RESOURCE ALLOCATION BASED ON RATES, LOAD AND THROUGHPUT FOR VIDEO TRANSCODING IN A CLOUD

Fareed Jokhio*, Umair Ali Khan*, Intesab Hussain Sadhayo*, Anees Ahmed Soomro*, Azhar Uddin Jokhio**

ABSTRACT

This paper presents resource allocation algorithms for video transcoding service on a cloud. The main objective of the proposed algorithms is to allocate and de-allocate Virtual Machines (VMs) horizontally in a cluster of video transcoding servers. For cost-efficiency and better utilization of resources, video segmentation at *group of pictures* level is used. With video segmentation, a video is split into smaller segments that can be sent for transcoding on any transcoding server. To demonstrate the efficiency of the proposed algorithms, a discrete-event simulation is used. The proposed algorithms are also compared with the existing video transcoding algorithms in a cloud computing environment. The existing VM allocation algorithms are based on accumulative play rate and transcoding rate, while our improved proactive VM allocation algorithms also take into account the overall computation load and system throughput. The results indicate that the proposed algorithms are more cost-efficient than the existing algorithms.

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

1. INTRODUCTION

The cloud computing provides computing and storage services over the Internet. These services may include computer processing power, data storage space, software applications and networks. The provisioning of resources while needed and releasing the resources while not required is possible in a cloud computing environment. There exist a number of cloud computing service models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Data as a Service(DaaS) [15].

In IaaS model, the service provider is responsible for the services including hosting and running of the applications, and maintenance of the equipment. In PaaS model, the service provider provides computational resources through an operating system. PaaS provides an environment to build and deploy applications. In SaaS model, the service provider provides implementation of a specific software application. The customer does not have to worry about getting a licence or getting acquinted with any other technical details. In DaaS model, the data is provided to the users on demand regardless of the physical location. The computing resources offered by IaaS clouds, such as computing units or virtual machines (VMs) [14], can be used to perform very intensive computations such as video transcoding [16] of several video streams simultaneously.

Video trasnscoding is a process in which a compressed video is converted from one format into another. It is needed because a video stored at the server side in a certain format might not be compatible with a device at the client side. A video usually requires a huge disk space. Therefore, it is always stored in compressed format such as MPEG-4 [17] and H.264 [18]. A video transcoding operation may change video format, frame resolution, frame rate, bit rate, or any combination of these.

The video transcoding requires intensive computation and it is not always possible to perform it on a client device such as a mobile phone in a real-time. The transcoding can be performed at server side or in a cloud computing environment and the transcoded video can be stored to avoid repeated transcoding operations. Cloud computing is getting popular due to its computing environments, software services and it can provide a VM with theoretically infinite capabilities. Therefore, it is suitable for video transcoding [3]. However, the number of video transcoding requests may vary with time. Hence, determining the number of resources required to perform computation in a cloud computing environment is still an open research problem.

Real-time transcoding of video streams requires a number of video frames according to the play rate of the stream to avoid violation of the client-side quality of service (QoS) requirements [6]. It is often possible to segment a video into small parts and perform the transcoding operation [12]. It helps to have better utilization of computing resources.

In this paper, we present resource allocation algorithms for video transcoding in a cloud computing environment. Load prediction with a load tracker and a load predictor [2] using linear regression model is used to predict the

* Department of Computer Systems Engineering, Quaid-e-Awam University of Engineering, Science & Technology, Nawabshah, Pakistan. Email: {fajokhio, umair.khan, intesab, anees.soomro}@quest.edu.pk

** Department of Electronics Engineering, MUET Jamshoro, Pakistan.

work load. Video segmentation is also used to share the computing resources. The proposed approach is evaluated in different experiments using discrete event simulation. The experiments show that the proposed algorithms are more cost-efficient as compared to the existing resource allocation algorithms.

We proceed as follows. In Section 2, we present the system architecture. Section 3 describes dynamic resource allocation algorithms. Section 4 presents our simulation results and Section 5 presents conclusion.

2. SYSTEM ARCHITECTURE

The system architecture of video transcoding in a cloud is shown in Figure 1. Users send requests to the streaming server which in turn sends them responses. Once the streaming server receives a request, it first checks weather the requested video is available in the video repository in the required format. If it finds the video in the repository, it sends it to the user. However, if the video is not available in the required format, it is sent for transcoding. As a video is sent for transcoding, it is segmented into small segments by the video splitter. A video consists of different types of frames having inter-dependencies among them. Therefore, segmentation is possible at key frames or *I* frames only. Other types of frames which require reference frames for decoding are *P* frames and *B* frames. The key frames are independent and do not require any other reference frame for decoding. Therefore, a video consists of *I* frames after a certain interval of frames. This is termed as a *Group Of Pictures (GOP)*. Distributed video transcoding with video segmentation at GOP level is used in [12][9]. These papers provide the implementation of a video transcoder using the Message Passing Interface (MPI) model and analyze various possible segmentation methods. In addition to the above works, several other works on video transcoding in the cloud computing [5][10][11] have adopted the video segmentation at GOP level. In our proposed approach, the video segmentation is also performed at GOP level. Once a video is segmented, it is sent to the *load balancer* for load balancing and scheduling.

