CloudMoV: Cloud-based Mobile Social TV

Yu Wu *[†], Zhizhong Zhang[†], Chuan Wu[†], Zongpeng Li[‡], Francis C.M. Lau[†]
[†]Department of Computer Science, The University of Hong Kong, Email: {ywu,zzzhang,cwu,fcmlau}@cs.hku.hk
[‡]Department of Computer Science, University of Calgary, Canada, Email: zongpeng@ucalgary.ca

Abstract—**The rapidly increasing power of personal mobile devices (smartphones, tablets, etc.) is providing much richer contents and social interactions to users on the move. This trend however is throttled by the limited battery lifetime of mobile devices and unstable wireless connectivity, making the highest possible quality of service experienced by mobile users not feasible. The recent cloud computing technology, with its rich resources to compensate for the limitations of mobile devices and connections, can potentially provide an ideal platform to support the desired mobile services. Tough challenges arise on how to effectively exploit cloud resources to facilitate mobile services, especially those with stringent interaction delay requirements. In this paper, we propose the design of a Cloud-based, novel Mobile sOcial tV system (CloudMoV). The system effectively utilizes both PaaS (Platform-as-a-Service) and IaaS (Infrastructure-as-a-Service) cloud services to offer the living-room experience of video watching to a group of disparate mobile users who can interact socially while sharing the video. To guarantee good streaming quality as experienced by the mobile users with time-varying wireless connectivity, we employ a surrogate for each user in the IaaS cloud for video downloading and social exchanges on behalf of the user. The surrogate performs efficient stream transcoding that matches the current connectivity quality of the mobile user. Given the battery life as a key performance bottleneck, we advocate the use of burst transmission from the surrogates to the mobile users, and carefully decide the burst size which can lead to high energy efficiency and streaming quality. Social interactions among the users, in terms of spontaneous textual exchanges, are effectively achieved by efficient designs of data storage with BigTable and dynamic handling of large volumes of concurrent messages in a typical PaaS cloud. These various designs for flexible transcoding capabilities, battery efficiency of mobile devices and spontaneous social interactivity together provide an ideal platform for mobile social TV services. We have implemented CloudMoV on Amazon EC2 and Google App Engine and verified its superior performance based on real-world experiments.**

I. Introduction

Thanks to the revolutionary "reinventing the phone" campaigns initiated by Apple Inc. in 2007, smartphones nowadays are shipped with multiple microprocessor cores and gigabyte RAMs; they possess more computation power than personal computers of a few years ago. On the other hand, the wide deployment of 3G broadband cellular infrastructures further fuels the trend. Apart from common productivity tasks like emails and web surfing, smartphones are flexing their strengths in more challenging scenarios such as realtime video streaming and online gaming, as well as serving as a main tool for social exchanges.

Although many mobile social or media applications have emerged, truely killer ones gaining mass acceptance are still impeded by the limitations of the current mobile and wireless technologies, among which battery lifetime and unstable connection bandwidth are the most difficult ones. It is natural to resort to cloud computing, the newly-emerged computing paradigm for low-cost, agile, scalable resource supply, to support power-efficient mobile data communication. With virtually infinite hardware and software resources, the cloud can offload the computation and other tasks involved in a mobile application and may significantly reduce battery consumption at the mobile devices, if a proper design is in place. The big challenge in front of us is how to effectively exploit cloud services to facilitate mobile applications. There have been a few studies on designing mobile cloud computing systems [1][2][3], but none of them deal in particular with stringent delay requirements for spontaneous social interactivity among mobile users.

In this paper, we describe the design of a novel mobile social TV system, *CloudMoV*, which can effectively utilize the cloud computing paradigm to offer a living-room experience of video watching to disparate mobile users with spontaneous social interactions. In *CloudMoV*, mobile users can import a live or on-demand video to watch from any video streaming site, invite their friends to watch the video concurrently, and chat with their friends while enjoying the video. It therefore blends viewing experience and social awareness among friends on the go. As opposed to traditional TV watching, mobile social TV is well suited to today's life style, where family and friends may be separated geographically but hope to share a *co-viewing* experience. While social TV enabled by set-top boxes over the traditional TV systems is already available [4] [5], it remains a challenge to achieve mobile social TV, where the concurrently viewing experience with friends is enabled on mobile devices.

We design *CloudMoV* to seamlessly utilize agile resource support and rich functionalities offered by both an IaaS (Infrastructure-as-a-Service) cloud and a PaaS (Platform-as-a-Service) cloud. Our design achieves the following goals.

Encoding flexibility. Different mobile devices have differently sized displays, customized playback hardwares, and various codecs. Traditional solutions would adopt a few encoding formats ahead of the release of a video program. But even the most generous content providers would not be able to attend to all possible mobile platforms, if not only to the current hottest models. *CloudMoV* customizes the streams for different devices at real time, by offloading the transcoding tasks to an

The research was supported in part by a grant from RGC under the contract HKU 717812E, an open grant from Shenzhen Key Lab of Cloud Computing Tech. & App. (SZCCTA).

IaaS cloud. In particular, we novelly employ a *surrogate* for each user, which is a virtual machine (VM) in the IaaS cloud. The surrogate downloads the video on behalf of the user and transcodes it into the desired formats, while catering to the specific configurations of the mobile device as well as the current connectivity quality.

Battery efficiency. A breakdown analysis conducted by Carroll *et al.* [6] indicates that the network modules (both Wi-Fi and 3G) and the display contribute to a significant portion of the overall power consumption in a mobile device, dwarfing usages from other hardware modules including CPU, memory, etc. We target at energy saving coming from the network module of smartphones through an efficient data transmission mechanism design. We focus on 3G wireless networking as it is getting more widely used and challenging in our design than Wi-Fi based transmissions. Based on cellular network traces from real-world 3G carriers, we investigate the key 3G configuration parameters such as the power states and the inactivity timers, and design a novel burst transmission mechanism for streaming from the surrogates to the mobile devices. The burst transmission mechanism makes careful decisions on burst sizes and opportunistic transitions among high/low power consumption modes at the devices, in order to effectively increase the battery lifetime.

Spontaneous social interactivity. Multiple mechanisms are included in the design of *CloudMoV* to enable spontaneous social, co-viewing experience. *First*, efficient synchronization mechanisms are proposed to guarantee that friends joining in a video program may watch the same portion (if they choose to), and share immediate reactions and comments. Although synchronized playback is inherently a feature of traditional TV, the current Internet video services (*e.g.*, Web 2.0 TV) rarely offer such a service. *Second*, efficient message communication mechanisms are designed for social interactions among friends, and different types of messages are prioritized in their retrieval frequencies to avoid unnecessary interruptions of the viewing progress. For example, online friend lists can be retrieved at longer intervals at each user, while invitation and chat messages should be delivered more timely. We adopt textual chat messages rather than voice in our current design, believing that text chats are less distractive to viewers and easier to read/write and manage by any user.

These mechanisms are seamlessly integrated with functionalities provided by a typical PaaS cloud, via an efficient design of data storage with BigTable and dynamic handling of large volumes of concurrent messages. We exploit a PaaS cloud for social interaction support due to its provision of robust underlying platforms (other than simply hardware resources provided by an IaaS cloud), with transparent, automatic scaling of users' applications onto the cloud.

Portability. A prototype *CloudMov* system is implemented following the philosophy of "Write Once, Run Anywhere" (WORA): both the front-end mobile modules and the back-end server modules are implemented in "100% Pure Java" [7], with well-designed generic data models suitable for any BigTable-like data store; the only exception is the transcoding module, which is implemented using ANSI C for performance reasons and uses no platform-dependent or proprietary APIs. The client module can run on any mobile devices supporting HTML5, including Android phones, iOS systems, etc. To showcase its performance, we deploy the system on Amazon EC2 and Google App Engine, and conduct thorough tests on iOS platforms. Our prototype can be readily migrated to various cloud and mobile platforms with little effort.

The remainder of this paper is organized as follows. In Sec. II, we compare our work with the existing literature and highlight our novelties. In Sec. III, we present the architecture of *CloudMoV* and the design of individual modules. A real-world prototype implementation follows and is described in Sec. IV, We discuss experimental evaluations in Sec. V. Finally, we conclude the paper in Sec. VI.

