2010 14th International Conference Information Visualisation

A System for Real-Time Transcoding and Delivery of Video to Smartphones

Lior D. Shefer and Francis T. Marchese
Pace University
Computer Science Department
NY, NY 10038
{shefer.lior@gmail.com, fmarchese@pace.edu}

***Abstract*—This paper presents a report of a system that delivers customized video content to mobile devices. Constructed from open source components, it can stream transcoded video to mobile devices in real-time. In addition, it allows publishers to add content into their video streams on-the-fly. As a demonstration of this system's capabilities, an application was designed to deliver transcoded Adobe Flash content to Apple iPhones in real-time with the insertion of randomly selected commercial content.**

Keywords-mobile computing, multimedia, Adobe Flash, Apple iPhone, video transcoding, open source.

I. Introduction

Access to online video content has increased significantly. As of November of 2008, Americans spent 12.5% of their time online viewing videos [1], watching over 14 billion clips a month [2]. Fifty-seven percent of all Americans now have high-speed Internet access at home, a number that is expected to rise to ninety percent by 2012 [3]. With users increasingly connected to more powerful networks, it is only natural to conclude that these users will expect their mobile devices, particularly smartphones, to deliver this same content at a similar level of quality. However, there is a major stumbling block – Adobe Flash.

Adobe Flash is the standard format for delivering online video content to traditional computers, because of its high-quality/low bit rate (small file size) capabilities. Nearly 99% of online users are able to view Flash files [4]. Due in part to Flash's significant power consumption, delivery of Flash encoded videos to mobile devices, such as smart phones, remains problematic [5]. Adobe itself has addressed this challenge by partnering with processor manufacturers such as Texas instruments to furnish versions of Adobe Flash Player optimized for smartphones and internet devices based on these chips [6], with a beta version of Flash Player 10 that had been expected to appear in October 2009 [7]. But Apple's iPhone and recently introduced iPad will not run Adobe's Flash Player, even though both Apple and Adobe have pledged to solve this problem. And given that the iPhone owns nearly 50% of the United States smartphone traffic [8], this remains a growing problem with no clear end in sight [9].

In sum, the delivery of Internet-based video content to mobile devices remains an open problem. The part of the problem we consider here is the transcoding of video for smartphones in order to address the question of who is responsible for transforming video into a form understood by the mobile device - the hardware manufacturer (e.g. Texas Instruments), the software developer (e.g. Adobe), or the smartphone manufacturer (e.g. Apple). It does this by bypassing these stakeholders completely; building an extensible platform based on a loosely coupled collection of components founded on open source technologies.

The system prototype we present is called Vmoox (wordplay on Video multiplexer), a customizable and scalable video encoding and publishing system for a web publisher that includes a smartphone client application. Vmoox not only delivers video content to mobile devices in real-time, but allows publishers as well to inject additional content into video streams. The version of Vmoox demonstrated here transcodes Adobe Flash for Apple iPhones and injects random commercial content.

In the following section we will briefly cover the problem background. Section 3 contains the system design. Sections 4 and 5 present the implementation and a sample session, followed by a discussion in Section 6.

II. Background

Transcoding from Internet video to smartphone requires a codec that a smartphone understands. Such a codec is the H.264/AVC open standard codec developed by the ITU-T Video Coding Experts Group [10]. The advantage of H.264/AVC for smartphones is two-fold: a small file size that contains high quality, MPEG-4 compressed video, and a wide range of possible implementations that currently include Research in Motion's Blackberry, Palm's Pre, and HTC's G1 with Google, and Apple's iPhone.

