Emergint:一种支持多节点并发动态增删的P2P路由算法

韩 华,代亚非,李晓明
(北京大学信息学院网络实验室,北京 100871)

摘 要: 针对P2P网络动态构建问题,提出了一种能够支持多节点并发动态加入和退出的P2P路由算法.该算法具有如下特点:(1)自治性:节点的增删不需要统一控制;(2)动态性:节点的增删不影响其他节点路由过程的正确性;(3)并发性:多节点可以同时加入和退出系统,彼此不干扰.该算法的路由延迟为logN.模拟测试结果表明:该算法的平均RDP为115,增删节点的代价为O(logN).

关键词: P2P; 路由算法; 并发计算
中图分类号: TP393 文献标识码: A 文章编号: 037222112(2004)0921579206

Emergint: A P2P Routing Algorithm That Supports Multi2Node Dynamic Concurrent Join and Leave

HAN Hua, DAI Ya2fei, LI Xiao2ming
(Network Laboratory, Information Science and Technology Institute, Peking University, Beijing 100871, China)

Abstract: Emergint is a P2P routing algorithm which aims at the issue of P2P network dynamic construction. Emergint has 3 characteristics: (1) autonomy: node insert or delete does not need the control of central coordinator; (2) dynamicity: node insert or delete does not affect the correctness of the ongoing routing process of other nodes; (3) concurrency: multiple nodes can join or leave the system at the same time without any interference with each other. We examined the performance of Emergint by network simula2 tion. Examination result shows: The RDP of Emergint is approximately 1.5. The overhead of node insert or delete is O(logN).

Key words: P2P; peer to peer; routing algorithm; concurrency

1 引言

P2P路由算法是P2P网络研究的一个基本问题.目前,人们已经提出了几种P2P路由算法,如CAN[2],Chord[3]和Pas2 try[4].这些算法给出了各自的基本路由方法,但是它们对P2P网络的动态构建,即:节点的动态增加与删除,均没有给出完善的解决方案,从而影响了算法的实用性.

本文提出的Emergint算法给出了P2P网络动态构建的一种解决方法.Emergint算法建立在经典算法[1]的基础之上,并对其进行了改进和扩充.Emergint算法有三个特点:(1)各节点的加入和退出完全是自治的,不需要一个全局管理者的控制;(2)节点的加入与退出是动态的,不影响系统的正常路由;(3)多个节点可以同时加入和退出,并且互不干扰.这三个特点保证了,采用Emergint算法,实际系统可以自发地构建出一个P2P网络,并且在运行过程中自发地对网络进行维护.

Emergint算法的基本路由方法与算法[1]相似,一次路由过程的最大延迟为logN,其证明可以参见算法[1],本文不再赘述.采用网络模拟的方法,我们对Emergint算法的性能进行了测试.测试结果表明:Emergint路由的平均RDP(Relative De2 lay Penaty)为115节点加入与退出的代价为O(logN).

2 系统模型定义

211 基本运算符

ó begin(x, l):整数x的l位前缀(左为前)

ó end(x, l):整数x的l位后缀

ó at(x, i):整数x的第i位(i>=0)

212 关于节点和P2P网络

ó N:系统能够支持的最大节点数. $N=B^{L_n}$, B和L_n均为小整数,例如:B=8, L_n=4

ó nid:节点的唯一标识符,为一随机生成的B进制L_n位整数

ó $n_{left}(n)$:节点n的左邻居节点. $n_{left}(n)=\{n_x | n_x$是系统中现有节点, $nid_x<nid$且nid_x与nid相邻$\}$

ó $n_{right}(n)$:节点n的右邻居节点. $n_{right}(n)=\{n_x | n_x$是系统中现有节点, $nid_x>nid$且nid_x与nid相邻$\}$

ó 节点路由表构造方法:

每个节点拥有一个路由邻居节点列表,该列表包含L_n* B * C个路由邻居节点表项,其中C为大于1的小常数(如3),每个邻居表项保存该邻居节点的IP地址.用向量(i, j, k)为节点的邻居节点编号,0<=i<L_n, 0<=j<B, 0<=k<C.

