分级有序 P2P 超级节点拓扑构造

冯劲潇 陈贵海 谢俊元
(南京大学计算机软件新技术国家重点实验室 南京 210093)

摘 要 拓扑构造是 P2P 网络研究中的核心问题之一。在当前的超级节点拓扑构造中,采用固定的两层结构和基于谣言的无序构造方式,不仅限制了系统性能,而且产生了过多的负载,使超级节点成为系统的热点。同时,无序构造方式也带来较高的成本和安全问题。据此,提出一种分级有序的超级节点拓扑构造方法(HOST),按照网络规模对超级节点进行自适应分级,并采用有序的节点加入和退出算法。模拟结果和分析表明,HOST 能有效控制超级节点的产生,平衡超级节点间负载,同时显著降低拓扑构造和拓扑修复过程中产生的成本。

关键词 对等网络,超级节点,拓扑构造,自适应分级,有序构造
中图法分类号 TP393 **文献标识码** A

Hierarchical and Ordered P2P Super-peer Topology Construction

FENG Jin-xiao CHEN Gui-hai XIE Jun-yuan
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

Abstract Topology construction is one of the most essential problems in P2P network research. The current super-peer topology construction employs a fixed two-layer structure and an unordered approach based on the gossip-based paradigm, which not only restrains the system performance but also produces too many traffic loads and makes super-peer hotspot of the system. Meanwhile it brings about the higher cost and the security issue. The paper presented a hierarchical and ordered super-peer topology, called HOST, which established an adaptive hierarchy structure of super-peers according to the network scale and exploited an ordered algorithm to regulate peer joining and leaving. The simulation results and analysis show that HOST can effectively control the generation and load balancing of super-peers, and remarkably reduce the topology construction and repair cost.

Keywords P2P network, Super-peer, Topology construction, Adaptive hierarchy, Ordered construction

P2P 是下一代互联网架构的重要分支,有着广阔的应用前景和重要的研究价值。在 P2P 网络中,拓扑架构是基础,决定着 P2P 网络的路由和应用模型。目前最主要的 P2P 体系架构包括集中索引式、纯分布式非结构化、分布式结构化和超级节点结构 4 种类型。集中索引式结构[1]采用少量中心服务器为所有节点提供资源索引服务,查询效率较高、管理简单,但存在单点失效与可扩展差等问题。在纯分布式非结构化模型[2]中,所有节点处于完全对等的地位,不存在单点失效问题,但节点间采用的泛洪(Flooding)查询方式,产生过多的网络流量,可扩展性较差。分布式结构化模型[3,4]通过 DHT (Distributed Hash Table)机制实现资源到节点的映射,支持资源直接定位,具备较好的可扩展性,但由于在支持模糊查询方面的不足以及在高度动态的环境中存在路由维护代价过高的缺点,基于 DHT 机制的分布式结构化模型并不适合目前流行的 P2P 文件共享应用。相比之下,基于节点异构性的超级节点结构[5]通过让能力强的节点(超级节点)成为能力弱的节点(普通节点)代理,查询只在超级节点之间进行转发的方式,有效降低了参与泛洪的超级节点数量以及弱节点带来的性能瓶颈问题,同时多个超级节点的存在降低了单点失效风险,因而 P2P 超级节点结构是一种很有潜力的拓扑模型。如何设计一种高效安全的超级节点拓扑结构,已成为 P2P 网络模型中一个核心问题。

1 P2P 超级节点拓扑技术和存在的问题

在最初的 P2P 结构中,索引节点被认为是对等的,每个节点在系统中是 Servent(Server+Client)角色,既作为 Server 向其他节点提供资源,又作为 Client 向其他节点请求资源。但是通过对 P2P 网络的测量研究发现,P2P 网络中节点能力和在线时间是不对等的,系统的共享资源由少数能力强的节点提供,普通的 P2P 节点在线时间一般很短并且存在 Free-ride 现象。在这种情况下,把能力强的节点转变为网络拓扑核心,普通节点直接连接到这些超级节点,由超级节点代理其进行查询和路由转发,可大大减少系统消息成本。目前的超级节点拓扑采用类似于图 1 的两层结构。

到稿日期:2008-11-05 返修日期:2009-02-08 本文受国家自然科学基金(60573131,60721002),国家"九七三"重点基础研究发展规划项目基金(2006CB303000)资助。

冯劲潇(1976-),男,博士生,主要研究方向为 P2P 网络等,E-mail:fjx_mailbox@126.com;陈贵海(1963-),男,教授,博士生导师,主要研究方向为并行计算和无线网络等;谢俊元(1961-),男,教授,博士生导师,主要研究方向为人工智能、网络安全、网络计算等。

