LiveSync: a Tool for Real Time Video Streaming Synchronization from Independent Sources

Marcello N. de Amorim
UFES
Av. Fernando Ferrari, 514
Vitória - ES, Brazil
novaes@inf.ufes.br

Ricardo M. C. Segundo
UFES
Av. Fernando Ferrari, 514
Vitória - ES, Brazil
rmcs87@gmail.com

Celso A. S. Santos
UFES
Av. Fernando Ferrari, 514
Vitória - ES, Brazil
saibel@inf.ufes.br

ABSTRACT

falar: da forma ativa e passiva (usuário escolhe o que sincronizar e o que não), de ser leve: servidor http e node, o resto é reaproveitamento dos servidores de vídeo. Desenhar e planejar as figuras a serem usadas. Focar em descrever o problema: streams de servidores independentes ....

2do.

This work presents a tool thas allows users synchronize live videos from multiple sources such as Youtube or any other video streamming source. The proposed approach to proceed the multiple camera video synchronization is based in crowdsourcing techniques, using the power of a crowd of colaborators to synchronize many videos, requiring from each user the sync of only a pairs of videos. Aditional sync relations are infered from the crowd contributions, using transitivities properties and an strutucture apropriated for this inference activitie.

CCS Concepts

•Information systems → Crowdsourcing; •Human-centered computing → Synchronous editors;

Keywords

live video; synchronization; crowdsourcing

1. INTRODUCTION

Multimedia systems allow the data streams integration of different types, including continuous (audio and video) and discrete media (text, data, images). Synchronization is essential for the integration of these media [?], and is focus of researches for a long time [?, ?].

Multiple camera video synchronization is one such research area within multimedia. Automatic video synchronization (AVS) is a form to synchronize multiple video streams. Multiple works and tools can be cited: [?] synchronizes videos in time and space, allowing to navigate between videos by resemblance and time. The synchronization works for multiple videos and different cameras; [?] presents video synchronization using audio analysis. Fingerprints are generated for each audio, and a comparison between videos is performed, taking into account the number of similar fingerprints: when there are more occurrences, chances of synchronization also increase; [?] shows another method for synchronizing videos based in their audio. Its approach uses Chroma analysis from audio, grouping and synchronizing audio from a same event. Videos generated by users are called UGV – Unorganized and Unsynchronized User Generated Videos; [?] works both as a primary and secondary source for papers in the area. It presents important results in automatic video synchronization, presenting a technique that analyses differences between frames to find synchronization points between videos. Furthermore, it presents many references that are related to automatic video synchronization. It also presents main challenges for automatic synchronization algorithms:

1. Wide baselines: Approaches that rely on matching texture features suffer from the limited invariance towards view point changes. Hence feature-based approaches not only exclude scenes where the cameras stand opposite to one another but also scenes with wide angle overlapping views.
2. Camera motion: A "shaking" camera renders many synchronization algorithms unusable. However, the application of video synchronization of casually captured multi-perspective events requires an algorithm that can handle also this kind of scenario.
3. Dynamic backgrounds: A changing background can confuse any synchronization algorithm in identifying the real object of interest.
4. Occlusions: Another problematic effect, especially for approaches that involve the tracking of features, is the temporary disappearance of objects of interest, as well as self-occlusions of multiple moving objects."

Other characteristic of most of the AVS is that they require processing full content to synchronize the videos, a inconvenient condition when we desire live video streaming synchronization. We propose in this work the use of crowdosurcing techniques to synchronize these videos. Crowdsourcing [1] in our scope refers to the use of the crowd as part of a computational problem that can be solved easily by a human than by a machine. The "easily" word can mean that the human approach: is cheaper, faster or can be done more efficiently by humans.