收稿日期:2003210227;修回日期:2004204228
基金项目:国家自然科学基金(No.60303002);国家863项目基金(No.2001AA111013)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

用 $it_{i,j,k}(x)$ 表示节点 x 的第(i, j, k)邻居节点. 若 $it_{i,j,k}(x) = y$, 则节点 y 具备如下条件: (1) $begin(nid_y, i) = begin(nid_x, i)$; (2) $at(nid_y, i) = j$; (3) y 是满足上述两个条件且与 x 节点距离第 k+1 近的节点

Ó 节点反向路由表构造方法:

每个节点还有一个反向路由邻居列表, 用来保存所有以它为邻居的节点. 令 $rit_i(x)$ 表示节点 x 的第 i($0 <= i < L_n$)组反向路由邻居节点. 若 $rit_i(x) = s$, 则节点集合 s 具备如下条件: P y I s, v j 和 k, $0 <= j < B$, $0 <= k < C$, 使得 $it_{i,j,k}(y) = x$

Ó (b, e): 一个路由表项在路由表中的区域. 路由表项 $it_{i,j,k}$ 在路由表中的区域为(b, e), 其中, b 为路由表第 i 列中, 纵坐标小于 j 且不为空的相邻表项的纵坐标; e 为路由表第 i 列中, 纵坐标大于 j 且不为空的相邻表项的纵坐标

说明: Emergint 的路由表构造方法与算法[1]的路由表构造方法不同之处在于: 算法[1]是基于 nid 的后缀来构造; Emergint 是基于 nid 前缀来构造. 一个节点的数据结构见图 1.

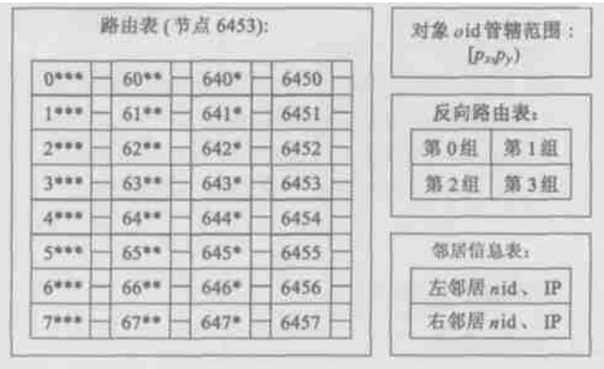

图 1 节点上的数据结构($B = 8$, $L_n = 4$, $C = 2$)

2.3 关于系统对象

Ó oid: 系统中对象的唯一标识符, 为一随机生成的 B 进制 L_o 位整数, $L_o >= L_n$

Ó $nid_{dst}(oid)$: 对象 oid 的目标 nid, 令 $nid_{dst}(oid) = begin(oid, L_n)$

Ó LeftNeighbor(oid): 系统已有节点中, nid 小于 $nid_{dst}(oid)$ 且最接近 $nid_{dst}(oid)$ 的节点

Ó RightNeighbor(oid): 系统已有节点中, nid 大于 $nid_{dst}(oid)$ 且最接近 $nid_{dst}(oid)$ 的节点

Ó root(oid): 对象 oid 对应的根节点的 nid. 令 root(oid) 为: 如果系统已有节点中存在 nid 数值与 $nid_{dst}(oid)$ 数值相等的节点, 则该节点是对象 oid 的根节点. 否则, 根节点是 LeftNeighbor(oid) 和 RightNeighbor(oid) 中的一个, 具体是哪个要依赖于具体的路由过程.

说明: 如图 2 所示, 由于 nid 和 oid 都是随机生成的, 因此 nid 和 $nid_{dst}(oid)$ 均匀地分布在 nid 整数空间中. 每个对象都能找到相应的根节点. 每个节点所拥有的以其为根节点的对象数量大致相等. P2P 路由算法的基本任务是: 从系统中任何一个节点出发定位对象时, 将定位请求发送到该对象所对应的根节点上.

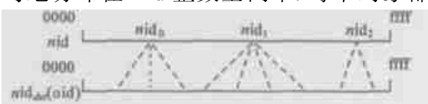

图 2 对象 oid 与节点的根对应关系

3 基本路由算法及其特性分析

Emergint 算法由一系列子算法构成, 其中包括基本路由算法、节点加入算法、节点退出算法和对象 oid 分配算法. 各子算法的相互关系如图 3 所示.

图 3 节点上的算法结构

3.1 基本路由算法

算法 1: MSG(oid, t, s, r): 节点之间发送的消息. 其中, oid 表示目标对象的 oid, t 表示消息类型(t = 1、2 或 3); s 表示已经转发的步数; r 表示用户请求的实际内容.

send(d, m): 将消息 m 直接发送给节点 d.

