Fareed Jokhio | Adnan Ashraf | Sébastien Lafond | Ivan Porres | Johan Lilius

Cost-Efficient Dynamically Scalable Video Transcoding in Cloud Computing

Turku Centre *for* Computer Science

TUCS Technical Report
No 1098, December 2013

Cost-Efficient Dynamically Scalable Video Transcoding in Cloud Computing

Fareed Jokhio
Adnan Ashraf
Sébastien Lafond
Ivan Porres
Johan Lilius

Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5A, 20520, Turku, Finland
{fjokhio, aashraf, slafond, iporres, jolilius}@abo.fi

TUCS Technical Report
No 1098, December 2013

Abstract

Video transcoding of a large number of on-demand videos requires a large scale cluster of transcoding servers. Moreover, storage of multiple transcoded versions of each source video requires a large amount of disk space. Infrastructure as a Service (IaaS) clouds provide virtual machines (VMs) for creating a dynamically scalable cluster of servers. Likewise, a cloud storage service may be used to store a large number of transcoded videos. Moreover, it may be possible to reduce the total IaaS cost by trading storage for computation, or vice versa. In this paper, we present prediction-based dynamic resource allocation algorithms to scale on-demand video transcoding service on a given IaaS cloud. The proposed algorithms provide mechanisms for allocation and deallocation of VMs to a dynamically scalable cluster of video transcoding servers in a horizontal fashion. We also present a computation and storage trade-off strategy for cost-efficient video transcoding in the cloud called cost and popularity score based strategy. The proposed strategy estimates computation cost, storage cost, and video popularity of individual transcoded videos and then uses this information to make decisions on how long a video should be stored or how frequently it should be re-transcoded from a given source video. The proposed algorithms and the trade-off strategy are demonstrated in a discrete-event simulation and are empirically evaluated using a realistic load pattern.

Keywords: Video transcoding, dynamic resource allocation, computation and storage trade-off, cost-efficiency, cloud computing

TUCS Laboratory
Embedded Systems Laboratory
Software Engineering Laboratory

1 Introduction

With an ever increasing number of digital videos delivered everyday via the Internet, the number of video formats and video codecs used for digital video representation are also increasing rapidly. Moreover, since video streaming of a large number of videos requires a lot of server-side resources, digital videos are often stored and transmitted in compressed formats to conserve storage space and communication bandwidth. With the emergence of a large number of video compression techniques and packaging formats, such as MPEG-4 [32] and H.264 [33], the diversity of digital video content representation has grown even faster. However, for a client-side device, it is practically impossible to support all the existing video formats. Therefore, an unsupported format needs to be converted into one of the supported formats before the video could be played on the device.

The process of converting a compressed digital video from one format to another format is termed as video transcoding [31]. It may involve extracting video and audio tracks from the file container, decoding the tracks, down-scaling frame-size, dropping of frames, reducing bit-rate by applying coarser quantization, encoding the audio and video tracks into a suitable format, and packing those tracks into a new container. Since video transcoding is a compute-intensive operation, transcoding of a large number of on-demand videos requires a large scale cluster of transcoding servers. Similarly, storage of multiple transcoded versions of each source video requires a large amount of disk space. Moreover, in order to be able to handle different load conditions in a cost-efficient manner, the cluster of transcoding servers should be dynamically scalable.

Cloud computing provides theoretically infinite computing and storage resources, which can be provisioned in an on-demand fashion under the pay-per-use business model [4]. Infrastructure as a Service (IaaS) clouds, such as Amazon Elastic Compute Cloud (EC2)[1], provide Virtual Machines (VMs) for creating a dynamically scalable cluster of servers. Likewise, a cloud storage service may be used to store a large number of transcoded videos. Determining the number of VMs and the amount of storage to provision from an IaaS cloud is an important problem. The exact number of VMs and the exact amount of storage needed at a specific time depend on the incoming load from service users and their performance requirements.

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-definition (HD) video streams may require a lot of time, which may violate the client-side performance requirements of the desired play rate [9]. Another approach is to split the video streams into smaller segments and then transcode them independently of one another [19]. In this approach, one VM can be used to transcode a large number of

[1] http://aws.amazon.com/ec2/

video segments belonging to different video streams. Moreover, video segments of a particular stream can be transcoded on multiple VMs.

In this paper, we present prediction-based dynamic resource allocation and deallocation algorithms [22] 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 [2], which predicts the video transcoding rate a few steps ahead in the future to allow proactive resource allocation under soft realtime constraints. For cost-efficiency, we share VM resources among multiple video streams. The sharing of the VM resources is based on video segmentation, which splits the streams into smaller segments that can be transcoded independently of one another [22]. We also investigate the computation and storage cost trade-off for video transcoding in the cloud and present a cost-efficient strategy called cost and popularity score based strategy [21]. The proposed strategy estimates computation cost, storage cost, and video popularity of individual transcoded videos and then uses this information to make decisions on how long a video should be stored or how frequently it should be re-transcoded from its source video. The objective is to reduce the total IaaS cost by trading storage for computation, or vice versa. Thus, the paper makes two contributions: (1) proactive resource allocation and deallocation algorithms to scale video transcoding service on a given IaaS cloud; and (2) a computation and storage cost trade-off strategy for video transcoding in cloud computing. It extends the works published in [20], [21], and [22] and provides an extended evaluation. The proposed algorithms and the trade-off strategy are demonstrated in discrete-event simulations and are empirically evaluated using a realistic load pattern.

We proceed as follows. Section 2 presents the system architecture of an on-demand video transcoding service and sets the context for the proposed dynamic resource allocation algorithms and the proposed trade-off strategy. Section 3 describes the proposed algorithms. The proposed trade-off strategy is presented in Section 4. Section 5 describes experimental design and presents the results of the experimental evaluation. In Section 6, we discuss important related works before concluding in Section 7.

2 System Architecture

The system architecture of the cloud-based on-demand video transcoding service is shown in Figure 1. It consists of a *streaming server*, a *video splitter*, a *video merger*, a *video repository*, a dynamically scalable cluster of *transcoding servers*, a *load balancer*, a *master controller*, and a *load predictor*. The video requests and responses are routed through the streaming server. It uses an output video buffer, which temporarily stores the transcoded videos at the server-side. Our resource allocation algorithms are designed to avoid over and underflow of the video buffer.

