A Cloud-based Transcoding Framework for Real-time Mobile Video Conferencing System

Rui Cheng, Wenjun Wu, Yongquan Chen, Yihua Lou
State Key Laboratory of Software Development Environment
Beihang University
Beijing, China
{chengrui,wwj,cyq,louyh}@nlsde.buaa.edu.cn

***Abstract*—High-definition video applications are often challenging for mobile devices due to their limited processing capability and bandwidth-constrained network connection. To deal with this issue, this paper proposes a cloud based transcoding framework to achieve scalable and efficient video adaption for mobile devices. This framework introduces a prediction-based scheduling algorithm to optimize both latency requirement and cloud utility cost for mobile clients. Following this framework, we developed a prototype system based on Windows Azure. Experiment results show that our system can effectively provide low-latency and cost-efficient video service to mobile clients.**

Keywords—transcode, cloud, azure

I. INTRODUCTION

In recent years, multimedia applications on mobile devices especially smart phones have been very popular. Thanks to the rapid development of wireless communication technologies, bandwidth of mobile connections including both 3G and 4G network become sufficient to sustain real-time video streaming. But high-definition video streaming still poses challenge for mobile devices due to their limited capability. Compared to the regular website and desktop client, mobile clients often have limited screen size, relatively slow network connection as well as restrained battery time. Such mismatch between the capability of mobile devices and video application can be further elaborated in two aspects:

1) Variability of the network environment: In contrast with the wired network for desktop-based Real-time Multimedia System, wireless networks for mobile clients tend to have more lossy and unstable connection where their bandwidth often changes in a more frequent and intense way. In order to guarantee smooth and high-quality video delivery for a mobile client, the system must dynamically adjust the bitrate of the video stream sent to the client to cope with the variability of mobile network environment.

2) Diversity in capability of mobile devices: Mobile devices are very diversified in terms of video processing. High end smart phones often have hardware accelator for video decoding. For example, the built-in support for hardware-decoding H.264 video, has been available among several models of smart phones, whereas a majority of mobile phones are not able to support hardware decoding. In addition, mobile devices' screen size and resolution are also varied, ranging from 800*480 to 1280*800 etc. For better video rendering performance, the system has to send videos in the suitable resolution to match the capability of the clients. In this case the system should send video streams in different encoding methods, based on hardware support situations for different kinds of clients.

Thus, it is essential to have a video transcoding proxy to perform video adaption for mobile devices to receive high-resolution and low-latency video streams. Traditionally, video transcoding is implemented on either peer-to-peer or distributed proxy architecture. But those approaches are not easy to scale up in practice. Because a peer-to-peer overlay consisting of dynamical on-and-off host machines cannot provide large amount of stable computing resources to run the computational intensive tasks demanded by video transcoding. With the high development of Cloud computing, especially mobile cloud [1][2], enables Cloud based transcoding service an effective solution to the problems arose in high quality mobile video streaming. It cannot only provision computing resources for transcoding workload in on-demand way but also introduce new features in optimizing transcoding processing, intelligently adapting video contents and reducing computation overhead on behalf of mobile devices.

The key design issue of a cloud-based real-time transcoding system is how to utilize cloud resources to provide the best quality-of-service to mobile clients with the minimization of video communication latency and utility cost incurred by transcoding. Our system uses Microsoft's Azure as the cloud platform, which provides reliable and comprehensive virtual platform so that developers can easily create/destroy the virtual machines, set message service bus, transfer information using the PAAS provided by Azure. This paper implements a real-time multimedia transcode system based on Windows Azure, which has the following features:

1) High elasticity: Taking the advantage of the Windows Azure platform, the system can quickly allocate computing and communication resources towards the dynamic needs of mobile clients. Whenever there are new requests issued by a mobile client, the system can quickly build a transcoding pipeline to match its capability with the requested video stream. Based on the workload predication model, the system can provision sufficient virtual machines for running this pipeline and network bandwidth for video transmission.

978-1-4799-4425-5/14 $31.00 © 2014 IEEE
DOI 10.1109/MobileCloud.2014.31

2) Low Latency Performance: Real-time video conferencing has latency constraints for end-to-end video communication. As the intermediate processing node between a video sender and a mobile receiver, the transcoding pipeline should be placed in the right Internet domain to avoid unnecessary communication delay. As an Internet-scale cloud service, Azure has data centers across all the continents, with high-speed network connecting each other. It becomes an ideal video delivery overlay where the video deliver filter in a transcoding pipeline can be allocated to the data center located in the Internet domain close to the destined mobile receiver.

3) Low cost: Cloud utility cost is a major concern for deploy mobile services in a public cloud infrastructure. The good strategy of reducing the cost incurred in video transcoding is to reuse overlapping media filters in multiple video transcoding pipeline for different mobile devices. This system introduces a transcoding-oriented task scheduling algorithm, which performs VM consolidation by deploying several transcoding filters into a single virtual machine and reuse some filters if possible. At the same time, the system also manages to process the video streams within the same data center as much as possible, resulting in the reduction of the network bandwidth.

The organization of the paper is as follows: Section 2 introduces the related work. Section 3 presents a system overview of our transcoding system based on mobile cloud. Section 4 describes a workload model of transcoding pipelines and discusses the optimization methods for pipeline construction, resource allocation and VM consolidation. Section 6 illustrates the experiment result of our prototype implementation. Section 7 gives the inclusion.

