CS237 Project
Interactive Movie Streaming Service

Wei Pan, panw4@uci.edu
Yi Zhou, zhouy46@uci.edu
Xiaomi Liu, xiaoml6@uci.edu

1. Introduction

Video has been an important media for communications and entertainment for many decades. There is a very diverse range of different video communication and streaming applications. For example, in some applications, video may be pre-encoded (stored) or may be encoded in real-time. In this project, we will concentrate on the pre-encoded video streaming. VOD (Video-On-Demand) streaming applications, such as YouTube and Netflix, are the most representative platforms. In these applications, because videos exist far in advance before the users view them, they can be preprocessed at will. They can use heavy compression and prepare the video for different qualities and transmission rates. Also, they can be streamed through adaptive streaming with relative ease. Also, they rely on buffering to provide a greater quality despite network issues, and to be able to-use a larger compression frame.

Beside a basic streaming platform, inspired by Netflix's movie *Black Mirror: Bandersnatch*, we want to provide interactive movie streaming service. *Black Mirror: Bandersnatch* is an interactive film in the science fiction anthology series *Black Mirror*, produced by Netflix in 2018. The audience can make decisions for the main characters and change the story. That is, they can choose the video clips of the movie to watch. The video streaming service may, in our view, suffer from a performance drop when a large portion of the audience make the same choice (choose the same video clip to stream) at the same time.

The objective of this project is to implement a video streaming service that:
(1) provide a basic video streaming function,
(2) dynamically redistribute the video clips in caches based on the user requests for fast retrieval, and
(3) provide an interactive user-interface that allows the users to make selections of the video.

2. Related Works

J. G. Apostolopoulos, W. Tan and S. J. Wee have summarized the fundamental concepts in video streaming, including forms of video communication, video storage, video compression, media streaming protocols, communication channels, transmission rate and so on [1].

2.1 Transmission Protocols

J. G. Apostolopoulos and his colleagues only considered streaming media over TCP and UDP. These days, HTTP adaptive streaming is becoming popular for video delivery because it dynamically provides high quality content. Thomas Stockhammer gives us an example of a possible media distribution architecture for HTTP-based streaming [2], as shown in Fig. 1.

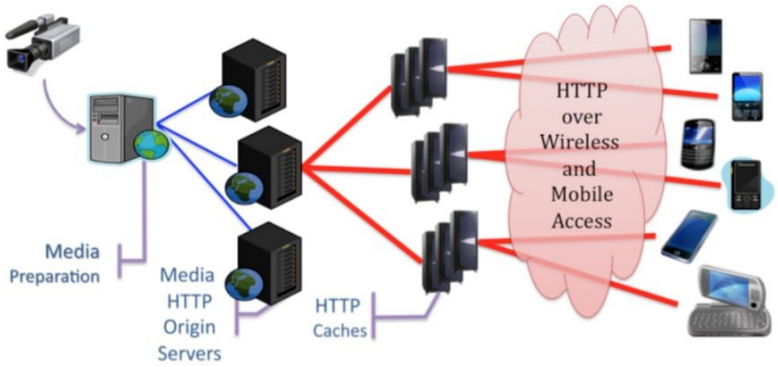

Fig. 1 Example Media Distribution Architecture

The media preparation process typically generates segments that contain different encoded versions of one or several of the media components of the media content. The segments are then hosted on one or several media origin servers typically, along with the media presentation description (MPD). The media origin server is preferably an HTTP server such that any communication with the server is HTTP-based. Based on this MPD metadata information that describes the relation of the segments and how they form a media presentation, clients request the segment using HTTP GET or partial GET methods.

In this project, we choose HTTP as the delivery protocol for our streaming service for several reasons: (1) HTTP streaming is spreading widely as a form of multimedia delivery of Internet video. (2) HTTP-based delivery provides reliability, deployment simplicity and the ability to use standard HTTP servers and caches to deliver the content. (3) HTTP-based delivery provides the ability to move control of "streaming session" entirely to the client and the client can automatically choose initial content rate to match initial available bandwidth without requiring the negotiation with the streaming server. (4) HTTP-based streaming provides a simple means to seamlessly change content rate on-the-fly in reaction to changes in bandwidth, and has the potential to accelerate fixed-mobile convergence of video streaming services.