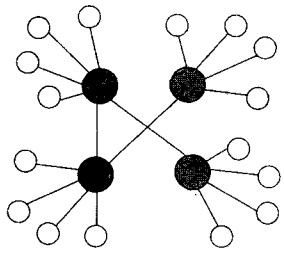


图 1　两层超级节点结构

在图 1 中，节点间的关系分为两种：一种是超级节点和叶节点之间的关系，主要包含资源索引和代理查询关系；另一种是超级节点和超级节点之间的关系，主要涉及到超级节点之间的路由组织以及负载均衡调整。

对代理关系来说，系统中的超级节点数是由网络规模和超级节点容量决定的。假定系统全部节点数为 N，超级节点数为 N_s，每个超级节点代理的普通节点数为 C_n（$C_{min} \leqslant C_n \leqslant C_{max}$），其中 C_{min} 和 C_{max} 分别为超级节点代理的最小子节点数和最大子节点数（$C_{min} \neq 0$，C_{max} 受超级节点硬件能力约束），则

$$N/C_{max} \leqslant N_s \leqslant N/C_{min} \tag{1}$$

超级节点之间的路由效率主要体现为查询经历的步数。设一个超级节点平均向 N_b（$N_b > 0$）个邻居超级节点转发消息，消息泛洪到网络中所有超级节点所需的跳数（Hops）为 H_{max}，则泛洪消息 M 的增长方式为 $\sum_{i=0}^{H_{max}} N_b^i$，设消息重复率 $R=(M-N_s)/M$，则

$$R=(\sum_{i=0}^{H_{max}} N_b^i - N_s)/\sum_{i=0}^{H_{max}} N_b^i \tag{2}$$

$$\sum_{i=0}^{H_{max}} N_b^i = N_s/(1-R) \tag{3}$$

$$\begin{aligned}\sum_{i=0}^{H_{max}} N_b^i &= 1+N_b+N_b^2+N_b^3+\cdots+N_b^{H_{max}} \\ &= N_b^{H_{max}} \times (\frac{1}{N_b^{H_{max}}}+\frac{1}{N_b^{H_{max}-1}}+\frac{1}{N_b^{H_{max}-2}}+\cdots+1) \\ &= N_b^{H_{max}} \times (1-\frac{1}{N_b^{H_{max}+1}})/(1-\frac{1}{N_b})\end{aligned} \tag{4}$$

当 H_{max} 较大时，$\frac{1}{N_b^{H_{max}+1}}$ 近似为 0，所以式(4)近似表示为

$$\sum_{i=0}^{H_{max}} N_b^i = N_b^{H_{max}}/(1-\frac{1}{N_b}) \tag{5}$$

$$N_b^{H_{max}} = (1-\frac{1}{N_b}) \times N_s/(1-R) \tag{6}$$

设 Log 是以 2 为底的对数，则由式(6)可得

$$H_{max} \times \mathrm{Log}N_b = \mathrm{Log}((1-\frac{1}{N_b}) \times N_s)/(1-R)) \tag{7}$$

$$H_{max} = \frac{\mathrm{Log}((1-\frac{1}{N_b}) \times N_s/(1-R))}{\mathrm{Log}N_b} \tag{8}$$

超级节点拓扑按超级节点层是否支持 DHT 机制分为结构化超级节点拓扑和非结构化超级节点拓扑。前一种类型超级节点结构在超级节点层支持 DHT 机制，叶节点间可以采用 DHT 机制。如 Garc′es-Erice[6] 等人提出一个通用的分层 DHT 模型，该模型将节点分成多个组，组内节点采用 DHT 方式进行管理，每个组选出几个超级节点，所有超级节点组成一个顶层覆盖网，进行路由转发。叶节点间也可不采用 DHT，而是直接连到超级节点，叶节点通过超级节点发布内容，如 ISP2P[7]。通过引入超级节点机制，这类结构改善了查询效率，但是仍然面临不适应拓扑动态性和无法有效解决模糊查询问题。我们已在另一篇研究中提出了基于分层象限空间的结构化超级节点拓扑，用于解决以上问题。

非结构化超级节点拓扑包括 Gnutella0.6[8]、KaZaa[9] 等。目前的非结构化超级节点拓扑结构采用与图 1 相似的两层结构。系统分为两层：上层由能力较强的节点组成，称为超级节点或 Hub 等；下层是普通节点，普通节点不参与系统的路由，查询通过超级节点进行。在拓扑构造上，节点主要采用基于谣言（Gossip-based）的方式来传递共享消息，如 Montresor[10] 等人提出 SG-1。在 SG-1 中，每个节点随机地与邻居节点交换信息，并根据相关的能力值信息进行超级节点角色转换。这种固定的两层结构和随机的拓扑构造方式给非结构化超级节点拓扑带来一些性能和安全性方面的不足，表现在：

(1)性能改进和负载均衡受到层次限制。超级节点结构在路由性能上优于非超级节点拓扑，得益于路由只在超级节点间传递。当网络规模持续增大时，两层结构下系统无法减少超级节点数量，因此随着网络规模持续增加，系统效率会不断下降。另一方面，为降低系统超级节点数量，超级节点会尽可能多地接纳叶节点，使自身处于过载状态，成为系统热点。

