Scalable Video-on-Demand With Edge Resources

Jiayuan Chen

Capstone Report

Scalable Video-on-Demand With Edge Resources

Jiayuan Chen

A paper submitted in partial fulfillment of the
University of California, Berkeley
requirements of the degree of
Master of Engineering
in
Electrical Engineering and Computer Sciences

April 2016

Contents

Chapter 1

Technical Contributions

1 Introduction

1.1 Project Description

Video-on-demand (VoD) is a fast growing industry around the world. Thanks to the advancement of Internet technology, more people are having easy access to network resources that could support online video streaming. As a result, great innovations have been made in VoD technology in recent years. Internet exchanges and data centers built by large VoD providers such as Netflix, Amazon, and Google are exploiting the reliability and economies of scale from the centralized server architecture (Lee et al. 2012, Lee et al. 2013). The bottleneck of this architecture lies in the bandwidth of the server that needs to handle massive traffic from large number of users. On the other hand, the edges of the internet (tablets, laptops, smartphones) are growing at a astonishing speed (Lee et al. 2012). Therefore, it is possible to exploit the resources in those edges devices, to maximize the non-server traffic usage, and reduce the server bandwidth. A highly distributed edge-based VoD architecture also tolerates the unreliability of the network. The server still keeps the original data, so that a disruption in the network can be recovered by data relocation. In addition, the product that built on this system architecture will be popular in regions where connectivity resources are limited, such as India and China. Startups can use our technology to provide high quality VoD contents even without a reliable and expensive infrastructure.

In this capstone project, we developed a video-on-demand (VoD) system that delivers video content in a highly robust and distributed way. Different from traditional VoD technologies that contain the simple server-user architecture, our project utilizes caches in edge devices in a

P2P network. The caches will act as mini-servers that continuously download small chunks of videos in the network, and upload them to other users, which effectively reduce the server load and increase the overall scalability. The theory and algorithms for video content distribution have previously been developed by our advising laboratory (Berkeley Audio-visual Signal processing and Communication Systems Group). These algorithms define and regulate the interactions between nodes in the system, in such a way that everyone's resources are maximally utilized. The capstone team was expected to deliver a fully-integrated application built on this theoretical framework.

Based on the simple prototype already built by our predecessors, we've designed the data flow in the network, increased the robustness of the system by adding functionalities, scaled up the system onto public cloud, and created better user experiences with graphical user interfaces (GUI) on desktop and web browser.

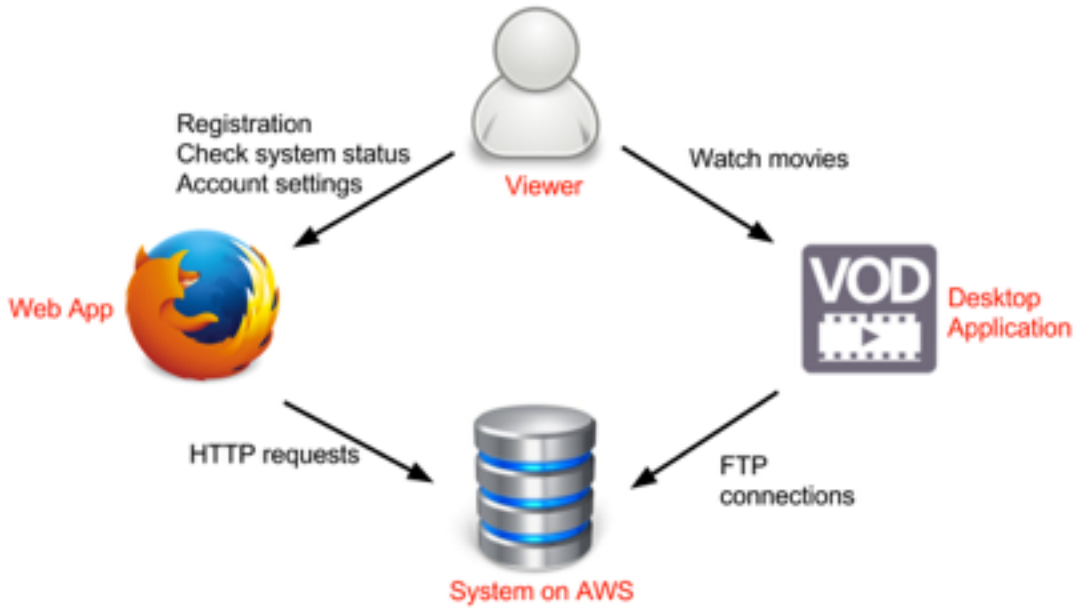


Figure 1. components of the project deliverable (system on AWS, web app, desktop app).

In terms of product, our final deliverable consists of three major components: A server running on Amazon Web Services (AWS) that is able to handle huge traffic; a desktop application (GUI) for users to login, browse videos, and configure their cache settings; a web application that is not only a hub for users registration, but also a dashboard to monitor the real-time status of the system. Figure 1 above shows how these components are connected in our system. Breakdown structure of associated tasks will be discussed in following section. This paper will mainly focus on technical details with which I've been heavily involved, as a complement to other two teammate's papers.

1.2 Associated Tasks & Work Breakdown Structure

The broad scope of this project consists of numerous tasks and challenges throughout the development, from code design to market analysis. Therefore, the capstone team roughly split up the workloads and each was assigned tasks in various areas. The analysis of each accomplishment will be distributed in three final project papers.

In order to better organize the structure of the reporting deliverables, this project can be broken down to six milestones, each are built up with related tasks (marked with [H], [C], and [K], indicating that the task will be discussed in Alagu Sanjana Haribhaskaran's paper, Jiayuan Chen's paper, and Ryan Kashi's paper respectively).

I. Build the system that runs the video distribution algorithm on one computer locally

- System design: Tracker, Server, Cache, User [C]
- Database that keep tracks of system state [C]

- Preprocessing and encoding the videos into chunks [K]
- Migration from .flv to .mkv format for stored videos [K]
- Use Pyftpdlib (a FTP library) to transfer data between nodes [C]
- Shell scripts and downloaded file managements [C]

II. Reconfigure the system so that it can run publicly over the internet

- Allow user to dynamically change and access other entities on public domain name [K]
- Running nodes publicly via IPv6 over multiple machines [H]

III. Increase the robustness of the system by adding critical features and increase stability

- Ensure cache and user are successfully removed from the system when disconnected [K]
- Incentive mechanism that encourages users to contribute resources and bandwidths [K]
- Accounts that can be associated with running user and caches [C]

IV. Create graphical user interfaces (GUI) for both desktop app and online app

- Desktop GUI using Python [H]
- Integrate VLC player with the user interface [H]
- Simple online application with Webpy [C]
- Visualization for data flow between users and caches in the network [K]

V. Move the server to a public cloud infrastructure that provides scale up capability

- Cloud basics and migrate the project onto Amazon AWS [H]
- Scaling up the system on AWS [C]

VI. Create viable business model and conduct market research, will be discussed in Chapter 2

2 List of Accomplished Tasks

2.1 System design: Tracker, Server, Cache, User

The design of the system topology is the fundamental building block of this project. Because all other features are added on top of it, this step was conducted with most cautious and effort. The algorithm and a primitive prototype using Python are described in paper co-authored by our grad student advisor (Lee et al. 2012, Lee et al. 2013), and it has been improved since then. The system consists of four major components: *tracker* and *server* that run in the cloud, along with *cache* and *user* that run in each edge device. These components will be referred as *nodes* in this paper. Also note that the notation of *user* is referring to a program instance, not to the real person who watches the video. A real person acts as an account who can run one cache and one user in his/her edge device. Figure 2 below demonstrates the data flow between *tracker*, *server*, *cache*, and *user*.

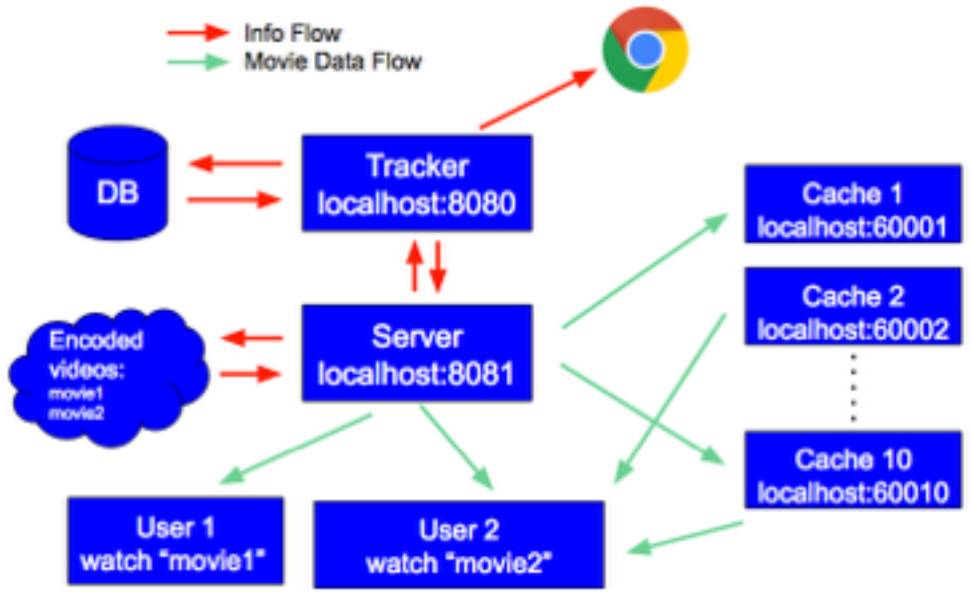


Figure 2. The system design and date flow between nodes (tracker, server, cache, user).

