広島大学先進理工系科学研究科
情報科学プログラム

2024 年度

修士論文

A P2P-Based Framework for Distributed Video Stream Caching and Aggregation

広島大学先進理工系科学研究科

2024 年 1 月 27 日提出
指導教員　藤田　聡　教授

耿　傑然
Kou　Ketsuzen

Abstract

The exponential increase in Internet traffic and expansion in size have notably influenced the proliferation of video streams, with a significant emphasis on HTTP video streams. This phenomenon is particularly evident in content sharing and distribution, where a substantial portion of videos revolves around common themes, such as Oktoberfest. In response to this trend, a novel system leveraging Peer-to-Peer (P2P) technology for efficient caching and bundling of thematically similar video content is proposed. This system innovatively utilizes the distributed processing capabilities of P2P nodes to cache HTTP video segments and subsequently synthesize new video streams. The experimental evaluation, which employed traditional servers as a benchmark, demonstrated the system's efficacy. It yielded markedly better performance for scenarios involving low-quality video and high request volumes. Additionally, the system showed comparably favorable outcomes in other scenarios, underscoring its potential for optimizing video content delivery in diverse conditions.

Contents

Chapter 1

Introduction

1.1 Introduction

The exponential growth of internet connectivity, projected to encompass two-thirds of the global population by 2023 according to Cisco's 2018-2023 forecast, has significantly contributed to the burgeoning popularity of video sharing platforms like YouTube, TikTok, and Netflix. This escalation in internet users correlates directly with an increase in video traffic, thereby solidifying HTTP video streaming's role as a pivotal technology for delivering a diverse array of high-resolution content with minimal latency. This trend is further reinforced by recent scholarly work highlighting major advancements in video codecs and the widespread adoption of video-enabled devices [1–3].

Central to the mechanism of HTTP streaming is HTTP Live Streaming (HLS), introduced by Apple in 2009. HLS utilizes segmented MPEG2-TS containers, enabling the delivery of fragmented video and audio files via HTTP/1.1. Originating from HDTV broadcasting technology, MPEG2-TS segments video data into small transport packets, which are instrumental for error correction, packet reordering, and simultaneous playback/download. HLS operation depends on a manifest file, located on the web server, that delineates the segmented media files' locations for a given video. Analogous to HLS, MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [4], ratified by ISO MPEG in 2012, employs an XML-formatted MPD (Media Presentation Description) as its manifest file. Other prominent HTTP/1.1 streaming technologies include Microsoft's Smooth Streaming [5] and the CMAF (Common Media Application Format) [6].

A hallmark of HTTP streaming is its capacity to divide video content into segments with variable bitrates, facilitating adaptive bitrate (ABR) switching in response to network conditions. This feature enables clients experiencing bandwidth fluctuations to request lower-resolution chunks, thereby maintaining continuous playback. However, with the escalating number of users and the voluminous increase in video content, media servers face an increased load. This can result in unintentional bitrate reductions due to server congestion, adversely affecting the user experience.

These observations highlight the challenges in achieving scalable bitrate switching for high-quality HTTP streaming within conventional client-server frameworks dependent on dedicated servers. A viable alternative to address these challenges is the integration of peer-to-peer (P2P) technology, wherein clients contribute their upload bandwidth to facilitate direct video content sharing among themselves. This decentralized methodology presents several advantages, including:

- **Reduced Server Load:** P2P distribution alleviates the burden on central servers, particularly during peak usage, by offloading video delivery, thereby mitigating congestion and bandwidth bottlenecks.
- **Enhanced Scalability:** The combined upload capacity of additional clients in the network contributes to an expanded resource pool, allowing for the network's seamless expansion in line with user growth.

- **Improved Resilience:** P2P systems inherently possess fault tolerance and redundancy, circumventing single points of failure. In case a peer is unavailable, other nodes can continue content distribution, thus maintaining system stability and minimizing disruptions.

The feasibility and potential of P2P technology in augmenting video streaming have been extensively explored in the last two decades. A multitude of studies have examined various P2P streaming architectures and protocols [7–12], providing a substantial theoretical foundation for the development of effective and efficient P2P-based solutions for high-quality HTTP streaming. These insights and findings are instrumental in addressing the constraints inherent in traditional server-centric models.

This paper introduces a crowdsourced video streaming system to explore novel possibilities for P2P-assisted HTTP video streaming. The system is designed to facilitate the sharing of multiple video streams captured by users participating in large events, such as Oktoberfests and parades, on a P2P network comprising nodes associated with participating users. User-generated video streams undergo conversion to the HLS format, ensuring system versatility, a new video stream bundled from cached chunk in the P2P network, available for download upon requests. Activation of the system aligns with the event's timing, enabling users to join and leave at their discretion. The proposed system avoids centralized online cache management, reducing the risk of processing bottlenecks and single points of failure. Only the bootstrap node (coordinator), serving as the entry point for participating peers, is installed, managing limited tasks: bootstrapping and restarting the maintenance of the P2P overlay.

The architecture of the system incorporates a bespoke P2P network, termed the Worker Network, which is established on the foundations of the InterPlanetary File System (IPFS). By leveraging the inherent infrastructure components of IPFS, such as the Bitswap protocol and the Pubsub model, the system effectively constructs a P2P network grounded in the kad-DHT (Kademlia Distributed Hash Table) framework. This approach not only ensures the system's performance and availability but also simultaneously enhances its security and flexibility. The foundational technology of the system is based on FFmpeg, a leading multimedia framework.

Furthermore, a specialized video streaming module has been developed, also grounded in the FFmpeg framework. This module operates autonomously within each node of the P2P network and is tasked with the distributed processing of video streams. Its design is optimized to capitalize on the substantial computational resources often lying idle within the network. By harnessing these resources, the module enhances the efficiency and scalability of video processing across the network, thereby maximizing the overall utility of the system's distributed architecture.

The proposed system is implemented in a simulate network environment created via Docker using the Go language, and the evaluation assesses the time required for storing the generated video stream in the P2P network and retrieving/playing the cached video content. Experimental results confirm the effectiveness of the proposed system, demonstrating its suitability for sharing user-generated video contents in an ad hoc manner.

1.2 Related Work

In the realm of P2P caching systems, significant contributions have been made in the context of adaptive video streaming. Al-Habashna et al. presented a P2P caching system specifically designed for DASH (Dynamic Adaptive Streaming over HTTP) in their publication [13]. Conversely, Roberto Roverso and colleagues adopted a combined CDN-P2P (Content Delivery Network-Peer-to-Peer) strategy to augment the efficiency of adaptive HTTP Live Streaming [14].

However, the integration of video processing capabilities within P2P networks remains relatively underexplored. This gap in the research can be attributed to the substantial challenges associated with such an integration. Specifically, the transmission of large-scale video content over networks demands considerable bandwidth, and the processing of these videos requires significant computational power.

Silva et al. recognized the immense potential of P2P systems in caching videos that share common themes or subject matter [15]. Yet, their work did not extend into the realm of video processing.

