Synchronized YouTube video playback application

CS 237 Project Slides (Group 2)

Apoorva Muthineni
amuthine

Chukka Bhargav
bhargavc

Tanvi Gupta
guptat2

Introduction

Our applications lets users watch videos together on youtube from over the internet. It features high accuracy video content streaming among the clients aiming at providing a ***seamless group-watching experience*** for the users of the application.

Design

With simplicity and high performance functionality in mind, we chose to leverage the simplicity of the ***client-server architecture***. This architecture choice allows us to have more ***fine-grained control*** over the video playback speeds at the application client. To offload the large number of responsibilities at the server, we create multiple server instances and utilize a ***load balancer*** to distribute the requests across the servers. To facilitate the distributed operations at the server, we use a ***publish/subscribe architecture*** between the multiple servers and use websocket topics to guide server-specific information to the rightful client.

Architecture

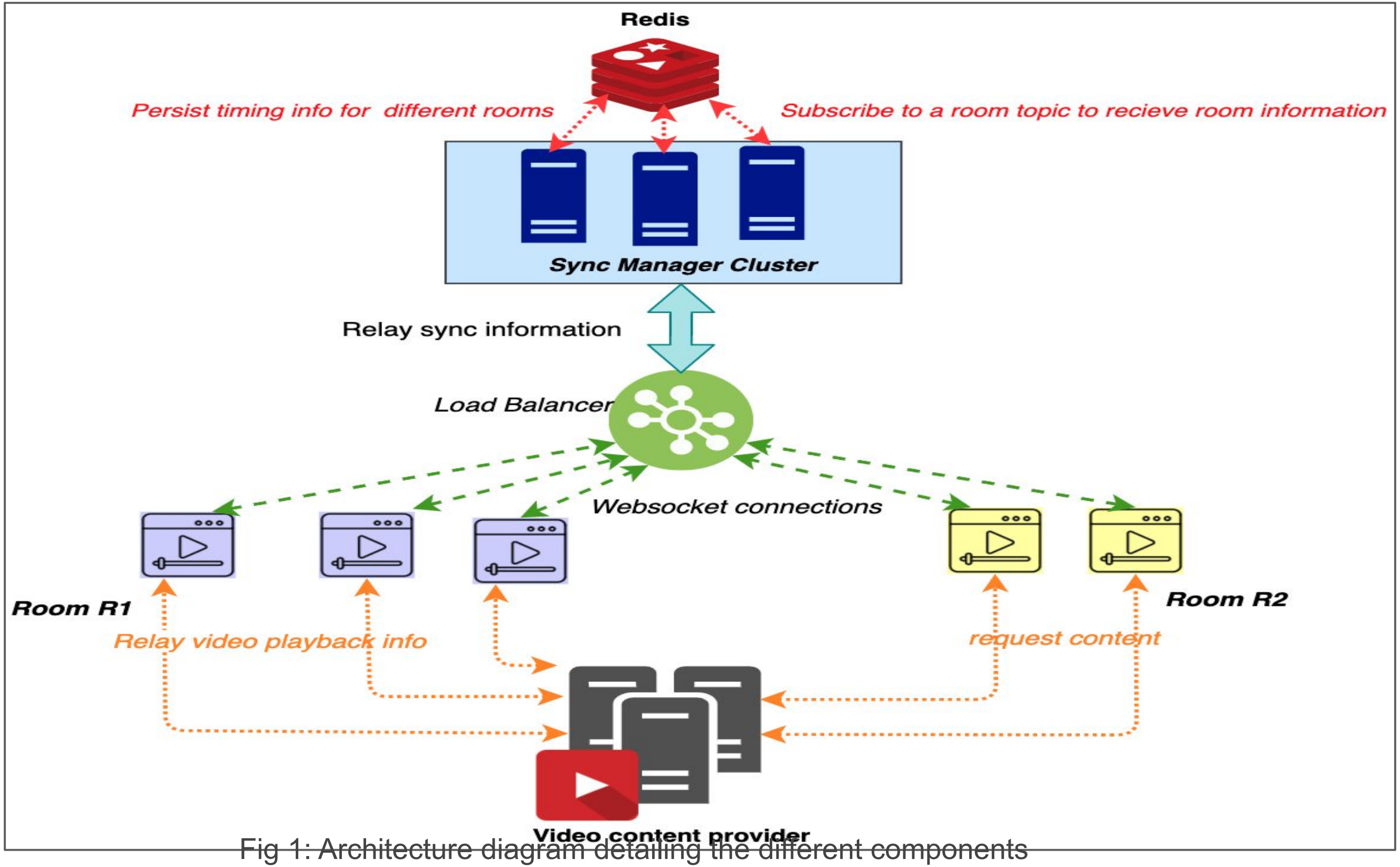
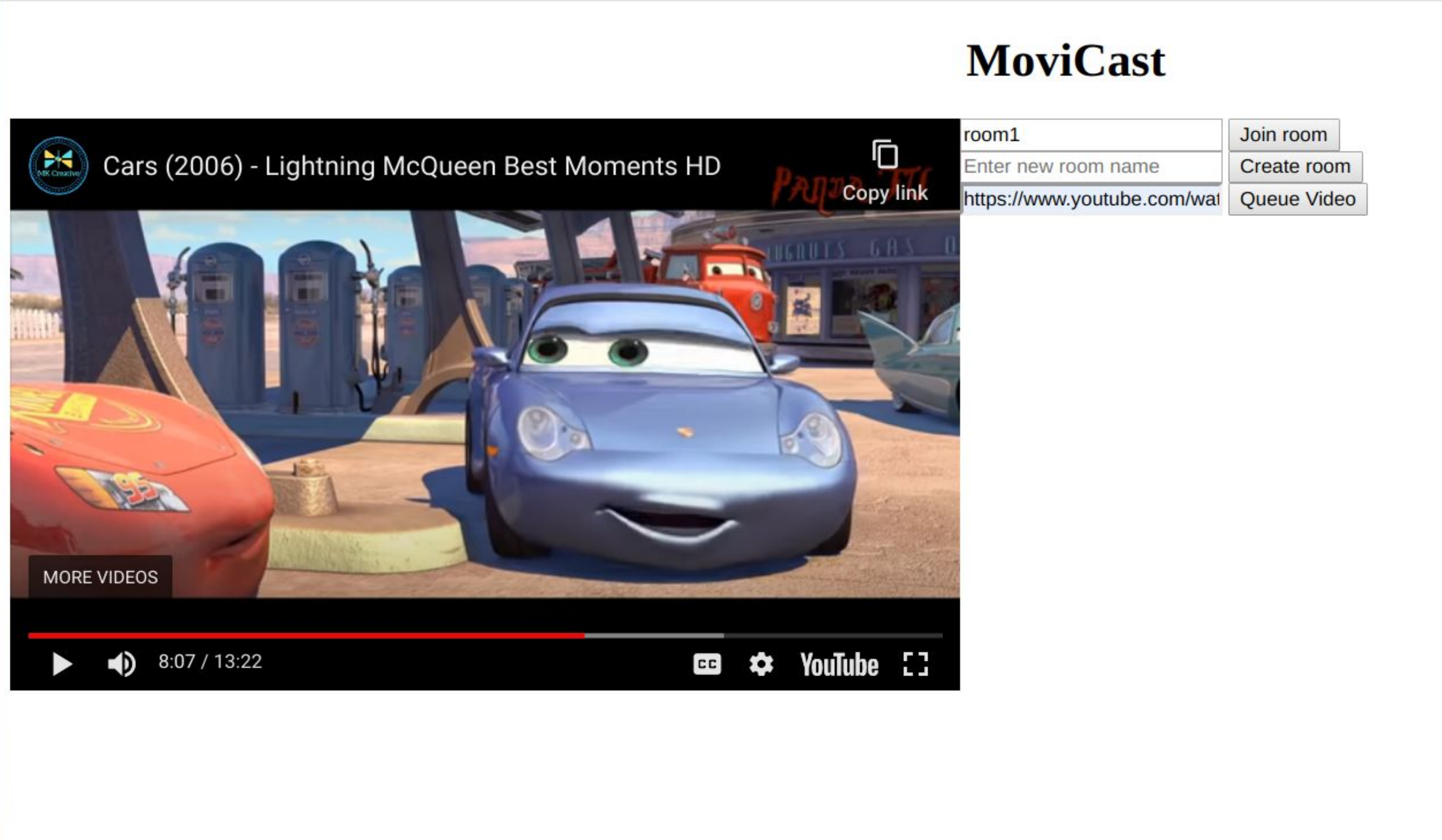