II. RELATED WORK

A. Overlay-based Transcoding

As for most distributed video application systems, transcoding is still the major solution to deal with the gap between requirement of high-resolution videos and regular mobile devices. Content Delivery Networks (CDNs) such as Akamai [3] are designed for large scale of data distribution through autonomous proxy nodes running at the edge of Internet. CDN can enable Internet service providers to achieve better load balancing and higher throughput, while reducing access latency and bandwidth requirements of their traffic. Proxy servers in a CDN can also be successfully used as video transcoding platforms to provide the necessary video adaption such as rate scaling, spatial converting, error handling and caching between multiple heterogeneous networks [4]–[6].

However, a CDN based transcoding solution has three major disadvantages in comparison to Cloud based solution: Firstly, the major CDN providers often charge customers with a high price even though the utilization of network traffic is low at the beginning. Secondly, the programmability of CDN service used to be not flexible enough to allow users to deploy customized services in their CDN nodes. Most research papers only discuss one or two transcoding process methods and barely mention how to dynamically deploy different transcoding filters on a proxy server. As a matter of fact, without virtualization technology, a CDN proxy can only support limited user configuration to customize their video output streams. Interestingly, the major CDN provider Akamai announced its cloud-based transcoding product named by Sola [7]. Last but not least, CDN transcoding service including Akamai's Sola is mostly designed for streaming applications, which are not latency sensitive as interactive videoconferencing applications. Our research focuses on latency-aware transcoding scheme based on cloud computing.

B. Peer-to-peer transcoding

Researchers [8] [9] have proposed to adopt Peer-to-Peer overlay as the underlying computing platform for video transcoding. In a peer-to-peer based scheme, a transcoding task is split up into segments and sent to every peer node in the system. Every node receiving the video stream is at same time

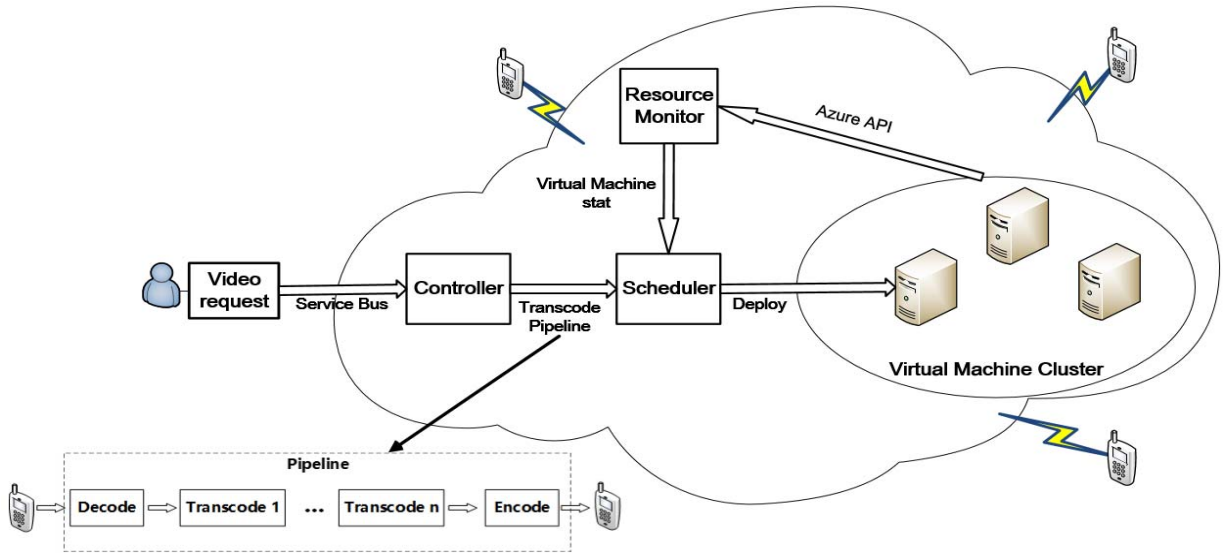

Figure 1. Architecture of the Cloud-based Video Transcoding System

undertaking either video transcoding or transmission task. Such a scheme can be effective with desktop based peer nodes if high ratio of peer nodes can stay reliable through video sessions. However this approach is not suitable for mobile device because they have limited computing power and constrained network connection to sustain computing-intensive transcoding tasks. Moreover, given the limited battery time for a mobile device, it cannot keep up running a time-lasting transcoding task through a long video session, which resulting in the instability of the entire peer-to-peer network. All the concerns in Peer-to-Peer based video transcoding can be easily addressed by mobile cloud-based approach. We can run proxies or surrogates on behalf of mobile devices to handle the computational and communication overhead of video transcoding in a stable and elastic cloud infrastructure.

C. Cloud based Video Transcoding

A few cloud-based transcoding systems [10] [11] [12] based on cloud have been developed. Some of the research efforts such as [10] [11] focus on video streaming applications that are less sensitive to communication latency. It introduces a map-reduce based method to support parallel video transcoding, in which a video stream is firstly divided into small unique fragments and then they are mapped to transcoding filters for processing. After transcoded, these generated new fragments will be merged and restored to a new video stream. Apparently this method can only work in a batch mode, which is not good for real-time video applications. Other cloud-based research papers haven't thoroughly discussed the issue of latency and utility cost in the context of cloud environment. This paper tries to introduce a cost-aware scheduling algorithm to minimize both the VM usage and communication latency for video transcoders running in Windows Azure.

III. SYSTEM OVERVIEW

Fig. 1 shows the major architecture of the system consisting of four components: resource monitor, transcoder, controller and task scheduler. All these components coordinate to manage transcoding pipelines for adapting mobile devices into the real-time video conferencing system. Communications among these components are implemented on top of reliable message queue service provided by Windows Azure, called service bus. The whole system can be deployed on virtual machines in the cloud platform, which can be launched dynamically according to the number of incoming requests. Each virtual machine runs a process monitor and several controllers, task schedulers or transcoding pipelines.