模块 1: 用户发送针对对象 oid 的请求 r(r 为/ 插入0、/ 删除0 或/ 访问0)

Deliver(oid, r) {
    m = MSG(oid, 1, 0, r);
    send(localhost, m);   /* 发送给本地节点 */
}

模块 2: 节点接收到消息 m 后的转发机制

Routing(m) {
    if (m.s 等于 L_n)
        路由结束, 调用 Accept(m); /* 到达根节点 */
    i = m.s;
    switch (m.t) {
    case 1:   /* 处理 1 类请求 */
        j = at((nid_dst(m.oid)), i);
        if($it_{i,j,0}$ 不为空) {
            send($it_{i,j,0}$, MSG(m.oid, 1, m.s+1, r));
        }else{   /* 对空表项的处理 */
            $j_1 = min(\{j_x | 0 <= j_x < j$ 且 $it_{i,j,0}$ 不为空$\})$;
            $j_2 = min(\{j_x | l_n > j_x > j$ 且 $it_{i,j,0}$ 不为空$\})$;
            if(j_1 和 j_2 都存在) {
            if(| j_1 - j | < | j_2 - j |)
            send($it_{i,j1,0}$, MSG(m.oid, 2, m.s+1, r));
            else
            send($it_{i,j2,0}$, MSG(m.oid, 3, m.s+1, r));
            }else{
            if (j_2 不存在)
            send($it_{i,j1,0}$, MSG(m.oid, 2, m.s+1, r));
            else
            send($it_{i,j2,0}$, MSG(m.oid, 3, m.s+1, r));
    }}
case 2:   /* 处理 2 类请求 */
    j = max($\{j_x | it_{i,jx,0}$ 不为空$\}$);
    send($it_{i,j,0}$, MSG(m.oid, 2, m.s+1, r));
    case 3:   /* 处理 3 类请求 */
    j = min($\{j_x | it_{i,jx,0}$ 不为空$\}$);

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
    send(it_{i,j,0}, MSG(m.oid, 3, m.s+1, r));
}}
    模块 3: 根节点消息处理
Accept(m){
    If (m.oid 属于自己管辖的对象 oid 范围){
            路由成功;
    }else{
if (m.oid 属于左邻居节点 n_left管辖的对象 oid 范围)
            send(m, n_left);
    else
            send(m, n_right);
    路由成功;
}}
```

算法说明: 用户调用 Deliver(oid, r) 向一个对象发出请求. 不管从系统中哪个节点出发, 用户只要知道对象的 oid, 路由算法就可以将请求转发到 root(oid). 模块 3 是路由的目标节点 root(oid) 进行消息处理的方法, 在第 4 节中对它进行详细说明.

模块 2 中, 如果消息转发过程没有碰到空表项, 则转发处理与算法[1] 相同; 如果碰到了空表项, 算法在路由表的相同列中寻找距离该空表项最近的非空表项, 其目的是为了将请求发送到对象 oid 的根节点 root(oid) 上. 碰到空表项后, 后续转发的消息都是 2 类消息或 3 类消息: 2 类消息处理模块将消息发送到 LeftNeighbor(oid), 3 类消息处理模块将消息发送到 RightNeighbor(oid).

312　基本路由算法(算法 1)的三个性质

性质 1　如果系统中一个节点的路由表项 $it_{i,j,0}$ 为空, 则系统中所有 nid 与该节点 nid 具有相同 i 位前缀的节点的路由表项 $it_{i,j,0}$ 也为空.

性质 2　在针对对象 oid 的路由过程中, 设在所有路由路径上, 从第一次碰到空表项到根所经过节点集合为 S, 则 S 中所有节点的 nid 都与 nid_{dst}(oid) 具有最大长度的相同前缀.

性质 3(路由表项复用定理)　对一个节点 n 而言, 设系统其他节点中与其 nid 有 l 位相同前缀的节点集合为 S(0< = l< L_n). 如果 n_x I S, 则将节点 n_x 的路由表项 $it_{i,j,k}$(0< = i < = l, 0< = j< B 且 j< > at(nid, i), 0< = k< C) 作为节点 n 的路由表项 $it_{i,j,k}$, 也同样是正确的.

性质 1、性质 2 和性质 3 的证明见文[9]. 这三个性质是节点动态增删算法的基础.

313　对象 oid 在根节点的分布

插入对象时, 调用基本路由算法, 对象定位信息会按照对象 oid 分布到相应的根节点上. 分析对象 oid 与根节点的对应关系: 设 n_x、n_y 是系统中 nid 数值相邻的两个节点, 它们的 nid 分别为 nid_x 和 nid_y(nid_x< nid_y). 根据路由算法, 在 nid_{dst}(oid) 属于[nid_x, nid_y] 的对象中, 一部分以 n_x 为根节点, 一部分以 n_y 为根节点. 对于这些对象, 令 p_x 为以 n_x 为根节点的对象 nid_{dst}(oid) 所在区域, p_y 为以 n_y 为根节点的对象 nid_{dst}(oid) 所在区域. p_x 和 p_y 的求解过程如下:

分析针对 nid_{dst}(oid) I [nid_x, nid_y] 的对象的路由过程: (1) 对于 nid_{dst}(oid)= nid_x(或 nid_y) 的对象, 路由过程不会碰到空表项, 它会被发送到节点 n_x(或 n_y) 上, 从而可知: nid_x I p_x, nid_y I p_y; (2) 对于 nid_{dst}(oid) 属于(nid_x, nid_y) 的对象, 路由过程会碰到空表项. 设路由过程中第一次碰到的空表项为 $it_{i,j,0}$, 它在路由表的第 i 列中的区域为(b, e). 由于本节点肯定会在路由表第 i 列中占据一个表项, 因此, b 和 e 至少有一个不为空. 根据(b, e) 的不同类型, 系统中所有的路由过程可以分为三类, 根据路由算法可以得到相应的 p_x 和 p_y:

(1) b 和 e 都不为空(式(1))

p_x= [nid_x, (begin(nid_y, i) * B + õ(b+ e)/28) * $B^{L_n - i - 1}$)

p_y= [(begin(nid_y, i) * B + õ(b+ e)/28) * $B^{L_n - i - 1}$, nid_y]

(2) b 为空, e 不为空(式(2))

p_x= [nid_x, begin(nid_y, i) * $B^{L_n - i}$)

p_y= [begin(nid_y, i) * $B^{L_n - i}$, nid_y]

(3) e 为空, b 不为空(式(3))

p_x= [nid_x, (begin(nid_y, i) + 1) * $B^{L_n - i}$)

p_y= [(begin(nid_y, i) + 1) * $B^{L_n - i}$, nid_y]

分析边缘情况: 当 nid_x< 0(或 nid_y> = N)(式(4))

p_x= NULL, p_y= [0, nid_y] (或 p_x= [nid_x, N - 1], p_y= NULL)

注意到在上述各式中, i 是由 nid_x 和 nid_y 决定的, 与 nid_{dst}(oid) 无关, 因此可以给出根据 nid_x 和 nid_y 来确定 p_x 和 p_y 的算法, 如算法 2 所示. 算法 2 的返回结果是: 为了计算 p_x 和 p_y, 代入式(1)、式(2) 和式(3) 的 i 值. 使用算法 2, 一个节点可以根据自己和相邻节点的 nid, 计算出以自己为根节点的对象 oid 范围, 这对于后面介绍的 P2P 网络的动态构建方法至关重要.

算法 2: 对象 oid 在两个相邻节点之间的分配情况