WebMedia '16 Teresina, Piauí Brazil

© 2016 ACM. ISBN 123-4567-24-567/08/06. . . $15.00

DOI: 10.475/123_4

In video synchronization, we know that humans can fulfil all challenges presented by [?]. A person can identify if two videos are synchronized or not independently of occlusions, change of the background, camera motion or view point changes. The main challenge is how to use the human abilities to synchronize the videos, and permit that other persons can benefit from these contributions. So our tool uses the power of the crowd to synchronize live streaming videos and provide a form that other persons that want to watch those videos can receive both videos and synchronization info.

The remaining of this paper is presented as follows: section 2 explains our approach to synchronize videos; section 3 details our tool; and section 4 presents our final remarks.

2. VIDEO SYNCHRONIZATION

Multimedia synchronization is about providing coherent playout of orchestrated contents. In a broader view, coherence in a playout addresses temporal, spatial, semantic and sensorial constraints related to the combined content presentation, although our work focus only on the temporal issue.

To synchronize the video streams we use an synchronization technique presented in [3]: the Remote Temporal Couplers for aligning videos. The technique solves the problem where each user is independent from the others, synchronizing contents content in his environment and the sources are also independent, such as is our case: we have multiple streams coming from video servers that don't share any information, and clients (the crowd) that receives and synchronize these contents.

For each video that need synchronization (we suppose that the videos have relations, so we don't focus on how to group these videos) we create a coupler that aligns each pair of videos. Each contribution from one user generates a coupler. When a common user access two videos (A and B), if there is a generated coupler for these videos, the videos are immediately synchronized, else the person that create the coupler himself.

Other characteristic of this technique is that we can imply unknown couplers from known ones. If we know the synchronization of Video A with B, and we also know B with C, we can imply the synchronization point of A and C.

At this point, a question arises: what happens if the presentation of one video is delayed (stopped to buffer) after it started being presented? Wouldn't the timeline be misaligned? Yes, it would, and all synchronization would be lost. There are two solutions for such problem: (i) resynchronize based on the time of the coupler; (ii) apply the same delay on the other video.

2.1 D.A.L.

Dynamic Alignment List (DALs) are estruturas capazes de definir a relação temporal entre diferentes assets multimídia. A matriz possui informações de duração, diferença temporal de inicio e os identificadores de cada asset, de forma que a partir da matriz é possível integrar um conjunto de objetos multimídia independentes em uma apresentação multimídia sincronizada. Besides these direct information that are fundamental to present videos synchronously, in a DAL we can store all contributions made by the crowd about the videos, this way we can always update the temporal relations when new contributions are made.

Além de permitir a apresentação de um conjunto de assets de forma síncrona, o uso da matriz permite a adição de novas relações, ou seja, a adição de novos assets a qualquer momento. Para isto basta adicionar o asset à matriz e relacioná-lo com pelo menos outro asset conhecido da matriz. A partir desta relação é possível calcular a relação do novo asset com os demais que fazem parte da matriz.

Como a relação de apenas um asset pode ser utilizada para sincronizá-lo com os demais, é possível fazer a relação de diferentes matrizes construídas separadamente, e que podem ser incorporadas em uma nova relação, gerando uma hierarquia de relações entre as matrizes.

A DAL derivates from a matrix m x n, com m e n números naturais não nulos, onde m=n is the quantity of videos that can be synchronized at that moment. Each position of the matrix represents the relation of two videos, in other words, the value necessary to align them, in our synchronization scenario, we can say that it is where we store a coupler. This value is calculated using the following equation:

$$\Delta_{i,j} = begin(id_{i,0}) - begin(id_{0,j}), para 0 < i, j \neq m \quad (1)$$

Onde begin(x) é o tempo em que um asset x inicia na apresentação. Um $\Delta_{i,j} > 0$, implica que o asset da coluna começa $\Delta_{i,j}$ antes que o asset da linha, caso $\Delta_{i,j} < 0$, o asset da linha começa $Delta_{i,j}$ antes do asset referenciado na coluna e caso $Delta_{i,j} = 0$, os dois assets iniciam ao mesmo tempo.