Addressing this gap, the proposed system in this paper presents a novel solution that mitigates the issues of high bandwidth and computational demands. This is achieved by distributing segmented HLS (HTTP Live Streaming) video slices, rather than entire videos, across the P2P network. Such an approach significantly reduces the bandwidth and arithmetic power required for video processing in P2P networks, thereby enabling more efficient and scalable video distribution and processing.

1.3 Paper Structure

The remainder of this paper is organized as follows. Chapter 2 outlines key technology stack. After an overview of the proposed system in Chapter 3, the management of the P2P overlay and the storage and retrieval of user-generated video streams are described in Chapter 4 and Chapter 5, respectively. Chapter 6 describes the results of experiments conducted on our prototype system. Finally, Chapter 7 concludes the paper with future work.

Chapter 2

Technology Stack

In the proposed system, the foundational architecture of the Worker Network is meticulously constructed utilizing the core components of the InterPlanetary File System (IPFS). Concurrently, the Video Processing Module is adeptly realized through the application of FFmpeg. Together, these technologies form the cornerstone of the system's core technology stack, underpinning its functionality and efficiency.

2.1 InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is a pivotal component in the architecture of the system, serving as a fundamental library for the implementation of the distributed network. IPFS is a peer-to-peer (P2P) hypermedia protocol, designed with the aim to make the web faster, safer, and more open. It has emerged as a prominent solution for decentralized storage and sharing of data in a distributed network.

IPFS has following core functions:

Decentralized Distribution Unlike traditional client-server protocols, IPFS operates on a P2P basis, eliminating reliance on centralized servers. This decentralization facilitates more robust and resilient data storage and access.

Content Addressing IPFS uses content-based addressing rather than location-based addressing. Each file and all blocks within it are given a unique fingerprint called a cryptographic hash. This ensures that every piece of content can be uniquely identified and retrieved based on its content, not its location.

Efficient File Storage and Retrieval By storing files in a distributed network and retrieving them through global content addresses, IPFS optimizes bandwidth usage and improves file transfer speeds.

Version Control and Linking IPFS integrates features akin to version control systems. It can track versions of files and manage data in a way that makes it easy to link from one piece of content to another.

In the system, IPFS is instrumental in the creation and maintenance of the Worker Network. It enables efficient and reliable storage and retrieval of video content across the network. The use of IPFS's DHT (Distributed Hash Table) for content discovery and the Bitswap protocol for data exchange are key in achieving a decentralized and efficient distribution of video content. Moreover, the Pubsub model of IPFS facilitates a reactive and dynamic content distribution strategy, enhancing the system's overall performance.

IPFS's integration into the system significantly contributes to overcoming challenges associated with traditional centralized networks, such as single points of failure and scalability issues. By leveraging the robustness and efficiency of IPFS, the system achieves a decentralized, scalable, and resilient architecture for video content distribution and processing.

2.2 FFmpeg

In the domain of the system's video processing and streaming functionalities, FFmpeg stands as a cornerstone library. FFmpeg is an open-source software suite, widely renowned for its comprehensive capabilities in handling multimedia data. It encompasses a vast array of tools and libraries for recording, converting, and streaming audio and video in various formats.

There are some principal features of FFmpeg:

Extensive Format Support FFmpeg is celebrated for its broad support of multimedia file formats, codecs, and protocols. This versatility makes it an invaluable tool for a system that needs to handle a diverse range of video and audio data.

Transcoding and Processing At its core, FFmpeg excels in transcoding multimedia files - converting them from one format to another. This feature is essential for transforming source video files into compatible formats for efficient streaming and processing.

Streaming Capabilities FFmpeg also facilitates live streaming functionalities. Its capability to encode and stream media in real-time is pivotal for systems requiring live broadcast or real-time video processing features.

High Performance Known for its high performance and quality output, FFmpeg processes and transcodes video and audio with minimal loss of quality, ensuring optimal playback and viewing experiences.

Within the system, FFmpeg serves a vital role in video processing tasks. This encompasses a spectrum of functions including encoding, decoding, and bundling of video content, all of which are executed through calls to the FFmpeg library. Detailed insights into its implementation and operational role are provided in Chapter 5 of this paper.

In summary, the integration of FFmpeg into the system provides a robust, efficient, and flexible framework for video processing and streaming. Its comprehensive set of tools and capabilities significantly contributes to the system's ability to handle complex video processing tasks, making it a vital component in the system's overall architecture.

Chapter 3

System Overview

3.1 Overview

The proposed system is composed of multiple worker nodes and a dedicated coordinator. At its core, the system relies on the **worker network**, a peer-to-peer (P2P) network of worker nodes responsible for storing chunk data as the primary source of video streaming. Detailed information about the worker network is provided in Chapter 4. The principal role of the coordinator is to oversee the DHT utilized by the worker network and to facilitate the efficient transfer of video chunks within the worker network.

All client devices involved in the system, including laptop PCs and tablets, function as worker nodes. These nodes encompass creators, who act as producers of video chunks, and requesters.

3.2 Implementation Details

The system is implemented using Golang version 1.12.3 and comprises several modules. Among them, the Video Stream Processing (VSP) module is responsible for video streaming, while the Peer-to-Peer (P2P) module maintains and manages the worker network. The VSP module leverages the advanced features of FFmpeg (version 2021-12-23, git-60ead5cd68-essential_build), ensuring state-of-the-art video processing capabilities within the system. Further details about the VSP module are provided in Chapter 5.

The P2P module is developed using the Golang-based IPFS client, kubo. Each node within the worker network is equipped with an integrated kubo client, establishing a private IPFS-based network. Additionally, the module facilitates direct information exchange between endpoints through the HTTP protocol, primarily utilized during specific operations, such as the initialization and termination of IPFS services.

Within the system, file resources, such as video chunks, are identified as Tasks, each assigned a unique Task Identifier (TID) generated using the SHA-256 algorithm. Worker nodes are effectively managed using a specialized address format named 'multiaddr,' encapsulating essential information such as the workerID (generated by the SHA-256 algorithm), the protocol in use, and the port number. The module employs a cryptographic algorithm to generate a private key for each workerID, ensuring the verification of authenticity when workers attempt to establish a P2P connection.

In the current implementation, certain functionalities enabling interaction with the public IPFS network are intentionally disabled to enhance security. Specifically, the IPFS Remote Procedure Call (RPC) and IPFS gateway features, common in typical IPFS deployments, are deactivated. This proactive measure strengthens the system's security framework, preserving the private network's insulation from potential vulnerabilities associated with public network interfaces.

