



北京大学

# 硕士研究生学位论文

题目： 基于 RTMP 协议的高可用  
直播系统的设计与实现

姓 名：	韩光亮
学 号：	1301210646
院 系：	软件与微电子学院
专 业：	软件工程
研究方向：	软件开发
导师姓名：	蒋严冰 副教授

二〇一六年七月



## 版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。



## 摘要

随着网络基础设施的迅猛发展，带宽不再成为数据传输的主要障碍，人们也日益不满足于文字图片等传统的沟通交流方式。与此同时，实时的视频直播技术将人与人之间的沟通拉倒了零距离，受到了人们的青睐。本文将利用现在已有的开源技术搭建一套高可用的直播流媒体系统及其可视化系统。

本文首先介绍了现有直播流媒体技术的发展状况、直播协议以及负载均衡方式。在介绍直播流媒体技术发展状况时主要调查了目前市场上主流的直播协议；在介绍直播协议部分分析对比了各种主流协议的优缺点以及选用 **RTMP** 协议作为直播协议的原因；在介绍负载均衡方式部分，介绍了市面上采取的主流的负载均衡方式。

之后介绍了 **RTMP** 协议的相关原理，论证了负载均衡的必要性以及可行性。提出并实现了使用 **DNS** 负载均衡技术+**LVS** 负载均衡技术+**SRS** 服务器的高可用直播流媒体系统解决方案。其中 **SRS** 服务器组建为集群真实对外提供服务，**DNS** 负载均衡作为最上层的全局负载均衡技术，初步调度用户请求，**LVS** 负载均衡作为下层的局部负载均衡技术，再次将请求均匀的散落到具体的各台真实服务器上，各台真实的服务器对外提供服务。

之后利用现有技术，针对集群做出一个可视化系统，便于运维人员对于整个集群进行数据监测以及异常管理。

最后对于直播流媒体系统进行了部署以及验证测试，测试包含功能测试以及压力测试两部分，证明了系统的高效性以及鲁棒性，解决了单机流媒体服务器存在的不可靠、无法支持大并发的问题。对于可视化系统做了功能验证，较好的达到了预期目标。

关键词：直播，负载均衡，集群



# The Design and Implement of High-availability Live Stream Cluster Based on RTMP Protocol

Han Guangliang( Software Development )

Directed by Jiang Yanbing

## ABSTRACT

With the rapid development of network infrastructure, bandwidth is no longer a major obstacle for data transmission, and people are increasingly unsatisfied with the traditional communication ways like text and pictures. Real-time live video communication technology solves this problem, it pulls closer between people all over the world and wins its way of favor. This paper uses open source technology to build a high availability of live streaming media system and visualization system.

This paper introduces the development of the existing live streaming media technology, broadcast protocols and load balancing methods in the beginning. The part of live streaming media technology introduction mainly investigated the mainstream broadcast protocol in the field. In the next, it analyses the advantages and disadvantages of presented live protocol and gives the reason why to choose RTMP protocol as the basic live broadcast protocol. At the last, it introduces the mainstream load balancing methods in the market.

Following the introduction of the principle expounded RTMP protocol demonstrates the necessity and feasibility of load balancing. It proposed to use DNS load balancing technology and LVS load balancing and SRS server to achieve high availability of open-source programs live streaming system. Wherein the SRS servers cluster formation, DNS load balancing as uppermost global load balancing method, initially scheduled user requests, LVS load balancing as a local load balancing lower allocation request which is from DNS scheduling again into concrete individual real server, users are served by each real server directly.

After the use of existing technology to make a visualization system for the cluster, easy the operation and maintenance personnel for the entire cluster data monitoring and exception management.

Finally, for live streaming media cluster system the paper executed the deployment and verification testing. Test which contains function testing and stress testing proved the whole system having great efficiency and the robustness and can solve the issues that the

stand-alone streaming media server unable to support large concurrency requests. For visualization system to do functional verification, proved it achieving the desired goals.

**KEY WORDS:** Live, Load Balancing, Cluster



## 目录

<b>第一章 引言</b>	<b>1</b>
1.1 课题背景	1
1.2 研究意义	2
1.3 研究内容	2
1.4 本文的组织结构	2
<b>第二章 相关理论研究</b>	<b>4</b>
2.1 流媒体概述	4
2.2 主流直播协议	4
2.3 RTMP 协议	5
2.3.1 RTMP 消息	5
2.3.2 RTMP 消息块	6
2.3.3 RTMP 消息分块过程	6
2.3.4 RTMP 消息传递过程	7
2.3.5 RTMP 地址格式	7
2.4 服务器直播过程	7
2.5 负载均衡技术	8
2.6 J2EE 技术	9
2.7 本章小结	10
<b>第三章 系统的需求分析</b>	<b>11</b>
3.1 目标分解	11
3.2 直播流媒体系统的需求分析	11
3.2.1 直播流媒体系统的需求描述	11
3.2.2 需求分解	11
3.2.3 技术可行性分析	12
3.3 可视化系统的需求分析	13
3.3.1 需求描述	13
3.3.2 需求分析	14
3.3.3 用例模型分析	15
3.4 难点以及创新点	16

3.5 本章小结 .....	17
<b>第四章 直播流媒体系统的设计与实现 .....</b>	<b>18</b>
4.1 基准流媒体服务器选择 .....	18
4.2 负载均衡技术选择 .....	19
4.3 系统设计 .....	20
4.3.1 系统整体架构设计 .....	20
4.3.2 DNS 集群架构设计 .....	21
4.3.3 LVS 集群架构设计 .....	22
4.3.4 SRS 集群架构设计 .....	26
4.3.5 防恶意穿透攻击 .....	28
4.4 关键模块的实现 .....	30
4.4.1 DNS 负载均衡 .....	30
4.4.2 LVS 负载均衡 .....	30
4.4.3 SRS 服务器集群 .....	33
4.4 本章小结 .....	35
<b>第五章 可视化系统的设计与实现 .....</b>	<b>36</b>
5.1 系统整体架构 .....	36
5.2 数据库存储设计 .....	37
5.2.1 采集程序数据库设计 .....	37
5.2.2 展示程序数据库设计 .....	38
5.3 主要模块设计与实现 .....	40
5.3.1 数据采集模块设计与实现 .....	40
5.3.2 数据处理模块设计与实现 .....	41
5.3.3 数据同步模块设计与实现 .....	43
5.3.4 数据展现模块设计与实现 .....	44
5.4 本章小结 .....	46
<b>第六章 系统功能验证与性能测试 .....</b>	<b>48</b>
6.1 测试策略选择 .....	48
6.1.1 流媒体系统测试策略 .....	48
6.1.1 可视化系统测试策略 .....	48
6.2 部署及测试环境 .....	48
6.2.1 直播流媒体系统 .....	48
6.2.2 可视化系统 .....	50

6.3 流媒体集群测试.....	50
6.3.1 SRS 单服务器测试.....	50
6.3.2 SRS 集群测试.....	51
6.3.3 LVS 功能测试.....	52
6.3.4 集群功能测试 .....	54
6.3.5 集群性能测试 .....	55
6.4 可视化系统测试.....	58
6.4.1 机器注册测试 .....	58
6.4.2 数据展现测试 .....	60
6.5 本章小结.....	61
<b>第七章 结论及展望 .....</b>	<b>62</b>
7.1 回顾与总结.....	62
7.2 下一步优化方向.....	62
<b>参考文献 .....</b>	<b>64</b>
<b>致谢 .....</b>	<b>66</b>
<b>北京大学学位论文原创性声明和使用授权说明 .....</b>	<b>67</b>



## 第一章 引言

### 1.1 课题背景

随着网络基础设施的快速完善，人们对于网络服务的需求也在不断的发生变化。在互联网初期，网络带宽窄，无法支撑大规模的数据传输，人们获取信息以及沟通的方式也仅仅局限于文字图片等低带宽要求的技术手段。并且伴随着网络带宽的提升，以及多媒体压缩编码技术的快速进步，流媒体市场也得到了迅猛的发展，开始逐步打破以文本和图片为主的传统互联网市场的垄断<sup>[1]</sup>。直播这种可以随时互动的沟通方式更易满足人们的真实需求，更加受到人们的青睐，同时沟通中产生的认同感也更容易让人感到满足。

在流媒体市场已经逐步发展成一个朝阳产业的背景下，越来越多的企业开始涉足，大力发展流媒体技术，纷纷推出了各自的直播业务，整个行业呈现出一种欣欣向荣的景象。

流媒体直播技术的核心在于流媒体服务器，其是向终端用户提供视频服务的最重要的一环，承载着提供服务的职责。接收、缓存、调度以及传输流媒体内容是流媒体服务器的主要功能，流媒体系统的高效及稳定都在很大程度上取决于流媒体服务器的性能以及服务质量<sup>[2]</sup>。因此，只有具有较大的数据吞吐量的服务器系统才能应对随时可能接入的数据请求，为终端用户提供稳定可靠的服务。在目前环境下，庞大的用户群与薄弱的流媒体服务器接入及处理能够成为了流媒体业务发展的瓶颈。目前单台流媒体服务器可容纳链接在 1000-5000 之间，同时目前单服务器网卡容量一般为万兆网卡，而一个 720P 的直播视频经过 H.264 压缩编码也需要 1M 的带宽进行传输。在不考虑服务器可容纳链接的情况下，最大可容纳同时直播人数约为 10000 人，远不能满足庞大用户群的访问需要。同时，从目前流媒体技术的发展来看，网络带宽的增长远低于处理器速度和内存访问速度的增长，制约着流媒体技术的发展，越来越多的瓶颈会出现在服务器端<sup>[3]</sup>。

如何解决大规模环境下单服务器可容纳连接数与大规模用户接入导致的服务器拒绝服务的问题、日益增长的视频带宽占用与带宽限制的问题，如何为终端用户提供更加优质流畅的直播体验是各家企业都在努力解决的问题。

同时，针对于集群系统，集群数据的可视化也是其中重要的一环，相较于单机系统，数据更加丰富庞杂，如何组织协调这些数据，使得运维人员能够直观的看到集群的运行情况，也是需要去解决的问题。

## 1.2 研究意义

直播流媒体服务器作为直播技术的关键，直接为用户提供服务。但单服务器的接入以及处理能力是有限的，仅靠提升单服务器的处理性能远不能满足高并发环境下的用户访问需求。

同时由于视频直播大多基于长连接，故相较于普通 HTTP 环境下的高频次短服务时间的并发来说，视频直播对于资源占用更多时间更长，对于系统的要求更高，并发问题也更加难以解决。

单服务器提供直播服务无法保证服务的可交付性，一旦遇到宕机等问题，就会使得服务不可用，用户无法正常访问，从而造成非常不友好的用户体验。保证系统的鲁棒性是当前急需达成的目标。

同时，如何友好直观的看到集群的运转情况，也是运维人员需要面对的一个难点。

针对以上问题，本文提出一套有效合理的解决方案，能够使得系统能够支撑单服务器无法达到的高并发性能，并提升系统可用性，安全性，同时用现有 web 技术实现一套美观简单的数据可视化系统，利于运维人员的监控，以上便是本文意义所在。

## 1.3 研究内容

针对以上提出的问题，本文将基于 RTMP 协议以及开源软件提出一套高可用的直播解决方案，并针对该解决方案进行实现以及优化，同时为整个系统提供一个简易直观的可视化工具供集群数据监测。具体内容如下：

- 1、选定稳定高效的流媒体服务器作为基准服务器。
- 2、搭建流媒体服务器的集群
- 3、设计基于 DNS 技术的负载均衡策略的全局负载均衡集群。
- 4、设计基于 LVS 技术的负载均衡策略的局部负载均衡集群。
- 5、供运维使用的数据可视化系统。

## 1.4 本文的组织结构

本文按照发现问题、提出问题，解决问题，最终进行验证的顺序组织文章结构。总体结构如下：

第 1 章引言首先介绍了本论文的研究背景，指明了当前流媒体迅猛发展，欣欣向荣，并指出当前单台直播流媒体服务器的诸多不足，明确了本文的研究内容，并说明了本文的研究意义。

第 2 章相关理论研究，首先对流媒体技术做了一个概述，然后对比了当前主流的

直播流媒体协议，讲明了选择 RTMP 协议的原因所在，之后分析了目前市场上针对此类问题的普遍解决方案，并介绍负载均衡相关理论，最后讲解了设计以及实现可视化运维系统使用到的相关 J2EE 技术。

第 3 章系统的需求分析，首先将系统的整体需求分解为直播流媒体系统需求以及数据可视化系统需求两大部分。针对直播流媒体系统，首先做了需求描述，然后针对需求进行了相关目标分解，最后进行了可行性分析。针对可视化系统，首先描述了需求，然后进行了相关分析，最后建立用例图描述整个需求。本章最后讲述了系统的难点以及创新点。

第 4 章直播流媒体系统的设计与实现，首先提出了问题解决的思路，然后通过对比选择出了基准流媒体服务器以及负载均衡协议。然后将系统分为多个子模块进行了详细的设计，之后给出了关键技术的具体实现过程。

第 5 章可视化系统的设计与实现，首先对整体系统进行了架构设计，然后描述了系统的数据库存储设计，最后描述了主要模块的设计过程以及具体实现。

第 6 章系统功能验证与性能测试，首先介绍了系统的部署以及测试环境，然后针对直播流媒体集群分别进行了功能测试以及压力测试，最后针对可视化系统进行了整体流程的功能测试。

第 7 章是总结与展望，对全文工作做总结，以及目前正在进行的工作、下一步需要研究的重点和将来工作的展望。

## 第二章 相关理论研究

### 2.1 流媒体概述

流媒体是指采用流式网络技术在网络中进行传输的连续时基媒体,比如目前市场中视频、音频或其他类似的流媒体文件,采取的播放技术并非在播放前进行下载,而是将部分媒体内容放入服务器内存,载入多少就向用户传输多少,做到数据流随时传送随时播放,这样如果流媒体数据的传输速度大于或等于需求速度,用户看到的内容就是连续不断的,与下载后观看的效果完全相同。流式传输技术避免了用户必须下载全部文件才能观看的缺陷<sup>[4]</sup>。

目前流媒体技术大体可分为在线直播技术以及网络点播技术两种<sup>[5]</sup>。

在线直播技术是指用户能够实时地观看到在网络另一端同步发生的状况,用户所看到的内容不是事先录制好的视频内容,而是网络一端采集的实时数据。需要采集端以及用户同时在线方可完成直播功能。在线直播应用领域广泛,视频聊天、网络会议、在线教育等均为基于视频的在线直播模式<sup>[6,7]</sup>。

网络点播技术则要求视频内容被提前录制完成放在服务器端,当用户来访问时,将视频流式的传递给用户,使得用户获取到相应数据,用户观看到的并非是实时产生的数据,而是以往固化下来的数据。

### 2.2 主流直播协议

当前直播流媒体技术主要分为两种:文件 Cache 与流式 Cache。其中流式 Cache 的实时性要优于文件 Cache。目前采用流式 Cache 的协议主要有:RTSP (Real Time Streaming Protocol)、RTMP (Real Time Messaging Protocol)、MMS (Microsoft Media Server Protocol) 等,采用文件 Cache 的主要为苹果公司提出的 HLS (HTTP Live Streaming) 协议<sup>[7]</sup>。

RTMP 协议是 Adobe 公司在 1990 年提出。用于在 Flash 平台之间传递音视频以及数据,RTMP 协议包含多媒体数据的传输以及多媒体播放控制两部分,协议下层基于 TCP 协议,是一种提供可靠交付服务的协议<sup>[1,8]</sup>。

HLS 协议是苹果公司制定并推广的一种流媒体直播协议,协议下层基于 HTTP 协议<sup>[9]</sup>。由于基于 HTTP 协议,因而协议简单,易实现,其交互也非常方便,无需第三方客户端,只要有浏览器就可使用,实现成本低。但 HLS 协议用于直播则有将近 30s 的延迟。因此不适合延时要求高的场景。



RTSP 由哥伦比亚大学、网景以及 RealNetworks 公司设计及提交的应用层协议，协议提供了一种有效地通过网络传送多媒体数据的方式。<sup>[10]</sup>

MMS 是由微软公司提出的应用层协议，其用于访问 Windows 媒体发布点上提供的内容<sup>[1]</sup>。

由于 RTSP 协议需要开发单独的客户端，MMS 协议仅用于微软平台上，故目前网络直播领域，RTMP 协议以及 HLS 协议占了绝大多数的市场份额。

RTMP 协议基于 Flash 平台，同时目前 Flash 高度普及，占据了事实上的市场垄断地位，故也可认为终端用户已经将 Flash 作为基准工具安装进了使用设备，所以不需要额外的客户端，易推广。再次，相较于 HLS 协议，RTMP 协议拥有更低的延迟，更加受到各大直播厂商的青睐。

目前在国内的主流直播平台中，斗鱼、虎牙、战旗、龙珠、熊猫等均采用 RTMP 协议作为平台直播协议，在美国 Twitch 等直播平台也采用 RTMP 协议作为平台直播协议。本文也将采用 RTMP 协议作为基准协议进行设计实现。

## 2.3 RTMP 协议

RTMP 协议包含一整套的传输以及控制格式，其首先将所需传递的数据封装为 RTMP 数据包，然后将数据包通过互联网进行网络传输<sup>[1,8,11]</sup>。在 RTMP 协议中，连续不断的一组数据称之为媒体流，媒体流被流唯一标识（Stream ID）所标志。基本的数据单元被称之为消息（Message），消息必须归属于某一个特定的媒体流。但在真实的传输过程中，由于 TCP 协议可承载数据量的限制，消息会被拆分成下一级的数据单元，称之为消息块（Chunk）。

### 2.3.1 RTMP 消息

消息是 RTMP 协议中基本的数据单元。RTMP 协议共定义了十余种消息类型，分别发挥着不同的作用。协议中 Message Type ID 是消息类型的唯一标识。Message Type ID 在 1-7 为协议自用，用于协议间时序的控制等；Message Type ID 为 8 的消息则代表当前进行的是音频数据的传输；Message Type ID 为 9 的消息用于标明视频数据的传输；Message Type ID 为 15-20 的消息则被用于发送一些命令，如播放，暂停等。

RTMP 的消息头部（Message Header）分为四大部分：消息类型（Message Type ID），消息的负载长度（Payload Length），时间戳（Timestamp）消息所属媒体流（Stream ID）<sup>[1,8]</sup>。消息的整体结构如图 2.1 所示。

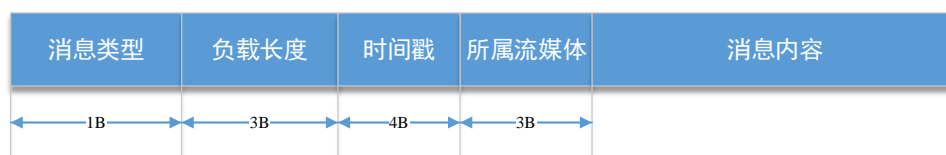


图 2.1 RTMP 消息格式

### 2.3.2 RTMP 消息块

RTMP 消息是协议层面的数据单元，其并未规定消息内容的大小，但实际网络传输中，TCP 协议所承载的数据块有大小限制。故依据 RTMP 协议规定，在传输过程中传递的消息内容不能超过一定大小的消息块，默认为不得大于 128 字节，大于该字节数的消息会被切割为满足条件的消息块进行传输。消息块首部（Chunk Header）有三部分组成：标识本块的块头部（Chunk Basic Header），标识本块负载所属消息的所属消息头部（Chunk Message Header），以及当时间戳溢出时才有内容的扩展时间戳（Extended Timestamp）<sup>[1,8]</sup>。消息块的报文结构如图 2.2 所示。



图 2.2 消息块报文结构

### 2.3.3 RTMP 消息分块过程

在切割过程中，消息内容被切割成大小固定的数据块，并在每一个数据块首部加上消息块首部组成相应消息块<sup>[1,8]</sup>。消息分块过程如图 2.3 所示，一个大小为 350 字节的消息被切割成两个 128 字节的消息块以及一个 94 字节的消息块。

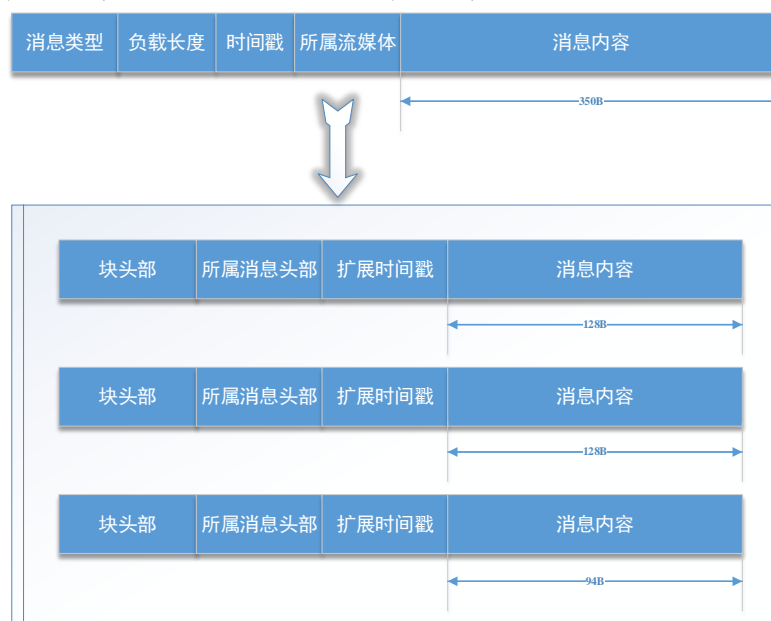


图 2.3 RTMP 消息分块模块过程

### 2.3.4 RTMP 消息传递过程

RTMP 消息传递的过程如下：

1. 发送端指定媒体流唯一标识
2. 发送端将媒体数据封装成消息并切割成块
3. 基于 TCP 协议，发送端将消息块传递出去
4. 同样基于 TCP 协议，接收端拿到消息块并进行重新组合
5. 接收端将组合后的消息进行解封装，恢复出原始媒体数据

### 2.3.5 RTMP 地址格式

RTMP 协议规定，利用 RTMP 协议进行的直播，其 URL 地址格式如图 2.4 所示：



图 2.4 RTMP 地址格式

其中开头的 rtmp 为协议头；后续的 www.abc.com 为域名或者亦可为 IP 地址；需要能够指向有效的 RTMP 服务器上；再向后的 channelId 为频道的唯一标识，统一 channelId 可视为统一频道，在现有的公共 rtmp 流媒体服务提供商中，一般将一个频道 ID 分配给某一个消费的公司或者机构，该频道 ID 下的所有 StreamID 均归属公司或者机构所有，由其进行分配，在不同的 channelId 情况下，同一个流 ID 亦代表不同的直播流对象；streamID 为流唯一标识，在同频道下，streamId 用于区分流与流的不同。

## 2.4 服务器直播过程

上文描述了 RTMP 消息的传递过程，而 RTMP 流媒体服务器则承载链接建立以及消息中转的功能。直播的发送方以及接收方并非直接建立一条稳定的数据传输通道，而是必须通过服务器进行数据中转，整体过程如下：

1. 数据发送端进行数据采集，压缩编码以及消息分块
2. 发送端与服务器建立一条稳定的长连接
3. 发送端将数据发送至服务器，服务器开辟出一个类似缓存池的结构存储接收到的数据
4. 接收端请求数据，与服务器建立稳定的长连接
5. 服务器将暂存区的数据转发给接收端，没有则继续一直接收发送端发来的数

