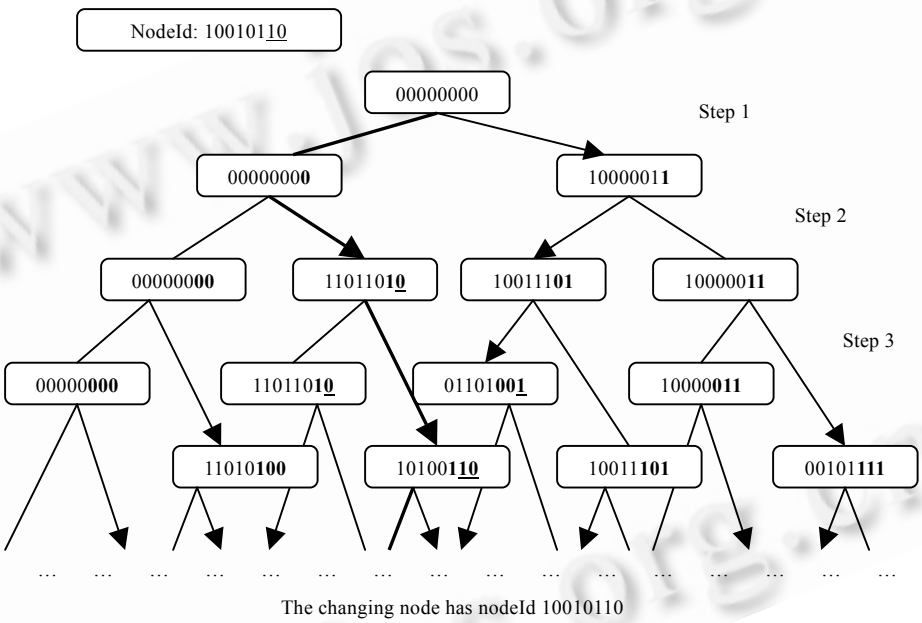

Fig.4 An illustration of multicast

图 4 消息组播过程的示例图

2.2 Leafset entries的维护

Pastry 结点为维护 leaf set,定期发消息给 leaf set 中所有结点,由于 leaf set 是对称的,如果某个结点没有按时发来消息,则专门发消息进行询问,如果询问消息没有回应,则认为该结点已离开,从 leaf set 中摘除.在使用这种方法时,在一般情况下(leaf set 中 32 个结点,每 30s 发一次消息),为维护 leaf set,一秒钟要发送一个消息.

SmartBoa 采用更节省开销的维护算法:结点 M 只定期发消息给左右两个相邻结点 L 和 N,按照对称性,如果某个相邻结点(例如右边的 N)没有按时发来消息,则发询问消息,若无回应,则认为 N 已离开.M 将 N 从 leafset entries 中去除后,需向 leafset entries 中右侧的另一结点索要 leafset entries,以补足自己的 leafset entries 至 $l/2$ 项.之后,M 还需将 N 离开的事件通告给 leafset entries 的左半边的所有结点,并附上 leafset entries 的右半边.收到该通告的结点将 N 从自己的 leafset entries 中去除,并根据 M 发来的结点集修正自己的 leafset entries.当 M 发现左边相邻结点 L 离开时,依对称方法处理.

与 Pastry 一样,SmartBoa 的这种维护算法保证结点在 nodeId 环上紧密连接,除非 $l/2$ 个相邻结点同时离开,连接不会被破坏.

2.3 Top entries的维护

top entries 中记录着当前级别最高的几个结点,统计数据表明,高带宽的用户常常也是在线时间长的用户,因此 top entries 中的结点失效的可能性很小,并且 top entries 不常被使用,故而使用懒维护.

M 向其顶结点发送变更汇报时需要顶结点回应,并且回应消息中应附加当前可用的 t 个其他顶结点(顶结点之间必然全连通),用以更新 top entries;若变更汇报没有被回应,则 M 将消息重新发给 top entries 中的其他顶结点;若 top entries 中所有结点均不回应,则向一个与特征串同为 α_M 的结点或 M 的一个超结点(一般情况下,routing entries 中会包含一些超结点,如图 1 中的"00110011"项)索要 top entries,继续进行变更汇报.