Chapter 4

Worker Network

4.1 Overview

In the proposed system, user-generated videos converted into the HLS format by the creators are distributed and cached on the worker network in a distributed manner. Video chunks cached on the network are requested by users who wish to view the video, and workers holding the video chunks transfer them to the requester. The main role of the worker network is to perform the above process efficiently without delay, and for this purpose, it effectively utilizes mechanisms provided by the IPFS, such as distributed hash tables (DHTs) and Bitswap. In this section, we provide an overview of kad-DHT, a concrete DHT employed in the proposed method, followed by the description of the Pubsub model used to distribute video chunks and a resource swap mechanism responsible for the discovery and transfer of cached chunks.

4.2 kad-DHT as an Underlying Overlay

The worker network incorporates a distributed hash table known as Kademlia (kad-DHT) within its logical structure. Kademlia is a peer-to-peer network designed for storing key-value pairs, where the key is typically a hash value, and the value is a record representing the network location of the resource corresponding to the hash value. The address space of kad-DHT spans from 0 to $2^{256} - 1$, with peers and records mapped to this range using SHA256. Each record is stored on the nearest peer in the address space, and the connection between adjacent peers in the kad-DHT is established through an overlay that mimics the skip list, enabling a quick lookup in $O(\log N)$ hops, where N represents the number of peers in the kad-DHT. The logical structure of kad-DHT is illustrated in Figure 4.1. In the graph, the black nodes maintain K connections at each of the circled nodes, called $K - buckets$. Where K is a value that varies with the state of the system

kad-DHT provides support for the joining and leaving of peers. In the proposed system, a designated node called the coordinator acts as a bootstrap server, ensuring correct maintenance of the overall network structure as long as each worker appropriately executes the join/leave procedure of the kad-DHT. However, in real network environments, workers joining the kad-DHT may leave unexpectedly, temporarily turn off device, or delete cached chunks or critical information in the routing table. Such a discrepancy can result in the coordinator's understanding of the network structure not aligning with the actual state, leading to malfunctioning join procedures and chunk retrievals.

To address this issue, the system delegates the coordinator with the authority to centrally restore the entire network structure. The restoration process aims to reconstruct the complete DHT utilizing only the currently online workers in the system. The worker network is structured to enable the coordinator to receive reports on the success or miss of queries executed on the DHT. In this context, a query miss indicates that the resource requested by a worker cannot be found within a predefined timeout period,

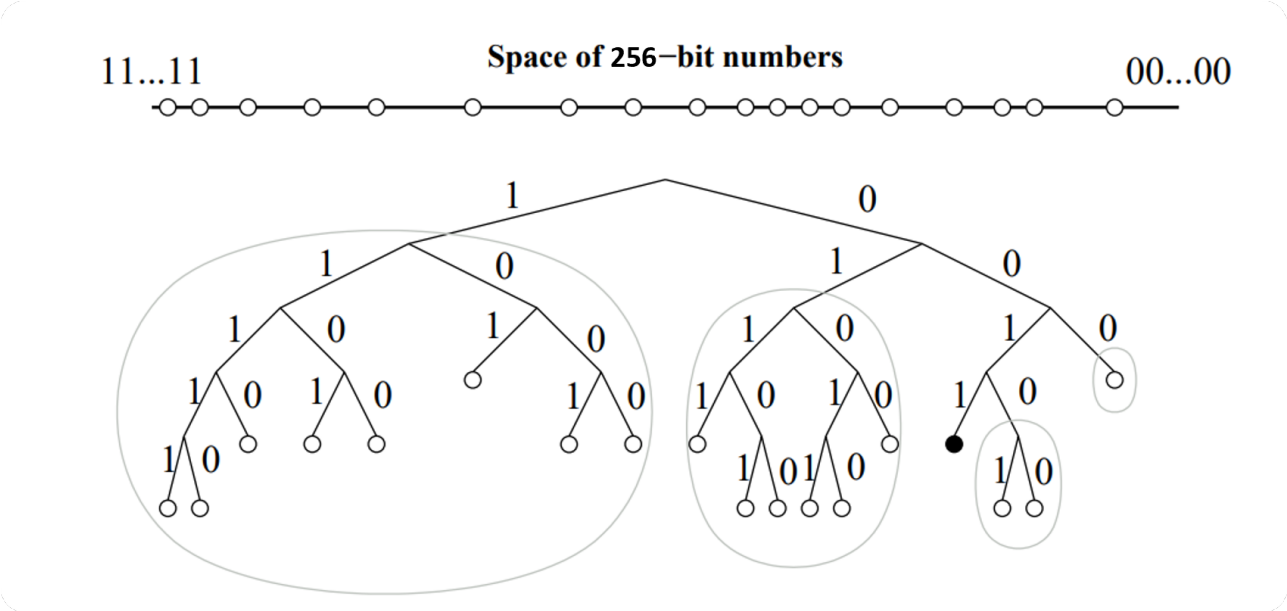

Figure 4.1: Logical structure of P2P overlay used in kad-DHT. [16]

typically due to node departure and/or cache erasure.

The coordinator triggers the restore procedure when the percentage of query misses since the last restore surpasses 15%, signaling a potential degradation in system performance. Upon completion of the restoration, the coordinator relinquishes its centralized management role, reverting to its normal role as a bootstrap server. This proactive approach ensures the continuous efficiency and robustness of the network structure.

4.3 Peer-to-Peer Module

In the presented system, the maintenance and management of the worker network are orchestrated through a dedicated peer-to-peer (P2P) module. This module encompasses two information exchange mechanisms—Distributed Hash Table (DHT) and the Publish/Subscribe (Pubsub) model—as well as a resource transfer mechanism named resource swap. While this subsection delves into the detailed explanation of the information exchange mechanisms, the subsequent subsection provides an overview of the resource swap mechanism. All these mechanisms are implemented using libraries provided by the IPFS.

Within this system, resources, such as video chunks, are conceptualized as tasks, each assigned a unique Task ID (TID). The DHT functions as a means for workers to locate specific tasks. Precisely, a query to the DHT with the TID as a key retrieves a record containing the address of the worker node responsible for that task. The Pubsub model, another mechanism facilitated by the P2P module, is employed for push-based information exchange. In this model, a message associated with a specific topic is disseminated to surrounding worker nodes upon publication. The message is then forwarded and subscribed to exclusively by worker nodes that have subscribed to that particular topic. This Pubsub model proves highly effective for distributing commands like the service restart directive. The coordinator leverages this mechanism to guide and direct other worker nodes, fostering rational and effective communication within the system.