据，当接收的数据大于暂存区大小时，如超过 3s 时长，则覆盖掉原有数据，使暂存区的数据永远最新

#### 6. 接收端拿到数据后进行解码数据整合然后播放

## 2.5 负载均衡技术

并发可分为两大类。一类是所有的用户在同一时刻做同一件事情，这是严格意义上的并发，另外一类是所有的用户在同一时刻可做同一件事情也可做不同的事情。这种并发与前一种并发概念有一点区别，就是其所作的操作类型的含义不一样，前一种并发所作操作可以简单的认为是一件事，而后一种并发范围就相对要广，对它而言，这些请求的类型可以相同，也可以不同。对从整个系统的层面来看，其本质是一样的，就是有很多用户同时对系统发起了请求或者进行了操作<sup>[12,13]</sup>。

目前在解决高并发的问题上，负载均衡技术是目前比较主流且应用效果较好解决方案。负载均衡英文名称为 **Load Balance**，其原理就是将本来单操作单元需要完成的任务通过一定的算法分配个多个操作单元上并行执行。负载均衡技术使得企业拥有相当于整片云的处理及存储能力<sup>[14]</sup>，能够成功地解决高并发的难题。但在目前已有的负载均衡的方式中，每类基本都是针对某一领域或相关领域而提出的解决方案，市场上并没有某一种负载均衡方案能够解决所有问题，对于负载均衡方案的选型以及设计也直接影响到系统的效率。

负载均衡可分为从硬件以及软件两个维度进行的负载均衡，其中：

- 软件负载均衡分为两大类：基于操作系统的软件负载均衡以及基于应用的软件负载均衡。基于操作系统的负载均衡是指操作系统本身支持负载均衡，例如 **LVS** 就是 **Linux** 操作系统内置的负载均衡模块；基于应用的软件负载均衡是指在操作系统上安装一个或多个附加软件来达成负载均衡的能力，如 **HAProxy**，**Nginx** 等。软件负载均衡的优点是基于特定的环境进行简单的配置即可进行负载均衡，由于大部分的负载均衡软件均为开源软件，所以其成本相对较低，对于一般的负载均衡需求软件负载均衡方案就可以满足。

- 硬件负载均衡是使用一种单独的硬件负载均衡设备，该设备处于服务器与外网之间，它由于特定的设备完成特定的任务，硬件设备与操作系统无关，由于其整体硬件均为解决该特定问题而生，故整体性能相对于软件负载均衡而言较高，同时其能够进行多样化策略的负载均衡，能够对流量进行智能化的流量，对于负载均衡需求能够达到很好的满足<sup>[15]</sup>。但硬件负载均衡设备价格高昂，动辄百万，是其弊端所在。

## 2.6 J2EE 技术

J2EE 是 Sun 公司推出的一种 Web 模型,采用分层结构,使得层与层之间相互独立,每个层面只做特定的任务,无需关心上下层所做内容。在分层结构中,下层向上层通过接口提供服务,对于上层而言,下层的具体实现是透明的,在接口不变的情况下,实现发生变化,对其他层产生的影响会达到最小。大大降低了系统之间的耦合性,而且使得系统具有更好的可维护性和可扩展性,分层后使各层功能变得简单清晰明了,只需要对所需接口进行编程即可<sup>[16,17]</sup>。

以下是 J2EE 的典型架构:

- **客户层** 即将提供的服务通过何种形式表现给用户。目前主流是通过 HTML 的形式展现给用户,也有少部分内容通过 applet 等其他的一些基于网络的程序展现。
- **表示层** 表示层将数据组装为用户可识别的内容,然后传递给浏览器解析。表示层一般使用 JSP、Velocity、Servlets 完成动态内容渲染显示和隶属于内容展示的流程逻辑控制。
- **业务层** 为表示层提供正确有效的业务数据及相关的业务逻辑,业务层掩盖了数据获取流程,对于表示层来说,底层的实现方式是透明的,无需得知下层处理逻辑,只需要关心展示逻辑以及内容的正确性即可。
- **数据库存储层** 为业务层提供数据支撑,底层由一个或多个数据库组成,封装了隶属于数据库的所有操作,业务层只需要针对相应接口进行编程便可获得正确的数据,无需关心底层数据库如何实现。

Spring Web MVC 是一种表示层的请求驱动类型的轻量级 Web 框架,指使用请求-响应的模型,依据 MVC 架构,将表示层与业务层进行解耦合。

Spring 是目前非常流行的开源业务层框架,其设计目的为降低程序开发复杂性。分层架构是框架的最主要的优势之一,这一架构允许程序编写者使用最合适的组件,同时提供一种集成框架<sup>[18,19]</sup>。

MyBatis 是一个基于 Java 的数据持久层框架,是对象 / 关系映射的优秀解决方案。MyBatis 的源码开放,是性能较高持久层框架。内容包含 SQL Map 和 Data Access Objects (DAO) 等。

MySQL 是目前最流行的开源关系型数据库,由于其开源的特点以及出色的数据处理能力,其性价比极高,被目前各大互联网公司采用。

Quartz 是 OpenSymphony 开源组织贡献的一个开源定时项目,它可以同 J2EE 或 J2SE 应用程序相结合使用也可以单独使用。Quartz 性能非常优秀,可以用来创建以万级量级的任务,被目前的定时任务广泛采用。

Sigar 是 Hyperic-HQ 产品的基础包,是 Hyperic HQ 主要的数据收集组件。它支持

多平台的系统信息收集和处理。

## 2.7 本章小结

本章介绍了与本论文相关的理论，为全文奠定理论基础。首先对流媒体技术做了一个简单的介绍，然后讲述了目前业界主流的直播协议，下来重点介绍了直播协议中的 RTMP 协议，讲解了 RTMP 消息及消息块以及其整个的交互过程，并指明了采用 RTMP 协议的原因，紧接着介绍了负载均衡相关的理论知识，最后介绍了数据可视化系统用到的相关 J2EE 的内容。

## 第三章 系统的需求分析

通过上一章对本系统中相关技术的分析，对相关技术的概念有了一定的了解。系统最终的设计目的为采用 RTMP 协议作为基准直播协议进行直播系统的设计，同时欲采用传统的 J2EE 的方式为直播系统搭建一个简单的可视化系统。本章主要对系统进行详细的需求分析，明确系统的所要达成的目标。

### 3.1 目标分解

由上述描述可知，整体系统欲达成的目标包含一个基于 RTMP 协议的直播流媒体系统以及一个针对于该系统的供运维人员查看的数据可视化系统。所以本文的后续需求分析、系统设计实现以及测试均分为直播流媒体系统以及可视化系统两大部分展开。

### 3.2 直播流媒体系统的需求分析

#### 3.2.1 直播流媒体系统的需求描述

目前单台流媒体服务器可容纳链接在 1000-8000 之间，同时由于目前单服务器网卡最大容量为万兆网卡，而一个 720P 的直播视频经过 H.264 压缩编码单用户也需要 1M 左右带宽。在不考虑服务器可容纳链接以及最大化带宽使用的情况下，最大可容纳同时直播人数约为 10000 人，远不能满足庞大用户群的访问需要。

直播流媒体系统欲达成的目标便是通过现有技术解决单服务器所不能支撑的请求高并发。

#### 3.2.2 需求分解

在本次直播系统的设计过程中，为了达成系统的高效性及鲁棒性，需要考虑以下问题：

- 1) 服务器性能问题，需要挑选出性能较优的 RTMP 服务器作为基准服务器。
- 2) 服务器性能瓶颈问题，需要针对单服务器无法承受的并发量进行集群组建。
- 3) 集群流分发问题，假设直播流输入 A 服务器，当用户访问到 B 服务器时如何能够看到直播流。
- 4) 集群稳定性问题，如何保证集群中某一台服务器异常终止服务不影响整体服务提供。
- 5) 请求调度问题，如何将请求均匀的散落在各台服务器上。

- 6) 健康监测问题, 前端请求调度器如何自动忽略已经暂停或终止服务的服务器, 让请求智能的避开该服务器。
- 7) 安全问题, 如何避免请求绕过前端调度服务器, 直接攻击真实服务器。

### 3.2.3 技术可行性分析

经过上文分析可知, 直播系统要求达到高效性以及鲁棒性的目标, 需要以下四类资源:

**RTMP 基准服务器:** 作为数据最终载体, 基准服务器最终承担了服务提供者的角色, 其将直接为用户提供服务。

**RTMP 集群:** RTMP 服务器为用户直接提供服务, 而为了保证高可用性, 系统不可以只采用一台流媒体服务器, 采用多台的情况下, 流媒体服务器则必须能够组成集群统一对外提供服务。

**负载均衡资源:** 有了流媒体集群, 需要一种行之有效的方式, 使得用户请求能够均匀的散落到集群中的每一台服务器上, 目前业界的解决方式是通过负载均衡这一技术实现。本文亦采用此种方式解决该问题, 所以需求负载均衡资源。

**集群安全处理资源:** 处于互联网中的任何一台机器都是不安全的, 本系统中的流媒体服务器集群亦存在此问题, 故需求集群安全处理资源。

经过调研, 目前业界对于上述四类需求有以下资源。

#### ● RTMP 基准服务器资源

目前市场上主流的 RTMP 服务器有 FMS、Wowza、Red5、Nginx-rtmp-module、Crtmpserver、SRS(Simple-rtmp-server)等 RTMP 流媒体服务器。其中 FMS 以及 Wowza 为商业软件, 需要许可证, 其余均为开源软件。本文欲对比现有开源软件从而选择出一款高性能服务器作为 RTMP 基准服务器。

#### ● RTMP 集群搭建资源

开源 RTMP 服务器中 Red5 支持源/边缘(origin/edge)以及开源 Java 集群平台 Terracotta 的方式配置集群。Nginx-rtmp-module 支持推/拉(push/pull)模式配置集群, Crtmpserver 支持主从复制的方式配置集群, Simple-rtmp-server 支持源/边缘(origin/edge)以及转发(forward)的方式配置集群<sup>[20]</sup>。

#### ● 负载均衡资源

目前 DNS、Nginx、LVS、HAProxy 等是最流行的软件层面的负载均衡方式, 其中 DNS 是基于域名系统的负载均衡; Nginx 是基于应用的负载均衡, 支持 HTTP 以及 MAIL 协议的负载均衡; LVS 基于网络层 IP 的负载均衡; HAProxy 是基于 TCP 和 HTTP 应用代理的负载均衡。

RTMP 协议是应用层协议, 在网络层基于 TCP 协议收发数据, 故可采用网络层及



以下的软件负载均衡进行调度。

#### ● 集群安全处理资源

集群安全分为内部安全和外部安全。外部安全主要防入侵、防 DDoS 攻击，内部安全主要防木马的执行、安全策略、补丁、弱口令安全检测、防病毒、资源监控、文件目录保护、定期杀毒等。

在直播流媒体集群中，最大的安全来源于用户透过前端调度服务器直接攻击防火墙后的真实服务器，浪费服务器资源。而服务器对外提供服务是通过 IP 地址+端口号的方式进行。目前业界主要采用包含对外暴露统一 IP 以及关闭对外服务端口号或者更换服务端口号的方式进行防范。

综上已有资源可以看出，依据目前业界开源产品能够达到设计目标，设计出一款高可用的直播流媒体系统。

### 3.3 可视化系统的需求分析

#### 3.3.1 需求描述

在直播流媒体集群搭建完成后，如何得知集群整体以及其中各台服务器的运转情况就成为了一件非常重要的事情。如果无法得知情况，运维人员便无法有效的对集群进行管理，例如在集群负载低时关闭一些提供服务的机器，在负载高时临时开启一些服务器等

为此，需要设计出一款数据可视化系统供运维人员观察，以便运维人员做决策。

本文针对此情况，欲设计出一款简单美观的数据可视化系统，能够定时采集各台机器的运行状况，并通过一定的整理分析，有效合理的反映机器的运行状况，为运维人员的决策提供帮助。

为此，针对一个简单美观的可视化系统，提出以下需求：

- 1、功能全面，不应当只展示集群当前情况，应当能够看到每台机器的具体运行状况
- 2、简单易用，要求每当一台新机器启动时，只需要做非常少的操作便能将机器加入监控集群中
- 3、数据准确，不能产生错误数据，A 机器数据不应当展现在 B 机器的运行状态中
- 4、数据充足，不能仅仅展现出当前的实时数据，也应当展现一定时间间隔维度下的运行数据
- 5、页面简洁美观，能够给用户带来良好的用户体验

### 3.3.2 需求分析

在基于 RTMP 协议的直播过程中，RTMP 服务器承担的是数据转发的工作，它将采集端传递来的数据暂时的保存在内存中，任何播放请求进来，则将内存中的数据块转发一份给该请求，从而完成直播的交互过程。

一台服务器整体上包含有以下资源：CPU、内存、网卡、带宽等。在直播流媒体服务器中，最为重要的资源便是 CPU 的转发处理能力以及服务器的带宽。其中，CPU 的转发处理能力决定了直播延迟的高低，而服务器的带宽则决定了可接入请求数的多少。以普通千兆网卡为例，千兆网卡的最大出网带宽为 1000M，而当前 720P 的视频对于带宽的占用大于 1M，在这种情况下，一台拥有千兆网卡的机器所能支撑的最大接入请求数为 1000 个请求。故 CPU 的使用情况以及带宽的占用情况对于运维人员非常关键。

同时，集群的请求接入数量对于运维人员也非常重要，正常一台 RTMP 服务器的接入请求数从 1000-5000 不等，如果当前机器的 RTMP 连接建立数远远大于该数量，那么证明服务器出现了异常情况，如遭到恶意攻击等，此时运维人员应当去排查具体情况，及时有效的解决该问题。

针对以上需求描述，可以得知，可视化系统应当分为两大部分：

- 处于真实服务器上的数据采集程序  
数据采集程序主体功能为实时采集当前服务器的信息，共展现程序使用；
- 处于 Web 机器上的数据展现程序。  
数据展现程序主体功能为及时获取各台真实服务器的机器实时运行信息，并将多台机器数据做整合，形成集群数据。

整体系统结构如图 3.1 所示：

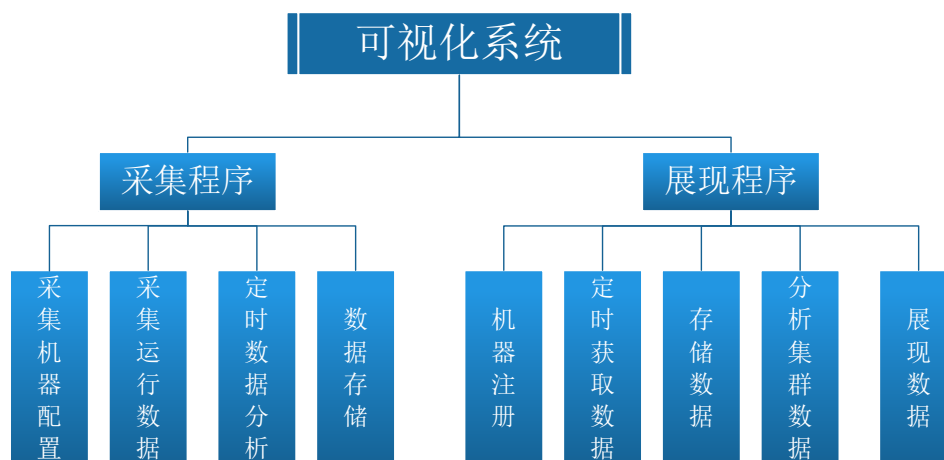


图 3.1 可视化系统整体架构

**采集程序需要包含以下功能：**

- 采集机器数据  
包含 CPU 型号、内存大小、网卡带宽大小、IP 地址、MAC 地址等机器基本数据共采集端使用。
- 定时采集机器的运行数据  
包含 CPU 实时使用情况、内存使用情况、入网流量、出网流量等数据，并定时进行分析，得到一定时间段内如 1 分钟、5 分钟、10 分钟、1 小时、1 天、7 天等数据。
- 数据存储  
对于采集得到的机器运行数据，应当进行数据持久化，供己方数据分析以及集群请求使用。
- 数据定时分析  
对于实时数据，可以实时采集，但是对于机器的非实时数据，只能通过已有的实时数据求得平均值得到，所以需要对于数据进行定时分析并存储。已共己方及集群方使用。

**展示程序需要包含以下功能：**

- 机器注册  
依据机器的 IP 地址主动的去真实服务器请求服务器的配置信息并持久化。这样可以。
- 定时获取数据  
定时去各台真实服务器上获取其运行数据并将其持久化至本地
- 存储数据  
从各台真实服务器上获得的长期数据需存至本地数据库。
- 分析集群数据  
获得各台服务器数据后，需要进行分析才能获得整个集群的运行情况，如每台机器获得的实时数据进行分析后就是集群的实时数据，每台机器一分钟的数据经过分析后是集群的一分钟的平均数据，以此类推。
- 展现数据  
获得了各个机器的数据，并且依据这些数据进行了分析得到集群的数据后，需要对这些数据进行可视化，展现给终端运维人员。

### 3.3.3 用例模型分析

上一节对需求进行了相应的分析，便可以得出系统的用例模型，如图 3.2 所示：

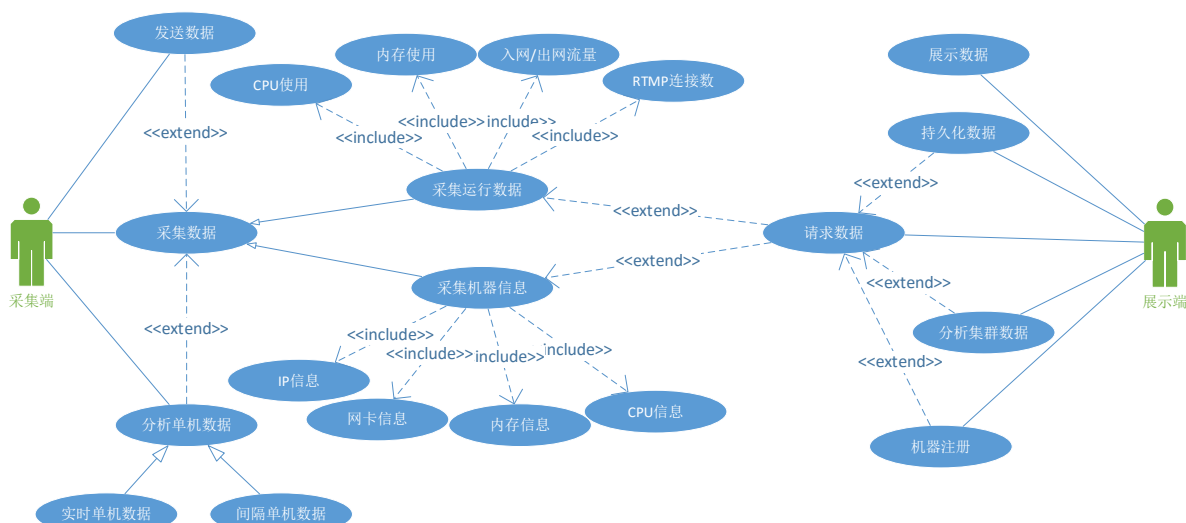


图 3.2 可视化系统用例模型

采集端需要采集机器信息以及机器的运行数据，机器信息包含 IP、网卡、内存、CPU 信息。运行数据包含 CPU 使用请、内存使用情况、入网/出网流量、RTMP 连接数等数据。采集端通过定时采集，然后定时将数据分析为 1 分钟、5 分钟、10 分钟、1 小时、1 天、7 天等时间维度的数据并进行持久化。

展示端首先会去注册集群中的各台机器，确定集群范围。然后定时去采集端获得运行数据，将其持久化至本地，然后通过一定的分析将其转化为集群数据。然后展示端对于这些数据进行一定维度的处理，最终通过浏览器呈现给终端用户。

### 3.4 难点以及创新点

从上述章节我们了解到本系统是一个针对 RTMP 流媒体服务器的高可用的集群系统及一个简单可视化系统。流媒体系统既需要保证对终端用户透明度，又需要提升现有系统的并发量以及可用性。系统的难点主要体现在以下几个方面：

#### 1) 系统整体架构设计

由于 RTMP 协议非 HTTP 协议等，其基于 TCP 长连接，故无法采用传统的 7 层负载均衡技术来解决问题。且不同与普通互联网行业，拥有大量解决方案，各大厂商整体架构图均被一览无余，可以借鉴采纳。在直播领域，相关系统架构均为各大厂商核心技术，目前尚无一个完整的解决方案对外暴露。且市面上关于 RTMP 协议的文档资源均较少，需要自己分析问题逐步剥离，最终解决问题。

#### 2) 负载均衡技术的实际使用

单层负载均衡并不能够很好的完成本文的需求，本文借鉴市面上已有负载均衡

组织方式，并结合直播需求，采用双层负载均衡的方式，从而使得基于 TCP 长连接的请求亦能被有效的散落至各台真实服务器上。

#### 3) 可视化系统的数据均值分析

可视化系统应当不仅仅看到实时数据，也应当能够可视化出一段时间内的数据。固定时间间隔的机器数据采集依靠第三方 API 即可实现，但如何将各台机器上的数据进行汇总以及如何最准确的描述一个时间段的机器平均使用情况是需要解决的问题，同时历史数据的保存方式也是一个需要讨论的问题。

### 3.5 本章小结

本章主要对于系统做了一个整体的需求分析，明确了系统所要达成的目标，并针对该目标做了任务分解。

首先在第一节详细阐述了本文试图解决的问题，并且将目标分解成了直播流媒体系统的设计实现以及可视化系统的设计实现两部分。

第二节对于直播流媒体系统的需求进行了描述，讲解了其基本需求，后对于直播流媒体系统进行了任务的目标分解，然后分析了系统设计的可行性。

第三节对于集群可视化系统的需求进行了描述，对其系统业务流程进行了分析。

第四节详细阐述了本次系统设计的难点以及创新点，这些将会在下文重点讲解，贯穿整个设计部分。

## 第四章 直播流媒体系统的设计与实现

### 4.1 基准流媒体服务器选择

目前市场上主流的基于 RTMP 协议的流媒体服务器有以下几种：

- Red5 是由 Java 语言开发的开源 Flash 流媒体服务器，其使用 RTMP 作为基准流媒体传输协议。能够实现直播以及点播功能，由于采用 Java 实现，天生具有跨平台性，但性能较低且开源社区基本已经不再更新。
- CRTMPserver 是 Evostream Media Server 的社区版本。作为 RTMP 流媒体服务器性能较好，可以实现直播与点播功能多终端支持功能，但协议支持性较差，不支持 HLS 协议且代码较为臃肿。
- Nginx-RTMP-Module 是 Nginx 的一个第三方模块，具备流媒体服务器的常见功能，支持 RTMP 及 HLS 两种流媒体协议。但相较于其他流媒体服务器，功能相对简单。
- Simple-RTMP-Server（以下简称为 SRS）是由观止创想 CTO 杨成立设计实现，其定位是运营级的互联网直播服务器，其主体核心为基于 RTMP 协议的直播流的接收以及转发，同时 2.0 版本中支持 HLS 协议。目前其拥有商业版本 BMS（Bravo Media Server）。

经过查阅各项资料以及进行测试，各类 RTMP 服务器性能如图 4.1 所示：

对比属性	Red5	Nginx-RTMP	CRTMPServer	SRS
并发量	1k	3k-4k	2k	3k-5k
多线程支持	不支持	支持	不支持	不支持
最低延迟	2s	0.8s	1s	0.8s
HLS支持	不支持	支持	不支持	支持
集群支持	支持	支持	支持	支持
HTTP回调	不支持	支持	不支持	支持
社区维护	已停止	已停止	有	有