II. Related Work

A number of mobile TV systems have sprung up in recent years, driven by both hardware and software advances in mobile devices. Some early systems [8][9] bring the "living-room" experience to small screens on the move. But they focus more on barrier clearance in order to realize the convergence of the television network and the mobile network, than exploring the demand of "social" interactions among mobile users. There is another trend in which efforts are dedicated to extending social elements to television systems [4] [5][10]. Coppens *et al.* [4] try to add rich social interactions to TV but their design is limited to traditional broadcast program channels. Oehllberg *et al.* [5] conduct a series of experiments on human social activities while watching different kinds of programs. Though inspiring, these designs are not that suitable for being applied directly in a mobile environment. Chuah *et al.* [11] extend the social experiences of viewing traditional broadcast programs to mobile devices, but have yet to deliver a well integrated framework. Schatz *et al.* [12][13] have designed a mobile social TV system, which is customized for DVB-H networks and Symbian devices as opposed to a wider audience. Compared to these prior work and systems, we target at a design for a generic, portable mobile social TV framework, featuring *co-viewing* experiences among friends over geographical separations through mobile devices. Our framework is open to all Internet-based video programs, either live or on-demand, and supports a wide range of devices with HTML5 compatible browsers installed, without any other mandatory component on the devices.

For any application targeted at mobile devices, reducing power consumption is perennially one of the major concerns and challenges. Flinn *et al.* [14] exploit collaborations between the mobile OS and the mobile applications to balance the energy conservation and application performance. Yuan *et al.* [15] investigate mobile multimedia streaming, similar to most of the other work, by adjusting the CPU power for energy saving; however, according to the recent measurement work of Carroll *et al.* [6], the display and the wireless network card (including the cellular module) and not the CPU consume more than half of the overall power consumption in smart phones nowadays. Our work is able to achieve a significant (about 30%) power saving, by opportunistically switching the device between high-power and low-power transmission modes during streaming. Some existing work (*e.g.*, Anastasi *et al.* [16]) have provided valuable guidelines for energy saving over WiFi transmissions; our work focuses on 3G cellular transmissions which have significantly different power models; 3G is a more practical wireless connection technology for mobile TVs on the go at the present time.

Cloud computing had its debut with much fanfare and is now deemed a most powerful hosting platform in many areas including mobile computing. Satyanarayanan *et al.* [1] suggest offloading mobile devices' computation workload to a nearby resource-rich infrastructure (*i.e., Cloudlets*) by dynamic VM synthesis. Kosta *et al.* [2] propose a virtualization framework for mobile code offloading to the cloud. Zhang *et al.* [17] introduce an elastic mobile application model by offloading part of the applications (*weblets*) to an IaaS cloud. All these work target at computational job offloading. Recently, attentions have been drawn to enabling media applications using the cloud, for both media storage [18] and processing [19]. We are aware of a recent work by Huang *et al.* [3] which, in resemblance to ours, also leverages cloud resources for video transcoding. But they advocate scalable video coding (SVC) using multiple cluster nodes, which is not suitable in a mobile social TV scenario due to the encoding complexity of SVC (hence leading to intolerable delays), when realtime video retrievals and social interactions via mobile devices are desired. We instead advocate non-layered coding in such delay-sensitive mobile applications, although the detailed transcoding algorithm designs are out of the scope of this work. In addition, we novelly employ a surrogate for each mobile user in the cloud rather than relying on a dedicated cluster, which can be more easily implemented in practice. Liu *et al.* [20] build a mobile-based social interaction framework on top of the Google App Engine and offer an iOS implementation. We set out to design a portable, generic, and robust framework to enable realtime streaming and social interaction concurrently, which is not bound to any specific cloud platform. Although our prototype is implemented on only two public clouds, *i.e.*, Amazon EC2 and Google App Engine, it can be easily ported to other cloud systems as long as the targeted cloud platforms conform to the unified standard.

A recent work by Zhang *et al.* [21] investigates the media caching management problem under HTTP adaptive bit rate streaming over a wireless network environment, which can complement our work when video streams are required to be transcoded into multiple bit rates.

Finally, we are aware of the lack of a richly-featured cloud-based mobile social TV system in real life. The only system coming close to ours is Live Stream [22] on the iOS platform. This iOS-locked application only supports live video channels, and all its social functions are bound to Facebook open APIs. Conversely, the prototype we implement is browser-based and platform independent; it supports both live channels, VoD channels and even personal channels hosted by any user, with wider usage ranges and flexible extensibility. The framework we propose can be readily applied to other cloud-assisted mobile media applications as well.

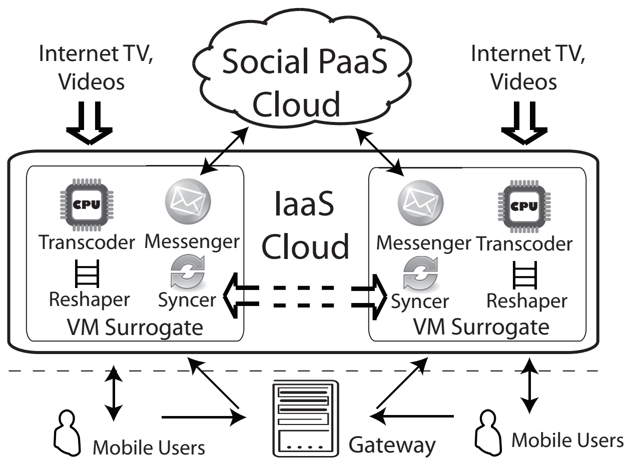


Fig. 1. The architecture of CloudMoV.

III. CloudMoV: Architecture and Design

As a novel Cloud-based Mobile sOcial tV system, *CloudMoV* provides two major functionalities to participating mobile users: (1) **Universal streaming.** A user can stream a live or on-demand video from any video sources he chooses, such as a TV program provider or an Internet video streaming site, with tailored encoding formats and rates for the device each time. (2) **Co-viewing with social exchanges.** A user can invite multiple friends to watch the same video, and exchange text messages while watching. The group of friends watching the same video is referred to as a *session*. The mobile user who initiates a session is the *host* of the session. We present the architecture of *CloudMoV* and the detailed designs of the different modules in the following.

A. Key Modules

Fig. 1 gives an overview of the architecture of *CloudMoV*. A *surrogate* (*i.e.*, a virtual machine (VM) instance), or a *VM surrogate* equivalently, is created for each online mobile user in an IaaS cloud infrastructure. The surrogate acts as a proxy between the mobile device and the video sources, providing transcoding services as well as segmenting the streaming traffic for burst transmission to the user. Besides, they are also responsible for handling frequently exchanged social messages among their corresponding users in a timely and efficient manner, shielding mobile devices from unnecessary traffic and enabling battery efficient, spontaneous social interactions. The surrogates exchange social messages via a back-end PaaS cloud, which adds scalability and robustness to the system. There is a gateway server in *CloudMoV* that keeps track of participating users and their VM surrogates, which can be implemented by a standalone server or VMs in the IaaS cloud.

The design of *CloudMoV* can be divided into the following major functional modules.

▷ **Transcoder.** It resides in each surrogate, and is responsible for dynamically deciding how to encode the video stream from the video source in the appropriate format, dimension, and bit rate. Before delivery to the user, the video stream is further encapsulated into a proper transport stream. In our implementation, each video is exported as MPEG-2 transport streams, which is the de facto standard nowadays to deliver digital video and audio streams over lossy medium.

▷ **Reshaper.** The reshaper in each surrogate receives the encoded transport stream from the transcoder, chops it into segments, and then sends each segment in a burst to the mobile device upon its request (*i.e.*, a burst transmission mechanism), to achieve the best power efficiency of the device. The burst size, *i.e.*, the amount of data in each burst, is carefully decided according to the 3G technologies implemented by the corresponding carrier.

▷ **Social Cloud.** The social cloud is built on top of any general PaaS cloud services with BigTable-like data store to yield better economies of scale without being locked down to any specific proprietary platforms. Despite its implementation on Google App Engine (GAE) as a proof of concept, our prototype can be readily ported to other platforms. It stores all the social data in the system, including the online statuses of all users, records of the existing sessions, and messages (invitations and chat histories) in each session. The social data are categorized into different *kinds* and split into different *entities* (in analogy to *tables* and *rows* in traditional relational database, respectively) [23]. The social cloud is queried from time to time by the VM surrogates.