(2)拓扑构造效率低，成本高。在基于谣言方式的随机探测机制下，超级节点与普通节点之间的代理关系不断变换，导致较高的拓扑构造成本。当超级节点失效时，其子节点需要重新通过谣言方式在全局范围内查找新的代理超级节点，失效修复成本较高。

(3)拓扑无序性。节点的角色变换缺少相应的监控机制，节点可以自由决定是否成为超级节点，这使得一些在线时间短但硬件能力强的节点有可能成为超级节点。同时，一些恶意节点也可以方便地成为超级节点，影响拓扑的稳定性。拓扑缺少一种有序的控制机制。

为解决这些问题，本文提出一种分层有序的超级节点拓扑，称为 HOST。在 HOST 中，超级节点层次会随网络规模增加进行自适应调整，其调整模型可用下式表示：

$$N_s \rightarrow \frac{N}{C_n^L} \tag{9}$$

其中，N 为网络节点总量，C_n 为超级节点平均代理的子节点数，L 为超级节点分层数。将式(1)中的超级节点 N_s 进一步分级为参与路由的顶层超级节点 N_{st} 和不参与路由的内部超级节点 N_{si}，则在式(9)中，只要根据网络规模 N 适当地调整 C_n 和 L，就能使 N_{st} 数量控制在一定范围内，并能避免超级节点处于过载状态，从而使系统具有自适应分级调整特征。与两层的超级节点拓扑相比，HOST 中 N_{st} 和 N_{si} 之间存在一些管理成本，但是如果能进行两者之间的合理分工，比如由 N_{st} 处理路由消息，N_{si} 处理资源索引，那么 HOST 中超级节点自身处理成本的减少可以弥补增加的层次处理成本。另外在分层中，不采用基于谣言的方式，而是由上层超级节点从叶节点选择候选超级节点。在超级节点失效后，候选超级节点会接替其角色。这种构造方式使超级节点的产生处于有序控制下，并减少超级节点的失效修复成本。在目前百万节点的网络规模下，按超级节点容量为 50、负载率平均为 40%计算，均衡的三层 HOST 超级节点拓扑中理论上顶层超级节点数只有 2500。在这样的规模下，路由和查询性能是较高的。本文也以 L 不超过 2 进行说明。

2 系统模型

定义 1 Peer 节点定义为一个三元组 $P=(I,C,D)$，其中 I 为节点在系统范围内的唯一标识，C 表示节点容量，D 表示节点连接度。

定义 2 称 L_p 为节点 p 在拓扑结构中所处的层次，ML 为系统约定的最大层次，则 p 层次满足 $0 \leqslant L_p \leqslant ML$。如果 $L_p==0$，称节点 p 为普通节点(Ordinary Peer，OP)；如果 $(L_p==ML) \vee (L_p<ML \wedge (\neg \exists q, L_q>L_p))$，称 p 为顶层超级节点(Top-layer Super-peer，TSP)，否则称 p 为内部超级节点(Inner-layer Super-peer，ISP)。本文中 ISP 的 $L_p==1$。

定义 3 节点路由表定义为一个四元组 $R=(TR,PR,CR,CSR)$，其中 TR 指查询转发路由表，每个邻居 TSP 占用 TR 中的一个条目，所有 TSP 通过 TR 形成一个顶层覆盖网，邻居 TSP 之间要定期发送消息，进行 TR 更新；PR 是父节点路由表，子节点通过 PR 向父节点发出查询请求或者定期进行元数据更新；CR 是子节点路由表，包含子节点地址及能力值等相关信息，父节点通过 CR 对子节点按在线时间、硬件能力等进行排序，最优子节点作为候选超级节点，候选超级节点用来取代失效的父超级节点。父节点定期将候选超级节点的连接信息发送给子节点，并定期将 CR 发送给候选超级节点。CSR 是候选超级节点路由表，当父节点失效时，其子节点通过 CSR 连接候选超级节点。

定义 4 HOST 结构为满足下列规则的无向图 $G:\langle V,E\rangle$ (V 代表节点集，E 代表节点之间的连接)。

(1) $p,q \in V$，如果 $L_p<L_q \wedge \langle p,q\rangle \in E$，则 $L_q=L_p+1$。

(2) $p,q \in V$，如果 $L_p==L_q \wedge \langle p,q\rangle \in E$，则 $p \in TSP \wedge q \in TSP$。

