Prediction-Based Dynamic Resource Allocation for Video Transcoding in Cloud Computing

Fareed Jokhio*†, Adnan Ashraf*‡§, Sébastien Lafond*‡, Ivan Porres*‡, Johan Lilius*‡
* Department of Information Technologies, Åbo Akademi University, Turku, Finland.
Email: { fjokhio, aashraf, slafond, iporres, jolilius }@abo.fi
† Quaid-e-Awam University of Engineering, Science & Technology, Nawabshah, Pakistan.
‡ Turku Centre for Computer Science (TUCS), Turku, Finland.
§ Department of Software Engineering, International Islamic University, Islamabad, Pakistan.

***Abstract*—This paper presents prediction-based dynamic resource allocation algorithms to scale video transcoding service on a given Infrastructure as a Service cloud. The proposed algorithms provide mechanisms for allocation and deallocation of virtual machines (VMs) to a cluster of video transcoding servers in a horizontal fashion. We use a two-step load prediction method, which allows proactive resource allocation with high prediction accuracy under real-time constraints. For cost-efficiency, our work supports transcoding of multiple on-demand video streams concurrently on a single VM, resulting in a reduced number of required VMs. We use video segmentation at *group of pictures* level, which splits video streams into smaller segments that can be transcoded independently of one another. The approach is demonstrated in a discrete-event simulation and an experimental evaluation involving two different load patterns.**

***Keywords*-Video transcoding; cloud computing; resource allocation; load prediction**

I. Introduction

As technology advances, so does the user expectations and demand for higher quality digital videos. Currently, a number of Television channels are broadcasting video contents in High-Definition (HD). Moreover, everyday, a large number of videos are uploaded on video hosting sites such as YouTube. Viewer expectations are always high as they want to download a video in the shortest possible time and want to watch videos in high quality.

For efficient use of storage and transmission media, digital videos are often stored and transmitted in compressed formats. Currently, there is a large number of video compression formats, such as MPEG-4 [1] and H.264 [2]. However, a device at the client-side may support only a subset of the existing formats. Therefore, for video streaming, an unsupported video format needs to be converted into another format, which is supported by the target device. Converting a compressed video into another compressed video is known as video transcoding [3]. This may change video format, bit rate, frame resolution, frame rate, or any combination of these.

Video transcoding is a computationally intensive operation, performed at the server-side. It may be done in real-time or in batch processing. However, for an on-demand video streaming service, if the required video is not available in the desired format, the transcoding needs to be done on-the-fly in real-time. One of the main challenges of a real-time video transcoding operation is that it must avoid over and underflow of the output video buffer, which temporarily stores the transcoded videos at the server-side. The overflow occurs if the video transcoding rate exceeds the video play rate and the capacity of the buffer. Likewise, the buffer underflow may occur when the play rate exceeds the transcoding rate, while the buffer does not contain enough frames either to avoid the underflow situation.

Video transcoding of a large number of video streams requires a large-scale cluster-based distributed system. Moreover, to handle varying amounts of load in a cost-efficient manner, the cluster should be dynamically scalable. Cloud computing provides theoretically infinite computing resources, which can be provisioned in an on-demand fashion under the pay-per-use business model [4]. Infrastructure as a Service (IaaS) clouds currently offer computing resources, such as virtual machines (VMs), storage, and network bandwidth [5], which can be used to create a dynamically scalable cluster of video transcoding servers.

In a cloud environment, a video transcoding operation can be performed in several different ways. For example, it is possible to map an entire video stream on a dedicated VM. However, it requires a large number of VMs to transcode several simultaneous streams. Moreover, transcoding of high resolution HD video streams can take more time, which may violate the client-side quality of service (QoS) requirements of desired play rate [6]. Another approach is to split the video streams into smaller segments and then transcode them independently of one another [7]. In this approach, one VM can be used to transcode a large number of video segments belonging to different video streams. Moreover, video segments of one particular stream can be transcoded on multiple VMs.

In this paper, we present prediction-based dynamic resource allocation and deallocation algorithms to scale video transcoding service on a given IaaS cloud in a horizontal fashion. The proposed algorithms allocate and deallocate VMs to a dynamically scalable cluster of video transcoding servers. We use a two-step load prediction method [8], which predicts a few steps ahead in the future to allow proactive resource allocation. For cost-efficiency, we share VM resources among multiple video streams. The sharing of the VM resources is based on the video segmentation, which splits the streams into

smaller segments that can be transcoded independently of one another. The proposed approach is evaluated in two simulation-based experiments involving two different load patterns. The results show that it provides cost-efficient resource allocation for a large number of simultaneous streams while avoiding over and underflow of the output video buffer.

We proceed as follows. In Section II, we describe video bit stream structure. Section III presents the system architecture. Section IV describes dynamic resource allocation algorithms. Our load prediction approach is detailed in Section V. Section VI presents simulation results. Section VII discusses important related works and Section VIII presents conclusion.

II. Video Bit Stream Structure

Video transcoding refers to the process of converting a compressed video stream from one format to another [3]. A compressed video stream consists of several independent sequences, as shown in Figure 1. A video sequence comprises a sequence header and one or more Groups of Pictures (GOPs). A GOP consists of different types of frames such as *I* (intra), *P* (predicted), and *B* (bi-directional predicted) containing all necessary information required to decode them. Both *I* and *P* frames can be used as reference frames. However, an *I* frame is an independent reference frame that does not require any other reference frame in the decoding process. A GOP starts with an *I* frame, which is followed by a number of *P* and *B* frames. Both *P* and *B* frames always require reference frames in the decoding process [1].