▷ **Messenger.** It is the client side of the social cloud, residing in each surrogate in the IaaS cloud. The Messenger periodically queries the social cloud for the social data on behalf of the mobile user and pre-processes the data into a light-weighted format (plain text files), at a much lower frequency. The plain text files (in XML formats) are asynchronously delivered from the surrogate to the user in a traffic-friendly manner, *i.e.*, little traffic is incurred. In the reverse direction, the messenger disseminates this user's messages (invitations and chat messages) to other users via the data store of the social cloud.

▷ **Syncer.** The syncer on a surrogate guarantees that viewing progress of this user is within a time window of other users in the same session (if the user chooses to synchronize with others). To achieve this, the syncer periodically retrieves the current playback progress of the session host and instructs its mobile user to adjust its playback position. In this way, friends can enjoy the "sitting together" viewing experience. Different from the design of communication among messagers, syncers on different VM surrogates communicate directly with each other as only limited amounts of traffic are involved.

▷ **Mobile Client.** The mobile client is not required to install any specific client software in order to use *CloudMoV*, as long as it has an HTML5 compatible browser (*e.g.*, Mobile Safari, Chrome, etc.) and supports the HTTP Live Streaming protocol [24]. Both are widely supported on most state-of-the-art smartphones.

▷ **Gateway.** The gateway provides authentication services for users to log in to the *CloudMoV* system, and stores users' credentials in a permanent table of a MySQL database it has installed. It also stores information of the pool of currently available VMs in the IaaS cloud in another in-memory table. After a user successfully logs in to the system, a VM surrogate will be assigned from the pool to the user. The in-memory table is used to guarantee small query latencies, since the VM pool is updated frequently as the gateway reserves and destroys VM instances according to the current workload. In addition, the gateway also stores each user's friend list in a plain text file (in XML formats), which is immediately uploaded to the surrogate after it is assigned to the user.

We describe the key designs in *CloudMov* in the following.

B. Loosely Coupled Interfaces

Similar in spirit to web services, the interfaces between different modules in *CloudMov*, *i.e.*, mobile users, VM surrogates, and the social cloud, are based on HTTP, a universal standard for all Internet-connected devices or platforms. Thanks to the loose coupling between users and the infrastructure, almost any mobile device is ready to gain access to the *CloudMoV* services, as long as it is installed with an HTTP browser. The VM surrogates provisioned in the IaaS cloud cooperate with the social cloud implemented on a PaaS cloud service via HTTP as well, with no knowledge of the inner components and underlying technologies of each other, which contributes significantly to the portability and easy maintenance of the system.

For social message exchanges among friends, *CloudMoV* employs asynchronous communication. All the exchanged messages are routed via the surrogates to the social cloud, which efficiently organizes and stores the large volumes of data in a BigTable-like data store. The VM surrogates query the social cloud frequently and processes the retrieved data into XML files, for later retrieval by users in an asynchronous fashion. Such a design effectively separates the mobile users from the social cloud to significantly simplify the architecture, while the extra delay introduced at the VM surrogates is ignorable, as shown in Sec. V.

C. Pipelined Video Processing

Both live streaming of realtime contents and on-demand streaming of stored contents are supported in *CloudMoV*. Video processing in each surrogate is designed to work on the fly, *i.e.*, the transcoder conducts realtime encoding from the video source, the encoded video is fed immediately into the reshaper for segmentation and transmission, and a mobile user can start viewing the video as soon as the first segment is received. To support dynamic bit rate switch, the transcoder launches multiple threads to transcode the video into multiple bit rates once the connection speed between the surrogate and the mobile user changes. The IaaS cloud where the surrogates are deployed, represents an ideal platform for implementing such computation intensive jobs.

D. Burst Transmissions

1) 3G power states: Different from Wi-Fi which is more similar to the LANed Internet access, 3G cellular services suffer from the limited radio resources, and therefore each user equipment (UE) needs to be regulated by a Radio Resource Control (RRC) state machine [25]. Different 3G carriers may customize and deploy complex states in their respective cellular networks. Different states indicate different levels of allocated radio resources, and hence different levels of energy consumptions. For ease of implementation, we consider three basic states in our design, which are commonly employed by many carriers, namely CELL_DCH (a dedicated physical channel is allocated to the UE in both the uplink and the downlink), CELL_FACH (no dedicated channel is allocated but the UE is assigned a default common transport channel in the uplink), and IDLE, in decreasing order of power levels [25]. Contrary to intuition, the energy consumption for data transmission depends largely on the state a UE is working in, but has little to do with the volume of data transmitted, *i.e.*, a UE may stay at a high-power state (CELL_DCH) for data transmission even the data rate is very low [25] (this has also been verified in our experiments in Sec. V).

A 3G carrier may commonly transfer a UE from a high-power state to a low-power state (state demotion), for releasing radio channels allocated to this UE to other users. For example, if a UE working at a high-power state does not incur any data traffic for a pre-configured period of time (measured by a critical inactivity timer), the state of the UE will be transferred to a low-power one; when the volume of data traffic rises, the UE "wakes up" from a low-power state and moves to a high-power one. Timeouts of the critical inactivity timers for state transitions are properly set by the carrier to guarantee performance in both delay and energy consumption, since extra delay and energy consumption are potentially incurred for acquiring new radio channels when the UE transits from a low-power state to a high-power one later (state promotion).

2) Transmission mechanism: In *CloudMoV*, we aim at maximum conservation of the battery capacity of the mobile device, and design a burst transmission mechanism for streaming between the surrogate and the device. Using the HTTP live streaming protocol [24], the mobile device sends out requests for the next segment of the video stream from time to time. The surrogate divides the video into segments, and sends each segment in a burst transmission to the mobile device, upon such a request. When the mobile device is receiving a segment, it operates in the high-power state (CELL_DCH); when there is nothing to receive, it transfers to the low-power state (IDLE) via the intermediate state (CELL_FACH), and remains there until the next burst (segment) arrives.

3) Burst size: To decide the burst size, *i.e.*, the size of the segment transmitted in one burst, we need to take into consideration characteristics of mobile streaming and energy consumption during state transitions. For video streaming using a fixed device without power concerns, it is desirable to download as much of a video as what the connection bandwidth allows; however, for streaming over a cellular network, we should avoid downloading more than what is being watched for one main reason: users may switch among channels from time to time and those prefetched contents are probably never watched, leading to a waste of the battery power and the cellular data fee due to their download. Hence, the bursty size should be kept small, to minimize battery

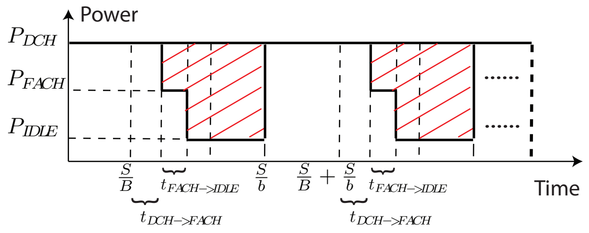


Fig. 2. Power consumption over time.

consumption and traffic charges. On the other hand, state transitions introduce latency and energy overheads, so the burst should be large enough to avoid frequent state transitions; otherwise, such overheads may diminish the energy saving achieved by an intelligent state transition mechanism. We next derive a lower bound on the burst size, which guarantees positive energy saving by such intelligent state transition.