一个简单的三层 HOST 结构如图 2 所示。

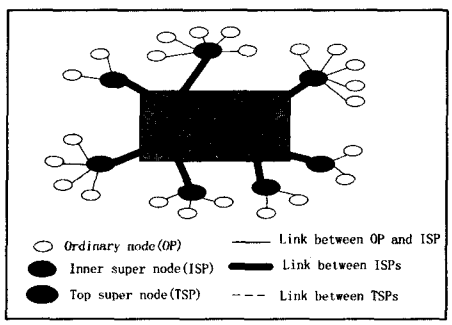


图 2 三层 HOST 结构

3 自适应分级调整算法

系统的自适应分级调整算法要实现两个目标：一是保持超级节点负载的相对均衡，二是自动控制参与路由的顶层超级节点数量。这分别通过分裂(Split)和调整(Adjust)算法来实现。

3.1 分裂算法

当一个超级节点代理的子节点数达到硬件能力限制后，SG-1 通过寻找更优的超级节点来转移自身的负载，HOST 则通过分裂来解决这个问题，即超级节点从自己的子节点中选出具有一定比例的在线时间且硬件能力强的子节点作为自己的子超级节点，其他子节点均匀分配给这些子超级节点。这样，既可以使系统中的超级节点负载保持均衡，又可以使超级节点的指定建立在可控的基础上，防止在线时间短和硬件能力弱的节点成为超级节点，同时可以降低恶意节点成为超级节点的概率。

假设 C_p 表示节点 p 的子节点数，S_{pi} 表示节点 p 的第 i 个子节点，i 为 0 表示第 1 个子节点，依次类推。Add(peer q，peer p)表示节点 p 将节点 q 加为子节点。Update(peer p)表示更新节点 p 状态，这些状态包括节点连接度、路由表等信息。设 α 表示超级节点的分裂比例，sn 表示父节点选择分裂的子超级节点数，s 代表位置变量，则分裂算法 Split(peer p)步骤如下：

(1)计算子超级节点数 sn，因为 p 的子节点是经过排序的，所以 p 的 0 到 $sn-1$ 范围内的子节点被选为子超级节点。

$sn=\lfloor C_p \times \alpha \rfloor$；

(2)如果 $L_p==1$，则从子节点中提拔 sn 个子超级节点，再将其他子节点按固定的间隔均匀分配给上述子超级节点。这种分配方式可使各个子超级节点中的子节点能力保持相对均衡。

```
i=0;
while(i<sn)
  s=i;s=s+sn;
  while(s<Cp)
    Add(Sps,Spi);Update(Sps);s=s+sn;
  endwhile;
  LSpi++;Update(Spi);i++;
endwhile
Lp++;Update(p);
```

(3)如果 $L_p==2$，则对 p 的每个子超级节点重复执行步骤(1)和步骤(2)，从叶节点中选择新一层超级节点，然后将节点 p 和 p 的子超级节点层次加 1 并更新其状态。

在 Split 算法中，第(1)步的时间复杂度为 O(1)，在第(2)步中，设 C_p 的值为 n，容易看出外循环执行 sn 次，内循环执行 n/sn 次，所以第(2)步时间复杂度为 $O(n)$，第(3)步的时间复杂度为 $O(n^2)$。因为 n 受到节点硬件能力约束，所以分裂算法产生的消息负载并不大。

3.2 调整算法

超级节点通过分裂可以将负载均匀分给子超级节点承担，但这种方法会带来两个问题：(1)由于节点所在的簇(超级节点及其子孙节点组成的节点集)通过提升层次，可以接纳任意多的新节点，使得顶层超级节点数量不能随网络规模的增加而相应增长。(2)当一些簇总是比其他簇接收到更多节点加入请求时，会出现簇与簇之间层次不平衡问题。为克服这两个问题，需要对超级节点最大层次 ML 进行限定。当超级节点 p 经过分裂层次超过 ML 时，需要对 p 进行调整，使网络中顶层超级节点数量及拓扑层次达到较好的均衡。

调整算法 Adjust(peer p)如下：

(1)将节点 p 层次减 1，并将其子节点路由表 CR_p 状态通知给全部子超级节点，然后将这些子超级节点从 p 的 CR_p 移动到顶层超级节点查询转发路由表 TR_p 中，使 p 与子节点的关系从父子关系变成邻居关系。

(2)构造节点 p 每个子超级节点的 TR 路由表，该 TR 路由表包括节点 p 和 p 的其他子超级节点，各个子超级节点将 p 从父节点路由表 PR 移动到 TR 中。

(3)节点 p 的原候选超级节点 S_{p0} 将一定比例子节点转

移给 p。假设 S_{p0} 代理的子节点数为 C，转移比例为 β，则转移的节点数为 $\min(C/2,\beta C_p)$。

在调整算法中，由于节点 p 经过调整已经没有子节点，因此让其分担一定比例的节点，有利于拓扑的负载均衡。同时因为节点 p 的容量 C_p 乘以 β 可能会超出 S_{p0} 的子节点数 C，所以取 $C/2$ 和 βC_p 中的最小值作为转移节点数。从第(2)步可以看出调整算法的时间复杂度为 $O(n)$。