A. Resource monitor

Resource monitor is used to monitor the available resource of its hosting virtual machine as well as the resource occupied by the related system processes. These resources include CPU, memory, network bandwidth, and etc. It can check the resource status on both process level and system level. On each VM instance, Resource monitor can deploy a process probe to capture the status consumption of every process running in the VM. It can also query the status information for all the VMs through the Azure API. The information is collected periodically by the resource monitor and stored in the storage table which is a storage service provided by Azure. This information storage is accessible to all the other components including controllers and task schedulers so that they could make right decisions of system management based on up-to-date system status.

Table. 1 transcoding filters in our system

Name	Description
Unpack Filter	Unpack data packets in transport protocol such as RTP/RTCP
Pack Filter	Pack compressed data into transport protocol such as RTP/RTCP
Transmitter Filter	Send video packets to its destination
Decoder Filter	Decode compressed data to raw data
Encoder Filter	Encode raw multimedia data
Scale Filter	Change the video resolution
Frame Rate Filter	Change FPS in a video stream
ROI Extract Filter	Extract Region of Interest as a new video stream

B. Transcoder framework

For each mobile device joining in the system, it needs a transcoding pipeline with multiple media filters to transform the source video stream into a new stream for its processing. Each filter is responsible for a certain function, such as decoding, encoding, scaling, and frame rate controlling. In addition, custom transcode filters can be added to the system when a model of mobile device needs special video processing. Table 1 lists a few common filters that have been implemented in our system.

C. Controller

Controller is responsible for constructing a transcoding pipeline in response to the request by a mobile device. Such a request specifies the basic parameters of the video stream (video size, codec format, bitrate) wanted by the device based on its network bandwidth and hardware capability. After receiving the request message, the controller compares the request parameters of the target video stream and the attributes of the source video stream to start pipeline construction. The procedure of pipeline construction is an iterative process, during which the controller keeps searching for the suitable media filters from the list of the available filters and building up a cascading pipeline. The construction process ends when the last filter's output format matches the capability of the mobile device. After the pipeline is generated, its description will be forwarded to the task scheduler for further deployment and activation

D. Task Scheduler

Task scheduler is responsible for deploying every filter in the specified pipeline into a group of virtual machines. As mentioned in previous sections, it is a challenge to optimize the allocation and placement of transcoding pipelines in terms of computation overhead, latency and utility cost. This paper proposes a prediction based algorithm to achieve effective scheduling for transcoding tasks. This algorithm consists of two phases: training phase and scheduling phase.

Table.2. Parameters in Prediction Model

Parameters	Definition
width	The width of raw frame in pixel
height	The height of raw frame in pixel
fps	Frame rate per second of a video stream
framesize	The size of a raw video frame in byte, *framesize = (width * height * 1.5)/1024*
streamsize	The size of encoded stream to a transcoder in byte.

The training phase intends to profile all the media filters by running them in different types of virtual machines. After collecting the detailed information of resource consumption, the algorithm can build a linear regression model for the media filter. Combing all the models of the media filters in a transcoding pipeline, the task scheduler can make an accurate prediction of the transcoder's workload. During the schedule phase, the task scheduler performs VM allocation and placement to minimize the latency and utility cost for the pipeline. Multiple optimization strategies can be adopted in this process. For example, the scheduler can consolidate multiple transcoding pipelines in a single VM as long as the VM is not overloaded. The details of the workload prediction model and scheduling algorithm will be discussed in the next section

IV. OPTIMIZATION OF TRANSCODING PIPELINES

A. Transcoding Workload Prediction Model

Prediction model could predict the resource usage that a new transcoding task may consume. The CPU usage percent is the most important computing resource to judge whether a new transcoding task can be add to a virtual machine. Table 2 defines all the parameters used to predict the CPU usage. In our video conference system, the video resolution criteria which are usually used are *1920x1080, 1280x720, 640x480, 320x240 and*

176x144, the frame rate are *30fps, 25fps, 20fps, 15fps and 10fps*. Firstly we collect the average CPU usage percent of each kind transcoder and the parameters that listed in table 2, and then we build a linear prediction formula for this transcoder by linear regression [13].

For the decoder and encoder of H.264, the CPU usage is primarily affected by the size of input stream [14]. The input of decoder transcoder is compressed video data and scheduler can get the *streamsize* from the video sender before prediction. In Figure 2(a), the x-axis is the *streamsize* of which the unit is kilobytes per second and y-axis is the CPU usage percent. As it is shown in Figure 2(a), the CPU usage percent value varies linearly with the *streamsize*, and in different resolution, the line has different slope, while when the resolution is smaller than *640x480*, the prediction result can be consider as a constant value. So we can define the prediction formula of decoder as below:

$$P_{decoder} = \begin{cases} a * streamsize + b & width > 320 \; and \; height > 240 \\ C_{decoder} \end{cases} \quad (1)$$

