P2P网络系统中节点自组织管理机制

杨文俊
(华中科技大学 控制科学与工程系,湖北 武汉 430074)

摘 要:为了优化对等网络(P2P Network)中节点间的数据传输性能,加强对等网络的系统稳定性,提出了一个基于对等网络的新型节点管理机制。按照P2P网络中节点的网络特性对节点进行分组、分组内节点按照传输性能进行分层。系统基于这种双层结构,利用根节点和组内主节点进行集中式控制和管理分组的创建,组成员的加入、离开,数据传播和失效等行为,能够较好地支持实时的多应用。通过与传统随机构建的网络结构进行仿真对比,证明构建这种结构化层状网络,可有效地减少节点间的网络距离,提高应用层多播的传输性能。

关键词:对等网络;节点分组;结构化应用层多播网络
中图分类号:TN915.9 **文献标识码**:A **文章编号**:1673-629X(2006)07-0057-04

A Novel Self-Organization Mechanism for Nodes Management in P2P Networks

YANG Wen-jun
(Huazhong University of Science & Technology, Wuhan 430074, China)

Abstract: In order to improve the performance of transmission and stability of system, a self-organization mechanism for nodes management in P2P network was proposed. Nodes are grouped according to its network attributes in the overlay network. Nodes in the same group are layered according to the performance of transmission. Its group management and member management were in centralized way. It is proved to be effective in shortening the net distance and improving the performance of the application layer multicast through this layered network.

Key words: peer-to-peer; node grouping; structural application layer multicast

0 引 言

决定P2P(Peer-to-Peer)网络传输性能和系统稳定性的关键因素是网络的拓扑结构和节点管理办法。当前的P2P网络有多种拓扑结构,如mesh,tree和有组织的混合结构等[1]。以tree为主的网络,如Yoid[2]和HMTP[3](见图1)等,采用层状的树形网络,结构的稳定性和失效恢复策略成为主要的难点。以mesh为主的网络,如Narada[4,5](见图2),节点管理采取随机组合的方式,其控制信息开销过大($O(N^2)$,N为网络中节点数),造成网络中数据传输性能不高、网络不易大规模扩展。

文中提出的节点分组、组内分层的节点自组织管理机制,将tree和mesh两种网络结构进行了结合和扩展,既降低了网络传输延时,提高了网络的服务质量,又使得网络的可扩展性大大提高。

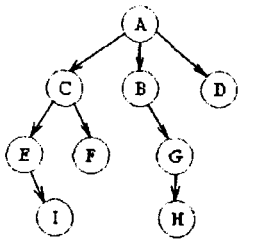

图1 以Tree为主的网络拓扑结构

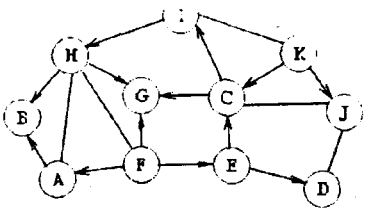

图2 以Mesh为主的网络拓扑结构

1 影响P2P网络性能的主要因素与对策

当前的P2P网络主要应用于大型文件、数据的共享与分发,因此提高网络中应用层多播的效率是优化网络性能的首要工作。应用层多播的效率首先体现在节点间的数据传输效率上,它往往取决于节点的路由开销和实际的网络环境。如果网络中的节点以随机的方式构建,那么在网络中相邻的两个节点可能在实际的网络环境中相距很远,导致路由开销过大,传输性能不理想,如图3所示。因

收稿日期:2005-09-29
作者简介:杨文俊(1981-),男,湖北武汉人,硕士研究生,研究方向为P2P网络、数据共享与应用。

此需要构建一个高效的对等网络,在具备完善的多播功能的同时,降低控制开销和额外的网络路由负载(如图 4 所示)。为了达到这个目标,可以将注意力放在以下两方面:

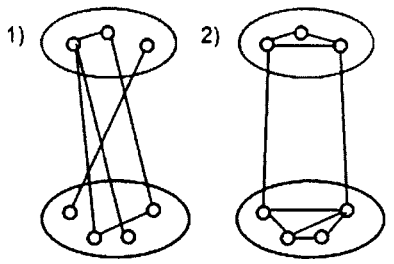

图 3 采取随机的方式选择邻节点　　图 4 根据网络特性合理选择邻节点

1)高效性:网络调度算法应该保证节点间的通讯简洁高效,其网络开销和直接的两点间开销相比不应过高。根据这一要求,需要将节点的实际网络位置作为构建对等网络的一个考量对象,使对等网络中的相邻节点在实际网络环境中也十分相近,以减少网络延时和通讯开销,同时增强传输质量。

2)可扩容性:当节点数量和数据传输量增加时,依然要保证对等网络的高效性。对等网络的性能由数据传输管理与节点位置管理共同决定。为了在节点数巨大的情况下达到这一目标,需要对控制信息进行分布式的管理,同时减少节点选择的随机性以减少互联节点间的实际网络距离。

通过以上分析,需要构建一种考虑到节点实际网络位置的对等网络。此网络应该具有较低的网络维护开销和较低的定位复杂度。下文阐述的节点自组织管理机制就是以此为目标而设计的一个 P2P 网络应用方案。

2 节点自组织管理机制的设计与实现

节点自组织管理机制主要包含两个方面,首先按照 P2P 网络中节点的网络特性对节点进行分组,其次分组内按照节点传输性能进行分层,由此使得 P2P 网络形成 tree 和 mesh 结构相结合的混合式拓扑结构,如图 5 所示。

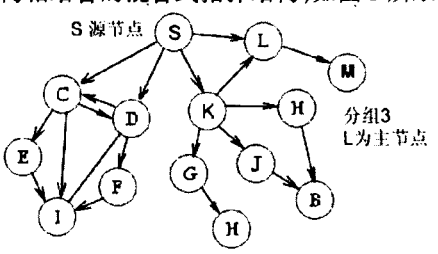

图 5 采用自组织机制后形成的混合式网络拓扑结构

为了阐明节点的自组织管理机制,首先需要说明几个主要的概念:

1)网络距离。两个节点间的网络距离可以由网络延时、数据往返时间或两者间的最小带宽等属性来衡量。当然,也可以是以上这些属性的综合考量。

2)分组。将网络属性相类似的节点进行归类,即分组。同一类型的节点间网络距离相当,传输延时小,数据传输速度快,对外属性也趋于一致。分组的好处在于极大的优化节点间的传输性能,并将节点的控制从服务器下放到分组,采取分布式的方法让节点在对等网络中自主去寻找合适的位置。

3)邻节点。在对等网络中已经建立连接的两个节点,可以称之为对等网络中的邻节点。

4)邻分组。每个组可以和其他若干个组进行信息交换,这些组彼此称之为邻分组。这些邻分组在实际的网络中也应该是相邻的,彼此间网络距离较近。

5)主节点。分组内网络性能好,输出带宽大,负责维护分组信息的节点。

节点随机选取传输对象会造成大量连接都建立在网络距离很远的节点间,从而使得平均邻节点距离变长。一个理想的对等网络结构是,大多数连接都建立在分组内部,使得节点获取较高质量的传输对象,分组间仅建立很少的连接,从而构建一个分为两层的网络架构,上层包含各分组,底层包含分组中的节点。分组中的节点彼此共享数据信息,分组间主要共享控制信息。

分组算法主要包括分组的创立、成员的添加与分组的维护等方面。

2.1 分组判断标准

每一个分组包含一个彼此网络距离相近的节点队列。对于实际网络存在的任一位置 P,如果节点 A 和 P 间的距离与节点 B 与 P 间的距离相同,就认为 A、B 两节点在同一个分组中。