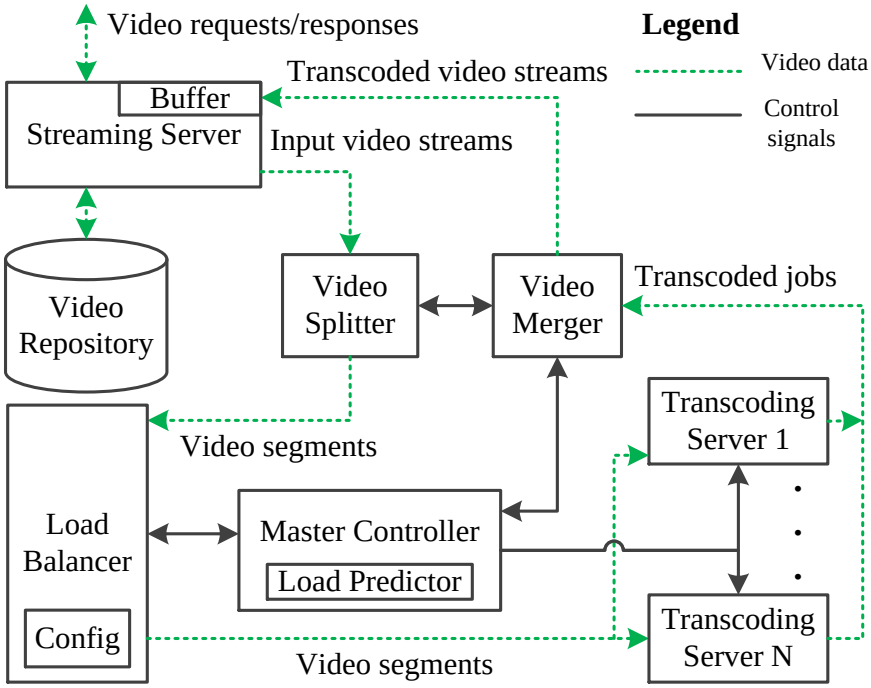

Figure 1: System architecture of the cloud-based on-demand video transcoding service

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. Since the main focus of this paper is on video transcoding, we assume that the streaming server is not a bottleneck.

The video streams in certain compressed formats are stored in the video repository. The streaming server accepts video requests from users and checks if the required video is available in the video repository. If it finds the video in the desired format and resolution, it starts streaming the video. However, if it finds that the requested video is stored only in another format or resolution than the one desired by the user, it sends the video for segmentation and subsequent transcoding. Then, as soon as it receives the transcoded video from the video merger, it starts streaming the video.

After each transcoding operation, the computation and storage trade-off strategy determines if the transcoded video should be stored in the video repository or not. Moreover, if a transcoded video is stored, then the trade-off strategy also determines the duration for which the video should be stored. Therefore, it allows us to trade computation for storage or vice versa in order to reduce the total operational cost and to improve performance of the transcoding service.

The video splitter splits the video streams into smaller segments called jobs, which are placed into the job queue. A compressed video consists of three

different types of frames namely, *I*-frames (intracoded frames), *P*-frames (predicted frames), and *B*-frames (bi-directional predicted frames). Due to interdependencies among different types of frames, the video splitting or segmentation is performed at the key frames, which are always I-frames. An I-frame followed by P and B frames is termed as a group of pictures (GOP). GOPs represent atomic units that can be transcoded independently of one another [22]. Video segmentation at GOP level is discussed in more detail in [19] and [23].

The load balancer employs a task assignment policy, which distributes load on the transcoding servers. In other words, it decides when and to which transcoding server a transcoding job should be sent. It maintains a configuration file, which contains information about transcoding servers that perform the transcoding operations. As a result of the 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. It implements one or more job scheduling policies, such as, the *shortest queue length* policy, which selects a transcoding server with the shortest queue length and the *shortest queue waiting time* policy, which selects a transcoding server with the least queue waiting time.

The actual transcoding is performed by the transcoding servers. They get compressed video segments, perform the required transcoding operations, and return the transcoded video segments for merging. 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 from where it is subsequently processed.

The master controller acts as the main controller and the resource allocator. It implements prediction-based dynamic resource allocation and deallocation algorithms, as described in Section 3. It also implements one or more computation and storage trade-off strategies, such as the proposed cost and popularity score based strategy, which is presented in Section 4. In our approach, the resource allocation and deallocation is mainly based on the target play rate of the video streams and the predicted transcoding rate of the transcoding servers. For load prediction, the master controller uses load predictor, which predicts future load on the transcoding servers. The video merger merges the transcoded jobs into video streams, which form video responses. Our load prediction approach is described in detail in [7] and [22]. It consists of a load tracker and a load predictor [2]. We use exponential moving average (EMA) for the load tracker and a simple linear regression model [26] for the load predictor.

3 Proactive VM Allocation Algorithms

In this section, the proposed dynamic VM allocation and deallocation algorithms for video transcoding in the cloud are presented. The objective is to reduce the over and under allocation of resources while satisfying the client-side performance

requirements. For the sake of clarity, the concepts used in the algorithms and their notation are summarized in Table 1. The algorithms implement proactive control, which uses a two-step load prediction approach [2] in which the current and the past system load is tracked to predict the future system load. The predicted system load is then used to make decisions on the allocation and deallocation of VMs to a dynamically scalable cluster of transcoding servers. Moreover, a fixed minimum number of transcoding servers is always maintained, which represents the base capacity N_B.

On discrete-time intervals, the master controller obtains the play rate of all video streams and adds them together to get the total target play rate $PR(t)$. It then obtains the video transcoding rate from each transcoding server and calculates the total transcoding rate $TR(t)$. Moreover, for proactive VM allocation, it uses load predictor to predict the total transcoding rate $\hat{TR}(t)$ a few steps ahead in the future.

The algorithms are designed to be cost-efficient while minimizing potential oscillations in the number of VMs [34]. This is desirable because, in practice, provisioning of a VM takes a few minutes [5], [6]. 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 VM allocation operations until previous VM allocation operations have been realized [18]. Furthermore, for cost-efficiency, the deallocation algorithm terminates only those VMs whose renting period approaches its completion.

3.1 VM Allocation Algorithm

The VM allocation algorithm is given as Algorithm 1. The first two steps deal with the calculation of the target play rate $PR(t)$ of all streams and the total transcoding rate $TR(t)$ of all transcoding servers (lines 3–7). The algorithm then obtains the predicted total transcoding rate $\hat{TR}(t)$ from the load predictor (line 8). 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 $B_S(t)$. If the target play rate exceeds the predicted transcoding rate while the buffer size $B_S(t)$ falls below its lower threshold B_L (line 9), the algorithm chooses to allocate resources by provisioning one or more VMs (line 10). The number of VMs to provision $N_P(t)$ is calculated as follows

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

where $|S(t)|$ is the number of transcoding servers at time t. The VM allocation algorithm also takes into account the number of jobs waiting in the servers' queues. It checks the average queue length of all servers $avgQJobs(t)$ and if the average queue length is above a predefined maximum upper threshold $MAXQL_{UT}$