The *tracker* keeps track of the system state by maintaining a database of the real-time data flow. It will update the database every time a new node is connected, a user downloads a chunk of video from the server, or a new video is added to the server. The tracker also runs a web application using Webpy (a minimalist web framework written in Python), which renders a well-designed website for administrators to monitor the system status, and for users to register and check their incentive points. We've also created two visualizations for data flow between cache nodes and user nodes in the tracker, that show which caches are sending data to a particular user, and which users are receiving data from a particular cache.

The *server* runs with the tracker at the same location. It is a FTP server that stores all erasure-coded videos. Whenever a cache or a user joins the network and registers itself into the tracker, the server starts to distribute small chunks of videos to those nodes according to the data distributing algorithm. The basic FTP functionalities are implemented with Pyftpdlib, an open source Python library that provides a portable interface to write efficient, scalable and asynchronous servers.

The *caches* are not only FTP clients, but also mini FTP servers that send data to connected users. The data distribution algorithm will determine which chunks of video will be downloaded by each cache, and how to send them to a user. According to the current design, an account can run 1 cache and 1 user on the edge device it is using. It's not necessarily for a cache to only download video currently watched by the cache owner. Sometimes a cache will download a video chunk only to serve other active users in the network, because that video is popular. By adjusting the cache's configurations (size and bandwidth), the account can gain more incentive points to purchase new videos from the system. The more resources he/she is willing to contribute, the faster the incentive points are accumulated.

Lastly, the *user* is simply a FTP client that connects itself to both the serve and the cache. After the account selects the video to watch, the system will check how the content of that video is distributed over the server and caches. Optimally, the chunks of that video have been fully downloaded by several caches, so that they will be sent to user to reconstruct by caches only, without the help of server. Therefore, the traffic from the server is reduced to nearly 0%.

2.2 Database that keep tracks of system state

In order to track the real-time state of the system, the tracker is maintaining a SQLite database called tracker.db via the Python web.database module. Unlike other relational database systems that use client-server engine, SQLite is embedded into end programs. It is a popular choice among application softwares especially those for web browser. We've categorized the data to five types, and stored them separately in five tables.

- **Nodes table:** Stores the ip, port, and type (server/cache/user) of each node. It also stores the video that a user node is currently watching, and video chunks that reside in each caches
- **Account table:** Stores username and password for all accounts registered via the website, a new user cannot use our system without prior registration
- **Points table:** Stores total amount of data uploaded, incentive points earned, and all previously watched videos for each account. Every time an account runs a cache, the total bytes uploaded will be accumulated, and will eventually be used to obtain free incentive points. User can use

those points to "purchase" an unwatched video. The list of previously watched videos is saved so that the account will not get double charged for re-watching the same video

- **Videos table:** Stores a list of video names that are preprocessed with erasure coding, along with their critical data (chunk_size and last_chunk_size)
- **Account_cache table:** Stores the relationship between a running cache and a registered account, so that the bytes uploaded by that cache will be added to its owner account in the points table for incentive points

2.3 Use Pyftpdlib to transfer data between nodes

Transferring data with FTP between nodes is one of the cornerstones of our project. The prototype we received at the beginning of this year already implemented the FTP connections between nodes including server-cache, cache-user, and server-user. We've added functionalities and improved performance. The source code works only with Pyftpdlib version 0.7.0, which was outdated and lacked of support. It got updated to Pyftpdlib version 1.4.0, that supports FTPS, IPv6, Unicode file names, and MLSD/MLST commands, etc (Pyftpdlib 2015). We were also able to detect disconnections from the system and notify the tracker to remove them from database, thanks to the functionalities added by the update.

Every time the system is running, an instance of FTP server is created. On a new client connection, the server creates a new FTP handler. For each FTP handler, when a new transfer request is made, PASV mode is set, and then on transfer instantiation a DTP handler is created. The server will start listening to connections from cache and user after that with a streaming rate of 5 Mbps. The cache is implemented in a similar way. A new cache opens up a FTP mini-server in one thread, and opens up a connection to the server in another thread, both having a stream rate of 15 Mbps. It checks if itself is properly serving user-needs at every time interval. Customers can choose to config the disk size and maximum bandwidth of the cache they are running. A multiplier (0.2, 0.5 or 1) will be applied to cache's default bandwidth cap (10 Mbps) and storage cap (1.5 GB) The user node simply opens FTP client threads to receive the data from server and cache.

2.4 Downloaded file managements & shell scripts

Essentially, our project is not a torrent downloader that keeps movie permanently on customer's disk. Therefore, the customers should not be able to access the downloaded files and watch them directly using their own video player. There were three methods used to achieve this goal. First, we cut the videos into chunks and encode them into non-video format, so that anyone with chunks of movie cannot directly watch it. Second, we created obsolete file handler that can detect the size and age of user-downloaded video files. Each time the user fires up the desktop application, it will automatically delete old (7 days) videos reside in hard drive. Last but not least, we hid the downloaded file directory for additional layer of security.

In addition to programming each building blocks with Python, we also created several shell scripts to improve our development efficiency. For example, when we are trying to run a tracker publicly on the Internet, the only step to perform is to run ./populate_tracker.sh. The script takes care of creating directory, cleaning up, and runs configuration files that are required to run the tracker. Similarity, we have shell scripts to run other nodes, as well as to interact with cloud contents on AWS. Those scripts enable us to fast prototype our idea, and to scale up the system without excessive manual operations.

2.5 Accounts that can be associated with running user and caches

After creating the working VoD system as the foundation, we moved forward to improve the system design. Instead of letting customers watching videos anonymously, we created the account system, so that the videos are only available to watch after user registration. This feature is important that it not only enables us to implement user-specific incentive mechanism, but also provides vast amount of user data analyzation.

Currently, the account mechanism works as follows: A person has to visit the public address of the tracker, where the web application is running, to register a new account. The tracker will then update its database accordingly. Later on, the same person can log back to the website to view data analytics associated with the account, such as incentive points and running nodes. Although there is not much interaction available between the person and the website, the account is important for running the desktop application. When trying to watch a video using our desktop GUI, the person must log-in with a registered account. After that, the GUI will ask the person to configure the cache that is going to run on his/her edge devices. The tracker will then associate this newly created cache's uploaded data with the corresponding account. The more data an account's cache has uploaded, the more incentive points an account can obtain. Ryan Kashi's paper has an in-depth introduction on how incentive works in our system.

2.6 Simple online application with Webpy

Webpy is an open source web framework that we used to create our web app on tracker. Its simplicity helped us to develop exquisite web pages and visualizations, without spending too much time on configurations (Webpy 2016). There is no complex data structure required by Webpy, we could embed it into our tracker's code. It handles RESTful API requests for page rendering and database manipulation. A particular feature that we've implemented with Webpy was the login/logout function. Only logged in users can view their dashboard, and only the admin user can access the global system state and data visualizations. We used Webpy's sessions API to create a session file containing a unique hash code whenever a user logs in. That session file is deleted whenever the allowed timeframe has passed, or the user logs out.

Altogether, we've created views for landing page, login/signup page, admin dashboard page, user dashboard page, and about page. Screenshot of these pages can be found in the appendix.

2.7 Scaling up the system on AWS

Scalability can only be tested after our project is moved to the cloud and able to support a large scale of traffic. Instead of setting up our own physical servers, we migrated our project onto an Amazon Web Services (AWS) EC2 instance. We chose AWS because it provides reliable cloud infrastructures, such as computing power and storages, to customers to deploy their services. More importantly, resources on AWS are elastic, which can be shrunk or inflated to caters the real-time system requirements.

My teammate, Alagu Sanjana Haribhaskaran, had a initial AWS instance configured in the Fall semester. The system could run properly on that EC2 Linux instance, without communicate with hosts outside that instance. The issue was that whenever an external node attempts to connect to the FTP server in that EC2 instance, the connection got refused. This issue persisted to Spring semester, before I took over the task. After investigating the characteristics of AWS's EC2 instance, the issue was resolved by having a separate node initialization step for AWS in our code. When running a node on laptop, we'd have to choose whether to run the system locally or publicly, by either assigning the IP address to 0.0.0.0 or the laptop's IPv6 address. However, all nodes running locally on AWS are publicly accessible over the internet. Therefore, in order for external nodes to connect to them, the nodes on AWS has to be run locally, but provides their public address to the tracker. Local addresses like 0.0.0.0 will lead to wrong destination. After this fix, we were able to run tracker and server on the EC2 instance, and connect them with caches and users running in our edge devices, as shown in Figure 3.

