Digital Democracy Video Manager

by
Scott Lam

Computer Science Department
College of Engineering
California Polytechnic State University
2016

Date Submitted: March 18, 2016
Advisor: Alex Dekhtyar

Table of Contents

Table of Figures ii
Abstract iii
Background 1
Use Case 2
Design 3
- Original Design 3
- After Further Research 3

Implementation 4
- Technology & Frameworks 4
- File Hierarchy & File Descriptions 4
- Environment Variables 5
- API 5
- Cache 5

Setup & Testing 6
- Setup 6
- Testing 6

Conclusion 7

Table of Figures

Figure 1. Use Case Example Showing Video Manager Integration With Video Cutting Feature. 2
Figure 2. Digital Democracy Video Manager Design 3
Figure 3. Video Manager File Hierarchy 4

Abstract

The purpose of this senior project was to design and implement a video manager that provides scalability by caching videos. Currently, there are about 3600 videos associated with the Digital Democracy project. These videos are only for the California legislative system and as the project expands to multiple states the amount of videos will rapidly increase. By creating this video manager, disk space on the Digital Democracy server will be saved. All videos can be hosted on the cloud storage service and when operations need to be performed on those videos, they will be downloaded from the cloud.

Background

Digital Democracy is a project created by The Institute for Advanced Technology and Public Policy (IATPP) at California Polytechnic State University, San Luis Obispo. Digital Democracy delivers its database of California legislative hearings and transcriptions through its website http://www.digitaldemocracy.org. The goal of the project is to make legislative data, normally difficult to find or not available to the public at all, available to the public in an easily searchable format. IATPP seeks to provide more features in its product, for example, the ability for Digital Democracy users to cut videos and stitch the videos together to make a collage.

Use Case

The figure below illustrates how the video manager would interact with one of Digital Democracy's features – video cutting/collaging. First, the user requests that a video be cut or multiple videos be concatenated from the Digital Democracy website. Next, that video is sent to the video cutting tool. In order for operations to be performed on videos, they need to be available locally. This is where the video manager comes in. The video cutting tool sends a request to the video manager for the video(s) and the video manager returns the path to those video(s). The video cutting tool then performs operations on those video(s) and the result is passed back to the user.

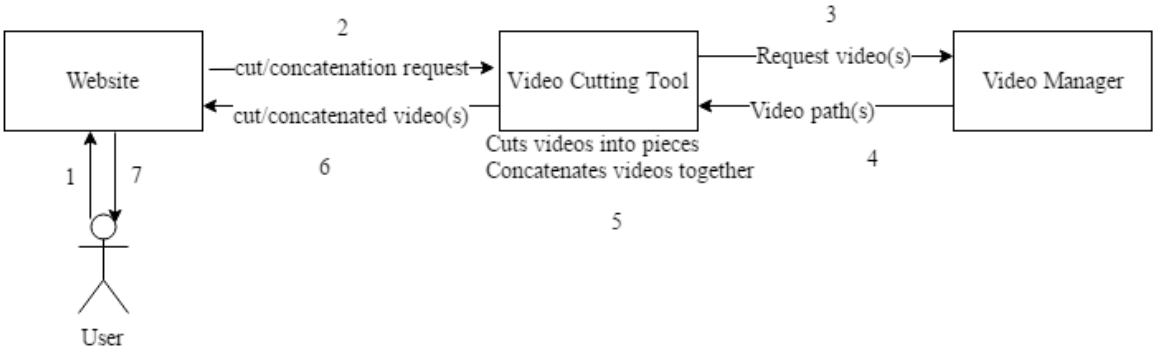

Figure 1. Use Case Example Showing Video Manager Integration With Video Cutting Feature.

Design

The design of the video manager is shown below in Figure 1. An API provides a high-level interface to interact with the cache. If there is a hit, the path to the video is returned. Otherwise, the video is downloaded from the cloud storage service and the path to the video is returned to the API.