As previously said, the list is derived from a matrix, presenting different aspects. Firstly, it does not presents all cells of a matrix. In the matrix, $\Delta_{i,j}$ is equal to $Delta_{j,i}$, so we don't need to store both relations, and with that, only the superior part of the relations are stored. Secondly, each "cell" of the list, presents multiple properties to help in the management of the crowd contributions. As each relation can contain multiple contributions, we need to calculate a mean of these contributions to set as the actual Δ. Lastly, each relation has a new dimension: a list with all contributions for that relation, that identifies the value of the contribution and the user that made it (if we want to identify who made it).

Figure 1, shows the representation of a DAL. There we have 5 assets (videos in our scope) labelled: A, B, C, D and E. Each asset represents a struct with the assets: URI, label, duration and a list of relations with the other assets. Following the asset A we find its relations with the other assets: AB, AC, AD and AE. We do not need to represent AA because $\Delta_{A,A}$ is always 0. Taking teh relations from B, we have: BC, BD and BE. Again, $\Delta_{B,B} = 0$. Thus, we do not represent BA, because we previously represented AB, and as said before, we can find BA by negating AB ($\Delta_{A,B} = -\Delta_{A,B}$). And this will happens to the other relations.

In the end of the DAL, we have the contributions for each asset (represented in the figure by a cloud due to figure dimensions). Each relation can receive multiple contributions, and are stored in the list. These contributions are processed (means) and generate the value of Δ for its relations. Fields such as confidence factor and number of contributions are used to know how robust those contributions are, and if its necessary more contributions to verify the synchronization.

3. LIVESYNC

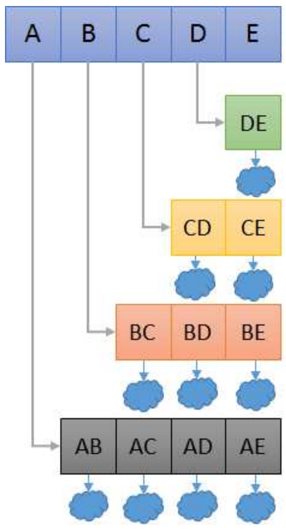


Figure 1: Dynamic Alignment List with 5 Assets (A,B,C,D,E)

Live synchronization includes the scenario where a viewer has access to an event that is live streamed by more than one content generator. These content generators are independent, so their videos do not have initial resources that allow their automatic synchronization to viewers, requiring a video analysis to generate synchronization points (Couplers). This synchronization fits as a problem that can be solved by using the power of the crowd. This occurs because videos are generated independently, without synchronization points and without previous description of what is about to be shown in screen, neither how can it be correlated to other videos. This way, human perception is used in real time to generate unknown synchronization points.

As example for this scenario, we can take a public manifestation. In the event, multiple people can take their cell phones and start streaming the event. In their house, other people can watch their videos. However, the multiple videos from different source will be asynchronous. We need then a way to synchronize the UGV. We use the crowd to achieve it. To this objective we group all videos in a mashup application that connects to a coupler server that contains all synchronization data. Both synchronizing and playing the videos are made using this mashup, that can receive videos from multiple sources.

3.1 Method

In a live presentation, it is assumed that content must be consumed right after its generation. It is of extreme importance then, that the synchronization method can be performed in playback time, to allow the integration of live content.

The way synchronization is performed in live video poses a single requirement: in a live situation, there is no need to analyse an entire video to find synchronization points, only an estimated stream delay must be considered. This occurs because the event is happening in real time (live), sources are also live and, therefore, the lack of synchronization during playback is caused by video streaming delays. In this specific case, synchronization through the crowd may be achieved by using tools that allow a user to manipulate time from videos: he can, for instance, keep pausing videos alternately , until he feels they are synchronized. This is a consequence of having a reduced search space, when limited to stream delays between videos.

However, not all viewers are required to be part of the crowd to achieve synchronization. If a synchronization made by a single member of the crowd is accepted as accurate, it can be transferred to the remaining viewers, this way each one will have his content locally synchronized.

