基于画面切分的流媒体传输算法

张弛　　吴仕渠　　卞思沅

日期：2021 年 4 月 13 日

目录

1 介绍

随着互联网技术的发展，流媒体传输在人们的生活中越来越重要。通常来说，视频的发布与播放会经过下面几个步骤：视频创作者向平台发布视频，平台进行转码和处理，最后由 CDN 或云服务器分发给用户，用户设备上进行播放。

由于用户的硬件设备和网络情况各异，平台往往会将一份视频转换为多个不同码率的片段，供用户在不同网络环境下流畅播放。而用户接收视频流时，应用也会通过 ABR 算法通过各种用户参数动态调整加载视频片段的码率，以取得最佳的观看体验。

如今，ABR 算法已经非常成熟，可以适应不同的视频与不同的用户设备。然而，ABR 算法受到以下条件的限制：视频的码率有限，视频流分割的片段固定。在这种情况下，ABR 算法无法发挥最大的作用。

在本文中，我们提出了基于画面切分的视频切分方法 Video-Split，减少了视频平台转码所需的时间和存储所需的容量。与此同时，我们在此基础上实现了浏览器中的多路视频流同步技术 Buffered-Sync，以及多路视频流公平传输算法 Multi-ABR，保证用户端接收视频的质量。为了验证这一方案的有效性，我们开发了 Chaos-Proxy 网络环境模拟器，在 HTTP 应用层模拟不同的网络环境，验证 Multi-ABR 算法的可用性。

1.1 ABR 算法简介

ABR（Adaptive bitrate streaming）算法又称自适应码率算法。它可以根据用户网络状况、CPU 占用率等参数，动态调整视频码率，以提供使用者最佳的观看体验（QoE, Quality of Experience）。QoE 包括的指标一般有视频质量、卡顿和码率抖动等 [1]。现今的 ABR 算法主要基于 HTTP 协议，被广泛应用于因特网上的视频传输。

为用户提供高质量的视频体验需要我们优化两个一定程度上相互矛盾的变量：我们一方面希望视频码率越高越好。然而，高分辨率的视频会占用更多的网络带宽。当视频的比特率大于网络带宽时，便有可能造成视频的暂停（rebuffer），造成极差的体验。

视频播放器大多拥有一块缓存，用以存储已下载，但是仍未播放的视频块。缓存的存在可以有效缓解网路状态突然改变对用户视频观看造成的影响。然而，由于用户设备的限制，缓存不可能无限增大，因而其对网络状态改变的适应能力是有限的。视频缓存的另一个问题是，在观看视频时用户可能时不时地拉动进度条，从而跳过缓存部分。

主流的 ABR 算法会将视频切分成长度只有几秒的视频块。这些视频块被编码成不同码率。在用户播放视频时，系统根据用户的设备与网络状况自动传输最合适的码率。

对 ABR 算法的优化主要集中于利用时间序列分析 [2-3]、优化算法 [4-6]、机器学习 [7-9] 等等算法。通过更好地预测网络状况及用户的空余缓存容量等，我们可以选择最佳的视频码率。

2 设计与实现

2.1 视频分割与同步

2.1.1 Video-Split

我们视频的切分方式可以大体分为两步。基于 HLS 标准 [10] 先对视频在画面上切分，生成多个视频块；而后在时间上切分，生成多条 HLS 流。视频切分基于 ffmpeg 实现。

与普通的视频切分代码相比，我们的视频切分模块支持手动选择切分后单个视频流文件的大小，以及手动选择视频的画面切分策略。其中，视频的时间切分使用了 HLS-Stream-Creator

[11] 的实现代码，以实现多进程编码，减少转码消耗时间。

画面分割如图 1 所示。

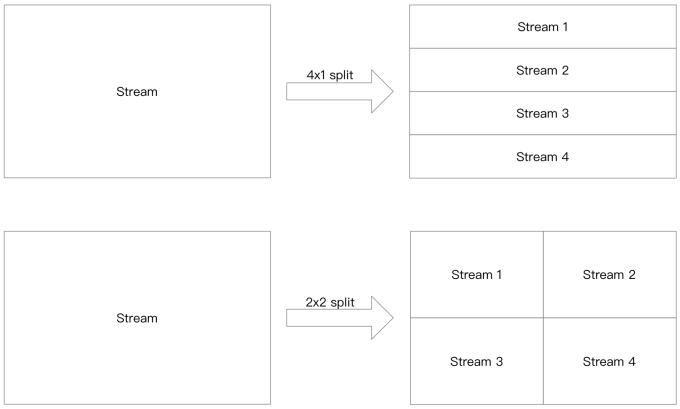

图 1: 画面分割示例

下面简要地介绍一下视频切分的步骤。我们首先按照指定的画面切分方式，将视频切成与原视频相同长度、相同比特率的多个视频块。之后，再按照指定的视频流大小，在第一步生成的视频块基础上，按照 HLS 标准生成不同码率的多个视频流。

在默认模式中，最高清晰度下，每个视频流文件的大小均保持在 4MB 左右。由于考虑到画面切分可能会带来一些压缩率的损失，我们在 HLS 标准的基础上，将视频码率码率放大到原标准的 1.1 倍左右。此时肉眼观察视频，清晰度几乎与不进行画面切分的标准方法生成的视频相同。