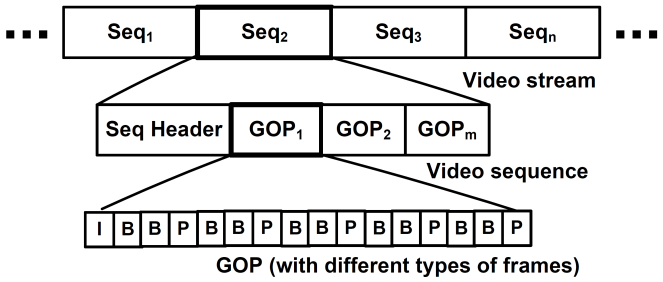

Fig. 1. Structure of a video down to frame level [1]

There are two types of group of pictures: Open-GOP and closed-GOP. In open-GOPs, a reference frame can be from any other GOP. Due to inter-dependency among them, the transcoding process requires frames from both GOPs. In closed-GOPs, all reference frames belong to the same GOP. Therfore, it represents an independent unit which can be transcoded without having any frames of other GOPs. In this paper, we assume that all video streams have closed-GOPs.

A GOP in MPEG-4 can have from 0 to 3 *B* frames between successive *P* frames. Usually, it is 2. The distance between successive *I* frames is *N*, which includes both *P* and *B* frames. In many cases, the value of *N* is 12, but it can be any value between 1 and a few hundreds.

Different kinds of frames also require a different amount of memory. Typically, *I* frames require the largest number of bytes to represent images, for example 300 Kilobytes (KB). The *P* frames require less memory, for example 160 KB. The *B* frames require even less, for example 40 KB.

III. System Architecture

The system architecture of the video transcoding service based on a dynamically scalable cluster of servers consists of a number of components, as shown in Figure 2. The video requests and responses are routed through the *streaming server*. Since the main focus of this paper is on dynamic resource allocation for the video transcoding service, we assume that the *streaming server* is not a bottleneck.

The video streams in certain compressed formats are stored in the *video repository*. After each transcoding operation, we store a copy of the transcoded video into the *video repository* for a certain amount of time, typically for two to three weeks. It allows us to avoid unnecessary repetition of the transcoding operations for a particular video. The *video splitter* splits the video streams into smaller segments called jobs, which are placed into the job queue. The *load balancer* routes and load balances these jobs to the *transcoding servers*. It maintains a configuration file, which contains information about *transcoding servers* that perform the transcoding operations. As a result of dynamic resource allocation and deallocation operations, the configuration file is often updated with new information. The *load balancer* serves the jobs in FIFO (First In, First Out) order. The *load balancer* implements the *shortest queue length* policy, which selects a *transcoding server* with the shortest queue length.

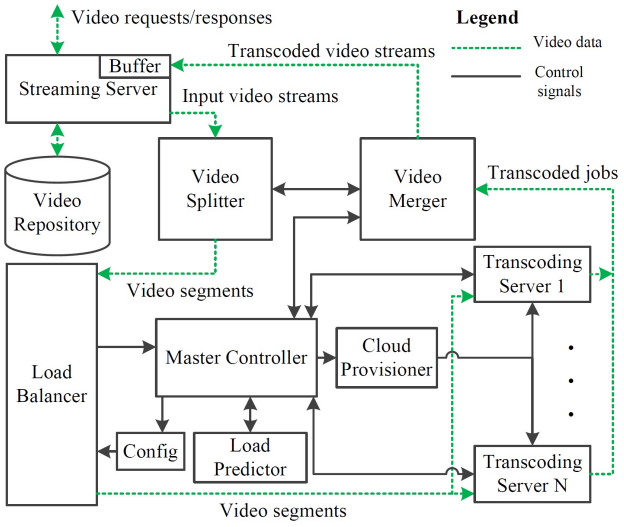

Fig. 2. System architecture

A *transcoding server* runs on a dynamically provisioned VM. Each *transcoding server* processes one or more simultaneous jobs. When a transcoding job arrives at a *transcoding server*, it is placed in the server's queue. The *master controller* acts as the main controller and resource allocator. It implements prediction-based dynamic resource allocation and deallocation algorithms, as described in Section IV. For load prediction, the *master controller* uses *load predictor*, which is

elaborated in Section V. The *cloud provisioner* refers to the cloud provisioner in an IaaS cloud, such as the provisioner in Amazon Elastic Compute Cloud (EC2) [9]. It performs the actual lower level tasks of starting and terminating VMs. The *video merger* merges the transcoded jobs into video streams, which form video responses.

Due to inter-dependencies among different types of frames, the video segmentation is performed at the key frames. The key frames are always *I* frames. Video segmentation at GOP level is discussed in more detail in [7], [10]. According to [10], the required time of segmentation for a 220 MB video is 0.75 seconds, while the transcoding time for the spatial resolution reduction from $16CIF(1408\text{x}1152)$ to $CIF(352\text{x}288)$ is 208 seconds. Therefore, when compared to the transcoding time, the overhead introduced by segmentation is negligible. In this paper, the video segmentation is performed at the GOP level. In our splitting method, a GOP is an atomic Unit. We split a video in such a way that each video segment has at least one GOP. Once a video is segmented, it consists of different independent segments that can be transcoded on different *transcoding servers* in any order.