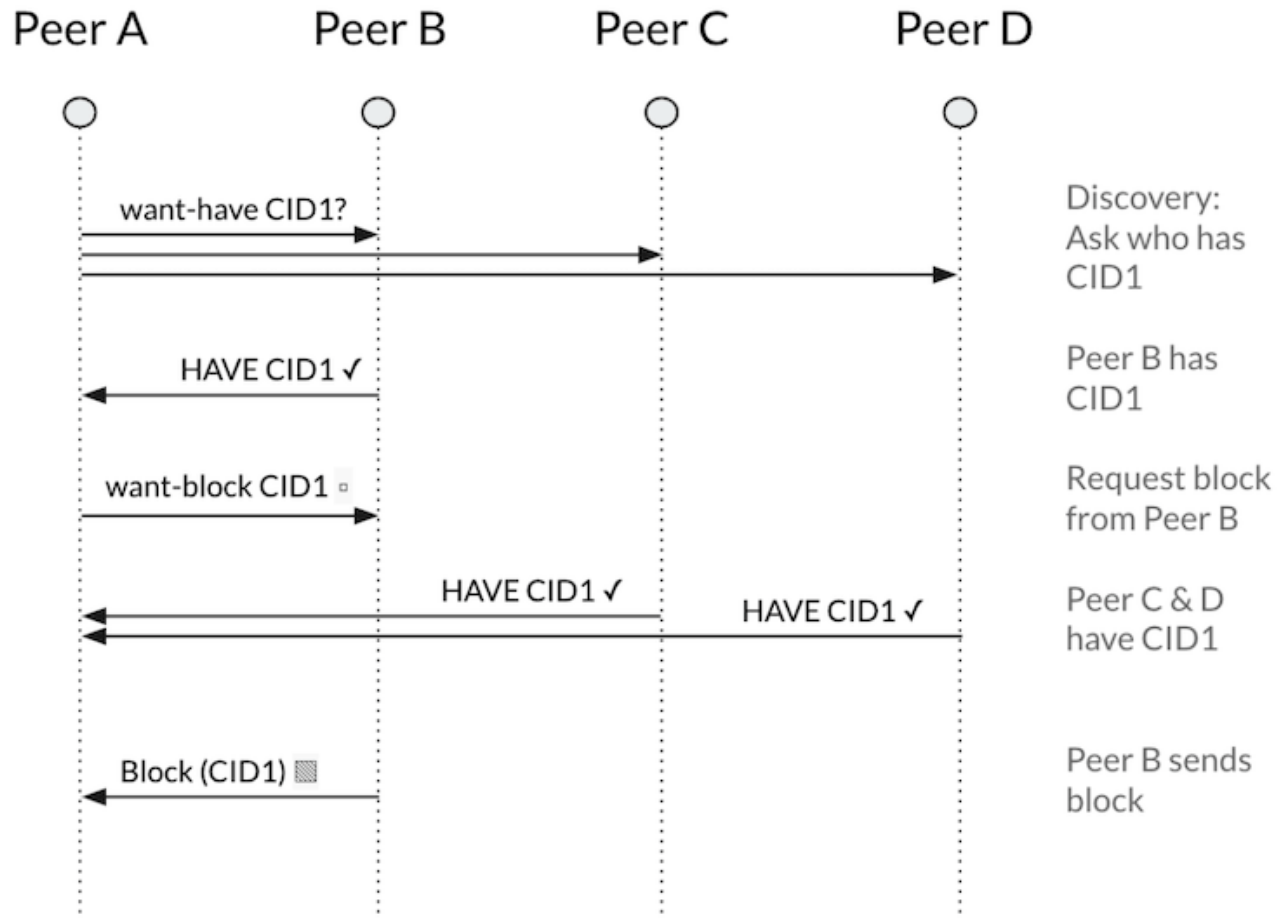

Figure 4.2: Resource Swap Process [17].

4.4 Resource Swap Service for Transferring Segment Files

The system employs a resource swap service, developed based on the Bitswap protocol, a fundamental component of the IPFS, for the distribution and retrieval of video segments. In the resource swap service, file transfer involves dividing each task into smaller units known as blocks. Each block is assigned a unique content identifier (CID) and organized into a Directed Acyclic Graph (DAG) structure. The TID is then derived from the hash value of the string obtained by concatenating all the CIDs in the task.

The primary functions of the resource swap service encompass task acquisition and distribution.As shown in the figure 4.2 ,during task acquisition, the service efficiently obtains video segments requested by clients from available worker nodes in the network. The discovery of a worker node owning a specific file begins with the issuance of a “want-have” request to all known workers. This request includes the CID of the root block of the DAG, and the recipient responds with a “have” message if it owns the root block, entering the subsequent transfer process, or a “dont-have” message if it does not possess the block. The Bitswap service aggregates responses, creating a map indicating which worker owns each block. If the file holder cannot be found through the “want-have” request, the DHT is queried to identify the worker holding the file.

Once the worker holding the desired block is identified, the requester issues a “want-block” request, and the identified worker transfers (the sequence of) requested blocks. Data transfer occurs through socket connections between worker nodes, with multi-addr format used for addressing communication partners. To maximize bandwidth utilization, each worker node can concurrently create up to three socket threads, and each block is transmitted and received as an independent data unit in a predefined socket order. In the event of a connection drop during file transfer, the system responds by

issuing a “want list” of CIDs for the remaining blocks. This list is distributed throughout the system, prompting requests for the retransmission of outstanding blocks.

Chapter 5

Video Stream Processing Module

5.1 Overview of HTTP Live Streaming (HLS)

The proposed system adopts HTTP Live Streaming (HLS) as the video format for optimal compatibility and efficient video delivery. HLS operates by transcoding an input MP4 file into an index file (.m3u8) and multiple segment files (.ts). Each .ts file contains video data, while the .m3u8 file specifies the order, playback time, and delivery format of each .ts file. Acting as a playlist for client-side playback, the .m3u8 file ensures seamless rendering of segment files with different resolutions and bit rates in consecutive order, facilitated by the requirement of consecutive time stamps.

5.2 Generation of Video Chunks

In the envisioned system, user-generated video streams undergo a sequential process involving recording, transcoding, delivery, bundling, and playback. The transcoding step involves converting an MP4 file originating from a video camera or similar device into a sequence of segment files. In compressed video formats like MP4, a video stream is encoded into a series of picture groups (GOPs), each comprising multiple frames and commencing with a key frame referred to as an I-frame.

During transcoding, each segment file encapsulates an entire GOP, potentially causing variations in chunk length compared to the specified duration. Even slight discrepancies, measured in milliseconds, can lead to blank spaces when bundling multiple streams into a single stream—an undesirable outcome. To overcome this issue, the proposed system employs a strategy where the creator inserts keyframes at regular intervals *before* initiating transcoding, ensuring uniform chunk lengths. The keyframe insertion is achieved by resetting the IBP frame sequences using the libx264 library, maintaining the integrity of the video content while addressing the specified problem.

5.3 Distribution of Generated Chunks

The video chunks generated through MP4 transcoding undergo distribution to workers and are automatically cached by the receiving workers. The destinations for chunk distribution are governed by the IDs assigned to the chunks. The implemented prototype system employs specific rules for ID assignment to enhance the efficiency of the subsequent bundling process:

1. Video chunks originating from the same MP4 stream typically share the same ID. However, when the MP4 stream exceeds a predefined length, such as 20 minutes, a new ID is generated and assigned to chunks each time it surpasses this specific duration. This approach prevents an excessive number of chunks from being cached on the same worker, promoting a balanced distribution of chunks from the same video camera across the worker network.