我们认为，此种视频切分方法通过将视频画面切分，为视频流的传输提供了更大的灵活性。同时，此种方法相对传统的视频流生成方法，不会有太大的性能、存储差距。由于第一步切分的视频与原视频分辨率相同，无需生成多种码率，因此较为容易，不会消耗太多时间。其次，第一步生成的画面切分后的视频块只是临时文件。在生成视频流文件后便可将其丢弃，不会长时间占用太多空间。我们将在实验部分具体展示我们的视频切分方法与传统切分方法的时间与存贮占用的对比。

2.1.2 Buffered-Sync

使用 Video-Split 进行视频切分固然可以起到节约转码时间、减小存储空间，同时可调码率范围的功能。但是，客户端接收到视频流后，需要重新拼接。现在视频流大多通过网页的形式传递。而浏览器提供的功能有限，无法将多个视频流同步。因此，我们引入 Buffered-Sync 解决多视频流同步的问题。

流媒体服务大多以网页的形式呈现视频，以时间为标准分割视频。比如爱奇艺、bilibili 等网站，通过网页端提供视频流，供用户观看。视频流被分割为若干个等时间或等大小的小块，根据用户的网络情况、设备情况按需加载，在客户端拼接成完整的视频，依次播放。

浏览器提供视频同步机制有限，无法满足多视频流同时播放的需求。首先，如图 2 所示，视频的播放收到设备的限制。一些设备的硬件条件有限，无法保证视频的每一帧都被绘制到显示

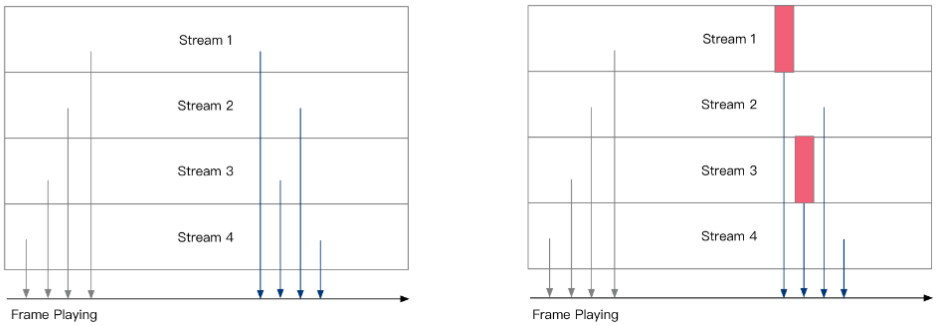

图 2: Buffered-Sync

器上，导致跳帧、丢帧的情况发生。其次，如图 3 所示，网页从收到播放指令到真正开始播放之间，需要一段时间，导致同时通过代码播放的视频，在用户看来，最终往往不是同步播放的。再次，视频的播放过程会受到系统负载的影响。即使是同时播放的视频，也会因为各种原因，或提前、或延后，有三到四帧的延迟。总之，浏览器只提供基本的单视频播放控制功能，不能保证多视频流的同步播放。

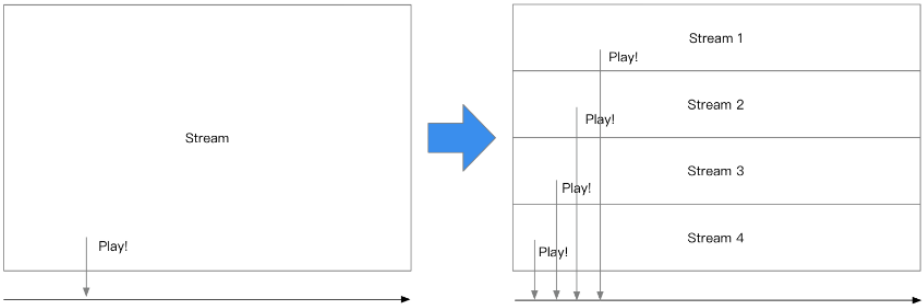

图 3: 多路视频播放不同步

Chrome 浏览器提供高效的视频处理 API，使得手动同步视频成为可能。Chrome 浏览器（或 Chromium 项目）提供 requestVideoFrameCallback API [12]。通过这个 API，我们可以获取当前视频播放的画面。由此，我们可以手动控制视频播放的流程，从而达到同步多个视频流的效果。

Buffered-Sync 利用浏览器提供的 Canvas API 和 VideoFrame API 手动同步多视频流。

如图 4 所示。网页中含有四个视频流 (Stream 1, Stream 2, ..., Stream 4)，由浏览器从服务端接收并绘制到屏幕上。首先，我们把这四条视频流在网页上隐藏起来，不把它们直接展现给用户。

而后，我们在后台创建多个虚拟画布 (Canvas)，用于缓冲接收到的视频数据。虚拟画布的个数与视频帧率、用户硬件配置、网络情况有关。虚拟画布个数设置为用户设备上多视频流帧数差值的预估值最佳。在这个例子中，我们创建 4 个虚拟画布。