A number of commercial solutions are available for transcoding video that include Flash [11] - [17]. However, these systems consider only a part of a video content publishers workflow. For example, these systems do not address the issue of smartphone 'apps.' It remains the publisher's responsibility to develop and deploy 'apps' for each mobile platform. Indeed, smaller publishers may not even have the requisite resources to develop 'apps' in-house or be able to provide the support necessary for maintaining

1550-6037/10 $26.00 © 2010 IEEE
DOI 10.1109/IV.2010.75

and delivering their video content. In addition, smartphone technology and the 'apps' that run on them are still in their early stages of development. Apple released its iPhone in July 2007 and Google's Nexus One was introduced in January 2010. Add to this that video encoding schemes along with the means of video delivery remain areas of active research [18] [19], particularly for the delivery of high-definition video content [20] [21] [22]. Thus, the challenges facing organizations that wish to deliver a diversity of smartphone content range from start-up costs to continued development, and the long term maintenance of these systems – costs that small and modest size organizations may not reasonably be able to bear.

One natural solution to this problem is to create a complete system that addresses both the consumer and content provider. Such a system should be open, flexible, and have the ability to add, delete, or adapt components at will. There are number of ways in which to do this. At one extreme is to build the system from scratch; at the other is to design the system so that a collection of independent parts may be integrated to meet the requirements of the stakeholders. This latter approach was taken, creating a system from pre-existing, open-source technologies.

III. System Design

Vmoox is designed as a loosely coupled collection of modules. Different modules in the system interact with each other through the API regardless of how each component has been implemented. This design improves the software's maintainability and readability as well as opening up Vmoox to third parties who may use Vmoox in combination with their own software applications.

The two main services which Vmoox provides are a publisher service interface and a video transcoding service. The publisher service interface is an API that enables publishers to initialize accounts, retrieve data from their websites, distribute video content to mobile devices, and perform updates as needed. The video transcoding service performs offline and real-time encoding transformations of Flash to H.264/AVC videos. It is designed to support the publisher service interface as part of an offline initialization setup and update, as well as real-time on-demand encoding requests. This is accomplished by Vmoox's five core components (Fig. 1):

1. Video publisher service interface (VPSI).
2. Video Encoding Service (VES) that performs offline and real-time encoding and thumbnail creation.
3. Web service layer that powers both native and web application layers employing VES for real-time encoding.
4. Service engine that provides pre-roll video content before each video clip plays.
5. Front-end application that enables users to consume publisher's video content on a mobile device.

We will discuss each component in turn.

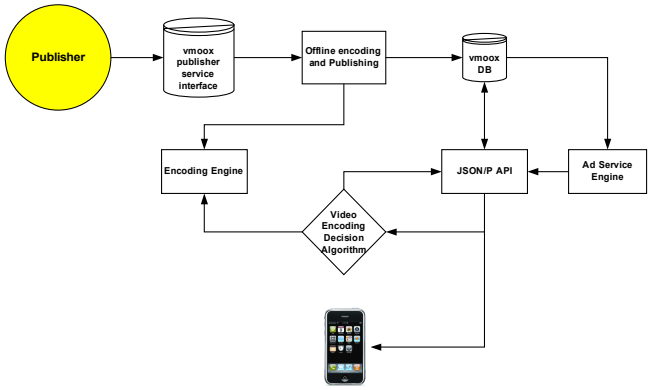

Figure 1. Vmoox architecture

A. Video Publisher Service Interface

The Video Publisher Service Interface (VPSI) enables web publishers to distribute their video content on mobile devices. It retrieves data from a publisher via an API to initialize accounts and perform updates when new content is available. As part of account initialization, VSPI generates a Vmoox internal user Id that identifies the video's owner and its meta-data. This information is stored in XML format within Vmoox's database and includes the following fields: video_id, image_url, tz_image_url, splash_image_url, video_url, tease_txt, can_syndicate, and vid_duration.

Once an Id is provided, the publisher may begin initializing an account that integrates its data within the Vmoox system. Integration is customizable so that videos may stream from either the publisher's or Vmoox's data center. If an external integration has been defined, the Vmoox publisher service will upload the original video content to its own server and pass the video to the Video Encoding Service (VES) so as to immediately create thumbnails and initiate a partial H.264/AVC encoding.

Finally, as part of the initializing process, the publisher receives its own customized iPhone application.

