Synchronising Video Playback Information within a Distributed Framework

Craig Bojko, Mu Mu, Nicholas Race

School of Computing and Communications, Lancaster University

c.bojko@lancaster.ac.uk, m.mu@lancaster.ac.uk, n.race@lancaster.ac.uk

Abstract: **This paper details a mechanism to distribute time coded objects to devices in order to synchronise and control video playback within a web environment. We comment on a three-tier architecture of server, client and second screen and the implementation of a system that offers the functionality to track playback sessions and distribute them to peer devices in order to fulfil use cases and generate a communal aspect around a particular piece of media content.**

Keywords: Media Synchronisation, Distributed, Video Playback, LIMO, Google App Engine, HTML5

1 INTRODUCTION

With ever increasing bandwidth and speed on the Internet, new methods of streaming television content have emerged. With the rise of IPTV, novel ideas have emerged on how we can integrate multimedia streaming better into our lives and how we can interact with what we watch on a daily basis. As mobile phones gain increasingly more processing power and faster connectivity, we can further develop and integrate these two mediums for a more immersive viewing experience.

The central theme of the presented work will be to design, implement and evaluate a system to allow an audience to view a set of video streams while situated in different locations, limited only by the reach of the Internet, and be able to communally discuss, interact and contribute to the application for future viewers. The system will be designed as a framework called the Distributed Video Playback (DVP) framework [1]. The system would require a synchronisation mechanism running through the application, ensuring all parties are connected and aware of the other's presence and current state, along with a method of being able to store data so as all devices are able to access the information and present the user with an immersive experience.

The system's capability to allow a distributed audience relies upon an architectural bedrock to provide the mechanisms to communicate between devices. However, the features and scenarios we can demonstrate and build on top of this architecture will exploit another framework developed within the EC FP7 P2P-Next project [2]. This framework is named LIMO [3], developed in collaboration with the BBC R&D, and is a mechanism to facilitate with interactive viewing of video streams. LIMO defines a way of synchronising additional material during a video stream; it does this by being provided with time-based objects, outlined in JSON based manifest files.

2 BACKGROUND AND CHALLENGES

At the core of the LIMO framework is the LIMO manifest, this is a JSON file that holds LIMO objects, needed by the LIMO engine to determine the start and end time for each LIMO object. A manifest can be designed for several different scenarios, such as quizzes, subtitles, chapters or more. A LIMO object contains a start time, duration and payload along with the possibility of other contents to facilitate in creating an event at the specified time within a playback session.

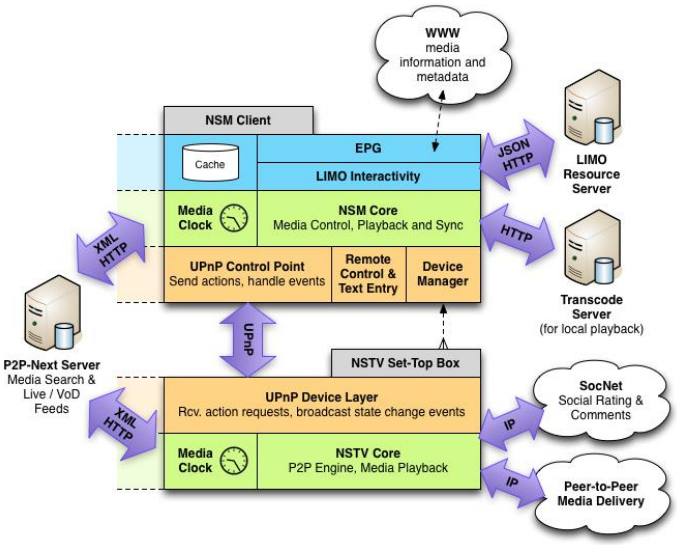

Fig. 1: LIMO Architecture

Figure 1 outlines the architecture of the P2P-Next system and the LIMO engine providing interactive objects within it. The LIMO [4] engine is the main application aspect of the framework, it sits between the manifest and the presentation device. We can describe it better as the controller part of a model-view-controller architecture, where the manifest is the model and the user interface or web browser is the viewer. The job of the engine is to parse the manifest and load the objects into usable states, which it can then determine at every time interval whether a state needs to be activated. The engine has two clocks, internal and remote. The internal clock resides within the engine itself, in terms of the web framework, this can come directly from the HTML video object. The remote clock would reside within an external application, this would be a second screen device to allow for synchronisation between the two platforms.