Video segmentation of four video streams is shown in Figure 3. The output of the video splitter consists of a number of jobs, where each job has at least one GOP. The video splitter tries to manage segmentation in such a way that each user gets a smooth video stream from the *streaming server*. It takes into account the transcoding time and the play time of the video segment. Once a video segment is sent for transcoding, the next segment of the same stream is sent after some delay, based on the difference between the play time and the transcoding time.

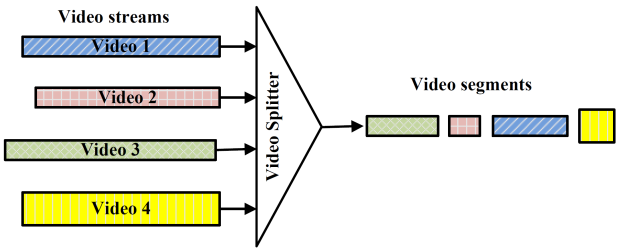

Fig. 3. Video segmentation

IV. Dynamic Resource Allocation Algorithms

In this section, we present the dynamic resource allocation and deallocation algorithms. For the sake of clarity, the concepts used in the algorithms and their notation are summarized in Table I. The algorithms implement prediction-based proactive control. They maintain a fixed minimum number of *transcoding servers* representing the base capacity N_B. Then, based on the incoming load, they add or remove *transcoding servers* from the dynamically scalable cluster.

On discrete-time intervals, the *master controller* obtains the target play rate (sum of target play rates of all video streams) $PR(t_i)$ from the *video merger* and the video transcoding rate from each *transcoding server*. Based on the individual server transcoding rates, it calculates the total transcoding rate $TR(t_i)$. Moreover, for proactive resource allocation, it uses *load predictor* to predict the total transcoding rate of all *transcoding servers* $\hat{TR}(t_i)$ a few steps ahead in the future.

TABLE I
Summary of Concepts and Their Notation

$count_{over}(t_i)$	over allocation count at discrete-time t_i
$S(t_i)$	set of *transcoding servers* at t_i
$S_p(t_i)$	set of newly provisioned servers at t_i
$S_c(t_i)$	servers close to completion of renting period at t_i
$S_t(t_i)$	servers selected for termination at t_i
$PR(t_i)$	sum of target play rates of all streams at time t_i
$TR(t_i)$	total transcoding rate of all servers at time t_i
$\hat{TR}(t_i)$	predicted total transcoding rate at time t_i
$RT(s,t_i)$	remaining time of server s at t_i
$V(t_i)$	set of video streams at t_i
$N_P(t_i)$	number of servers to provision at t_i
$N_T(t_i)$	number of servers to terminate at t_i
$getPR()$	get $PR(t_i)$ from video merger
$getTR(s)$	get transcoding rate of server s
$get\hat{TR}()$	get $\hat{TR}(t_i)$ from load predictor
$calN_P()$	calculate the value of $N_P(t_i)$
$calN_T()$	calculate the value of $N_T(t_i)$
$calRT(s,t_i)$	calculate the value of $RT(s,t_i)$
$delay()$	delay function
$provision(n)$	provision n servers
$select(n)$	select n servers for termination
$sort(S)$	sort servers S on remaining time
$terminate(S)$	terminate servers S
C_T	over allocation count threshold
RT_U	remaining time upper threshold
RT_L	remaining time lower threshold
B_L	buffer size lower threshold in megabytes
$B_S(t_i)$	size of the output video buffer in megabytes
B_U	buffer size upper threshold in megabytes
N_B	number of servers to use as base capacity
$startUp$	server startup delay
$jobCompletion$	job completion delay

The algorithms are designed to be cost-efficient while minimizing potential oscillations in the number of VMs [11]. This is desirable because, in practice, provisioning of a VM takes a few minutes [12]. Therefore, oscillations in the number of VMs may lead to deteriorated performance. Moreover, since some contemporary IaaS providers, such as Amazon EC2, charge on hourly basis, oscillations will result in a higher provisioning cost. Therefore, the algorithms counteract oscillations by delaying new resource allocation operations until previous resource allocation operations have been realized [13]. Furthermore, for cost-efficiency, the deallocation algorithm terminates only those VMs whose renting period approaches its completion.

A. Resource Allocation Algorithm

The resource allocation algorithm is given as Algorithm 1. The first two steps deal with the calculation of the target play rate $PR(t_i)$ of all streams and the total transcoding rate $TR(t_i)$ of all *transcoding servers*. The algorithm then obtains the predicted total transcoding rate $\hat{TR}(t_i)$ from the *load predictor*. Moreover, to avoid underflow of the output video buffer that temporarily stores transcoded jobs at the server-side, it considers the size of the output video buffer $B_S(t_i)$. If the target play rate exceeds the predicted transcoding rate

while the buffer size $B_S(t_i)$ falls below its lower threshold B_L, the algorithm chooses to allocate resources by provisioning one or more VMs. The number of VMs to provision $N_P(t_i)$ is calculated as follows

$$N_P(t_i) = \left\lceil \frac{PR(t_i) - \hat{TR}(t_i)}{\frac{TR(t_i)}{|S(t_i)|}} \right\rceil \quad (1)$$

The algorithm then provisions $N_P(t_i)$ VMs, which are added to the cluster of *transcoding servers*. To minimize potential oscillations due to unnecessary resource allocations, the algorithm adds a delay for the VM startup time. Moreover, to avoid the unnecessary provisioning of VMs, it ensures that the total number of VMs $|S(t_i)|$ does not exceed the total number of video streams $|V(t_i)|$. The algorithm adjusts the number of VMs to provision $N_P(t_i)$ if $|S(t_i)| + N_P(t_i)$ exceeds $|V(t_i)|$. This is desirable because the transcoding rate of a video on a single VM is usually higher than the required play rate.