B. Video Encoding Service

The Video Encoding Service (VES) performs offline and real-time encoding transformations. Its core functionality is to transform Adobe Flash-based video content into H.264/AVC encoding so mobile users can consume this content. VES supports both the publisher service interface as part of the offline initialization setup and update process, as well as real-time on demand encoding requests. The offline encoding service receives the original Flash based video content and decides which part of the original content should be encoded. As an initial setup, the video is divided into three minute "chapters" and the first chapter (i.e., three minute section) of each video is immediately transcoded, although this variable is configurable. (More information on chapter division and the decision to transcode only the first three minutes of each video is provided in the discussion section of this paper.)

Once chapters have been produced and the first three minutes transcoded, VES creates thumbnails and a playlist of

chapter names with respective durations. The offline encoding and publishing service concludes by updating the Vmoox database. From here on the iPhone application is ready for use.

C. Web Service Layer

The Vmoox web service supports both web-based and native iPhone applications (Fig. 2). The web service provides video information for the following search criteria supplied by the user:

a. List of randomly selected video's meta-data such as: title, description, duration, and image.
b. Relevant meta-data for a given video Id.
c. List of videos' meta-data that contain a given keyword in the video title and/or description – this supports the search functionality.
d. List of videos' media URLs for a given video Id.
e. Media service URL for a given video Id. A service module appends pre-roll video content before serving the publisher content.

The Web Service works in the following way. When a user selects a video file from the smartphone app an HTTP GET request is sent to the Web Service. The Web service then retrieves a list of chapters for the video and pre-roll content (e.g. ads). This list is returned to the app.

In a synchronous process, VES determines which of the chapters have been transcoded and transcodes the chapters that have not. Chapters that have already been transcoded are saved in the system and marked as such to avoid transcoding content more than once per video.

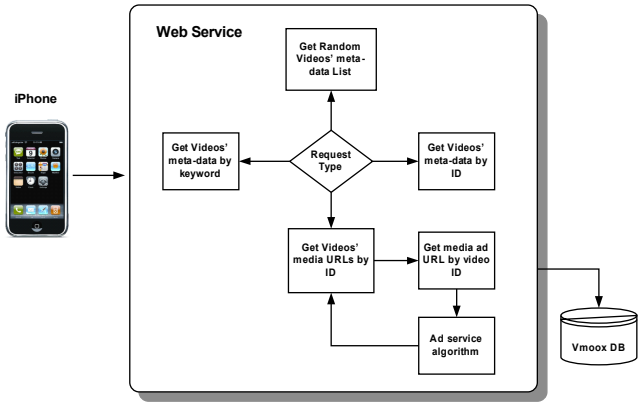

Figure 2. Vmoox's web service interface.

D. Service Engine

The Vmoox web service provides a video's meta-data and H.264 media playlist links for each video Id. As part of a video's play-list, the system includes a link to a pre-roll video. The system stores a reference to publishers of pre-roll content to ensure the correct association between video and content. The system offers some intelligence for serving pre-roll content that insures the same content will not repeat during the same user session.

The service engine can inject any kind of video content. For example, if the Flash videos supported an educational institution's delivery of teaching materials, then timely announcements could be sent along with the video feed. Another use is as a revenue generator. Publishers can include appropriate pre-roll ads as part of their video content. As illustrated in Fig. 2, when a request to get videos' media URLs enters the Vmoox web service, the web service issues an addition request to the ad serving engine to receive the proper ad(s).

E. Front-end Application

Vmoox supplies each web publisher with its own customized smartphone application that is a front-end to the Vmoox system, enabling users to consume the publisher's content on their mobile devices. The design of the front-end application represents a consideration of several key structural attributes: mode of delivery, customizability, and conformance to the model-view controller (MVC) design pattern.

Publishers have the ability to receive either a native smartphone application (specific to a particular device) or a web application that runs in the browser of any device. Both platforms share the majority of the code base, that is the Vmoox Web Service is dedicated to these platforms, while Vmoox's other modules such as VEDA and the transcoder are platform independent.