```
int allocateOID(nid_x, nid_y){
if (nid_x< 0)
    {按照公式 4 处理; return min({i| at(nid_y, i)< > 0, 0< = i< L_n});}
if (nid_y> = N)
    {按照公式 4 处理; return min({i| at(nid_y, i)< > B- 1, 0< = i< L_n});}
if (nid_y- nid_x 等于 1)
    {p_x= nid_x; p_y= nid_y; return- 1;}
f= LOOK_FOR_CROSS;
for(i = 0; i< L_n; i+ + ){
    if (f 等于 LOOK_FOR_CROSS){
    if (at(nid_x, i) 等于 at(nid_y, i)){
        continue;
    }else{
        if ((at(nid_y, i)- at(nid_x, i))> 1){
            按公式 1 求得 p_x 和 p_y; return i;
        }else{
            f = LOOK_FOR_NULL; continue;
        }}
    }else{
    if((at(nid_x, i) 等于 B21) 并且(at(nid_y, i) 等于 0)){
        continue;
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

```
    }else{
        if(at(nid_x, i)< B- 1)
            按式(3)求得 p_x 和p_y; return i;
        if(at(nid_y, i)> 0)
            按式(2)求得 p_x 和p_y; return i;
}}}}
```

4 节点动态增删算法

节点动态增删算法的基本任务是:(1)设置新增节点的路由表,调整相关节点的路由表;(2)设置新增节点的对象 oid 管辖范围,调整相关节点的对象 oid 管辖范围.

对于设置节点路由表,考虑到在实际应用中,无法掌握节点两两之间距离的全局信息,因此采用一种渐进的方法处理.分析路由表构造方法(见 212),其中第 1 点和第 2 点是为了保证路由算法的正确性;第 3 点(对各表项的网络距离要求)是为了提高路由的效率,因此,关键在于建立路由表的 $it_{i,j,0}$ 表项.对于设置节点的对象 oid 管辖范围,我们利用算法 2 对相关节点进行计算,并重新分配.