Table 1: Summary of concepts and their notation for VM allocation algorithms

Notation	Description
$avgQJobs(t)$	average queue length of all servers at discrete-time t
$count_{over}(t)$	over allocation count at t
$N_P(t)$	number of servers to provision at t based on $PR(t)$ and $\hat{TR}(t)$
$N_{P_Q}(t)$	number of servers to provision at t based on $avgQJobs(t)$
$N_T(t)$	number of servers to terminate at t
$PR(t)$	sum of target play rates of all streams at t
$S(t)$	set of transcoding servers at t
$S_p(t)$	set of newly provisioned servers at t
$S_c(t)$	servers close to completion of renting period at t
$S_t(t)$	servers selected for termination at t
$TR(t)$	total transcoding rate of all servers at t
$\hat{TR}(t)$	predicted total transcoding rate of all servers at t
$RT(s,t)$	remaining time of server s at t with respect to renting hour
$V(t)$	set of video streams at t
B_L	buffer size lower threshold in megabytes
$B_S(t)$	size of the output video buffer in megabytes
B_U	buffer size upper threshold in megabytes
C_T	over allocation count threshold
$jobCompletion$	job completion delay
$MAXQL_{UT}$	maximum queue length upper threshold
N_B	number of servers to use as base capacity
RT_L	remaining time lower threshold
RT_U	remaining time upper threshold
$startUp$	server startup delay
$calcN_P()$	calculate the value of $N_P(t)$
$calcN_T()$	calculate the value of $N_T(t)$
$calcQN_P()$	calculate the value of $N_{P_Q}(t)$ based on queue length
$calRT(s,t)$	calculate the value of $RT(s,t)$
$delay(d)$	delay for duration d
$getPR()$	get $PR(t)$ from video merger
$getTR(s)$	get transcoding rate of server s
$get\hat{TR}()$	get $\hat{TR}(t)$ from load predictor
$provision(n)$	provision n servers
$select(n)$	select n servers for termination
$sort(S)$	sort servers S on remaining time
$terminate(S)$	terminate servers S

(line 12), it chooses to provision one or more servers (line 13). In this case, the number of VMs to provision $N_{P_Q}(t)$ is calculated as follows

$$N_{P_Q}(t) = \left\lceil \frac{avgQJobs(t)}{MAXQL_{UT}} \right\rceil \tag{2}$$

The algorithm then provisions $N_P(t) + N_{P_Q}(t)$ VMs, which are added to the cluster of transcoding servers (lines 20–21). To minimize potential oscillations due to unnecessary VM allocations, the algorithm adds a delay for the VM startup time (line 22). Furthermore, it ensures that the total number of VMs $|S(t)|$ does not exceed the total number of video streams $|V(t)|$. The algorithm adjusts the number of VMs to provision $N_P(t)$ if $|S(t)| + N_P(t)$ exceeds $|V(t)|$ (lines 16–18). This is desirable because the transcoding rate of a video on a single VM is usually higher than the required play rate.

Algorithm 1 VM allocation algorithm

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

3.2 VM Deallocation Algorithm

The VM deallocation algorithm is presented in Algorithm 2. The main objective of the algorithm is to minimize the 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 $B_S(t)$. After obtaining the target play rate $PR(t)$ and the predicted total transcoding rate $\hat{TR}(t)$ (lines 2–7), the algorithm makes a comparison. If $\hat{TR}(t)$ exceeds $PR(t)$ while the buffer size $B_S(t)$ exceeds its upper threshold B_U (line 8), 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.

Algorithm 2 VM deallocation algorithm

```
 1: while true do
 2:   PR(t) := getPR()
 3:   TR(t) := 0
 4:   for sϵS(t) do
 5:     TR(t) := TR(t) + getTR(s)
 6:   end for
 7:   T̂R(t) := getT̂R(TR(t))
 8:   if T̂R(t) > PR(t) ∧ B_S(t) > B_U ∧ count_over(t) > C_T then
 9:     for sϵS(t) do
10:       RT(s,t) := calRT(s,t)
11:     end for
12:     S_c(t) := {∀sϵS(t)|RT(s,t) < RT_U ∧ RT(s,t) > RT_L}
13:     if |S_c(t)| ≥ 1 then
14:       N_T(t) := calcN_T()
15:       N_T(t) := min(N_T(t), |S_c(t)|)
16:       if N_T(t) ≥ 1 then
17:         sort(S_c(t))
18:         S_t(t) := select(N_T(t))
19:         S(t) := S(t) \ S_t(t)
20:         delay(jobCompletion)
21:         terminate(S_t(t))
22:       end if
23:     end if
24:   end if
25: end while
```

In the next step, the algorithm calculates the remaining time of each transcoding server $RT(s,t)$ with respect to the completion of the renting period (lines 9–11). 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 (line 12). The objective is to terminate only those servers whose renting period is close to the completion, while excluding any servers that are extremely close to the completion of their renting period. Therefore, it is not practically feasible to complete all running and pending jobs on them before the start of the next renting period. If the algorithm finds at least one such server $S_c(t)$ (line 13), it calculates the number of servers to terminate $N_T(t)$ as

$$N_T(t) = \left\lceil \frac{\hat{TR}(t) - PR(t)}{\frac{TR(t)}{|S(t)|}} \right\rceil - N_B \tag{3}$$

Then, it sorts the transcoding servers in $S_c(t)$ on the basis of their remaining time (line 17), and selects the servers with the lowest remaining time for termination (line 18). 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 [5], [6]. 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 (line 20). Finally, the selected servers are terminated and removed from the cluster (line 21).

4 Computation and Storage Trade-off Strategy

In this section, we present the proposed computation and storage trade-off strategy. For the sake of clarity, we provide a summary of the notations in Table 2. The proposed cost and popularity score based strategy estimates the computation cost, the storage cost, and the video popularity of individual transcoded videos and then uses this information to make decisions on how long a video should be stored or how frequently it should be re-transcoded from a given source video. In an on-demand video streaming service, the source videos are usually high quality videos that comprise the primary datasets. Therefore, irrespective of their computation and storage costs, they are never deleted from the video repository. The transcoded videos, on the other hand, are the derived datasets that can be regenerated on-demand from their source videos. Therefore, they should only be stored in the video repository when it is cost-efficient to store them. Thus, the proposed strategy is only applicable to the transcoded videos. In other words, since the computation and the storage costs of the source videos are not relevant, the proposed

Table 2: Summary of concepts and their notation for trade-off strategy

Notation	Description
τ	set of transcoded videos
τ_i	i^{th} transcoded video
NS_{τ_i}	new cost and popularity score of τ_i
RC_T	renting cost of a transcoding server per renting hour
S_{τ_i}	total cumulative cost and popularity score of τ_i
SC_{τ_i}	storage cost of τ_i per time unit
SC_m	monthly storage cost per 1 gigabytes
SD_{τ_i}	storage duration for transcoded video τ_i
TC_{τ_i}	transcoding cost of τ_i
TT_{τ_i}	transcoding time of τ_i
$VSmb_{\tau_i}$	transcoded video τ_i size in megabytes
DC	decrement in S_{τ_i}
GB_{mb}	megabytes to gigabytes conversion factor
H_{sec}	hour to seconds conversion factor
RP_S	month to desired time unit conversion factor
$calcNS(\tau_i)$	calculate NS_{τ_i}
$calcSC(\tau_i)$	calculate SC_{τ_i}
$calcTC(\tau_i)$	calculate TC_{τ_i}
$delay(SD_{\tau_i})$	delay for SD_{τ_i}
$getS(\tau_i)$	get S_{τ_i}
$getSC(\tau_i)$	get SC_{τ_i}
$getTC(\tau_i)$	get TC_{τ_i}
$removeVideo(\tau_i)$	remove video τ_i

strategy is based only on the computation and storage costs of the transcoded videos.