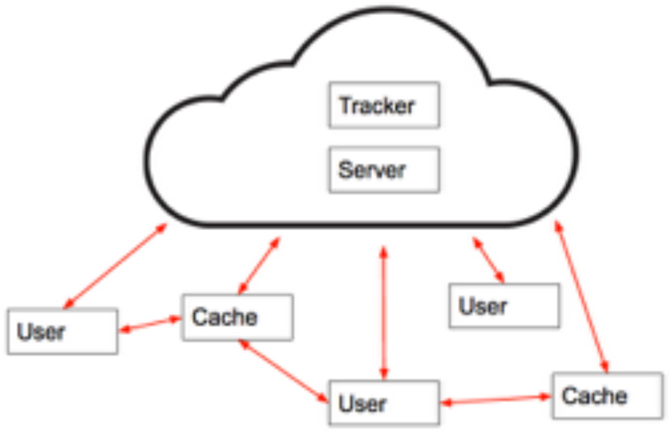


Figure 3. running *tracker* and *server* on the cloud, *user* and *cache* on edge devices.

We took advantage of many services provided by AWS, here are the complete list:

- EC2 (Elastic Cloud Compute): provides compute capacity
- EBS (Elastic Block Storage): stores large videos of multiple gigabytes
- CloudWatch: monitor the usage and cost of the instance
- Elastic IP: quickly remap the address to another instance when the current one break

- Amazon Machine Images (AMI): the snapshot of an existing instance, in which contains data templates and configurations to quickly launch new instances, used for scale up tests

With the resources provided by the AWS Free Tier (AWS Free Tier 2016), we were able to test the system on a scale of 20 users and 20 caches. Although it already has a better scalability than the system on laptop, testing with an even bigger scale (on the order of hundreds of nodes) is critical to prove the scalability of our distributed system.

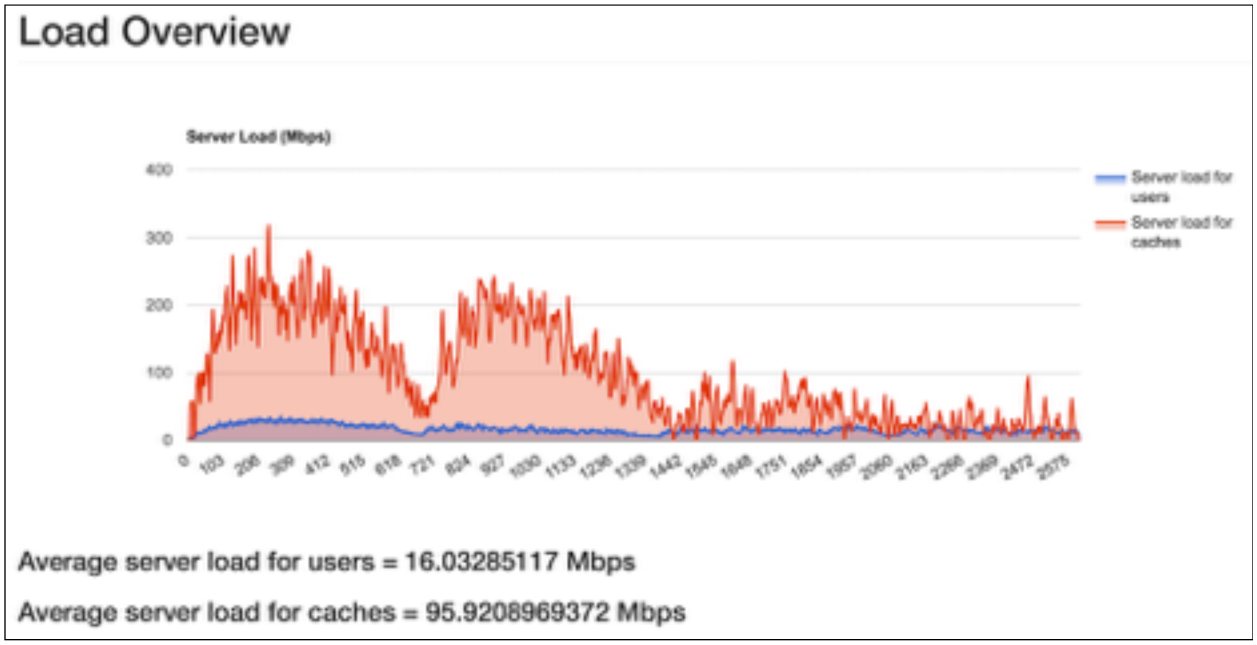
We've updated both the computing power and storage spaces of the original AWS instances, in order to scale up the stress test. Comparison between old and new configurations is shown in table 1 below (AWS EC2 Instance Types 2016).

Original	
server instance	**Computing (1 instance):** EC2 t2.micro Linux Instance (vCPU = 1, Mem = 1 GiB, Max Bandwidth = N/A) **Storage:** EBS General Purpose SSD (GP2), 1 GiB each
cache and user instance	**Computing (2 instances):** EC2 t2.micro Linux Instance (vCPU = 1, Mem = 1 GiB, Max Bandwidth = N/A) **Storage:** EBS General Purpose SSD (GP2), 1 GiB each
Upgraded	
server instance	**Computing (1 instance):** EC2 m4.xlarge Linux Instance (vCPU = 4, Mem = 16 GiB, Max Bandwidth = 93.75 Mb/s) **Storage:** EBS General Purpose SSD (GP2), 40 GiB each

cache and user instance	**Computing (6 instances):** EC2 m4.large Linux Instance (vCPU = 2, Mem = 8 GiB, Max Bandwidth = 56.25 Mb/s) **Storage:** EBS General Purpose SSD (GP2), 40 GiB each

Table 1. Infrastructure comparison after upgrades

Running on seven AWS instances, the VoD system was able to support 60 caches and 60 users, in which one instance ran server and tracker, three instances ran 20 caches each, and another three instances ran 20 users each. The 60 users were added to the system one at a time every 10 seconds, giving the system enough time to update the topology. After they were connected, these users randomly chose a movie to watch (at a streaming rate of 3.2 Mbps), and proceeded to next movie when the previous movie was finished. Monitoring the system with the tracker's web app, we observed the system to run smoothly, as shown in Figure 4.


Load Overview

Average server load for users = 16.03285117 Mbps

Average server load for caches = 95.9208969372 Mbps

Figure 4. Load overview when running 60 caches and 60 users.

This figure shows the server load to caches and users after running for 40 minutes. As caches received more movie chunks for server, they were able to stream video contents to other users, therefore relieved the load from server. The peer-to-peer traffic between edge devices was taking up more workloads, so that the serve would send less data to both caches and users as time goes by.

The average server load, as seen in Figure 4, is about 112 Mbps (sum of 16 Mbps to users and 96 Mbps to caches). This is 40% less than 192 Mbps (3.2 Mbps * 60 users), the load in the traditional server-client VoD architecture. This percentage is still increasing, because the server load will converge to a very small number after running for a long period. This test verified the original algorithm design, as well as the scalability of this VoD system running in a large scale. A even larger test with thousands of users and caches can be made in the future with more infrastructure resources. We strongly believe that the system will still working properly in that context.

3 Future Works & Conclusion

Overall, the project aligned with our original designs. Most functionalities in our initial scope were developed, along with many other new ideas emerged. Given the limited duration of development, we cannot implement all features suggested. There are possible future improvements that allow our VoD system to achieve a higher success.

First and foremost, advanced file securities can be applied to our system. Currently, it is still feasible for user to locate the downloaded videos and copy them out of the directory, essentially making our product a movie downloader. In order to maintain the exclusiveness of user experience and intellectual property, file encryptions and special video-chunking techniques can be applied. Secondly, Our product is not yet a fully packaged software that can be directly downloaded and installed. In contrast, the project can only be configured manually (install dependencies by hand, which requires adequate computer science skills). Packaging the project requires more research on related techniques, which needs additional time we do not possess. Thirdly, the system is deployed on a single AWS EC2 instance, which has limited computation power, storage, and bandwidth. In order to scale up the system to support tens of thousands of users, a more powerful infrastructure needs to be obtained. Nevertheless, the cost to maintain such infrastructure will radically increase simultaneously.

There were also changes of plan as we got more insights of our project's capability. Initially, the project was on track to become a profitable product in the emerging industry of VoD. However, after careful consideration and research, we decided to open source the project in the future. We realized that its technical depth does not guarantee a significant technical advantage on the extremely competitive VoD market. On the other hand, the open source community can help our project to become a popular file distribution system, not only for movies, but also for documents and pictures sharing. In addition, the incentive mechanism was a great idea came up during the development. In additional to a simple video player, having a point system increases user stickiness, and differentiates our product with other existing VoD systems.

Our capstone team maintained a good pace during the development. We've created clear and detailed documentations at the same time, so that any successors or open source contributor can pick up the project and start adding features. We realized that team communication and project planning are equally important as good programming skills. We cannot make such progress without those factors working together, and we've gained invaluable hands-on experiences in this capstone project.

Work Cited

AWS EC2 Instance Types. "Amazon EC2 Instance Types" *Amazon Web Services, Inc.* Amazon Web Services. Web. 09 Apr. 2016. <https://aws.amazon.com/ec2/instance-types/>.

AWS Free Tier. "Free Cloud Services – AWS Free Tier." *Amazon Web Services, Inc.* Amazon Web Services. Web. 09 Apr. 2016. < https://aws.amazon.com/free/ >.