2.2 Rate Adaption

The network resources and end-to-end bandwidth are varying and sometimes whimsical. In order to prevent client buffer from under-flowing or over-flowing and improve the user experience of multimedia streaming services, C. Liu et al. propose a receiver-driven rate adaptation algorithm for adaptive HTTP streaming [3]. For deciding switch-up or switch-down operations between different representations, a smoothed HTTP/TCP throughput measurement method is presented that compares

the segment fetch time with the media playback time contained in that segment shortly media segment duration. When probing the spare network capacity, a step-wise switch-up method is used to switch to a higher representation. Upon detecting network congestion, an aggressive switch down method is deployed to prevent playback interruptions. Possible switch up and switch down operations are assessed each time after receiving a media segment. In order to save network bandwidth and memory resources for the users, a method for determining the idle time between two consecutive requests for media segments is deployed, thus limiting the maximum amount of media pre-fetching.

The first generation of ABR techniques strictly relied on throughput estimation for the video rate selection decisions. More recently, throughput-based mechanisms have been enhanced and other parameters such as buffer occupancy, power level, cost etc. are also being used in the decision process [4]. All these different approaches try to satisfy a set of general requirements [5], such as ensuring video is streamed with minimal number rebuffering events, maximizing both the minimum and the average video quality level, and ensuring a consistent experience by minimizing the number of quality level switches.

2.3 Storage and Architecture

C. Federighi and L. A. Rowe from UC Berkeley prompts a novel hierarchical storage manager for distributed caching of videos on the file servers and scheduling load requests to the tertiary devices [6]. Berkeley Distributed VOD System provides some good intuitions for dealing with video clips. For the flexibility of extracting small segments from a large movie for playback or incorporation into other documents, the system provides Compound Media Object (CMO), which allows a movie to be composed of many subjects, which may in turn be shared with other movies. We think such data structure might bring us much convenience for our project, since an interactive movie is composed of many small video clips. Moreover, their system has an intelligent cache management technique. As we know, the success of any caching scheme depends on the ability of the system to predict future access patterns and keep frequently accessed data available in high speed storage. The system employs simple cache replacement policies such as least recently used (LRU) or random replacement. We will adapt this technique in our project by using detailed access statistics and user caching directives to improve cache performance, for example, the weight we give to a video clip will become higher when more people watch it.

VMesh is a distributed segment storage for peer-to-peer interactive video streaming prompted by W. P Len Yiu et al [7]**.** In VMesh, videos are divided into segments and stored at peers' local storage in a distributed manner and an overlay mesh is built on peers to support random forward/backward seek, pause and restart during playback.

Hadoop distributed file system (HDFS) [8] stores file system metadata and application data separately. The metadata is stored on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. Each cluster has a single NameNode and multiple DataNodes and clients. Each file's content is broken down into chunks of size 64-128MB and is stored on multiple DataNodes, three by default. The NameNode maintains the mapping of the file blocks in the DataNodes. All servers are fully connected and communicate with each other using TCP based protocols. HDFS replicates data on multiple DataNodes. This creates more opportunities for locating

computation near the needed data. For critical files or files which are accessed very often, having a higher replication factor improves their tolerance against faults.

As mentioned above in the HDFS section, file contents are broken down into chunks of 64-128MB as default. When the file is much smaller than 64MB but in large quantity, e.g. text file, the name node will get overloaded due to the huge namespace. Moreover, HDFS lacks the ability to efficiently support the random reading of small files because of its high capacity design. HBase [9] is a distributed column-oriented data store built on top of HDFS. It is ideally suited for random write and read of data that is stored in HDFS and allows for dynamic changes in storage.

3. Implementation

3.1 Video streaming

The video data is stored on server using HDFS. We use servlet in Java to receive and respond to requests from web clients. Since we use the built-in player HTML 5, our servlet now supports only several video formats, like MP4, OGG and WebM. For simplicity, all the video clips in our project are in MP4 format. A unique ID is assigned to mapped with each video clip. Whenever a request is received by the server, it searches its cache for that video clip ID. If not found, it will load from HDFS. The structure of our video streaming service is illustrated in Fig. 2.