One way of live synchronization can work is as follows: a person selects and synchronizes two videos with the help of a manipulation tool. This becomes a candidate synchronization point. Several people can do the same, and the results can be based on multiple synchronizations. Having these synchronization points defined, synchronization information can be sent to other viewers interested in watching those videos. As simple example, take two independent sources that are transmitting an event. A mash-up system allows the user to watch both videos at the same time is his device. However the videos are asynchronous and the user notes that. He then access the option to synchronize the videos. After he achieve a synchronous result, implicitly his contribution is sent to a server that will feed other users that choose to watch the same videos with the synchronization specification. If the user thinks the content is not synchronised yet, he can synchronize it himself and send another contribution.

3.2 Main Functionalities

The main functionalities provided by or tool are:

Synchronized Live Video Player - the tool permits users to watch multiple videos synchronized. He selects from a list of sources the videos he wish to watch and them they are synchronized using information provided by other users.

Video Synchronization - If a pair of videos does not have any information about their synchronization, users are invited to contribute and synchronize the videos.

Infer Synchronization - In some cases that there is no direct information about the synchronization of two videos, the tool is able to infer the synchronization about them, based on the contributions of other videos. We use the transitive attribute of video synchronizations, where if we know AB and BC synchronization info, we can infer AC. To infer this value we travel trough the DAL, finding the unknown relations (for example, CE), and try to find a path of known relations where we can infer CE. Taking the DAL in Figure 1, we can infer CE if we know: AC and AE. This a two steps rout, but try all possible routes when inferring, in a way that we fill as much relations as possible.

Video Aggregation - Although the focus of the LiveSync tool being on synchronization, we allow users to add new stream sources to the application. He only needs to set the video source, and the video will be added to the DAL and list of videos. However, we don't do any filtering about the added video, this means that the user can add any video to the application, even ones that contains none relation with the other videos. In future versions we plan to add options where other users can mark the video as not related, and then remove them.

Multiple Platforms Support - One keypoint of our tool is the use of other platforms as video sources. The videos that we play to users and that we synchronize, are not provided by us, but by other live video stream platforms. To be compatible with our tool, two requisites are required:

1. Remote Player: we need that the platform allows embeddable player on third pages, allowing us to control the player with its basic functionalities such as: play, pause and stop;
2. Uptime Support: a second and fundamental requisite is an API that allows us to retrieve the video Uptime. Video uptime is the time since the beginning of the video that is presented on the video player. This is fundamental to create and replicate the couplers generated in synchronization process.

Serverless Architecture - Serverless architectures refer to applications that significantly depend on third-party services and putting much of the application behavior and logic on the front end. Such architectures remove the need for the traditional server system sitting behind an application [2]. More of this characteristic will be addressed in the next topic: Architecture.

Multiplatform - LiveSync is a Web Based application designed and developed in compatibility with HTML5 standard to its front-end (MashUp Player) component. It allows our application to run on multiple browsers, operational systems and devices.

Active X Passive Contributions - Two branches of the LiveSync are currently on our repositories. They differ only in one aspect: who what videos are to be synchronized, the crowd or the application? The active version allows user to navigate freely through the videos, synchronizing them when they wish to. The focus of this branch is to allow users to contribute if they want to. On the other hand, the Passive branch gives the crowd exactly what video they will synchronize. The focus here is to rapidly synchronize all videos, so the focus isn't to make users watch the videos, but force them to synchronize all the base for other porpoises. The active branch is the focus here, but can easily converted to the passive.

3.3 Architecture

The LiveSync tool has three main components (Figure 2): the Content Providers (Video Sources), the Coupler and the Mashup Player.

3.3.1 Content Providers

Content Providers are third-parties videos streamers platforms. There are multiple Live Video Stream applications in the market, such as YouTube Live, LiveStream, TwitCast, Twitch and Ustream. One of our object was to allow the use of different platforms as video sources, so we maximize the number of videos for an event and allow user already in market platforms.