Lee, Kangwook., Hao Zhang, Ziyu Shao, Minghua Chen, Abhay Parekh, and Kannan Ramchandran. "An Optimized Distributed Video-on-demand Streaming System: Theory and Design." *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*(2012): n. pag. Web.

Lee, Kangwook., Yan Lisa, Parekh Abhay, and Kannan Ramchandran. "A VoD System for Massively Scaled, Heterogeneous Environments: Design and Implementation." 2013 *IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (2013): n. pag. Web.

Pyftpdlib. “Pyftpdlib 1.5.0 : Python Package Index." Pyftpdlib 1.5.0 : *Python Package Index.* Python Community, 13 Dec. 2015. Web. 09 Apr. 2016. < https://pypi.python.org/pypi/pyftpdlib/ >.

Swartz, Aaron. "Welcome to Web.py! (web.py)." *Welcome to Web.py! (web.py).* Webpy, 5 Apr. 2016. Web. 09 Apr. 2016. < http://webpy.org/ >.

Chapter 2

Engineering Leadership

(Co-written with Alagu Santana Haribhiskaran and Ryan Kashi)

Among all engineering leadership topics and tools we learnt over the last year, we chose the three most relevant modules to our Video on Demand (VoD) system. This paper will analyze the current industry of video content delivery, market research that can validate the commercial success of our product and the management of the software intellectual property.

Industrial Analysis

As of 2014, broadband internet is nearly ubiquitous around the world with the “global average connection speed exceeding the 4 Mbps broadband threshold” (Young 2014) according to a report generated from the Akamai Intelligent Platform. With the increasing availability of high speed Internet and Internet connected devices, the delivery of entertainment video directly over the Internet has become a convenient solution for consumers. Video streaming services have increasingly been accepted by consumers as the alternative to multichannel video programming distributors (MVPD) such as satellite and cable television systems and physical media rental services.

Our platform is backed behind a novel heuristic that reduces bandwidth costs on a centralized server to potentially nothing, which gives our system an edge over current video streaming services such as Netflix, Amazon Video, and Hulu. Despite this, it is important to observe and analyze the current competitors of this industry to see what could be done for us to successfully enter this line of business.

In the realm of entertainment video delivery over the Internet, Netflix dominates and controls the market. Netflix members are able to enjoy streaming media over their televisions, mobile devices, and computers. Members can play and pause content at their leisure without commercials, commitments, or restrictions. Netflix, Inc. is a membership service, but members have unlimited access to a pool of available content during their time of membership. As of July 2011, Netflix, Inc. has increased its emphasis on the removal of DVD based entertainment, focusing more on its robust video streaming platform,

An advantage Netflix, Inc. has over the market for entertainment video delivery is its emphasis on making its service compatible with many devices while maintaining streaming functionality. The company holds partnerships with “satellite and telecommunications operators to make their service available through the television set-up boxes of these service providers,” (Netflix 10-K 2014).

Netflix, Inc. “strives for consumers to choose (Netflix) in their moment of free time,” (Netflix 10-K 2014). Netflix, Inc. refers to this part of their company’s objective as the “winning moments of truth” (Netflix 10-K 2014) and places emphasis on their “recommendation and merchandising technology to predict and recommend titles for members to enjoy,” (Netflix 10-K 2014). In addition, Netflix has begun rolling out its own Content Delivery Network (CDN) called Open Connect that will reduce cost and improve the delivery of the service’s content by reducing network overhead (Netflix 2014). Because Netflix is a large percentage of the traffic ISPs deliver to their end-users, Netflix, Inc. has offered a solution to peer directly with Netflix. CDN, however, will only provide a reduction in latency for users and has no effect on the total amount of data sent from Netflix servers to its customers. Thus, despite their claims, our platform has the capability of providing customers with the same video at a greatly reduced price.

A concern and risk factor for Netflix, Inc. (Netflix 10-K 2014) is the retention and attraction of members to the service. Consumers need to seek value with the service which can be primarily found in the speedy delivery of content as well as “compelling content choices, as well quality experience for selecting and viewing TV shows and movies,” (Netflix 10-K 2014).

Market Research

Over the last two decades, there has been an exponential increase in the interest shown in watching videos be it news, music, or movies indicating that the VoD system has a potentially huge market. The current leading competitors in the VoD market are Netflix, Hulu and Amazon Prime Video and the revenue generated by these companies indicate consumer’s inclination towards the video streaming service is increasing and this technology is slowly replacing traditional satellite and cable television systems.

The core idea of success behind our VoD system is based on our conclusion from market research that by caching the few popular videos that are on high demand we can significantly reduce the bandwidth of the system, while delivering content to users at an improved speed. According to a Video statistics and Market research by Tubular, a video intelligence software, only 5% of the videos uploaded on YouTube drive most of the views (Marshall 2015b). Out of 1.1 billion videos on YouTube, only 58.6 million have more than 10,000 views and these videos alone have generated 7.4 trillion views (Marshall 2015b).

Anyone who owns an internet connected device is a potential customer to our product. The Interactive Advertising Bureau, a nonprofit organization, states that there has been an increase of 35% in views collected by people watching videos on their smart devices in the past year and also one third of the people surveyed said that they watch at least one video lasting five minutes everyday (Marshall 2015a). Smartphones with bigger screen, improved display, memory, and battery life indicate that the technological advancements in the edge devices go hand in hand with these social trends.

Cisco predicts that by 2018, 84% of the consumer internet traffic will come from video and clog the internet (Zccone 2014). Consumers are streaming more video content than ever before, and Internet Service Providers are not able to maintain sufficient bandwidth required to enable constant high quality video streaming. Being well aware of this problem, Verizon and Comcast have made special agreement with the heaviest traffic producing consumers that for an additional cost, the traffic from these customers will be handled at a higher priority (Grill 2015). The technological innovations like ‘capture camera’ and ‘retina display TV with UHD 4k videos’ worsen the problem, as the backend is not able to keep up with the user demands. Reducing video bitrates is not a solution, as it will only lead to poor viewing experience and make the product lose its value (Grill 2015). Our system comes up with a feasible solution to this problem. if we cache the popular videos on edge devices, the user would not have to get the data from the central server, thus reducing internet traffic and improving the performance in terms of speed and quality.

This project can be made into a commercial product or an open source project contributing to technological evolution. An ideal marketing strategy for this system would be to develop a final product with a good user interface, and advertise it as a commercial product. Our system, can provide high quality video at a cheaper price, making it easier to sell our product and replace existing VoD companies. In conclusion, our market research indicates potential success of our product and the ability to revolutionize the VoD market.

Intellectual Property (IP)

The core technology of this project is the VoD architecture, including the concepts of using caches as edge resources, algorithm ruling the data flow, and methods of chunking and distributing videos. Several related papers have been published on a theoretical level by the advising laboratory (Lee et al. 2012, Lee et al. 2013), and a working prototype has been built on top of them. We believe that this system architecture is patentable, and holding such patent could bring numerous advantages to the product in business.

First, a patent enables us to provide the stable and fast VoD experience to customers in an exclusive way. It will maintains the differentiate value from the technology we developed. Second, having a protection of the core technology allows the ongoing product development to focus on the product itself, without worrying about similar services being released by competitors. Last but not least, a patent can add value to our project and thus make it more likely that big players in the market will buy our technology. Typically, filing a utility patent in U.S. will cost around $20,000 (Quinn 2016). Nevertheless, this is a small investment compared to the future return, as current market competitors are generating 1 billion dollars approximately of revenue (MarketsAndMarkets 2016).

On the other hand, there are difficulties in protecting this IP as a patent. VoD market has become extremely competitive in recent years. Companies such as Rovi, AT&T and Microsoft are holding large patent portfolios. Moreover, software patents have historically been regarded as hard to define. As a result, improperly defined IPs can lead to lawsuits if it is filed without careful prior research. For example, in 2013, the author of the patent "Video-on-demand systems (Nomura 2007)" suited Youtube and Amazon's CloudFront services of infringe his entire patent (Randles 2016). Also from a bigger scope, the product's bandwidth and resource-saving advantages might be invalidated by network technology advancement, as the network resources will not be a issue any more.

Alternative solutions to protect our IP including transform it into a trade secret, sell the project to existing VoD vendors, or make it an open source project. Our project can be a valuable addition to the existing VoD platforms as an upgrade of their original service. Nevertheless, if the technology is considered non-profitable after throughout market research, the source code can be released to community, and be transformed into a popular free VoD (or file sharing) solution.

Conclusion

Analysis of the industry landscape, market research and IP above helped us to better understand the technical potential of the product, as well as a clearer idea of how to commercialize it. Tools learnt from the engineering leadership courses have played a crucial role in composing this paper, and we are going to utilize them in the future stage of development.

Work Cited

Gill, Dror. "As The Internet Grows Bigger, What Do We Do About Video?" *ReelSEO*. Reelseo & Tubular Labs, Inc., 15 Jan. 2015. Web. 05 Mar. 2016. <http://www.reelseo.com/internet-bigger-video/>.

Lee, Kangwook., Hao Zhang, Ziyu Shao, Minghua Chen, Abhay Parekh, and Kannan Ramchandran. "An Optimized Distributed Video-on-demand Streaming System: Theory and Design." *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*(2012): n. pag. Web.