最后，我们在网页上创建一个用户可见的画布，用于展示最终的视频。

当浏览器在后台开始处理某一个视频流的画面时，requestVideoFrameCallback 就会被调用。这时，Buffered-Sync 同步程序将某一视频流当前播放的帧绘制到虚拟画布上。一旦某个虚拟画布的四条视频流都已就绪（如图 X 第一个虚拟画布），这一画布上的内容就会被复制到用户可见画布上。如果确认某一条视频流的某一帧丢帧（如图 X 的第二个虚拟画布），这一画面就不会被

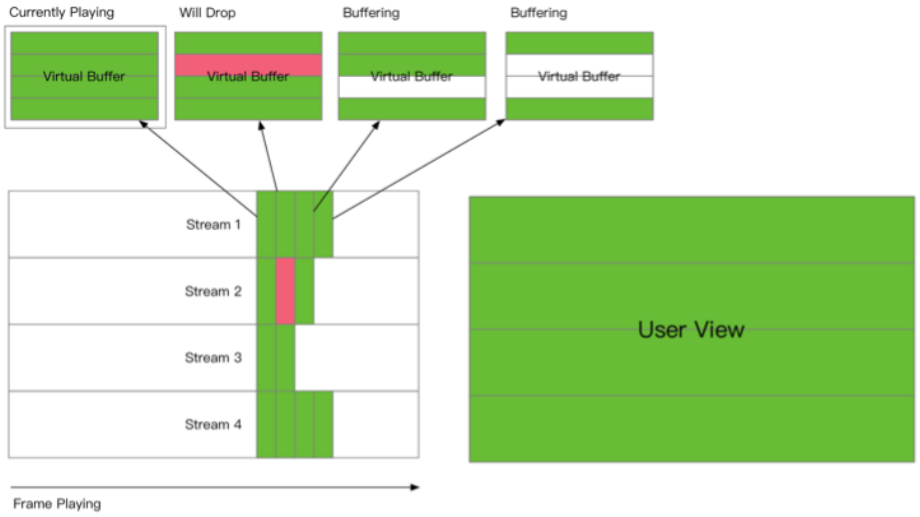

图 4: 虚拟画布的实现

绘制到用户可见画布上。

虚拟画布可以被循环利用，从而减小绘制开销。设当前播放的视频帧为第 x 帧。当我们在用户可见画布上播放完某一帧时，这一虚拟画布将被用来缓存第 x + 4 帧的内容。由此，我们无需清空虚拟画布的内容，只需要一直用新内容覆盖旧内容，即可完成同步。

Buffered-Sync 在最新版 Chrome 浏览器中使用 React 框架和 JavaScript 语言实现。

2.2 多路 ABR 设计与实现

实现了视频的切分与多路视频的同步播放后，我们还需要考虑视频加载的问题。在前文中，我们介绍了 ABR 算法，用于动态调整视频码率，以适应用户的网络和硬件环境。通常来说，一条视频流使用一个 ABR 控制器，执行 ABR 算法，控制缓存的使用和视频的播放。然而，在多视频流同时播放时，这一传统设计存在如下问题。

- **ABR 控制器独立工作，却互相影响**。如下图5所示，ABR 控制器会根据之前用户加载视频的情况，来确定下一次加载视频时采用何种码率。四条视频流同步加载，必然会导致视频流争抢有限的带宽，从而导致一些视频加载快、一些视频加载慢的情况。
- **ABR 控制器对于整体带宽估计量偏差大**。视频加载并不是一个持续很久的过程。加载视频往往只需要一瞬间。如果这些加载瞬间 (burst moment) 之间互不重叠，就会导致 ABR 控制器估计的带宽为总带宽。这样一来，四路 ABR 控制器同时工作时，加载视频的总码率会比带宽上限更高，从而导致卡顿的情况。由此，要同时播放四路视频，我们还要对 ABR 控制器做改进，防止视频缓冲不均匀导致卡顿的情况发生。我们引入 Multi-ABR 算法。Multi-ABR 算法由全局 ABR 控制器和多个子 ABR 控制器组成。全局 ABR 控制器收集所有子控制器的视频加载数据，根据总加载的带宽为每个子控制器分配带宽，从而保证子控制器之间协同工作，共享带宽。

ABR 算法主要分为两大步。一步是根据视频加载的统计数据预估网络带宽，另一步是根据预估的网络带宽选择合适的码率。Multi-ABR 算法接管了这两个过程，保证多 ABR 控制器播放视频的公平性。