4 拓扑自组织

HOST 中的自适应分级调整算法能控制拓扑的结构和层次，但是要使该算法能真正作用于拓扑构造，还需要将其与一套有序的节点加入和退出机制结合起来。

4.1 节点加入

当新节点加入 P2P 网络时，需要知道网络中一个已存在的节点，这可以通过建立知名注册服务器来实现。节点通过查询该注册服务器返回系统中的一个已知节点，也可以在客户端软件中预置一定数量的节点列表。软件启动时，向这些节点发出加入请求，按照反馈结果选择一个活动节点。这里假设节点通过某种机制知道网络中一个已存在的活动节点。

在 HOST 中，为了保证拓扑的有序性，新节点加入网络时先成为一个普通节点。随着在线时间的增加，能力强的节点有机会被父节点选为候选超级节点。因为已知的活动节点类型为普通节点、内部超级节点或者顶层超级节点 3 种情况之一，所以 HOST 需要根据其不同类型做相应处理。新节点 q 加入已知活动节点 p 的算法如下：

(1)如果 $L_p==0$，则 p 向 q 返回自己的父节点，q 向 p 的父节点请求加入。

(2)如果 $L_p==1$，p 检测自己是否有空闲能力。如果有，则接受 q 加入；否则 p 检查自己是否有父节点。如果有，则将自己的父节点信息返回给 q，q 向 p 的父节点请求加入；否则 p 调用分裂算法 Split(p)，提拔一定比例的子超级节点，然后返回最空闲的子超级节点信息给 q，q 向该节点请求加入。

(3)如果 $L_p==2$，则 p 检查最空闲的子超级节点是否有空闲能力。如果有，p 向 q 返回该节点信息，以便 q 请求加入；如果没有，则调用分裂算法 Split(p)，经过分裂，节点 p 的层次会超出最大层限制，所以调用调整算法 Adjust(p)，然后 p 返回最空闲的子超级节点信息给 q，q 向该节点请求加入。

(4)当新节点找到合适的超级节点加入时，父子节点分别建立相应的路由表，建立代理关系。

有序的加入算法保证了自适应分级调整算法在拓扑构造中的实现，该算法的时间复杂度是由分裂算法和调整算法决定的。在加入算法中，由于父节点总是动态地选择最空闲的子超级节点信息发送给请求加入的新节点，因此父节点能使自己的子超级节点维持比较好的负载均衡。

4.2 节点退出

在 P2P 网络中，节点具有比较强的动态性，因而节点退出行为对拓扑结构有重要影响。节点退出有多种原因，为简化起见，这里不再做进一步区分。为提高节点失效修复的效率，HOST 对不同层次的网络节点退出进行不同的处理：普通节点退出时处理比较简单，只需从父节点中删除即可，HOST 的有序加入算法保证了频繁加入和退出的节点总是成为普通节点，其退出行为对拓扑结构影响不大。当超级节点退出时，HOST 利用候选超级节点来代替退出的超级节点，这样既能维持拓扑的有序性，也可以将超级节点的失效影响限制在本簇节点集内。

设 B_p 表示节点 p 的候选超级节点，F_p 表示 p 的父节点，Remove(peer p,peer q)表示从 p 中删除节点 q 信息，IsFailed(peer p)判断 p 是否失效，真值为失效状态，Clean(peer p)表示清除节点 p 中所有信息，Add(q,p)与 Update(p)的含义与分裂算法相同，p 的失效修复算法 Repair(peer p, int flag)如下：

(1)如果 $L_p==0$，则直接从父节点 F_p 中删除 p。

(2)如果 $L_p==1$，则用 p 的候选超级节点 B_p 取代节点 p，B_p 将 p 的其余子节点添加为自己的子节点。因为 B_p 是 p 的第一个子节点，所以从 p 第 2 个子女，即 $i=1$ 开始添加。flag 是一个标志，用于标识父节点对子节点的不同处理方式。flag 为 0 时，表示正常修复，需要从父节点中删除失效的节点；flag 为 1 时，表示处理的节点并没有失效，而是一个要替代失效顶层超级节点的候选超级节点。该节点也有自己的子节点，因而在用该候选超级节点取代顶层超级节点之前，也需要对拓扑进行修复，即用自己的下一级候选超级节点来接受自己的其余子节点。

```
for(i=1;i<Cp;i++)
  Add(Spi,Bp);Update(Spi);
endfor
if(flag==0)then
  if(Fp! =null)then
    Remove(Fp,p);Add(Bp,Fp);Update(Fp);
  endif
else
    Clean(p);Add(Bp,p);Lp++;Update(p);
endif
LBp++;Update(Bp);
```

(3)如果 $L_p==2$，则首先判断 p 是否有子超级节点同时处于失效状态。如果是，则先对失效的子超级节点进行修复。然后在用候选超级节点取代顶层超级节点之前，先对该候选超级节点进行修复。修复结束后，该候选超级节点接受顶层超级节点的其他子节点作为自己的子节点，以取代失效的顶层超级节点角色。

```
for(i=0;i< Cp;i++)
  if IsFailed(Spi)then
    Repair(Spi,0); /* go step(2) */
  endif
endfor
Repair(Bp,1); /* go step(2) */
for(i=1;i<Cp;i++)
  Remove(Spi,p); Add(Spi,Bp);Update(Spi);
endfor
Update(Bp);
```