Lee, Kangwook., Yan Lisa, Parekh Abhay, and Kannan Ramchandran. "A VoD System for Massively Scaled, Heterogeneous Environments: Design and Implementation." 2013 *IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (2013): n. pag. Web.

MarketAndMarket. "Video on Demand (VOD) Market worth $61.40 Billion by 2019." *Video on Demand (VOD) Market worth $61.40 Billion by 2019*. Marketsandmarkets, Feb. 2015. Web. 05 Mar. 2016.

Marshall, Carla. "We're All Watching More Video on Mobile Devices [Study]." *ReelSEO*. Reelseo & Tubular Labs, Inc., 10 June 2015a. Web. 05 Mar. 2016. <http://www.reelseo.com/increase-mobile-video-consumption/>.

Marshall, Carla. "5% of Videos on YouTube Drive 95% of the Views." *ReelSEO*. Reelseo & Tubular Labs, Inc., 04 Sept. 2015b. Web. 05 Mar. 2016. <http://www.reelseo.com/5-percent-youtube-videos-drive-95-percent-views/>.

Netflix Inc. *2014 10-K Report*. Netflix Inc. 03 Feb. 2014. Web. 06 Mar. 2016

Netflix. *OpenConnect-Deployment-Guide.pdf.* N.p.: Netflix, 2014.*Oc.nflxvideo.net*. Netflix, 2014. Web. 06 Mar. 2016. <http://oc.nflxvideo.net/docs/OpenConnect-Deployment -Guide.pdf>

Nomura, Tetsuya. Video-on-demand System. Tetsuya Nomura, Tommy Sun, assignee. Patent 7254622. 7 Aug. 2007. Print.

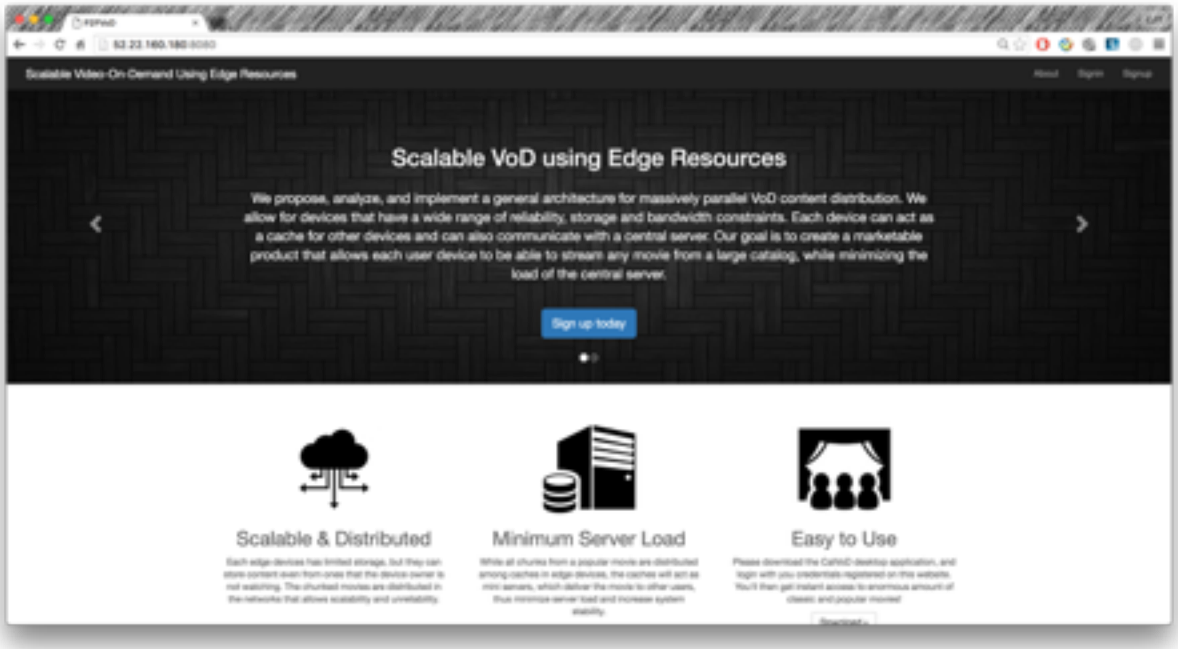
Randles, Jonathan. "YouTube, Amazon Escape Video On Demand Patent Suits - Law360." *YouTube, Amazon Escape Video On Demand Patent Suits - Law360.* Law360, 13 Sept. 2013. Web. 05 Mar. 2016.<http://www.law360.com/articles/472533/youtube-amazon-escape-video-on-demand-patent-suits>

Quinn, Gene. "The Cost of Obtaining a Patent in the US - IPWatchdog.com | Patents & Patent Law." *IPWatchdogcom Patents Patent Law The Cost of Obtaining a Patent in the US Comments.* IPWatchdog, 04 Apr. 2015. Web. 05 Mar. 2016. <http://www.ipwatchdog.com/2015/04/04/the-cost-of-obtaining-a-patent-in-the-us/>

Young, Jeff. "Akamai Releases Second Quarter 2014 'State of the Internet' Report." *Akamai*. Akamai, 30 Sept. 2014. Web. 06 Mar. 2016.

Zaccone, Emanuela. "Why Video Consumption and Instant Messaging Rule Social Trends." *Medium*. N.p., 29 Jan. 2015. Web. 05 Mar. 2016. <https://medium.com/ @zatomas/why-video-consumption-and-instant-messaging-rule-social-trends-5d86e51e4355#.rfvko98rx>.

Appendix



Scalable VoD using Edge Resources

We propose, analyze, and implement a general architecture for massively parallel VoD content distribution. We allow for devices that have a wide range of reliability, storage and bandwidth constraints. Each device can act as a cache for other devices and can also communicate with a central server. Our goal is to create a marketable product that allows each user device to be able to stream any movie from a large catalog, while minimizing the load of the central server.

Scalable & Distributed

Each edge devices has limited storage, but they can store content even from ones that the device owner is not watching. The chunked movies are distributed in the networks that allows scalability and unreliability.

Minimum Server Load

While all chunks from a popular movie are distributed among caches in edge devices, the caches will act as mini servers, which deliver the movie to other users, thus minimize server load and increase system stability.

Easy to Use

Please download the CaVoD desktop application, and login with you credentials registered on this website. You'll then get instant access to enormous amount of classic and popular movies!

Figure i. Landing page of the web app.

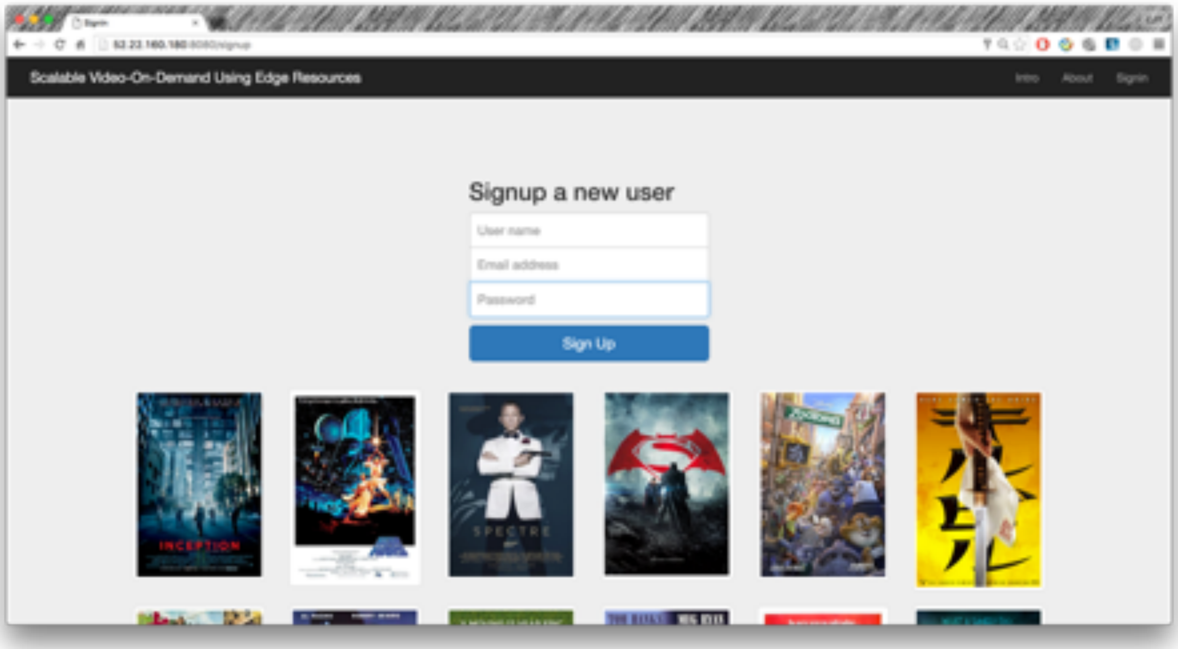
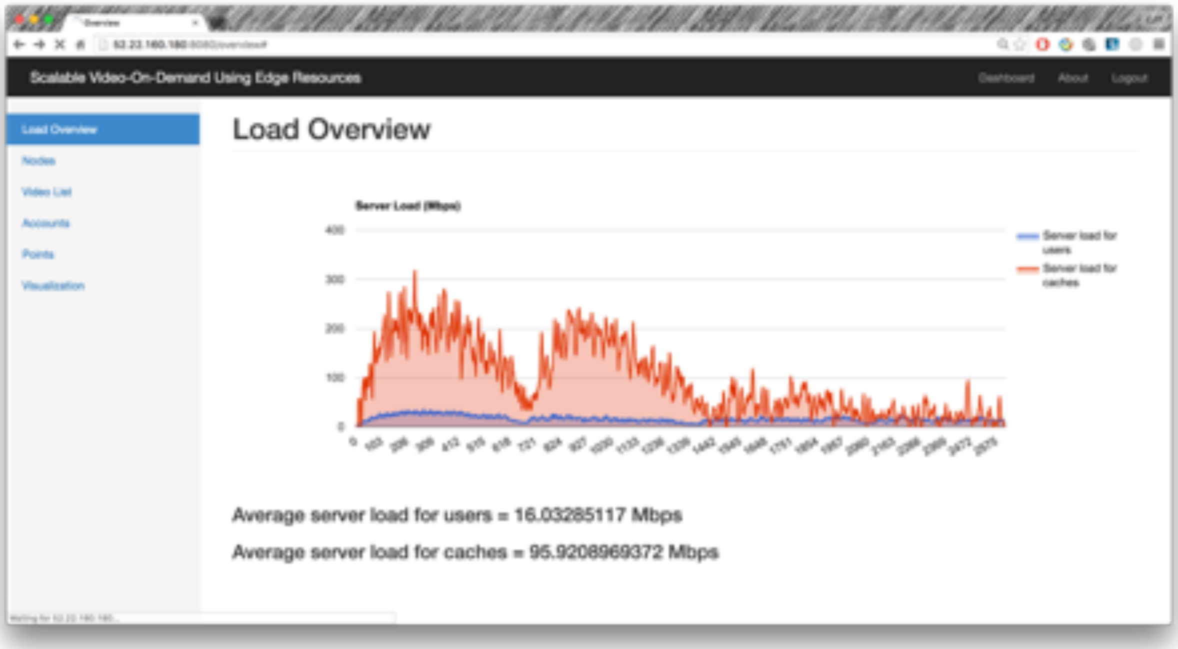