Algorithm 1 Resource allocation algorithm

```
1:  while true do
2:    PR(t_i) := getPR()
3:    TR(t_i) := 0
4:    for s∈S(t_i) do
5:      TR(t_i) := TR(t_i) + getTR(s)
6:    end for
7:    T̂R(t_i) := getT̂R(TR(t_i))
8:    if T̂R(t_i) < PR(t_i) ∧ B_S(t_i) < B_L then
9:      N_P(t_i) := calN_P()
10:     if |S(t_i)| + N_P(t_i) > |V(t_i)| then
11:       N_P(t_i) := |V(t_i)| − |S(t_i)|
12:     end if
13:     if N_P(t_i) ≥ 1 then
14:       S_p(t_i) := provision(N_P(t_i))
15:       S(t_i) := S(t_i) ∪ S_p(t_i)
16:       delay(startUp)
17:     end if
18:   end if
19: end while
```

B. Resource Deallocation Algorithm

The resource deallocation algorithm is presented in Algorithm 2. The main objective of the algorithm is to minimize VM provisioning cost, which is a function of the number of VMs and time. Thus, it terminates any redundant VMs as soon as possible. Moreover, to avoid overflow of the output video buffer, it considers the size of the output video buffer $B_S(t_i)$. After obtaining the target play rate $PR(t_i)$ and the predicted total transcoding rate $\hat{TR}(t_i)$, the algorithm makes a comparison. If $\hat{TR}(t_i)$ exceeds $PR(t_i)$ while the buffer size $B_S(t_i)$ exceeds its upper threshold B_U, it may choose to deallocate resources by terminating one or more VMs. However, to minimize unnecessary oscillations, it deallocates resources only when the buffer overflow situation persists for a predetermined minimum amount of time.

In the next step, the algorithm calculates the remaining time of each *transcoding server* $RT(s, t_i)$ with respect to the completion of the renting period. It then checks if there are any *transcoding servers* whose remaining time is less than the predetermined upper threshold of remaining time RT_U and more than the lower threshold of remaining time RT_L. The objective is to terminate only those servers whose renting period is close to completion, while excluding any servers that are extremely close to the completion of their renting period and therefore it is difficult to terminate them before the start of the next renting period. If the algorithm finds at least one such server $S_c(t_i)$, it calculates the number of servers to terminate $N_T(t_i)$ as

$$N_T(t_i) = \left\lceil \frac{\hat{TR}(t_i) - PR(t_i)}{\frac{TR(t_i)}{|S(t_i)|}} \right\rceil - N_B \quad (2)$$

Then, it sorts the *transcoding servers* in $S_c(t_i)$ on the basis of their remaining time, and selects the servers with the lowest remaining time for termination. The rationale of sorting of servers is to ensure cost-efficiency by selecting the servers closer to completion of their renting period. A VM that has been selected for termination might have some pending jobs in its queue. Therefore, it is necessary to ensure that the termination of a VM does not abandon any jobs in its queue. One way to do this is to migrate all pending jobs to other VMs and then terminate the VM [12]. However, since transcoding of video segments takes relatively less time to complete, it is more reasonable to let the jobs complete their execution without requiring them to migrate and then terminate a VM when there are no more running and pending jobs on it. Therefore, the deallocation algorithm terminates a VM only when the VM renting period approaches its completion and all jobs on the server complete their execution. Finally, the selected servers are terminated and removed from the cluster.

V. Load Prediction

The existing load prediction models for web-based systems, such as [8], [14], [15], can be adapted to predict transcoding rate of the *transcoding servers* a few steps ahead in the future. Andreolini and Casolari [8] proposed a two-step approach to predict future load behavior under real-time constraints. The approach involves load trackers that provide a representative view of the load behavior to the load predictors, thus achieving two steps.

A load tracker (LT) filters out noise in the raw data to yield a more regular view of the load behavior [8]. It is a function $LT(\overrightarrow{S_n}(t_i)) : \mathbb{R}^n \rightarrow \mathbb{R}$, which inputs a measure s_i monitored at time t_i, and a set of previously collected n measures, that is $\overrightarrow{S_n}(t_i) = (s_{i-n}, ..., s_i)$, and provides a representation of the load behavior l_i at time t_i [8]. A sequence of LT values yields a regular view of the load behavior. There are different classes of LTs, such as simple moving average (SMA), exponential moving average (EMA), and cubic spline