我们采用/两阶段0策略来处理节点的加入与退出.第一阶段:为相关节点路由表设置 $it_{i,j,0}$ 表项,设置相关节点对象 oid 管辖范围,以确保路由算法的正确;第二阶段:为相关节点路由表设置正确的 $it_{i,j,k}$(1<= k< C)表项,以提高路由的容错性和效率.

411 节点加入算法

算法 3:节点增加

n_{new}:新增节点

$n_{closest}$:系统现有节点中,距离 n_{new} 最近的节点

AddNode(){

(1)随机生成节点 n_{new} 的 nid2nid_{new},n_{new} 采用 nid_{new} 初始化自己的路由表.

(2)节点 n_{new} 直接向 $n_{closest}$ 发送增加节点请求(nid_{new},ADD_NODE,IP_{new}).

(3)$n_{closest}$ 调用 Deliver(nid_{new}, r),其中 r=(ADD_NODE,IP_{new}).开始一次路由过程.

(4)在路由过程中,所有处理 MSG(nid_{new}, 1, l, r)的中间节点将路由表项 $it_{i,j,k}$(0<= i<= l, 0<= j< B 且 j<>at(nid, i), 0<= k< C)直接发送给节点 n_{new}.n_{new} 根据性质 3 将其填入相应的路由表项中.对于多余的表项,n_{new} 将其保存下来,以供第二阶段处理使用.

(5)路由终点为 root(nid_{new}),设其为 n_{root}.n_{root} 根据算法 2 计算 depth= allocateOID(nid_{new}, nid_{root}).假设 nid_{new}< nid_{root}(或 nid_{new}> nid_{root}),得到 p_{new} 和 p_{root}.根据 p_{new} 和 p_{root},n_{root} 将其上属于 n_{new} 的对象定位信息移到 n_{new} 上.然后,n_{root} 将请求 r 直接发送给左邻居节点 n_{left}(或右邻居节点 n_{right}).n_{left}(或 n_{right})同样会根据算法 2 计算 allocateOID(nid_{left}, nid_{new})(或 allocateOID(nid_{new}, nid_{right})),得到 p_{new} 和 p_{left}(或 p_{right}).n_{left}(或 n_{right})将其上属于 n_{new} 的对象定位信息移到 n_{new} 上.

(6)n_{root} 将 nid_{new} 加入路由表.并从 n_{root} 开始,将请求 r 沿着反向路由表的 rit_i 指针进行扩散,扩散步数为 depth(在(5)中计算),i 的取值依次为 B-1, B-2, , , B2depth.在扩散过程中,每一个接收到扩散请求的节点将 nid_{new} 加入路由表.

}

算法说明:节点加入的第一阶段处理工作分成三个部分:(1)设置新增节点的路由表;(2)调整相关节点的对象 oid 管辖范围;(3)调整相关节点的路由表.$n_{closest}$ 可以手工指定,也可以借助某些通用服务(如 sonar[6])来获得.到第 4 步结束,n_{new} 的路由表就已经构造好了,性质 3 保证了路由表项的正确性.在第 2 步,n_{new} 可以向系统中任何一个节点发送节点加入请求.n_{new} 之所以向 $n_{closest}$ 发送请求,是为了使得 n_{new} 生成的路由表项更接近于路由表构造方法中提出的距离要求.在第 5 步,根据算法 2,将对象定位信息在 n_{left}(或 n_{right})、n_{new} 和 n_{root} 之间进行重新分配.在第 6 步,由 n_{root} 发起修改系统现有节点空表项的工作.之所以扩散 depth 步,可以根据性质 2 获知.

412 节点退出算法

算法 4:节点退出

n_{delete}:退出节点

DeleteNode(){

(1)设 n_{delete} 负责的对象 oid 范围为:[x, y].n_{delete} 根据算法 2 计算 allocateOID(nid_{left}, nid_{right}),得到:在自己退出后,[x, y]中应该由左邻居节点 n_{left} 和右邻居节点 n_{right} 负责的 oid 范围.并按照算法 2 的计算结果将相应的对象 oid 相关信息直接转移到 n_{left} 和 n_{right} 上.

(2)n_{delete} 根据反向路由表通知所有的相关节点将 nid_{delete} 从各自的路由表中删除.然后,n_{delete} 再根据路由表通知所有的相关节点,将 nid_{delete} 从各自的反向路由表中删除.

}

算法说明:一个节点退出的第一阶段处理工作可以分为两部分:(1)调整相关节点的对象 oid 管辖范围;(2)调整相关节点路由表.对于第一部分工作,我们同样采用算法 2 来确定退出节点的左、右邻居节点上 oid 的分布情况,然后将退出节点上对象 oid 相关信息转移到左邻居和右邻居上;对于第二部分工作,我们根据退出节点的反向路由表,通知所有指向该节点的节点从路由表中删除相关表项.并根据退出节点的路由表,通知所有相关节点从反向路由表中删除相关表项.