图 4.1 各类 RTMP 服务器性能对比图

由图可以看到在性能方面，Nginx-RTMP 模块以及 SRS 具有相对较好的性能。而 SRS 在性能方面更胜一筹，同时，SRS 作为 BMS 的开源版本，其更新迭代一直在持续，目前最后一次更新在今年 1 月，社区较为活跃且功能更为全面。故本文欲采用 Simple-RTMP-Server 作为基准服务器。

## 4.2 负载均衡技术选择

RTMP 协议在 TCP/IP 七层协议的第四层传输层采用 TCP 协议进行数据传输，故只可采取支持四层及以下各层的负载均衡的负载均衡技术。同时由于 RTMP 协议流是基于 TCP 长连接，要求在同一时刻建立起大量的连接，而任何单负载均衡设备均无法保障同一时刻的大规模并发且达到高可用效果，故需采用双层负载均衡的方式。

目前市场上主流第一级负载均衡采取 DNS 负载均衡技术。DNS 负载均衡是最早期出现的一种负载均衡技术，其利用域名解析实现负载均衡效果，在 DNS 服务器上配置多条服务器记录，这些服务器记录对应的服务器构成集群。

目前在市场上的运行在第四层软负载均衡的开源协议主要有 LVS（Linux Virtual Server）以及 HAProxy 两种。

LVS 即 Linux 虚拟服务器，是一个虚拟的服务器集群系统。LVS 项目在 1998 年 5 月由章文嵩博士成立，LVS 集群采用 IP 负载均衡技术以及基于内容请求分发技术来达到负载均衡的目的。LVS 能够将请求均衡地分发到相应的服务器上执行，同时调度器能够自动或集成其他故障感知系统对后端服务器进行故障检测，将出现故障的服务器进行智能屏蔽，最终将多台服务器组成一个高性能、高可用的虚拟服务器集群。在负载均衡的过程中，客户端和服务器的代码结构无需进行任何修改，同时对客户而言，整个集群的结构是透明的。在 Linux 内核 2.4 之后，LVS 已经作为 Linux 标准内核的一部分<sup>[21]</sup>。

HAProxy 是一款提供高可用负载均衡的代理软件，其可基于 TCP 协议也可基于 HTTP 协议应用，HAProxy 相较于 LVS 而言多了上层 HTTP 协议的良好支持，支持 Session 保持、支持正则处理、可做动静分离上层特性<sup>[22]</sup>，而其本身性能相较于 LVS 较差。

LVS 由于工作在 TCP 层之上仅作数据分发，没有流量的产生，这也决定了它在目前有所的负载均衡软件中性能最强、最稳定、对内存和 CPU 资源消耗最低。其本身抗负载能力也非常强，用有完整成熟的双机热备方案，如 LVS+Keepalived、LVS+Heartbeat 等<sup>[23]</sup>。

RTMP 协议与 HTTP 相比是两种不同的上层协议。本身并不需要 TCP 层及以上的上层特性，其做负载均衡仅仅需要流量的转发，在同等对比情况下，LVS 负载均衡方

式更加适合做 RTMP 服务器的负载均衡。

所以本文欲采用 DNS 作为上层负载均衡技术，LVS 作为下层负载均衡技术。

## 4.3 系统设计

### 4.3.1 系统整体架构设计

流媒体系统整体架构设计为 DNS 负载均衡作为上层负载均衡技术，LVS 作为下层负载均衡技术，采用 SRS 作为基准服务器进行流媒体系统环境搭建。

系统整体架构图如图 4.2 所示：

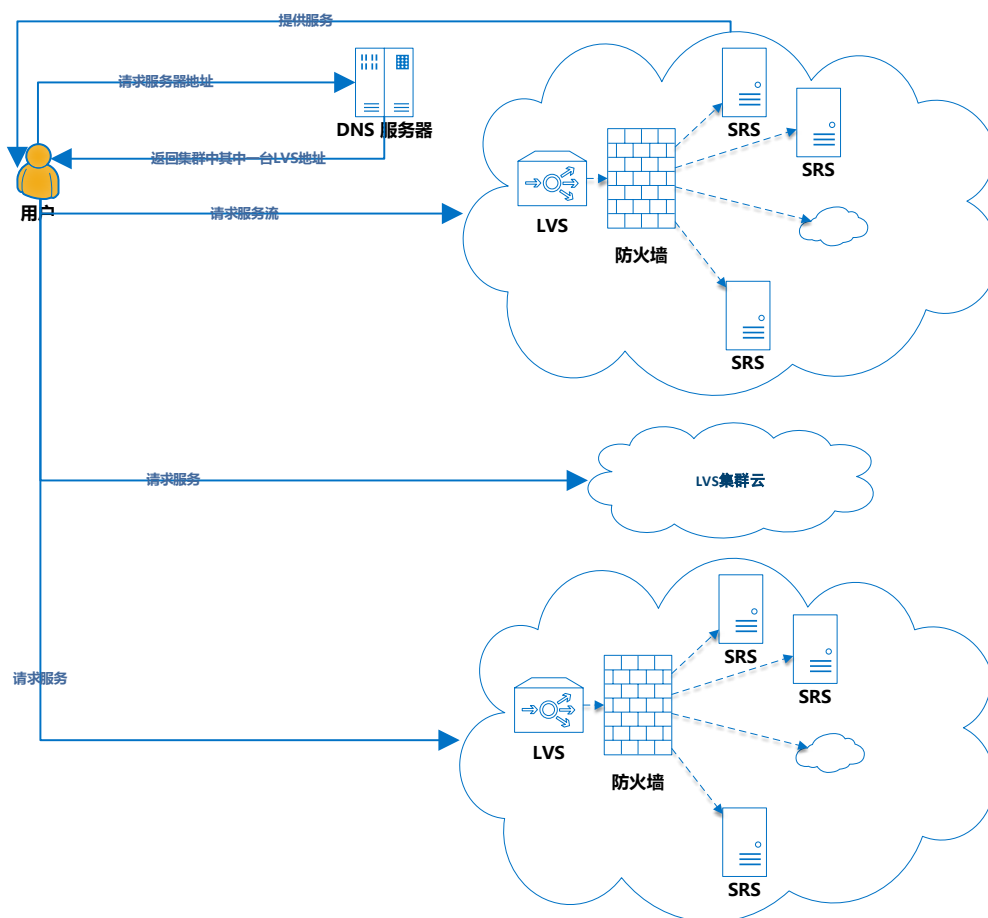


图 4.2 流媒体系统整体架构图

用户访问到真正服务器的整体过程如下：

- 用户发出请求；
- 路由器将用户请求发送至网关，网关找到相应的 DNS 解析器；
- DNS 解析器返回 LVS 服务器地址；
- 用户访问 LVS 地址；



- e) LVS 接收到请求，将请求转发给一台当前可用的 SRS 服务器；
- f) SRS 服务器为用户提供服务；
- g) 用户接收到数据，开始上传视频数据或者观看

### 4.3.2 DNS 集群架构设计

DNS 负载均衡技术的原理是，在 DNS 服务器中为同一个域名配置多个 IP 地址，每一个指向域名的请求都需要去 DNS 服务器查询提供服务的机器真实 IP，DNS 服务器会取出 DNS 文件中主机记录的真实 IP 地址并按一定的算法返回相应的结果，从而将客户端的请求散落到不同的机器上，达到负载均衡的目的<sup>[24,25]</sup>。

目前常用的负载均衡算法有以下几种<sup>[26]</sup>：

- 轮转算法（round robin scheduling）：调度器将按照 IP 轮换的方式返回给用户。
- 最少连接数算法（least counted based scheduling）：调度器将当前连接数最少的 IP 地址返回给用户。
- 最轻负载算法（least loaded scheduling）：调度器将当前负载最轻的服务器的 IP 地址返回给用户。
- RTT 值分配法：调度器将分配客户端到服务器回路时间最少的服务器 IP 地址返回给用户。
- 比例分配法：其在每个服务器运行一个监听程序，当客户需要请求服务时，会向所有服务器都发出请求，所有服务器上的监听程序接收到后会按照一种比例算法决定最终进行服务服务器。

系统采用 DNS 负载均衡作为最上层的负载调度，承载着流量入口。如果 DNS 负载均衡失败，整个系统都会崩溃，同时 DNS 负载均衡算法过于复杂，则会导致整体服务延迟增大。再一点在于如果下层服务器并非传统服务器，则以上调度算法均有缺点。为了调度公平起见，系统采用传统的 RR 策略的 DNS 负载均衡进行第一层负载均衡，RR 策略拥有算法简单，延迟低，配置简易的优点，为此需保证后端各个子集群的处理能力大致相同。整体结构图 4.3 所示：

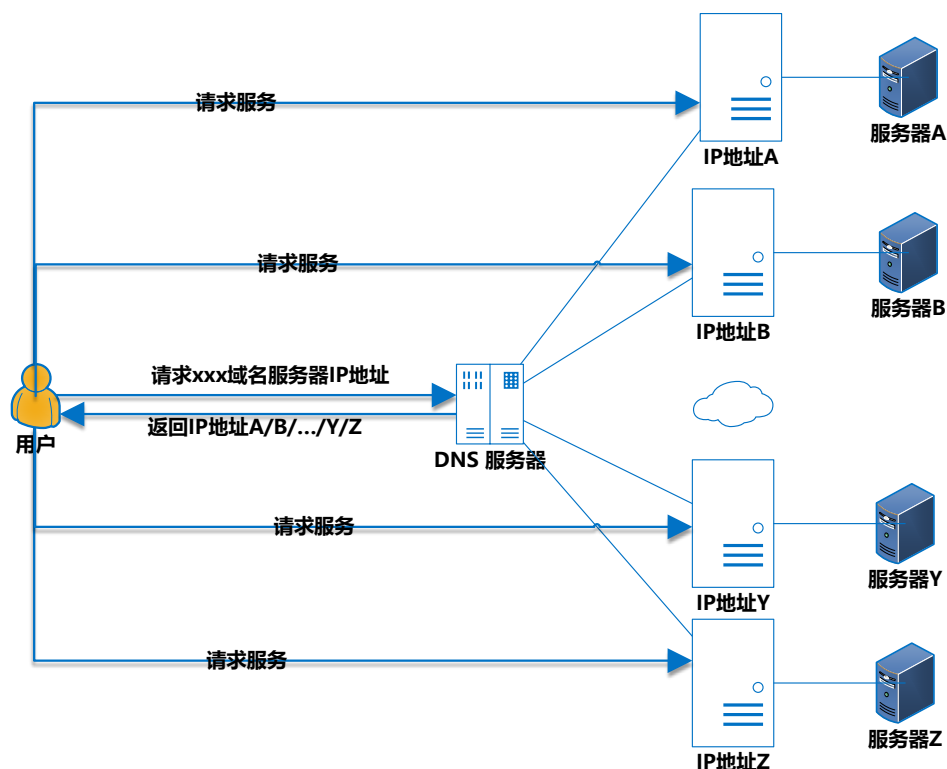


图 4.3 DNS 负载均衡方式

步骤如下：

1. 用户访问域名为 xxx 的系统；
2. DNS 解析服务器依据 RR(轮询算法)选出当前应该服务的机器，并将该机器的 IP 地址返回给用户；
3. 用户拿到需要提供服务的服务器 IP 地址后将请求发送到该服务器请求服务；
4. 服务器接到请求后为用户提供服务；

#### 4.3.3 LVS 集群架构设计

通过上一层的 DNS 负载均衡后，请求理论上便应该落到真实服务器上由该服务器为用户提供服务，但 DNS 负载均衡的 RR（轮询算法）存在以下问题，因此下层仍需要再做一次负载均衡：

- a) DNS 负载均衡器无法实时感知后端服务器状态；
- b) DNS 负载均衡器无法做端口转换等安全策略；
- c) 真实服务器 IP 地址被直接暴漏给外界用户，真实服务器容易受到攻击；

LVS 采用调度算法将 IP 层上的请求均匀地分配到不同的真实服务器（Real Server 简称 RS）上<sup>[27]</sup>。

一个完整的 LVS 集群一般由前端调度器层、中间服务器集群层和后端存储系统层

组成<sup>[28]</sup>。调度器层是集群唯一入口，处于整个集群的最前端。当请求到来时，首先由调度器处理请求，它调度器是分配服务器的决策者，其将依据调度算法将确定将由集群中的哪一台服务器进行服务然后将请求转发到相应的服务器上。中间服务器集群层由真实处理请求的 RS 组成，机器接到请求后进行处理然后将数据通过一定的规则返回给用户。在请求发送方而言，集群的整体结构是透明的，服务是来自于同一个 IP 地址，即 LVS 上所配置的虚拟 IP（Virtual IP，简称 VIP），。后端存储系统负责保证数据存储在的一致性，使得服务器集群有相同的内容<sup>[29]</sup>。

LVS 配置主要需要配置负载均衡转发模式、负载均衡调度算法以及容灾策略三部分内容<sup>[30]</sup>。

### 1、负载均衡转发模式选择

LVS 在 IP 层包含 3 种负载均衡转发模型：网络地址转换模型、IP 隧道模型、直接路由模型。

#### ● NAT 模型 -- (NetWork Address Translation - 网络地址转换):

NAT 模型原理图 4.4 所示:

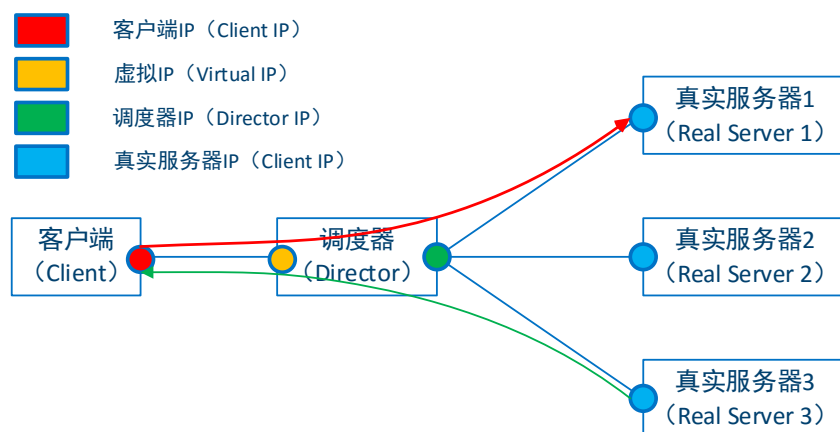


图 4.4 NAT 模型原理图

请求的数据包的 IP 头部包含处理服务器的目的地址，请求经过调度器时，该 IP 地址会被换成将要真正处理的 RS 的 IP 地址，并将此请求转发至该 RS。RS 处理完成后会把数据再次交给调度器，调度器再将数据包的源 IP 改为自己的 IP 地址，将目的 IP 改为用户 IP 地址，在整个过程中，进来以及出去的流量，都必须经过调度器<sup>[31]</sup>。

NAT 模式优势在于只有调度器需要一个对外的公网 IP 地址，集群中的其余物理服务器可以采用任何支持 TCP/IP 协议的操作系统。

NAT 模式的缺点在于扩展性不足。由于所有的请求数据以及应答数据都必须经过调度器，当 RS 增长过多时，大量的数据包都累积在调度器，调度器将成为整个系统的瓶颈，系统的整体性能就会急剧下降甚至拒绝请求。

# ● IP TUN 模型 -- (IP Tunneling IP 隧道):

IP-TUN 模型原理如图 4.5 所示:

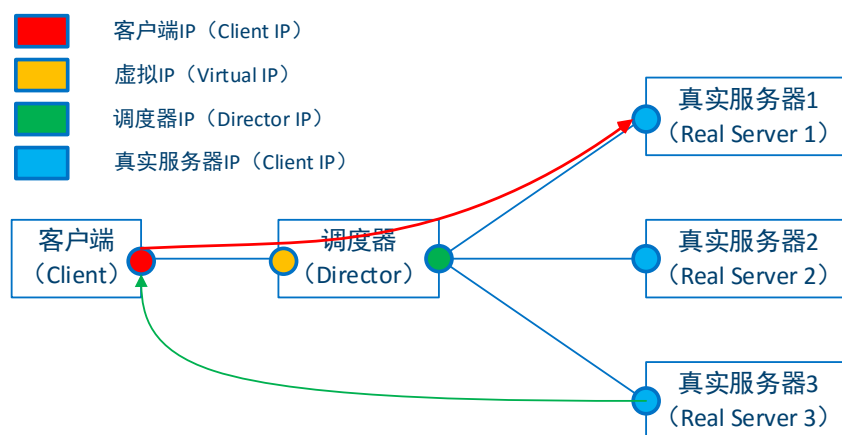


图 4.5 IP-TUN 模型

隧道模式下, 当请求到达调度器时, 调度器将生成一个新的 IP 头标记, 封装在用户发来的数据包外, 标记中仅仅包含目的 IP, 然后转发给 RS, RS 收到数据包后, 先将数据包的头反解, 还原出真实的数据包, 请求处理完成后数据会被直接返回给用户, 不再经过调度器。由于 RS 需要对调度器发过来的数据包进行反解, 所以 RS 必须支持 IP TUNNEL 协议。

IP-TUN 模式的优点在于调度器通过规定一个规则, 对源数据进行包装分发给后端的 RS, 而 RS 返回的应答包不再经过调度器, 直接发给用户, 减少了调度器的大量数据流动, 调度器不再是系统的瓶颈。在这种模式下, 一台调度器能够为很多 RS 进行分发, 同时由于其封装了新的 IP 头标记, 则在公网上能够进行不同地域的数据分发。

IP-TUN 模式的缺点在于其需要所有的服务器均支持 “IP Tunneling” 协议, 因此服务器可能只局限在部分 Linux 系统上。

# ● DR 模型 -- (Director Routing - 直接路由):

DR 模型原理图与 IP-TUN 模型原理图相同, 如图 4.6 所示:

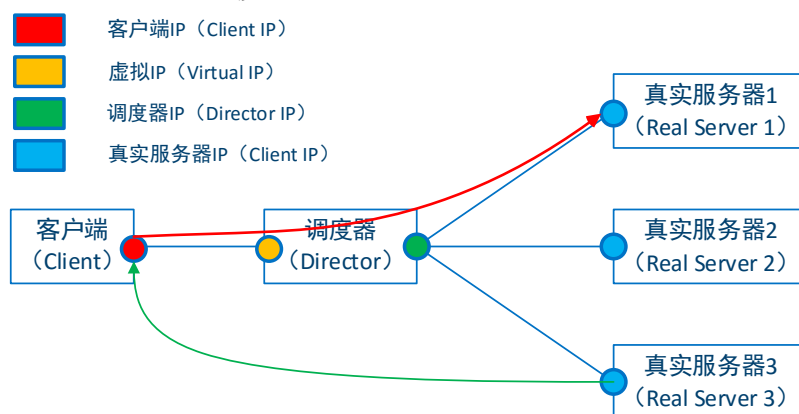


图 4.6 DR 模型原理图

区别在于，DR 模式下，调度器和 RS 使用同一个 IP 地址，称之为虚拟 IP（Virtual IP，简称 VIP）。但只有调度器对 ARP 请求进行响应，所有的 RS 对该 IP 的 ARP 请求保持静默。即调度器以及 RS 都会监听同一个 VIP，但网关只会把针对该 IP 的请求定向给调度器，调度器收到数据包后将依据调度算法找出将要处理该请求的 RS，然后将数据包中的目的 MAC 地址替换为 RS 的 MAC 地址并将请求分发给这台 RS。这时 RS 收到数据包后，判断 IP 地址是自己的 IP 且 MAC 地址与自己相同，便会进行处理，处理完毕后，由于 IP 一致，RS 可以无需做任何修改直接将数据返给用户。由于调度器要对包头进行 MAC 地址改换，所以调度器和 RS 之间必须在同一个广播域。

DR 模式最大的优势在于调度器只是分发请求，应答包将直接返回给用户。相较于 IP-TUN 模型，DR 模型无需隧道结构的强制需求，因此绝大多数操作系统均可满足要求，做为物理服务器。同时由于在用户看来，只有一台机器在为自己服务，并不对外暴露真实服务器的 IP 地址，相对安全性较好。

DR 模式的唯一缺点在于要求调度器的网卡必须与 RS 的网卡在一个广播域上。

在上述三种模型中，IP-TUN 模型由于对于系统要求较为特殊且安全性较差，故使用相对较少，NAT 模型要求请求入网以及出网均通过负载均衡器，而直播流媒体又会产生大量的流量，网卡很容易产生瓶颈，所以不适宜采用 NAT 模型。DR 模式请求入网通过网卡而出网流量由 RS 直接返回，故更加满足条件。

## 2、负载均衡调度算法选择

LVS 具有 10 种请求调度算法：分别为轮询算法、加权轮询算法、最少链接算法、加权最少链接算法、基于局部性的最少链接算法、带复制的基于局部性最少链接算法、目标地址散列算法、源地址散列算法、最短的期望的延迟算法以及最少队列调度算法等 10 种<sup>[23]</sup>。

直播服务器主要需求网卡资源，同时需要系统 CPU 具备较快的处理速度以保证直播延迟。

由于系统搭建并不能保证后端真实服务器均具有相同的直播处理能力，故不可以采取基础的轮询调度算法等调度算法作为基准调度算法。同时由于服务器处理能力不同导致的可容纳连接数不同，故不能采用基于连接数的调度算法。而系统又期望请求能够按照服务器的处理能力进行请求转发，加权轮询调度算法（Weighted Round-Robin，简称 WRR）能够依据真实服务器的不同处理能力来均衡的进行请求转发，这样可以保证处理能力强的服务器能处理相对较多的请求，处理能力弱的服务器可少处理一些请求。因此加权轮询算法最为适合直播系统。

## 3、容灾策略

加权轮询算法能够将请求依据服务器的权重进行分发，但仍然会有两个问题存在，一个是进行分发的过程中如果无法感知后端服务器的运行状态，就很容易将请求转发

给一台已经不工作的机器上去，另外一个如果是 LVS 服务器宕机，那么整个系统就会处于不可用的危险状态。所以在选择了负载均衡转发模式以及负载均衡调度算法后，需要设计相应的容灾策略提升系统的可用性。

容灾部分需要考虑两方面的容灾，LVS 层面的容灾以及 RS 集群层面的容灾。LVS 的容灾将保证 LVS 负载均衡的高可用性，RS 集群的容灾将保证集群高可用性。本文对于系统容灾主要是通过 KeepAlived 来实现。

KeepAlived 是一个 TCP/IP 协议栈中第 3、4、5 层交换机制的软件<sup>[32]</sup>。它的作用是通过定期发送请求检测该服务器的状态，如果其中一台服务器出现故障或者宕机等不能正常工作的情况，KeepAlived 会将从整体系统中剔除该服务器以保证当前集群内所有的集群均为正常工作的机器，当服务器正常工作后，KeepAlived 则会自动将该服务器加入到现有的服务器群中<sup>[33]</sup>。

LVS 的容灾策略将通过主备+心跳的方式实现。主 LVS 失去心跳后，备 LVS 可以作为热备立即替换。检测心跳由 KeepAlived 完成。

RS 集群的容灾同样将通过定期健康检测实现，如果某台 RS 失去心跳，则认为其已经下线，会自动将其剔除出可用服务器列表。

#### 4.3.4 SRS 集群架构设计

在单台流媒体服务器提供服务的情况下，音视频流的输入与输出均在同一台服务器，故不会存在输入输出不同服务器的情况。当多台服务器组成集群时，由于对外暴露为一个 IP 地址，就会出现如图 4.7 所示问题。