Algorithm 2 Resource deallocation algorithm

while true do
  $PR(t_i) := getPR()$
  $TR(t_i) := 0$
  for $s \epsilon S(t_i)$ **do**
    $TR(t_i) := TR(t_i) + getTR(s)$
  end for
  $\hat{TR}(t_i) := get\hat{TR}(TR(t_i))$
  if $\hat{TR}(t_i) > PR(t_i) \wedge B_S(t_i) > B_U \wedge count_{over}(t_i) > C_T$ **then**
    for $s \epsilon S(t_i)$ **do**
      $RT(s, t_i) := calRT(s, t_i)$
    end for
    $S_c(t_i) := \{\forall s \epsilon S(t_i) | RT(s, t_i) < RT_U \wedge RT(s, t_i) > RT_L\}$
    if $|S_c(t_i)| \geq 1$ **then**
      $N_T(t_i) := calN_T()$
      $N_T(t_i) := min(N_T(t_i), |S_c(t_i)|)$
      if $N_T(t_i) \geq 1$ **then**
        $sort(S_c(t_i))$
        $S_t(t_i) := select(N_T(t_i))$
        $S(t_i) := S(t_i) \setminus S_t(t_i)$
        $delay(jobCompletion)$
        $terminate(S_t(t_i))$
      end if
    end if
  end if
end while

(CS) [14]. More sophisticated (time-series) models often require training periods to compute the parameters and/or off-line analyses [8]. Likewise, the linear (auto) regressive models, such as ARMA and ARIMA, may require frequent updates to their parameters [8], [15]. Therefore, in our approach, the *load predictor* implements an LT based on the EMA model, which limits the computation delay without incurring oscillations and computes an LT value for each measure with high prediction accuracy.

EMA is the weighted mean of the n measures in the vector $\overrightarrow{S_n}(t_i)$, computed at time t_i where $i > n$, where the weights decrease exponentially [8]. An EMA-based LT is defined as

$$EMA(\overrightarrow{S_n}(t_i)) = \alpha \cdot s_i + (1 - \alpha) \cdot EMA(\overrightarrow{S_n}(t_{i-1})) \quad (3)$$

where $\alpha = \frac{2}{n+1}$. The initial value of EMA, that is $EMA(\overrightarrow{S_n}(t_n))$, is computed as the arithmetic mean of the first n measures [8].

The load predictor (LP) is a function $LP_h(\overrightarrow{L_q}(t_i)) : \mathbb{R}^q \rightarrow \mathbb{R}$, which inputs a sequence of LT values $\overrightarrow{L_q}(t_i) = l_{i-q}, ..., l_i$ and outputs a predicted future value at time t_{i+h}, where $h > 0$ [8]. The LP is characterized by the prediction window h and the past time window q. Andreolini and Casolari [8] and Saripalli et al. [15] used linear regression of only two LT values, which are the first l_{i-q} and the last l_i values in the past time window. Ashraf et al. [16] used simple linear regression model [17], which takes into account all LT values $\overrightarrow{L_q}(t_i)$ in the past time window. The LP of the LT in this approach is based on a straight line defined as

$$l = \gamma_0 + \gamma_1 t \quad (4)$$

where γ_0 and γ_1 are called regression coefficients, which can be estimated at runtime based on the LT values [16], [17].

The results of the two-step approach depend upon selection of right values for the LT and LP parameters. The selected value of n should represent a good tradeoff between a reduced delay and a reduced degree of oscillations [8]. Likewise, q and h should be selected carefully.

VI. Simulation Results

Software simulations are often used to test and evaluate new algorithms involving complex environments [18]. We have developed a discrete-event simulation for the proposed resource allocation approach. The simulation is written in the Python programming language and is based on the SimPy simulation framework [19].

A. Experimental Design and Setup

We considered two different synthetic load patterns in two separate experiments. Load pattern 1 in experiment 1 consists of two load peaks, while load pattern 2 in experiment 2 has six load peaks. For simplicity, the renting period was assumed to be 600 seconds. The remaining time upper threshold RT_U was 60 seconds, while the remaining time lower threshold RT_L was 12 seconds. The LT and LP parameters were as follows: $n = 15$, $q = 30$, and $h = 120$.

The experiments used both SD (Standard-Definition) and HD video streams. At present, 10% of YouTube's videos are available in HD, while YouTube has more HD content than any other video hosting site [20]. However, the ratio of HD versus SD is expected to increase in the near future. Therefore, the load generation assumed 70% SD and 30% HD video streams. The video segmentation was performed at the GOPs level. The segmentation produced video segments, which were sent to the *transcoding servers* for execution. For HD videos, the average size of a video segment was 75 frames with a standard deviation of 7 frames. Likewise, for SD videos, the average size of a segment was 250 frames with a standard deviation of 20 frames. The total number of frames in a video stream was in the range of 15000 to 18000.

The desired play rate for a video stream is often fixed: 30 frames per seconds (fps) for SD videos and 24 fps for HD videos. Whereas, the transcoding rate depends on the video contents, such as, frame resolution, type of video format, type of frames, and contents of blocks. Different transcoding mechanisms also require different times.

In our experiments, the maximum transcoding rate for SD videos was assumed to be four times of its play rate. We further assumed that the transcoding rate is always higher than the the play rate of all video streams. Similarly, the minimum transcoding rate for SD videos was assumed to be double of its play rate. Since HD videos require more computation, the

maximum transcoding rate for an HD video was assumed to be double of the play rate, with the minimum transcoding rate at 1.5 times the play rate.