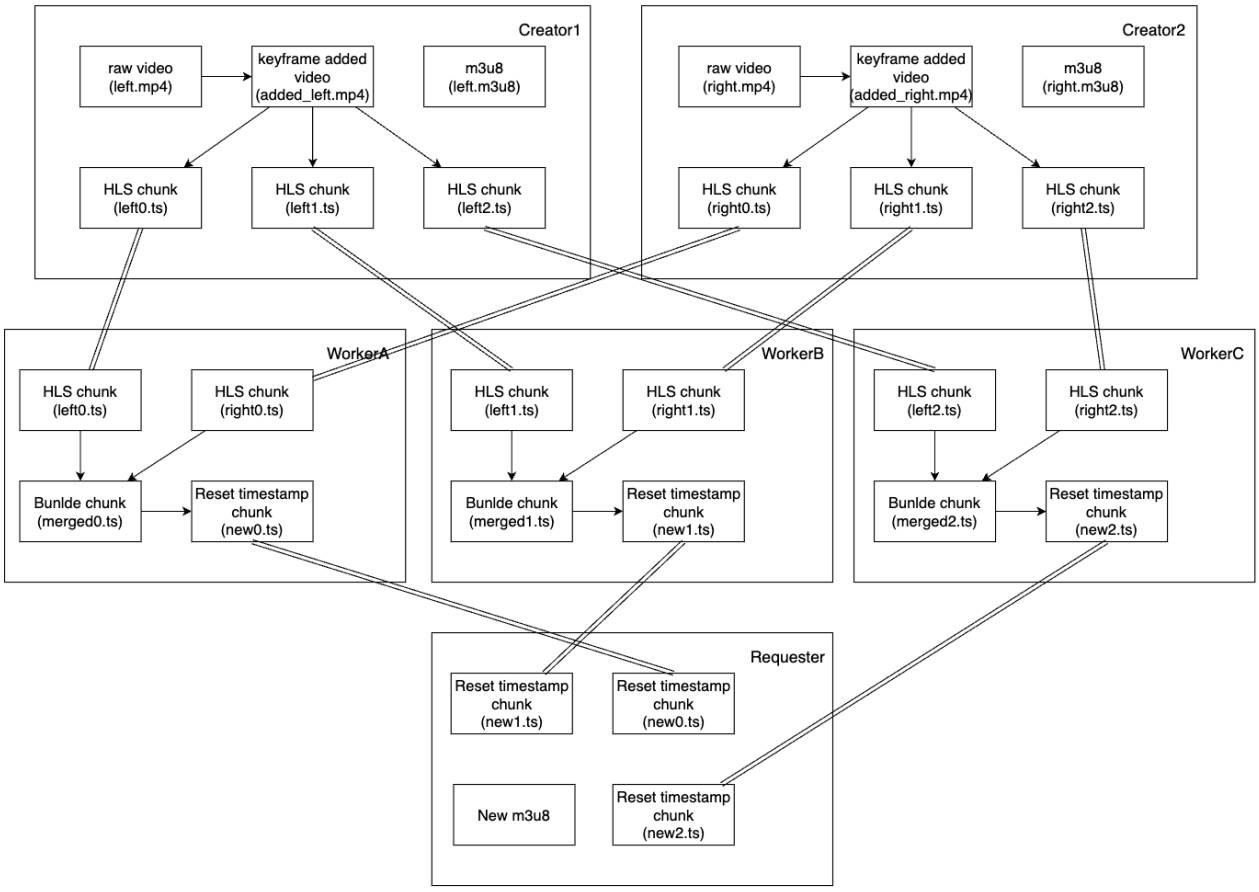

Figure 5.1: Video Stream Process.

2. Video chunks that have the potential to be bundled together are assigned the same ID. In cases where the bundling potential cannot be predicted in advance, these chunks are permitted to be pre-cached on any worker. Upon receiving a request for a bundled stream, relevant chunks are then transferred between corresponding workers. It is crucial to note that in such scenarios, the download time for the bundled stream may increase due to the transfer time.

5.4 Bundling Multiple Video Streams

In this paper, bundling refers to the process of aligning multiple videos either horizontally or vertically to create a unified video. For instance, horizontally bundling two 24-second videos of 480*720 resolution results in a new 24-second video with 960*720 resolution. Figure 5.2 provides a screenshot illustrating the bundling process. The system supports various alignments such as 2*1 (1*2), 2*2, and 2*3 (3*2). While some deviations can be adjusted using FFmpeg's pad filter, it is essential to be mindful that a significant difference in the bit rate or resolution of aligned videos may compromise the naturalness of the resulting video.

In the proposed system, the bundling process is applied directly to segment files, resulting in the generation of new .ts and .m3u8 files, where generated .ts files inherit the timestamp of the input files. Consequently, when a requester seeks a bundled video stream, a fresh .m3u8 file is created, enabling the requester to download and play solely the newly generated segment files, where the responsibility of maintaining the new .m3u8 file lies with the worker who conducted the bundling. The hash value of the freshly generated .ts file is annotated as a comment in the .m3u8 file. This resulting .m3u8 file serves both as a playlist for the bundled video stream and as a torrent file in BitTorrent.