Let B be the average available bandwidth over a wireless connection, S be the burst size in the CELL_DCH state, and b be the video playback rate at the mobile user. P_{DCH}, P_{FACH}, and P_{IDLE} denote the power levels at states CELL_DCH, CELL_FACH, and IDLE, respectively. $t_{DCH\to FACH}$ is the timeout of the critical inactivity timer (the transition time) for state transition from CELL_DCH to CELL_FACH, and $t_{FACH\to IDLE}$ is the timeout of the inactivity timer for state transition from CELL_FACH to IDLE. Let $P_{IDLE\to FACH}$, $P_{FACH\to DCH}$, and $P_{IDLE\to DCH}$ be the energy needed for state promotion denoted by the respective subscript. We ignore the delay overhead in the state promotion from a low-power state to a high-power state, since its value is small—less than one second—based on our real-life measurements as reported in Sec. V. We consider two cases: (1) transmission of a video according to our burst transmission mechanism, with $P_{burst}(t)$ being the power level at time t during the transmission; (2) continuous transmission of the video stream whenever there are transcoded contents ready, with $P_{cont}(t)$ being the power level at time t during the transmission. An illustration of power consumption in both cases is given in Fig. 2. The burst transmission operates at state CELL_DCH to send a total amount of data S for a duration of $\frac{S}{B}$; then it transits to state IDLE via state CELL_FACH, and remains there for duration $\frac{S}{b}-\frac{S}{B}-t_{DCH\to FACH}-t_{FACH\to IDLE}$ ($\frac{S}{b}$ is the time taken for the mobile user to play the segment of size S). Note that the power consumption level during transition periods $t_{DCH\to FACH}$ and $t_{FACH\to IDLE}$, remains at P_{DCH} and P_{FACH}, respectively, although no data is transmitted then. The continuous transmission always operates at the high-power state CELL_DCH with power level $P_{cont}(t) = P_{DCH}$. We calculate the overall energy saving (ΔE) by burst transmission of the video over the time span T (multiples of $\frac{S}{b}$), as compared to the continuous transmission, as follows:

$$\begin{aligned}
&\Delta E = \int_0^T (P_{cont}(t) - P_{burst}(t))dt \\
&= \int_0^{\frac{S}{b}} (P_{cont}(t) - P_{burst}(t))dt \times \frac{T}{S/b} \\
&= \frac{T\times b}{S} \int_0^{\frac{S}{b}-\frac{S}{B}} (P_{cont}(t) - P_{burst}(t))dt \\
&= \frac{T\times b}{S} \times ((P_{DCH} - P_{FACH}) \times t_{FACH\to IDLE} \\
&+ (P_{DCH} - P_{IDLE}) \times (\frac{S}{b} - \frac{S}{B} - t_{FACH\to IDLE} \\
&- t_{DCH\to FACH}) - P_{IDLE\to FACH} - P_{FACH\to DCH}).
\end{aligned} \quad (1)$$

The burst size S should be chosen such that positive energy saving, $\Delta E > 0$, can be achieved. A lower bound of the burst size can be decided using $\Delta E > 0$. We also see that the larger S is, the more energy saving we can achieve using burst transmission. However, with a large S, a user has to wait for a long time before the first segment is transcoded and delivered, and there is also the risk that it may download contents it will never watch. We evaluate the tradeoffs in selecting different values of S in our experiments in Sec. V.

IV. CloudMoV: Prototype Implementation

Following the design guidelines in Sec. III, we have implemented a real-world mobile social TV system, and deployed it on the Google App Engine (GAE) and Amazon EC2 clouds, which are the two most widely used public PaaS and IaaS cloud platforms.

GAE, as a PaaS cloud, provides rich services on top of Google's data centers and enables rapid deployment of Java-based and Python-based applications. *Data store*, a thin layer built on top of Google's famous BigTable [26], handles "big" data queries well with linear and modular scalability even for high-throughput usage scenarios. Hence, GAE is an ideal platform for implementing our social cloud, which dynamically handles large volumes of messages. On the other hand, GAE imposes many constraints on application deployment, *e.g.*, lack of support for multi-threading, file storage, etc., which may hinder both computation-intensive jobs and content distribution applications.

Amazon EC2 [27] is a representative IaaS cloud, offering raw hardware resources including CPU, storage, and networks to users. Most EC2 VM instances are launched with Linux kernels, and are Xen-para-virtualized as domU guests on top of dom0, which run directly on the bare-metal hardware upon booting. As the leading virtualization technology in the Linux community together with KVM [28], Xen supports para-virtualization on almost all hardware with Linux drivers, and hence gives close-to-native performance, especially for CPU virtualization and I/O virtualization, as has been verified by extensive measurements including ours. Comparing to a common PaaS cloud, EC2 is an appropriate platform for computation-intensive tasks in *CloudMoV*, *i.e.*, those the surrogates carry out.

We will show that a hybrid of the IaaS cloud, working as the computing unit, and the PaaS cloud, as the back-end NoSQL data store, serves as a perfect substrate in *CloudMoV*.

A. *Client Use of* CloudMov

All mobile devices installed with HTML5 compatible browsers can use *CloudMoV* services, as long as the HTTP Live Streaming (HLS) [24] protocol is supported. The user first connects to the login page of *CloudMoV*, as illustrated in the top left corner of Fig. 3. After the user successfully logs in through the gateway, he is assigned a VM surrogate from the VM pool (the hostnames of available VMs, *e.g.*, ec2-50-16-xx-xx.compute-1.amazonaws.com, are maintained in an in-memory table of a MySQL database deployed in the gateway). Then the user is automatically redirected to the assigned VM

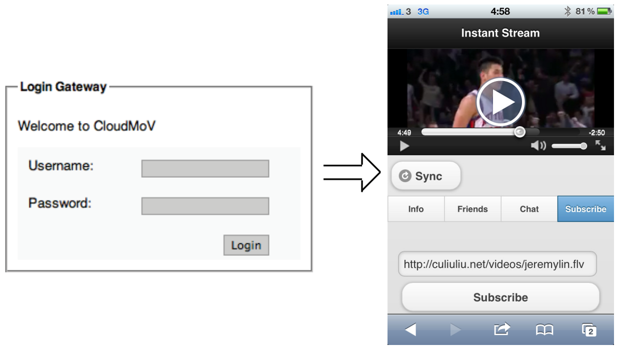


Fig. 3. Client UI of *CloudMoV*.

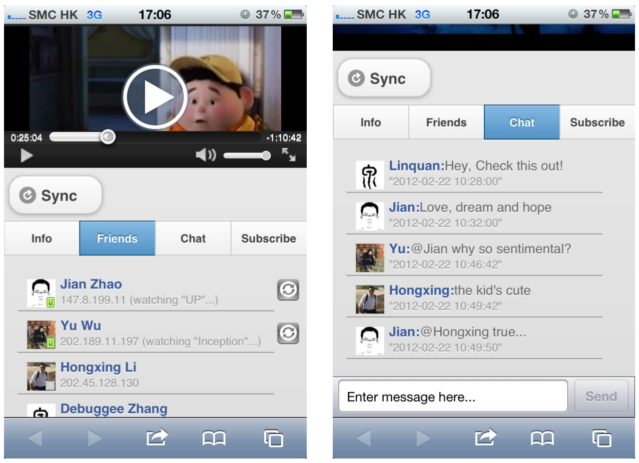
surrogate, and welcomed by a portal page as shown on the right-hand side of Fig. 3. Upon user login, the portal collects the device configuration information by examining the "User-Agent" header values, and this information will be sent to its surrogate for decision making of the video encoding formats. The user can enter the URL of the video or live broadcast he wishes to watch, on the "Subscribe" tab of the portal; after he clicks the "Subscribe" button, the address of the video is sent to the VM surrogate, which downloads the stream on the user's behalf, transcodes and sends properly encoded segments to the user. From the surrogate to the mobile device, the video stream delivered using HLS is always divided into multiple segments, with a playlist file (.m3u8) giving the indices. When the mobile client subscribes to a video, the playlist is first downloaded and individual segments are requested by the client in the following. A playlist file may become outdated if new contents are generated, *e.g.*, in case of a live broadcast. In that case, the mobile client needs to download the playlist again to keep the indices updated. The client starts to play the video as soon as the first segment is received.

When watching a video, the user can check out his friends' status (online or offline, which video they are currently watching) on the "Friends" tab (a snapshot is given in Fig. 4(a)), and invite one or more friends to join him in watching the video. When a user receives an invitation from a friend (profile pictures of those friends who have sent invitations will be highlighted on the "Friends" tab) and decides to join the session, he can choose to watch from the start, or catch up with the viewing progresses of others by clicking the "Sync" button, which triggers the Syncer functionality in the surrogate. Users in the same session can exchange opinions and comments on the "Chat" tab (a snapshot is given in Fig. 4(b)), where new chat messages can be entered and the chat history of the session is shown. The "Info" tab shows an abstract of the video, as edited by the session host.

All the data (.xml files) updates are delivered in an asynchronous manner based on AJAX techniques without the need of reloading the portal page, as has been introduced in Sec. III.