在修复算法中，第(1)步的时间复杂度为 $O(1)$，第(2)步时间复杂度为 $O(n)$。第(3)步中，第一个循环的步数取决于同时失效的子节点数，在最坏情况下其时间复杂度为 $O(n^2)$，Repair(B_p,1)的时间复杂度为 $O(n)$，第二个循环的时间复杂度为 $O(n)$，所以最坏情况下失效修复算法的时间复杂度为 $O(n^2)$，这里 n 指超级节点的子节点数。因为该值受制于节点

硬件能力约束，所以修复算法的开销并不大。

5 实验与分析

本节首先介绍模拟环境，然后从拓扑构建成本、负载均衡、失效修复成本3个方面对SG-1和HOST进行比较。

5.1 模拟环境

我们基于PeerSim[11]实现了HOST模拟程序，同时对SG-1模拟程序进行了相应扩展。例如在SG-1中增加了将节点容量值输出至文件功能，然后在HOST中读取该文件，这样能够使两个协议中的节点容量保持一致。我们用最大连接度，即可以与某个节点直接相连的最大节点数抽象地表示该节点容量。在仿真中，网络中节点的容量分布满足Power-Law规则，分布指数为1.8，节点最大容量设为100。HOST分裂算法中超级节点分裂比例α设为10%，调整算法中转移比例β设为30%。

5.2 拓扑构建成本

相邻节点之间的交互通过消息传递来实现。我们用传递的消息数来表示抽象成本，拓扑构造过程中主要涉及到3类成本：

(1)请求成本，节点向其他节点请求加入产生的消息数。

(2)响应成本，节点响应其他节点的请求产生的消息数。

(3)移动成本，叶节点在超级节点之间移动产生的消息数。

因为请求成本与响应成本是相对应的，差异仅在于目标节点是否为有效节点，所以为简化起见，只考虑请求成本与移动成本。

图3比较了SG-1和HOST的总请求成本与总移动成本情况。在总请求成本方面，HOST要略低于SG-1，并且当网络规模增加时，两者之间的差距有增大的趋势，这是因为SG-1中的节点在拓扑构造中始终通过谣言方式随机选择邻居节点。而在HOST中，节点除了在第一次加入时随机选择网络中的一个节点外，在以后的请求中都是有序选择邻居节点，因而请求成本要低于SG-1。在总移动成本方面，HOST要明显优于SG-1，总移动开销只是SG-1的10%左右。原因是SG-1中，超级节点不断寻找容量更优的节点，找到后会将自己的子节点转移给该节点，导致移动成本大幅增加。而在HOST中，当超级节点接受的子节点数达到自己的容量限制时，只需进行小范围的分裂即可，节点移动被限制在本簇范围内。

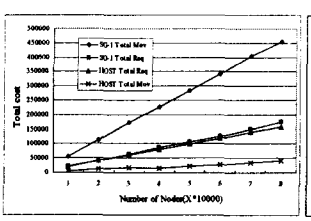


图3 总成本比较

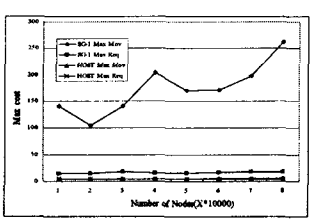


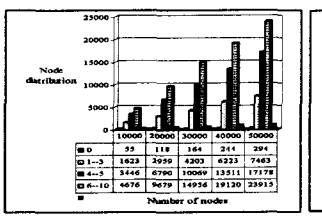
图4 单个节点最大成本比较

图4对拓扑构造中单个节点所产生的最大请求成本和最大移动成本进行了比较。图中数据显示出两个协议的最大请求成本在不同网络规模下都能保持均衡性。在SG-1中，该值保持在15～18之间；而在HOST中，单个节点最大请求成本值为4。在最大移动成本方面，SG-1中该值在不同网络规模下呈不规则上升趋势，而在HOST中该值始终保持4～5的均衡水平。从数值上看，在网络节点规模为8万时，SG-1的节点最大移动成本值达到262，而在HOST中该值仅为5，两者相差非常大。这些数据表明HOST比SG-1能更有效地控制单个节点的最大成本。

5.3 负载均衡性

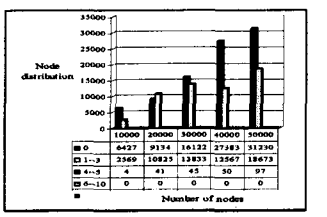
从两个方面来比较SG-1和HOST的负载均衡性：一是节点的成本负载，二是节点的连接负载。因为两个协议中节点的请求成本比较接近，所以这里只考虑移动成本负载情况，同时因为两个协议中普通节点都只连接一个超级节点，即普通节点的连接度均为1，所以只考虑超级节点连接负载。