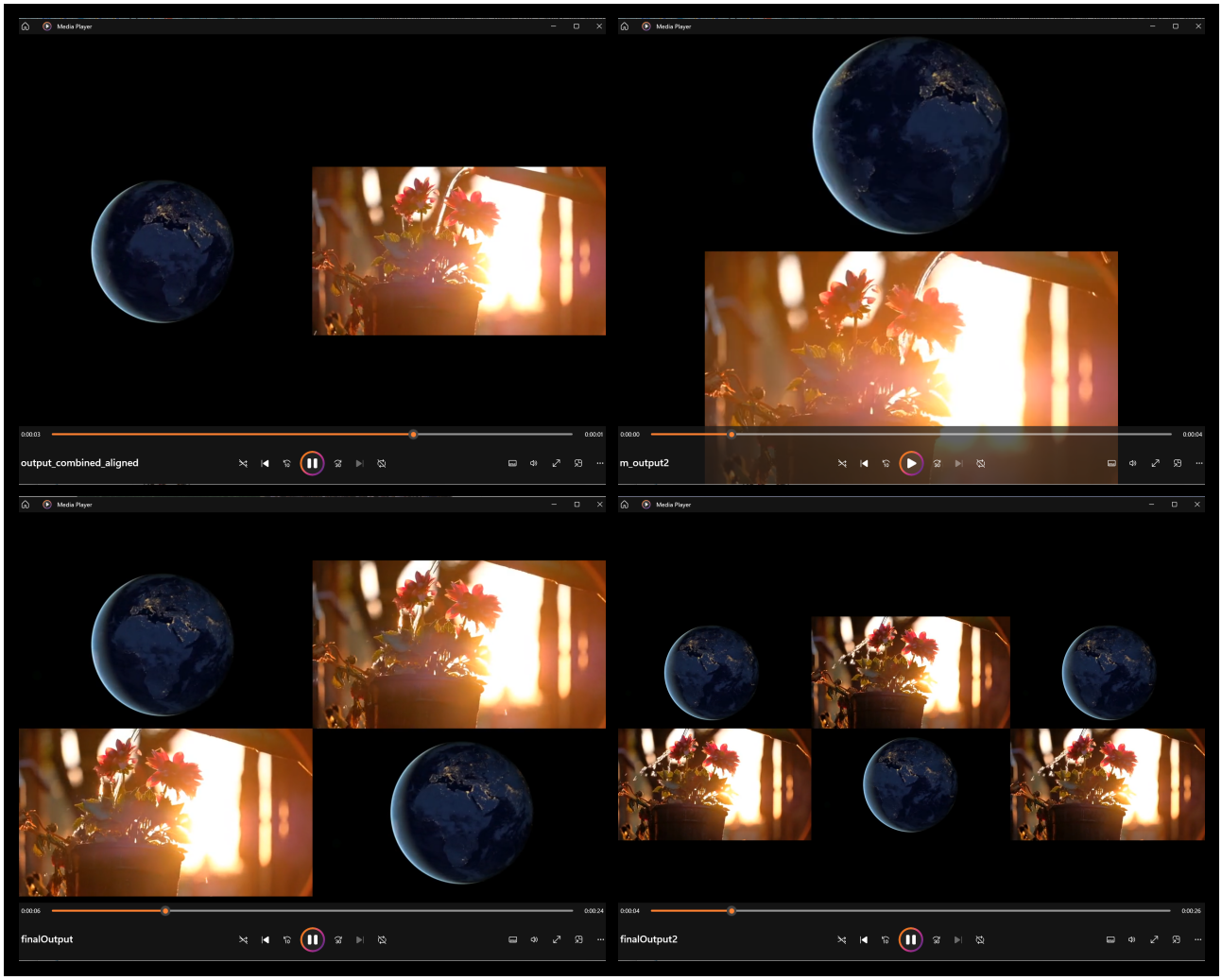

Figure 5.2: Screenshot of video stream.

The bundling process leverages FFmpeg libraries effectively. Specifically, the hstack (or vstack) operation combines frames with minimal frame loss, and the outcome is then rendered onto a larger canvas using the pad filter. The amerge filter is employed to combine audio tracks into a multi-channel stream, which is subsequently converted to stereo.

Chapter 6

Evaluation

6.1 Evaluation Setup

To rigorously evaluate the performance capabilities of the proposed system, a comprehensive suite of experiments was methodically designed and executed. As a benchmark for comparison, the same set of videos was also processed and distributed employing the conventional Client/Server (C/S) model. This comparative analysis aims to provide a clear and objective assessment of the system's efficiency and effectiveness in contrast to traditional methodologies.

6.1.1 Video Set

For the experimental evaluation of the system, six videos, encompassing three distinct resolution types, were selected as test subjects. These resolution categories include 360P, representing low-definition video streams; 1080P, denoting the most commonly used and widespread resolution in contemporary video applications; and 4K, epitomizing high-definition video streams that maintain the original quality of the captured footage. Two videos from each resolution category were incorporated into the experiment, culminating in a total of six test videos. The specific parameters and characteristics of these videos are systematically detailed in the accompanying table for a comprehensive understanding of their properties.6.1

Table 6.1: Video Set

	left.mp4	right.mp4	1080left.mp4	1080right.mp4	4k30left.mp4	4k30right.mp4
Resolution	640*360	640*360	1920*1080	1920*1080	3840*2160	3840*2160
Duration(second)	31	30	36	35	30	30
FPS	30	30	30	30	30	30
Number of chunks	7	7	8	8	7	7
Decoder	MPEG-4 AAC,H.264	H.264	H.264	H.264	MPEG-4 AAC,HEVC	MPEG-4 AAC,HEVC
DataSize(KB)	3042	1990	48737	59035	93842	112563

6.1.2 Evaluation Setup

The evaluation of the proposed system was meticulously conducted by simulating a real-world network environment. This simulation involved the deployment of the client within a Docker container, thereby creating a controlled yet realistic testing scenario.

The Docker host utilized for the simulation is a desktop machine, equipped with an AMD 5600X CPU, 16GB DDR4 RAM, and running the Windows 11 23H2 operating system. Each docker container is limited to a maximum of 3 cpu cores with 6GB of RAM. The Docker image used in the experiment was based on the official Golang image, with FFmpeg and the kubo client installed within it. Each Docker container in this setup is configured to represent an individual worker node in the network.

The network throughput assigned to each worker node is calibrated to mirror the average throughput of a 4G network, approximately 8Gbps, to replicate typical real-world network conditions.

For the purpose of benchmark comparison, a control experiment was conducted using the traditional Client/Server (C/S) model. In this setup, a content delivery server was established using an Azure cloud server located in Tokyo. The server' s configuration included Standard B1s specifications (1 vCPU, 1GiB RAM, and a 30 GiB SSD), and it boasted a bandwidth of approximately 328 Mbps. Additionally, a desktop machine with similar specifications as the Docker host (AMD 5600X CPU and 16GB DDR4 RAM) was deployed as a video processing server. This configuration ensured negligible latency between the video processing server and the content distribution server. For client-side operations, several devices running either MacOS or Windows were utilized to send and request video streams to and from the server.

The system repeatedly runs the complete video distribution-processing-downloading process in the simulation6.1 and benchmarking environments6.2, recording its consumption time.

This comprehensive evaluation approach, encompassing both the simulation of a network environment and the deployment of a traditional C/S model for benchmarking, provides a robust framework for assessing the performance and efficiency of the proposed system in a controlled, yet realistic, network setting.

