R-Youtube - Replication for caching Youtube videos

Distributed Systems Implementation Report - Group 5

Aniruddha Gupta, Ankit Jain, Shushman Choudhury, Utkarsh Jaiswal

Abstract

This report focuses on the implementation of **R-Youtube**, a replicated system for caching Youtube videos so as to reduce the average load on the internet in IIT Kharagpur. There are three levels to the system - a single master server, multiple replicas of the master which can be considered as secondary servers, and the clients. In general, if the video exists in the system, it is sent to the client that requests it. If the video does not exist, it is streamed from youtube and cached as well.

1 Introduction

IIT Kharagpur has a single link of bandwidth 200Mbps for connection to the internet. With the population being over 10000, the average bandwidth per person is then 200kbps, which is not entirely adequate for streaming high quality videos. Given the popularity of Youtube in the campus, it is also reasonable to expect that a fair amount of that bandwidth is used in streaming Youtube videos.

To that end, we propose to cache these videos locally within the LAN. This would lead to a huge memory requirement if done on just 1 server, however, and that is where replication of the videos can be a way to solve this problem. We have designed a three-tier system, with each level meant to serve a specific purpose, described subsequently. In the following sections, we describe the design choices we made, how the various components of the system work, and how we tested the same.

2 Design

There are two kinds of replication that are going on here. The first is of cached videos, which are stored with the users of the system. The second is of our data tables, i.e. of the maps of user-to-video and video-to-user. In this context, the various design issues have been elaborated in this section.

2.1 Architecture

There are three kinds of entities in this system, each part of a tier or layer, as shown in Figure 2.1. Moving from bottom to top, firstly we have the *clients*, which are the users that request and cache videos. The videos are stored locally on clients and can be transferred from client to client, which will be described later.

Next we have what we call the *parent* servers, to which the clients send their requests and queries. Each client is connected to one and only one parent. Each parent server contains an

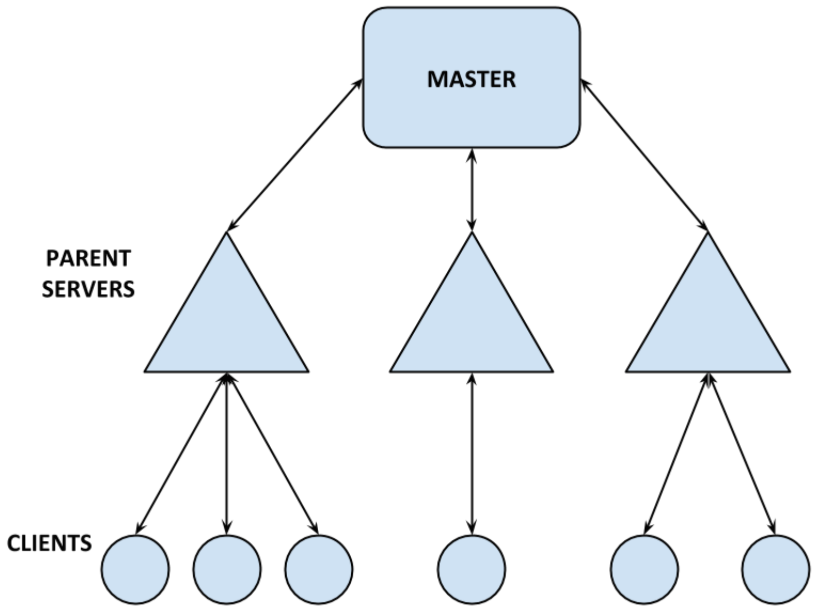


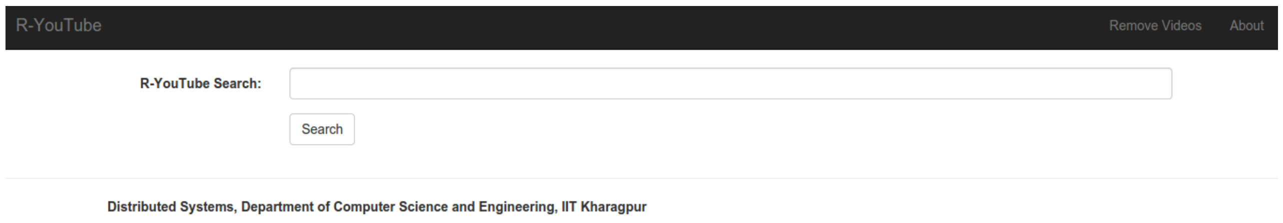
Figure 2.1: An overview of R-YouTube's three-tier system