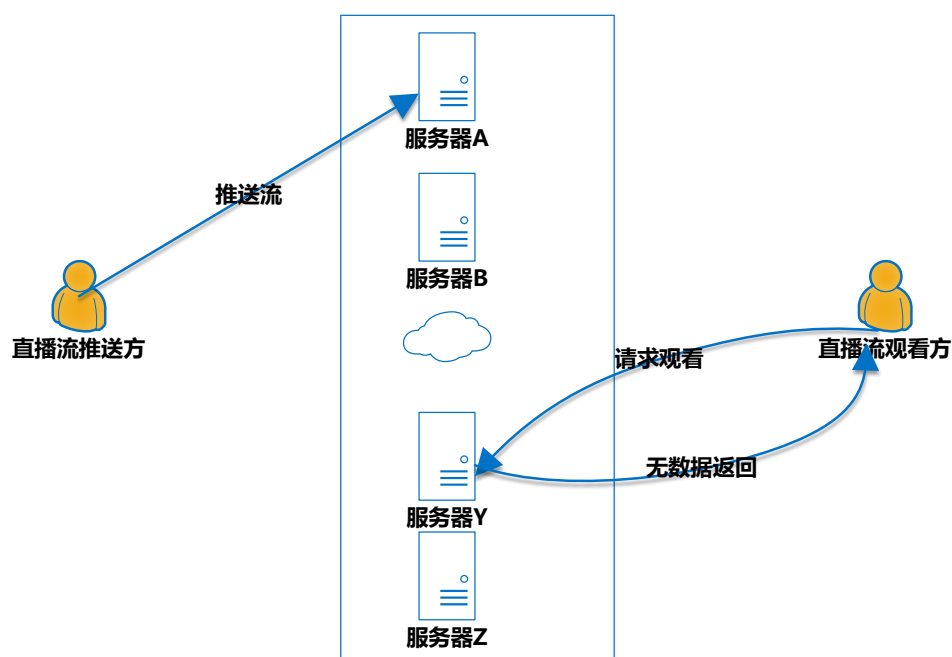


图 4.7 输入流以及输出流不在同一个服务器

集群含有  $N$  台服务器，其中直播流推送方将流推送至服务器 A，而用户去访问服务器 Y，而此时用户推送的流并没有推送到 Y 服务器上，就会发生找不到流的情况，从而无法提供服务。

SRS 支持源/边缘（origin/edge）以及转发（forward）的方式配置集群，其中源/边缘（origin/edge）的方式只在需要时方去源服务器上请求数据，而转发模式则从流启动时便开始转发至指定服务器，相比之下，源/边缘模式效率更高，对于资源占用更少，所以本文欲采用源/边缘（origin/edge）的方式配置集群，其实现原理如图 4.8 所示：

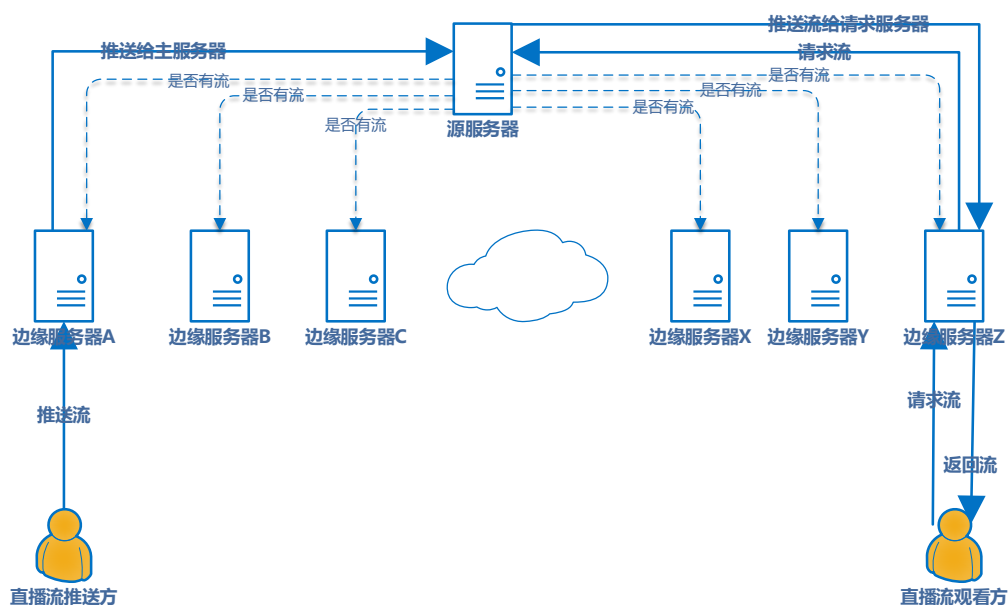


图 4.8 源站返回流

整体流程如下：

1. 整个集群中，设定一台服务器为源服务器，若干台服务器为边缘服务器，边缘服务器记录下源服务器 IP 地址，启动时向源服务器注册自己。
2. 用户向服务器 Z 请求流；
3. 边缘服务器 Z 在本机流记录中未发现该流，向源服务器发出流请求；
4. 源服务器在本机流记录中也未发现流，向所有已经在自己这里注册的边缘服务器发送请求，询问那个边缘服务器中流记录中包含所需要的流；
5. 边缘服务器 A 收到请求后在自己的流记录中发现了所请求流，发起一路流传递给源服务器；
6. 源服务器拿到流，将该流发起一路传递给边缘服务器 Z；
7. 边缘服务器 Z 拿到流之后将该流返回给用户；
8. 用户接收到流后开始观看；

通过上述的过程，就能够解决上述问题，使得流真正的分布到集群中的各台服务器



上，实现分布式流媒体集群。

上述模型较好的解决了单服务器扩展为集群时碰到的服务器间直播流不互通的问题，但还是存在一个问题，就是源服务器成为了整个集群脆弱的瓶颈。一旦源服务器宕机，整个集群就又会处于一个分散不互通的状态。

为了解决上述问题，系统引入了备份服务器，形成如图 4.9 所示结构：

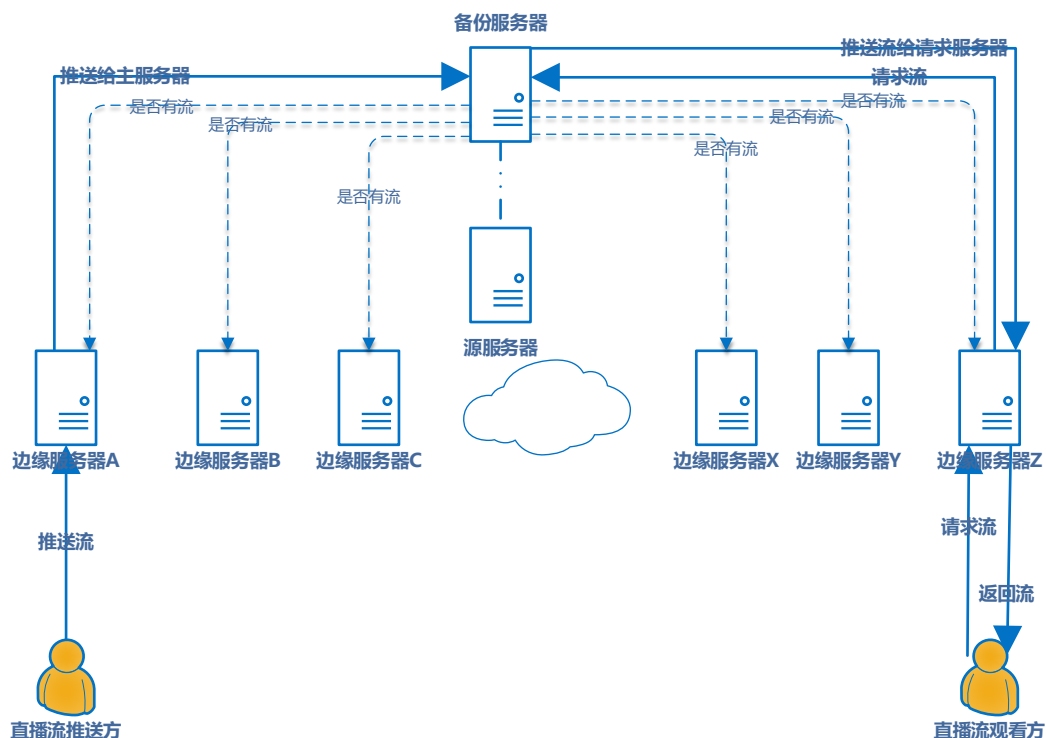


图 4.9 源站带备份服务器

上述模型为源服务器增加备份服务器一台，当源服务器宕机时，系统会无缝切换到备份服务器上，此时备份服务器会完全充当源服务器的作用，所有的流的流转均通过备份服务器，从而大大提高了系统的可用性。

### 4.3.5 防恶意穿透攻击

任何系统设计均需考虑系统的安全问题，本系统也需要考虑该问题。在分布式系统中，均存在以下一个问题，即恶意请求穿透负载均衡器直接攻击某台单台服务器<sup>[14]</sup>。

众所周知，任何域名地址均会被转化为 IP 地址，IP 地址是网络资源的一组标识。在此次设计的系统中，LVS 负载均衡层面采用了直接路由（Direct Route）模式，集群内的各台机器一旦接收到调度，就会直接为用户提供服务，不再经过调度器，所以集群内的任何一台对外提供服务的机器与数据请求方物理上互联互通，即服务器发送出去的包必须能被请求方正常接收，集群内的服务器就需要一个公网 IP。

RTMP 协议规定的默认端口号为 1935，如果不使用 1935 端口，则必须在 url 地址



中明确写明端口号。正常访问方式如图 4.10 所示：

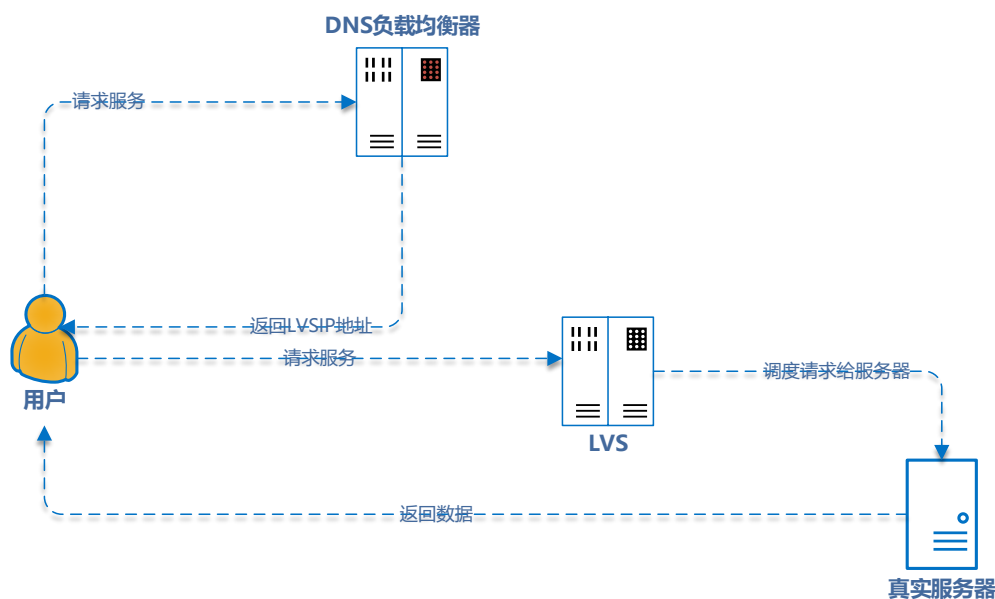


图 4.10 正常请求访问方式

如果机器的公网 IP 地址被泄露外加端口号，那么这台机器就很容易遭到攻击。即如图 4.11 描述的情况。

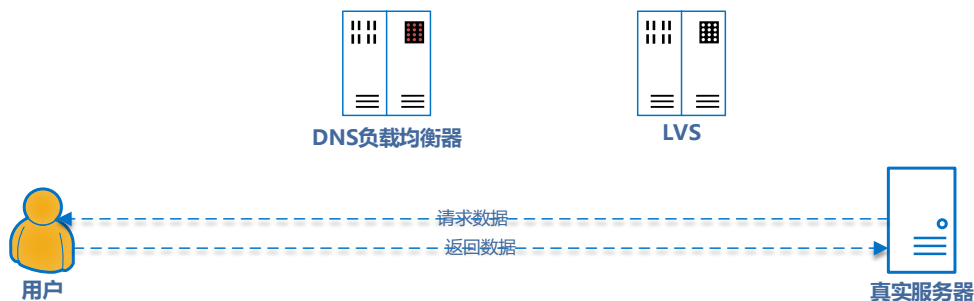


图 4.11 恶意访问

现在业界较为主流的做法是通过更改服务机器的服务端口的方式来解决该问题，即负载均衡服务器通过 A 端口接收数据，然后将数据传递到真实服务器的 B 端口上。

但上文讲到，LVS 的 DR 模式的传递原理是更改数据包的 MAC 地址为目标处理机器的 MAC 地址，数据包中的端口并未改变，所以无法做到 LVS 在 A 端口接收到数据，然后转发给真实服务器的 B 端口的效果，LVS 与真实服务器必须采取等同的端口号。这样 LVS 对外暴露的端口就是真实服务器对外暴露的端口，外界很容易探知真实服务器暴露出来的服务端口。

此次设计中，采取了在 LVS 负载均衡服务器的防火墙上做文章来解决问题，将 1935 端口在防火墙层面转发到 19350 端口，同时 LVS 监听 19350 端口，真实服务器关闭 1935

端口，监听 19350 端口。

当数据到来时，由于设置了 1935 端口转发到 19350 端口进行处理，所以数据包会被转发给 19350 端口，此时 LVS 正在监听 19350 端口，就能够拿到数据，转发给下面的而由于设置了端口转发，所以数据包会被自动转发给 19350 端口，LVS 拿到数据包后判定出需要处理的真实服务器，将 MAC 地址改为该服务器 MAC 地址后转发给该服务器。

由于真实服务器未监听 1935 端口，所以当用户通过原方式直连时，并不能够连通，从而无法进行访问以及攻击，整体过程如图 4.12 所示：

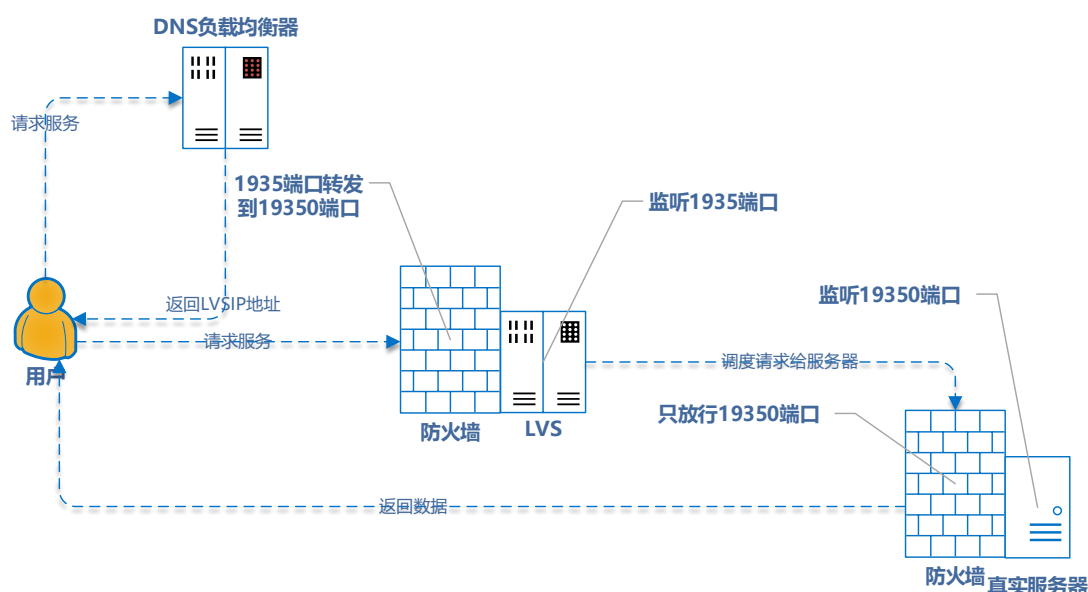


图 4.12 防止恶意穿透攻击

## 4.4 关键模块的实现

### 4.4.1 DNS 负载均衡

DNS 负载均衡采用轮询（RR）策略，请求会均匀的落到后端的服务器上。在真实场景中，在公网地址中，需要在 DNS 解析商那里配置好相应的地址，采用了简单高效的轮询策略。

### 4.4.2 LVS 负载均衡

自 Linux 2.4 内核之后，LVS 就已经成为 Linux 标准内核的一部分。所以在 Linux 2.4 版本之后的系统中并不需要安装，LVS 的内核模块名称为 ip\_vs。本次实验环境为 CentOS 7.0 版本系统，基于 Linux Version 3.10.0-123.el7.x86\_64，故内核中已经集成了 LVS，不需要重新安装。本文采用 ipvsadm 作为 LVS 的集群监测工具，KeepAlived 作

为健康监测工具。

由于采用了 DR 模式，LVS 的负载均衡部分需要做两部分的内容，一部分在于 LVS 本机上配置，另外一部分需要在真实服务器上做配置。其中在 LVS 上需要配置转发以及健康监测相关内容，在真实服务器上需要配置 ARP 抑制、虚拟 IP 等内容。

### ● LVS 端配置

LVS 端需对 KeepAlived 进行配置，其中包含了转发模型、负载均衡算法的配置，虚拟 IP、健康监测条件等。

关键部分如下：

```

vrrp_instance VI_1 {
    state MASTER    //此处指明了 LVS 是主服务器还是备服务器，主服务器
                    MASTER，而备服务器为 SLAVE
    interface enp0s3 //此处指明 LVS 所使用网卡
    virtual_router_id 101 //虚拟路由编号，主备服务器需要保持一致
    priority 200    //优先级，数值越大，优先级越高，MASTER 一定要大于 SLAVE
    advert_int 1    //主/备 LVS 检查间隔，单位为秒
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.0.201 //此处为定义的虚拟 IP (VIP)，可设置多个，对于外界来
                      说此 IP 地址方为用户看到的服务器的 IP 地址
    }
}

//定义对外提供服务器的 VIP 以及端口号，例如依照此处配置，则用户访问 IP 地址
//为 192.168.0.201，端口号为 1935 就能获得服务
virtual_server 192.168.0.201 1935 {
    delay_loop 6    //真实服务器的健康检查间隔
    lb_algo wrr     //负载均衡算法为加权轮询 (wrr)
    lb_kind DR      //负载均衡的转发模式为直接路由模式 (DR)
    persistence_timeout 14400 //超时时间，解决 tcp 连接时间过短导致的 reset 问题
    nat_mask 255.255.255.0
    persistence_timeout 0
    protocol TCP

```

```
//真实服务器的 IP 地址以及提供服务的端口号，下同
real_server 192.168.0.164 19350 {
    weight 1    //采用 WRR 算法，此处为节点权值，值越大被调度到的概率
                越高
    TCP_CHECK {    //健康检查部分
        connect_timeout 10    //超时时间，应答超过此时间则认为机器不可用
        nb_get_retry 3    //重试次数
        delay_before_retry 3    //重试前的等待时间
        connect_port 19350    //健康检查所访问的端口号
    }
}

real_server 192.168.0.165 19350 {
    weight 10    //采用 WRR 算法，此处为节点权值，值越大被调度到的概率
                越高
    TCP_CHECK {    //健康检查部分
        connect_timeout 10    //超时时间，应答超过此时间则认为机器不可用
        nb_get_retry 3    //重试次数
        delay_before_retry 3    //重试前的等待时间
        connect_port 1935    //健康检查所访问的端口号
    }
}

.....此处省略
}
```

配置完成后，运行 `ipvsadm -l` 命令，就可以看到类似结果如图 4.13 所示：

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP   realserver1:macromedia-fcs rr
  -> 192.168.0.163:macromedia-fcs Route    1      0          0
  -> 192.168.0.164:macromedia-fcs Route    1      0          0
  -> 192.168.0.165:macromedia-fcs Route    1      0          0
  -> 192.168.0.166:macromedia-fcs Route    1      0          0
```

图 4.13 LVS 配置结果图

### ● 真实服务器端配置

上文讲述过 LVS 中直接路由（DR）模式的原理，其要求所有请求均从 LVS 进，并且 LVS 调度之后，将数据包中的 MAC 地址换为所选择的真实服务器的 MAC，然后再次转发给后端服务器，后端服务器接收到数据包后，判断 IP 与本机相等且 MAC 地址与本机相等才会接受。

对于真实服务器来说，如果自己是被 LVS 选中的服务器，那么 MAC 地址必然是相等的，但 LVS 并不修改 IP 地址，所以传递过来的数据包的 IP 地址仍为 LVS 的 IP 地址，如果想要满足 IP 地址相等且 MAC 地址相等的条件，则必须使得本机也具有一个与 LVS 相等的 IP 地址。即需要本机的网卡也监听 LVS 的 IP 地址。

关键配置如下：

```
//定义 LVS 的虚拟 IP 地址
```

```
SNS_VIP=192.168.0.201
```

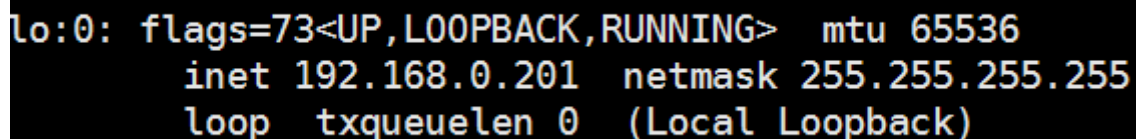
```
//将 LVS 的虚拟 IP 配置至本机的网卡上
```

```
ifconfig lo:0 $SNS_VIP netmask 255.255.255.255 broadcast $SNS_VIP
```

```
//为本机添加一个网卡
```

```
/sbin/route add -host $SNS_VIP dev lo:0
```

配置完成后，运行 ifconfig 命令，可得结果类似图 4.14 所示：



```
lo:0: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 192.168.0.201 netmask 255.255.255.255
    loop txqueuelen 0 (Local Loopback)
```

图 4.14 虚拟网卡设置结果示意图

通过以上配置，就能够使得本机监听 LVS 的 IP 地址，从而能够在得到 LVS 传递的数据包。同时，还需要设置 ARP 抑制等内容，由于此部分并非关键技术点，此处便不再展开讲述。

### 4.4.3 SRS 服务器集群

本文采用源/边缘（Origin/Edge）模式配置 SRS 集群，方式为设置其中一台服务器为源服务器，其余所有服务器为边缘服务器，边缘服务器的配置中指定源服务器提供服务的 IP 地址以及端口号，在启动时向源服务器注册自己。

关键部分配置如下：

### ● 源服务器关键配置

```
listen 29350; //服务监听端口号
```

```

max_connections    10000;    //允许最大连接数
pid                objs/origin.pid;    //pid 存储路径
srs_log_file       ./objs/origin.log;    //日志存储路径
vhost __defaultVhost__ {
    //基础配置可留空，即无条件转发，如果有低延迟等配置可添加相关逻辑至此
}

```

源服务器没有多余配置，只需要指明自己为源服务器即可，所有隶属于它服务的边缘服务器会在启动时向其注册。

### ● 边缘服务器关键配置

```

listen            19350;    //服务监听端口号
max_connections    8000;    //允许最大连接数
pid                objs/edge.pid;    //pid 存储路径
srs_log_file       ./objs/edge.log;    //日志存储路径
vhost __defaultVhost__ {
    mode            remote;    //指明运行模式为边缘
    origin           192.168.0.162:29350;    //指明源服务器服务的 IP 地址以及端口号
    //基础内容依旧留空，如果有低延迟等配置可添加相关逻辑至此
}

```

边缘服务器需要标明自己为边缘服务器，并且设置在没有流存在时为其服务的源服务器 IP 地址以及端口号是哪一个，以便在启动时注册以及找不到直播流时能够去申请该流。别的配置如低延迟等配置与常规服务器相同。

### ● 低延迟配置