ISBN/EAN – 978-90-5986-439-9

2.1 Web Sockets

One notable feature of the new HTML5 standard is the introduction of JavaScript based web sockets. These are similar to any other type of socket within other programming languages, however, they are only able to open and listen for incoming data, while this is a limitation, when combined with HTTP requests this allows developers to devise push based mechanisms to allow clients to receive data as and when it is available rather than having to keep polling the server to check for updates, thus providing a transport mechanism for low latency, effective communication between client and server - an ideal mechanism to distribute synchronisation data amongst devices.

Web sockets are a way to reduce latency and server load compared to long polling methods like Ajax, as described in Nikolai Qveflander's paper [5] on implementing web sockets, we can see the enormous effect on performance these mechanisms provide to servers. Nikolai details the need and possibility to scale applications on distributed systems using web sockets and evaluates the effect this has, adding more servers and load balancers to increase performance. Google App Engine, the Cloud server used for the production of this system, provides much of this by default, as the server is already distributed among many physical sets of hardware, with load balancers to handle requests, all of which is transparent and unseen to the viewer and developer.

2.2 Other Synchronisation Techniques

A paper titled "Audio Watermarking: Features, Applications and Algorithms" [7] looks into how watermarking can be achieved in audio files. The paper outlines how to insert secret and public watermarks into audio files that also persist after compression. The paper also details several applications of watermarking, one of which is to transfer information, which the various other applications have utilised in order to create new interactive content for their viewers.

There are potential issues with this type of synchronisation, for example, if the area around the video device is noisy, it can interfere with the audio detection, possibly causing the mobile device to miss the sound clip and thus depriving the user of additional interactive content. There is also the problem of how accurate to a specific time point this method can link to. For instance if the audio clip being listened for is three seconds long, by the time the mobile device has detected, processed, looked up and received the data from the server, it could be five to ten seconds after the start of the audio clip. While careful placement and timing can cause something to happen relatively at an intended time, this method is more suited to providing additional information alongside the video stream, as opposed to fine-grained time dependent actions, this is the type of synchronisation we look at in this paper.

3 SYSTEM DESIGN

The overall architecture will be a three tier design, the first being the server level, second being the users client machine, and the third being second screen mobile device. The main architectural design is based on a client server model, however the server will be built on a distributed hardware platform to spread server load across multiple platforms and make the system more efficient.

Figure 2 shows the data communication paths between the three architectural tiers. The main data synchronisation will take place between the client and server; this will be to alert other clients as to the video position of each user. The mobile component, while optional, will also require sync data, which can be obtained directly from the source (client machine).

The server will feature a Model-View-Controller design pattern allowing us to separate the concerns and different aspects of the system. The models being the state objects

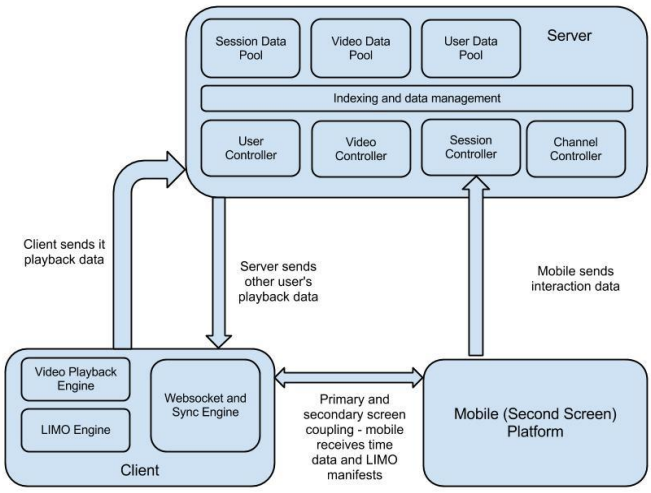

Fig. 2: System Architecture

relating to user sessions, video playback and synchronisation states, the controllers will allow client devices to interface with the server and provide the logic to alter their respective states on the server. Further to this, the server controllers will facilitate in the distribution of session and playback data to other clients connected within the same session pool. The views will be provided in the form of Java Server Pages sent to the client to visualise the data distributed amongst them.