immediately-updated set of data tables (defined earlier) for the clients it serves. Eventually, it will also obtain the data tables of the other parent servers in the system, due to the layer above.

The topmost tier of the system is a single *master* server, which can be viewed as having the primary copy of the data tables, with parents being secondary replicas. It has immediately-updated information about all the agents in the system. The parent servers propagate their own changes to the master as soon as they happen. They also forward client queries in certain scenarios. Clients do not directly interact with the master, except while entering the system. The IP address of the master is known to the clients, and on connecting to the master initially, the clients are provided a list of parent servers in increasing order of number of clients they serve - a simple instance of load-balancing.

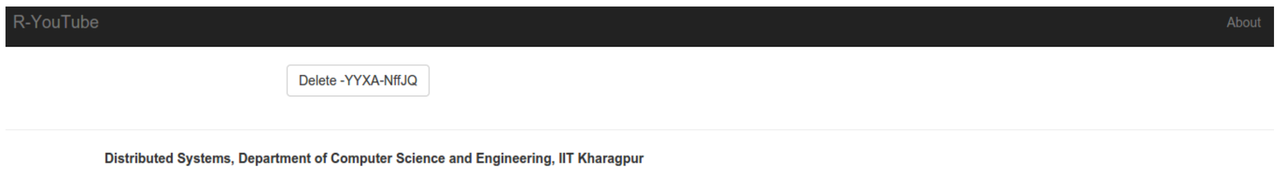
The general rationale behind the system is this - having a single master makes it easy to co-ordinate the system, keep track of how many clients are being served by each parent, add new parents and redistribute clients among parents in case of failure. If more advanced functionality is desired from the system, that can also be accommodated. The existence of multiple parent servers as secondary replicas help to restrict the number of queries sent to the master, thereby informing the clients very quickly about if the video is available in the system or not. This, along with the quick client-to-client transfer, will incentivize the user to utilize this system instead of directly using the internet for Youtube, which fulfills our high-level objective.

2.2 Interface



Distributed Systems, Department of Computer Science and Engineering, IIT Kharagpur

(a) A snapshot of the search page



Distributed Systems, Department of Computer Science and Engineering, IIT Kharagpur

(b) A snapshot of the delete page

Figure 2.2: Snapshots of R-YouTube

R-YouTube has a simple interface that allows users to *search* for videos, *view* them and also *remove* them from the local cache. We have implemented *remove* through our interface so that the system can be aware of removal immediately. However, even local deletes through the filesystem of the user, can also be handled, by checking the user's file list every 1 second. Search queries are re-routed to the YouTube search page. Selecting a particular video thereafter opens up a custom player embedded in the browser, that plays that video, after either streaming from YouTube directly, or from another client.

2.3 Consistency Model

Broadly speaking, there are two kinds of consistency ensured by R-YouTube. For the addition of new videos in the system, there is *eventual consistency*, through which the data tables of the parent server of the involved client, along with the master server, are updated immediately, while those of the other parent servers only get updated when the clients that they serve make queries of them that require the information of this update.

For the removal of cached videos (through R-YouTube), we ensure a stronger model of consistency. A client can only follow through with the deletion of a video once its parent, the master, and all other parents have been informed of this event, and have sent an acknowledgment of the same. The assumption made here is that deletions will happen much less frequently than other operations in the system, and the extra time is worth reducing the number of video queries that will lead to clients that do not actually have the video, thereby requiring another query.

3 System Operations

In the previous section, we have described some of the design choices and issues related to the system, and how that affects the operations that the system can perform. Now we briefly describe the working behind the various functionalities of R-YouTube.