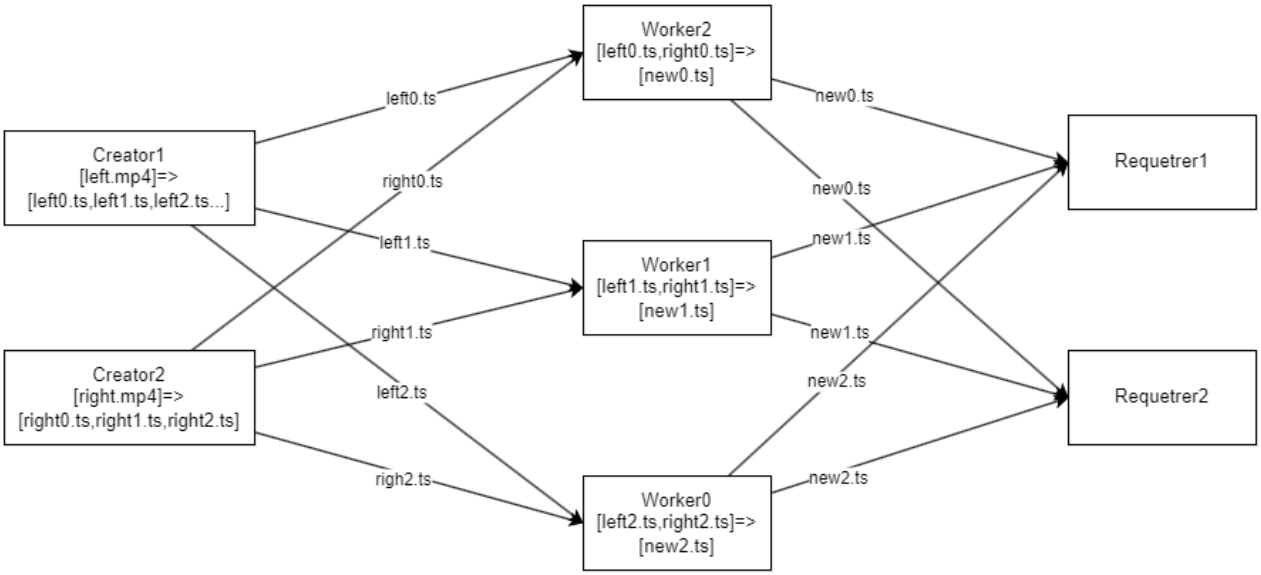

Figure 6.1: P2P Evaluation Process.

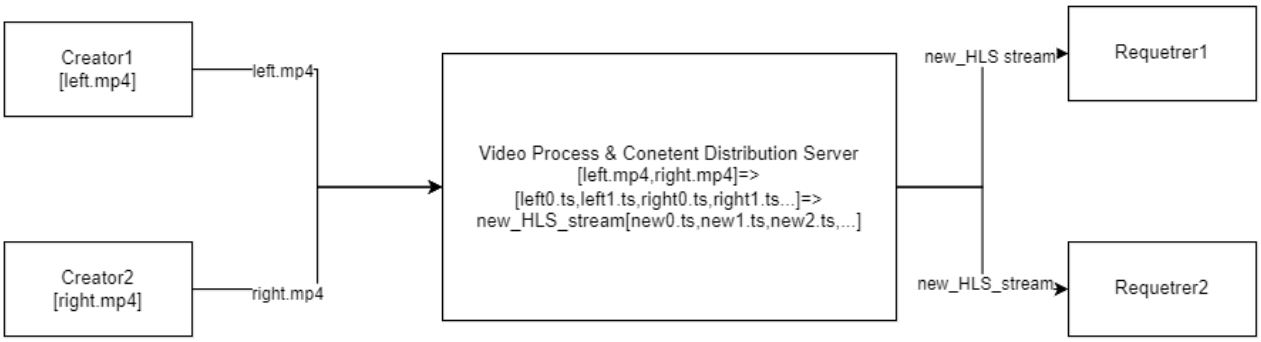

Figure 6.2: Benchmark Evaluation Process.

6.2 Results of Experiment

In the conducted experiments, all videos were generated by two designated creators. The experimental setup was structured to evaluate the processing and storage module under varying conditions, utilizing different numbers of worker nodes. Specifically, experimental groups were established with 3, 8, 12, and 50 worker nodes to represent distinct scenarios: the minimal worker setup, the optimal processing efficiency scenario (where each worker is responsible for handling just one pair of blocks), a configuration with some degree of redundancy, and a scenario featuring significant redundancy. In all these cases, every worker node was actively engaged in the processing and storage of video chunks.All workers perform a 2*1 binding process on the received chunks.

On the receiver side, the number of requester was carefully chosen to simulate user load, with groups of 2, 8, 12, and 50 requester representing different user count scenarios. This approach was intended to assess the system's performance under varying levels of demand and to understand how the number of requester impacts the overall system efficiency and effectiveness.

The results of these experiments are as follows,all time units are in seconds:

In system, the video processing time within the system is defined as the duration necessary to complete two critical operations: transcoding MP4 files into HLS streams at the creator level, and bundling multiple chunks at the worker nodes to generate new video streams. Given that both the conversion of video streams and the bundling of chunks are distributed processes executed concurrently across multiple workers in the system, the overall video processing time is determined by aggregating the two most time-consuming operations. Specifically, it is the sum of the longest duration taken for video stream conversion and the longest duration required for the bundling of chunks among all workers.

Table 6.2: Video processing time

Process Unit	3Workers	8Workers	12Worker s	50Workers	Server
360P Convert	2.5	2.5	2.5	2.5	3.77
360P Bundle	2.7	1.33	1.36	1.29	3.58
1080P Convert	31.5	31.5	31.5	31.5	16.17
1080P Bundle	20.14	10.1	10.12	10.08	23.66
4k Convert	134	134	134	134	71.99
4K Bundle	101.21	53.22	53.52	53.48	104.70

In traditional server-based architectures, the predominant time-consuming operations are typically associated with the uploading of recorded videos to the server and the subsequent downloading of the processed video streams. In the architecture of the proposed system, analogous processes are observed to be the primary consumers of time. Specifically, the transfer of HLS chunks from the creator to the worker nodes, which are responsible for storage and bundling, constitutes a significant portion of the processing time. Additionally, the time taken for requesters to download the content from the workers also represents a substantial part of the overall time expenditure. These operations, integral to the system' s functionality, mirror the upload and download processes in conventional server setups, albeit within the distributed framework of the proposed system.

6.3 Analysis of Experiment

6.3.1 Video Processing Efficiency

The analysis6.2 reveals a notable trend in the video processing segment of the system. It was observed that with a sufficient number of worker nodes, there is a significant decrease in the time

Table 6.3: Video transmission time

(a) 360P video transmission time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	5.16	5.47	5.60	5.97	5.66
8Requesters	10.09	5.69	5.00	5.16	6.02
12Requesters	13.62	7.01	5.11	5.33	6.04
50Requesters	47.68	19.76	14.18	5.34	6.27

(b) 1080P video transmission time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	125.78	130.30	128.71	126.86	125.06
8Requesters	241.98	129.69	124.90	128.81	130.54
12Requesters	334.31	160.71	129.13	127.37	126.22
50Requesters	1201.07	489.93	345.12	127.56	142.69

(c) 4K video transmission time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	256.87	260.29	262.97	265.94	260.20
8Requesters	511.12	259.72	264.92	258.99	259.27
12Requesters	708.61	337.22	266.13	261.06	257.98
50Requesters	2595.20	1042.88	734.58	258.71	294.27

Table 6.4: Total time

(a) 360P video total time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	10.36	9.30	9.46	9.76	13.01
8Requesters	15.29	9.52	8.86	8.95	13.37
12Requesters	18.82	10.84	8.97	9.12	13.39
50Requesters	52.88	23.59	18.04	9.13	13.62

(b) 1080P video total time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	177.42	171.90	170.33	168.44	164.89
8Requesters	293.62	171.29	166.52	170.39	170.37
12Requesters	385.95	202.31	170.75	168.95	166.05
50Requesters	1252.71	531.53	386.74	169.14	182.52

(c) 4K video total time

Process Unit	3Workers	8Workers	12Workers	50Workers	Server
2Requesters	492.08	447.51	450.49	453.42	436.89
8Requesters	746.33	446.94	452.44	446.47	435.96
12Requesters	943.82	524.44	453.65	448.54	434.67
50Requesters	2830.41	1230.10	922.10	446.19	470.96

required for video processing. This trend, however, plateaus when the number of workers reaches or exceeds the total number of video chunks.