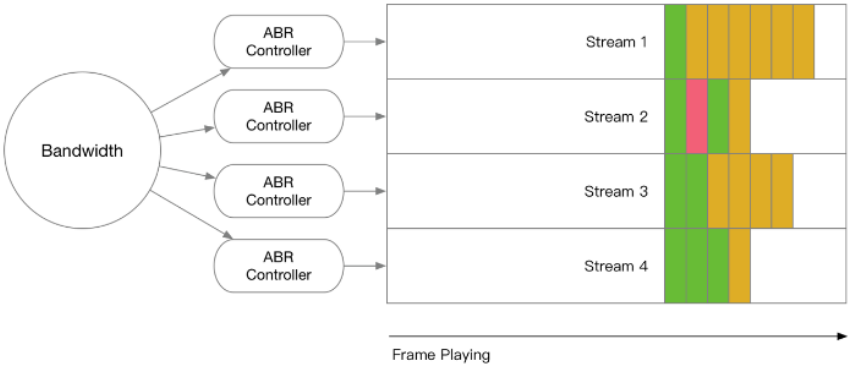

图 5: 多路 ABR 独立工作导致缓冲不均匀

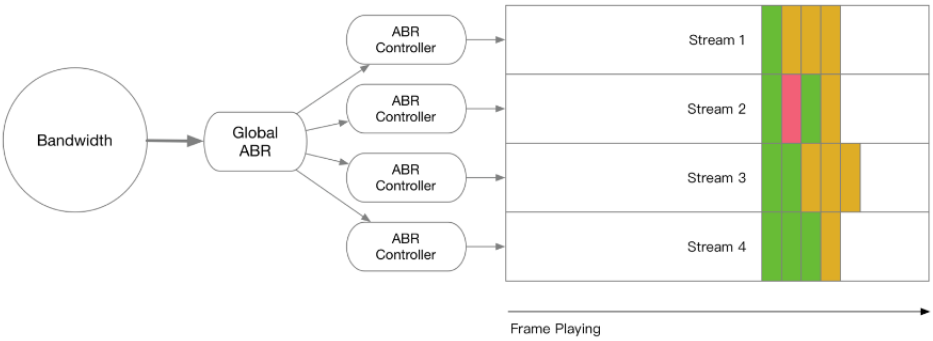

图 6: Multi-ABR 实现原理

在多个 ABR 控制器同时传输视频数据时，全局 ABR 控制器会将子控制器占用的网络带宽累加，从而估计出总网络带宽。和引入 Multi-ABR 算法之前的情况相比，Multi-ABR 使得我们可以正确估计总网络带宽，防止子控制器传输数据总和超过带宽的情况发生。

在子 ABR 控制器决定下一次传输视频码率的时候，全局 ABR 控制器告知子控制器，当前视频流最大允许的带宽有多大。Multi-ABR 算法采用完全公平 (completely-fair) 的调度方法，为每一个控制器分配相同的带宽用量。由此，子 ABR 控制器可以协同工作，按需缓存，减少多视频流播放时卡顿的情况发生。

我们修改了开源项目 hls.js，通过 TypeScript 实现了 Multi-ABR 算法。hls.js 提供一个完整的流视频处理和播放平台，和一个简单的 ABR 算法。我们对它的 ABR 算法进行了改进，对它的带宽估计器进行了重写，从而实现了 Multi-ABR 算法。

2.3 Chaos-Proxy 设计与实现

2.3.1 简介

一个典型的 Web 应用与用户间的交互可以分为四个部分：

1. 用户（User）通过浏览器发送一个 HTTP 请求；
2. 服务器（Server）收到请求，生成一个 HTML 文档；

3. 服务器（Server）把 HTML 文档作为 HTTP 响应的 Body 发送给浏览器；
4. 浏览器收到 HTTP 响应，从 HTTP Body 取出 HTML 文档并显示给用户。

所以，简单的 Web 应用就是先把 HTML 用文件保存好，用一个 HTTP 服务器接收用户请求，从本地读取文件，生成 HTML 文档并返回给客户端。这就是 Apache、Nginx、Lighttpd 等一些常见的静态服务器的实现原理，本项目的服务器属于此类静态服务器。在现实中，静态服务器传输文档时常会遇到网络拥塞，服务器传输速率将会受到限制。并且，现实不稳定的网络环境将时不时产生网络抖动 Jitter，导致传输速率跳变，甚至出现断网。为了证实我们提出的新型 ABR 算法可以适应现有的网络环境，我们需要模拟多种不同的网络状况，以确定新型 ABR 算法能针对不同的环境进行较好自适应调节，使得 QoE 指标维持在一个相对较高的水平。

2.3.2 总体思路

在 Server 端进行模拟需要改变服务器内置代码才能达到目的，需要对服务器的框架和主体代码做出较大改动。而我们通过增设一个代理服务器，使得原先客户端向服务器直接的请求变成向代理服务器发送请求，通过代理服务器再向原服务器发送原请求。代理服务器（Proxy Server）内置响应延迟时长（Response Latency）、传输速率上限（Transmission Speed）、网络抖动概率（Jitter）、断网概率（Disconnetion）等参数，以模拟各种典型的网络状况。代理服务器接受到原服务器的 HTTP 相应之后，再向客户端发送响应报文，从而在客户端达到模拟各种真实网络状况的目的。