B. *VM Surrogates*

All the VM surrogates are provisioned from Amazon EC2 web services and tracked by the gateway. We create our own AMI (ami-b6f220df) based on Linux kernel 2.6.35.14, the default image Amazon provides [27]. Due to the intensive



Jian Zhao
147.8.199.11 (watching "UP"...)
Yu Wu
202.189.11.197 (watching "Inception"...)
Hongxing Li
202.45.128.130
Debuggee Zhang

Linquan:Hey, Check this out!
"2012-02-22 10:28:00"
Jian:Love, dream and hope
"2012-02-22 10:32:00"
Yu:@Jian why so sentimental?
"2012-02-22 10:46:42"
Hongxing:the kid's cute
"2012-02-22 10:49:42"
Jian:@Hongxing true...
"2012-02-22 10:49:50"

(a) "Friend tab" (b) "Chat tab"

Fig. 4. "Friend" and "Chat" tabs.

computation involved, we propose to implement all the video processing related tasks using ANSI C, to guarantee the performance. In particular, we install FFmpeg together with libavcodec as the groundsill library [29] to develop the transcoding, segmentation and reshaping modules on the VM surrogates. We have also installed a Tomcat web server (version 6.5) to serve as a Servlet container and a file server on each surrogate. Both FFmpeg and Tomcat are open source projects. Once a VM surrogate receives a video subscription request from the user, it downloads the video from the source URL, and processes the video stream by transcoding and segmentation, based on the collected device configurations by the portal. For example, in our experiments, the downloaded stream is transcoded into a high-quality stream and a low-quality stream in real time with H264/AAC codecs. The high-quality stream has a "480x272" resolution with 24 frames per second, while the low-quality one has a "240x136" resolution with 10 frames per second. A mobile user dynamically requests segments of these two different video streams, according to his current network connection speed. The transcoded stream is further exported to an MPEG-2 transporting stream (.ts), which is segmented for burst transmission to the user. The burst sizes depend on both the network bandwidth and video bit rate. We evaluate the impact of different burst sizes on the streaming quality and energy consumption in details in Sec. V. Fig. 5 shows the streaming architecture in our customized VM image. Here, the modules on social message exchanges are omitted, which will be presented in Fig. 6.

C. Data Models in the Social Cloud

We use GAE mainly as the back-end data store to keep the transient states and data of *CloudMoV*, including users' online presence status, social messages (invitation and chat messages) in all the sessions. With Jetty as the underlying Servlet container, most Java-based applications can be easily migrated to GAE, under limited usage constraints, where no platform-specific APIs are enforced for the deployment. GAE provides both its Java Persistence API (JPA 1.0, part of JSR 220) adapter and a set of proprietary low-level APIs to map the relational data. We choose to use the former only in *CloudMoV*

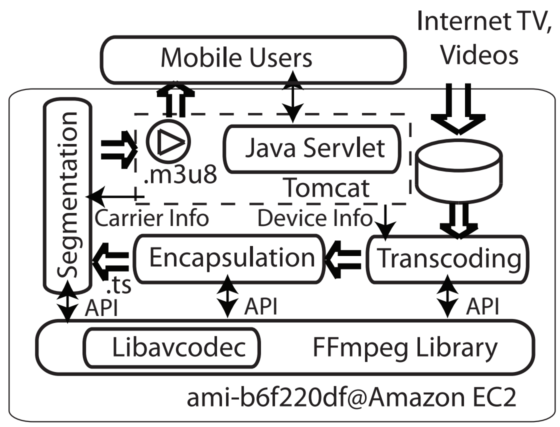


Fig. 5. Streaming architecture in each customized VM image (ami-b6f220df).

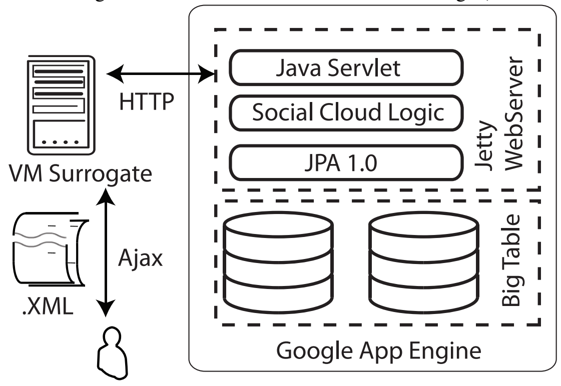


Fig. 6. Social message exchanges via Google App Engine.

such that *CloudMoV* can be easily migrated to other PaaS clouds as well.

Once a user logs in to the system and enters the URL of a video to watch, a session ID is generated for the new session (corresponding to viewing of this video), by combining the user's "username" in the system with the time stamp when the session is created. The gateway delivers an HTTP request to a Servlet listener running on GAE, to notify it that an entry for the newly joined user should be added, with the user's "username" as the key and other information (URL of the subscribed video, the session ID, etc.) as the value. This entry will then be periodically retrieved through a public Servlet interface by surrogates representing the user's friends, in order to learn the updated status of the user over time. The default interval for retrieving updates of friends' online status is five minutes. When the user goes offline, the user online status record will be deleted. .

Whenever a user decides to join a session hosted by his friend upon invitation, his VM surrogate switches to download the video of the session, and at the same time sends an HTTP request to the social cloud, for updating the session ID in this user's entry to the new one. If the user wishes to synchronize his playback progress with that of the session host, his VM surrogate synchronizes with the session host to maintain the playback "currenttime" value (HTML5 property).

The social cloud maintains a "Logs" entry for each existing session in *CloudMoV*, with the session ID as the primary key and an array list as the value, which corresponds to individual messages in this session. When a user in a session posts a comment, this message is first sent to his VM surrogate, which further injects the message into the social cloud via another Servlet listener. The message is stored as a "Message" entry in the social cloud, with the message content as the value, and an auto-generated integer as the key. Entries "Logs" and "Message" are annotated by a *@OneToMany* relationship, to facilitate the data management. VM surrogates of users in the same session send periodical HTTP query requests to the social cloud for the latest comments from others. The default interval for retrieval of new comments is 10 seconds. The retrieved messages are stored and updated on the surrogates, which process them into well-formed XML formats for efficient parsing at the user devices. The user devices retrieve the XML files from the surrogates at a lower frequency (with default interval 1 minute), in order to minimize the power consumption and the traffic. Fig. 6 presents social message exchanges among a mobile user, his VM surrogate, and the GAE.

A large number of entries in the social cloud becomes outdated very soon, since users may switch from one session to another, quit the system, and so on. We launch a cron job behind the scene every 10 minutes to clear those outdated entries. For example, for sessions of which everybody has left, their "Logs" entries and all the associated "Message" entries are deleted in a single transaction.

V. Real-World Experiments

We carry out both unit tests and performance evaluations of *CloudMoV* deployed on Amazon EC2 and Google App Engine, using a number of iPhone 4S smart phones (iOS 5.01) as the mobile clients, which have been registered on the Apple developer site. The gateway is implemented on a Virtual Private Server (VPS) hosted by Bluehost [30]. Unless stated otherwise, the experiments are conducted over the 3G cellular network of 3HK [31], which is one of the largest Telecom operators in Hong Kong.

A. Measuring the RRC States

We first design measurement experiments to discover the timeout values of the critical inactivity timers employed in 3HK's 3G network, as discussed in Sec. III-D. We enable logging functions on an fully charged iPhone 4S and use the Mobile Safari (the HTML5-compatible browser on iPhone) to watch a YouTube video using *CloudMoV* services. The battery consumption traces on the phone are profiled by "Instruments", a powerful tool of Xcode [32]. The playback rate of the video on the phone is about 254 Kbps.

Fig. 7 shows the power consumption levels on the phone over time, in terms of portions of the highest device power level. The red vertical lines represent the starting points of playback periods when the Safari runs in the foreground, and the green lines represent the finish times of playback periods when the Safari is suspended in the background. We can see that our state transition model in Fig. 2 is verified by these real-world measurements: when there is data transmission, the device operates at the high power mode; when data transmission stops, the transmission power of the device first

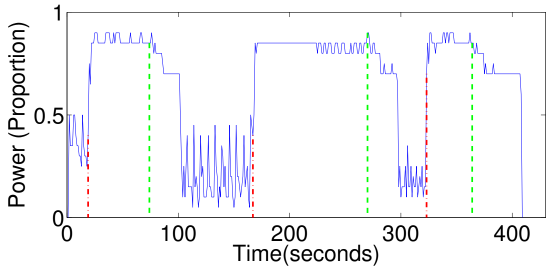


Fig. 7. Power consumption over time on an iPhone 4S

decreases to an intermediate level, and then to a very low level. We also find out that timeouts of the inactivity timers $t_{DCH \rightarrow FACH}$ and $t_{FACH \rightarrow IDLE}$ are approximately 14 and 16 seconds, respectively. It shows that the interval between the finish of one burst transmission to the start of the next burst transmission should be at least 30 seconds, to allow the phone to enter a low-power mode. Following these measurement results, in our following experiments conducted over 3HK 3G network, we set the default *burst transmission interval* to 60 seconds, the time from the start of one burst transmission to the start of the next burst transmission, corresponding to the playback time of one burst segment, $\frac{S}{b}$, where S is the burst size.