In cloud computing, the computation cost is essentially the cost of using VMs, which is usually calculated on an hourly basis. The storage cost, on the other hand, is often computed on a monthly basis. The computation cost of a transcoded video depends on its transcoding time and on how often the video is re-transcoded. Thus, if a video is frequently re-transcoded, the computation cost would increase rapidly. On the other hand, the storage cost of a transcoded video depends on the length of the storage duration and the video size on disk. Therefore, it increases gradually with the passage of time. The longer the duration, the higher the cost. Thus, our proposed strategy estimates an equilibrium point on the time axis where the computation cost and the storage cost of a transcoded video become equal. This estimated equilibrium point indicates the minimum duration for which the video should be stored in the video repository. Figure 2 shows that if a video

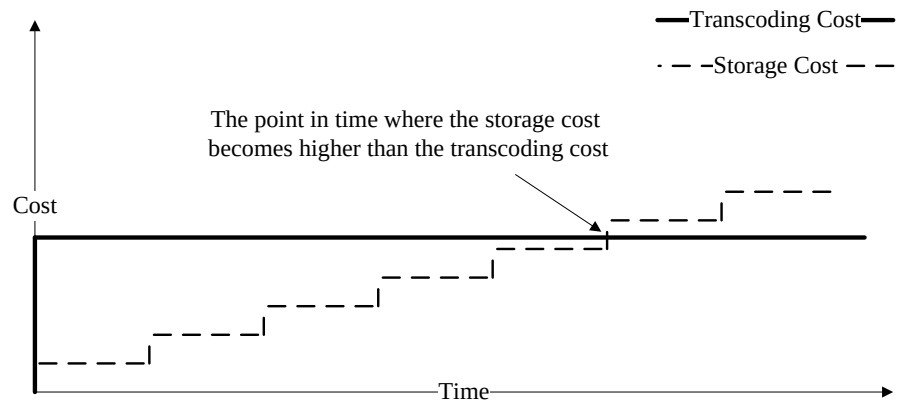

Figure 2: The estimated equilibrium point between the storage cost and the transcoding cost of a transcoded video

is transcoded once and stored in the video repository, then initially the computation cost is higher than the storage cost. However, with the passage of time, the storage cost continues to increase until it becomes equal to the computation cost and then it grows even further unless the video is removed from the video repository. Thus, if the video is deleted before its estimated equilibrium point and then it is subsequently requested, the computation cost will increase due to unnecessary re-transcoding. Likewise, if the video is stored beyond its estimated equilibrium point and then it does not receive a subsequent request, the storage cost will increase unnecessarily.

In an on-demand video streaming service, each transcoded video may be requested and viewed a number of times. Frequently viewed, popular videos get a lot of requests. While, sporadically viewed, less popular videos get only a few requests. For cost-efficient storage, it is essential to use an estimate of the popularity of the individual transcoded videos. This information can then be used to determine the exact duration for which a video should be stored in the video repository. Therefore, the proposed strategy accounts for the popularity of individual transcoded videos. It uses the estimated computation cost, the estimated storage cost, and the video popularity information to calculate a cost and popularity score S_{τ_i} for each transcoded video τ_i. The higher the score the longer the video is stored in the video repository. Thus, with the incorporation of the video cost and popularity score, it becomes justifiable to store popular transcoded videos beyond their estimated equilibrium point. In other words, it differentiates popular videos that should be stored for a longer duration.

In our proposed strategy, the storage cost SC_{τ_i} of a transcoded video τ_i is calculated as

$$SC_{\tau_i} = \frac{VSmb_{\tau_i}}{GB_{mb}} \cdot \frac{SC_m}{RP_S} \cdot SD_{\tau_i} \tag{4}$$

where $VSmb_{\tau_i}$ is the size of the transcoded video τ_i in megabytes, GB_{mb} is the

megabytes to gigabytes conversion factor, SC_m is the monthly storage cost per 1 gigabytes of storage, RP_S is the month to desired time unit conversion factor, and SD_{τ_i} is the length of the storage duration for the transcoded video τ_i. Similarly, the transcoding cost TC_{τ_i} of a transcoded video τ_i is calculated as

$$TC_{\tau_i} = TT_{\tau_i} \cdot \frac{RC_T}{H_{sec}} \tag{5}$$

where TT_{τ_i} is the transcoding time of τ_i, RC_T is the renting cost of a transcoding server per renting hour, and H_{sec} is the hour to seconds conversion factor, which is used to normalize the computation cost to a per second basis.

Whenever a new request for a transcoded video τ_i arrives at the streaming server, the video cost and popularity score S_{τ_i} is updated to reflect the new costs and the new popularity information. The new cost and popularity score NS_{τ_i} represents the estimated equilibrium point where the computation cost and the storage cost of τ_i become equal. Therefore, it indicates the minimum duration for which the video should be stored. The new cost and popularity score NS_{τ_i} of a video τ_i is calculated as the ratio of the transcoding cost TC_{τ_i} and the storage cost SC_{τ_i}

$$NS_{\tau_i} = \frac{TC_{\tau_i}}{SC_{\tau_i}} \tag{6}$$

Finally, the total cost and popularity score S_{τ_i} of a video τ_i is calculated by accumulating the new cost and popularity score NS_{τ_i} of the said video over time. That is, for each new request of a transcoded video τ_i, we obtain the previous value of the total cost and popularity score S_{τ_i} of the transcoded video, calculate NS_{τ_i}, and then add them together to produce the new value of the S_{τ_i}. Moreover, the total cost and popularity score of a video that was not stored previously is set to NS_{τ_i}. The total cost and popularity score S_{τ_i} determines the exact duration for which a video τ_i should be stored. The pseudocode for score calculation is presented in Algorithm 3.

