第32卷 第19期 计 算 机 工 程 2006年10月
Vol.32 № 19 Computer Engineering October 2006
·多媒体技术及应用· 文章编号：1000—3428(2006)19—0227—03 文献标识码：A 中图分类号：TP37

基于ZIGZAG的流媒体直播系统的研究和实现

张 朕，应吉康
（华东师范大学计算中心，上海 200062）

摘 要：分析了现有P2P流媒体系统存在的问题，研究了ZIGZAG系统控制拓扑和数据传输拓扑两方面的主要技术，设计并实现了一个单源的基于ZIGZAG模型的P2P流媒体在线直播系统P2P-Streaming。系统通过局域网运行测试，分析表明P2P-Streaming有效地降低了节点开销，具有很好的扩展性。

关键词：应用层组播；P2P；流媒体

Research and Realization of Media Streaming Live System Based on ZIGZAG

ZHANG Zhen, YING Jikang
(Computer Center, East China Normal University, Shanghai 200062)

【Abstract】 This paper analyzes the problems of existing P2P media streaming systems and makes research on the control topology technology and data topology technology of ZIGZAG. Hereby an P2P live media streaming system——P2P-Streaming is designed. Moreover, it implements the P2P-Streaming system and runs it at the LAN. The analysis and test shows that P2P-Streaming reduces the overhead of peers efficiently, and has a good scalability.

【Key words】 Application layer multicast; P2P; Media streaming

1 概述

P2P计算已经被关注了很长一段时间，大量的基于P2P的文件共享系统被开发了出来，例如Napster、Chord、Pastry、Gnutella、eMule、BitTorrent等。本文研究的是P2P技术在流媒体直播领域的应用。当前国外一些大学和研究机构都提出了P2P网络的应用层多播方案，它们各有特色，不论是何种应用层多播方案，核心问题都是如何建立多播的分发树。分发树的衡量标准主要有两个指标[1]：(1)强度(Stress): 在一条物理链路中发送相同数据包的数量；(2)伸展度(Stretch): 在覆盖网分发拓扑中从源到成员的延迟与利用单播直接传输的延迟的比例。

本文关注的是 Internet 上占用大量带宽的直播媒体流从单个的流媒体源传送到大量的接收者时所面临的问题。目前，常见的针对单个流媒体源的 P2P 流媒体分发架构，其分发树通常以流媒体源为根节点，然后以所有的接收者为子节点。接收者的一个子集直接从根节点获取流媒体数据，其他的节点从它们的父节点获取流媒体数据。此种模式宣称可以解决大规模流媒体系统所面临的一些问题：(1)流媒体源的网络带宽瓶颈；(2)部署额外服务器的开销，这是内容分发网络不可避免的问题；(3)目前 IP 组播还无法广泛的应用。

但是，基于P2P的流媒体分发系统自身也面临很多问题。设计一个高效的P2P流媒体模式必须要考虑一下几点[2]：

(1)流媒体源到接收者的端到端延迟可能会过大。这是因为流媒体数据可能不得不经过许多的中间节点才能到达最终的接收节点。为了缩短延迟，分发树的高度应该保持很小并且新节点的加入过程要迅速。不过，由于分发树节点出现瓶颈的影响，端到端延迟可能还是很长。最容易出现节点瓶颈的分发树结构是以流媒体源为中心的星型结构。如果分发树结构是链状的，则可以最大程度地减少节点瓶颈。然而，在这种模式下叶子节点会有很长的延迟。因此，除了要控制分发树的高度之外，还十分需要控制节点的度。

(2)接收节点的行为是不可预见的。它们可以不顾自己的子节点而任意地加入或离开。为了防止流媒体服务被中断，必须提供一个健壮的恢复机制。

(3)为了保持节点间的连通性和增强 P2P 网络的效率，接收节会保存一些本地的数据结构并且和其他节点交换状态信息。这样就会存在控制开销。为了避免浪费额外的网络资源和占用每个接收节点的有限资源，节点的控制开销必须要小。对于一个拥有大量节点的系统来说，这点尤为重要。