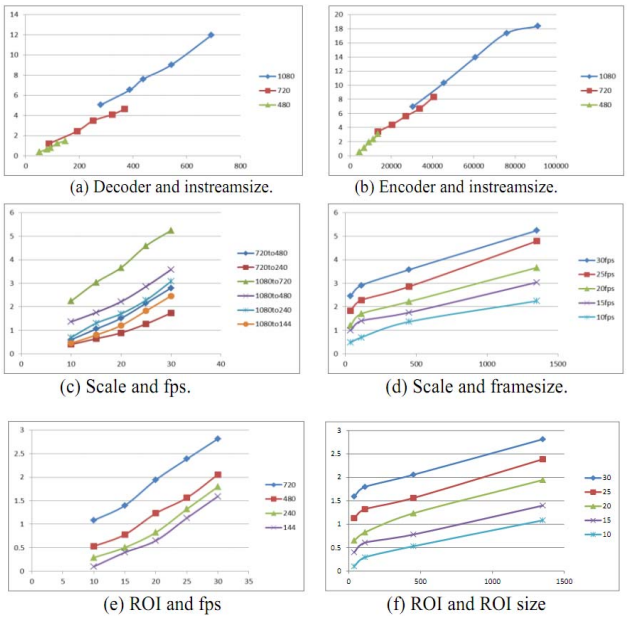

Fig.2. CPU usage of each kind of transcoder and its prediction parameters

The R value of linear regression of decoder is as high as 0.99, therefore we have obtained a reasonable good linear model to describe the relationship between the CPU usage percent and input bitrate. If the resolution of the input video is below 640x480, such as 320x240 and 176 x144, the CPU usage average will too small to be predicted. The experiment data show that the average of the CPU will not exceed 0.5%.

The CPU usage of encoder is mostly decided by the *framesize* and *fps* of the input video as it is shown in Figure 2(b), the x-axis is the input *streamsize* which equal with *fps*framesize*, its units is also KB/S, the y-axis is the CPU usage percent. Similar to decoder, we can use linear regression to construct a linear prediction formula for encoder. The same as decoder, the CPU usage percent of encoder can be regarded as a constant. Therefore the prediction formula will be:

$$P_{encoder} = \begin{cases} a \times fps \times b & width > 320 \; and \; height > 240 \\ C_{encoder} \end{cases} \quad (2)$$

The prediction model of the scale transcoding is defined in (3), which a little bit more complicated than that of decoder and encoder. There are three factors that can impact the CPU usage of the scale transcoder which are input *framesize*, output *framesize* and *fps*. Figure 2(c) shows the relationship of CPU usage and fps in different scale resolution pair, in which "1080to720" means the scale pair is a 1920x1080 video frame to a 1280x720 one, x-axis is the *fps* and y-axis is the CPU usage percent. Figure 2(d) shows the increase trend of CPU usage percent with the output *framesize*, the x-axis is the bytes in output frame of which the unit is KB/S. The R value is 0.9654 and that means this formula is still accurate enough. We do not use *streamsize* to predict because that the R value is only 0.72.

$$P_{scale} = a_1 \times in_{framesize} + a_2 \times fps + a_3 \times out_{framesize} + b \quad (3)$$

The I/O threads of the transcoding process will also consume some CPU resource. Figure 2(f) shows the relationship between CPU usage and the I/O stream size, the x-axis is the input or output stream size and the unit is KB/S, the y-axis is the CPU usage.

$$P_{I/O} = a \times streamsize + b \quad 4)$$

With the prediction formula of each kind of filter, we can predict the CPU usage of a new transcoding task by formula. And the total CPU usage of the transcoding process will be predicted by formula. Based on the prediction formula above, we can predict the total transcoding process with a linear equation as formula (5).

$$P_{totalCPU} = CPU_0 + \sum P(i) \quad (5)$$

CPU_0 is current CPU usage percent of transcoding service, and *P(i)* is the prediction CPU usage of the transcoding operation in new transcoding task.

B. Delay and Cost Model for Deploying Transcoding Pipelines on Windows Azure

In addition to the CPU usage, the delay performance is also an important factor for the deployment of a transcoding pipeline on a public cloud such as Windows Azure. Especially for real-time video conferencing applications, the latency of two-way video communication needs to be below 250ms. Network latency between VMs in a public cloud depends upon their location in the system. There are six regional data centers in Windows Azure, namely East Asia, South Asia, West Europe, North Europe, West America and East America. Due to the topology of interconnection in Windows Azure, communication latency within the same center is much lower than that between two different centers. In the scenario of single-source multicast, which is a common communication pattern in a video conference, the intermediate transcoding pipeline needs to be scattered into multiple centers to establish a single-source-tree overlay for mobile clients located in different regions of the world. Thus, the topology of the overlay actually determines the major fraction of its communication delay.

Assume there are ***N*** mobile clients who want to receive the same source video source ***S*** and ***M*** intermediate VMs hosting the transcoding pipelines for all the mobile clients. All the nodes construct a tree structure with ***S*** as its root node, VMs as its intermediate nodes and mobile clients as its leaf nodes. Let the delay of each client c be $Delay_c$. And its path to the video source *S* consists h_c hops: VM_1, …, VM_{hc} ..

$$Delay_c = Delay(S, V_0) + \sum_{i=1}^{h_c-1} Delay(V_i, V_{i+1}) + Delay(V_{h_c}, c) \quad (6)$$

The delay of each solution is the average delay of each client.