Algorithm 3 Calculation of cost and popularity score

while true do
  if τ_i is requested **then**
    $SC_{\tau_i} := calcSC(\tau_i)$
    $TC_{\tau_i} := calcTC(\tau_i)$
    $NS_{\tau_i} := calcNS(\tau_i)$
    $S_{\tau_i} := \begin{cases} S_{\tau_i} + NS_{\tau_i}, & \text{if } \tau_i \text{ was stored previously} \\ NS_{\tau_i}, & \text{otherwise} \end{cases}$
  end if
end while

Each transcoded video τ_i should be stored in the video repository for as long as it is cost-efficient to store it. However, when a video loses its popularity, it

should be subsequently deleted to avoid unnecessary storage cost. Therefore, on certain time intervals, the proposed strategy performs the following steps for each transcoded video τ_i. It obtains the storage cost SC_{τ_i}, the cost and popularity score S_{τ_i}, and the transcoding cost TC_{τ_i}. Then, it multiplies S_{τ_i} and TC_{τ_i} and compares it with SC_{τ_i} as follows

$$SC_{\tau_i} > TC_{\tau_i} \cdot S_{\tau_i} \tag{7}$$

If the inequality holds, it implies that it is cost-efficient to delete the transcoded video. Therefore, the video is removed from the video repository. However, if the inequality does not hold, it indicates that it is not cost-efficient to delete the video. Therefore, the video is not removed. Moreover, the cost and popularity score S_{τ_i} is decremented in accordance with the length of the time interval to reflect the passage of time. In this way, when a popular video loses its popularity, it starts losing its cost and popularity score as well until it is removed from the video repository or it gets some new requests to regain its popularity. The pseudocode to decrement cost and popularity score S_{τ_i} and to remove a video is given as Algorithm 4.

Algorithm 4 Decrementing score and removing a video

```
1:  while true do
2:    for τ_i ϵ τ do
3:      SC_τi := getSC(τ_i)
4:      TC_τi := getTC(τ_i)
5:      S_τi := getS(τ_i)
6:      if SC_τi > TC_τi · S_τi then
7:        removeVideo(τ_i)
8:      else
9:        S_τi := S_τi − DC
10:     end if
11:   end for
12:   delay(SD_τi)
13: end while
```

5 Experimental Evaluation

Software simulations are often used to test and evaluate new approaches and strategies involving complex environments [10], [8]. For our proposed resource allocation algorithms and trade-off strategy, we have developed a discrete-event simulation in the Python programming language. It is based on the SimPy simulation framework [25]. Also, for a comparison of the results with the alternative existing approaches, we have developed discrete-event simulations for two intuitive computation and storage trade-off strategies, which are the store all strategy

and the usage based strategy [36]. The store all strategy stores all transcoded videos irrespective of their costs and popularity. While the usage based strategy stores only popular videos and removes the rest. That is, it does not account for the computation and storage costs.

5.1 Experimental Design and Setup

For the computation and storage costs, we used the Amazon EC2 and the Amazon S3[2] cost models. The computation cost in Amazon EC2 is based on an hourly charge model. Whereas, the storage cost of Amazon S3 is based on a monthly charge model. In our experiment, we used only small instances. As of writing of this paper, the cost of a small instance in Amazon EC2 is $\$0.06$ per hour. Whereas, the cost of storage space in Amazon S3 is based on a nonlinear cost model as shown in Table 3.

The experiment used HD, SD (Standard-Definition), and mobile video streams. Since SD videos currently have a higher demand than the HD and mobile videos, we considered 20% HD, 30% mobile, and 50% SD video streams. The GOP size for different types of videos was different. For HD videos, the average size of a video segment was 75 frames with a standard deviation of 7 frames. Likewise, for SD and mobile videos, the average size of a segment was 250 frames with a standard deviation of 20 frames.

In an on-demand video transcoding service, a source video is usually transcoded in many different formats. Therefore, we assumed that a source video can be transcoded into a maximum of 30 different formats. Likewise, since in an on-demand video streaming service, the number of source videos always continue to grow, we used a continuously increasing number of source videos in our experiment. However, since the number of the newly uploaded source videos is usually only a small fraction of the total number of downloaded videos, the video upload rate in our experiment was assumed to be 1% of the total number of the video download requests. The desired time unit for storage, as used in the month to desired time unit conversion factor RP_S, was assumed to be one day. Therefore, RP_S was 30. Moreover, the minimum storage duration for a transcoded video SD_{τ_i} was also assumed to be one day.

The objective of the experiment was to evaluate the proposed algorithms and trade-off strategy for a realistic load pattern. Therefore, it used a real load pattern, which constitutes real video access data from Bambuser AB[3]. The load pattern consists of approximately 40 days of real video access data. The total number of frames in a video stream was in the range of 18000 to 90000, which represents an approximate play time of 10 to 50 minutes with the frame rate of 30 frames per second.

[2]http://aws.amazon.com/s3/

[3]http://bambuser.com/

Table 3: Amazon S3 storage pricing

	Standard Storage
First 1 TB per month	$ 0.095 per GB
Next 49 TB per month	$ 0.080 per GB
Next 450 TB per month	$ 0.070 per GB
Next 500 TB per month	$ 0.065 per GB
Next 4000 TB per month	$ 0.060 per GB
Over 5000 TB per month	$ 0.055 per GB

5.2 Results and Analysis

In this section, we compare the experimental results of the proposed strategy with that of the store all strategy and the usage based strategy. Each result in Figure 3 to Figure 5 consists of seven different plots, which are number of user requests, number of transcoding servers, transcoding cost, storage cost, storage size, number of source videos, and number of transcoded videos. The number of user requests plot represents the load pattern of the video access data. In other words, it is the user load on the streaming server. Due to data confidentiality, the exact volume of the load can not be revealed. Therefore, we have omitted the scale of this plot from all the results. The number of transcoding servers plot shows the total number of transcoding servers being used at a particular time. The transcoding cost plot represents the total computation cost of all transcoded videos in US dollars. Similarly, the storage cost plot shows the storage cost in US dollars of all transcoded videos, which are stored in the video repository. The storage size plot represents the total size of the cloud storage used to store the transcoded videos. The number of source videos plot shows the total number of source videos in the video repository. Likewise, the number of transcoded videos is the total number of transcoded videos in the video repository. The results are also summarized in Table 4.

Figure 3 presents the simulation results of the store all strategy. The results span over a period of 40 days. At the end of the simulation, the total number of transcoded videos in the video repository was 206590, while the total number of source videos was 20902. The average number of transcoding servers was 102, the total transcoding cost was $\$4458.42$, the total storage cost was $\$4911.36$, and the total storage size was 42.16 terabytes. Since the store all strategy stores all transcoded videos irrespective of their computation and storage costs, the storage cost was very high due to a large number of transcoded videos stored in the video repository. Therefore, the results indicate that the store all strategy is not cost-efficient.

Figure 4 presents the results of the usage based strategy. At the end of the simulation, the total number of transcoded videos in the video repository was 190734 for the same number of source videos as used in the store all strategy.