The synchronisation packets will be transmitted at an interval of two to three seconds, incorporating local session data and the relative timestamp of the video playback clock. As the content will be the same across devices, the relative time will allow us to determine each user's location within the playback and thus open up the possibilities of remote management or use cases around the video content. The synchronization data transmitted via the server will allow devices to be aware of viewer presence and playback location across the distributed domain, whereas the internal synchronization of the LIMO engine will manage HTML events to provide added value to the video playback. Some use cases we focused on were the ability to have interactive quizzes during the playback, each question being triggered at specific intervals, another case is to time code comments that will trigger and be displayed during playback. This is similar to the way subtitles are

ISBN/EAN – 978-90-5986-439-9

displayed, however with commenting, we allow users to put back and augment the content whilst consuming it simultaneously, creating a community around a piece of content and allowing for direct discussion about a piece of content. Figure 3 shows the main video streaming and synchronised page with use cases implemented and other user playback positions visualised below the video window.

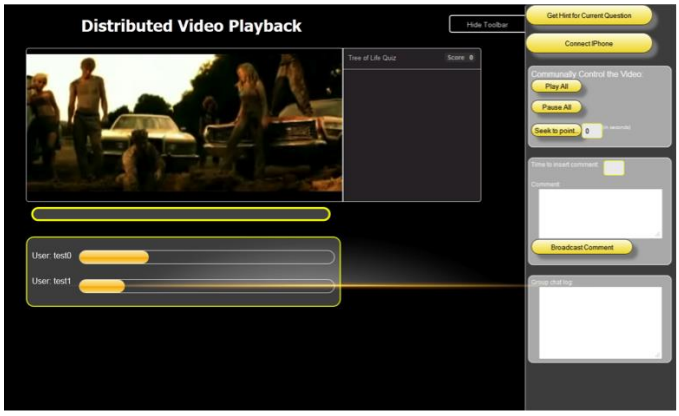

Fig. 3: Video Playback and Synchronisation page

4 IMPLEMENTATION AND EVALUATION

The primary focus in the implementation of the system was to build a mechanism to allow for group video watching on a wide area network. The entire framework centres around the ability to distribute playback information to groups of users connected within a session regardless of their physical location. The range of the system is limited only by the size of the Internet as the central synchronisation server is globally accessible to allow as many users to participate as possible.

The overall model for the system is every device requesting content from the server is connected into a lobby, designed to collate and distribute synchronisation data to other connected clients viewing the same content. The sycnhronisation works on a publish-subscribe model allowing for new clients to start receiving data from any point that they enter into the stream.

The system is implemented upon the Google App Engine platform, this being a cloud server offering application level APIs and services. The application is developed in the Java programming language, with server classes and servlets being completely Java based, and the front end using Java Server Page code to dynamically compile web pages before sending them to the user.

Many elements of the application front end will utilise HTML objects; this includes the video player, along with web sockets. The justification for this is that we can contain much of the application code within the JavaScript domain, building libraries of functions to support video manipulation and interface with the video object in turn with the LIMO framework. The web socket aspect of the HTML5 standard allows us to create socket level connections to the server to create push-based mechanisms to facilitate in user communication and synchronisation.

The client's requests are handled by servlets on the server. Each main aspect of the server has its own servlet, these being login and global session data, video data, the channel/socket functionality and the local video session data. Each data pool also has an index object associated with it, this is to help the application find and keep track of objects that are saved into the pool, and is updated at any time that an object is created or removed. The data pools are built on top of the Google data store, this is an abstracted form of the Java Persistence Manager, which is available in Java 6, and allows an application to save data within a local data store as opposed to a file on the physical disk. A unique identifier needs to be stored along with an object, this allows the persistence manager to correctly identify and load a requested object.

A limitation with the app engine, as is the case with many web servers, the developer is unable to create new execution threads, make system calls to the server or open sockets and pass data across networks. The business model of the Google App Engine is to provide cloud computing as a service, and abstracting many low level operations for security or to allow infrastructure engineers to assign global rules. While this sandbox model promotes better security and safety, it means that applications must stick to processing on the host server. However, the design of the engine promotes scalability for multiple users, as such developers can concentrate on the application and not have to worry about how the engine manages the processes [6].