图5和图6说明了SG-1和HOST成本负载在节点间的分布情况。从图5中可以看出，SG-1中节点的移动成本主要分布在4～10之间。以网络规模3万为例，节点的移动成本在上述区间的比例占到了83.42%。此外，有608个节点的移动成本达到10，占到2.03%。而在图6中，HOST的节点移动成本主要分布在0～3之间，以相同的网络规模为例，移动成本为0的节点占到了53.74%，移动成本在1～3之间的节点占到46.11%，这两项比例之和为99.85%，仅有45个节点成本在4～5之间，只占0.15%。这表明与SG-1相比，HOST中节点的移动成本负载不仅小，而且更加均衡，消除了SG-1中存在的热点问题。



	10000	20000	30000	40000	50000
0	55	118	164	244	294
1--3	1623	2959	4203	6223	7463
4--5	3446	6790	10069	13511	17178
6--10	6676	9679	14936	19120	23915

图5 SG-1成本分布



	10000	20000	30000	40000	50000
0	6427	9134	16122	27283	31290
1--3	3569	10825	13833	13567	18673
4--5	4	41	45	50	97
6--10	0	0	0	0	0

图6 HOST成本分布

图7比较了SG-1和HOST的超级节点连接负载率情况。超级节点连接负载率为超级节点代理的子节点数与超级节点容量之比。在HOST中，同时考虑了顶层超级节点连接负载(TSCL)率和内部超级节点连接负载(ISCL)率。从图8可以看出，SG-1的超级节点连接负载(SPCL)率在各种网络规模下都保持在99%以上；而在HOST中，顶层超级节点的连接负载率不超过7%，内部超级节点在25.49%和42.66%之间。这表明与SG-1相比，HOST的自适应分级调整方法能有效降低超级节点的连接负载率，防止热点形成。

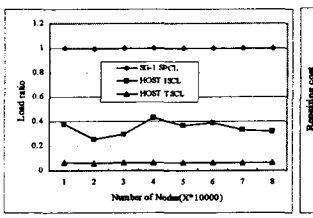


图7 超级节点连接负载率比较

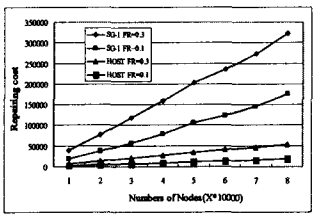


图8 拓扑修复成本比较

5.4 失效修复成本

由于P2P网络中节点的动态性特性，节点失效产生的修复成本是衡量拓扑性能的一个重要指标。在SG-1和HOST中，普通节点失效只需从父节点中进行删除即可，不需对拓扑进行修复，因此只考虑超级节点失效产生的修复成本。这里的修复成本是请求成本和移动成本之和。

图8比较了两个协议在超级节点失效比例(FR)分别为

(下转第175页)

节可以不做任何考虑。

3)构件复用性得到了增强。因为仅仅利用元组空间进行通讯,Agent 不存在常规构件那样复杂的依赖关系。依赖限制的减少增加了 Agent 的可复用场景。

4)基于分散式元组空间的协作系统开发效率得到提高。基于开发的软件体系结构图形化开发工具,可以快速将设计转换为可运行代码。

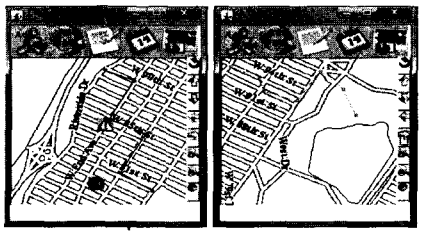


图 3 基于分散式元组空间的协作标定系统

结束语 为分散式协作系统找到一种方便、快捷、具有形式化语义支持的开发模式,其对于计算机支持的协同工作(CSCW)、群决策系统、移动自组织网应用系统等领域的发展十分关键。元组空间模型由于其固有特性,尤其适合作为此类开发模式的基础。本文提出了一种面向分散式元组空间的软件体系结构模型 DTSArch,用于支持此类系统的设计与实现。较目前已存在的软件体系结构模型,DTSArch 充分利用了分散式元组空间的优势,为分散式协作系统这个领域的开发实践提出了新的思路。

DTSArch 描述了系统的结构和行为。基于 DTSArch 的精确语义,开发人员能够快速建立实际协调系统的软件体系结构模型。本文还采用 EMF 和 GEF 框架对模型进行了实现,并建立了图形化编辑器以及相关的系统代码生成工具。利用它们设计者能够直观地描述或设计系统实例并生成相应的系统代码,对此类系统开发有较大实用价值。该模型和工具已经应用于移动协同平台中。