```

listen            19350;    //服务监听端口号
max_connections    8000;    //允许最大连接数
vhost __defaultVhost__ {
    gop_cache       0.5;    //视频中两个关键 I 帧的时间间隔，0.5 表明每 0.5s 向 flash 播放器发送一个 I 帧
    queue_length     10;    //缓冲区长度，越小延迟越低
    min_latency      on;    //低延迟是否开启
    tcp_nodelay      on;    //TCP 包接收到立即转发
}

```

低延迟设置中需要调低 GOP-Cache，GOP 是视频流中两个 I 帧的时间距离，Flash 的解码器拿到 GOP 才会开始进行解码以及播放。即服务器一般会先给 Flash 播放器一

个 I 帧。假设 GOP 是 15 秒，也就是每隔 15 秒才传递一个关键帧，如果用户在第 5s 时到来，那么就会产生 10s 的黑屏。本文将 GOP 改为 0.5s，大大降低第一帧的时间，使得用户能够更快的看到画面。

### 4.5 本章小结

上一章对于整体系统的需求做了梳理以及分析，本章主要在需求已经梳理清晰的情况下，对直播流媒体系统进行模块设计以及实现。

首先在第一节，本文借鉴业界现有的一些非针对直播系统的解决方案，设计出一套行之有效的解决思路。本章的余下内容均依据提出的解决思路展开，进行细化设计，最终解决提出的问题。

第二节通过对比现有市场上的各种 RTMP 服务器，经过分析对比，选出一款最适合直播且最利于扩展的基准流媒体服务器，其将承担终端服务用户的任务。

第三节进行了详细的系统设计，包含整体架构设计、DNS 集群架构设计、LVS 集群架构设计、SRS 集群架构设计以及防恶意穿透攻击等五部分，是整个系统的主题解决方案。

第四节针对上一节提出的解决方案进行了实现，列出了其中关键模块的实现方式。

## 第五章 可视化系统的设计与实现

通过第三章第三节对于可视化系统的需求分析,已经明确所需实现的内容,本章将在需求明确的基础上进行设计并进行实现。

### 5.1 系统整体架构

根据需求分析和系统分析,采用 Spring MVC+Spring+MyBatis+Quartz 的传统 Web 架构,对可视化系统进行的架构模型的分析与设计。如图 5.1 所示:

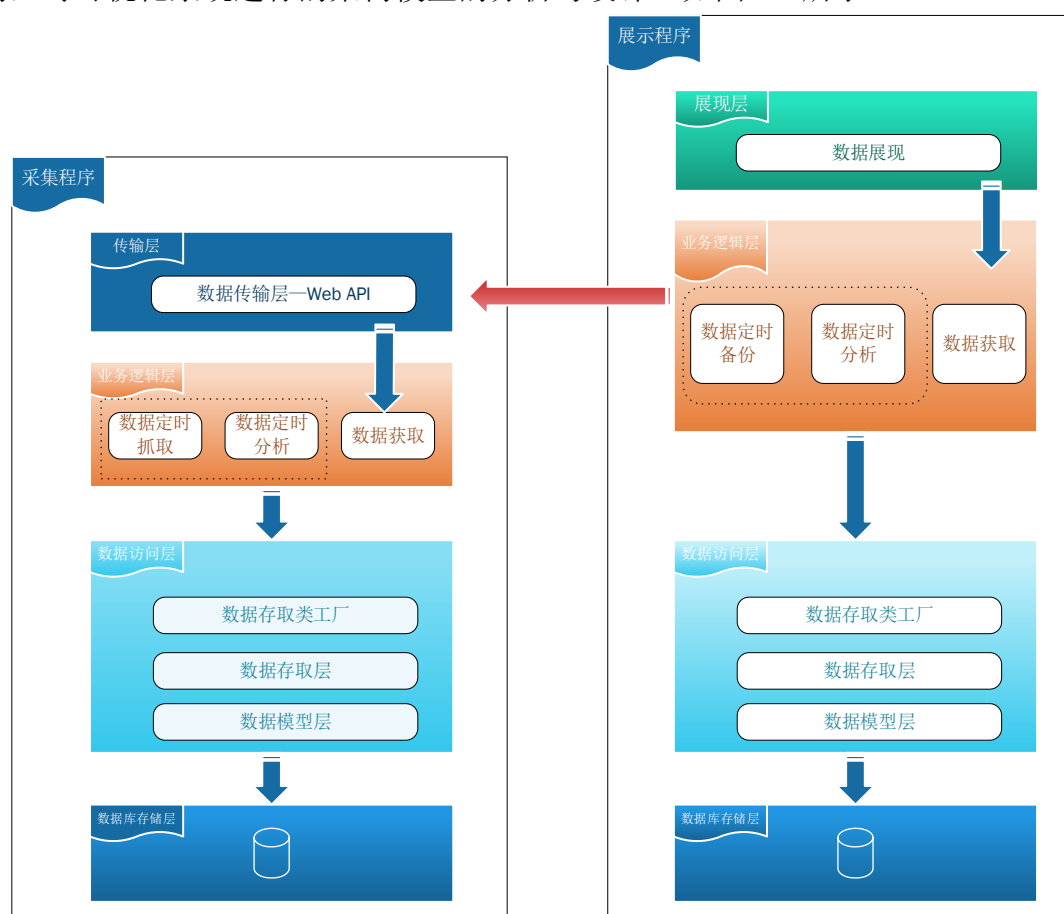


图 5.1 可视化系统整体模型图

系统总共包含采集程序 Monitor 及展示程序 Control 两个程序。

由于采集程序不涉及数据展示,故采集程序采用业务逻辑层、数据访问层两层架构;展示程序采用表现层、业务逻辑层、数据访问层的三层软件设计架构。二者之间并通过基于 HTTP 的 Web API 进行数据传递。符合软件设置的“高内聚,低耦合”的程序设计原则。



整体系统包含数据采集模块、数据同步模块、数据处理模块、定时任务模块、数据展现模块等模块。

数据采集模块将对于数据进行机器的数据进行实时抓取并进行持久化。

数据同步模块将每台真实服务器的数据统一汇合只 Web 服务器上并进行备份数据的持久化。

数据处理模块则会分为两部分，一部分为采集程序的数据处理，其将依据一定的时间间隔算出本机的在这段间隔内的平均负载；另外一部分则为展示程序的数据处理，在数据同步模块中，数据从真实服务器的采集程序中同步至 Web 服务器上的展示程序中，此时在展示程序中，拥有的是各台机器的本机数据，还需要进行处理才能够形成集群的数据，采集的处理模块则将综合处理所有的单机数据，将其整合处理为集群数据给展示模块使用。

数据展示模块则实现了最终呈现给用户的所有功能，其将集群以及单机的数据通过前端技术友好的展现给用户。

## 5.2 数据库存储设计

在对系统业务概念、业务概念之间的关系与系统的业务逻辑进行充分理解的基础上，设计系统的数据库存储方案。

系统分为采集程序以及展示程序，理论上采集程序的功能仅应局限于数据的采集，采集获得的数据实时推送至展示程序所在数据库中，但由于系统数据的非常规性，实时采集的数据会非常庞大，以 5s 做一次采集为例，一台服务器一天就会产生 17280 条数据，那么假设一个集群仅拥有 100 台 RTMP 服务器，那么一天的数据量就会是 1728000 条，而且此时不论对于系统做分析还是提取均会造成比较大的问题。所以采取了双数据库的方式，即采集程序拥有自己本地的数据库，用于存放本机采集数据，展示程序也拥有自己的数据库，用于存放集群内机器配置以及集群运行数据等数据。

### 5.2.1 采集程序数据库设计

数据库的设计基于上层服务所需，采集程序主要负责本机数据的存储，除此之外不存储任何数据。实时数据欲 5s 采集一次，以此数据作为基准数据。其后系统会每 1 分钟、5 分钟、10 分钟、1 小时、1 天、1 周做一次平均计算，得到该间隔内的平均数据，所以具体存储模型如图 5.2 所示：



图 5.2 采集程序数据库设计

共分为两种类型的表：

- 实时数据表  
存储实时 CPU、内存、出网流量、入网流量、RTMP 连接数等数据
- 分析所得数据表  
存储分析所得 CPU、内存、出网流量、入网流量、RTMP 连接数等数据，同时需要记录该条记录由多少条记录整合而成，以及该条记录承载那段时间的信息。

### 5.2.2 展示程序数据库设计

展示程序用于集群数据整合以及数据展示，可视化程序的主体在展示程序。数据库层面需机器信息、存储集群数据、真实服务器备份数据三类数据。。具体存储模型如图 5.3 所示：

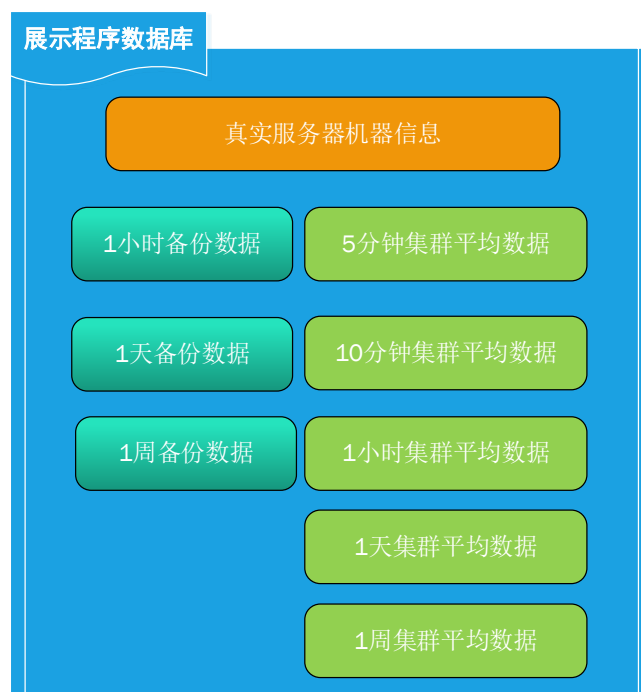


图 5.3 展示数据数据库

共分为三类数据：

- 服务器机器信息

机器信息源于机器注册时，真实服务器传递，然后所进行的持久化形成。由于机器可能存在多 CPU、网卡、内存等，所以机器信息应当拆成 4 部分存储，一张用于存放基本信息，一张存放 cpu 信息，一张存放内存信息，一张存放网卡信息，如图 5.4 所示：

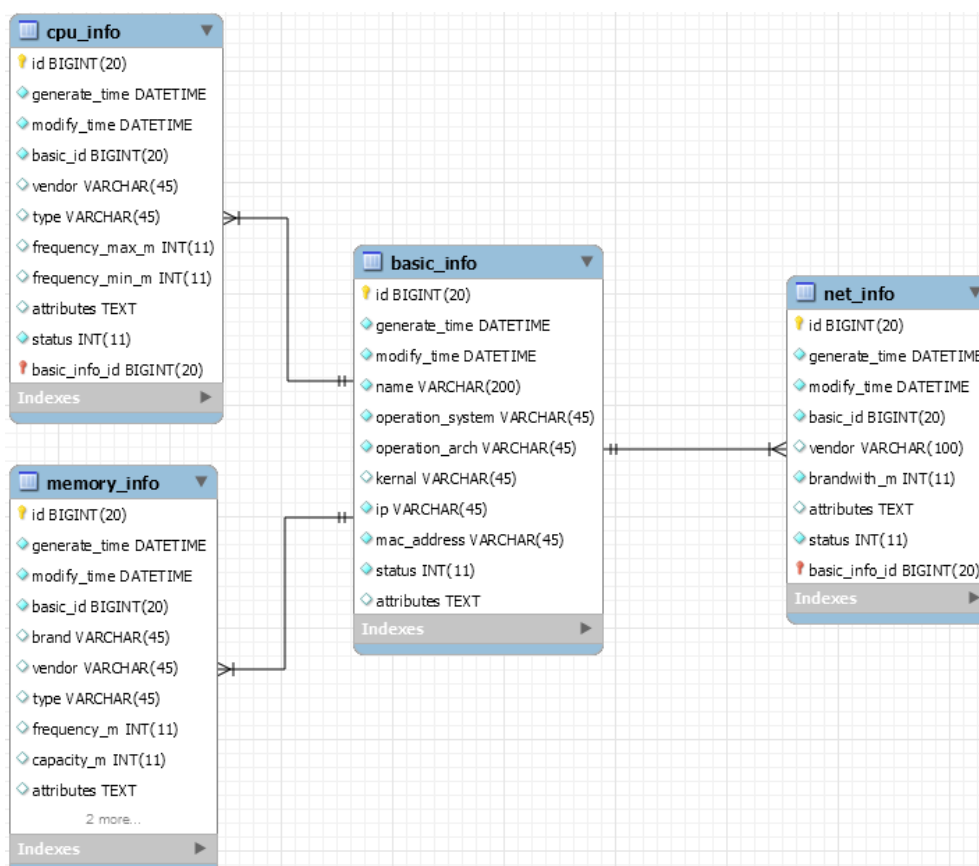


图 5.4 机器信息

- 备份数据  
备份数据来源于展示程序定时向各台真实服务器请求所得数据的备份。
- 集群平均数据  
集群数据来源于各台真实服务器负载的均值

## 5.3 主要模块设计与实现

### 5.3.1 数据采集模块设计与实现

数据采集模块仅存在于采集程序中，其每通过定时任务，每 5s 进行一次原始数据采集。采集之后，通过 MyBatis 访问数据库，将其存入数据库中，供后续使用。同时，由于数据采集会产生大量原始数据，原始数据在一段时间内可为数据分析使用，超过一定时间限制，就不会再被关心。同时由于其占用了大量的数据资源，每天会产生约 1 万 7 千余条数据，如果长时间不清理，则会使得数据库不堪重负，所以需要定期对数据库进行定期清理。模块整体运行时序如图 5.5 所示：

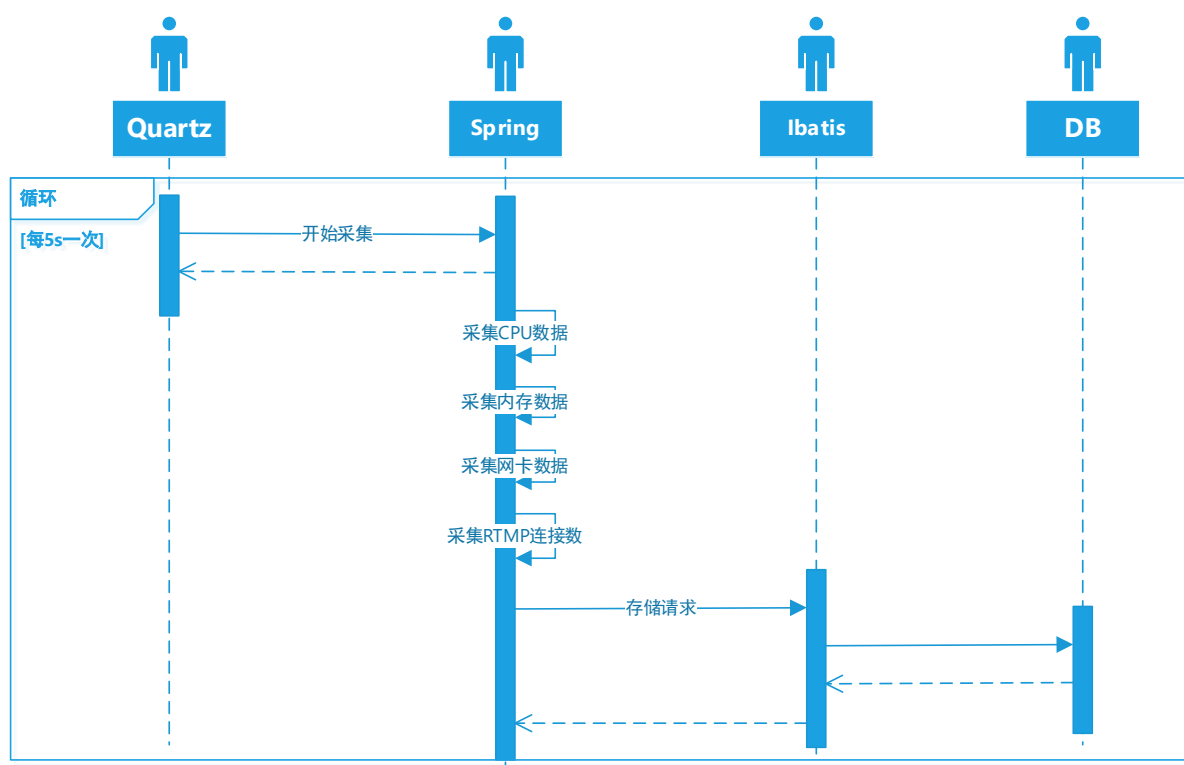


图 5.5 采集时序图

整体运行流程如下：

1. 定时器定时发采集请求
2. Spring 先后采集出 CPU 使用数据、内存使用数据、网卡数据、RTMP 连接数数据
3. 对所取出的数据求均值
4. 通过 MyBatis 存入数据库中

定时程序采用 Quartz，设置为每 5s 运行一次；采集 CPU、内存、网卡等信息使用 Sigar 进行，采集 RTMP 请求通过 Java 的输入输出流执行 Linux 命令。

### 5.3.2 数据处理模块设计与实现

数据处理模块需要分为采集端的数据处理以及展现端的数据处理两部分。

#### ● 采集端数据处理

在数据采集程序中，数据每 5s 做一次采集，能够反应出机器的实时运转情况。但在实际的可视化系统中，运维人员可能需要多更综合的数据，如一个小时中每分钟的负载变化，每五分钟的数据变化，此时就需要对原始的实时数据进行分析处理求出均值。采集程序的数据处理时序图如图 5.6 所示：

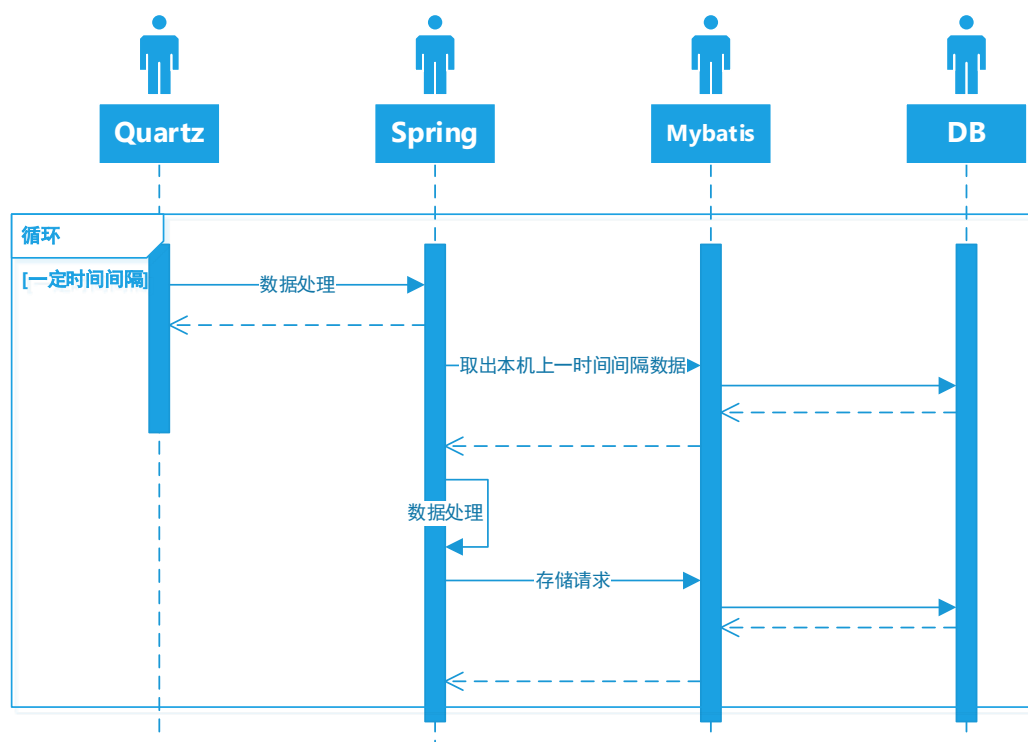


图 5.6 采集端数据处理

整体运行流程如下：

1. 定时器定时发计算请求
2. Spring 接到后首先取出上一个时间段的所有数据，比如说本次为计算 5 分钟间隔的数据，那么上一个时间段便是 1 分钟，则取出 5 分钟内的五条 1 分钟数据
3. 对所取出的数据求均值
4. 通过 MyBatis 存入数据库中

#### ● 展现端数据处理

对于集群来说，其并不像单台服务器那样，数据采集便能够进行展现，集群的实时数据会不停的扫描各台机器，会对集群内的真实服务器产生较大压力，所以在业界，运维人员并不需要每一时刻的实时数据，他们所需要关心的数据是一定时间内集群的运行情况，主要是看出集群的负载变化情况等，以便于其做出增减机器等的判断，所以展现端并不坐实时数据的整合，所做的是一个时间段内的集群负载变化的展现。

其运行时序图如图 5.7 所示：

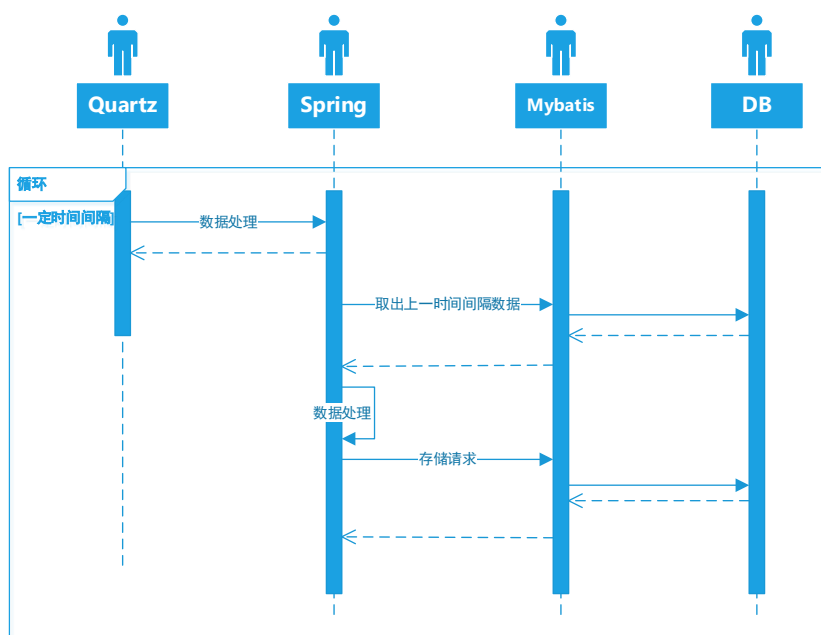


图 5.7 展现端数据处理

整体运行流程如下：

1. 定时器定时发计算请求
2. 集群的同步模块会每隔 1 分钟同步各台机器的数据放至数据库中，作为基准数据供处理
3. **Spring** 接到后首先取出上一个时间段的所有数据，此时数据库中一定已经有了计算好的数据，假设为求 5 分钟间隔的数据，那么取出上一个时间间隔也就是 1 分钟的数据
4. 对所获得的数据求均值
5. 通过 **MyBatis** 存入数据库中

### 5.3.3 数据同步模块设计与实现

采集端与展现端的数据交互靠同步模块实现。数据同步模块采用基于 **HTTP** 协议的 **Web API** 来实现。整体运行时序如图 5.8 所示：

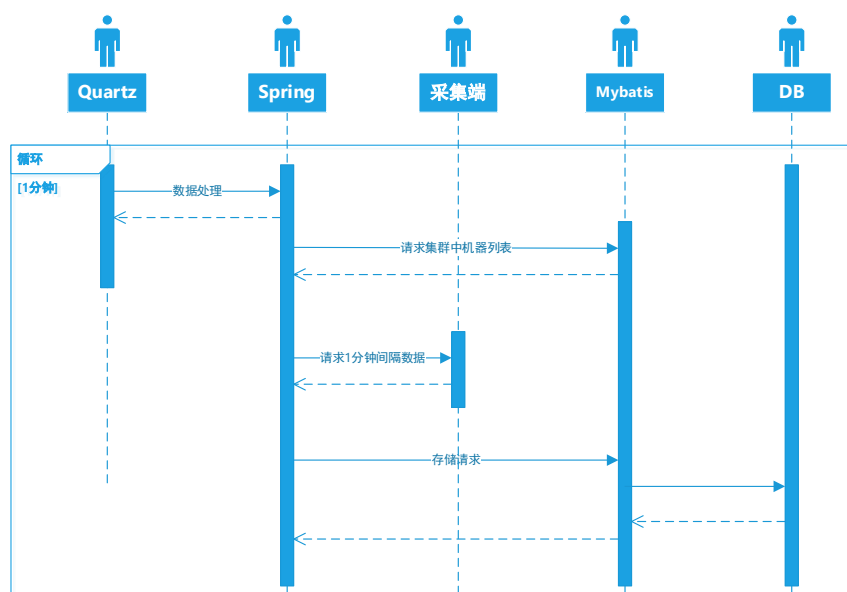


图 5.8 数据同步时序图

整体运行流程如下：

1. 展现端定时器定时发数据同步请求
2. Spring 接到后从数据库中取出集群的所有有效机器
3. 通过循环的方式向每一台机器发送数据备份请求
4. 每一台真实服务器接收到请求后会将 1 分钟间隔的数据发送给请求方
5. 展现端拿到数据后通过 MyBatis 存入数据库

### 5.3.4 数据展现模块设计与实现

数据展现模块是呈现给用户所需数据的模块，作为最上层模块，上述数据采集模块、数据处理模块、数据同步模块均为其服务，其承载着终端数据可视化的功能。本模块采用后端 Spring MVC 作为展现层框架获取数据，前端采用 Bootstrap 作为页面构建框架，通过 HighCharts 将数据展现出来。

模块数据分为两大类：集群数据以及单机数据。上文数据库设计模块详细设计了数据的存储情况，集群数据主要源于数据库，但仍有部分数据，如 RTMP 连接数等实时性要求稍高的数据源于定时获取。而单机数据则稍有不同，由于考虑到存储成本的问题，备份过来的数据相对轻量级，只备份了以天为单位的数据，所以像分钟、小时力度的数据仍需要去真实服务器请求。

#### ● 集群数据展示部分：

集群数据展现时序图如图 5.9 所示：



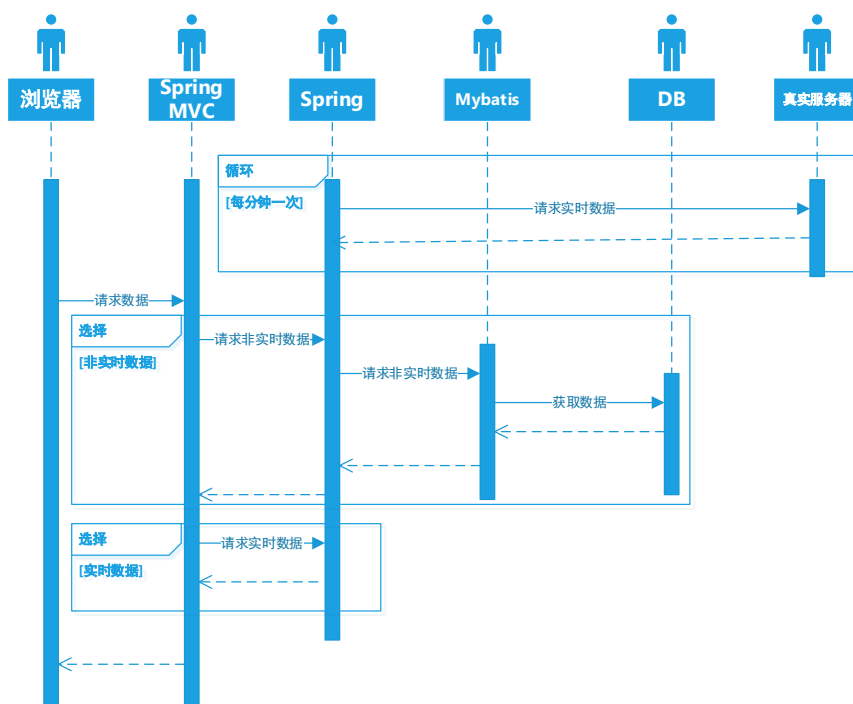


图 5.9 集群数据展现时序图

整体运行流程如下：

1. 展示程序每分钟向各台真实服务器发请求，请求一次系统所需数据，如当前 RTMP 连接数量等
2. 浏览器发出集群数据展示请求
3. Spring MVC 向 Spring 申请数据
4. Spring 接收到请求后需要先判断请求类型，如果是非实时请求，Spring 通过 MyBatis 从数据库中取出数据返回给 Spring MVC
5. 如果是实时请求，此时 Spring 将从内存中取出数据返回给 Spring MVC
6. Spring MVC 返回给浏览器，浏览器渲染展示

#### ● 单机数据展示部分

单机数据展现时序图如图 5.10 所示：

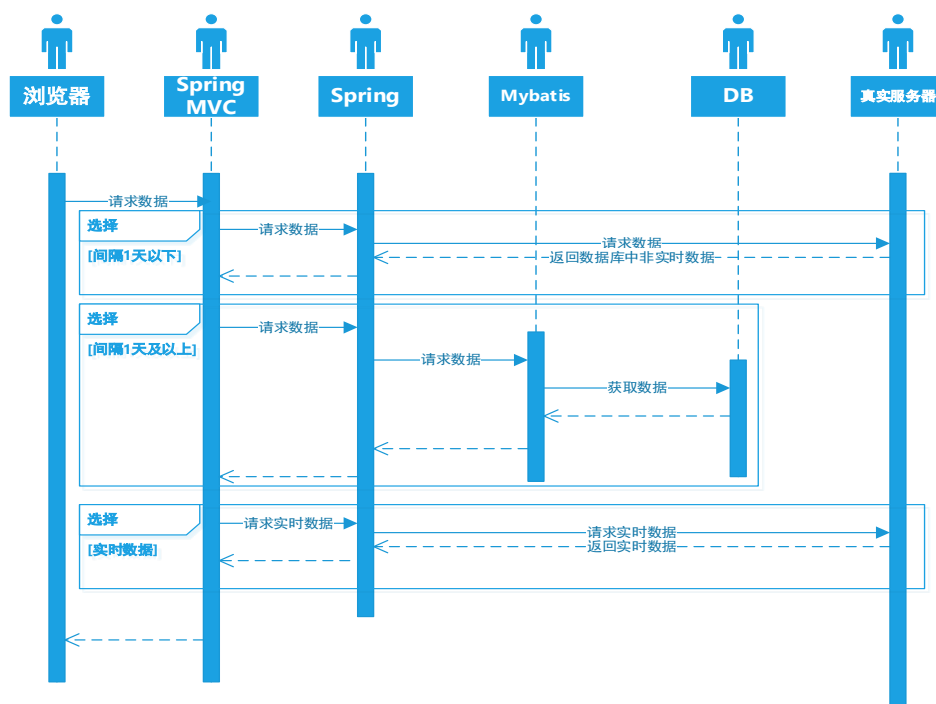


图 5.10 单机数据展现时序图

整体运行流程如下：

1. 浏览器发出集群数据展示请求
2. Spring MVC 判断请求是实时数据请求还是非实时数据请求，并向 Spring 申请数据
3. 如果是非实时请求，Spring 还需判断其所求数据的时间间隔是否小于 1 天，如果小于 1 天，如 5 分钟、10 分钟等，则 Spring 需要向真实服务器发送请求，真实服务器从自己的数据库中取出相应数据返回给展现端 Spring，该 Spring 返回给 Spring MVC
4. 如果是非实时请求且间隔大于 1 天，则 Spring 通过 MyBatis 从数据库中取出备份的数据返回给 Spring MVC
5. Spring MVC 返回给浏览器，浏览器渲染展示

## 5.4 本章小结

第三章对于整体系统的需求做了梳理以及分析，本章主要在需求已经梳理清晰的情况下，对可视化系统进行模块设计以及实现。

首先在第一节，本文描述了可视化系统的设计整体架构，描述了如何能够最终实现前文提到的各类需求。

第二节详细介绍了可视化系统的数据库存储的设计，分为采集端数据库以及展现端

数据库两部分，并讲解了采用这类设计的原因及优点。