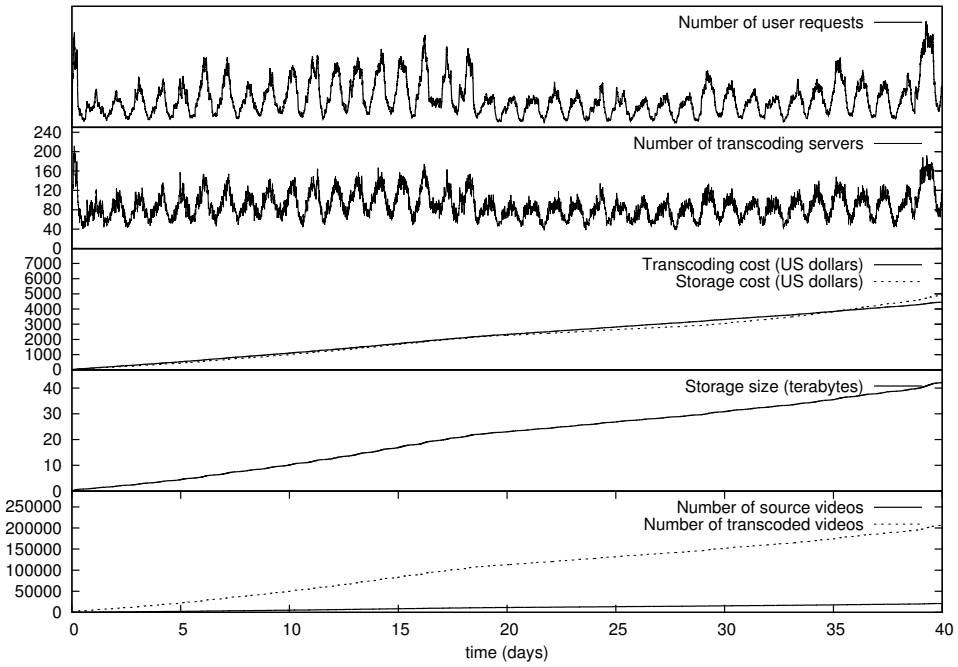

Figure 3: Store all strategy

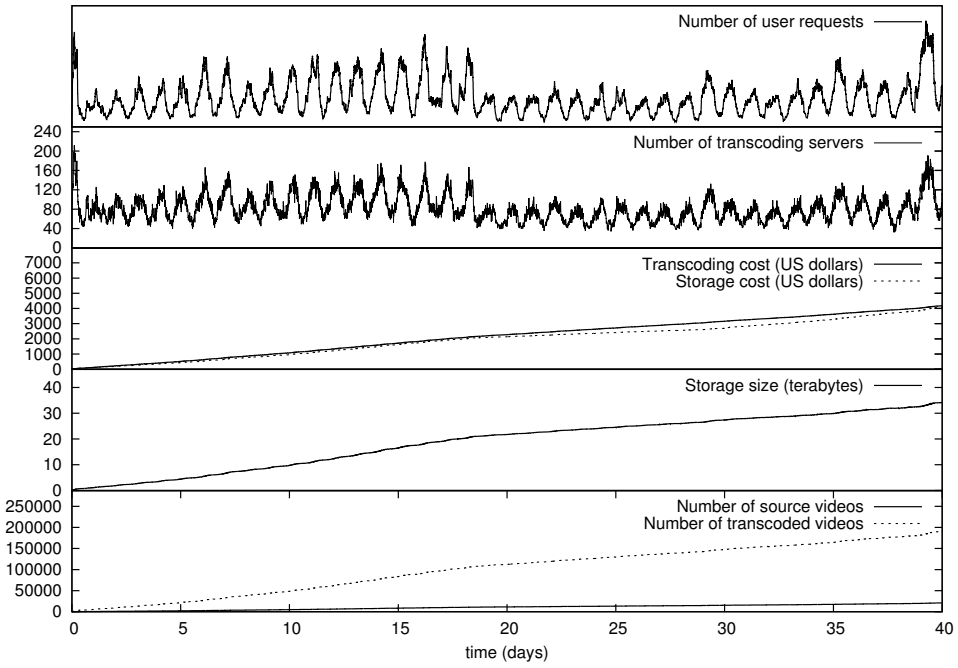

Figure 4: Usage based strategy

Table 4: Summary of results

Strategy	Avg. servers	Transcoding cost	Storage cost	Total cost
Store all	102	$4458.42	$4911.36	$9369.78
Usage based	94	$4179.12	$4090.56	$8269.68
Score based	107	$4893.60	$2307.84	$7201.44

The average number of transcoding servers was 94, the total transcoding cost was $4179.12, the total storage cost was $4090.56, and the total storage size was 34.19 terabytes. Since the usage based strategy stores only popular videos, the storage cost of the usage based strategy was slightly less than that of the store all strategy. Therefore, the results indicate that the usage based strategy is cost-efficient when compared to the store all strategy. However, since it does not account for the computation and the storage costs, it may remove some videos that have a high transcoding cost.

Figure 5 presents the results of the proposed score based strategy. At the end of the simulation, the total number of transcoded videos in the video repository was 64392 for the same number of source videos as used in the store all strategy and the usage based strategy. The average number of transcoding servers was 107, the total transcoding cost was $4893.60, the total storage cost was $2307.84, and the total storage size was 14.93 terabytes. Since the proposed strategy accounts for the computation cost, the storage cost, and the video popularity information, the storage cost was much less than that of the store all strategy and the usage based strategy.

Figure 6 presents a comparison of the total costs, which consists of the computation cost and the storage cost. The results show that the store all strategy has the highest total cost. The usage based strategy has slightly less total cost than the store all strategy. Moreover, the proposed storage has the least total cost among all the three strategies. Therefore, the results indicate that the proposed strategy is cost-efficient when compared to the store all and the usage based strategies.

6 Related Work

Distributed video transcoding with video segmentation was proposed in [19] and [23]. Jokhio et al. [19] presented bit rate reduction video transcoding using multiple processing units, while [23] analyzed different video segmentation methods to perform spatial resolution reduction video transcoding. Huang et al. [17] presented a cloud-based video proxy to deliver transcoded videos for streaming. The main contribution of their work is a multilevel transcoding parallelization framework. Li et al. [24] proposed a cloud transcoder, which uses a compute cloud as an intermediate platform to provide transcoding service. Shin and Koh [30]