413 路由表优化

完成第一阶段处理工作之后,新增节点 n_{new} 根据路由表,开始向系统中所有节点扩散通知消息,目的是告知所有节点它已经加入了系统.获得此通知消息且还没有意识到新节点加入的节点 n_x,会将 nid_{new} 加入路由表,并根据 IP_{new} 通知 n_{new},令其将 nid_x 加入路由表.然后继续扩散该通知消息,直到系统中所有节点都知道了新节点的加入.删除节点的第二部分处理工作与此类似.

5 节点增删过程路由分析

在一个节点加入(或退出)时,如果完成了第一阶段处理工作,路由算法的正确性就得到了保证.但是在处理过程中,路由的正确性没有保证,这样会大大降低算法实用性.以下分析存在的问题,并给出解决方法.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

511　节点加入和退出过程路由性质

性质 4　设 n_x 和 n_y 是系统现有节点中两个具有相邻 nid 的节点, 且 $nid_x < nid_y$. 新加入节点 n_{new} 的 nid 为 nid_{new}, 且 nid_{new} I (nid_x, nid_y). n_{new} 的加入过程只会对 nid_{dst}(oid) I (nid_x, nid_y) 的对象的路由过程有影响, 对 nid_{dst}(oid) | (nid_x, nid_y) 的对象的路由过程不会产生任何影响.

性质 5　设退出节点 n_{delete} 的左、右邻居节点分别为 n_x 和 n_y, 且 $nid_x < nid_y$. n_{delete} 的退出过程只会对 nid_{dst}(oid) I (nid_x, nid_y) 的对象的路由过程有影响, 对 nid_{dst}(oid) | (nid_x, nid_y) 的对象的路由过程不会产生任何影响.

性质 6　已知系统中已有节点的 nid 将 nid 空间分成一个个区间, nid 处于不同区间的多个节点同时加入和退出系统时, 加入和退出过程不会影响路由的正确性.

性质 4、性质 5 和性质 6 的证明见文[9].

512　对算法 1、算法 3 与算法 4 的补充

根据性质 4 和性质 5, 对算法 1 进行补充: 在每个节点上增加一个数据结构: 邻居节点信息, 用于保存本节点的左、右邻居节点信息, 见图 2. 邻居节点信息中包括 n_{left} 和 n_{right} 的 nid 和 IP. 基于邻居节点信息, 每个节点可以根据算法 2 计算出 nid_{dst}(oid) 属于 [nid_{left}, nid_{right}] 的对象 oid 在自己和左右邻居节点上的分布情况. 当由于系统中有节点加入(或退出) 引起该节点接收到不应该属于自己的请求时, 它可以根据对象 oid 在左右邻居节点上的分布情况, 将该请求转发到正确的节点 n_{left} 或 n_{right}, 从而保证了节点加入过程中路由的正确性. 具体方法见算法 1 的模块 3.

根据性质 6, 我们对算法 3 与算法 4 进行补充: 一个节点共有六个状态. 当一个节点要加入(或退出) 系统时, 首先要检查其左、右邻居的状态. 当左、右邻居都处于正常状态时, 节点可以执行加入(或退出) 操作. 当左、右邻居处于/ 邻居加入0 状态或/ 邻居退出0 时, 说明系统中有节点 n_x 正在执行加入或退出操作. 检查 nid_x 处于的区间, 如果 nid_x 与该节点的 nid 处于同一区间, 则不能执行该节点的加入(或退出) 操作; 如果 nid_x 与该节点的 nid 不处于同一区间, 则继续执行该节点的加入(或退出) 操作; 否则, 暂停执行该节点的加入(或退出) 操作, 直到左、右邻居都恢复到正常态, 再继续执行加入(或退出) 操作. 节点状态图如图 4 所示.

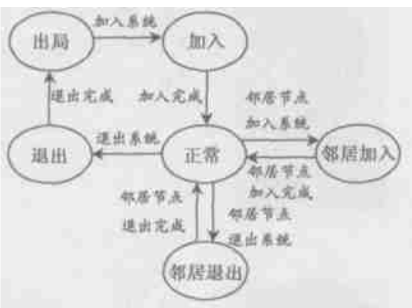

图 4　节点状态图

6　性能测试

我们采用网络模拟的方法对 Emergint 算法的性能进行测试. 我们对一个模拟 Internet 拓扑结构的软件包 GT-ITM[7,8] 进行改造, 使其能够生成网络末端节点. 然后用其生成一个包含 6054 个节点的 Internet 拓扑结构, 基于它实现了 Emergint 算法并进行了模拟运行, 得到了测试数据. 在以下所有的测试中, 设定 Emergint 的参数: B = 8, L_n = 4.