B. Impact of Burst Size on Power Consumption

The technique of video segmentation is widely employed in video streaming applications, but mostly for ease of distribution and not for battery efficiency at potential mobile users. Apple Inc., which proposed the HTTP Live Streaming protocol [24], suggests 10-second-playback segments, which has been followed in many streaming applications. We find this segment size is problematic and can drain the battery of a mobile device quickly. In Fig. 8, we compare the power consumption levels when burst transmission intervals of 10 seconds and 60 seconds are used, respectively, for the iPhone 4S to stream a 10-minute YouTube flash video (.flv). We note that, iOS devices can not play flash videos, but *CloudMoV* helps transcode the flash to the H264/AAC stream, which is compatible with our iPhone 4S.

We observe that the device remains at the high power mode (CELL_DCH) if the 10-second segmentation is used, since the state transition takes at least 30 seconds, as given in Sec. V-A. On the other hand, using 60-second burst transmission intervals, *CloudMoV* may transfer the device to the low power mode (IDLE) via the intermediate power mode (CELL_FACH) from time to time. In this way, *CloudMoV* can achieve approximately 29.1% power saving. We also observe some unexpected behavior of the power levels around 400 seconds, where the power does not drop. After checking Tomcat server logs on the VM surrogate, we find that the device requested the play list file (.m3u8) twice around that time, possibly due to packet loss, and the tiny traffic of the playlist (about 4 KB) deprived the device from a "sleep" chance.

To verify whether such playback list updates may always prohibit the device's power level from dropping, we have further conducted tests by creating a live broadcast stream

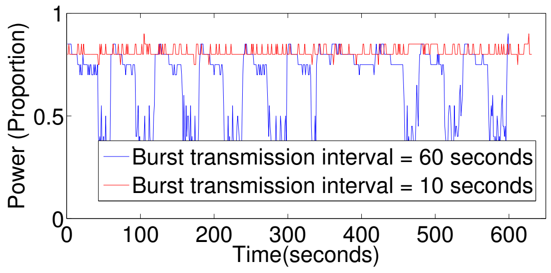


Fig. 8. Power consumption over time with different burst transmission sizes.

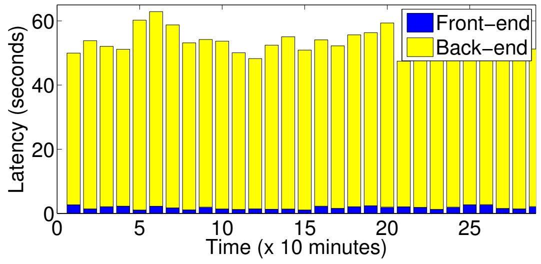


Fig. 9. Average user sign-in latencies over time.

from the same flash video and deploy it on *CloudMoV*. We find that regardless of the burst transmission intervals we set, the device's power level does not drop due to frequent play list update in a live streaming. We omit the plots of the results since they are similar to the red curve in Fig. 8. Different browsers may configure different update frequencies, since the value is not specified in the HTTP Live Streaming protocol. This value should be carefully set, for potentially increasing battery lifetime at the mobile users.

C. Sign-in Latency into the System

When a user signs into the *CloudMoV* system via the login gateway shown in Fig. 3 and gets identified, the gateway will request a virtual machine instance from the IaaS cloud to be the user's surrogate. The sign-in process finishes when the surrogate is initialized and the user is connected to the surrogate. In this experiment, five mobile users repeatedly join the system and log off as soon as the respective surrogate is initialized. We inject JavaScript snippets into the client of *CloudMov* on the mobile device to record the timestamps during the sign-in process. Fig. 9 shows the average sign-in latencies experienced by these clients during a 4.5-hour span. The "Front-end" latency consists of both the sign-in request/response and identification delays, while the "Back-end" latency is the surrogate VM provisioning delay from the IaaS cloud (Amazon EC2). We can see that most of the latencies are caused by the latter. The delay can be significantly reduced if a VM pool is maintained wherein idle surrogates are initialized before hand (based on estimated user numbers), ready for immediate allocation when new users sign in.

D. Startup Latency of Video Playback

We evaluate the transcoding performance on the surrogates in *CloudMoV*, first by measuring the playback startup latency on the surrogates, from the time when the video subscription request is received from the mobile user to the time when the first transcoded burst segment is generated. We deploy the VM surrogates (ami-b6f220df) on three types of instances provided by Amazon EC2, with the detailed configurations shown in Table. I. For fair comparison, all the instances are deployed in the zone "east-1-c", and they transcode the same flash video used in experiments of Sec. V-B. Fig. 10 shows the playback startup latencies when different VM instances are used as the surrogate for an iPhone 4S, and different burst transmission intervals are employed.

TABLE I
CONFIGURATIONS OF VM INSTANCES

Type	CPU	Memory	Zone
Micro	Up to 2 ECUs	613MB	east-1-c
Small	1 ECU	1.7 GB	east-1-c
Medium	5 ECUs (2 cores)	1.7 GB	east-1-c

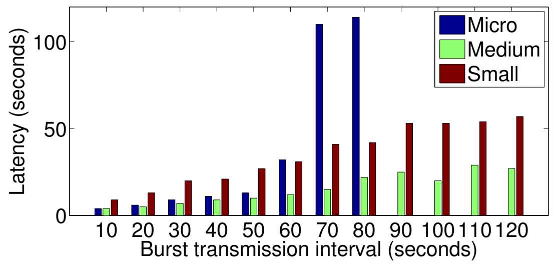


Fig. 10. Startup latency at different burst sizes.

In our experiments, we tested the network connection bandwidth between the Amazon EC2 instances and the YouTube website, and found that video downloading from YouTube website to the instances is very fast. Therefore, the startup latency depends mainly on the burst interval setting and the transcoding speed at the VM surrogate. Fig. 10 shows that in general, the longer the burst interval is, the larger the segment of video to transcode is, and thus the longer the startup latency is. We can see the medium instance achieves better transcoding performance with larger computation capacity, as compared to the small instance. The latency with the micro instance is unexpectedly large when the burst interval is longer than 60 seconds, and as such we need not collect the latencies for even longer burst intervals. However, the micro instance performs even better than the small instance with smaller burst intervals due to more CPU power (Amazon claims the micro instance has "UP to 2 ECUs".) The reason lies in memory thrashing on the micro instance (it has a smaller memory than other instances), when burst transmission intervals are longer than 60 seconds, when memory becomes the bottleneck. In case of the medium instance, we also find that the startup latency with 100-second burst intervals is smaller than that with 90-second bursts. We believe that it is caused by the overheads of load balancing between its two cores. This shows that the performance can be improved by a more efficient transcoding algorithm targeting at multi-core platforms, which will be part of our future work.

E. Dynamic Bit Rate Switch

We next evaluate whether *CloudMoV* can effectively transcode a video stream to different bit rates when the

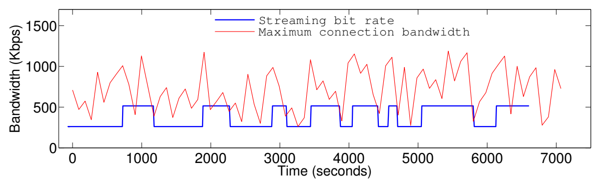


Fig. 11. Streaming rate with variation of the connection bandwidth.

connection bandwidth changes. We use a 1-hour-36-minute long movie produced by Pixar, in the original bit rate of 1017 Kbps and .avi format with the XviD codec. The movie is stored on an Apache web server isolated from the *CloudMoV* system. The format of the movie can not be directly played on an iPhone. *CloudMoV* dynamically transcodes this movie into two H264/AAC streams of different bit rates: a high-quality stream with a bit rate up to 515 Kbps and a low-quality stream at 261 Kbps. When the phone's wireless connection bandwidth is lower than 900 Kbps, *CloudMoV* directs the low-quality stream to it; otherwise, it transmits the high-quality stream.