第三节讲解了系统的数据采集模块、数据处理模块、数据同步模块以及数据展现模块这四个主要模块的设计以及实现过程。

## 第六章 系统功能验证与性能测试

通过上述章节，本文详细阐述了基于 RTMP 协议的高可用直播流媒体集群的原理以及搭建过程，同时亦设计并构建了一个简单的集群可视化系统。本章将对直播流媒体系统及可视化系统进行集成测试。

### 6.1 测试策略选择

在集成测试的过程中，由于直播流媒体系统需要测试其可用性及鲁棒性，故将进行功能测试以及性能测试来验证效果；可视化系统相对较为简单，受众为运维人员，无高并发等场景，故只进行功能测试。

#### 6.1.1 流媒体系统测试策略

整个流媒体集群由三大部分组成：DNS 负载均衡部分、LVS 负载均衡部分以及 SRS 集群部分。其中 DNS 负载均衡部分由于实验条件限制，不做实现故不进行测试。

针对 LVS 负载均衡部分以及 SRS 集群部分，各个模块功能的正常运转是整体集群正常运转的必要条件，所以应当对各个模块进行相关功能测试。故设计了针对 SRS 单服务器、SRS 集群、LVS 部分的功能测试，检测各个模块的功能运转情况。

在完成了各个模块的测试后，应当对集群进行整合，验证整合后的功能。所以针对整体流媒体集群，进行了集群层面的功能测试，用于验证整体集群的功能运转情况。集群整体功能测试中，主要设计了直播流的播放转发功能以及提供服务的服务器突然中断服务的情况下，服务切换的功能测试。

集群在设计层面解决了单机无法解决的高并发问题，针对这一特性，设计了性能测试来验证高并发请求访问情况下集群的可用性。

#### 6.1.1 可视化系统测试策略

可视化系统针对运维人员设计，整体系统并无高并发，安全侵入等问题，故只需进行功能测试，验证所设计功能是否符合预期。

### 6.2 部署及测试环境

#### 6.2.1 直播流媒体系统

- 部署环境

本文将采用 1 台 LVS 主服务器、1 台 LVS 备服务器、1 台 SRS 主服务器、1 台 SRS 备服务器、4 台 SRS 边缘服务器共计 8 台服务器搭建流媒体环境。

8 台服务器均通过 Oracle VM VirtualBox 5.0.10 r104061 虚拟化得来，宿主机具体配置如表 6.1 下：

表 6.1 宿主机配置

CPU	主板	内存	硬盘	网卡	操作系统
Core i7-4710MQ 2.50GHz 四核	联想  20ANCT01WW	16GB	128GB	英特尔 Ethernet Connection  I217-LM	Windows 10  专业版 64 位

8 台虚拟机配置如表 6.2 所示：

表 6.2 集群服务器配置

<b>LVS 服务器-主 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址：192.168.0.160 虚拟 IP：192.168.0.200
<b>LVS 服务器-备 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址：192.168.0.161 虚拟 IP：192.168.0.200
<b>SRS 源服务器-主 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址：192.168.0.162
<b>SRS 源服务器-备 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址：192.168.0.163 回环 IP：192.168.0.200
<b>SRS 边缘服务器 × 4</b>	单核处理器 + 512MB 内存 CentOS 7.0 IP 地址：192.168.0.164-192.168.0.167 回环 IP：192.168.0.200

### ● 测试环境

流媒体系统的测试将采用另外一台机器进行测试，测试机配置如表 6.3 下：

表 6.3 流媒体服务器测试用机

处理器	英特尔 奔腾双核 T4400 @2.2GHz
主板芯片组	英特尔 PM45
内存	4GB (金士顿 DDR3 1333MHz)
网卡	1000Mbps 以太网卡
操作系统	CentOS 7.0
IP 地址	192.168.0.103

### 6.2.2 可视化系统

#### ● 部署环境

在前文设计部分提到,可视化系统分为采集端以及展现端两个程序,采集端程序部署在流媒体集群的各台真实服务器上,展现端程序部署在单独的 Web 服务器上,故可视化系统共采用 7 台虚拟机进行测试,其中 6 台机器为直播流媒体系统所提到的机器,分别为 2 台 SRS 源服务器以及 4 台 SRS 边缘服务器,第 7 台机器为单独的 Web 服务器,配置如表 6.4 下:

表 6.4 可视化系统部署服务器配置

<b>SRS 源服务器-主 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址: 192.168.0.162
<b>SRS 源服务器-备 × 1</b>	单核处理器 + 1GB 内存 CentOS 7.0 IP 地址: 192.168.0.163
<b>SRS 边缘服务器 × 4</b>	单核处理器 + 512MB 内存 CentOS 7.0 IP 地址: 192.168.0.164-192.168.0.167
<b>Web 服务器 × 1</b>	单核处理器 + 512MB 内存 CentOS 7.0 IP 地址: 192.168.0.168

#### ● 测试环境

可视化系统功能相对较为简单,所以仅仅进行了黑盒功能测试。测试机采用开发机本机,配置如上表 6.1 所示。

## 6.3 流媒体集群测试

### 6.3.1 SRS 单服务器测试

SRS 的单服务器为检验 RTMP 播放器的直播功能。

测试将采用 IP 地址为 192.168.0.164 的机器来进行，为了简单直观，流推送软件采用 OBS（Open BroadCast Software）进行测试，由于不启用 LVS 负载均衡功能，故不需要 Virtual IP，测试方式为，使用 OBS 采集电脑桌面信息以及电脑摄像头信息上传至 SRS 服务器，然后通过 VLC Media Player 进行网络串流播放。效果如图 6.1 所示：

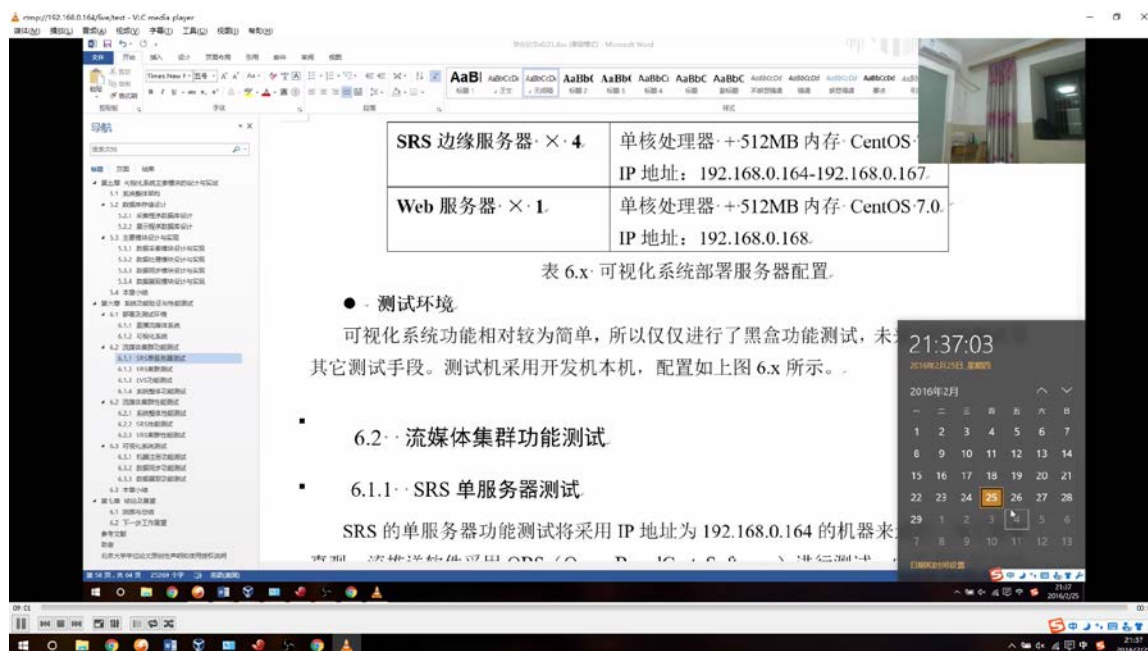


图 6.1 SRS 单服务器功能测试图

### 6.3.2 SRS 集群测试

SRS 集群测试目的为检验当直播流只输入至 A 服务器，而请求落在 B 服务器时能否能够得到直播流。

测试将启用一台 SRS 源服务器以及两台 SRS 边缘服务器来进行。SRS 源服务器 IP 地址为 192.168.0.162，两台 SRS 边缘服务器 IP 地址分别为 192.168.0.165 以及 192.168.0.166。

由于暂不涉及性能测试部分，为简单直观，依旧采用 OBS 进行数据采集，OBS 所在机器地址为 192.168.0.101。OBS 此次上传至 IP 地址为 192.168.0.155 的 RTMP 服务器上，VLC Media Player 将访问 192.168.0.166 进行网络串流播放。结果如图 6.2：

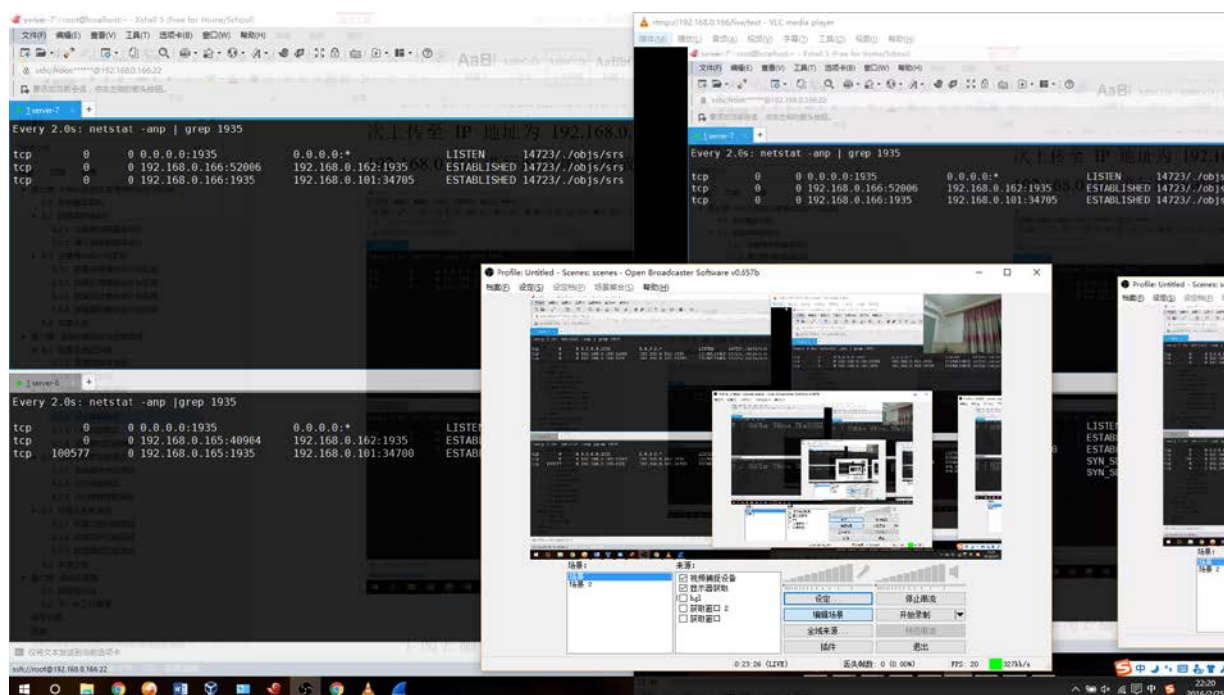


图 6.2 SRS 集群测试结果

上图左侧为 XShell 连接至两台边缘服务器，通过 `watch 'netstat -anp | grep 1935'` 命令监视 1935 端口的连接情况，中间为 OBS 进行数据采集并实时上传，右侧为 VLC Media Player 进行实时播放。从左侧图可以看出，两台边缘服务器均与 192.168.0.101 建立了连接，并且只建立了一个连接，排除了发布流以及接收流在同一台 RTMP 服务器上的可能性。同时可以看出每台边缘服务器均与 IP 地址为 192.168.0.162 的 SRS 源服务器建立了连接，也进一步说明了 192.168.0.166 的请求没有找到流，向源服务器申请，源服务器发现在 192.168.0.165 这台机器上有流，从而进行了一个转发。中间可以看出 OBS 右下角的上传标志为绿色，表明在实时上传，与此同时右侧的 VLC 也已经显示出了画面，说明直播流进入了一台边缘服务器，而请求落在另外一台边缘服务器上依然可以正常观看，达到了 SRS 集群搭建的目的。

### 6.3.3 LVS 功能测试

LVS 功能测试的目的在于检验当请求只发往 LVS 服务器，能否正常推送以及观看直播流。

测试方式将启用一台 LVS 服务器、一台 SRS 源服务器、3 台 SRS 边缘服务器来进行。LVS 服务器的 IP 地址为 192.168.0.160，VIP 为 192.168.0.201，SRS 源服务器地址为 192.168.0.162，4 台 SRS 边缘服务器 IP 地址分别为 192.168.0.164 -- 192.168.0.167。

同样为了直观，依旧采用 OBS 进行数据采集，发送至 192.168.0.200 这个虚拟 IP，LVS 服务器上并没有 SRS 服务器，并不能够直接提供服务。VLC Media Player 也向



192.168.0.200 这个 IP 地址请求服务。运行结果如图 6.3(a)、6.3(b)、6.3(c)所示:

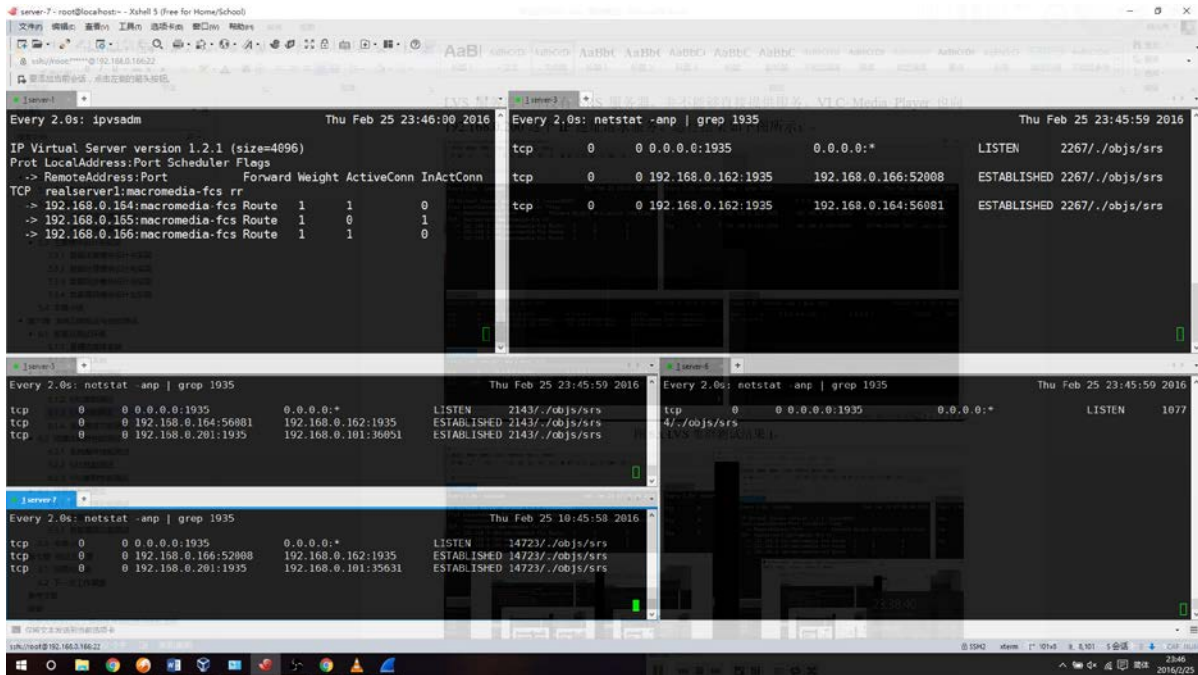


图 6.3(a) LVS 集群测试结果展示

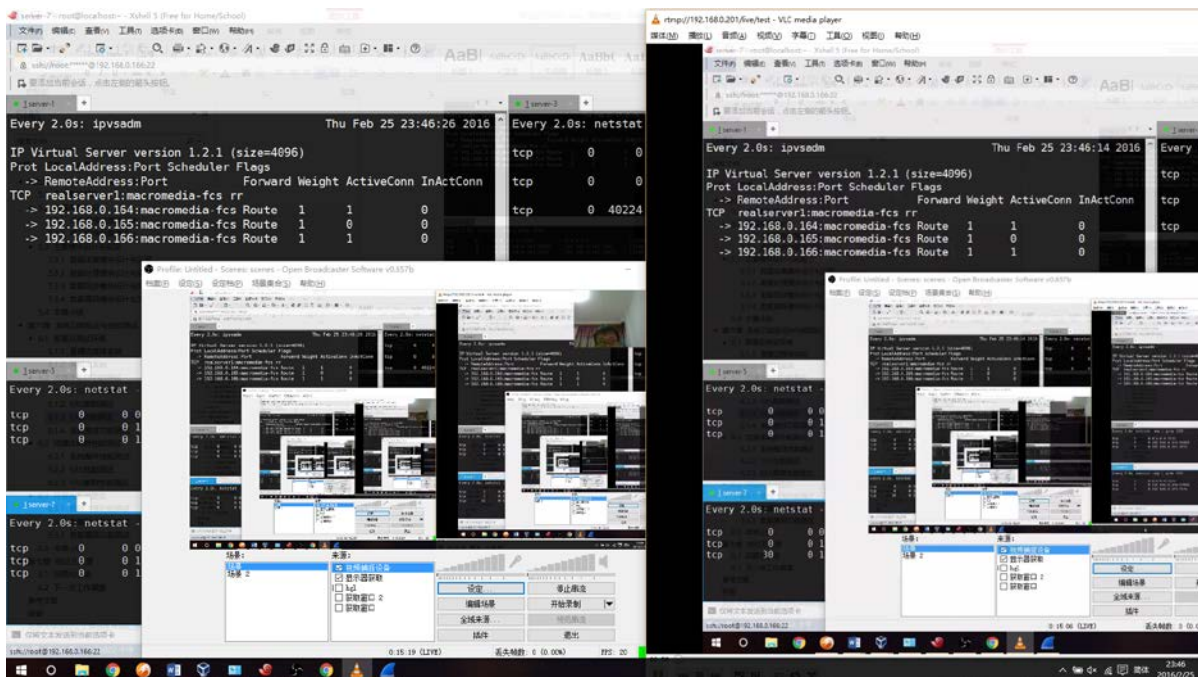


图 6.3(b) LVS 集群测试结果展示图

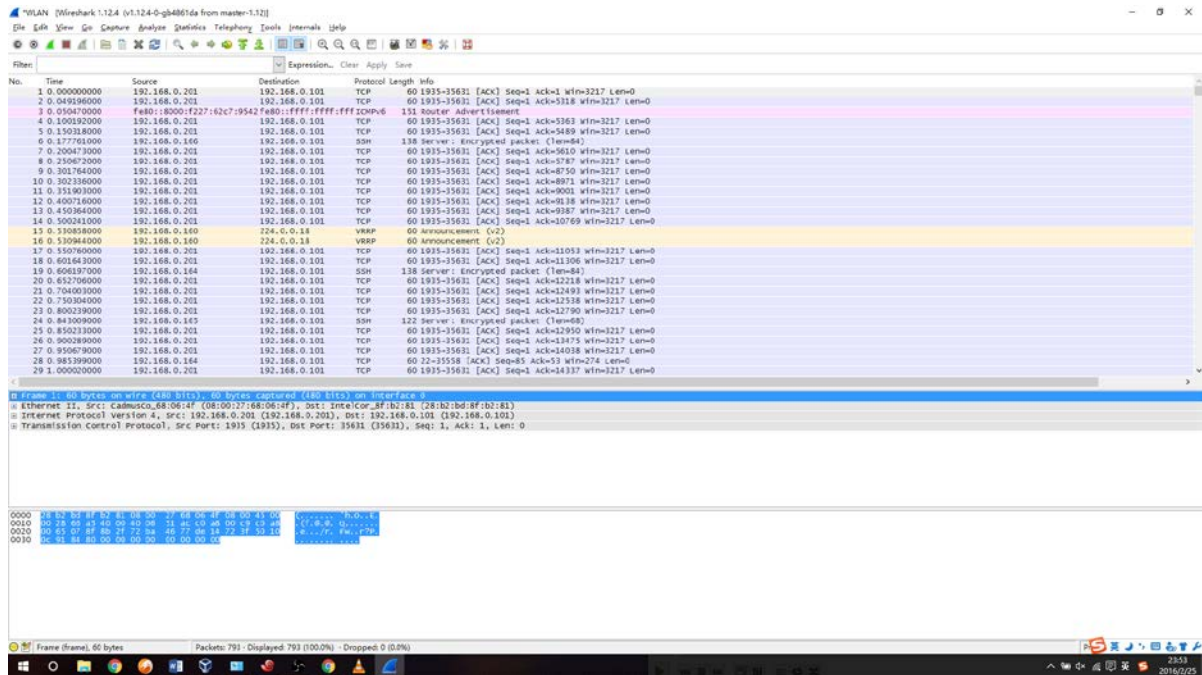


图 6.3(c) LVS 集群测试结果展示图

从测试结果 6.3(a)可以看出, OBS 指向 192.168.0.201 的流推送请求被导向到了服务器 192.168.0.166, 而 VLC Media Player 所发送至 192.168.0.201 的请求被导向到了服务器 192.168.0.164, 另外一台服务器暂时处于暂停服务状态。

从测试结果图 6.3(b)可以看出, OBS 在正常运行, 不停的发送请求, 而 VLC Media Player 从 192.168.0.201 所请求的数据亦正常显示, 图中由于服务器转发能力以及 VLC Media Player 接收流策略的问题, 产生了较大的延迟, 正常延迟应当保持在 3s 以内。

测试结果图 6.3(c)是使用抓包工具 Wireshark 1.12.4 进行的实时流数据包抓取的结果。从图中可以看到, 所有的数据包均是 192.168.0.201 与 192.168.0.101 之间的数据传递, 并没有向外暴露出真实服务器的地址, 也在一定程度上保证了系统的安全。

### 6.3.4 集群功能测试

系统整体测试将启用所有服务器, 包括 1 台 LVS 主服务器, 1 台 LVS 备服务器, 1 台 SRS 源服务器, 1 台 SRS 备服务器以及 4 台 SRS 边缘服务器。

由于上文已经对直播的主体功能进行了验证, 此处便不再验证直播功能, 仅仅验证系统的可用性, 即 LVS 的主备切换, 提升系统可用性部分, SRS 主备效果相同, 本文不再做展现。

测试采用的方突然间重启 192.168.0.160 的方式, 重启后由于 192.168.0.161 已经被设置为备份服务器, 故会接手充当新的负载均衡器, 当 192.168.0.160 重新启动起来后, 由于其权重较高, 会重新交给 192.168.0.160 进行负载均衡。结果如图 6.4 所示:

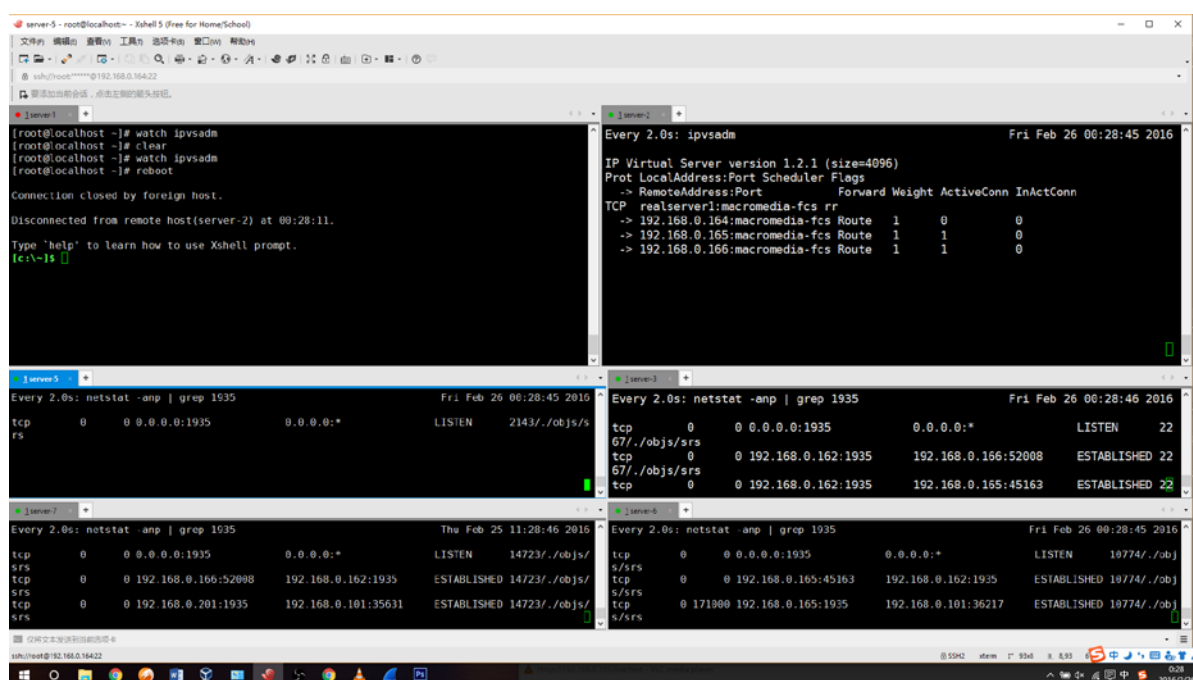


图 6.4 LVS 主备验证测试结果

在突然重启时，视频看的大概 1s 多不到 2s 的卡顿，就能够继续正常观看，达到了实验目的。

同时 DNS 的负载均衡采用了效率最高也最为简单的轮询 RR 策略，效果即为每来一个请求便换一个 VIP 转发请求，相对较为简单，此处省略验证过程。

通过上述功能验证，可以确保本系统的高可用性，不论在任何机器宕机的情况下，都不会影响系统的正常运转，实现了系统的鲁棒性。

### 6.3.5 集群性能测试

整体系统的压力测试将启用测试机、主 LVS 服务器、SRS 源服务器-主、三台 SRS 边缘服务器来进行。测试方式为通过 OBS 采集数据上传至 192.168.0.201，然后测试机使用 st-load 工具模拟 RTMP 请求，通过请求数逐级增加的方式进行，看不同请求数下，各台机器的负载情况变化。测试时所有机器均关闭 Linux 系统 1024 个文件的限制。

经过多次实验，本文取出其中最具代表性的一次实现结果，实验结果如图 6.5(a)、6.5(b)、6.5(c)、6.5(d)所示：



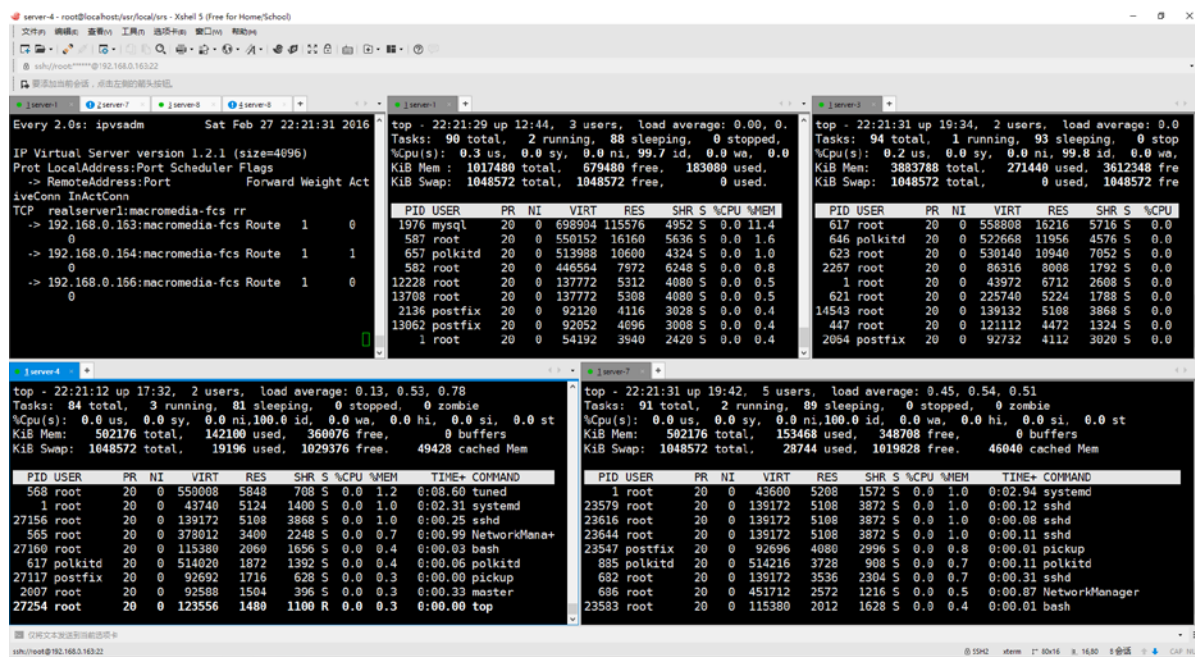


图 6.5(a) 测试前机器情况

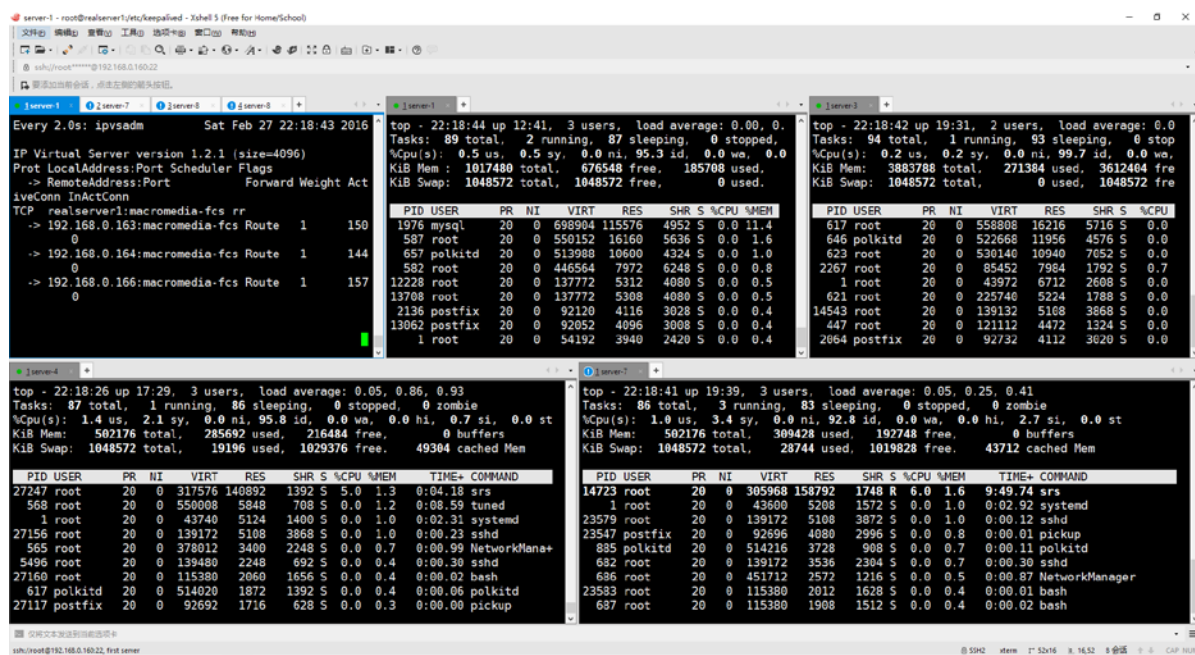


图 6.5(b) 第一次性能测试 450 请求

## 第六章 系统功能验证与性能测试

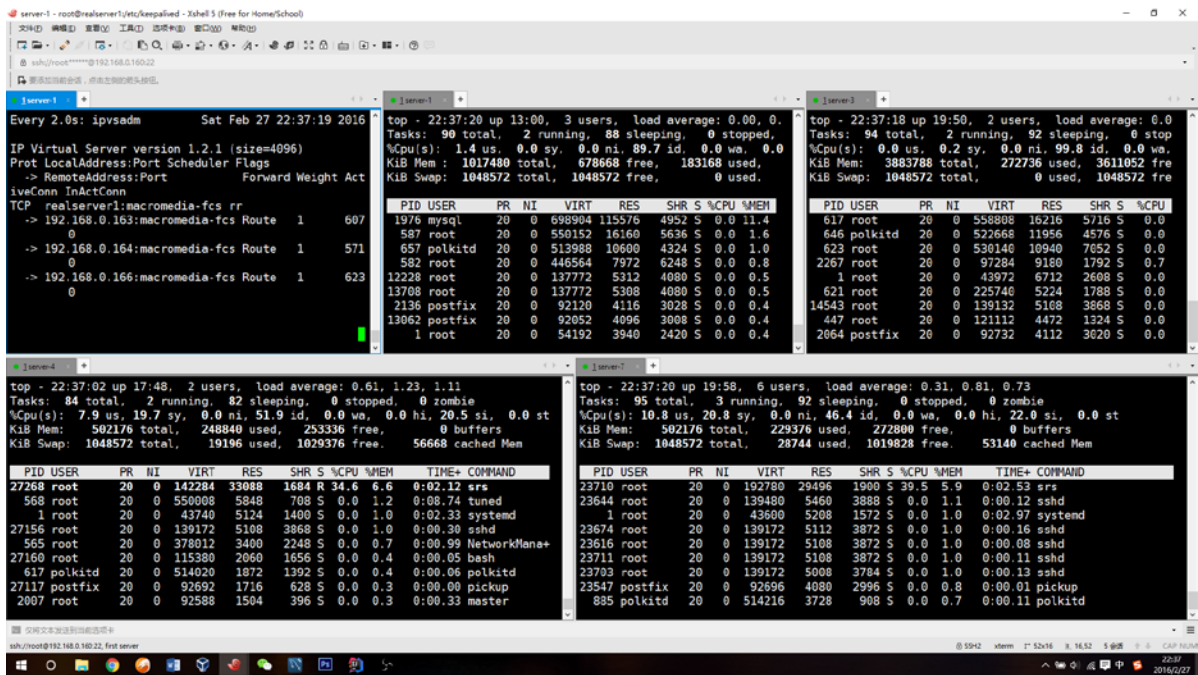


图 6.5(c) 第二次性能测试 1800 请求

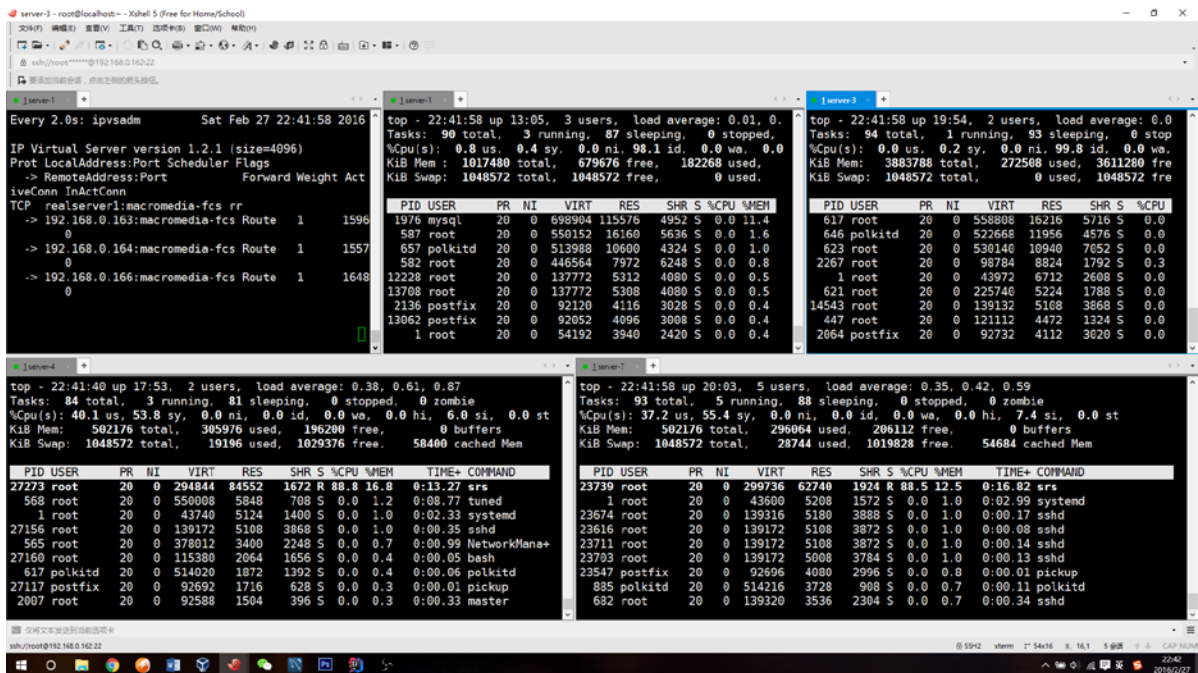