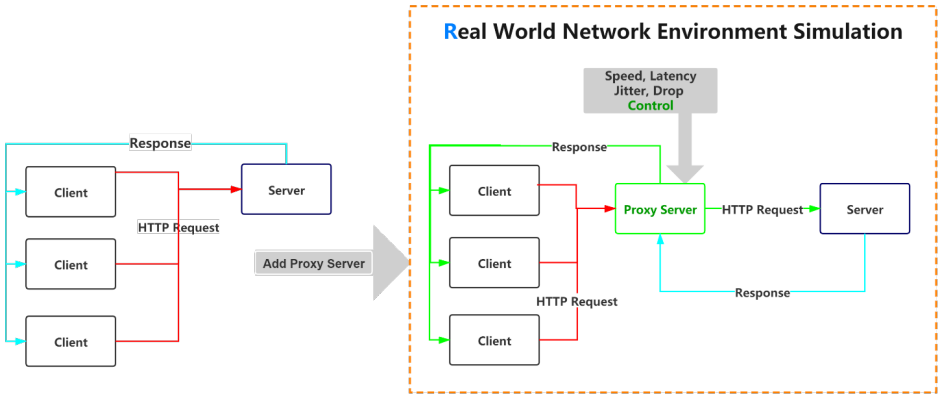

图 7: 模拟各种现实网络环境的总体思路

2.3.3 模块特性

我们使用了 Tornado Web Application 开发框架。

搭建 Web 应用程序常用的有三大框架：Django、Flask 以及 Tornado。三个框架各有特点，适合于不同场景的 Web 应用程序开发。其中 Django 是 Python 界最全能的 web 开发框架，各种 Web 功能完备，架构稳定，可维护性强。Django 完备的功能和普适性的框架适合于快速开发大型的项目。Django 的缺点就是处理速度慢，吞吐量小，其原因主要出在 Django 与数据库的交互上（这点可以通过 Celery 等工具解决）。Flask 适合于开发轻量级的 Web 应用程序，是微框架的典范。Flask 拥有简洁的接口和高度的灵活性，小型项目开发者可以使用 Flask 自由开发 Web 框

架和主体内容。然而这个灵活性也是一把双刃剑，经验不足的开发人员使用 Flask 框架容易写出低质量的代码，是得 Web 应用程序稳定性较差、难以维护。

Django 和 Flask 均是同步框架，在处理并发性请求时性能受限，不能很好的适应高吞吐量的应用场景。而 Tornado 天生自带异步、非阻塞特性，在 IO 密集型应用和多任务处理上占据着绝对性的优势。Tornado 良好的异步性能使得其搭建的 Web Application 在面对高并发的请求时能保证正常的吞吐量，适合于本项目的应用场景。协程（Coroutine）与异步框架效率高，因此在 Chaos-Proxy 中采用这种模式。

在实际服务器和客户端交互的过程中，Chaos-Proxy 支持两种模式的网络模拟。

Simple mode 是默认模式。可以使用 python config.py mode simple 设置。进入 simple mode 后，用户可以设置：

- Latency (ms) : HTTP 链接首字节传输的延迟；
- speed (bit/s)： 所有通过代理服务器的 HTTP 链接总速率限制。

Advanced Mode 是高级模式，在此模式下网络模拟器会引入随机的网络延迟、传输速率限制和抖动、断网等网络情况，可通过 python config.py mode advanced 设置。进入 Advanced Mode 后，用户可以设置以下模拟器参数：

1. latency-(min|max): HTTP 链接首字节传输延迟的上限和下限，默认情况下网络模拟器设置为 latency-min = latency-max = 1000ms；
2. speed-(min|max): 所有通过 chaos-proxy 的 HTTP 链接总速率限制上限和下限。通过设置速率的抖动上下限，可以模拟网络的不同稳定情况；
3. jitter-prob：网络抖动发生的概率（0-1 之间）；jitter-(min|max)：网络抖动的时间长度（以毫秒表示）。通过设置 jitter 参数，模拟器可以模拟网络抖动的情况和发生抖动的概率。
4. reset-enable：是否启用断网模式 (默认: false 启用: true)；reset-prob：在用户启用了断网模式后，断网发生的概率。网络模拟器通过主动断开与客户端的 HTTP 连接，来模拟服务器宕机的情况。

用户可通过组合多种参数的不同取值，来模拟复杂的网络环境（如传输速率跳变、响应延迟时长跳变、随机网络抖动等）。在测试过程中，我们先是使用 simple mode 对网络的基本环境进行了模拟。我们分别设置延迟时长（Response Latency）、传输速率上限（Transmission Speed）为 (300ms，5000bit/s), (500ms，3500bit/s), (1000ms，2000bit/s), (2000ms，800bit/s)，在服务器对应模拟网络状态下测试多个客户端并发请求不同字节客户端平均接收速率 V_{Client}。

3 实验

3.1 Video-Split

我们分别实验了以下几种视频切分方式，并对其**编码所需时间、占用空间大小、实现的码率档位划分数**进行了比较。

1. 不进行画面切分 (original);

2. 不进行画面切分，但是 Encoding ladder 调细，使得 Encoding ladder 总层数是原来的两倍 (2* encoding ladder);

3. 不进行画面切分，但是将 Encoding ladder 总层数改为原来的四倍 (4* encoding ladder);
4. 视频按照 2*1 的方式进行画面切分 (2*1 split);
5. 视频按照 4*1 的方式进行画面切分 (4*1 split);
6. 视频按照 2*2 的方式进行画面切分 (2*2 split)。