1)组内判断:通过与 A 组中的若干节点进行距离检测,如果与它们的平均距离与组 A 内所有成员的平均距离相当,则节点 P 属于分组 A。

2)组间判断:通过其他组进行距离检测,如果节点 P 和分组 A 的邻分组 B 的距离,与组 A 和组 B 的距离相当,则节点 P 属于分组 A。

2.2 分组的创立

当新加入的节点无法在网络中找到属于自己的分组时应自主创立一个新的分组。这种情况主要发生在对等网络建立的初期,此时它会向主机提出创建自己的分组。在分组建立起后,形成了一个以 tree 结构为主体的对等网络结构。

新创建的分组中节点数往往较少,为了保证服务质量,可以利用邻分组中的成员进行数据传输,直到本分组中的节点数已经满足了数据传输的要求。

当然,当对等网络中的节点数和分组数已经达到一定数量后,可以限制新分组的创立,此时新节点则必须加入距离最近的分组。

2.3 节点的加入

一个新的节点在加入对等网络时,首先需要判别它应属于哪个分组或是否应该主动建立新的分组。

新节点首先和主服务器连接,获取一个分组信息列

表,此列表中包含整个对等网络中部分分组的主节点信息。新节点的加入可有两种途径进行选择分组:

1)按照分组列表中的顺序,逐一判断与该分组的距离,寻找最接近的分组加入。选择分组列表的第一个分组,首先和主节点通讯,获取该分组部分成员节点的基本信息,然后和这些成员节点进行通讯并获取与这些节点的平均网络距离D0,再通过比较D0和该组内的平均距离判断是否属于该组。如果满足分组判断标准则加入该组,否则记录下此D0并选择下个分组重复此过程获取平均距离D1。如果仍不满足分组判断准则,则比较D0与D1,将较小者所对应的分组作为候选分组。当所有分组均遍历一次或在规定时间内无法找寻到符合分组判断标准的分组时,加入到距离最短的分组。

2)采取分组传递的方式,通过分组的邻分组寻找最近的分组加入。选择分组列表的第一个分组A,与该分组的主节点a进行通讯,判断其与a间的距离作为该节点与组A的距离,并从a获取组A的邻分组信息列表G。节点从G中顺序选取邻分组,并计算与这些邻分组间的距离,不妨设其与邻分组B的距离最短为D0min。如果在这一过程中满足了分组判断标准,则此节点属于A,否则继续与分组B重复这一过程。节点计算与分组B的所有邻分组间的距离,设其与邻分组C的距离最短为D1min。同样,如果在这一过程中满足了分组判断标准,则此节点属于分组B。否则,将D0min与D1min相比较,将距离较近的分组作为候选分组,并与C分组的所有邻分组进行比较,如此反复。当所有分组均遍历一次或在规定时间内无法找寻到符合分组判断标准的分组时,加入到距离最短的分组。节点的加入工作主要由节点自主完成,主服务器仅需提供必需的数据信息,减少了服务器的压力。

2.4 组内节点的分层与数据共享

分组内部、各节点间的互联拓扑结构以mesh为主,以便加强节点间的互联性从而实现应用层多播。对一个采用应用层多播的对等网络而言,数据分发速度是衡量网络整体性能的关键指标。由于节点所处网络环境不同,其上下行带宽各异,单个节点的传输性能也在动态变化。为了提高数据分发能力,自动适应不断变化的网络环境,组内节点采取动态分层的方式进行数据传输与信息管理。组内分层的目的是让性能优异的节点处于组内的上层,优先获取数据流,以便数据流能通过这些高输出带宽的节点尽快传播给组内的各成员。对每一个分组而言,这些性能优异的节点组成了主节点列表,由这些主节点负责分组内数据、信息的分发、维护与管理。主节点列表中的第一个节点称为当前负责节点,它需要监控分组内其他节点的运行情况,并依此定时更新主节点列表并发送给组内所有节点。主节点列表可以依据组内各节点的实际情况进行更新,例如可以按照节点的输出带宽、传输速度、是否为静态IP等条件进行有序排列,从而达到动态调整的目的。当然,主节点应该定时向所有节点声明其有效性,如果组内节点在规定的时间间隔内未收到主节点的声明信息,可以认为其已经失效,此时可将主节点列表中的第二个节点作为主节点。当第二个也失效时,再选取第三个作为主节点,依此类推。一般说来,当主节点列表较大时,全部失效的可能性不大,如果组成员节点发现所有主节点列表内的节点全部失效可以主动向服务器进行申请,成为新的主节点对该组进行管理。在主节点的帮助下,组内各成员可以获取到组内所有节点的信息。当有节点进入或退出该组时,也可以从主节点获知该信息。