611　RDP

RDP(Relative Delay Penalty) 是衡量 P2P 路由算法性能基本指标, RDP 的定义是: 对于两个节点之间传递的一条消息, 令通过 P2P 路由算法传递所要经过的 hop 数为 a, 直接传递所经过的 hop 数为 b, RDP = a/ b. RDP 反映了 P2P 路由算法的路由效率. 在系统 P2P 网络构建过程中, Emergint 的 RDP 情况如图 5 所示. 从图 5 可以看出: 系统规模接近饱和时, 平均 RDP 为 115 左右; 系统规模处于非饱和状态时, 平均 RDP 小于 115, 与其他 P2P 路由算法类似[2~4]. 可见, Emergint 的路由效率是令人满意的.

612　增加和删除一个节点的代价

对于增加一个节点的代价, 我们用增加节点第一阶段处理中所需要通知的系统已有节点的节点数来衡量. 对于删除一个节点的代价, 我们用删除节点第一阶段所需要通知的系统中已有节点的节点数来衡量. 测试结果如图 6 所示. 可以看出, 增加节点和删除节点的代价相对于 logN (N 为系统已有节点数) 呈近似线性增长. 增加和删除节点的代价是 O(logN) 级的.

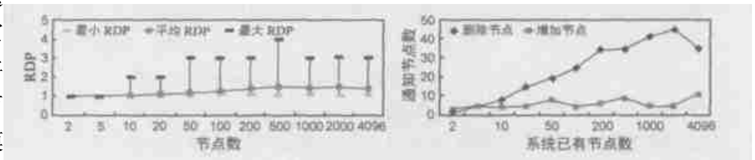

图 5　Emergint 的 RDP　　图 6　增加和删除一个节点需要通知的节点数

613　Emergint 优化过程中的 RDP

以上结果是在假设所有节点拥有理想的路由表项的情况下测得的. 但是我们知道, 在系统实际构建过程中, Emergint 需要经过一段时间的优化, 才能使得路由算法的效率达到最优. 图 7 反映了当系统有 1000 个节点时, 在系统优化过程中 RDP 的变化情况. 其中横轴表示每个节点掌握的系统已有节点的比例. 为了模拟最坏情况, 我们在构建这 1000 个节点的 P2P 网络时, 新加入节点从已有节点中随机选择一个作为自己的最近节点, 因此, 使得在优化之前 RDP 跨度较大. 从图 7 可以看出: 在系统的优化过程中, 虽然 RDP 变化较大, 但是平均 RDP 却保持相对稳定, 其值稳定在 2 左右. 可见, 系统优化过程中的路由性能也是令人满意的.

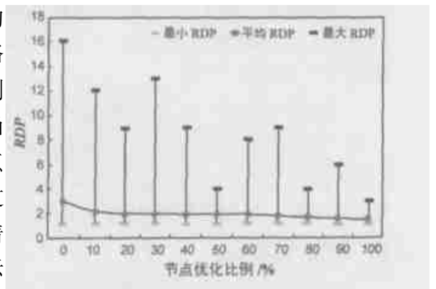

图 7　Emergint 优化过程中的 RDP

7　相关工作

算法[1]基于节点 nid 的后缀相同位构造 P2P 网络, 一次

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

路由过程的转发步数为 O(logN) . 该算法最大的特点是: 如果系统中的对象存在副本, 路由方法可以定位最近的对象副本. 其缺陷在于: 没有给出在实际网络环境中可行的 P2P 网络构造方法和维护方法.

Can[2] 根据几何平面划分的思想来建立 P2P 网络. 节点之间的通信消息按照节点所在几何平面的邻接关系进行传递, 实现路由过程. Can 对节点动态增加与删除考虑得不够周全. Can 的缺陷在于: P2P 网络的构建没有考虑(或考虑不够) 节点之间的物理网络距离, 无法定位最近的对象副本.

Chord[3] 通过节点 ID 的幂相邻关系建立起一种/ 弦环0 P2P 网络. 节点之间的通信消息按照幂相邻关系采用逐步趋近的方法进行传递, 实现路由过程. Chord 对节点动态增加与删除过程中的路由没有研究. 与 Can 类似, Chord 的缺陷也在于: P2P 网络的构建没有考虑节点之间的网络距离, 无法定位最近的对象副本. Can 和 Chord 比较适合于构造面向信息共享的 P2P 系统.