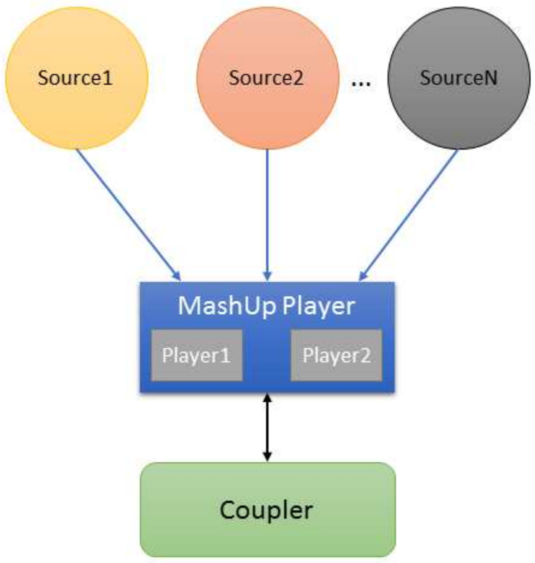


Figure 2: Modelo simplificado do live-sync

A Content Provider needs two requisites to be compatible with LiveSync: a Remote Player and Uptime Support. As each stream platform uses their own protocols, we opt to use their embeddable players into a MashUp application. These players must allow us to play, pause and stop the video stream. the second requisite, Uptime Support, is necessary to find the couplers among the videos. Uptime is the time passed since the beginning of the live stream until the video part being presented in the player.

At this time, LiveSync supports to video streamers platforms: the YouTube Live (https://support.google.com/youtube/answe and WebSocket Stream (http://phoboslab.org/log/2013/09/html5-live-video-streaming-via-websockets). YouTube live permits live stream from both desktop and mobile devices, allowing users to stream any event they wish with low effort: they only need to install the software and a YouTbe account. WebSocket Stream is an open source project that allows developers to implement live stream services with websockets and canvas and WebGl technologies. It works in any browser and was perfect for our first tests where we needed full control of the stream, something that YouTube Live doesn't support.

To add an video source form a Content Provider, it necessary only the video ID from YouTube Live or the video stream URI from the WebSocket Stream. These are added as assets to the LiveSync and can be accessed or synchronized.

3.3.2 Coupler

The Coupler is responsible for storage, distribution and calculation of synchronization points among videos streams from the Content Providers.

A coupler is composed of a DAL instance and Log files. This goes in direction of the Serverless Architecture. We wanted an architecture that needed low resources (another justification for using third party stream services) and easy deployment. All that is necessary do execute the coupler is a NODE.JS (https://nodejs.org/en/) server instance. This is possible because the couple is fully developed in and compatible with actual HTML5 standards. To deploy the Coupler, we use a Backend as a Service or "BaaS" platform, more specifically we use the Heroku one, that permits free use of NODE.JS instances.

It stores synchronization information only during the duration of the event, so its stance is finished with the end of the videos and all data is lost. In the current scope, the sync info is only necessary during the event, after it, there is no need to store the information. For reasons of testing and using the filmed videos from YouTube we create log files that contains all contributions made by the crowd. If it is important to maintain all contributions and data for post analyses and further use, unstable version of the LiveSync is being configured to use a fully transactional database. we use a fully transactional database because we want to maintain track of all contributions made by the crowd, something important in crowdsourcing and that is supported also by the DAL.

Other aspect of the Coupler is that it is responsible for the distribution of synchronization couplers. When an user chooses two videos, a message is sent from the MashUp to the coupler, containing the required relation. The coupler then answer with the required information. If the relations is unknown, it answer soliciting the user to synchronize and contribute with those two videos.

The last function of the Coupler, is to calculate the synchronization points among Videos. Each relation ($\Delta_{A,B}$) may contain several contributions, then it is necessary to calculate a value to that relation based on the contributions. In the current version we calculate a Geometric Mean of the contributions to find an ideal value. This however may not be the best value, because the more accurate the sync is, the better results, so older contributions must have a smaller weight, something that does not happen now. The other calculation made by the coupler, is to infer unknown relations based on the afore mentioned transitive relation.