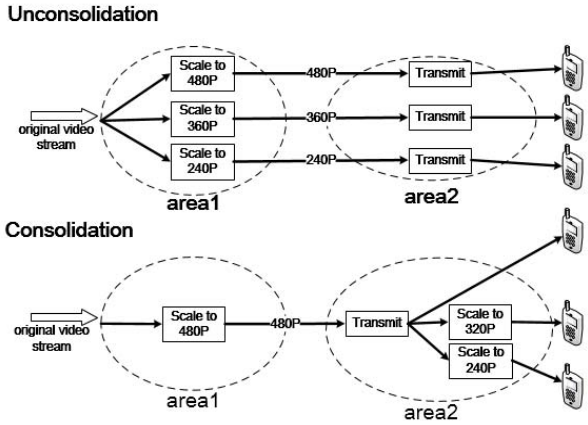

Fig. 3. Consolidation of transcoding pipelines with the same video source

$$\text{AvgDelay} = \frac{1}{n}\left(\sum_{i=1}^{n} Delay_i\right) \quad (7)$$

In addition to latency, utility cost of cloud resources is also a major concern for mobile users. For public cloud, users are typically charged by the amount of virtual machines and network traffic used by their applications. Take Windows Azure as an example. A small virtual machine (1.6GHz CPU, 1.75GRAM) running Linux costs $0.06 per hour and costs $0.09 per hour if it runs Windows. What's more, the Azure has several datacenters located all over the world. The inside data traffic within a datacenter is free whereas the outbound traffic cross datacenters costs 0.12 dollars per gigabyte. Given the intensive computing and communication nature of video transcoding, utility cost really matters. So it's always important to decrease the amount of virtual machine and data traffic across the datacenters. The total cost equals the sum of all the cost incured by VM usage and network traffic.

$$\text{Cost} = \sum virtual\ machine + \sum network\ traffic \quad (8)$$

C. Prediction-based Scheduling Alogrithm

Based on the above model of CPU, network latency and utility cost, we perform some preliminary experiments on Windows Azure and introduce optimization strategies for scheduling transcoding tasks

1) Locality of Virtual Machines: Transcode filter should be deployed in the domain where connected to the origin video stream.When the pipeline generated by the controller needs to cross multiple domains, we should allocate all the transcoding operations completed in a single cloud while other areas are just responsible to transmit the encoded video stream.

2) Consolidation of transcoding pipelines with the same video source: When a video source is delivered to multiple mobile devices with different display resolutions, a consolidation strategy can be applied to reduce extra traffic across multiple domains in Window Azure. Figue 3 illustrates such a scenario. Three mobile clients with the resolutions of 480P, 380P and 240P seperately want to receive a orginal video stream with the resolution of 720P. Conventional

approach is to send the transcoded video streams with the requested resolution to each client. But it will generates three video streams across two domains in Window Azure, which results in more utility cost. Our consolidation strategy takes a two-stage transcoding method, which converts 720P video into a new 480P stream and then transforms it to three different video streams. Thus, only a single inter-domain video transmission is needed. Because the relative small latency caused by transcoder, it can achieve better cost efficency without increasing too much cost.

3) Reuse duplicate transcode filters: When a video stream is transcoded into multiple target streams, there are some overlapping among the transcoding pipelines. They will include some filters with the same processing function and input video stream. During the deployment of these pipelines, the task scheduler can reuse the duplicate filters to reduce CPU usage. The scheduler first aligns the new pipeline description with the similar pipeline to figure out the overlapping part and determine the placement for reuse. Note that the alignment process sometimes needs to reorder the filters in the pipeline to increase the number of reusable filters. But it might introduce some overhead in pipeline execution. For example, when a transcoding pipeline has a scaling filter adjacent to a frame rate filter, although the position of two filters can be swapped, it is always a better choice to place the frame rate filter in front of the scaling filter. Because the frame rate filter often decreases the frame rate of a video stream, thus reducing the amount of the video data flowing into the scale filter.

Integrating the predication model and the optimization strategy, we can design a prediction-based transcoder scheduling algorithm shown in Figure 4. One of the major subroutine in the algorithm called tree-join algorithm is listed in Figure 5.

Suppose in the system there is the video source ***S***, and ***m*** virtual machines: VM_1, VM_2, …VM_m
When the controller receives a new incoming request of mobile client ***C***, it will create a transcoding pipeline PPL_c based on the capability of ***C*** and video source ***S***
According to the description of PPL_c = { $Filter_1$, .., $Filter_p$ },
The task scheduler checks the current status of the system and performs the following actions:

(1) Determine whether to create a new VM

The scheduler calculates the workload of the transcoding pipeline $P_{cpu}(PPL_c)$.
The schedule locates the domain Dc where the mobile client C resides.

If all the existing virtual machines cannot host more workload $P_{cpu}(PPL_c)$, go to Step (2).
Otherwise
It can check how many VMs are located in the same domain $\boldsymbol{D_c}$
If there are $n \leq m$ VMs in the $\boldsymbol{D_c}$ that can host the pipeline PPL_c, go to Step (3),

(2) Launch a new VM in the domain D_c for the transcoding pipeline PPL_c

Create a new VM_{m+1} based on the $P_{cpu}(PPL_c)$
Deploy PPL_c in the VM_{m+1}
Check the domain of the video source $\boldsymbol{D_s}$
If $\boldsymbol{D_s} == \boldsymbol{D_c}$, connect PPL_c directly to the video source ***S*** // in most cases, they are the same
Otherwise
Perform a tree-join algorithm to link PPL_c to the tree root ***S***

(3) Select suitable VM to place PPL_c in the D_c

The scheduler sort virtual machines according to the descending order by available workload $P_{cpu}(VM)$.
The scheduler select the first virtual machine in the sorted list VM0
Deploy the PPLc on VM_0
Update the available workload of VM0 $P_{cpu}(VM_0) = P_{cpu}(VM_0) + P_{cpu}(PPL_c)$
Perform a tree-join algorithm to link PPLc to the tree root S
Goto Step(3)

(4) Perform load balance algorithm to place PPL_c in *n* VMs in the D_c

In the sorted list (VM_0, VM_1, ……, VM_n) created in Step(3),
while($P_{cpu}(VM_0) - P_{cpu}(VM_n) > threhold$){
Move some task from VM_n to VM_0 and update the available workload
Resort the list according to descending order b available workload

Fig.4. Prediction-based transcoder scheduling algorithm

Suppose the tree root of video source S is *Root*
When a new pipeline PPL_c = { $Filter_1$, .., $Filter_p$ } is going to join the tree, the scheduler implement the tree-join algorithm in following actions.