为了解决这些问题，本文设计了一种新型的单源的流媒体在线直播系统P2P-Streaming。该系统基于ZIGZAG技术[2]，利用层次簇思想来构建组播树，可以有效地解决上述问题。

ZIGZAG 是由 Duc A. Tran 博士为首的研究组提出的基于 P2P 的流媒体分发体系模型。本文在分析研究了 ZIGZAG 模型的控制拓扑和数据传输拓扑两方面主要技术的基础上，利用 XML 技术和 Java 平台，设计实现了流媒体在线直播系统 P2P-Streaming。

2 基于 ZIGZAG 的系统原型

在 P2P-Streaming 系统中，包含两个重要部分：管理结构揭示了节点间的逻辑关系，而组播树说明了节点间的物理关系(例如，节点间的连接关系)。

2.1 管理结构

管理结构是用来维护系统中当前所拥有的节点，如图 1 所示。节点以层次簇的方式组织，其递归定义如下(H 表示层

作者简介：张 朕(1980－)，男，硕士生，主研方向：现代信息系统及其开发技术；应吉康，研究员
收稿日期：2005-12-24 **E-mail：**zhenzhang@citiz.net

数，k 是个常数，且 k > 3)：

(1)第 0 层包含所有节点。

(2)第 j 层(j < H - 1)的节点被划分为多个簇，每个簇的大小为[k, 3k]。第 H – 1 层只有一个簇，大小为[2, 3k]。

(3)第 j 层(j < H)每个簇中选出一个节点作为簇首。如果 j < H – 1，簇首节点变成第 j + 1 层的成员。任何包含服务器(流媒体源)节点 S 的簇，其簇首都是节点 S。

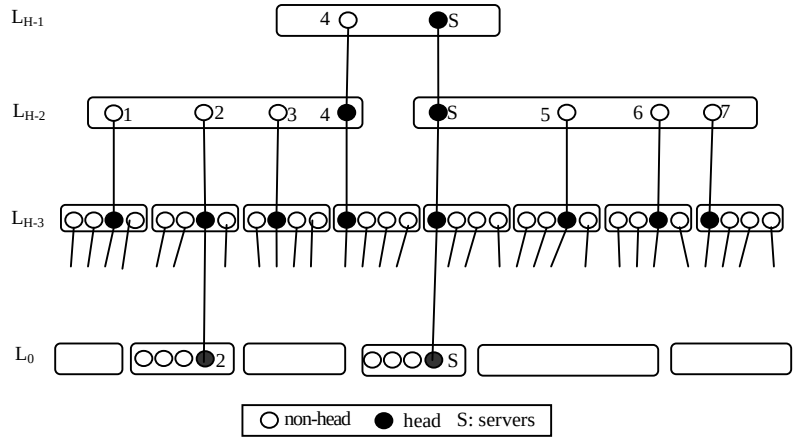

图 1 节点的管理结构

针对管理结构，定义如下术语：

(1)下属(Subordinate)：一个簇中的非簇首节点从属于一个节点 X，这些节点称为 X 的“下属”。

(2)外簇首(Foreign Head)：节点 X 的在第 j 层(j > 0)的簇友(clustermate)称为节点 X 第(j - 1)层下属的“外簇首”。其中，簇友必须是非簇首节点或者服务器节点。

(3)外下属(Foreign Subordinate)：节点 X 的第 j – 1 层下属称为节点 X 的第 j 层簇友的“外下属”。

(4)外簇(Foreign Cluster)：节点 X 所在的第(j - 1)层簇称为节点 X 第 j 层簇友的“外簇”。

2.2 组播树

第 1 个提出层次簇思想的是NICE组播树构建协议[3]，其缺点是层次越高的节点其负载越重，当系统规模很大时，可能成为系统瓶颈。与NICE不同，P2P-Streaming的管理结构并不意味着数据传输拓扑。例如，接下来很快就会发现第j层(j < H - 1)一个簇的簇首节点并不会转发数据给其下属。P2P-Streaming的组播树构建规则如图 2 所示。