To make the experiment reproducible and controllable, we test it over WiFi by connecting the phone to a TP-LINK WR741ND wireless router, instead of using a 3G connection (given that the cellular signal strength is hard to control). We believe this is a reasonable choice, for unit testing of the dynamic transcoding functionality only. By installing DD-WRT (v24-sp2 version 18007) on the router, we can have full control over the bandwidth limit through "tc" scripts. In our experiment, the maximum bandwidth (B_{max}) allowed at the wireless interface is updated every 90 seconds by setting $B_{max} = 300 + rand() \times 900$ (Kbps) (B_{max} thus ranges from 300 Kbps to 1200 Kbps). Tomcat server logs can capture each request for the video segments, which tells whether the high-quality or the low-quality stream is fetched. By synchronizing the timestamps of both the bandwidth variation sequence controlled by our bash script and the bit rate switch activity captured by the Tomcat logs, we plot in Fig. 11 the instantaneous bandwidths at the mobile phone over time. The red curve represents variation of the maximal bandwidth B_{max} and the blue one represents the streaming bit rate at the user. We can see that the streaming rates are effectively adjusted to adapt to the current connection bandwidth levels.

F. Jitters

By "Jitters", we mean the discontinuous video playback experienced by mobile users who have to wait for segments to be buffered, due to the dynamically varying download bandwidths. Following the same experiment settings as in Sec. V-E, we emulate a highly unstable 3G cellular network and measure the occurrence and stall duration of jitters when a mobile client is viewing the movie. We examine the download completion time for each segment: if this time is later than the playback deadline of the segment, a jitter is captured and the stall duration is estimated as the difference between the two. Fig. 12 compares the results of *CloudMoV* and the case where the movie is directly streamed to the mobile user without dynamic transcoding nor burst transmission mechanisms, *i.e.*, the case of "Normal Streaming". A line segment is plotted when a jitter happens at the corresponding time in the x axis, and the length of the line segment represents the stall duration. We can see that *CloudMoV* incurs no jitter while "Normal Streaming" suffers a total of 716 seconds of playback stall. This again verifies the excellent streaming playback performance of CloudMov clients.

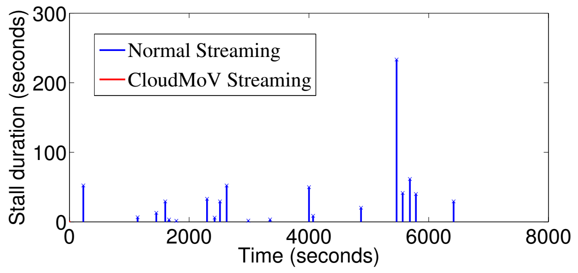


Fig. 12. Jitters and the stall durations.

G. Social Interaction Latencies

The service latency of Google App Engine is critical to the overall performance of *CloudMoV*. In this set of experiments, we launch a VM surrogate in each of four different regions (corresponding to four mobile users), *i.e.*, "east-1-a", "east-1-b", "east-1-c" and "east-1-d", all of which join the same session. Each surrogate keeps posting a short chat message every second and retrieves its own message immediately. We evaluate two critical latencies: one is the post latency to the GAE, *i.e.*, the time from when a message is sent out from a surrogate to the time when it receives confirmation from GAE that the message is successfully recorded in the social cloud; the other is the query latency, *i.e.*, the time from when a query is sent out from a surrogate to the time when the queried message is received at the surrogate.

Fig. 13 and Fig. 14 show the average values of the two types of latencies among surrogates in all regions, during a 155-second run of the experiments. Our results are mostly consistent with the 978-ms post latency and 106-ms query latency, given as the processing delays in the dashboard of GAE [33] (our latencies additionally include a round-trip time between a surrogate and the GAE). The query latency is relatively stable over time, while the post latency becomes larger after 70 seconds of the run. We reckon the reason to be either due to limitations imposed by Google on our free GAE account, or based on a side effect of the automatic scaling in GAE: given the large volumes of requests, *i.e.*, more than $16,000$ requests per minute (we used up our free GAE quota of 0.05-million requests within three minutes), GAE may well have distributed the newly posted messages to different geo-distributed data centers of Google. Confirming the detailed reason is part of our future work.

For further demonstration, we have also injected JavaScript snippets into the *CloudMov* client to capture the textual latencies, from the time a message is successfully posted by a client (stored into GAE) to the time the other clients receive the message. The results are given in Fig. 15. We can see all latencies that we have measured are short enough for realtime *CloudMoV* services.

H. Scalability

To evaluate the scalability of *CloudMoV*, we investigate the workload at the host of a session. As compared to a regular

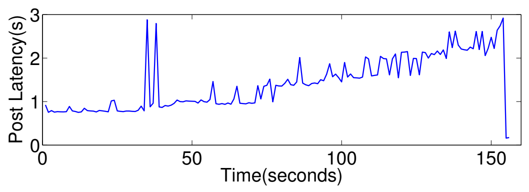


Fig. 13. Post latency to GAE.

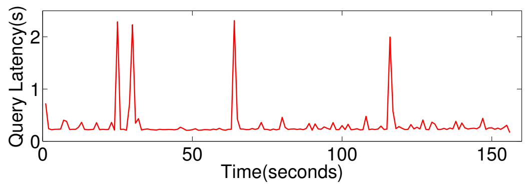


Fig. 14. Query latency from GAE.

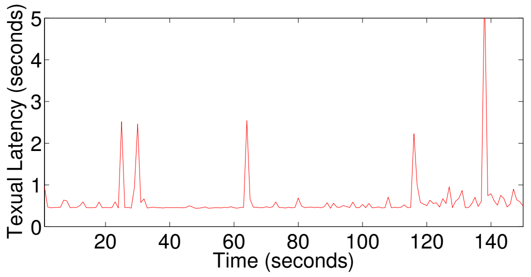
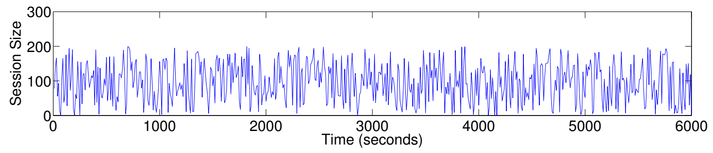


Fig. 15. Message retrieval latency at mobile clients.

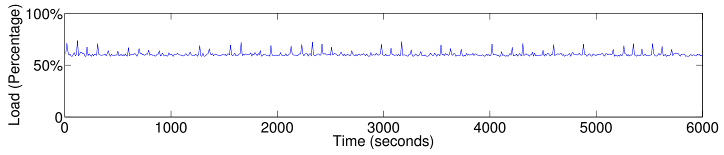
participant in a session, the surrogate of a session host is additionally responsible for maintaining the session group and carrying out synchronization for "co-viewing" experiences, besides its own transcoding tasks, which may potentially become a performance bottleneck in the system when the number of participants in the session is large. In this experiment, 200 users dynamically join and leave a session over a 100-minute interval, with dynamical session size given in Fig. 16(a). We also apply an extremely aggressive synchronization interval, *i.e.*, every participant synchronizes with the session host in every second, and assign a low-end "Micro"-type VM instance as the surrogate of the session host. Fig. 16(b) illustrates the workload on the surrogate of the session host, calculated as the percentage of busy time of the VM in each single second, when it is either handling the transcoding or the synchronization tasks. The surrogate of a session host is idling when it has finished these tasks. We can see that even under such an extreme setting, the surrogate of the session host can still finish all the computation and communication tasks within $70-80\%$ of its time, which again verifies the excellent scalability of our system, thanks mainly to the fully distributed design of surrogates.

VI. Concluding Remarks and Future Work

This paper presents our view of what might become a trend for mobile TV, *i.e.*, mobile social TV based on agile resource supports and rich functionalities of cloud computing services. We introduce a generic and portable mobile social TV framework, *CloudMoV*, that makes use of both an IaaS cloud and a PaaS cloud. The framework provides efficient transcoding services for most platforms under various network



(a) Session size over time



(b) Workload of the session host over time

Fig. 16. Session size and the workload on the surrogate of the session host over time.

conditions and supports for co-viewing experiences through timely chat exchanges among the viewing users. By employing one surrogate VM for each mobile user, we achieve ultimate scalability of the system. Through an in-depth investigation of the power states in commercial 3G cellular networks, we then propose an energy-efficient burst transmission mechanism that can effectively increase the battery lifetime of user devices.