在后续的工作中,主要就以下两个问题进行进一步研究:1)丰富和扩展 DTSArch 的元语,为常见的协作行为提供更直接的支持;2)从动态体系结构的角度考虑,扩展 DTSArch 的描述能力,使其能够描述软件体系结构运行时的演化。

参 考 文 献

[1] Gelernter D. Generative communication in Linda [J]. ACM Transactions on Programming Languages and Systems, 1985, 7 (1): 80-112

[2] Freeman E, Hupfer S, Arnold K. JavaSpaces(TM) Principles, Patterns, and Practice[M]. Addison-Wesley Pub. Co., 1999

[3] Wyckoff P, et al. T spaces[J]. IBM Systems Journal, 1998, 37 (3): 454-474

[4] GigaSpaces. GigaSpaces enterprise application grid 4.1 documentation[S]. 2005

[5] Murphy A L, et al. LIME: A coordination model and middleware supporting mobility of hosts and agents[J]. ACM Transactions on Software Engineering and Methodology, 2006, 15(3): 279-328

[6] Mamei M, Zambonelli F. Spatial Computing: The TOTA Approach[C]// SELF-STAR 2004. LNCS 3460. Berlin Heidelberg: Springer-Verlag, 2005: 307-324

[7] Busi N, et al. PeerSpaces: data-driven coordination in peer-to-peer networks[C]// Proceedings of the 2003 ACM Symposium on Applied Computing. Melbourne, Florida: ACM Press, 2003: 380-386

[8] Sangiorgi D, Walker D. The π-calculus: a Theory of Mobile Processes[M]. Cambrige University Press, 2001

[9] EMF. Eclipse Modeling-EMF Page[EB/OL]. http://www.eclipse.org/modeling/emf/

[10] GMF. Eclipse Graphical Modeling Framework Page[EB/OL]. http://www.eclipse.org/modeling/gmf

(上接第 131 页)

10%和 30%时修复拓扑所产生的成本。首先从成本趋势来看,两个协议的修复成本都随超级节点失效比例及网络规模的增加而增大,但 HOST 的修复成本增长趋势要慢于 SG-1。其次从数值上看,HOST 修复成本要明显低于 SG-1。例如在网络规模为 8 万、失效比率为 30%时,SG-1 的修复成本为 322730,而在 HOST 中该值为 53230,只占 SG-1 修复成本的 16.49%。差异原因在于:在 SG-1 中,超级节点失效后,其子节点需要在全局范围内通过谣言方式重新寻找其他超级节点加入,导致较高的失效修复成本。而在 HOST 中,通过有序的修复算法,候选超级节点会接管原有失效超级节点的角色,并将超级节点失效影响控制在本簇范围内,使修复成本大幅降低。

结束语 本文给出一种分级有序的 P2P 超级节点拓扑构造(HOST)方案。首先通过分裂和调整机制对拓扑进行控制,使拓扑层次和超级节点负载随网络规模增大进行自适应调整。其次通过有序的节点加入、退出以及失效修复算法,减少了拓扑构造的随意性。模拟结果表明,HOST 在降低拓扑构造成本、保持超级节点负载均衡以及减少失效修复成本 3 个方面均优于 SG-1 超级节点拓扑构造方案。

参 考 文 献

[1] Napster[OL]. http://www.napster.com

[2] The Gnutella Protocol Specification v0.4 [EB/OL]. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[3] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for Internet applications [C]// Cruz R, Varghese G, eds. Proceeding of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications(SigComm). New York: ACM Press, 2001: 149-160

[4] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems[C]// Guerraoui R, ed. Proceedings of the IFIP/ACM Int'l Middleware Conf. London: Springer-Verlag, 2001: 329-350

[5] Fasttrack[OL]. http://en.wikipedia.org/wiki/FastTrack

[6] Garc'es-Erice L, Biersack E W, Felber P A, et al. Hierarchical peer-to-peer systems[C]// Kosch H, Böszörményi L, Hellwagner H, eds. Proceeding of ACM/IFIP Int'l Conference on Parallel and Distributed Computing(Euro-Par 2003). Berlin: Springer-Verlag, 2003: 643-657

[7] 夏启志,谢高岗,闵应骅,等. IS-P2P:一种基于索引的结构化 P2P 网络模型[J]. 计算机学报,2006,29(4):602-610

[8] Gnutella protocol spec. v. 0.6 [EB/OL]. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[9] Liang J, Kumar R, Ross K W. Understanding KaZaA[OL]. 2004. http://cis.poly.edu/~ross/papers/UnderstandingKaZaA.pdf

[10] Montresor A. A Robust Protocol for Building Superpeer Overlay Topologies[C]// Caronni G, Weiler N, Shahmehri N, eds. Proceedings of the 4th International Conference on Peer-to-Peer Computing(P2P'04). Washington: IEEE Press, 2004: 202-209

[11] PeerSim Simulator[EB/OL]. http://peersim.sourceforge.net/