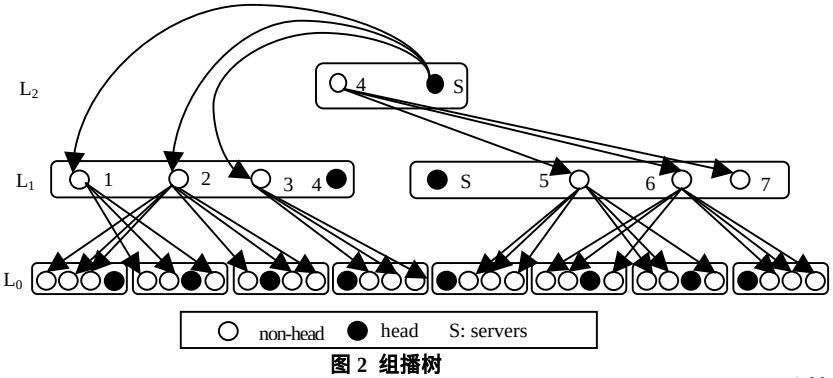

图 2 组播树

(1)一个节点，如果不是在其最高层，就不能连上或被连上任何其他节点。比如，在第 1 层的节点 4 没有任何外出和进入的连接。

(2)一个节点，如果在其最高层，就能且仅能连上其外下属。比如，第 2 层的节点 4 只能连接到第 1 层的节点 5、6 和 7，这些节点都是节点 4 的外下属。唯一的例外是服务器节点，在最高层，服务器节点连接它的所有下属。

(3)在第 j 层(j < H - 1)，因为一个簇中的非簇首节点不能从其簇首节点获得数据，它们只能从其它地方获得数据。在 P2P-Streaming，它们直接从其外簇首获得数据。比如，节点 1 在第 0 层的簇中的非簇首节点有一个来自它们外簇首(节点 2)的连接；节点 1、2 和 3 有一个来自它们外簇首(服务器节点 S)的连接。

2.3 控制协议

为维护组播树和管理结构中节点的位置和连接信息，一个第j层的簇中的节点X周期性地与其第j层的簇友、它在组播树中的孩子节点(这里的孩子节点、父亲节点与通常意义上的树的定义一样)和父亲节点通信。与节点交换信息的其他节点的数量等于节点X的度。如果与节点X交换信息的是一个簇首节点，则节点X还要发送给它一个列表L = {[X_1, d_1], [X_2, d_2],...}，这里，[X_i, d_i]表示节点 X_i 当前正在向其外簇中的 d_i 个节点转发流媒体数据。例如，在图 2 中，第 1 层中的节点 5 需要发送一个列表{[S, 3], [6, 3]}给服务器节点S。如果与节点X交换信息的是其父亲节点，X发送的数据则是：

(1)Boolean Reachable(X)：当且仅当在组播树中存在一条从节点 X 到一个第 0 层节点的路径时，Reachable(X)为 true。例如，在图 2 中，Reachable(7) = true，Reachable(4) = false。

(2)Boolean Addable(X)：当且仅当在组播树中存在一条从节点 X 到第 0 层的一个其所在簇的大小在[k, 3k - 1]之中的节点的路径时，Addable(X)为 true。

节点 X 的 Reachable 和 Addable 值根据节点 X 的孩子节点发送给它的交换信息而更新。例如，如果所有孩子节点都发送“Reachable = false”给节点 X，那么 Reachable(X)为 false；只要有一个节点 X 的孩子节点发来“Addable = true”，那么 Addable(X)为 true。

3 系统构造

3.1 结点加入

随着新节点的加入，组播树也随之变大，但是增大后的新组播树不能违反 2.2 节针对组播树的规则。下面将介绍节点加入算法。

一个新节点 P 向服务器提交一个加入请求，如果管理结构当前仅仅只有一层，则节点 P 直接和服务器节点建立连接。否则，节点 P 的加入请求就会沿着组播树一直向下转发，直到找到一个合适的节点进行加入操作。加入算法中需要用到两个函数，D(Y)表示从节点 Y 处获得的当前从节点 Y 到服务器节点 S 的端到端延迟；d(Y, P)表示当前节点 Y 和 P 通信过程中获得的从节点 Y 到节点 P 的延迟。