2.5 组间信息共享

在此对等网络中,每个分组是在不断变化的,分组间的相邻关系也是在动态调整中。每个分组创建时,其创建节点(第一个主节点)会根据与其他分组的距离选择部分分组作为其邻分组。邻分组间网络距离较近,但未达到分组判断标准,它们间的信息主要共享分组的主要信息。分组间的信息共享主要由主节点来完成。每个分组的主节点应该定时和邻分组的主节点交换信息,主要包括该组的主节点列表、组成员数等基本元素,主节点再将这些信息转发给分组内的其他节点。当一个分组内的节点无法在本分组内获取到足够的有效数据时,例如分组创建初期节点数较少时,可以从其他分组的节点进行通讯获取所需数据。

3 仿真及结论分析

为了验证此自组织管理机制的有效性,使用OMNeT++下的Swarm库进行相关的仿真分析。构造了两种网络拓扑结构来验证该自组织模型。首先,构造了以mesh为主体的网络拓扑结构,此结构中,所有的传输节点均为随机选择,未添加任何的优化设置。然后,根据文中提出的组织模型,构造了混合式的分组结构。在仿真的过程中,以节点间的网络距离(此处使用的是节点间的数据往返时间)作为考察对象,结果如图6所示。

在随机选择传输节点机制下,由于缺乏必要的优化,造成节点的网络距离过大,传输性能并不理想。而使用文中提出的节点自组织机制后,由于节点在传输前进行了优化选择,使得网络距离大幅降低,当网络规模逐渐增大时,网络距离逐渐趋于一定值。

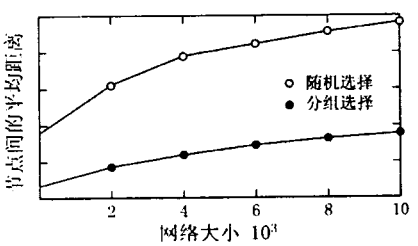

图6　不同节点选择机制下的网络距离比较

4 结束语

文中提出了一个基于对等网络应用层组播技术的节

点管理机制系统。它按照对等网络中节点的网络特性对节点进行分组,分组内节点按照传输性能进行分层,通过赋予网络自组织、自适应的功能来增强组播系统的传输性能。在一个大的对等网络中,节点根据分组标准自主形成中小规模的组播群,数据传输主要集中在组内进行。组内管理采用动态集中式控制,主节点掌握了组播组内成员的全局信息,节点通过彼此交换信息动态调整组内位置,使得分组内的数据传输性能最优化,从而加强了整个网络的性能。另一方面,由于采取了分组的机制,服务器仅需维护分组信息即可而不用管理所有节点,节点的加入、管理和数据传输等完全由节点间 P2P 完成,极大地减轻了服务器的负担,提高了可服务的节点数目。

采用这种节点分组、组内分层的节点自组织管理机制,能够调整各节点在对等网络中的位置,满足异构网络中各节点所处的不同网络环境。通过组内动态调整,使得输出带宽大的节点处于分组的顶端位置,加快数据的分发从而提高多播的效率以减少数据延时。