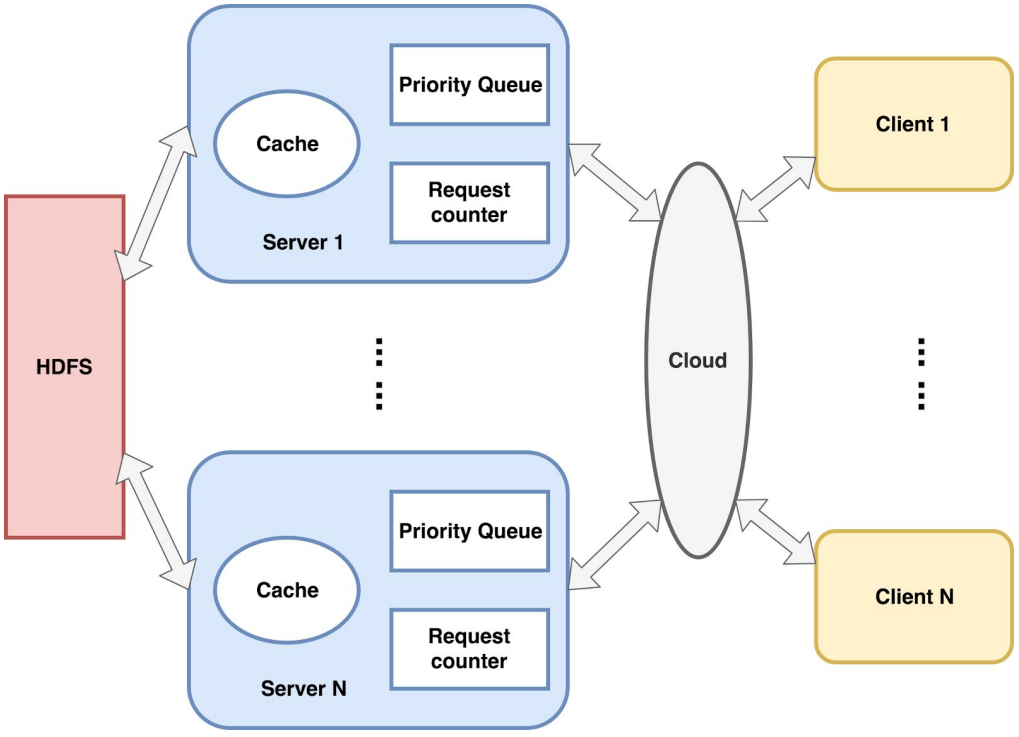

Fig. 2 The layout of interactive video streaming service

3.2 Dynamic caching

Since retrieving data from HDFS requires several mili-seconds, we use caching to speed up the loading of most frequently requested video clips. For each video clip, we keep track of the number of

times it is requested by the user in the server and keep a priority queue of video clip IDs with top 5 number of requests. The video clips with top 5 requests are preloaded in the cache of the server. Everytime a requested received, we update the play-count of the video clip as well as the priority queue: drop the video clip that is not frequently played and add in a new top 5.

If the number of requests are the same for two video clips, their sizes are compared to determine the priority. The video clip with larger size has higher priority. We do this for two reasons: firstly, there is a small chance that two video clips have identical size in bytes, which makes it easy for us to decide which one has higher priority; secondly, the larger file takes longer to load. Preloading it in cache will improve the performance.

And of course, the starting video clip of the whole movie will always be stored in cache, because it is a must play for every user.

3.3 User interface

The user interface is implemented in HTML 5. When a user finish watching a video clip, several choices with hints or descriptions regarding the content of the following videos are presented on the page. The user can make his/her own choices for what to watch next.