此实验在一台普通家用笔记本上完成，实验时控制了其它变量相同。实验结果如图8所示。

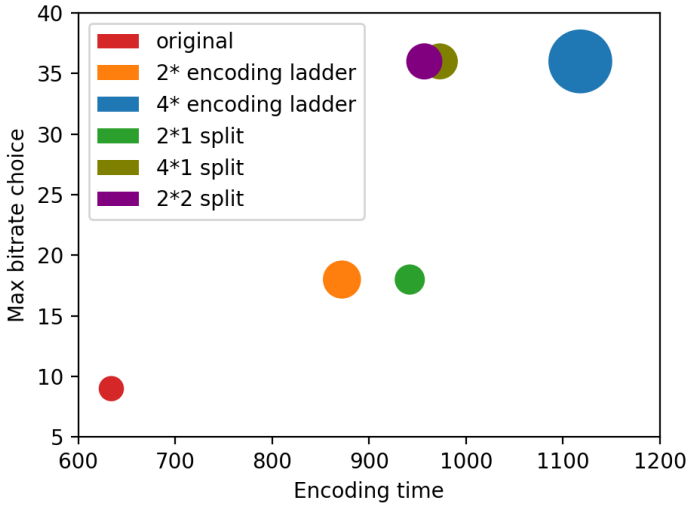

图 8: 不同切分方法对比实验结果

图中横坐标代表生成视频流编码所需的时间，纵坐标代表此种视频切分方式可以实现的码率调节档位（即等价的 encoding ladder 的层数）。图例的大小代表了此种编码方式生成的流视频空间占用量。

可以看到，将视频按照 4*1 或 2*2 的方式进行画面切分，可以在实现 36 种码率档位的同时消耗较少的时间并占用较少的体积。

3.2 Chaos-Proxy

我们通过 Apache Bench 对 Chaos-Proxy 进行了压力测试。测试包括：

- 速率限制测试。
- 延迟测试。
- 速率、延迟同时测试。
- 并发速率限制测试。
- 性能测试。

测试结果表明，Chaos-Proxy 可以同时处理上千的并发请求，每秒可以完成上百请求。

实验过程见附录。

3.3 Multi-ABR 测试与最终架构

最终演示与测试 Multi-ABR 算法的实验环境如图 9 所示。用户使用浏览器访问存在我们架设服务器上的网页。网页中含有 Multi-ABR 算法的实现。

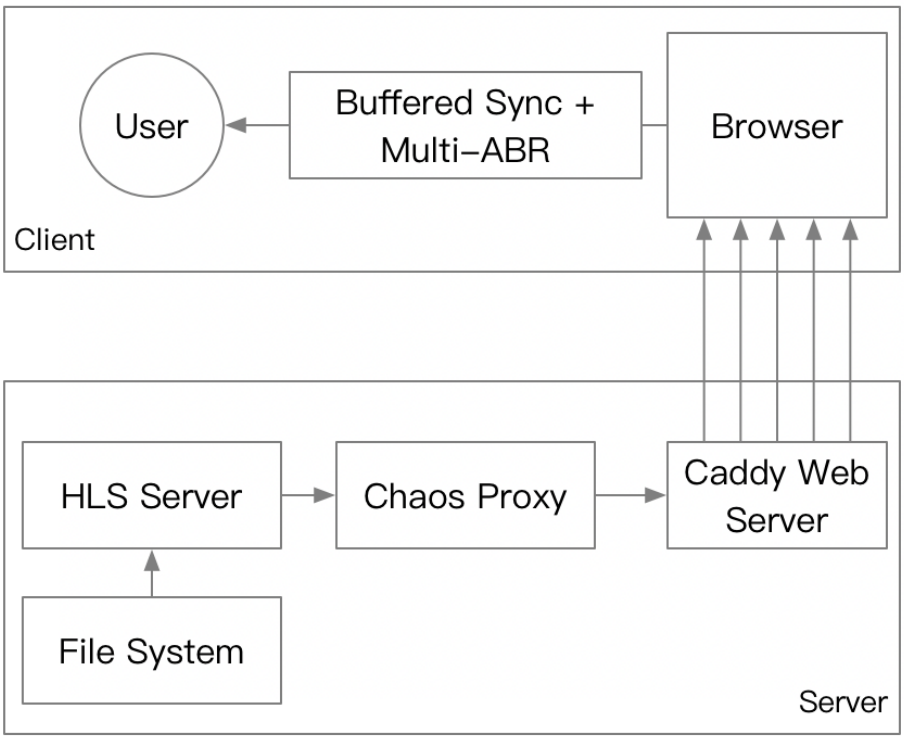

图 9

视频经过转码后以 HLS 协议兼容的方式保存在服务器上，通过 HLS Server 提供服务。Chaos-Proxy 代理所有到 HLS Server 的请求，从而模拟不同的网络环境。

用户打开网页后，Multi-ABR 算法开始工作，从服务器上同时加载多路视频数据。与此同时，Buffered-Sync 开始将视频数据绘制在浏览器中，展现给用户观看。

网页中还含有收集统计数据并可视化的功能。用户可以实时看到浏览器渲染的视频流是哪一帧，并获取当前播放的每个视频流的视频质量。

经测试，Buffered-Sync 可以完美同步多路视频流，减少多路视频播放不同步导致的画面撕裂感。Multi-ABR 算法使得多路视频加载更为均匀、高效，提升了观看体验。

我们的测试服务器 http://chaos.internal.skyzh.xyz 会一直在交大内网提供服务，直至学期结束或组长的交大云余额用完。

4 结论

- 基于画面的视频切分方法 Video-Split，有效地减少了视频平台转码所需的时间和存储所需的容量。
- 使用了浏览器中的多路视频流同步技术 Buffered-Sync 使得多路流媒体可以同步播放。
- 多路视频流公平传输算法 Multi-ABR 使得视频传输质量得到了保证，提升了加载视频的体验。
- 通过 Chaos-Proxy 网络环境模拟器，我们验证了 Multi-ABR 算法的有效性。

参考文献

[1] BENTALEB A, TAANI B, BEGEN A C, et al. A survey on bitrate adaptation schemes for streaming media over http[J]. IEEE Communications Surveys & Tutorials, 2018, 21(1):562-585.