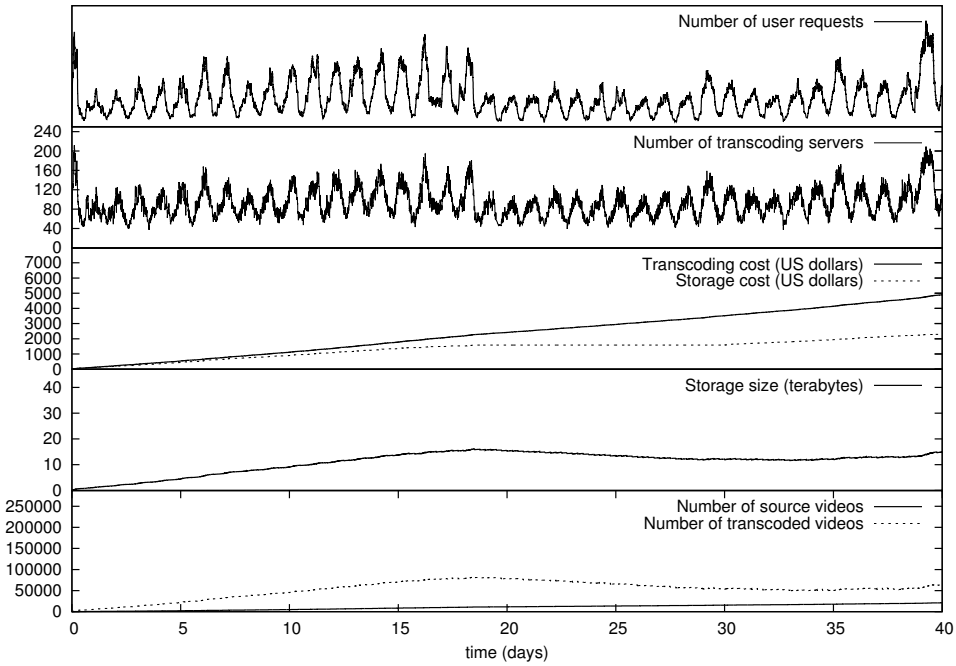

Figure 5: Proposed score based strategy

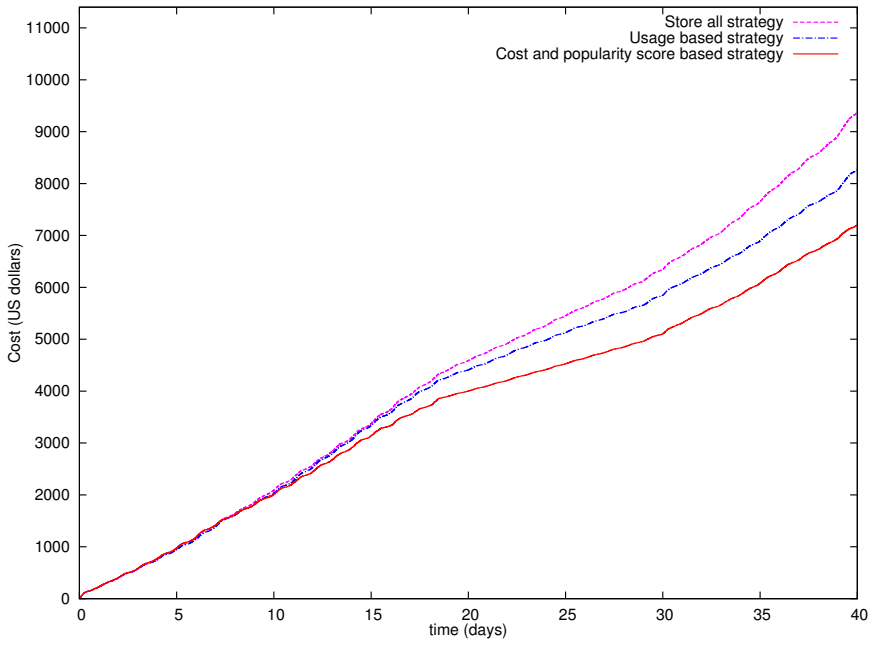

Figure 6: Cost comparison

presented a hybrid scheme to determine an optimal threshold between the static and dynamic transcoding. Ashraf et al. [8] proposed an admission control and job scheduling approach for video transcoding in the cloud. None of these papers addressed the VM allocation problem for video transcoding in cloud computing.

6.1 VM Allocation Approaches

The existing works on dynamic VM allocation can be classified into two main categories: Plan-based approaches and control theoretic approaches. The 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. [3], while TwoSpot [34], Hu et al. [16], Chieu et al. [11], Iqbal et al. [18] and Han et al. [15] use a performance dynamics model. Similarly, Dutreilh et al. [13], Pan et al. [27], Patikirikorala et al. [28], and Roy et al. [29] 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 performance and VM allocation metrics for video transcoding service, such as video play rate and server transcoding rate. Moreover, it is cost-efficient as it uses a reduced number of VMs for a large number of video streams, it provides proactive VM allocation under soft real-time constraints, and it does not depend upon performance and dynamics of the underlying system. A more detailed analysis of the VM allocation approaches can be found in [22].

6.2 Computation and Storage Trade-off Strategies

There are currently only a few works in the area of computation and storage trade-off analysis for cost-efficient usage of cloud resources. One of the earlier attempts include Adams et al. [1], who highlighted some of the important issues and factors involved in constructing a cost-benefit model, which can be used to analyze the trade-offs between computation and storage. However, they did not propose a strategy to find the right balance between computation and storage resources. Deelman et al. [12] studied cost and performance trade-offs for an astronomy application using Amazon EC2 and Amazon S3 cost models. The authors concluded that, based on the likelihood of reuse, storing popular datasets in the cloud can be cost-effective. However, they did not provide a concrete strategy for cost-effective computation and storage of scientific datasets in the cloud.

Nectar system [14] is designed to automate the management of data and computation in a data center. It initially stores all the derived datasets when they are generated. However, when the available disk space falls below a threshold, all obsolete or least-valued datasets are garbage collected to improve resource utilization. Although Nectar provides a computation and storage trade-off strategy, it is

not designed to reduce the total cost of computation and storage in a cloud-based service that uses IaaS resources.

Yuan et al. [36] proposed two strategies for cost-effective storage of scientific datasets in the cloud, which compare the computation cost and the storage cost of the datasets. They also presented a Cost Transitive Tournament Shortest Path (CTT-SP) algorithm to find the best trade-off between the computation and the storage resources. Their strategies are called cost rate based storage strategy [35], [38] and local-optimization based storage strategy [37]. The cost rate based storage strategy compares computation cost rate and storage cost rate to decide storage status of a dataset. Whereas, the local-optimization based storage strategy partitions a data dependency graph (DDG) of datasets into linear segments and applies the CTT-SP algorithm to achieve a localized optimization. In contrast to the cost rate based storage strategy [35], [38], our proposed trade-off strategy estimates an equilibrium point on the time axis where the computation cost and the storage cost of a transcoded video become equal. Moreover, it estimates video popularity of the individual transcoded videos to differentiate popular videos. The DDG-based local-optimization based storage strategy of Yuan et al. [37] is not much relevant for video transcoding because video transcoding does not involve a lot of data dependencies.

Most of the existing computation and storage trade-off strategies described above were originally proposed for scientific datasets. To the best of our limited knowledge, there are currently no existing computation and storage trade-off strategies for video transcoding. The difference of application domain may play a vital role when determining cost-efficiency of the existing strategies. Therefore, some of the existing strategies may have limited efficacy and little cost-efficiency for video transcoding.