We have implemented a realistic prototype of *CloudMoV*, deployed on Amazon EC2 and Google App Engine, where EC2 instances serve as the mobile users' surrogates and GAE as the social cloud to handle the large volumes of social message exchanges. We conducted carefully designed experiments on iPhone 4S platforms. The experimental results prove the superior performance of *CloudMoV*, in terms of transcoding efficiency, power saving, timely social interaction, and scalability. The experiments also highlight the drawbacks of the current HTTP Live Streaming protocol implementation on mobile devices [24] as compared to our proposed burst transmission mechanism which achieves a 29.1 % increase of battery lifetime.

Much more, however, can be done to enhance *CloudMoV* to have product-level performance. In the current prototype, we do not enable sharing of encoded streams (in the same format/bit rate) among surrogates of different users. In our future work, such sharing can be enabled and carried out in a peer-to-peer fashion, *e.g.*, the surrogate of a newly joined user may fetch the transcoded streams directly from other surrogates, if they are encoded in the format/bit rate that the new user wants.

For implementing social interactions, most BigTable-like data stores (including GAE) support memcache [34] which is a highly efficient secondary storage on the data stores. We seek to integrate memcache support into *CloudMoV*, by possibly memcaching the data (*e.g.*, chat histories) of sessions where friends chat actively, so as to further improve the query performance. To sustain the portability of the system, we will stick to standard API interfaces, *i.e.*, JCache (JSR 107), in our system.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*,

vol. 8, pp. 14–23, 2009.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of IEEE INFOCOM*, 2012.

[3] Z. Huang, C. Mei, L. E. Li, and T. Woo, "Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy," in *INFOCOM'11*, 2011, pp. 201–205.

[4] T. Coppens, L. Trappeniers, and M. Godon, "AmigoTV: towards a social TV experience," in *Proc. of EuroITV*, 2004.

[5] N. Ducheneaut, R. J. Moore, L. Oehlberg, J. D. Thornton, and E. Nickell, "Social TV: Designing for Distributed, Sociable Television Viewing," *International Journal of Human-Computer Interaction*, vol. 24, no. 2, pp. 136–154, 2008.

[6] A. Carroll and G. Heiser, "An analysis of power consumption in as smartphone," in *Proc. of USENIXATC*, 2010.

[7] *What is 100% Pure Java*, http://www.javacoffeebreak.com/faq/faq0006.html.

[8] J. Santos, D. Gomes, S. Sargento, R. L. Aguiar, N. Baker, M. Zafar, and A. Ikram, "Multicast/broadcast network convergence in next generation mobile networks," *Comput. Netw.*, vol. 52, pp. 228–247, January 2008.

[9] *DVB-H*, http://www.dvb-h.org/.

[10] K. Chorianopoulos and G. Lekakos, "Introduction to social tv: Enhancing the shared experience with interactive tv," *International Journal of Human- Computer Interaction*, vol. 24, no. 2, pp. 113–120, 2008.

[11] M. Chuah, "Reality instant messaging: injecting a dose of reality into online chat," in *CHI '03 extended abstracts on Human factors in computing systems*, ser. CHI EA '03, 2003, pp. 926–927.

[12] R. Schatz, S. Wagner, S. Egger, and N. Jordan, "Mobile TV becomes Social - Integrating Content with Communications," in *Proc. of ITI*, 2007.

[13] R. Schatz and S. Egger, "Social Interaction Features for Mobile TV Services," in *Proc. of 2008 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2008.

[14] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, ser. SOSP '99, 1999, pp. 48–63.

[15] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03, 2003, pp. 149–163.

[16] G. Anastasi, M. Conti, E. Gregori, and A. Passarella, "Saving energy in wi-fi hotspots through 802.11 psm: an analytical model," in *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation, ESSLLI-2000*, 2004, pp. 24–26.

[17] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing," *Mobile Networks and Applications*, pp. 1–15, Apr. 2011.

[18] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, pp. 59–69, 2011.

[19] R. Pereira and K. Breitman, "A cloud based architecture for improving video compression time efficiency: The split & merge approach," in *DCC'11*, 2011, pp. 471–471.

[20] Z. Liu, Y. Feng, and B. Li, "Socialize Spontaneously with Mobile Applications," in *Proc. of IEEE INFOCOM*, 2012.

[21] W. Zhang, Y. Wen, Z. Chen, and A. Khisti, "Qoe-driven cache management for http adaptive bit rate (abr) streaming over wireless networks," in *Proc. of IEEE Globecom*, 2012.

[22] *Livestream*, http://itunes.apple.com/us/app/livestream/id379623629?mt=8/.

[23] *NoSQL Date base*, http://nosql-database.org/.

[24] *HTTP Live Streaming*, http://tools.ietf.org/html/draft-pantos-http-live-streaming-01.

[25] *3GPP TS 25.331*, http://www.3gpp.org/ftp/Specs/html-info/25331.htm.

[26] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," in *Proc. of OSDI*, 2006.

[27] *Amazon EC2*, http://aws.amazon.com/ec2/.

[28] *Kernal Based Virtual Machine*, http://www.linux-kvm.org/.

[29] *FFmpeg*, http://ffmpeg.org/.

[30] *Bluehost*, http://www.bluehost.com/.

[31] *Three HongKong*, http://www.three.com.hk.

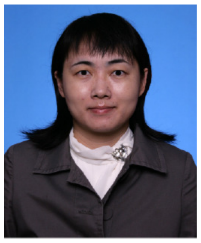
[32] *Xcode*, https://developer.apple.com/xcode/.

[33] *Google App Engine*, http://appengine.google.com/.

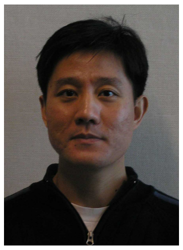
[34] *What is Memcached*, http://memcached.org/.

Yu Wu received his B.E. and M.E. degrees in 2006 and 2009 respectively from Department of Computer Science and Technology, Tsinghua University, China. He is currently a Ph.D. candidate in the Department of Computer Science, The University of Hong Kong, China. His research interests include cloud computing, mobile computing and SDN.

Zhizhong Zhang received his B.Sc. degree in 2011 from the Department of Computer Science, Sun Yat-sen University, China. He is currently a Ph.D. student in the Department of Computer Science, The University of Hong Kong, Hong Kong. His research interests include networks and systems.



Chuan Wu received her B.E. and M.E. degrees in 2000 and 2002 from Department of Computer Science and Technology, Tsinghua University, China, and her Ph.D. degree in 2008 from the Department of Electrical and Computer Engineering, University of Toronto, Canada. She is currently an assistant professor in the Department of Computer Science, The University of Hong Kong, China. Her research interests include cloud computing, peer-to-peer networks and online/mobile social network. She is a member of IEEE and ACM.



Zongpeng Li received his B.E. degree in Computer Science and Technology from Tsinghua University (Beijing) in 1999, his M.S. degree in Computer Science from University of Toronto in 2001, and his Ph.D. degree in Electrical and Computer Engineering from University of Toronto in 2005. Since August 2005, he has been with the Department of Computer Science in the University of Calgary. In 2011-2012, Zongpeng was a visitor at the Institute of Network Coding, Chinese University of Hong Kong. His research interests are in computer networks, particularly in network optimization, multicast algorithm design, network game theory and network coding. Zongpeng was named an Edward S. Rogers Sr. Scholar in 2004, won the Alberta Ingenuity New Faculty Award in 2007, was nominated for the Alfred P. Sloan Research Fellow in 2007, and received the Best Paper Award at PAM 2008 and at HotPOST 2012.

Francis C.M. Lau (SM, IEEE) received his Ph.D. in Computer Science from the University of Waterloo, Canada. He has been a faculty member in the Department of Computer Science, The University of Hong Kong, since 1987, where he served as the department head from 2000 to 2006. His research interests include networking, parallel and distributed computing, algorithms, and application of computing to art. He is the editor-in-chief of the Journal of Interconnection Networks.