Figure ii. Sign-up page of the web app.

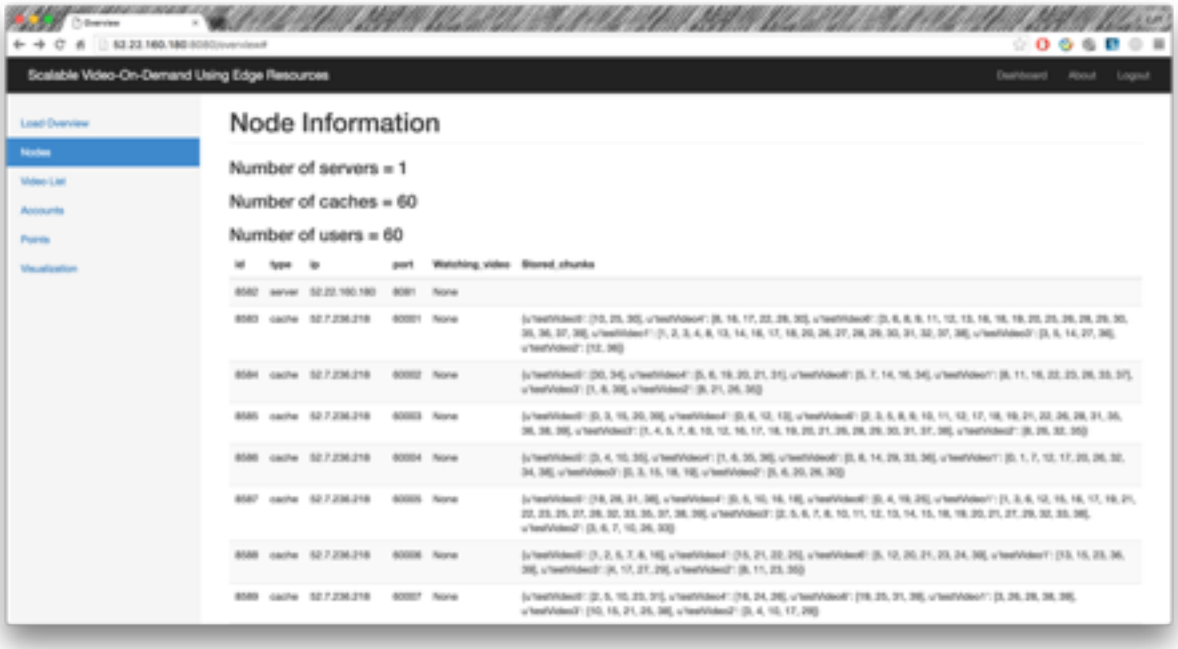


Load Overview

Average server load for users = 16.03285117 Mbps

Average server load for caches = 95.9208969372 Mbps

Figure iii. Admin dashboard page of the web app (load overview).