图 6.5(d) 第三次性能测试 4800 请求

由图中可以看出，测试结果整体如下：

对比属性	450请求	1800请求	4800请求
接到请求数	150 / 144 / 157	607/571/623	1596/1557/1648
CPU占用	5% / 5% / 6%	34.6% / 37.5% / 39.6%	88.8% / 87.5% / 88.5%
内存占用	1.3% / 1.3% / 1.6%	6.6% / 6.5% / 5.9%	16.8% / 17.1% / 12.5%
LVS-CPU	0%	0%	0%
LVS-内存	1.6%	1.6%	1.6%
SRS源-CPU	0.7%	0.7%	0.3%
SRS源-内存	0.5%	0.5%	0.4%

图 6.6 测试结果对比图

从图 6.6 中可以看出，在单核处理器+512MB 内存的情况下，SRS 的单机的处理能力在 1600 请求的时候就已经即将接近临界值，其中主要是处理器处理能力成为瓶颈。

而此时，不论请求如何增长，LVS 的 CPU 使用以及内存占用均不随之变化，也印证了 LVS 的直接路由模式的数据转发方式，能容纳百万级别的并发请求。

此时，SRS 的源站数据也可以表明其运作方式，由于本次实验只上传了一路直播流，由图 6.6(a)可以看出，直播流被 LVS 导向至 192.168.0.164 这台机器上，故 192.168.0.163 以及 192.168.0.166 均需要向源站请求流，源站从 192.168.0.164 服务器上拿到数据后分别转发给另外两台请求服务器，故在三次的实验中，源站的内存占用以及 CPU 占用均可忽略。

同时通过此次实验也侧面验证了 SRS 单机的处理能力，SRS 开发社区称其单机能力 SRS 本身单机处理能力在 3k-6k 之间，而本次采用的虚拟机单核处理器便已经能够达到 1600 并发的处理能力，其实际应用效果应该相对更好。

通过上述实验可得，本文在 3 台虚拟机的情况下便可支撑起近 5000 请求，而在真实环境中，只需要扩大 SRS 集群规模并配以相对更好的真实服务器配置，便可以真正达到百万级用户并发效果，较好的达到了设计预期。

## 6.4 可视化系统测试

### 6.4.1 机器注册测试

机器注册功能设计简单，拥有一个单独的页面，运维人员在页面中输入需要控制的

机器 IP 地址即可，后台会通过 HTTP 的方式向受控机器发送申请机器配置请求，拿到后会写入数据库中。如果取不成功会在页面上显示错误原因。效果如图 6.7(a)、6.7(b)、6.7(c)所示：

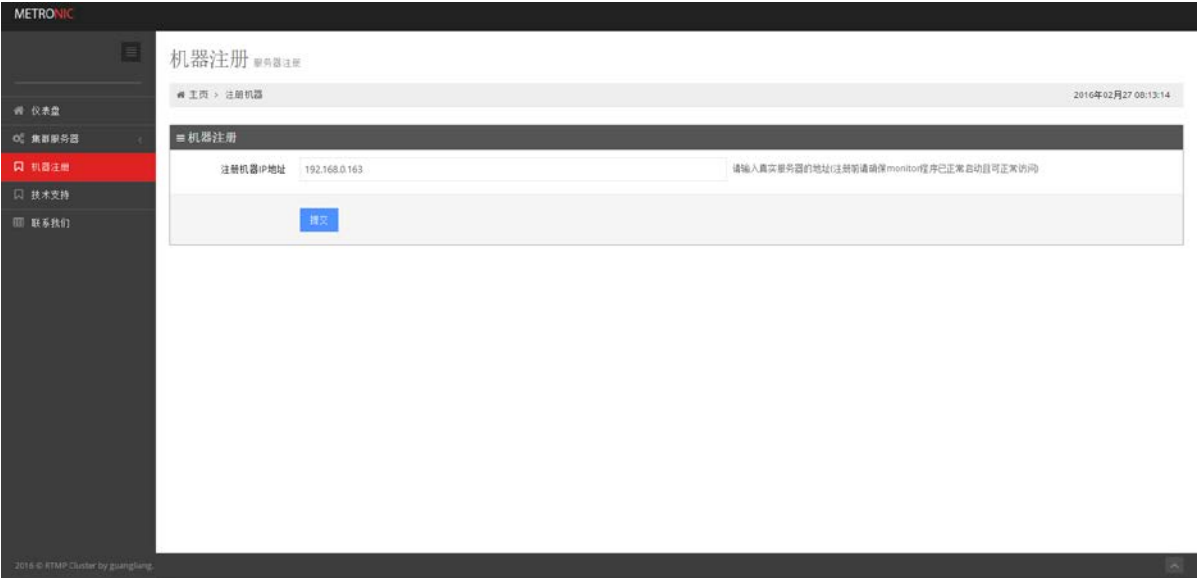


图 6.7(a) 设置页面截图

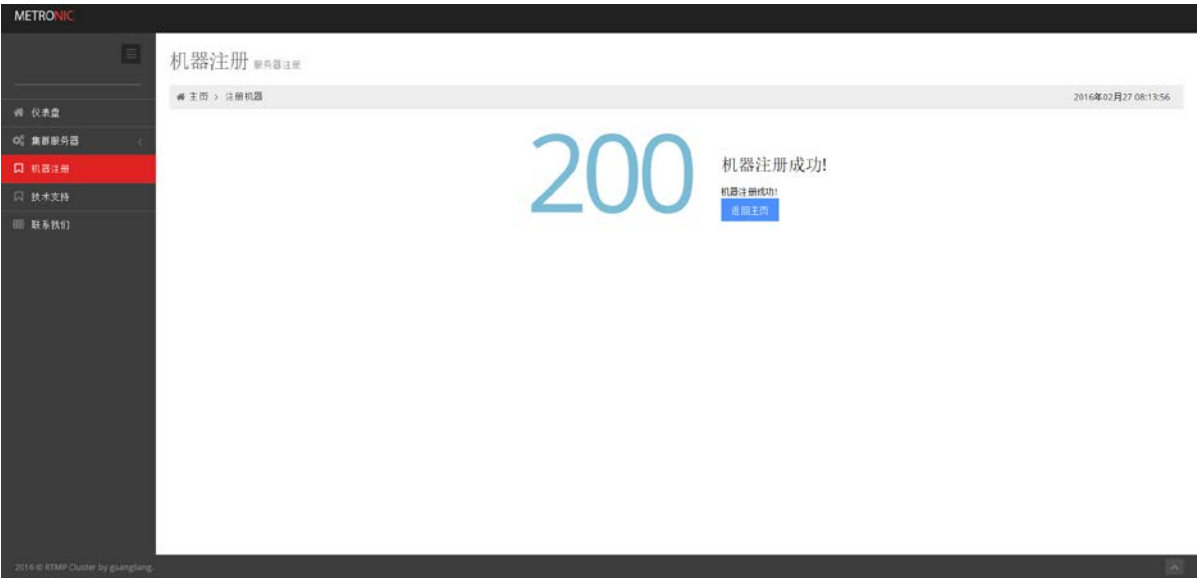


图 6.7(b) 注册成功截图

	id	generate_time	modify_time	name	operation_system	operation_arch	kernal	ip	mac_address	status	attributes
	20	2016-01-18 23:30:12	2016-01-18 23:30:12	Linux	CentOS 7.0.1406	x86_64	CentOS	192.168.0.166	08:00:27:68:06:4F	1	NULL
	21	2016-01-20 23:20:15	2016-01-20 23:20:15	Linux	CentOS 7.0.1406	x86_64	CentOS	192.168.0.165	08:00:27:76:5A:BA	1	NULL
	22	2016-01-19 02:50:42	2016-01-19 02:50:43	Linux	CentOS 7.0.1406	x86_64	CentOS	192.168.0.164	08:00:27:E7:9F:7B	1	NULL
▶	29	2016-02-27 21:13:56	2016-02-27 21:13:56	Linux	CentOS 7.0.1406	x86_64	CentOS	192.168.0.163	08:00:27:1B:0A:5E	1	NULL

图 6.7(c) 机器信息原始数据

由上图可看出，机器能够正确的插入数据库中，简洁美观的展示给用户，基本实现

了预期目标。

6.4.2 数据展现测试

数据展示功能为整个可视化系统最上层的模块，其余所有模块均为其服务，美观且有效的展现给用户是系统设计目标，此次测试采取功能测试的方式，通过整体系统运行 4 个小时形成有效数据，然后查看集群以及单服务器的展示情况，看起能否将数据准确有效的展现给用户来判断其是否达标。

图 6.8(a)、6.8(b)、6.8(c)、6.8(d)为相关截图：



图 6.8(a) 集群可视化系统运行截图

Id	generate_time	modified_time	cpu_average	memory_percent_average	net_send_vary_total	net_send_percent	net_recv_vary_total	net_recv_percent	rtmp_conn_total	success_number	failed_nu
757	2016-02-27 19:40:05	2016-02-27 19:40:05	0.0108	75.86	237	0	654	0	5	60	0
758	2016-02-27 19:45:05	2016-02-27 19:45:05	0.011	75.87	236	0	649	0	5	60	0
759	2016-02-27 19:50:05	2016-02-27 19:50:05	0.0115	75.84	237	0	650	0	5	60	0
760	2016-02-27 19:55:05	2016-02-27 19:55:05	0.0103	75.9	236	0	651	0	5	60	0
761	2016-02-27 20:00:05	2016-02-27 20:00:05	0.0105	75.9	236	0	649	0	5	60	0
762	2016-02-27 20:05:05	2016-02-27 20:05:05	0.0117	75.91	237	0	649	0	5	60	0
763	2016-02-27 20:10:05	2016-02-27 20:10:05	0.0118	75.92	236	0	656	0	5	60	0
764	2016-02-27 20:15:05	2016-02-27 20:15:05	0.0113	75.95	351	0	684	0	5	60	0
765	2016-02-27 20:20:05	2016-02-27 20:20:05	0.011	75.97	240	0	920	0	5	60	0
766	2016-02-27 20:25:05	2016-02-27 20:25:05	0.0112	75.99	865	0	926	0	5	60	0
767	2016-02-27 20:30:05	2016-02-27 20:30:05	0.0135	75.98	237	0	831	0	5	60	0
768	2016-02-27 20:35:05	2016-02-27 20:35:05	0.011	75.95	236	0	828	0	5	60	0

图 6.8(b) 集群五分钟维度原始数据





图 6.8(c) 单服务器可视化系统运行截图

847	2016-02-27 19:45:02	2016-02-27 19:45:02	0.01100026677337...	390138948	75.86883217570467	844837727	1067433546	236	649	5
848	2016-02-27 19:50:02	2016-02-27 19:50:02	0.01148692810457...	389970056	75.83598844495421	844851992	1067472672	237	650	5
849	2016-02-27 19:55:02	2016-02-27 19:55:02	0.01032679738562...	390319103	75.90386637354233	844866265	1067511687	236	651	5
850	2016-02-27 20:00:02	2016-02-27 20:00:02	0.01049999999999...	390295142	75.89920665264768	844880531	1067550837	236	649	5
851	2016-02-27 20:05:02	2016-02-27 20:05:02	0.01167483660130...	390363408	75.91248221075215	844894802	1067589813	237	649	5
852	2016-02-27 20:10:02	2016-02-27 20:10:02	0.01180065359477...	390414881	75.9224919815629	844909070	1067628881	236	656	5
853	2016-02-27 20:15:02	2016-02-27 20:15:02	0.01131372549019...	390561928	75.95108753371993	844924595	1067668605	351	684	5
854	2016-02-27 20:20:02	2016-02-27 20:20:02	0.01098692810457...	390643029	75.96685889674802	844944533	1067714656	240	920	5
855	2016-02-27 20:25:02	2016-02-27 20:25:02	0.01166666666666...	390742630	75.98622793602244	844986599	1067774438	865	926	5
856	2016-02-27 20:30:02	2016-02-27 20:30:02	0.01352147525676...	390736076	75.9849534824444	845010863	1067825778	237	831	5
857	2016-02-27 20:35:02	2016-02-27 20:35:02	0.01099346405228...	390545680	75.94792795089106	845025132	1067875511	236	828	5

图 6.8(d) 单服务器五分钟维度原始数据

由上述运行图可看出，机器能够正确的采集数据，并将数据写入相应的数据库中，并且集群也能够相应的时间做相应的数据处理，如定时同步数据，定时分析出集群的信息等，同时展示页面也能够正确且美观的展现出集群以及单服务器的数据，较好的完成了预期目标。

## 6.5 本章小结

第四章详细介绍了直播流媒体集群的设计与实现，第五章介绍了可视化系统的设计与实现。本章则对于整个系统进行了系统性的验证。

首先在第一节介绍了整体部署以及测试环境，为下文的测试奠定基础。

在第二节对于直播流媒体集群进行了整体的功能测试以及性能测试，验证了系统的整体功能，同时也验证了系统的高效性及鲁棒性。

第三节对于可视化系统进行了功能测试，验证了其现有功能符合预期设计效果，较好的完成了需求。

## 第七章 结论及展望

### 7.1 回顾与总结

在现有网络带宽得到大幅度改善的情况下，人们对于视频直播的需求益发的迫切，视频直播变成了一个越来越受欢迎的交互媒体，成为目前新兴的朝阳产业。而现有任意单直播流媒体服务器都不能支持大规模的用户并发请求，而目前集群技术以及负载均衡技术相对比较成熟，能够支撑系统的设计实现。在此背景下，本文确立了论文的研究重点与预期目标，即采用现有成熟集群及负载均衡技术，构建起一整套直播流媒体系统，以期实现大规模的用户请求数。同时为了方便运维人员的观察与维护，设计出一套可视化系统，能够方便直观的将集群的运行情况展现在运维人员面前。

本文第一章介绍了课题提出的背景以及直播技术的发展现状，然后在第二章详细介绍了本文使用到的相关理论知识，为后续系统的设计以及实现提供了理论支撑。

本文的第三章，本文对于整体需求做了详尽的分析。首先将目标进行拆解，将整体需求分为直播流媒体系统以及可视化系统两大子系统，然后对两个子系统的需求又进行了更为细化的描述与分析。

在本文的第四章与第五章，本文分别阐述了直播流媒体系统以及可视化系统的设计以及实现。首先对两个系统分别做了整体的架构设计，将具体需求转化为一个个的模块然后对于各个模块的技术关键点讲述了其具体实现过程。

最后本文对于直播流媒体系统做了功能以及性能测试，对可视化系统做了功能测试。得出测试结论，系统的所有功能均稳定可靠，性能良好，较好的实现了预期目标。

### 7.2 下一步优化方向

本文详细介绍了基于 RTMP 协议的直播流媒体系统及其可视化系统的设计与实现，较好的实现了稳定支撑大规模用户请求的接入的预期目标，同时设置了许多层面的主备功能，任何一条机器宕机均会有相应的服务器替代其服务器，保证了系统的高可用性。但对于系统的优化仍然还有一些工作需要进行。

首先本文在 DNS 负载均衡层面，并没有后端服务器的健康监测机制，设计中 DNS 针对的是 LVS 负载均衡器，但是如果 LVS 负载均衡器所在集群出现整个集群的宕机事件，由于缺乏健康监测机制，DNS 仍旧会将请求转发至该集群。较好的解决方式是自设 DNS 服务器，并且在 DNS 层面进行后端服务器的健康监测。