The current method of synchronisation takes the current playback time and pushes this data, along with the client ID, via HTML5 web sockets to the cloud, with an optional failover to Ajax POSTing. The system tracks when the last update was sent based on a timestamp, optimising disk I/O by ignoring late packets, and aggregating individual client data over time, but persisting and pushing, in real time, the collective session data required by other clients within the same playback session. This reduces the amount of write operations on the server, but introduces some risk to individual users if something were to fail. A three second delay reduces the processor load on a client machine when the updates are aggregated and pushed to other clients within the session. Between the time synchronisation packets received by other clients, we infer the playback location of other client's video session until we receive a new update specifying any change within their state. Certain events such as a pause or seek request instantly push a new synchronisation packet to the server to update others as to the change as soon as is possible. Once received and collated by the server, the updates are then pushed back to the connected clients via the web socket, or on the Ajax return if using the failover mechanism.

A benefit to using the cloud service is as the system grows and more clients connect to the service, more backend servlets could be instantiated on the App Engine to reduce the overall load, and a thin middleware application could be used to assign an instance to a client depending on load and potentially where a distributed session is located to remove the need for session propagation or migration. Obviously this allows a developer to produce a single application and create

ISBN/EAN – 978-90-5986-439-9

new instances extremely quickly depending on system load without having to worry about server differences or initial setup procedures.

5 CONCLUSIONS AND FUTURE WORK

The introduced work had many successes within the distributed domain. The synchronisation running through a centralised source could be critically evaluated as a bottleneck, however, due to the fundamental distributed nature of the cloud server, the effects of a processing power bottleneck are minimised due to the pure power of the server. Future implementations may wish to address this as bandwidth concerns are paramount, or larger amounts of data are needing to be sent, one solution could be to implement a peer-to-peer mechanism to aid in synchronisation and reduce the necessity and reliance on a centralised service.

An experimental addition was also implemented to allow a desktop platform to run a local server to allow second screen devices to synchronise together and allow a group to partake in interactive video streams when either not connected to the Google App Engine or when bandwidth constraints prevented devices synchronising effectively. This addition could be expanded to include a peer-to-peer aspect and allow users to synchronise playback times directly without going through the Google App Engine. This would mean that a mesh network would be created to facilitate in synchronisation, but all data relating to interactive features would still be pooled on the server to allow a global authoritative source to ensure all peers have the same data on their peers.

The DVP framework is successful in providing a mechanism to provide distributed synchronised viewing of videos on the Internet, and with the aid of the LIMO framework can provide viewers with the opportunity to interact with other users joining them in the video streaming session. The evaluation of the system has also shown us the strengths and weaknesses of the system and how we can improve on these if the framework is continued and expanded upon.

The DVP framework could also be refocused for other types of applications. The possibilities for this type of application are endless, and some ideas may include remote management and communal Internet activities such as gaming, or more interactive social networking paradigms.

6 ACKNOWLEDGEMENTS

The work presented in this paper is supported by the European Commission within the FP7 Project: STEER (A Social Telemedia Environment for Experimental Research). The DVP framework has developed from the LIMO (lightweight interactive media objects) conceived by colleagues from BBC Research and Development department. We greatly acknowledge their contribution to the work reported in this paper.

References

[1] Implementation of an Interactive Distributed Video Playback Framework. Craig Bojko, 2012 http://www.lancs.ac.uk/staff/bojko/msci

[2] Next generation peer-to-peer content delivery platform (P2P-Next brief project summary). Technical report, 2008 http://cordis.europa.eu/

[3] RadLIMO Wiki Development Site: http://limo.rad0.net/wiki/

[4] P2P-Based IPTV Services: Design, Deployment, and QoE Measurement, Mu, M., Ishmael, J., Knowles, W., Rouncefield, M., Race, N., Stuart, M. & Wright, G.1/12/2012 In : IEEE Transactions on Multimedia. 14, 6, p. 1515-1527 13 p.

[5] Pushing real time data using HTML5 Web Sockets, Nikolai Qveflander, Aug 2010 http://www8.cs.umu.se/education/examina/Rapporter/NikolaiQveflander.pdf

[6] Google App Engine: Analysis, Craig Bojko, 2011

[7] Audio Watermarking: Features, Applications and Algorithms, Michael Amold, http://ieeexplore.ieee.org/stamp/stamp.jsp? tp=&arnumber=871531

ISBN/EAN – 978-90-5986-439-9