如果当前系统中某顶结点 T 的级别为 l_1,此时有它的级别为 $l_0(l_0<l_1)$的超结点 T_0 新加入系统,T_0 应取代 T 及其他相关顶结点成为新的顶结点.T_0 在完成路由表构造后,将自己的到来告知其 routing entries 中所有的原顶结点.这些结点将 T_0 记入 top entries,以后再收到变更汇报时,在回应消息中附加入 T_0,并将变更汇报直接转发给 T_0.

2.4 Finger entries的维护

结点定期探测 finger entries 中的结点,这一探测需要回应.若结点 M 发现 finger entries 中某项失效(不妨设失效项应为某地址 N 的受理结点),则在 leafset entries 中找到级别比自己低或者与自己相同的某结点 C,向 C 索要其路由表中离 N 最近的结点 F (一般情况下,F 在 C 的 finger entries 中),当前 N 的受理结点应就在 F 附近,通过 F 的 leafset entries 即可查知.若 M 的 leafset entries 中都是级别高于自己的结点,则直接发起目标为 N 的路由找到 N 的受理结点即可.

3 其他问题

3.1 结点加入、级别设定与调整

新加入系统的结点首先完成路由表各部分的构造,然后将自己到来的消息通过第 2.1 节中的组播协议通知到所有相关结点.下面论述新结点如何构造其路由表.

设新结点为 X,为加入系统,它首先连接系统中的某已有结点 B,称"bootstrap 结点",然后完成下述 5 步操作:

(1) 确定 X 的初始级别.假设 B 的级别为 k_B,B 统计其近期运行中用以路由表维护的下行带宽为 W_B,设 X 规定能用以 SmartBoa 维护的带宽为 W_X,则可以估算 X 能承受的最高级别为 $k_{\max}=\left\lceil k_B+\log_2\frac{W_B}{W_X}\right\rceil$.事实上,一般 X 规定的初始级别会比 $k_{\max}$ 低很多,X 在开始运行后再提高自己的级别,以达到最佳状态,这个过程称为"慢启动",在下一节讨论.

(2) B 发起以 X 为目标地址的路由,最终到达 Z 结点,Z 结点将会成为 X 的邻居结点,X 根据 Z 的 leafset entries 构造自己的 leafset entries.

(3) B 的某顶结点为 T_B,X 通过 T_B 找到某结点 P,满足 $X>P$ 或 $P>X$ 或 $\alpha X=\alpha P$.由于 T_B 的级别很高,因此 T_B 的 routing entries 中很可能含有 P,如果不含,则顺着 T_B 的 leafset entries 在 nodeId 环上寻找.P 的顶结点亦为 X 的顶结点,记为 T_X,X 向 T_X 索要自己的 t 个 top entries.

(4) X 从 T_X 处下载 routing entries.这一下载需要占用比较大的带宽,因此,为缓解压力,T_X 可以将下载任务重定向到 X 的其他超结点或与 X 特征串相同的结点,甚至可以同时从多个结点下载 routing entries 的不同部分.采用什么样的下载算法能够最好地平衡结点压力是个值得研究的问题,但本文对此不再做进一步的讨论.

(5) routing entries 下载完毕后,X 即可知道 finger entries 中应含有哪些项,X 通过 leafset entries 中结点的帮助查知这些项,方法类同于第 2.4 节中的 finger entries 维护方法.

至此,路由表的 4 个部分都已构造完成,X 向 T_X 发出变更汇报后即可开始正常工作.

一个特殊情况是新加入的结点级别很高,系统中不存在它的顶结点.由于慢启动的原因,新结点总是从一个很低的级别开始运行,故这种情况不必讨论.

结点在运行过程中,发现能力不足时可以下调自己的级别,发现能力有余时可以提高级别.一般说来,能力的瓶颈在于带宽,当然,也受到 CPU、内存等资源的限制.结点还可能由于 SmartBoa 运行时路由消息过多,产生的带宽压力过大而降低级别.总之,结点可以随时灵活地调整自己的级别.在本文的后续讨论中,我们通常把 SmartBoa 的带宽限制在结点全部带宽能力的 10%,我们认为,这样基本可以保证 SmartBoa 的运行不影响用户的其他网络活动.事实上,结点带宽大多数时间处于闲置状态,可以充分利用可用带宽运行 SmartBoa,当用户需要下载数据时,将结点级别调低,保证下载速度,下载完毕后又可重新将级别调高.但这种方法引发结点级别的频繁变动,每次变动都会引发消息组播,增大维护消息的总数,成为其他结点的负担,因此并不十分提倡.