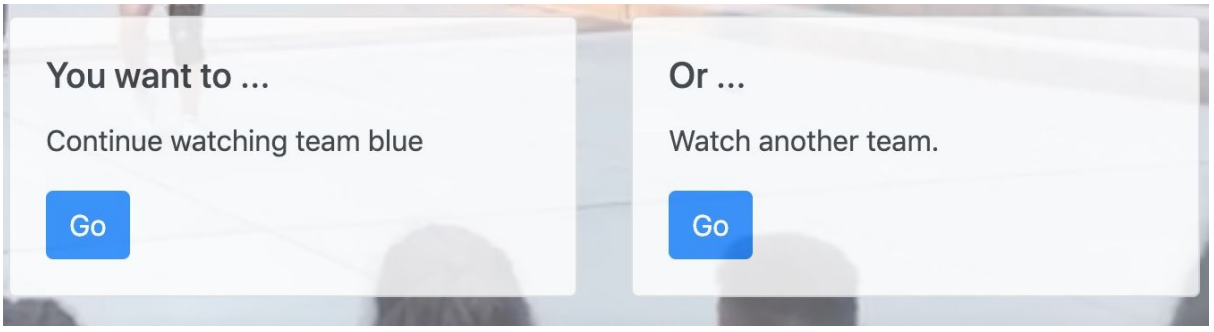

Fig. 3 User interface

As shown in Fig. 3, clicking the different buttons will stream different video clips. When the movie is coming to an end, an “END” button together with a “Back to the beginning” button are given, ending the streaming service or taking the user back to the starting video clip, respectively.

4. Evaluation

We use three MacBook Pros to test our video streaming service, with one of them being the server and the other being the clients. Both clients successfully connect to the web service and can make selections for video clips independently.

We also calculated the video clip loading time for each request from client to evaluate the performance of our dynamic caching approach.

Request No.	Requested video clip ID	Loading time (ms)

1	2	179
2	3	194
3	4	185
4	3	0
5	5	181
6	2	0

Tab. 1 Loading time for a sequence of requests

Tab. 1 shows the loading time calculated for a sequence of requests. As we can see in the table, the video clips being requested for a second time have a dramatic drop in loading time due to caching.

5. Conclusions

We implement an interactive video streaming service that has: (1) basic video streaming capability, (2) dynamic caching to improve the performance and (3) an interactive user interface for the user to make selection for the next video. For dynamic caching, we use an algorithm similar to LRU that keeps track of the most frequently requested video clips and preloads them in cache of the server to reduce the loading time on future requests.

The future extension of this project will be dynamic replication across different servers. With the statistics kept for the frequency of each video clip, we can replicate them on multiple servers according to the frequency. The more servers that have this video clips, the broader bandwidth and less response time the client is going to get for streaming. Due to the limitation of equipment and environment, we have only one computer as server and another two as clients, all of which are connected in a stable WiFi network. Thus, it is not possible to measure the performance gain from the dynamic replication.

6. References

[1] J. G. Apostolopoulos, W. Tan and S. J. Wee, “Video Streaming: Concepts, Algorithms, and Systems”, HP Laboratories, report HPL-2002-260 (2002)
[2] Thomas Stockhammer. Dynamic Adaptive Streaming over HTTP – Design Principles and Standards. In Proceedings of the second annual ACM conference on Multimedia systems, pp. 133-144 (2011)
[3] C. Liu, I. Bouazizi and M. Gabbouj, “Rate Adaptation for Adaptive HTTP Streaming”, Proceedings of the second annual ACM conference on Multimedia systems, pp. 169-174 (2011)
[4] G. Tian and Y. Liu, “Towards agile and smooth video adaptation in dynamic HTTP streaming,” Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol., pp. 109–120 (2012)

[5] Y. Sani, A. Mauthe and C. Edwards, “Adaptive Bitrate Selection: A Survey”, IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2985-3014 (2017)
[6] C. Federighi and L. A. Rowe, “Distributed hierarchical storage manager for a video-on-demand system”, Storage and Retrieval for Image and Video Databases II, International Society for Optics and Photonics, vol. 2158, pp. 185-198 (1994)
[7] Yiu W P K, Jin X, Chan S H G, “VMesh: Distributed segment storage for peer-to-peer interactive video streaming”, IEEE Journal on Selected Areas in Communications, vol. 25, pp. 1717-1731 (2007)
[8] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The hadoop distributed file system", IEEE 26th Symposium on Mass Storage Systems and Technologies, pp. 1-10 (2010)
[9] M. N. Vora, "Hadoop-HBase for large-scale data", Proceedings of International Conference on Computer Science and Network Technology (2011)