7 Conclusion

In this paper, we presented proactive VM 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 VM allocation under soft real-time constraints. For cost-efficiency, we used video segmentation which splits a video stream into smaller segments that can be transcoded independently of one another. This helped us to perform video transcoding of multiple simultaneous streams on a single server.

We also proposed a cost-efficient computation and storage trade-off strategy for video transcoding in the cloud. The proposed strategy estimates the computation cost, the storage cost, and the video popularity information of individual transcoded videos and then uses this information to make decisions on how long a

video should be stored or how frequently it should be re-transcoded from a given source video. The objective is to reduce the total IaaS cost by trading storage for computation, or vice versa.

The proposed approach is demonstrated in a discrete-event simulation and an experimental evaluation involving a realistic load pattern. Also, for the sake of comparison, we simulated two intuitive computation and storage trade-off strategies and compared their results with that of the proposed strategy. The results show that the proposed algorithms provide cost-efficient VM allocation for transcoding a large number of video streams while minimizing oscillations in the number of servers. The results also indicate that our proposed trade-off strategy is more cost-efficient than the two intuitive strategies as it provided a good trade-off between the computation and storage resources.

References

[1] Ian F. Adams, Darrell D. E. Long, Ethan L. Miller, Shankar Pasupathy, and Mark W. Storer. Maximizing efficiency by trading storage for computation. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[2] Mauro Andreolini, Sara Casolari, and Michele Colajanni. Models and framework for supporting runtime decisions in web-based systems. *ACM Trans. Web*, 2(3):17:1–17:43, July 2008.

[3] Danilo Ardagna, Carlo Ghezzi, Barbara Panicucci, and Marco Trubian. Service provisioning on the cloud: Distributed algorithms for joint capacity allocation and admission control. In Elisabetta Di Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2010.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[5] Adnan Ashraf, Benjamin Byholm, Joonas Lehtinen, and Ivan Porres. Feedback control algorithms to deploy and scale multiple web applications per virtual machine. In *Software Engineering and Advanced Applications (SEAA), 38th EUROMICRO Conference on*, pages 431–438, September 2012.

[6] Adnan Ashraf, Benjamin Byholm, and Ivan Porres. CRAMP: Cost-efficient resource allocation for multiple web applications with proactive scaling. *4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 581–586, 2012.

[7] Adnan Ashraf, Benjamin Byholm, and Ivan Porres. A session-based adaptive admission control approach for virtualized application servers. In *Utility and Cloud Computing (UCC), 5th IEEE/ACM International Conference on*, pages 65–72, 2012.

[8] Adnan Ashraf, Fareed Jokhio, Tewodros Deneke, Sebastien Lafond, Ivan Porres, and Johan Lilius. Stream-based admission control and scheduling for video transcoding in cloud computing. *in Cluster, Cloud and Grid Computing (CCGrid), 13th IEEE/ACM International Symposium on*, pages 482–489, 2013.

[9] N. Bjork and C. Christopoulos. Transcoder architectures for video coding. *Consumer Electronics, IEEE Transactions on*, 44(1):88 –98, February 1998.

[10] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Csar A. F. De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[11] T.C. Chieu, A. Mohindra, A.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*, pages 281 –286, October 2009.

[12] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the Montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[13] 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*, pages 410 –417, July 2010.

[14] Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang. Nectar: automatic management of data and computation in datacenters. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[15] Rui Han, Li Guo, M.M. Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 644 –651, May 2012.

[16] Ye Hu, Johnny Wong, Gabriel Iszlai, and Marin Litoiu. Resource provisioning for cloud computing. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '09, pages 101–111, New York, NY, USA, 2009. ACM.

[17] Zixia Huang, Chao Mei, Li Erran Li, and Thomas Woo. CloudStream: Delivering high-quality streaming videos through a cloud-based SVC proxy. In *INFOCOM, 2011 Proceedings IEEE*, pages 201–205, 2011.

[18] Waheed Iqbal, Matthew N. Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871 – 879, 2011.

[19] 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, pages 206 –212, February 2012.

[20] Fareed Jokhio, Adnan Ashraf, Tewodros Deneke, Sebastien Lafond, Ivan Porres, and Johan Lilius. *Developing Cloud Software: Algorithms, Applications, and Tools*, chapter Proactive Virtual Machine Allocation for Video Transcoding in the Cloud, pages 113–143. Turku Centre for Computer Science (TUCS) General Publication Number 60, October 2013.

[21] Fareed Jokhio, Adnan Ashraf, Sebastien Lafond, and Johan Lilius. A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud. In *Software Engineering and Advanced Applications (SEAA), 39th Euromicro Conference on*, pages 365–372, 2013.

[22] Fareed Jokhio, Adnan Ashraf, Sebastien Lafond, Ivan Porres, and Johan Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on*, pages 254–261, 2013.

[23] Fareed Ahmed Jokhio, Tewodros Deneke, Sébastien Lafond, and Johan Lilius. Analysis of video segmentation for spatial resolution reduction video transcoding. In *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposium on*, pages 1–6, December 2011.

[24] Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Zhi-Li Zhang, and Yafei 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.

[25] Norman Matloff. *A Discrete-Event Simulation Course Based on the SimPy Language*. University of California at Davis, 2006.

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

[27] 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*, pages 328 –334, September 2008.

[28] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping 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*, SEAMS '11, pages 218–227, 2011.

[29] 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*, pages 500 –507, July 2011.

[30] Ilhoon Shin and Kern Koh. Hybrid transcoding for QoS adaptive video-on-demand services. *IEEE Trans. on Consum. Electron.*, 50(2):732–736, May 2004.

[31] A. Vetro, C. Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: an overview. *Signal Processing Magazine, IEEE*, 20(2):18 – 29, March 2003.

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

[33] T. Wiegand, G. J. Sullivan, and A. Luthra. Draft ITU-T recommendation and final draft international standard of joint video specification. Technical report, 2003.

[34] Andreas Wolke and Gerhard Meixner. TwoSpot: A cloud platform for scaling out web applications dynamically. In Elisabetta Di Nitto and Ramin Yahyapour, editors, *Towards a Service-Based Internet*, volume 6481 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin / Heidelberg, 2010.

[35] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, 2010.

[36] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. Computation and storage trade-off for cost-effectively storing scientific datasets in the cloud. In Borko Furht and Armando Escalante, editors, *Handbook of Data Intensive Computing*, pages 129–153. Springer New York, 2011.

[37] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A local-optimisation based strategy for cost-effective datasets storage of scientific applications in the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 179–186, 2011.

[38] Dong Yuan, Yun Yang, Xiao Liu, Gaofeng Zhang, and Jinjun Chen. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. *Concurrency and Computation: Practice and Experience*, 24(9):956–976, 2012.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-3001-1
ISSN 1239-1891