The system demonstrates a marked advantage over traditional server models, particularly at lower resolutions. This enhanced performance is attributed to the ability of the worker nodes to easily handle videos of lower resolution and the benefits of parallel processing afforded by the distributed nature of the system. As the resolution increases, the proportion of time spent on bundling within the system shows a decline - from 34% at 360P resolution to 28.5% at 4K resolution. This suggests that as video resolution increases, the impact of bundling on the overall processing time diminishes.

6.3.2 Video Transmission Dynamics

In the video transmission phase6.3, the system's performance is influenced by the ratio of worker nodes to requesters. When the number of workers is greater than or equal to the number of requesters, the limiting factor in transmission speed becomes the download bandwidth of the requesters. Conversely, if there are fewer workers than requesters, the bottleneck shifts to the upload bandwidth of the worker nodes.

Comparatively, in a traditional server setup, the transmission bottleneck depends on the balance between the number of requesters and the server' s bandwidth. When the number of requesters exceeds the server's bandwidth capacity, the server becomes the bottleneck. In contrast, when the server bandwidth is not fully occupied, the limitation lies on the client side.

Significantly, in scenarios where the server bandwidth is fully utilized (as in the case with 50 requesters), the proposed system with an adequate number of workers demonstrates a considerable advantage. This advantage is particularly pronounced in configurations where the requester also functions as a worker node, highlighting the system's efficiency in scenarios with high demand.

6.4 Conclusion

The empirical data gleaned from the experiments underscores the distinct advantages of the proposed system, particularly attributed to its distributed processing architecture. For video streams of low resolution, the system exhibits a pronounced superiority. This advantage is anticipated to be even more significant in the case of longer-duration video content.

Furthermore, the system demonstrates a clear edge in scenarios characterized by a high volume of requests. In such high-demand situations, the distributed nature of the system efficiently manages the increased load, outperforming traditional centralized server models.

In scenarios that do not involve high-resolution videos or an elevated number of requests, the system still achieves performance comparable to that of high-performance servers. Remarkably, this is accomplished using devices with lower performance capabilities and limited bandwidth. This outcome not only highlights the efficiency of the system but also showcases its sophisticated design. The system effectively leverages the strengths of distributed computing to deliver robust performance, even in environments with constrained hardware resources.

Chapter 7

Future Work

In the research presented, the system has exhibited commendable performance in simulated environments, particularly in scenarios characterized by high demand for low-resolution video content. Looking forward, the focus of future work will be on broadening the experimental scope in two principal areas: diversifying the range of video content used in the experiments and extending the testing into real-world network environments.

Notably, the system has shown promising results in preliminary tests with 30-second video clips. It is anticipated that this performance could be further enhanced when applied to longer video duration, offering a more rigorous test of the system's capabilities.

The current experimental setup, conducted using Docker containers, effectively replicates typical computational and network conditions. However, real-world network environments present a more complex and varied landscape. Mobile devices, which are ubiquitously active, often face limitations in terms of computational power and network connectivity. Conversely, stationary devices like desktop PCs, while typically possessing higher performance capabilities and more stable network connections, may have less availability for running the system. Therefore, conducting experiments in such real-world scenarios is imperative to gain a holistic understanding of the system' s performance.

This shift to real-world testing will enable a more comprehensive evaluation of the system, taking into account the diverse range of devices and network conditions encountered in everyday use. Such an assessment is crucial for understanding the system's effectiveness in practical, everyday applications and for identifying areas for further optimization and development.

Bibliography

[1] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C. Begen, and David Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[2] Ricky K. P. Mok, Edmond W. W. Chan, and Rocky K. C. Chang. Measuring the quality of experience of http video streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 485–492, 2011.

[3] Ricky K.P. Mok, Edmond W.W. Chan, Xiapu Luo, and Rocky K.C. Chang. Inferring the qoe of http video streaming from user-viewing activities. In *Proceedings of the First ACM SIGCOMM Workshop on Measurements up the Stack*, W-MUST '11, page 31‑36, New York, NY, USA, 2011. Association for Computing Machinery.

[4] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[5] Alex Zambelli. Iis smooth streaming technical overview. *Microsoft Corporation*, 3(40), 2009.

[6] About the common media application format with http live streaming hls. `https://developer.apple.com/documentation/http-live-streaming/about-the-common-media-application-format-with-http-live-streaming-hls`. Accessed:2024-1-26.

[7] Tz-Heng Hsu and Yao-Min Tung. A social-aware p2p video transmission strategy for multimedia iot devices. *IEEE Access*, 8:95574–95584, 2020.

[8] Dan Jurca, Jacob Chakareski, Jean-Paul Wagner, and Pascal Frossard. Enabling adaptive video streaming in p2p systems [peer-to-peer multimedia streaming]. *IEEE Communications Magazine*, 45(6):108–114, 2007.

[9] Yong Liu, Yang Guo, and Chao Liang. A survey on peer-to-peer video streaming systems. *Peer-to-peer Networking and Applications*, 1:18–28, 2008.

[10] Kunwar Pal, Mahesh Chandra Govil, and Mushtaq Ahmed. Priority-based scheduling scheme for live video streaming in peer-to-peer network. *Multimedia Tools and Applications*, 77:24427–24457, 2018.

[11] Naeem Ramzan, Hyunggon Park, and Ebroul Izquierdo. Video streaming over p2p networks: Challenges and opportunities. *Signal Processing: Image Communication*, 27(5):401–411, 2012.

[12] Sabu M Thampi. A review on p2p video streaming. *arXiv preprint arXiv:1304.1235*, 2013.

[13] Ala'a Al-Habashna, Gabriel Wainer, and Stenio Fernandes. Improving video streaming over cellular networks with dash-based device-to-device streaming. In *2017 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 1–8, 2017.

[14] Roberto Roverso, Riccardo Reale, Sameh El-Ansary, and Seif Haridi. Smoothcache 2.0: Cdn-quality adaptive http live streaming on peer-to-peer overlays. *Proceedings of the 6th ACM Multimedia Systems Conference*, 2015.

[15] João A Silva, Filipe Cerqueira, Hervé Paulino, João M Lourenço, João Leitão, and Nuno Preguiça. It' s about thyme: On the design and implementation of a time-aware reactive storage system for pervasive edge computing environments. *Future Generation Computer Systems*, 118:14–36, 2021.

[16] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[17] Bitswap. `https://docs.ipfs.tech/concepts/bitswap`. Accessed:2024-1-26.