采取这种上层以 tree 与下层以 mesh 为主体的混合结构,极大地增强了网络的鲁棒性。当网络条件变化时,整个网络会随之动态调整。如新节点的退出或失效不会导致转发路径的断裂,转发路径具有自我修复、自我最优化的能力。因此,这种节点管理机制可以应用于大型文件的共享、流媒体等多种场合。

在实际的应用中,面对复杂的网络环境,如何较为准确地判断节点间的网络距离,进一步强化分组算法和组内管理算法,还有很多工作要做,这是在未来的工作中重点研究的内容。

参考文献:

[1] Banerjee S, Bhattacharjee B. A Comparative Study of Application Layer Multicast Protocols[Z]. US: University of Maryland, 2002.

[2] Francis F. Yoid: Extending the Multicast Internet Architecture, White paper[EB/OL]. http://www.aciri.org/yoid/.. 1999.

[3] Zhang B, Jamin S, Zhang L. Host multicast: A framework for delivering multicast to end users[A]. Proceedings of the IEEE INFOCOM 2002. Vol. 3[C]. New York: Institute of Electrical and Electronics Engineers Inc, 2002. 1366 - 1375.

[4] Chu Y H, Rao S G, Seshan S, et al. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture[A]. Proceedings of SIGCOMM 2001[C]. San Diego, CA: [s.n.], 2001.

[5] Chu Y H, Rao S G, Zhang H. A Case for End System Multicast[A]. Proceedings of ACM SIGMETRICS'00[C]. Santa Clara, CA: [s.n.], 2000. 1 - 12.

(上接第 56 页)

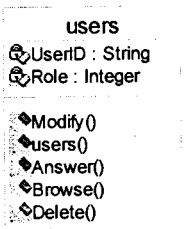

图 3 控制对象的类图

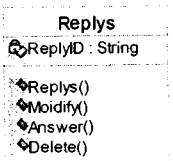

图 4 实体对象的类图

1.4 ASP.NET 下 Web 数据存储层架构

数据存储层的主要功能是把数据访问层的数据处理功能转换为具体的数据库或文件操作。从案例时序图中,可以确定实体对象以及实体类型的属性。实体类中的每个属性可以与数据库中的字段相对应,即每个实体类可以与数据库中的一张表对应。因此,在 ASP.NET 的 Web 应用系统中,可以通过 System.Data.OLEDB 命名空间或 System.Data.SQLClient[4,5] 命名空间中的数据库访问和控制类型构造与具体数据库相适应的类。在数据存储层对实体对象的属性和服务的访问中,可以通过这个类完成对数据库的访问和控制。

2 结 论

采用四层架构的思想,并运用 UML 的对象特点进行类的划分的架构设计,使得代码编写逻辑清晰,易于管理和维护,并且具有很好的代码的可重用性、适用性、易维护性和可移植性。四层架构的编写完全可以由四组人员同时进行,这样代码的维护管理上就更加清晰,并且可以缩短开发周期。但是同时进行的四层代码编写,需要有良好的前期的需求分析的支持。只有完备的需求分析(例如:使程序设计人员都清楚项目所包含或者项目中需要的类名和功能名称,当项目统一使用的时候,就不会因为名称不统一导致引用错误等问题),才可能真正实现四层架构的并行设计。

参考文献:

[1] 飞思科技产品研发中心. ASP.NET 应用开发指南[M]. 北京:电子工业出版社,2002.

[2] Boggs W, Boggs M. UML 与 Rational Rose 2002 从入门到精通[M]. 邱仲潘 等译. 北京:电子工业出版社,2002.

[3] 吴 建,郑 潮,汪 杰. UML 基础与 Rose 建模案例[M]. 北京:人民邮电出版社,2004.

[4] Kauffman J, Matsil B. ASP.NET 数据库入门经典[M]. 张哲峰,黄翔宇译. 北京:清华大学出版社,2003.

[5] Esposito D. 构建 Web 解决方案—应用 ASP.NET 和 ADO.NET[M]. 梁 超 译. 北京:清华大学出版社,2002.