其次，由于 RTMP 协议下层基于 TCP 长连接。就意味着每一个用户的到来，都需

要与服务器建立一个持续的 **Socket** 连接，此次链接就会永久的占用掉该服务器的输出带宽。而在现有的设计方案中，一个 **LVS** 集群中的边缘服务器理论上可进行不断的横向扩展，只需要指明主服务器即可。但在随着集群的逐渐扩大，源站将逐渐成为系统的瓶颈。例如源站为万兆网卡，机器网卡占用 90% 为可用阈值，输入流为 720P 视频，不考虑 **CPU** 转发能力的情况下，则可用带宽为 9000Mbps，该流占用带宽为 1Mbps，则整个集群只有一组输入流的情况下，可水平扩展 9000 台边缘服务器。如果有 10 组流，则只能扩展 900 台，如果有 100 组流，则只能横向扩展 90 台边缘服务器。如果想要更多的边缘服务器，则必须增加源站数量，也就是扩大 **LVS** 集群，而 **LVS** 本身可支持百万级并发，会导致 **LVS** 负载均衡机器的性能浪费。

同时，目前采用一源站一备站的方式来保证源站的高可用性，但如果主备源站同时宕机，那么会导致整个集群不可用，所以在现有设计下，源服务器会逐渐的成为整个系统的瓶颈。

在此情况下，可分为两步来解决该问题。

首先进行源站的集群化来解决集群中主备源站宕机导致的整个集群服务不可用的情况，即每次生效的源站不再为一台机器，而是为多来源站同时工作，组成源站集群，当其中一台机器宕机时，其余源站通过一致性 **Hash** 算法，动态分配出其应当承担的负载，将宕机源站的负载平摊至剩余源站服务器。

其次使用 **Fast Paxos** 算法进行全集群改造来最终解决源站全部宕机但边缘服务器未全部宕机，服务却不可用问题。假设集群中共 201 台机器，需要主源站机器 10 台，备源站机器 10 台，边缘服务器 181 台。**RTMP** 服务器在启动前先进行一次选举，选举出 20 台源服务器，剩下 181 台为边缘服务器。当源服务器宕机到一定阈值时，例如宕机一半，触发重新选举源站，生成集群。例如 20 台全部宕机，则对剩下 181 台机器进行选举，选举出 18 台源服务器，163 台边缘服务器，从而能够保证整体集群的正常运转。解决源站全部宕机导致的集群不可用问题。

最后，本文仅采用请求模拟工具进行请求模拟，缺少真实环境下的大规模请求接入的测试过程。

## 参考文献

- [1] 雷霄骅, 姜秀华, 王彩虹. 基于 RTMP 协议的流媒体技术的原理与应用[J]. 中国传媒大学学报: 自然科学版, 2013, 06 期. 45809807291
- [2] 孙超. 流媒体服务器 Red5 的扩展设计、测试与优化[D]. 上海交通大学, 2009.
- [3] Whillock A, Chan E, Manapragada S, et al. Imparting cryptographic information in network communications: WO, US 8284932 B2[P]. 2012.
- [4] 李炳林. 流媒体技术及应用[J]. 电力系统自动化, 2004, 25(24):68-71.
- [5] Arumugam M. VOD and live TV channels for aircraft broadband networks[J]. Wichita State University, 2008.
- [6] Narula A, Mcmillen S, Karusala C. Multi-out media distribution system and method: US, US 8521899 B2[P]. 2013.
- [7] Ketmaneechairat H, Seewungkum D. The Development of Distance Learning Media System on the Internet by Using Macromedia Flash Server[J]. International Journal of Digital Content Technology & Its Applic, 2013.
- [8] Surhone L M, Tennoe M T, Henssonow S F, et al. Real Time Messaging Protocol[M]. Betascript Publishing, 2010. (Real Time Messaging Protocol)
- [9] 朱倩. 新一代流媒体 HLS 关键技术研究及实现[D]. 大连理工大学, 2011.
- [10] Chanchí Golondrino G E, Urbano Ordoñez F A, Campo Muñoz W Y. Stress tests for videostreaming services based on RTSP protocol[J]. Tecnura, 2015, 19.
- [11] Lei X, Jiang X, Wang C. Design and implementation of streaming media processing software based on RTMP[C]// Image and Signal Processing (CISP), 2012 5th International Congress on. IEEE, 2012:192-196.
- [12] 施建杨. 基于 WEB 的远程教育系统性能测试及优化策略研究[D]. 北京邮电大学, 2008.
- [13] Lei X, Jiang X, Wang C. Design and implementation of streaming media processing software based on RTMP[C]// Image and Signal Processing (CISP), 2012 5th International Congress on. IEEE, 2012:192-196.
- [14] Whillock A, Chan E, Manapragada S, et al. Imparting cryptographic information in network communications: WO, US 8284932 B2[P]. 2012.
- [15] 李强. 分布式拒绝服务(DDoS)攻击及防范解析[J]. 计算机安全, 2009(9):49-52.
- [16] 曹鸣鹏, 赵伟, 许林英. J2EE 技术及其实现[C]// 全国第四届 Java 技术及应用学术会议. 2001.
- [17] Johnson R. J2EE development frameworks[J]. Computer, 2005, 38(1):107-110.
- [18] Kayal D. Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework[M]. Apress, 2008.
- [19] Johnson R, Hoeller J, Arendsen A, et al. Professional Java Development with the Spring Framework[J]. Apc, 2006:195-237.

- [20] Zheng N, Liu X. Load balance optimization of a Red5 cluster in the mobile classroom project[C]// Natural Computation (ICNC), 2013 Ninth International Conference on. IEEE, 2013:1783-1787.
- [21] 杨昌武. 基于 LVS 集群负载均衡技术研究与应用[D]. 重庆邮电大学, 2013.
- [22] Liu K. To Achieve Load-Balance of Elective System with HAProxy[J]. Computer Knowledge & Technology, 2011.
- [23] 刘敏娜, 张继涛. 基于 LVS+KEEPALIVED 的高可用负载均衡研究与应用[J]. 自动化技术与应用, 2014, 33(11):22-27.
- [24] 王云岚, 李增智. 基于 DNS 的负载均衡算法研究[J]. 计算机工程与应用, 2002, 38(4):11-13.
- [25] Dias D M, Kish W, Mukherjee R, et al. A scalable and highly available web server[C]// Compcon '96. 'Technologies for the Information Superhighway' Digest of Papers. IEEE, 1996:85-92.
- [26] Hwang S, Jung N. Dynamic Scheduling of Web Server Cluster[C]// Proceedings of the 9th International Conference on Parallel and Distributed Systems. IEEE Computer Society, 2002:563-568
- [27] Scheflan I, Scheflan A, Martin J N. ONLINE MARKETING TOOL USING VIDEOS TO PROMOTE PRINTABLE COUPONS: , US20090030794[P]. 2009.
- [28] 谢茂涛. LVS 集群系统负载均衡策略的分析与研究[J]. 计算机时代, 2005(6):33-34.
- [29] 王雪莲, 曹青媚. LVS 集群调度算法的实现分析[J]. 科技传播, 2011(4).
- [30] Scheflan I. Video Promotion for Online Directory Listings and Other Search Engine Websites that List Advertisers: US, US20090013288[P]. 2009.
- [31] O'Rourke P, Keefe M. Performance Evaluation of Linux Virtual Server[C]// Proceedings of the 15th USENIX conference on System administration. USENIX Association, 2001:79-92.
- [32] Cavalli E. High Availability con Linux[J]. Bollettino Del Cilea, 2002.
- [33] Qian J H, Liao L. Realization of Dynamic Floating IP Cluster Based on Keepalived[J]. Control & Instruments in Chemical Industry, 2012.

## 致谢

岁月如歌，研究生求学生涯一晃而过，回首走过的三年时光，感慨颇多。庆幸自己能够来到北大这个极其优秀的平台，庆幸自己能够遇到诸多良师益友，也庆幸自己并未辜负这三年时光，我将从中受益终生。

我的指导教师蒋严冰老师是一名优秀的，具有丰富经验的教师。这篇文章是在蒋严冰老师的引领指导下完成的，蒋老师为我打开了流媒体直播领域的大门，又在之后的过程中不断给予支持，给予答疑解惑，开阔了我的视野，亦培养了良好的实验习惯及科研精神。蒋老师无论为人异或治学，都将作为我的榜样，将深刻影响着我日后的工作与生活。

感谢三年来研究生期间的小伙伴们，是你们的存在使得三年的学习生涯充满了欢笑。彼此间的鼓励、支持及逗乐使得原本略显紧张枯燥的学习过程充满了动力。与你们相处日子中数不尽的温暖和快乐让我终生难忘，期许之后的工作生活中也有你们的存在。

同时也感谢这篇论文所涉及到的各位学者。本文引用了多位学者的研究文献，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作。

感谢关心和爱护我的父母和家人，三年的学习过程中，是他们为我创造了学习深造的良好机会，不停的支持、安慰以及鼓励我，给我物质和精神上的极大支持和帮助。

最后，再次感谢所有给予我帮助的人，感谢各位在我没有方向的时候悉心安慰开导我，使我能够顺利完成这三年的学习时光。

## 北京大学学位论文原创性声明和使用授权说明

### 原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：                    日期：        年    月    日

### 学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校☐一年/☐两年/☐三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：                    导师签名：

日期：        年    月    日