Each publisher is able to customize the front-end application's appearance to fit its own brand identity, target audience, and as functionality. Some publishers may prefer to control the order of videos presented, while others may prefer a random selection. Another requirement is the ability to control the amount and subject matter of pre-roll ads before each video.

Finally, the front-end application's architecture uses the MVC design pattern. The benefit of using the MVC design pattern is the ability to create independence among components. Data access code, business logic code, and presentation code are all separated. This allows the creation of different views for different publishers according to their specific needs. Maintaining low coupling between different types of classes, or the decoupling of data access, business logic, and presentation code make classes easier to maintain and reuse.

IV. Implementation

Vmoox was built with Java and runs on an Apple MacBook Pro powered by an Intel dual-core processor with 4GB of RAM. Vmoox's sample smartphone *app* was created for the iPhone employing Apple's iPhone SDK OS 3.0 [23]. VPSI, VEDA, the Vmoox Web Service, and the smartphone web *application* were all implemented utilizing the open source frameworks Hibernate [24], MySQL [25], Xstream [26], JSON [27], FFmpeg and libavcode [28], and GWT [29].

Hibernate is an object-relational mapping library for the Java language that provides a framework for mapping an object-oriented domain model to a traditional relational database and provides data query and retrieval facilities to

MySQL, the relational database management system used by Vmoox.

Xstream is an open source library that serializes Java objects to XML and JSON, and vise versa. When designing a web service it is often necessary to implement several end-points to services. For example, some clients may prefer to interact with the service's data through XML, while other types of applications may prefer JSON or pure Java objects through RPC or SOAP. XStream provides an abstraction layer on top of a business logic rules layer by separating business rules and presentation simplifying web service design through loose coupling. The main advantages of Xstream are its ease of use, uncomplicated object mapping, and performance. Xstream provides an abstraction layer on top of object serialization. Common use-cases are easy to implement and use. Most objects can be serialized without need for specifying mappings. Xstream serialization is fast and requires low memory consumption. Vmoox's web service uses XStream to afford clients a JSON/JSONP RESTful [30] service. Vmoox web service responds to a HTTP GET request by generating a java object using Hibernate. XStream then serializes these Java objects into JSONP and posts them back to the request.

JSON/JSONP is a lightweight protocol that enables developers to easily connect to Vmoox's web service and to use Vmoox's data in their own applications. JSON is a data format that naturally fits browser data consumption because it is a subset of JavaScript and can be easily parsed by a browser.

FFmpeg is a cross-platform framework to record, convert, and stream audio and video. It contains libavcodec, a library of codecs for encoding and decoding video and audio data. Vmoox uses FFmpeg and libavcodec to transcode Flash video files to H.264/AVC, create thumbnails of video files, generate chapters within the original video files, and obtain information about the files themselves.

GWT, the Google Web Toolkit, is designed for building and optimizing complex browser-based applications. The GWT SDK supplies a set of core Java APIs and Widgets that support writing AJAX applications in Java and then compiling these sources into optimized JavaScript that runs across all browsers, including mobile browsers for Android and the iPhone. Vmoox's iPhone app was implemented using GWT following the MVC design pattern.

V. Sample Session

Using CNN as a sample publisher and a native iPhone application as the sample client implementation, the sequence of steps that constitute the Vmoox service workflow are as follows:

CNN provides Vmoox with an XML file containing its videos and their meta-data as well as pre-roll content along with pairing criteria (i.e., which ads should be paired with which content). Vmoox's publisher service interface imports CNN's original Flash files onto its server and logs the meta-data in the Vmoox database. The video encoding service divides each video into three minute chapters, transcodes the first chapter, and creates a thumbnail for each video. At this point a native iPhone application is delivered to the client as per client design criteria. A user downloads the application and selects a video from the Featured View list (Fig. 3a). This is the first view a user sees upon launching the app and includes the latest (or preferred) video content from the publisher. When the user selects a video by clicking on the video image or the blue arrow directly to its right, the iPhone app sends an HTTP GET request to the Vmoox server which responds in turn with a JSON/JSONP list of chapters (Fig. 3b). Meanwhile, the video encoding service decides which chapters need transcoding and begins transcoding the ones that do. Video content is then paired with pre-roll content as per client specifications. Clicking on the video thumbnail in the Video View (Fig. 3b) initiates the video - playing the pre-roll content (Fig. 4a) and video (Fig. 4b).