1) Experiment 1: Relatively Normal Load: The objective of experiment 1 was to simulate a relatively normal load. It was designed to generate a load representing a maximum of 200 simultaneous video streams in two different load peaks. In the first peak, the streams were ramped-up from 0 to 200, while adding a new stream every 20 seconds. After the ramp-up phase, the number of streams was maintained constant for 1 hour and then ramped-down to 100 streams.

The second peak ramped-up from 100 streams to 200 streams, while adding a new stream every 30 seconds. The ramp-up phase was followed by a similar constant phase as in the first peak. Then, the ramp-down phase removed all streams from the system.

2) Experiment 2: Highly Variable Load: Experiment 2 was designed to simulate a load pattern of a highly variable video demand. It generated a load representing a maximum of 290 simultaneous video streams consisting of six different load peaks. In the first peak, the streams were ramped-up from 0 to 170. Then, in the second peak from 110 to 290. Likewise, 210 to 280, 215 to 250, 120 to 200, and 100 to 170, respectively, in the third, fourth, fifth, and sixth peak. The stream ramp-up rate was 1 new stream per 30 seconds. Each ramp-up phase was followed by a ramp-down phase. Finally, the last ramp-down phase removed all streams from the system.

B. Results and Analysis

In both Figures 4 and 5, the number of servers plot shows dynamic resource allocation for the cluster of *transcoding servers*. The transcoding jobs plot represents the total number of jobs in the system at a particular time instance. It includes the jobs in execution at the *transcoding servers* and the jobs that were waiting in the queues. The target play rate plot shows the sum of target play rates of all video streams in the system. Likewise, the actual transcoding rate plot represents the total transcoding rate of all servers, while the predicted transcoding rate plot shows results of the load prediction. As described in Section IV, the resource allocation decisions were mainly based on the target play rate and the predicted transcoding rate.

1) Experiment 1: Relatively Normal Load: Figure 4 presents results from experiment 1. Experiment 1 used a maximum of 93 *transcoding servers* for a maximum of 200 simultaneous streams. There were a maximum of 9890 jobs in the system at a particular time. Moreover, a total of 4596 streams consisting of approximately 5×10^5 transcoding operations and 7×10^7 video frames were completed in 4 hours and 38 minutes. Therefore, the results indicate that the prediction-based resource allocation with the sharing of the VM resources among multiple streams resulted in a reduced number of total servers, which minimizes VM provisioning cost. Moreover, the proposed algorithms did not produce unnecessary oscillations in the number of VMs, which was also desirable for cost-efficiency.

The results show that the actual transcoding rate was always close to the target play rate. This was desirable to avoid over and underflow of the output video buffer in the system, as discussed in Section IV. Although the actual transcoding rate was sometimes slightly above or below the target play rate, the proactive resource allocation helped to ensure that the cumulative number of transcoded frames was always greater than the cumulative number of played frames.

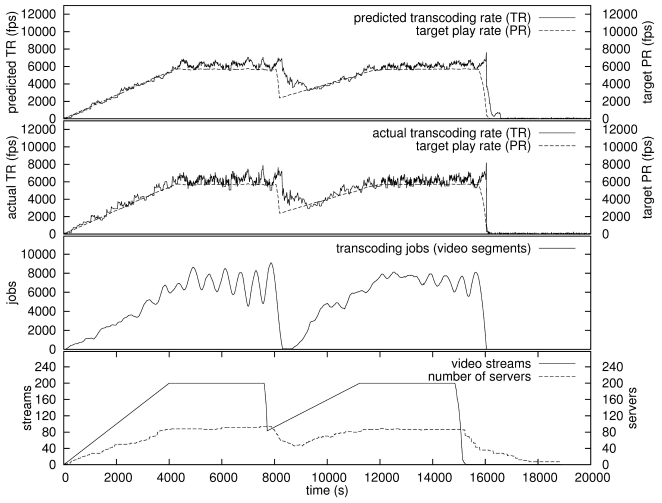

Fig. 4. Experiment 1 results

2) Experiment 2: Highly Variable Load: Figure 5 presents results from experiment 2. It used a maximum of 120 *transcoding servers* for a maximum of 290 simultaneous streams. There were a maximum of 14632 jobs in the system at a particular time. Moreover, a total of 7241 streams consisting of approximately 8×10^5 transcoding operations and 1×10^8 video frames were completed in 6 hours and 54 minutes. Although the number of streams was fluctuating rapidly, the algorithms provided a sustainable service with fewer VMs, while minimizing oscillations in the number of servers and avoiding the over and underflow of the output video buffer.

VII. Related Work

Distributed video transcoding with video segmentation was proposed in [7] and [10]. In these works, video segmentation was performed at the GOPs level. Jokhio et al. [7] presented bit rate reduction video transcoding using multiple processing units. The paper discussed computation, parallelization and data distribution among computing units. In [10], different video segmentation methods were analyzed to perform spatial resolution reduction video transcoding. The paper compared three possible methods of video segmentation. In both papers, video transcoding was not performed in the cloud and the resource allocation problem was not addressed. In contrast, the main focus of this paper is on resource allocation and deallocation algorithms.