3.1 New User

The IP address of the single master is known apriori to all new clients entering the system. Upon connecting to the master for the first time, a client is sent a list of IP addresses of all parent servers in the system, sorted in increasing order of clients currently being serviced. Therefore, the master tries to balance the load among the parents. Upon connecting to a parent, the client sends a list of videos currently in its cache to that parent.

3.2 Getting Videos

Users can search for videos through the interface of R-YouTube. A number of things happen when they do, as is outlined here:

- The search query is redirected to the YouTube page to show results.
- Upon clicking the video, our player opens up in the browser. If the user has the video locally, it is played from the local cache.
- If the user does not have the video, the client C1 queries the parent server P1 as to whether another client has it. The parent server checks its own data tables, and if it finds nothing, queries the master. If the master responds that another client C2 has the video, then P1 updates its data table and forwards the information to C1. If not, then C1's query is unsuccessful.
- If C1's query is unsuccessful, i.e. if the system does not know of any client with the video, then the video is streamed from YouTube into the video player, and simultaneously cached locally.
- If some other client C2 has the video, then there is a client-to-client transfer from C2 to C1, following which the video is displayed.
- The data tables of the parent P1 and the master are appropriately updated.

3.3 Deleting Video

The option to remove a video is exercised through a button on the R-YouTube interface. When this option is triggered, a notification is sent to the parent, and then to the master and all other parents, informing them that the video will be deleted. Only when they send back their acknowledgments of the same, does the video get removed from the local cache. This policy makes the deletes slower, but, as was explained earlier, helps us ensure a stricter consistency for deletes, and is acceptable due to the reasonable assumption that deletes will not be very frequent. We have also handled deletes from disk by checking the user's file list every 1 second, and updating the parent if a file has been deleted.

It is still possible of course, that stale information is sent to the user, if it makes a query for a video with another client at the same time the latter requests to delete it - but such scenarios will also be rare, and can be handled by just redirecting the client after it does not get the video, and updating the parent's data table accordingly. The same situation applies to when clients delete the videos from disk manually, i.e. without using R-YouTube.

3.4 User Disconnection

We assume that the movement of clients in and out of the system is fluid - clients may connect and disconnect frequently, and may choose to leave permanently as well. Keeping that in mind, the policy to handle client disconnection is described here:

- Clients periodically send heartbeat messages to their parent servers to inform them that they are connected. If no heartbeat is sent for 10 seconds, the parent is notified that the client has been disconnected.
- Each parent maintains a list of active users, i.e. the clients that are currently connected to it. When a user gets disconnected, it gets removed from the active list of its parent, and of the master.
- When a video is searched by some client, the list of clients returned (clients that have the video) is intersected with the list of active users, and that is the final list returned.
- Other parents, i.e. not of the disconnected client, are notified that the disconnection has happened when the result of a search made through that parent contains this disconnected user, thereby updating that parent's active user list.

3.5 Fault Tolerance

There are two kinds of failures that can happen in R-YouTube, the failure of the parent server and the failure of the master. For the former, there is a failure handling mechanism that has been implemented. The parent servers send periodic heartbeat messages to the master, and the absence of such messages indicates a failure of the parent. Hereafter, when the clients serviced by that parent cannot connect to them, they send a query to the master.

At this point, the master redistributes them among the remaining parents depending on how much load each other parent has. Necessary adjustments are made to the data tables.

The failure of the master is a bigger problem, as it is a single point of failure. It can have backups to increase the mean time of failure.

4 Implementation Details

Here we have outlined the plan for testing how R-YouTube might work in a number of different scenarios that might arise. We have also mentioned the libraries and open-source software components used to implement R-YouTube.