下面的节点加入算法描述了节点 X 得到一个加入请求后的操作过程：

```
Join( X, P) {
    If X is a leaf
  Add P to the only cluster of X
  Make P a new child of the parent of X
Else
  If Addable(X)
    Select a child Y :
      Addable(Y ) and D(Y )+d(Y , P) is min
    Forward the join request to Y
  Else
```

```
    Select a child Y:
      Reachable(Y ) and D(Y )+d(Y , P) is min
    Forward the join request to Y
}
```

该算法的目的是将节点 P 加入到第 0 层的簇 C 中，并且强制节点 P 从簇 C 的非簇首节点的父亲节点接收流媒体数据。为了防止簇的大小越界，簇 C 的大小应该保证在[k, 3k]之中。

对于叶子节点X，节点加入算法在第 4 步终止，节点X会告诉P簇中的其他成员，接着P会按照 2.3 节的控制协议运作。如果增加新成员后簇的大小仍然在[k, 3k]之中，就没有额外的操作。否则就要进行分裂操作以使簇的大小控制在[k, 3k]之中。为了减少分裂操作的开销，对在ZIGZAG分裂算法的基础上做了些改进，并不是在一个簇的大小变为 3k + 1 后立即执行分裂操作，而是周期性地进行检查。假设要分裂第j层(j∈[1, H - 2])中的一个簇，该簇的簇首为X，其他的非簇首节点为X_1, ..., X_n。此时这些非簇首节点从节点X′获取流媒体数据，而节点X从节点X″获取数据。设x_{il}为既是节点X_i的叶子节点同时又是节点X_l在第j－1 层下属的节点的数量。很明显，对所有i而言x_{il}= 0。节点分裂的步骤如下：

(1)将$\{x,x_1,\ldots,x_n\}$分为集合U和V，且满足如下条件：首先|U|，|V|∈[k, 3k]；其次应使

$$\sum_{X_l\in U, X_L\in V}(x_{il}+x_{li})$$

最小，此条件是为了有效地减少由于簇分裂而导致需要重新连接的节点数量。最后，$X\in U$。

(2)每个节点$X_i\in U$，每个节点$X_l\in V$，满足$x_{il}>0$，移掉所有从节点X_i到节点X_l第j－1 层下属的连接，并且从集合V中随机选择一个除节点X_l外的节点作为X_l的孩子节点的新的父亲节点。相反地，每个节点$X_i\in V$和每个节点$X_l\in U$ Q，且满足x_{il}>0，除了节点X不能作为父亲节点之外，其余步骤相同。

(3)为簇 V 选择一个新的簇首节点 Y。节点 Y 是 V 中拥有最小度的节点，这是因为希望分裂操作影响最少的节点。而且，节点 Y 成为第 j + 1 层中拥有节点 X 的簇的其中一个成员。同时，节点 Y 的孩子节点将不能再从节点 Y 获取流媒体数据。从簇 V 中再另外选择一个非簇首节点 Z 作为它们新的父亲节点。由于节点 Y 所处的最高层从 j 变成了 j + 1，因此移出当前从 X′到 Y 的连接，添加一个从 X″到 Y 的连接。此时的节点 Y 没有任何孩子节点，但并没有违反组播树的规则。

一件可能发生的事是第j＋1层的一个簇由于加入了节点 Y 而溢出，此时只能等待下一次周期性的分裂算法调用。分裂算法在每个簇首节点上运行，结果会发送给所有相关节点。

3.2 节点离开

考虑节点 X 的退出或者失效。根据 2.3 节的控制协议，节点 X 的控制节点、节点 X 的下属节点(如果有的话)和所有节点 X 的孩子节点(如果有的话)将会受到影响。X 的父亲节点需要删除到 X 的连接。如果 X 所在的最高层是第 0 层，则没有后续操作。