Node Information

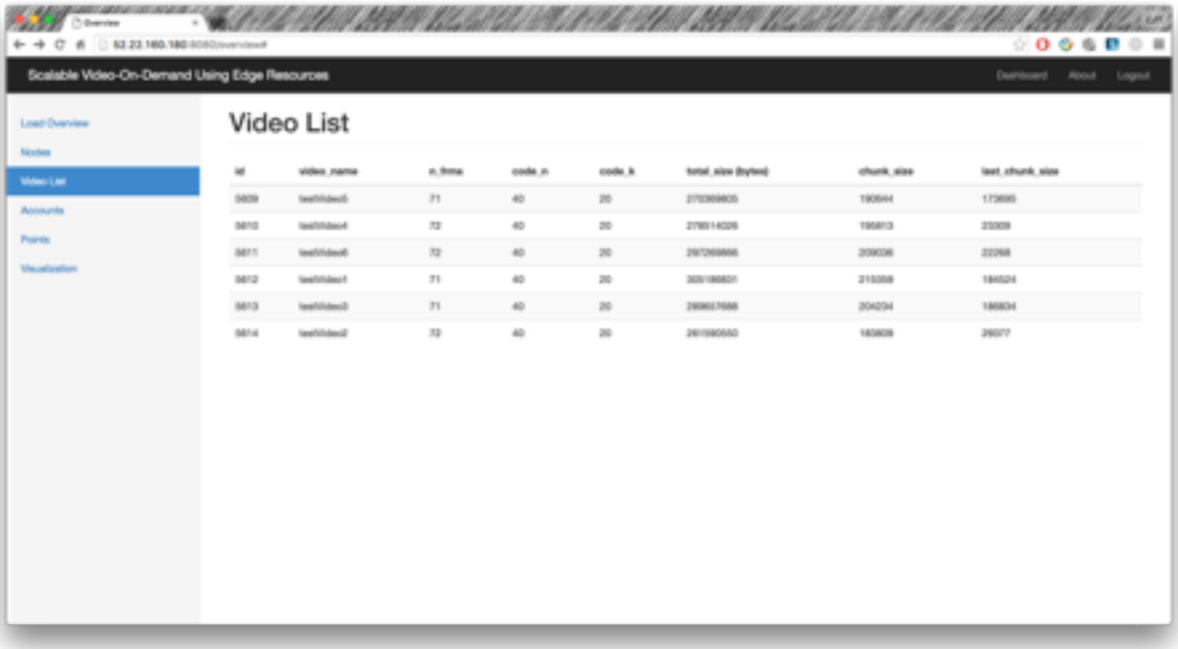
Number of servers = 1

Number of caches = 60

Number of users = 60

id	type	ip	port	Watching_video	Stored_chunks
8582	server	52.22.160.180	8081	None	
8583	cache	52.7.236.218	60001	None	{u'testVideo5': [10, 25, 30], u'testVideo4': [8, 16, 17, 22, 28, 30], u'testVideo6': [3, 6, 8, 9, 11, 12, 13, 16, 18, 19, 20, 25, 26, 28, 29, 30, 35, 36, 37, 39], u'testVideo1': [1, 2, 3, 4, 8, 13, 14, 16, 17, 18, 20, 26, 27, 28, 29, 30, 31, 32, 37, 38], u'testVideo3': [3, 5, 14, 27, 36], u'testVideo2': [12, 36]}
8584	cache	52.7.236.218	60002	None	{u'testVideo5': [30, 34], u'testVideo4': [5, 6, 19, 20, 21, 31], u'testVideo6': [5, 7, 14, 16, 34], u'testVideo1': [8, 11, 16, 22, 23, 26, 33, 37], u'testVideo3': [1, 8, 39], u'testVideo2': [8, 21, 26, 35]}
8585	cache	52.7.236.218	60003	None	{u'testVideo5': [0, 3, 15, 20, 39], u'testVideo4': [0, 6, 12, 13], u'testVideo6': [2, 3, 5, 8, 9, 10, 11, 12, 17, 18, 19, 21, 22, 26, 28, 31, 35, 36, 38, 39], u'testVideo3': [1, 4, 5, 7, 8, 10, 12, 16, 17, 18, 19, 20, 21, 26, 28, 29, 30, 31, 37, 38], u'testVideo2': [8, 26, 32, 35]}
8586	cache	52.7.236.218	60004	None	{u'testVideo5': [3, 4, 10, 35], u'testVideo4': [1, 6, 35, 36], u'testVideo6': [0, 8, 14, 29, 33, 36], u'testVideo1': [0, 1, 7, 12, 17, 20, 26, 32, 34, 38], u'testVideo3': [0, 3, 15, 18, 19], u'testVideo2': [5, 6, 20, 26, 30]}
8587	cache	52.7.236.218	60005	None	{u'testVideo5': [18, 28, 31, 38], u'testVideo4': [0, 5, 10, 16, 18], u'testVideo6': [0, 4, 19, 25], u'testVideo1': [1, 3, 6, 12, 15, 16, 17, 19, 21, 22, 23, 25, 27, 28, 32, 33, 35, 37, 38, 39], u'testVideo3': [2, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 18, 19, 20, 21, 27, 29, 32, 33, 38], u'testVideo2': [3, 6, 7, 10, 26, 33]}
8588	cache	52.7.236.218	60006	None	{u'testVideo5': [1, 2, 5, 7, 8, 16], u'testVideo4': [15, 21, 22, 25], u'testVideo6': [5, 12, 20, 21, 23, 24, 39], u'testVideo1': [13, 15, 23, 36, 39], u'testVideo3': [4, 17, 27, 29], u'testVideo2': [8, 11, 23, 35]}
8589	cache	52.7.236.218	60007	None	{u'testVideo5': [2, 5, 10, 23, 31], u'testVideo4': [16, 24, 26], u'testVideo6': [19, 25, 31, 39], u'testVideo1': [3, 26, 28, 38, 39], u'testVideo3': [10, 15, 21, 25, 38], u'testVideo2': [3, 4, 10, 17, 29]}

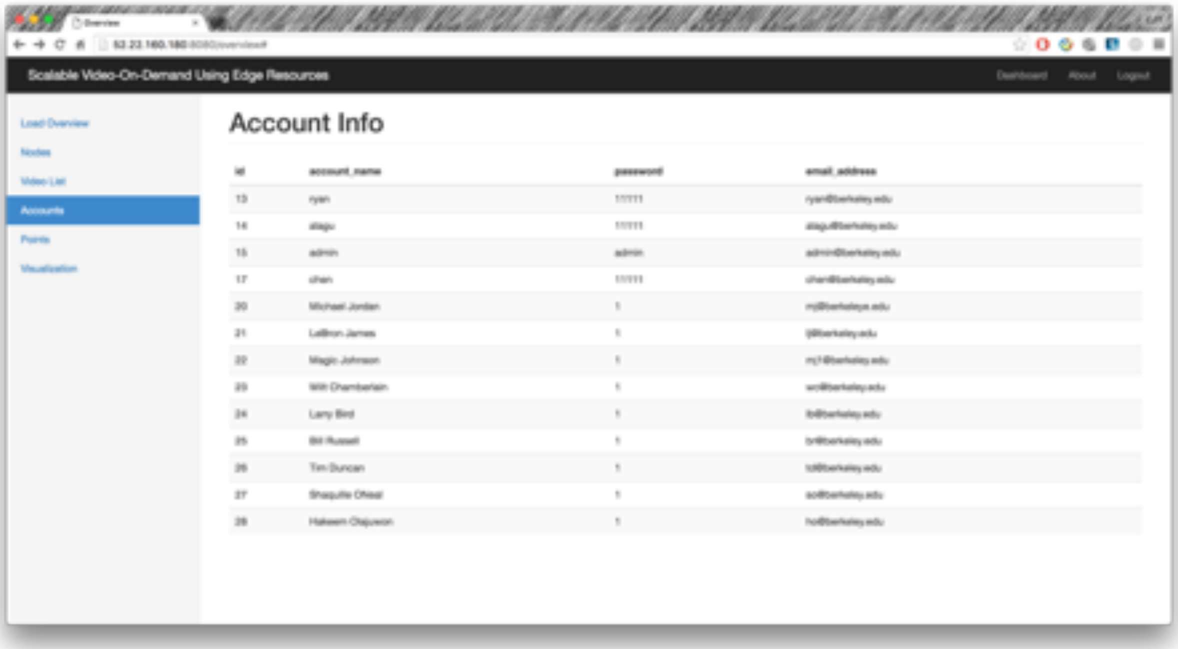
Figure iv. Admin dashboard page of the web app (node information).



Video List

id	video_name	n_frme	code_n	code_k	total_size (bytes)	chunk_size	last_chunk_size
5609	testVideo5	71	40	20	270369605	190644	173695
5610	testVideo4	72	40	20	278514026	195813	23009
5611	testVideo6	72	40	20	297269666	209036	22268
5612	testVideo1	71	40	20	305186801	215359	184524
5613	testVideo3	71	40	20	289657686	204234	186834
5614	testVideo2	72	40	20	261090550	183809	26077

Figure v. Admin dashboard page of the web app (video list).



Account Info

id	account_name	password	email_address
13	ryan	111111	ryan@berkeley.edu
14	alagu	111111	alagu@berkeley.edu
15	admin	admin	admin@berkeley.edu
17	chen	111111	chen@berkeley.edu
20	Michael Jordan	1	mj@berkeleye.edu
21	LeBron James	1	lj@berkeley.edu
22	Magic Johnson	1	mj1@berkeley.edu
23	Wilt Chamberlain	1	wc@berkeley.edu
24	Larry Bird	1	lb@berkeley.edu
25	Bill Russell	1	br@berkeley.edu
26	Tim Duncan	1	td@berkeley.edu
27	Shaquille ONeal	1	so@berkeley.edu
28	Hakeem Olajuwon	1	ho@berkeley.edu

Figure vi. Admin dashboard page of the web app (account info).

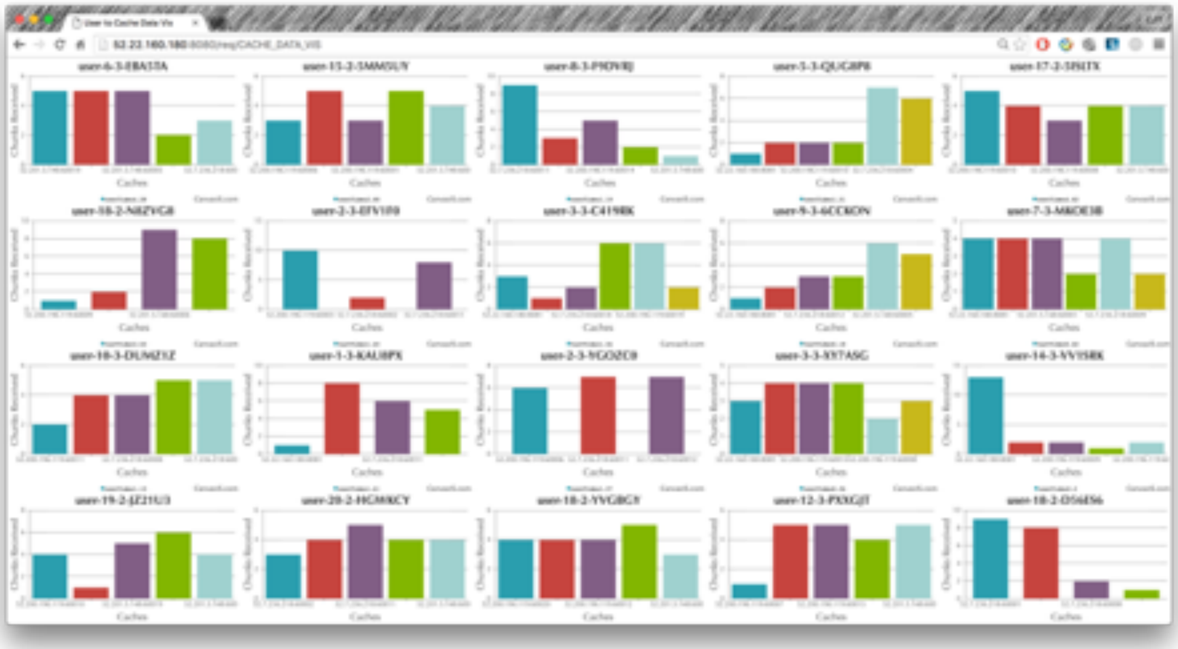


Figure vii. Admin dashboard page of the web app (User received chunks visualization).