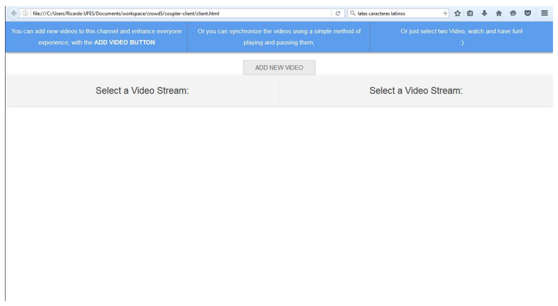
The communication Coupler - MashUp is made through Websocket [REF] communication. The MashUp creates a WebSocket channel with the Coupler, and requests the sync information or sends contributions from the crowd. A simple protocol is used in Json messages: act:value, data:object. The act field contains the action to be made and the data contains an object to complement the action. As example we have an act to send a new contribution ("contribution") that is complemented with a new relation that contains the assets involved, the value and an id to that contribution.

3.3.3 MashUp Player

Mashups are applications generated by combining content, presentation or other applications functionalities from disparate sources. They aim to combine these sources to create useful new applications or services (the offer and consumption of data between two devices) to users [?]. In LiveSync we combine videos coming from different sources and platforms with the synchronization information from the coupler to reproduce a synchronous presentation of these videos.

The MashUp Player is responsible for both presenting video synchronously and collecting the synchronization. Figure 3 represents the interface of the MashUp.

On the top we have all information necessary to the user. He can aggregate new videos using the ADD NEW VIDEO options, SYNCHRONIZE the videos if he thinks the videos are not synchronized or he can just select the videos he


file:///C:/Users/Ricardo UFES/Documents/workspace/crowdS/coupler-client/client.html
latex caracteres latinos
ADD NEW VIDEO
Select a Video Stream:
Select a Video Stream:

You can add new videos to this channel and enhance everyone experience, with the **ADD VIDEO BUTTON**

Or you can synchronize the videos using a simple method of playing and pausing them;

Or just select two Video, watch and have fun! :)

Figure 3: Modelo simplificado do live-sync

want to watch. When just playing two selected videos from the videos list, each video player creates an instance for the player that is compatible with that source (YouTube or WebSocket). It is invisible to the user where the video is coming from.

When the user adds a video, an input text is shown to him, and he can add the video URI (WebSocket) or video ID (YouTube). The page reloads and the new video is listed in the video list for every one that connects to the application. When the video is added by the user, an message is sent to the Coupler, containing the action to add a new asset to the DAL, and the specification of it, such as label and URI.

The last functionality of the MasUp is to synchronize the videos. When the user clicks on SYNCHRONIZE, a new mode of the application is revelled showing the synchronization tools. We use a Play 'n Pause approach to synchronize the videos. After he thinks the videos are synchronized, clicking the DONE button, his contributions is sent to the Coupler and stored in the DAL for further processing of the relation.

Play 'n Pause.

If two videos with a certain degree of similarity, are presented to an individual from the crowd, he can possibly notice that one video is ahead of the other. This way, he can pause the video that is ahead on time while the other remains playing, until reaching a point of synchronization. Then, the user can resume playback of the first video. This process can be repeated until an individual feels like both videos are being presented synchronously.

4. FINAL REMARKS

2do. Citations to articles [?, ?, ?, ?],

Fazer um teste de precisão: sincronizar em um computador, grava a tela do outro e ver qual foi a precisão.

Limitations: one instance = 1 room for 1 event. YouTube and WS Video Server only.

Within this work publication, the code will be made public through a GitHub repository under a Common Core License.

5. ACKNOWLEDGMENTS

The authors would like to thank the "Fundação de Amparo à Pesquisa e Inovação do Espírito Santo" (FAPES) for financial support.

6. REFERENCES

[1] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.

[2] M. Roberts. Serverless architectures.

[3] R. M. C. Segundo and C. A. S. Santos. Remote temporal couplers for multiple content synchronization. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 532–539. IEEE, 2015.