结点调低级别时,只需缩小 routing entries,扩大 finger entries 即可,填补 finger entries 中空缺项的方法仍类似于第 2.4 节,之后向顶结点发送变更汇报.调高级别时还需下载 routing entries 中的不足部分,可以从顶结点下载,也可以从其他超结点处下载.如结点本身已是顶结点,还需继续提高级别,则通过 leafset entries 找到尾串适宜的结点,通过它的顶结点下载所需的 routing entries 项,之后通告相关的原顶结点(参见第 2.3 节).例如,某特征串为“1”的顶结点 T 需要提高级别时,在 leafset entries 中找到某以“0”结尾的结点,它的顶结点若特征串为“0”,则可,若为“00”,则还需继续找到特征串为“01”的结点,联络它的顶结点,这一查找过程一直继续下去,直到 T 下载到全部结点的指针.由于顶结点的覆盖面已经很大,且 nodeId 均匀分布,这种查找很快就可以完成.

3.2 慢启动

one-hop overlay 的一个重要缺陷是启动慢.在一个 100 000 结点的 P2P 系统中运行 one-hop overlay,假设一个指针的大小为 200 位(至少需要 nodeId128 位,ip32 位,端口 32 位),则路由表的大小为 20Mb,对于一个 modem 连接的结点,即使能够用上全部下行带宽(64Kbps),在初始化时下载路由表也需要 5 分钟.如果系统规模达到 1 000 000 结点,则需要 50 分钟,这显然是用户难以忍受的.

SmartBoa 结点使用慢启动来解决这个问题.如上一节所述,结点在加入系统时可以通过 bootstrap 结点给出的参考信息估算出自己能承受的级别,进一步估算路由表的大小(对照 bootstrap 结点的级别及其路由表的大小即可知).这里我们做一个简单的估计:假设结点规模为 1 000 000,结点的平均在线时间为 1 小时,每个结点在线期间平均发生 3 次级别调整,那么每秒钟有 1000000×5/3600≈1389 个结点变化消息被组播,对于一个 modem 结点,如果我们使用其 10%的带宽能力(6.4Kbps)进行维护,一个更新消息的大小约为 500 位(指针大小+ip 包头+UDP 包头),那么它每秒能承受的消息数约为 12 个,因此能维护系统结点总数的 12/1389≈1/116,可设级别为 7,记录结点数为 1000000×1/128≈7813 个,大小为 1.56Mb,若使用全部 6.4Kbps 下载,也需 4 分钟.为减少启动时间,在启动时,结点先规定一个较小的级别(这里可设为 12),保证下载可以在 10s 之内完成,开始正常工作后,后台继续下载路由表,若使用全部带宽的 40%,可在 10 分钟左右完成,之后调回到正常级别继续工作.对于强结点,初始级别与最终的运行级别相差很大,可分几次逐步提高级别.

4 研究背景与相关工作

Structured overlay 最初被提出来是为了解决 unstructured overlay 扩展性不好的问题,主要算法包括 Tapestry,Pastry,CAN,Chord.Tapestry 和 Pastry 类似,路由表第 i 行记录 nodeId 与自己前 i–1 位相同,第 i 位不同且互不相同的 2^b–1 个结点(nodeId 看做是基于 2^b 的整数).CAN 将一个 d 维的虚拟空间分块,每个结点掌管其中一块,数据的标识哈希后总落入其中某块,相应数据就存放在掌管该块的结点上.Chord 的路由表记录 nodeId 环上的一系列折半指针.这些算法的特点是路由表小,易于维护,且路由保证在 $O(\log(N))$(Tapestry,Pastry 和 Chord)或 $O(N^{1/d})$ 跳完成,因此算法的扩展性比 unstructured overlay 好.structured overlay 的出现得到了更为广泛的关注.

在理论上,Xu[7]证明了就 CAN,Chord,Pastry 的路由表大小而言,它们所获得的路由效率(指跳数)已经在数量级上达到最优;Gummadi 等人[8]讨论了各种算法所使用的拓扑结构与系统对结点失败的适应性和对近邻选择的兼容度之间的关系;Loguinov 等人[9]用图论的观点比较了 CAN,Chord 和 de Brujin 图的路由性能和容错性.