(1) Initialize the index
Suppose the index i=1

(2) Search the proper child filter
List all the child filters of *Root*
If there exists a child filter *CFilter, CFilter* = $Filter_i$
Root = *CFilter*
i = i+1
goto step(2)
otherwise
goto step(3)

(3) Add the sub-pipeline to the *Root*
Add the sub-pipeline ($Filter_i$,……, $Filter_n$) to *Root*

Fig.5. tree-join algorithm of adding a new pipeline

V. EXPERIMENTAL EVALUATION

We built a test bed of multimedia mobile cloud using the virtual machine provided by azure. The size of the virtual machine varies from small (1.6GHz CPU, 1.75G RAM) to large (4*1.6GHz CPU, 56GRAM). The video stream under test is encoded in H.264 format. We adopt X264 encoder in the experiment and utilize FFMPEG to implement video filters such as decoding and scaling.

A. Estimation of Prediction Model Parameters

We collect data for CPU usage of all the transcode filters under different input video streams. Each test runs separately on three virtual machines with one, two and four cores. All the test results confirm the linear relationship between CPU usage and bitrate of input stream. Further analysis shows that the regression coefficients in all three cases are above 95%. Therefore, based on the linear model of transcoder filter workload, the task scheduler can get an accurate prediction of processing workload for the whole transcoding pipeline created by the controller. Table 3 shows all the CPU usages generated by the transcoding filters running on virtual machines with one, two and four core. Table 4 demonstrates the usage generated by a complete pipeline running on three virtual machines. As shown in the above tables, the workloads of running both single transcoding filters and the complete pipeline in a small VM are more cost efficient than running in a large VM. For example, for the scaling filter, its CPU usage on a single core is 5.1% under 720*480 resolution, which is nearly double than 2.25% CPU usage on a four-core VM. Because Windows Azure's utility policy is core based, it costs only 50% to run the scaling filter on a single core VM than a four-core VM. To show an intuitive comparison, we convert the results in the above tables by multiplying the CPU usage of a four-core virtual machine by four and the CPU usage of a two-core virtual machine by two to create Figure 6 and 7. The performance of running transcoding process on a four-node

TABLE. 3. The CPU usage of filters

transcode filter type		One core	Two cores	Four cores
decode	1080P	28.5%	14.0%	7.2%
	720P	14.5%	7.6%	3.6%
	480P	0.9%	1.19%	0.8%
encode	1080P	75.4%	37.8%	18.3%
	720P	33.5%	14.5%	6.77%
	480P	7.7%	6.3%	3.1%
scale	720P to 480P	5.1%	5%	2.25%
	720P to 320P	2.0%	2.0%	1.9%

TABLE.4. CPU usage of the complete pipeline

Input and Output Resolution		One core	Two cores	Four cores
Origin	Target			
1080P	720P	85.6%	47.7%	24.8%
	480P	69.8%	35.9%	17.6%
	320P	60.9%	32.8%	15.4%
	240P	60.1%	30.2%	15.7%
720P	480P	37.6%	22.9%	13.0%
	320P	30.2%	19.7%	9.8%
	240P	28.6%	16.1%	8.9%

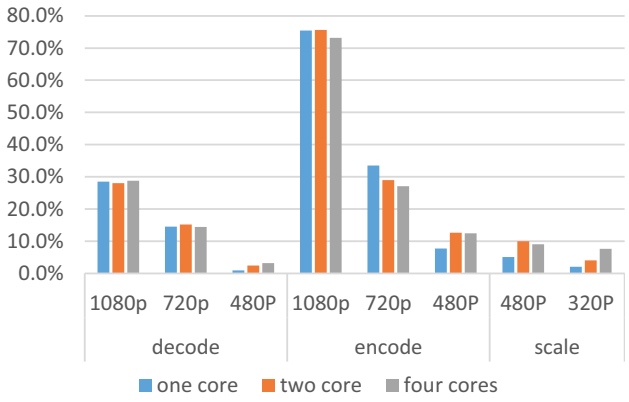

Fig.6. CPU usage of running video filters on different virtual machines

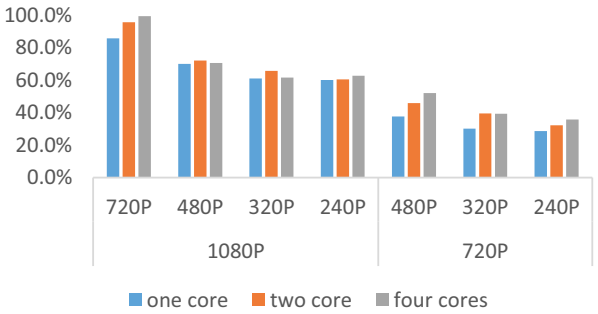

Fig. 7. The CPU usage of running a transcoding pipeline on different virtual machines