Huang et al. [21] presented a cloud-based video proxy to deliver transcoded videos for streaming. The main contribution

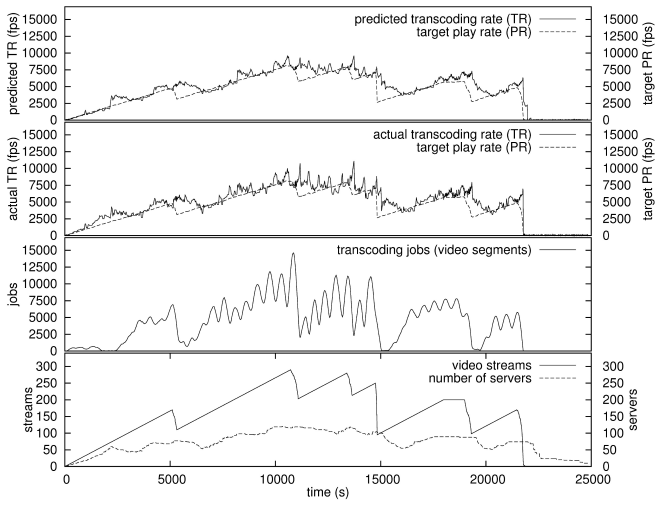

Fig. 5. Experiment 2 results

of their work is a multilevel transcoding parallelization framework. They used Hallsh-based and Lateness-first mapping to optimize transcoding speed and to reduce transcoding jitters. The performance evaluation was done on a campus cloud testbed and communication latency between cloud and video proxy was neglected. Li et al. [22] proposed *cloud transcoder*, which uses a compute cloud as an intermediate platform to provide transcoding service. Both papers do not discuss the resource allocation problem for video transcoding in cloud computing.

The existing works on dynamic resource allocation can be classified into two main categories: Plan-based approaches and control theoretic approaches. Plan-based approaches can be further classified into workload prediction approaches and performance dynamics model approaches. One example of the workload prediction approaches is Ardagna et al. [23], while TwoSpot [11], Hu et al. [24], Chieu et al. [25], Iqbal et al. [13] and Han et al. [26] use a performance dynamics model. Similarly, Dutreilh et al. [27], Pan et al. [28], Patikirikorala et al. [29], and Roy et al. [30] are control theoretic approaches. One common difference between all of these works and our proposed approach is that they are not designed specifically for video transcoding in cloud computing. In contrast, our proposed approach is based on the important resource allocation metrics for video transcoding service. Moreover, the proposed approach is cost-efficient as it uses fewer VMs for a large number of video streams and it counteracts possible oscillations in the number of VMs that may result in higher provisioning costs.

Ardagna et al. [23] proposed a distributed algorithm for managing Software as a Service (SaaS) cloud systems that addresses capacity allocation for multiple heterogeneous applications. The resource allocation algorithm takes into consideration a predicted future load for each application class and a predicted future performance of each VM, while determining possible SLA violations for each application type. The main challenge in the prediction-based approaches is in making good prediction models that should provide high prediction accuracy under real-time constraints. For this, we use a two-step prediction approach, which limits the computation delay without incurring oscillations, while providing high prediction accuracy.

TwoSpot [11] aims to combine existing open source technologies to support web applications written in different programming languages. It supports hosting of multiple web applications, which are automatically scaled up and down in a horizontal fashion. However, the scaling down is decentralized, which may lead to severe random drops in performance. For example, when all controllers independently choose to scale down at the same time. Hu et al. [24] proposed a heuristic algorithm that determines the server allocation strategy and job scheduling discipline which results in the minimum number of servers. They also presented an algorithm for determining the minimum number of required servers, based on the expected arrival rate, service rate, and SLA. Chieu et al. [25] presented an approach that scales servers for a particular web application based on the number of active user sessions. The main problem with this approach is in determining suitable threshold values on the number of user sessions. Iqbal et al. [13] proposed an approach for adaptive resource provisioning for read intensive multi-tier web applications. Based on response time and CPU utilization metrics, the approach determines the bottleneck tier and then scales it up by provisioning a new VM. Scaling down is supported by checking for any over-provisioned resources from time to time. Han et al. [26] proposed a reactive resource allocation approach to integrate VM-level scaling with a more fine-grained resource-level scaling. In contrast, the proposed approach provides proactive resource allocation, where the resource allocation decisions are based on the important video transcoding metrics, such as video play rate and server transcoding rate.

Dutreilh et al. [27] and Pan et al. [28] used control theoretic models for designing resource allocation solutions for cloud computing. Dutreilh et al. presented a comparison of static threshold-based and reinforcement learning techniques. Pan et al. used proportional integral (PI) controllers to provide QoS guarantees. Patikirikorala et al. [29] proposed a multi-model framework for implementing self-managing control systems for QoS management. Roy et al. [30] presented a look-ahead resource allocation algorithm based on the model predictive control. A common characteristic of the control theretic approaches is that they depend upon performance and dynamics of the underlying system. In contrast, the proposed approach does not require any knowledge about the performance and dynamics of the *transcoding servers*.