Figure 3. Two views of the iPhone application: a) Featured View, b) Video View.

Figure 4. A video being played with a) pre-roll ad, b) video content from CNN.com.

VI. Discussion

The Vmoox application was tested over a 10 Mbit wireless network with an iPod touch as a client. The iPhone app was tested using Apple's iPhone simulator running on the Vmoox host computer. In the latter case, the simultaneous processes of pulling video content from CNN, its real-time transcoding, monitoring of all processes, and playing on the simulator showed no degradation in video delivery. In the former case, no visible latency was detected. If this had been so, it would have been an issue with the network, not Vmoox.

While the CNN native iPhone application works well as a prototype, some changes to the design of the Vmoox system will need to be made as individual publishers purchase the service. Each client has a unique set of requirements and both the application and service will be modified slightly in each case to fit those.

One of the major components of the Vmoox prototype is the Video Encoding Service that divides videos into 3-minute chapters. This time span is adaptable, given a client needs and specifications. Here the decision to segment video files into 3 minute chapters comes from a desire to minimize waste of server space and computational resources by only performing a full transcode on files which are actually viewed. An initially transcoded segment provides a time buffer with which to avoid any wait time on the viewer's end. Simply put, while the user watches the first 3 minutes of the clip, the remainder of the clip is transcoded in the background. In the future, Vmoox's efficiency may be improved further by including a "stop transcode" feature that would identify when a user has stopped viewing a clip so the system may be instructed to cease transcoding any remaining chapters. Such an enhancement would be significant because recent research has shown that the average duration of video consumption on mobile devices is 3.2 minutes [31].

Data collected by publishers regarding their individual audiences may impact the way the Vmoox service interacts with a publisher's content (e.g. CNN viewers might have a lower or higher average view time, calling for an adjustment to chapter lengths). As such, we believe that customizability is essential to Vmoox's potential for success.

Finally, Vmoox's customizability is expressed in its simple, loosely coupled modular design that ensures enhancement and scalability. For example, a multimedia analytics module could be easily added that tracks mobile users behavior as they access this streaming content. In addition, all Vmoox's components could themselves be farmed out to Cloud services. For example, Vmoox could use Encoding.com's transcoding cloud as it back-end because Vmoox's design encapsulates all its processes within it class design. As such, the only data that other processes see are data that are communicated through its well defined interfaces.

References

[1] S. Radwanick, "The 2008 digital year in review," comScore Whitepaper, January 30, 2009.

[2] Nielsen, Inc., "Television, internet and mobile usage in the U.S," A2/M2 Three Screen Report, 1st Quarter, 2009.

[3] Nielsen, Inc., "An overview of home internet access in the U.S., December, 2008. http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/03/overview-of-home-internet-access-in-the-us-jan-6.pdf, accessed May 3, 2010.

[4] S. Jespers, "Flash video market share continues to grow," February 5, 2009. http://www.webkitchen.be/2009/02/05/flash-videomarketshare-continues-to-grow/, accessed May 3, 2010.

[5] R. Hansen, "Browser power consumption," SecTheory, December 1, 2008. http://www.sectheory.com/browser-power-consumption.htm, accessed May 3, 2010.

[6] Texas Instruments, "Adobe and Texas Instruments bring flash and AIR to OMAP™ Platform," 2009. http://focus.ti.com/pr/docs/preldetail.tsp?sectionId=594&prelId=sc09045, accessed May 3, 2010.