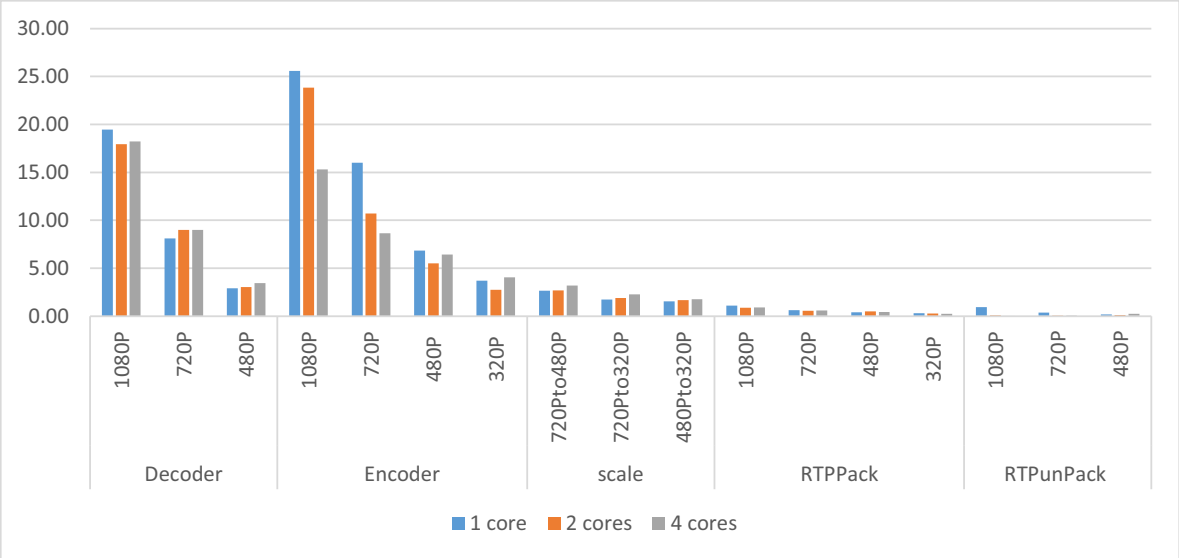

Fig.8. Latency introduced by different transcoding filters

Table 5. Delay between different domains(regions) in Window Azure (ms)

	SouthAsia	EastAsia	WestEurope	EastAmerica	NorthEurope	WestAmerica
SouthAsia	0	16	89	117	154	157
EastAsia		1	73	104	141	143
WestEurope			0	32	70	74
EastAmerica				0	39	40
NorthEurope					0	8
WestAmerica						0

cluster with one-core virtual machine is better than that of any other combinations including a two-node cluster with two-core VMs and a single node with four cores. Moreover, the performance difference among these combinations will decrease when the transcoding workload rises up in the case of transcoding a high-resolution video stream.

B. *Evaluation of latency of transcoding pipeline*

We measured the latency of each transcoding filter under different video inputs. The results are illustrated in Figure 8. The maximum delay caused by a video encoder is about 25ms. We also run ping-pong tests in Windows Azure to find out communication latency between domains of Windows Azure. From the result shown in Table 5, it confirms our hypothesis that the inter-domain latency is much higher than intra-domain latency. Moreover, the inter-domain latency is also much higher than the processing delay introduced by video filters. Therefore,
the majority of latency is caused by networking among VMs in the Windows Azure, which indicates the importance of latency-aware transcoding scheduling algorithm proposed in Section 4.