之后,一些新的 structured overlay 算法被提了出来,但都没有离开根本的设计思想:(1) 每个结点只记录很少一些指针,保证维护开销很小;(2) 确保路由跳数在一定的范围内,使得路由速度不会比结点之间直接通信慢

很多.针对这两个方面,又出现很多改进性的研究.

但是,从另一个角度来看,维护开销小也可以认为是浪费了结点的可用带宽.充分利用带宽可以扩大路由表,提高路由速度.于是又有使用大规模路由表的算法被提了出来.Kelips[10]将结点分成 k 组,组内结点全连接,通过 gossip 的方式进行更新,这样维护开销增大很多,但是路由可以在 2 跳完成.Gupta[6]又进一步指出,实际上以当前的 P2P 系统规模和结点变化频率来计算,即使所有结点全连接也是可行的,并且给出了具体的算法(one-hop overlay),但是算法的扩展性差,对强结点有数目要求且压力很大,并且存在启动时间过长的问题.

正是在这种背景下,我们提出 SmartBoa,试图对 Pastry 和 one-hop 两种类型的结构化覆盖网进行综合,不让算法本身对结点有任何强行要求,而尽力充分利用异构性非常强的 P2P 结点的整体带宽能力,使路由效率达到最优.SmartBoa 可以成为普适于各种网络环境的新型结构化覆盖网.

5 总 结

本文提出一种新的 P2P 路由算法 SmartBoa,其特点可以包括以下几个方面:(a) 充分利用结点允许的带宽;(b) 任何结点都可以加入系统,对结点没有带宽等任何方面的要求;(c) 适用于各种系统环境;(d) 结点级别可动态调节,适应网络情况的变化;(e) 简单、高效的路由;(f) 路由算法不会对部分结点产生过大压力;(g) 高可扩展性;(h) 可以通过慢启动避免启动时间长的问题.

我们的后续研究包括:(1) 容错性更高的组播算法;(2) 更合理的分布式路由表下载策略;(3) 更高效的路由方案.

References:

[1] Zhao B, Kubiatowicz J, Joseph A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Berkeley: Computer Science Division, 2001.

[2] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. of the Int'l Conf. on Distributed Systems Platforms. 2001. http://research.microsoft.com/~antr/Pastry/

[3] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Proc. of the SIGCOMM 2001. 2001. http://www.acm.org/sigs/sigcomm/sigcomm2001/

[4] Cox R, Muthitacharoen A, Morris R. Serving DNS using chord. In: Proc. of the 1st Int'l Workshop on Peer-to-Peer Systems. 2002. http://www.cs.rice.edu/Conferences/IPTPS02/

[5] Jain S, Mahajan R, Wetherall D. A study of the performance potential of DHT-based overlays. In: Proc. of the 4th USENIX Symp. on Internet Technologies and Systems. 2003. http://www.usenix.org/events/usits03/

[6] Gupta A, Liskov B, Rodrigues R. One hop lookups for peer-to-peer overlays. In: Proc. of the 9th Workshop on Hot Topics in Operating Systems. 2003. http://www.usenix.org/events/hotos03/

[7] Xu J. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In: Proc .of the 22nd Annual Joint Conf. of the IEEE Computer and Communications Societies. 2003. http://www.ieee-infocom.org/2003/

[8] Gummadi K, Gummadi R, Gribble S, Ratnasamy S, Shenker S, Stoica I. The impact of DHT routing geometry on resilience and proximity. In: Proc. of the SIGCOMM 2003. 2003. http://www.acm.org/sigs/sigcomm/sigcomm2003/

[9] Loguinov D, Kumar A, Rai V, Ganesh S. Graph-Theoretic analysis of structured peer-to-peer systems—routing distances and fault resilience. In: Proc. of the SIGCOMM 2003. 2003. http://www.acm.org/sigs/sigcomm/sigcomm2003/

[10] Gupta I, Birman K, Linga P, Demers A, van Renesse R. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In: Proc. of the 2nd Int'l Workshop on Peer-to-Peer Systems. 2003. http://iptps03.cs.berkeley.edu/