Fig 1: Architecture diagram detailing the different components

Sync Clients

- Clients have the ability to either create a room or join an existing room.
- Each client has control over a youtube video player and can specify the video URL used to play in the application.
- The sync clients communicate with the sync manager via *websocket connections*.
- A Websocket topic is created per room and the clients subscribe to their room topic.
- The possible playback events generated by a client are as follows:
  - PAUSE
  - PLAY
  - SEEK (change video position)
  - CHANGE VIDEO URL
- The video content viewed by the users is provided by the YouTube application.

Sync Client GUI



MoviCast

room1

Enter new room name

https://www.youtube.com/wat

Sync Manager

The sync manager module runs on a cluster of servers. It’s functionality includes:

- Servicing client requests to create a room, join/leave a room and maintain state information about each room.
- Onboards a new client to an existing room by fetching the information from storage(Redis).
- Receive video playback events generated by clients (pause, seek and play) and relay it other clients in the same room.
- Periodically calculates the *ideal* video position for the clients in each room.
- Persists the current state and periodic timing information received from the clients to storage (Redis).

Redis Cluster

- Multiple sync manager (server) applications receive information from different clients and store them in the in-memory data structures provided by the Redis cluster.
- Using the publish/subscribe feature provided by Redis, a channel is shared by all the sync manager servers to share the event information among themselves.
- The servers in-turn relays these events to the corresponding Websocket topics of the clients.

Scalability

- *HAProxy* is used as a load balancer for the Websocket connections to the sync manager servers.
- The load-balancer is configured to route new requests to the least loaded server.
- Once a Websocket connection is established between the client and server via HAProxy, the same connection is used for the subsequent communication between the client and server.
- Seamless integration of new servers to handle increased loads through HAProxy.

Failover

- When a Sync manager server goes down, all the associated clients (with that server) re-establish a new Websocket connection to the available servers through the load-balancer.
- When a server receives a *reconnect* event, it adds this new client information to its current clients list and uses this new connection for subsequent communication.

Implementation Details

- A javascript based frontend application is provided as GUI for the clients.
- SpringBoot (*v2.0.0.RELEASE*), built on the Spring framework, is used to deploy the server with WebSocket support in Java.
- A Redis cluster is used by the multiple sync managers to persist and receive client updates on a room currently serviced by the sync manager.

Operation - Joining an existing room

- Each client sends a request to join an existing room by entering a room name.
- The message gets forwarded to any of the available servers through the load balancer.
- The server persists the client information to Redis and sends back information which includes the client ID, current video URL and the video position to start streaming from.
- The client plays the video based on the received information.
- The client *periodically* sends its playback position to the server to sync across all clients.
- The server persists the periodic video information received from the client to Redis.
- The server then uses the received client updates to *periodically* calculate the ideal video position across all clients and relays the ideal position to the respective clients.

THANK YOU