[2] SOBHANI A, YASSINE A, SHIRMOHAMMADI S. A video bitrate adaptation and prediction mechanism for http adaptive streaming[J]. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2017, 13(2):1-25.

[3] CARDWELL N, CHENG Y, GUNN C S, et al. Bbr: Congestion-based congestion control[J]. Queue, 2016, 14(5):20-53.

[4] AKHTAR Z, NAM Y S, GOVINDAN R, et al. Oboe: auto-tuning video abr algorithms to network conditions[C]// Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2018: 44-58.

[5] SPITERI K, URGAONKAR R, SITARAMAN R K. Bola: Near-optimal bitrate adaptation for online videos[J]. IEEE/ACM Transactions on Networking, 2020, 28(4):1698-1711.

[6] YIN X, JINDAL A, SEKAR V, et al. A control-theoretic approach for dynamic adaptive video streaming over http[C]// Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. [S.l.: s.n.], 2015: 325-338.

[7] LI E, ZENG L, ZHOU Z, et al. Edge ai: On-demand accelerating deep neural network inference via edge computing[J]. IEEE Transactions on Wireless Communications, 2019, 19(1):447-457.

[8] CHIARIOTTI F, D'ARONCO S, TONI L, et al. Online learning adaptation strategy for dash clients[C]//Proceedings of the 7th International Conference on Multimedia Systems. [S.l.: s.n.], 2016: 1-12.

[9] MAO H, NETRAVALI R, ALIZADEH M. Neural adaptive video streaming with pensieve[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2017: 197-210.

[10] INC A. Http live streaming[EB/OL]. 2009. https://developer.apple.com/streaming/.

[11] Hls-stream-creator[EB/OL]. 2020. https://github.com/bentasker/HLS-Stream-Creator.

[12] STEINER T, VERBORGH R, VALLÉS J G, et al. Enabling on-the-fly video shot detection on youtube[C]//Proc. WWW. [S.l.: s.n.], 2012.

A Chaos-Proxy 压力测试

1. 测试服务器（Server）每秒传输速率上限。

   参数：Speed: 30000000000 bit/s（不做限制）Latency: 0 ms (不做限制)

```
wsq@wsq-vm:~$ ab -n 100 -c 1 http://localhost:2334/blank/500000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking localhost (be patient).....done
Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/500000
Document Length:        500000 bytes

Concurrency Level:      1
Time taken for tests:   2.069 seconds
Complete requests:      100
Failed requests:        0
```

```
Total transferred:      50032400 bytes
HTML transferred:       50000000 bytes
Requests per second:    48.32 [#/sec] (mean)
Time per request:       20.693 [ms] (mean)
Time per request:       20.693 [ms] (mean, across all concurrent requests)
Transfer rate:          23611.27 [Kbytes/sec] received
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:    15   20   6.6     17      38
Waiting:       13   18   6.0     15      35
Total:         15   21   6.6     17      39
Percentage of the requests served within a certain time (ms)
  50%     17
  66%     20
  75%     25
  80%     28
  90%     32
  95%     34
  98%     37
  99%     39
 100%     39 (longest request)
```

2. 测试 Latency 参数：

   参数：Speed: 30000000000（不做限制）Latency: 0 ms (不做限制)

```
wsq@wsq-vm:~$ ab -n 100 -c 1 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done


Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      1
Time taken for tests:   1.227 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    81.49 [#/sec] (mean)
```

```
Time per request:       12.272 [ms] (mean)
Time per request:       12.272 [ms] (mean, across all concurrent requests)
Transfer rate:          423.52 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:     8   12   3.7     10      19
Waiting:        7   11   3.5     10      18
Total:          8   12   3.8     10      19
Percentage of the requests served within a certain time (ms)
  50%     10
  66%     13
  75%     16
  80%     17
  90%     18
  95%     19
  98%     19
  99%     19
 100%     19 (longest request)
```

3. 测试 Latency 参数：

   参数：Speed: 30000000000（不做限制）Latency: 2000 ms

```
wsq@wsq-vm:~$ ab -n 100 -c 1 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      1
Time taken for tests:   202.369 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    0.49 [#/sec] (mean)
Time per request:       2023.694 [ms] (mean)
Time per request:       2023.694 [ms] (mean, across all concurrent requests
    )
```

```
Transfer rate:          2.57 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:  2012 2023   4.2   2024    2040
Waiting:     2012 2022   4.1   2023    2039
Total:       2012 2023   4.2   2024    2040

Percentage of the requests served within a certain time (ms)
  50%   2024
  66%   2025
  75%   2026
  80%   2026
  90%   2028
  95%   2030
  98%   2033
  99%   2040
 100%   2040 (longest request)
```