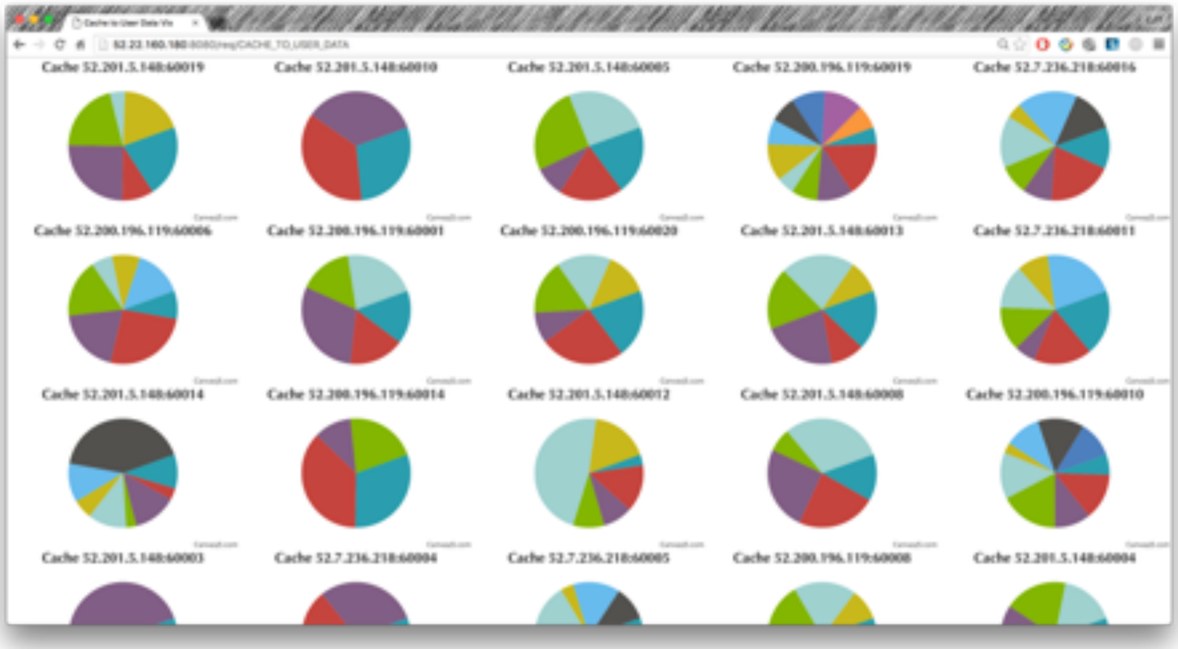
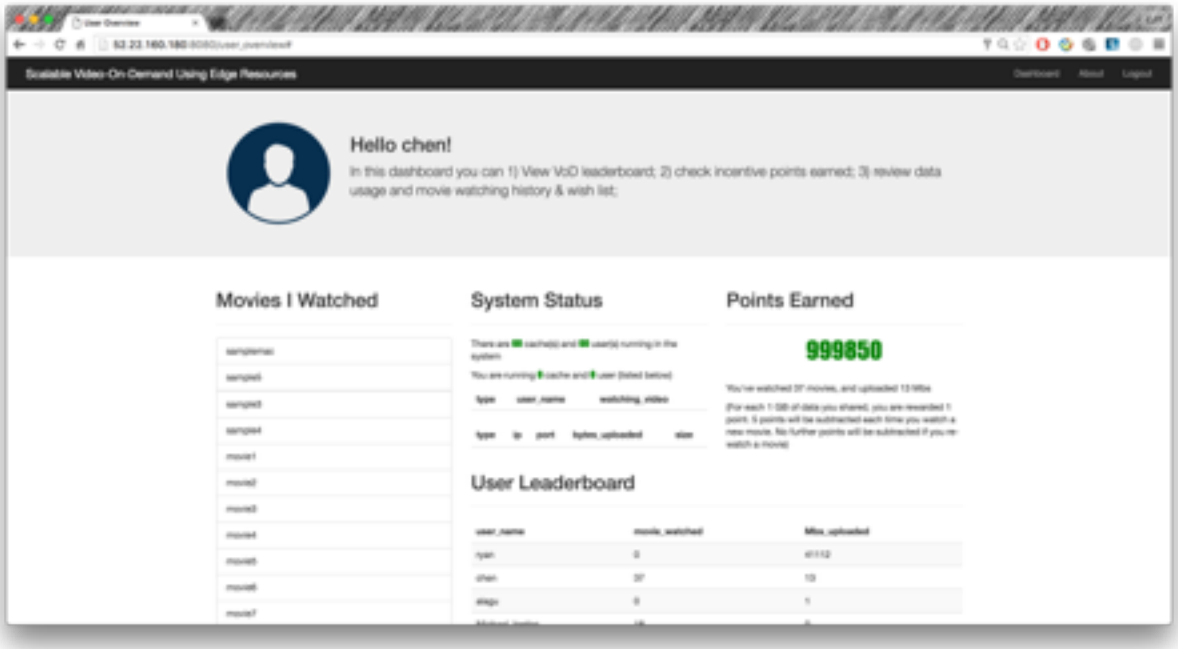


Figure viii. Admin dashboard page of the web app (Cache sent chunks visualization).



Scalable Video-On-Demand Using Edge Resources

Hello chen!

In this dashboard you can 1) View VoD leaderboard; 2) check incentive points earned; 3) review data usage and movie watching history & wish list;

Movies I Watched

- samplemac
- sample5
- sample3
- sample4
- movie1
- movie2
- movie3
- movie4
- movie5
- movie6
- movie7

System Status

There are 🟩 cache(s) and 🟩 user(s) running in the system

You are running 🟩 cache and 🟩 user (listed below)

type	user_name	watching_video

type	ip	port	bytes_uploaded	size

Points Earned

999850

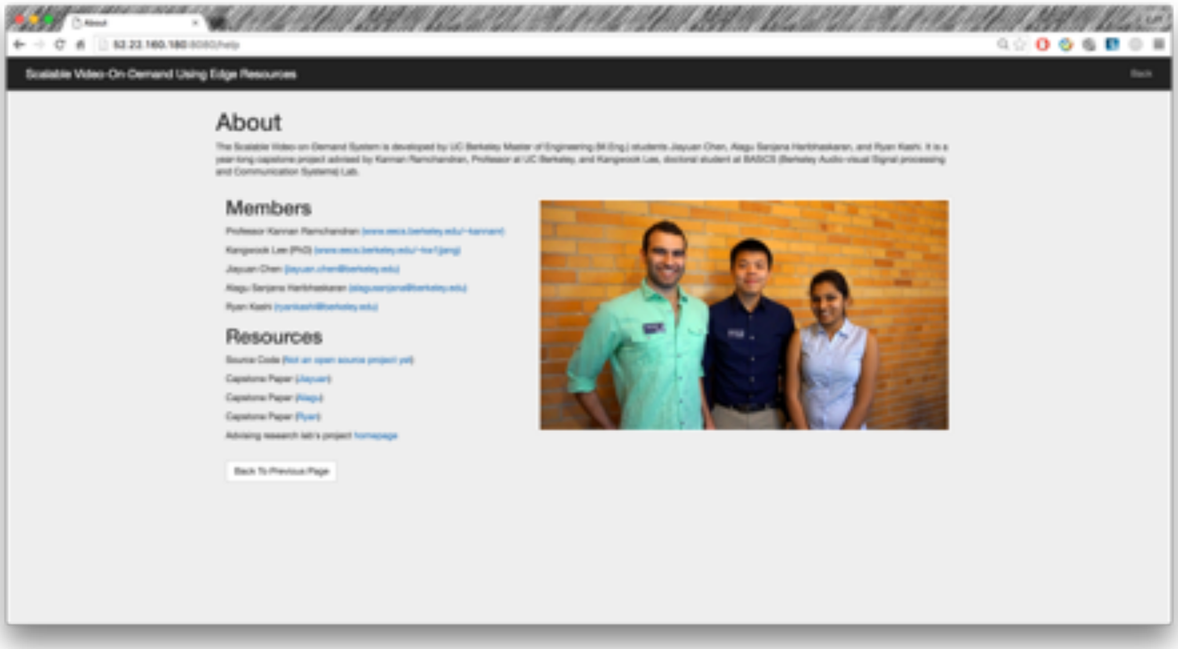
You've watched 37 movies, and uploaded 13 Mbs

(For each 1 GB of data you shared, you are rewarded 1 point. 5 points will be subtracted each time you watch a new movie. No further points will be subtracted if you re-watch a movie)

User Leaderboard

user_name	movie_watched	Mbs_uploaded
ryan	0	41112
chen	37	13
alagu	0	1
[illegible]	[illegible]	[illegible]

Figure ix. User dashboard page of the web app.



Scalable Video-On-Demand Using Edge Resources

About

The Scalable Video on Demand System is developed by UC Berkeley Master of Engineering (M.Eng.) students Jiayuan Chen, Alagu Sanjana Harikbhaskaran, and Ryan Kashi. It is a year-long capstone project advised by Kannan Ramchandran, Professor at UC Berkeley, and Kangwook Lee, doctoral student at BASiCS (Berkeley Audio-visual Signal processing and Communication Systems) Lab.

Members

Professor Kannan Ramchandran (www.eecs.berkeley.edu/~kannanr)

Kangwook Lee (PhD) (www.eecs.berkeley.edu/~kw1jjang)

Jiayuan Chen (jiayuan.chen@berkeley.edu)

Alagu Sanjana Harikbhaskaran (alagusanjana@berkeley.edu)

Ryan Kashi (ryankashi@berkeley.edu)

Resources

Source Code (Not an open source project yet)

Capstone Paper (Jiayuan)

Capstone Paper (Alagu)

Capstone Paper (Ryan)

Advising research lab's project homepage

Back To Previous Page

Figure x. About page of the web app.