VIII. Conclusion

In this paper, we presented prediction-based dynamic resource allocation algorithms to scale video transcoding service in a cloud environment. The proposed algorithms provide a mechanism for creating a dynamically scalable cluster of video transcoding servers by provisioning VMs from an IaaS cloud. The prediction of the future user load is based on

a two-step load prediction method, which allows proactive resource allocation with high prediction accuracy under real-time constraints. For cost-efficiency, we used segmentation of video streams, which splits a stream into smaller segments that can be transcoded independently of one another. This helped us to perform video transcoding of multiple streams on a single server. The proposed approach is demonstrated in a discrete-event simulation. The evaluation and analysis considered two different synthetic load patterns in two separate experiments. Experiment 1 used a relatively normal load, while experiment 2 used a highly variable load. The results show that the proposed approach provides cost-efficient resource allocation for transcoding a large number of video streams, while minimizing oscillations in the number of servers and avoiding over and underflow of the output video buffer.

ACKNOWLEDGEMENTS

This work was supported by the Cloud Software Finland research project and by an Amazon Web Services research grant. Fareed Jokhio and Adnan Ashraf were partially supported by a doctoral scholarship from the Higher Education Commission (HEC) of Pakistan.

REFERENCES

[1] J. Watkinson, *The MPEG Handbook: MPEG-1, MPEG-2, MPEG-4*, ser. Broadcasting and communications. Elsevier/Focal Press, 2004.

[2] T. Wiegand, G. J. Sullivan, and A. Luthra, "Draft ITU-T recommendation and final draft international standard of joint video specification," in *Technical Report*, 2003.

[3] A. Vetro, C. Christopoulos, and H. Sun, "Video transcoding architectures and techniques: an overview," *Signal Processing Magazine, IEEE*, vol. 20, no. 2, pp. 18 – 29, mar 2003.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[5] J. Rhoton and R. Haukioja, *Cloud Computing Architected: Solution Design Handbook*. Recursive Press, 2011.

[6] N. Bjork and C. Christopoulos, "Transcoder architectures for video coding," *Consumer Electronics, IEEE Transactions on*, vol. 44, no. 1, pp. 88 –98, feb 1998.

[7] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, "Bit rate reduction video transcoding with distributed computing," in *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, feb. 2012, pp. 206 –212.

[8] M. Andreolini and S. Casolari, "Load prediction models in web-based systems," in *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, ser. valuetools '06. New York, NY, USA: ACM, 2006.

[9] "Amazon Elastic Compute Cloud." [Online]. Available: http://aws.amazon.com/ec2/

[10] F. A. Jokhio, T. Deneke, S. Lafond, and J. Lilius, "Analysis of video segmentation for spatial resolution reduction video transcoding," in *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposiuml*, Dec 2011, p. 6 pp.

[11] A. Wolke and G. Meixner, "TwoSpot: A cloud platform for scaling out web applications dynamically," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, E. Di Nitto and R. Yahyapour, Eds. Springer Berlin / Heidelberg, 2010, vol. 6481, pp. 13–24.

[12] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, "Feedback control algorithms to deploy and scale multiple web applications per virtual machine," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, sept. 2012, pp. 431 –438.

[13] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871 – 879, 2011.

[14] M. Andreolini, S. Casolari, and M. Colajanni, "Models and framework for supporting runtime decisions in web-based systems," *ACM Trans. Web*, vol. 2, no. 3, pp. 17:1–17:43, Jul. 2008.

[15] P. Saripalli, G. Kiran, R. Shankar, H. Narware, and N. Bindal, "Load prediction and hot spot detection models for autonomic cloud computing," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, dec. 2011, pp. 397 –402.

[16] A. Ashraf, B. Byholm, and I. Porres, "A session-based adaptive admission control approach for virtualized application servers," in *Utility and Cloud Computing (UCC), 5th IEEE/ACM International Conference on*, 2012, pp. 65–72.

[17] D. Montgomery, E. Peck, and G. Vining, *Introduction to Linear Regression Analysis*, ser. Wiley Series in Probability and Statistics. John Wiley & Sons, 2012.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[19] N. Matloff, *A Discrete-Event Simulation Course Based on the SimPy Language*. University of California at Davis, 2006. [Online]. Available: http://heather.cs.ucdavis.edu/ matloff/simcourse.html

[20] "35 mind numbing youtube facts, figures and statistics infographic," 2012/05/23. [Online]. Available: http://www.jeffbullas.com/2012/05/23/35-mind-numbing-youtube-facts-figures-and-statistics-infographic/

[21] Z. Huang, C. Mei, L. E. Li, and T. Woo, "Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy," in *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*. IEEE, 2011, pp. 201–205.

[22] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai, "Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices," in *The 22nd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2012.

[23] D. Ardagna, C. Ghezzi, B. Panicucci, and M. Trubian, "Service provisioning on the cloud: Distributed algorithms for joint capacity allocation and admission control," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, E. Di Nitto and R. Yahyapour, Eds. Springer Berlin / Heidelberg, 2010, vol. 6481, pp. 1–12.

[24] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '09. New York, NY, USA: ACM, 2009, pp. 101–111.

[25] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, oct. 2009, pp. 281 –286.

[26] R. Han, L. Guo, M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, may 2012, pp. 644 –651.

[27] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck, "From data center resource allocation to control theory and back," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, july 2010, pp. 410 –417.

[28] W. Pan, D. Mu, H. Wu, and L. Yao, "Feedback control-based QoS guarantees in web application servers," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, sept. 2008, pp. 328 –334.

[29] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A multi-model framework to implement self-managing control systems for QoS management," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 218–227.

[30] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, july 2011, pp. 500 –507.