4. 测试 Speed 参数（对单用户是否达到速率限制）：

   参数：Speed: 5000 Latency: 0 ms (不做限制)

```
wsq@wsq-vm:~$ ab -n 100 -c 1 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      1
Time taken for tests:   102.372 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    0.98 [#/sec] (mean)
Time per request:       1023.716 [ms] (mean)
Time per request:       1023.716 [ms] (mean, across all concurrent requests
    )
Transfer rate:          5.08 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:  1011 1023   4.9   1024    1043
Waiting:        8   20   5.3     20      39
Total:       1011 1023   4.9   1024    1043

Percentage of the requests served within a certain time (ms)
  50%   1024
  66%   1025
  75%   1026
  80%   1026
  90%   1028
  95%   1029
  98%   1035
  99%   1043
 100%   1043 (longest request)
```

5. 测试 Speed 参数（对多用户是否达到速率限制）：此处采用并发性 100
   参数：Speed: 30000000000（不做限制）Latency: 0 ms (不做限制)

```
wsq@wsq-vm:~$ ab -n 100 -c 100 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      100
Time taken for tests:   1.037 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    96.44 [#/sec] (mean)
Time per request:       1036.932 [ms] (mean)
Time per request:       10.369 [ms] (mean, across all concurrent requests)
Transfer rate:          501.22 [Kbytes/sec] received

Connection Times (ms)
```

```
              min  mean[+/-sd] median   max
Connect:        0    7   2.4      7      22
Processing:    33  549 258.4    462    1003
Waiting:       11  538 257.0    457    1001
Total:         34  556 258.5    468    1011

Percentage of the requests served within a certain time (ms)
  50%    468
  66%    681
  75%    783
  80%    797
  90%    930
  95%   1006
  98%   1011
  99%   1011
 100%   1011 (longest request)
```

6. 测试 Speed 参数（对多用户是否达到速率限制）：此处采用并发性 100
   参数：Speed: 5000 Latency: 0 ms (不做限制)

```
wsq@wsq-vm:~$ ab -n 100 -c 100 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      100
Time taken for tests:   100.281 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    1.00 [#/sec] (mean)
Time per request:       100281.427 [ms] (mean)
Time per request:       1002.814 [ms] (mean, across all concurrent requests
    )
Transfer rate:          5.18 [Kbytes/sec] received
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    8   1.9      8      11
Processing:  1039 50644 29083.8  51148  100254
Waiting:       11  492 204.4    429     813
Total:       1039 50653 29083.9  51156  100261
```

```
Percentage of the requests served within a certain time (ms)
  50%  51156
  66%  67192
  75%  76220
  80%  81232
  90%  91252
  95%  96255
  98%  99261
  99%  100261
 100%  100261 (longest request)
```

7. 测试代理服务器性能

   参数：Speed: 30000000000（不做限制）Latency: 2000 ms

```
wsq@wsq-vm:~$ ab -n 100 -c 1 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334

Document Path:          /blank/5000
Document Length:        5000 bytes

Concurrency Level:      1
Time taken for tests:   202.369 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      532200 bytes
HTML transferred:       500000 bytes
Requests per second:    0.49 [#/sec] (mean)
Time per request:       2023.694 [ms] (mean)
Time per request:       2023.694 [ms] (mean, across all concurrent requests
    )
Transfer rate:          2.57 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:  2012 2023   4.2   2024    2040
Waiting:     2012 2022   4.1   2023    2039
Total:       2012 2023   4.2   2024    2040
```

```
Percentage of the requests served within a certain time (ms)
  50%   2024
  66%   2025
  75%   2026
  80%   2026
  90%   2028
  95%   2030
  98%   2033
  99%   2040
 100%   2040 (longest request)
```

8. 测试代理服务器性能

   参数：Speed: 30000000000（不做限制）Latency: 2000 ms

```
wsq@wsq-vm:~$ ab -n 800 -c 800 http://localhost:2334/blank/5000
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
Benchmarking localhost (be patient)
Server Software:        TornadoServer/5.1.1
Server Hostname:        localhost
Server Port:            2334
Document Path:          /blank/5000
Document Length:        5000 bytes
Concurrency Level:      800
Time taken for tests:   12.449 seconds
Complete requests:      800
Failed requests:        0
Total transferred:      4257600 bytes
HTML transferred:       4000000 bytes
Requests per second:    64.26 [#/sec] (mean)
Time per request:       12448.513 [ms] (mean)
Time per request:       15.561 [ms] (mean, across all concurrent requests)
Transfer rate:          334.00 [Kbytes/sec] received

              min  mean[+/-sd] median   max
Connect:        0  903 932.0   1021    3051
Processing:  2152 5237 1470.2   5305    7550
Waiting:     2023 5227 1472.3   5300    7537
Total:       2152 6140 2135.5   6325   10319

Percentage of the requests served within a certain time (ms)
  50%   6325
  66%   6744
  75%   7557
  80%   7934
  90%   9626
  95%  10022
```

```
 98%  10223
 99%  10294
100%  10319 (longest request)
```