4.1 Test Plan

- Caching the YouTube video on the local file system
  - Test Strategy: Once the user watches a video, he should have it in his local file system after downloading it either from YouTube or from some other client.
  - Passing criteria: The client checks for the video id, and should find it in his local file system.
  - Fail Criteria: The watched video does not exist in the local file system.
- Preference to a fellow client over YouTube while downloading the video
  - Environment: Some other client should have a video which is not cached in the client's local file system.
  - Test Strategy: The client having the video should get a request from the client not having the video for downloading it. Also, once the client not having the video obtains the video, it should send a request to the parent server indicating that, which should then be acknowledged by the parent server.
  - Passing criteria: The client having the video gets a get request for the video
  - Fail Criteria: The client does not get any request for the video.
- YouTube downloading in case the video is not found in any other client
  - Environment: No client in the system should have a particular video.
  - Test Strategy: The client will be displayed the YouTube player once he requests the video, and subsequently will download it.
  - Passing criteria: The client should see the Youtube downloader caching the video in background and it should exist in the local file system after the download is complete. Also, the client should send a request to the parent server that he has the video, which should be acknowledged by the parent server.
  - Fail Criteria: Any of the passing conditions are violated.

- The client from which the video is deleted should not appear in subsequent searches
  - Test Strategy: A client will delete a video. Another client will request for same video.
  - Passing criteria: When a search result is returned from the parent, it is also displayed in the parent server's terminal as : "Search result : .....". The IP address of the client from which the video is deleted, should not appear in it.
  - Fail Criteria: The address of the client which deleted the video, is shown in the results.
- The video list is cached locally in a parent server
  - Test Strategy: 2 clients (C1 and C2) connected to a single parent P1. One client(C3) is connected to another parent P2. C3 has a video. C1 requests for that video, and then deletes it. C2 then requests for the video.
  - Passing criteria: Whenever a search goes to the master, it displays on its terminal "Search : ¡number of user having that video¿". When C1 requests for the video, the master displays "Search : 1". When C2 requests for the video in this scenario, the master does not display anything but C2 downloads the video from C3.
  - Fail Criteria: Either MS displays "Search :..." or C3 downloads the video from YouTube.
- When a client requests a video that is with another client connected to its parent, the search does not go to YouTube
  - Test Strategy: 2 clients (C1 and C2) are connected to a single parent. C1 has the video, and C2 requests for that video.
  - Passing criteria: Whenever a search goes to the master, it displays on its terminal "Search : ¡number of user having that video¿". When C2 requests for the video, the master does not display anything but C2 downloads the video from C1.
  - Fail Criteria: Either MS displays "Search :..." or C2 downloads the video from YouTube.
- When a client disconnects, it is not displayed in subsequent searches
  - Test Strategy: A client that has a few videos, disconnected from the system. Thereafter, a search request is made by another client for one of the videos the disconnected client had.
  - Passing criteria: The IP address of the disconnected client should not appear in the results of the query shown in the parent server. Furthermore, when the client is disconnected, the parent server should display "User ¡user IP address¿ removed"

    - Fail Criteria: The address of the client which deleted the video, is shown in the results.

- When a parent disconnects, its clients are not displayed in subsequent searches
    - Test Strategy: There are at least two parent servers P1 and P2, each with multiple clients. Subsequently, P2 gets disconnected from the master.
    - Passing criteria: The master should display ”PS not responding: ¡PS IP¿”. Also, the clients that had been serviced by P1 should request the master for a list of parent server IPs. The master displays a list of IP addresses which it has sent to the client.
    - Fail Criteria: The address of the parent server is in the list displayed by the master.

- Master acts as a load balancer
    - Test Strategy: There are at least two parent servers P1 and P2, with P1 having more clients than P2. A new user enters the system and requests the master for a list of parent server IP addresses.
    - Passing criteria: The master should display the list of parent server IPs in increasing order of load, i.e. P2 should be displayed before P1.
    - Fail Criteria: The IP addresses are not shown in the correct order.

4.2 Libraries Used

- Python with Django
- Java with Spring Framework
- Retrofit
- dl-youtube - YouTube Downloader
- Gradle Project Automation Tool
- BeautifulSoup

5 Discussion

The design and implementation of R-YouTube as part of this term project gave us nice insights into how a system is developed, and what factors should be kept in mind while doing the same. The same functionality can have multiple alternatives for implementation, and typically, each alternative has a different tradeoff that should be identified and weighed comparatively before making a choice.