如果节点 X 所在的最高层为 j(j > 0)，失败恢复过程就会执行。对于第 j – 1 层的那些其非簇首节点是节点 X 的孩子节点的簇，簇首节点 Y 负责为它们找一个新的父亲节点。Y 选择了一个第 j 层的非簇首节点的簇友 Z，Z 具有最小的度，并通知其向 Y 的在第 j－1 层下属转发流媒体数据。

此外，因为节点 X 曾是 j 个簇的簇首(第 0 层，第 1 层，…，第 j－1 层)，这些簇都需要有一个新的簇首。任选一个 X 在第 0 层的下属 X′。X′将会取代 X 成为 j 个簇的簇首。X′将会获得一个来自 X 的父亲节点的连接。

大量节点离开的结果是一个簇的大小小于规定的最小值。这种情况下，将会在同一层进行合并操作。考虑簇 U 是第 j 层的一个大小不足的簇，它准备与同一层的另外一个簇 V 合并。选择簇 V 的最简单的方法就是查找一个第 j 层中成员最少的簇。合并操作的过程如下：

(1)新簇 U + V 的簇首节点在簇 U 的簇首 X 和簇 V 的簇首 Y 中选择。如果 Y(或者 X)在下一层中是 X(或者 Y)的簇首，Y(或者 X)就是新的簇首。如果 X 成为了新簇首，Y 就不会再出现在第 j + 1 层。

(2)从第 j + 1 层中选择一个 X(或者 Y)的非簇首簇友作为簇 U + V 的非簇首节点的新的父亲节点。这个新的父亲节点当前应该有最小的度。

(3)如果 Y 目前的孩子节点在 U 中，或者 X 的在 V 中，则合并操作在第(2)步就结束了。否则，有两种可能性会发生：

1)在第 j + 1 层，X 是簇首：对于每一个 Y 的孩子节点所在的簇，一个外部簇首 Z(Z ≠ Y，且度最小)将会成为新的父亲节点。

2)在第 j + 1 层，X 不是簇首：Y 目前的孩子节点的新父亲节点变成 X。

和分裂过程一样，合并过程也是周期性地调用，以减少开销。

4 系统实现

P2P-Streaming 系统采用 Java 语言实现，Java 语言具有跨平台能力，用 Java 开发的客户端软件便于移植并部署到不同类型的终端设备上。考虑到 Java 的执行效率和服务器负载问题，以后可能会加载一些非 Java 实现模块，比如部分用 C 语言实现的流媒体处理模块。

为了让系统结构清晰，系统分为应用层和通信管理层。在系统的通信管理层，实现了 ZIGZAG 算法，并且提供了 3 种不同的通信机制：TCP 连接方式，可靠有序的 UDP 方式以及一般 UDP 方式。通信管理层为上层应用提供基本的通信服务，上层应用可以根据具体需要，根据应用的特点，灵活选择不同的通信方式，如传输流媒体数据使用一般的 UDP 方式，对等网结点之间传输控制信令使用 TCP 连接方式。为了提高由 Java 语言实现的网络通信效率，在实现上采用了最新的 Java NIO 技术(JDK1.4 新引入了非阻塞式 IO)。

5 总结和展望

本文通过研究 ZIGZAG 模型的控制拓扑和数据传输拓扑两方面主要技术的基础上，设计实现了流媒体在线直播系统 P2P-Streaming。该系统解决了传统树型结构组播树中随着节点层次的增高，负载急剧增大而形成瓶颈的问题。

参考文献

1 Banerjee S, Bhattacharjee B.A Comparative Study of Application -layer Multicast Protocols[M]. Submitted for Publication, 2002.

2 Tran D A, Hua K A, Do T T. ZIGZAG: An Efficient Peer to Peer Scheme for Media Streaming[C]. Proc. of IEEE INFOCOM'03, San Francisco, CA, USA, 2003.

3 Chu Yanghua, Ray S G, Zhang Hui. A Case for End System Multicast[C]. Proc. of ACM SIGMETRICS, 2000.

4 Banerjee S, Bhattacharjee B, Kommareddy C. Scalable Application Layer Multicast[C]. Proc. of ACM SIGCOMM, Pittsburgh, PA, 2002.