C. *Evaluation of scheduling algorithm*

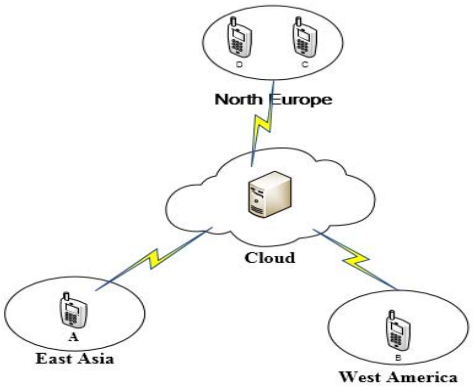

Fig. 9. Star-topology with a single VM connecting four clients

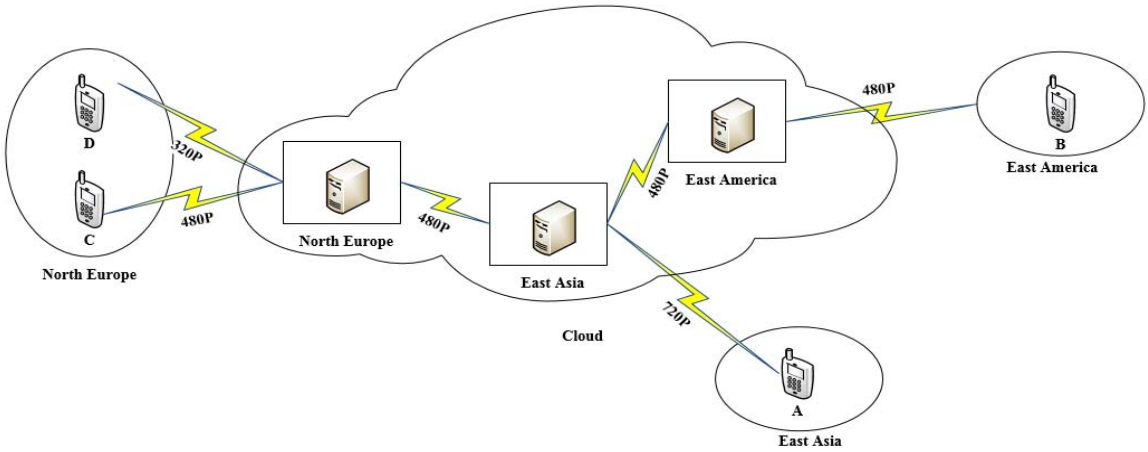

Fig 10. Optimized Deployment of Video Transcoding

We demonstrate the following videoconferencing scenario to show how our task scheduler performs optimization strategies to reduce the cost and delay of video transcoding pipelines. Assume that four clients, namely A, B, C, D, are having a video conference for two hours. Client A is a desktop video source located in the region of East Asia, where the rest are mobile clients residing in other regions. Client B stays in East America and C, D are in North Europe. Client A is sending a video stream with the resolution of 720P, while Client B and C can only handle video frames with the resolution of 480P, and Client D just can render video frames with the resolution of 320P. Without any optimization, the scheduler creates a star-topology for this single-source-multicast video session as shown in Figure 9. With the consolidation and reuse strategy in the algorithm, the schedule will construct a two-level tree topology shown in Figure 10.

1) star-topology: In the star-topology session, a centralized VM hosts two transcoding pipelines to transform 720p source stream into 480p and 320p videos for Client B-D. The delay result for Client B is $Delay_{WestAmerica} = 324ms$ and result for Client D and C is $Delay_{NorthEurope} = 358ms$. All the clients need to pay the same cost: $Cost_{East\ America} = Cost_{North\ Europe} = \$0.54$.

2) Two-level tree topology generated by our scheduling algorithm: According to the algorithm defined in Figure 4 and 5, the scheduler deploys three virtual machines in the domains of East Asia, North Europe and East America for all the clients. The VM running in the domain of East Asia receives the 720P video stream from Client A, transforms it into a 480P video stream and forwards it to VMs in North Europe and East America. The VMs can either directly forward the video stream or perform further transcoding to send a 320P video stream to its destination.

The average latency for client B-D is $232.24ms$ and the total cost is $1.193. Clearly, the algorithm can greatly reduce the latency of the transcoding pipeline while increase the utility cost by $0.65. The reason for the cost rise is because of increase in the number of VMs to lower the inter-domain latency. The scheduling algorithm can make a trade-off between latency and utility cost based on the priority of users. For users who are more sensitive to communication latency may want to pay more to get better QoS, while more tolerant users can decide not to. How to support user-level decision in the scheduling process is our further research topic.

VI. CONCLUSION

In this study, we design a cloud-based real-time transcoding system for bridging mobile devices into high definition video conferences with regular desktop clients. This system is developed on top of Windows Azure platform. In order to profile the workload of computationally intensive transcoders, we make extensive tests of running video filters on different virtual machines. Based on the workload predication model, this paper introduces a transcoding-oriented scheduling algorithm to decrease both network traffic and the computational overhead by performing optimal VM allocation based on the workload predication model. Preliminary experimental results on Windows Azure demonstrate the effective improvement of the scheduling algorithm. Future work includes better tree-optimization in the scheduling algorithm and large-scale use case studies.

REFERENCES

[1] Ruay-Shiung-Chang,Jerry Gao,Volker Gruhn, et al, Mobile Cloud Computing Research-Issues, Challenges and Needs, Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, Pages 442-453

[2] Niroshinie Fernando, Seng W. Loke, Wenny Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems (2013) page 84-108.

[3] B. Maggs and A. Technologies. Global internet content delivery. In First IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 12–12, 2001.

[4] T. Warabino, S. Ota, D. Morikawa, M. Ohashi, H. Nakamura, H. Iwashita, and F. Watanabe, "Video transcoding proxy for 3G wireless mobile Internet access," IEEE Commun. Mag., vol. 38, no. 10, pp. 66–71, Oct. 2000.

[5] Bo Shen, Sung-Ju Lee, Sujoy Basu ,Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks, IEEE Transcation on Multimedia, April, 2004.

[6] S. Dogan, A. Cellatoglu, M. Uyguroglu, A.H. Sadka, and A.M. Kondoz,Error-resilient video transcoding for robust inter-network communications using GPRS," IEEE Trans. Circuits Syst. Video Technol., vol. 12, pp. 453-464, June 2002.

[7] Akamai Sola, retrieved from http://www.akamai.com/html/solutions/sola-solutions.html

[8] Wu, J. C., Huang, P., Yao, J. J., & Chen, H. H. (2011). A collaborative transcoding strategy for live broadcasting over peer-to-peer IPTV networks.Circuits and Systems for Video Technology, IEEE Transactions on, 21(2), 220-224.

[9] Li, Z., Huang, Y., Liu, G., Wang, F., Zhang, Z. L., & Dai, Y. (2012, June). Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video (pp. 33-38). ACM.

[10] Lao Feng, Xinggong Zhang, Zongming Guo. Parallelizing Video Transcoding Using Map-Reduce-Based Cloud Computing[C], IEEE International Symposium on Circuits and Systems, May 20- 30. 2012.

[11] Adriana Garcia, Hari Kalva, Cloud Transcoding for Mobile Video Content Delivery[C], ICCE, 2011

[12] Bo Shen, Wai-Tian Tan, Frederic Huve. Dynamic Video Transcoding in Mobile Environments[C], IEEE MULTIMEDIA, 2008.

[13] Jiani Guo, Laxmi Narayan Bhuyan. Load Balancing in a Cluster-Based Web Server for Multimedia Applications. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 17, NO. 11, NOVEMBER 2006.

[14] A.D. Bavier, A.B. Montz, and L.L. Peterson, "Predicting Mpeg Execution Times," Proc. Int'l Conf. Measurements and Modeling of Computer Systems (SIGMETRICS), pp. 131-140, 1998.