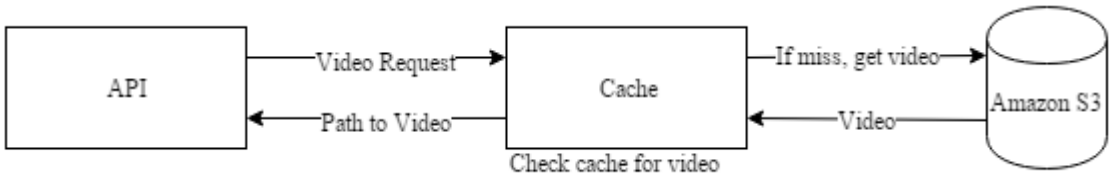

Figure 2. Digital Democracy Video Manager Design

Original Design

Originally, the video manager was designed as a daemon listening to a socket using Python's python-daemon and socket library. The cache would be a daemon listening to a socket and the API would communicate with the cache through that socket.

After Further Research

A better solution was found that would provide a higher level framework for achieving the same functionality and would be faster and easier to set up. This solution was to implement a HTTP server that would receive requests from the API. The API allows users to retrieve videos without having to know how it works.

Implementation

This section will describe, in detail, how the video manager was actually set up after the design was established.

Technology & Frameworks

- Amazon Web Services - EC2, S3
- Python 2.7 – Flask, WSGI
- Apache

File Hierarchy & File Descriptions

The figure below illustrates the structure of the video manager. Descriptions of the purpose of each file are also explained.

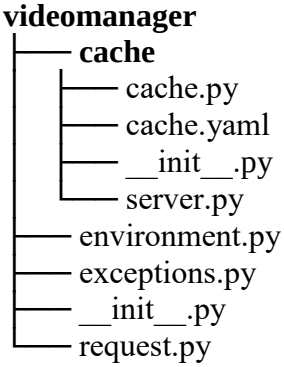

Figure 3. Video Manager File Hierarchy

environment.py – Contains functions that retrieve environment variables.

exceptions.py – Contains classes for exceptions thrown.

request.py – Contains API function calls.

cache.py – Creates the Flask application and handles HTTP requests.

server.py – Contains all server-related functions that are called by the cache.

cache.yaml – Settings file for the cache.

Environment Variables

- DD_CACHE_PATH – The path to the cache directory where videos are stored.
- DD_CACHE_HOST – The host where the cache is located.
- DD_CACHE_PORT – The port that the cache will be using.
- AWS_ACCESS_KEY_ID – The access key for Amazon S3.
- AWS_SECRET_ACCESS_KEY – The secret key for Amazon S3.
- AWS_BUCKET – Name for the bucket where videos are stored on Amazon S3.

API

The API sends a GET request to the HTTP server which is acting as the cache.

Cache

The cache implements the Least Recently Used (LRU) algorithm for keeping videos. It checks each video's modification time to determine the least recently used video. When a video is requested the cache:

1. Checks if the video is in the cache
    a. If it is, update the modification time and return its path; otherwise, continue.
2. Creates a connection to S3 and get the bucket that contains all videos.
3. Gets the threshold (cache size) from the settings file.
4. Calculates how much space will be occupied after the video is downloaded.
5. If the new size is greater than the threshold, keeps removing the least recently used video until there is enough space to accommodate the new video.
6. Downloads the video from Amazon S3.

Setup & Testing

Setup

The HTTP server was set up using Python's Web Server Gateway Interface (WSGI) as an interface to the Apache web server. Specifically, Apache's mod_wsgi module was used to set up the video manager. An entry in Apache's virtual-hosts file was created to register the host name and port to the video manager directory. A .wsgi file was created to be invoked by mod_wsgi.

Testing

The video manager was tested using the following test cases:

- Downloading a video larger than the threshold when the cache was empty.
- Downloading a video larger than the threshold when the cache was occupied.
- Requesting a video that did not exist in Amazon S3.
- The video existed in the cache (hit).
- The video did not exist in the cache, but existed in Amazon S3 (miss).

Conclusion

This system, the Digital Democracy Video Manager, was able to allow scalability regarding Digital Democracy's video storage by keeping all videos on the cloud storage service- Amazon S3 – instead of on the server. Those videos stay on Amazon S3 until operations are to be performed on one of those videos. The cache uses the Least Recently Used algorithm to determine which video will be replaced when another video is requested. One important aspect that wasn't able to be done during this project was gathering statistics since Digital Democracy's video manipulation features have not been deployed yet. However, in the future statistics such as cache hits and misses can be recorded and different cache algorithms can be implemented besides the Least Recently Used algorithm. By implementing different algorithms, the most efficient one can be found for the Digital Democracy Video Manager.