[7] M. Perez, "Flash coming to smartphones in October," InformationWeek, June 23, 2009. http://www.informationweek.com/news/personal_tech/ smartphones/showArticle.jhtml?articleID=218100917, accessed May 3, 2010.

[8] E. Schonfeld, "iPhone makes up 50 percent of smartphone web traffic in U.S., Android already 5 percent," TechCrunch, http://www.techcrunch.com/2009/03/24/iphone-now-50-percent-of-smartphone-web-traffic-in-the-us/, Mar 24, 2009. accessed May 3, 2010.

[9] MacWorld, "Adobe preps full flash player for smartphones," http://www.techcrunch.com/2009/03/24/iphone-now-50-percent-of-smartphone-web-traffic-in-the-us/, accessed May 3, 2010.

[10] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, 2003, pp. 560-576.

[11] Sorenson Media, "Sorenson360," http://www.sorensonmedia.com/video-delivery-network/, accessed May 3, 2010.

[12] Encoding.com. http://www.encoding.com/, accessed May 3, 2010.

[13] RipCode. "TransAct," http://www.ripcode.com/, accessed May 3, 2010.

[14] Panvidea, http://www.panvidea.com/, accessed May 3, 2010.

[15] HD Cloud, http://hdcloud.com/, accessed May 3, 2010.

[16] Ankoder, http://www.ankoder.com, accessed May 3, 2010.

[17] Hey!Watch, http://heywatch.com, accessed May 3, 2010.

[18] Z. Yetgin and G. Seckin, "Progressive download for multimedia broadcast multicast service," IEEE MultiMedia, vol. 16, no. 2 (Apr.-June), 2009, pp. 76-85.

[19] J. Zhou, Z. Ou, M. Rautiainen, T. Koskela, and M. Ylianttila, "Digital television for mobile devices," IEEE MultiMedia, vol. 16, no. 1 (Jan.-Mar), 2009, pp. 60-71.

[20] M. Budagavi, and M. Zhou, "Next generation video coding for mobile applications: industry requirements and technologies," Proc. SPIE Visual Communications and Image Processing (VCIP), San Jose, Jan. 2007.

[21] C-W Ku, C-C. Cheng,G-S. Yu, M-C. Tsai, and T-S. Chang, "A high-definition H.264/AVC intra-frame codec IP for digital video and still camera applications,", IEEE Transactions on Circuits and Systems for Video Technology, vol.16, no.8 (Aug.), 2006, pp.917-928.

[22] M. Budagavi and M. Zhou,"Video coding using compressed reference frames," IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2008 (March 31 -April 4), pp.1165-1168,

[23] iPhone SDK 3.0, "Developing for iPhone OS 3.0," http://developer.apple.com/iphone/, accessed May 3, 2010.

[24] Hibernate, "Relational persistence for Java and .NET," https://www.hibernate.org/., accessed May 3, 2010.

[25] MySQL, "The world's most popular open source database." http://www.mysql.com/, accessed May 3, 2010.

[26] M. Fitzgerald, “Serializing Java objects with Xstream,” August 18, 2004, http://www.xml.com/lpt/a/1462.

[27] D. Crockford, D. **“**Introducing JSON.” http://www.json.org/. Accessed May 3, 2010.

[28] Ffmpeg, http://ffmpeg.org/, accessed May 3, 2010.

[29] GWT, “Write AJAX apps in the Java language, then compile to optimized JavaScript,” .http://code.google.com/webtoolkit/overview.html, accessed May 3, 2010.

[30] S. Tilkov, “A brief introduction to REST,” InfoQ, Dec 10, 2007. http://www.infoq.com/articles/rest-introduction.

[31] comScore Press Release. “U.S. online video viewing surges 13 percent in record-setting december,” February 4, 2009. http://www.comscore.com/Press_Events/Press_Releases/2009/2/US_Online_Video_Viewing_Sets_Record.