Pastry[4] 的基本路由方法与算法[1] 相同. 为了进一步提高路由算法的效率和可靠性, Pastry 增加了/ 物理邻居集合0 和/ 逻辑邻居集合0. Pastry 对系统处于非饱和状态(即路由表中存在空表项) 下的路由处理与 Emergint 不同, 对非饱和状态的路由最终目标节点没有进行透彻分析. Pastry 对节点动态增加与删除过程中的路由过程, 尤其是多节点并发增加与删除的过程没有研究.

Tapstry[5] 基于算法[1] , 并对其进行了改进. Tapstry 的改进工作主要分为三部分: (1) 节点的动态增加与退出方法; (2) 系统处于非饱和状态下的路由与多根节点容错方法; (3) 对象动态复制方法. 与 Emergint 相比: Tapstry 的节点动态增加与删除方法和系统处于非饱和状态下的路由方法会造成系统路由错误, 无法找到的对象的根节点. 另外, Tapstry 采用定期刷新的方法(Softstate) 维护对象定位信息, 这样会带来很大网络开销. 而 Emergint 算法通过主动调整对象 oid 管辖范围保证在任何时候都能找到对象的根节点, 从而保证路由的正确性, 而且不会带来大的网络开销.

值得一提的是: 在现有的 P2P 路由算法中, Emergint 是唯一对/ 节点动态增删过程中的路由情况0 和/ 多节点并发增删过程0 进行了研究, 并给出了处理方案的 P2P 路由算法. 有关 Emergint 算法在实际系统中的应用可以参见论文[9] .

8 结论和展望

针对 P2P 网络动态构建问题, Emergint 算法对经典路由算法[1] 进行了改进和扩充. Emergint 算法的改进工作主要体现在: (1) 重新定义了对象 oid 与节点的根节点对应关系, 调整了基本路由方法, 使得系统处于非饱和状态时能够为每一个对象提供正确路由; (2) 增加了节点加入和退出系统的算法, 支持各节点采用自治的方式动态地构建 P2P 网络, 并进行网络结构的自动维护. 该算法支持多节点并发加入和退出系统, 并能保证节点加入或退出的过程中路由过程的正确性. 采用网络模拟的方法, 我们对 Emergint 的性能进行了测试, 测试结果表明: Emergint 算法的平均 RDP 在 115 左右; 增加或删除一个节点的代价为 O(logN) ; 系统优化过程中的平均 RDP 大致为 2. 对 Emergint 算法的下一阶段研究的重点为错误模型和失效处理. 节点加入后的消息扩散过程也有待于日后进行深入研究.

参考文献:

[1] C G Plaxton, R Rajaraman, A W Richa. Accessing nearby copies of replicated objects in a distributed environment[J] . Theory of Comput2ing Systems, 1999, 32: 241- 280.

[2] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. A scalable content addressable network[A] . Proceedings of the ACM SIGCOMM 01 Conference[C] . San Diego, California: ACM, August, 2001.

[3] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, Hari Bal2akrishnan. Chord: A scalable peer2to2peer lookup service for internet applications[A] . Proceedings of the ACM SIGCOMM . 01 Conference [C] . San Diego, California: ACM, August 2001.

[4] A Rowstron, P Druschel. Pastry: Scalable, distributed object location and routing for large2scale peer2to2peer systems[A] . IFIP/ ACM Inter2national Conference on Distributed Systems Platforms (Middleware) [C] . Heidelberg, Germany: IFIP, November 2001. 329- 350.

[5] Ben Y Zhao, John D Kubiatowicz, Anthony D Joseph. Tapestry: An In2frastructure for Fault2Tolerant Wide2area Location and Routing[R] . USA: CS Dept, Berkeley, April 2001.

[6] Moore, J cox, S Green. Sonara Network Proximity Service (Internet2 Draft) [DB/ OL] . http: / / www. netlib. org/ utk/ projects/ sonar/ , Feb 1996.

[7] Ellen W Zegura, Ken Calvert, S Bhattacharjee. How to model an inter2network[A] . Proceedings of IEEE Infocom . 96[C] . San Francisco, CA: IEEE, 1996.

[8] K L Calvert, M B Doar, E W Zegura. Modeling internet topology[J] . IEEE Communications Magazine, June 1997, 35(6) : 160- 162.

[9] 韩华. 面向 Internet 的分布式海量文件存储系统研究[D] . 北京: 北京大学计算机系, 2002, 7.

作者简介:

韩 华 男, 1972 年出生于湖北武汉, 北京大学信息学院博士后, 主要研究方向: 网络存储和分布式计算.

代亚非 女, 1958 年出生于黑龙江哈尔滨, 北京大学信息学院教授, 主要研究方向: 网络存储和分布式计算.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net