The *load balancer* uses the shortest queue length policy for scheduling. The actual transcoding is performed at *transcoding server* and then the video segments are sent to *video merger* for merging. Finally, the streaming server sends the response to the users. This system architecture is similar to the techniques proposed in [5][10][11].

The resource allocation algorithms which are discussed in section 3 are implemented by the *master controller*. Based on the decisions of the *master controller*, the *cloud provisioner* provisions and terminates the VMs. There also exists a *load predictor* which performs the load prediction. It uses a two-step approach with a load tracker and a load predictor which was proposed by Andreolini and Casolari [2] to predict future load behavior under real-time constraints.

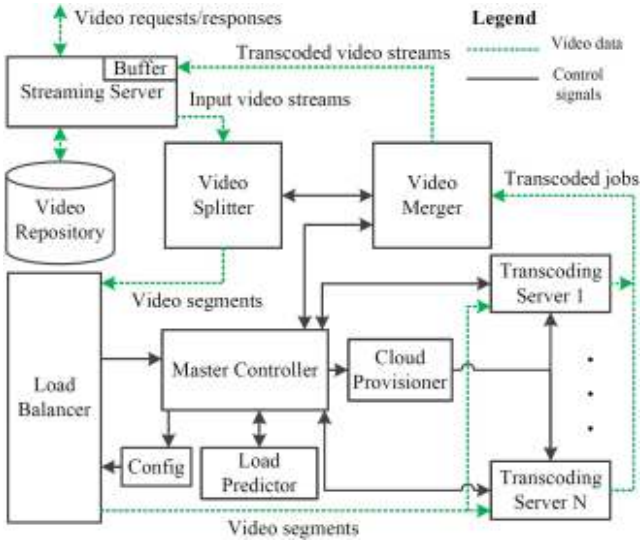

Figure 1: The system architecture

3. PROPOSED VM ALLOCATION AND DE-ALLOCATION TECHNIQUES

In this section, the proposed VM allocation and de-allocation algorithms are discussed which are based on accumulative play rate, accumulative transcoding rate, computation load and system throughput.

In [11], similar algorithms are proposed based on accumulative play rate and transcoding rate. However, resource allocation based on accumulative play rate and transcoding rate may have high provisioning cost. Additionally, it does not take into account the current computation load and the system throughput. The required transcoding rate could be lower due to improper load balancing. Therefore, provisioning more number of servers will result in a higher cost. In contrast, our proposed algorithms are based on accumulative play rate, transcoding rate, computation load and system throughput. The concepts and notations used in the algorithms are given in Table 1.

We apply a two-step load prediction approach [1][4][11] to predict the future workload. Our proposed technique tracks the current and past workload of the system and then predicts the future workload. As in [11], the system maintains a fixed number of VMs, termed as base capacity N_B. Therefore, our approach also maintains a fixed minimum number of servers.

The master controller collects the play rate $PR(t)$ and transcoding rate $TR(t)$ after some time interval. It performs the prediction and also calculates the predicted

transcoding rate $\hat{TR}(t)$. Since the provisioning of a VM requires some time [4], the algorithms avoid oscillations in the number of VMs [19] by delaying the new VMs allocation until the previous VM allocation operation is performed [8].

Notation	Description
$avgQJ(t)$	average queue length of all servers at t
$count_o(t)$	over allocation count at t
$N_P(t)$	number of servers to provision at t
$N_{P_Q}(t)$	number of servers to provision at t based on Queue length
$N_T(t)$	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 finish renting period at t
req_i	video transcoding $request_i$
C_j	$Class_j$ of video transcoding requests
$NSC_k(t)$	number of servers for $Class_k$ stream at time t
$NStrmC_k(t)$	number of streams in $Class_k$ at time t
$PRC_k(t)$	play rate of streams in $Class_k$ at time t
$avgTRC_k(t)$	average transcoding rate of $Class_k$ at time t
$NAS(t)$	total number of servers at time t
$S_t(t)$	servers selected for termination at t
$TR(t)$	total transcoding rate of all servers at t
$\hat{TR}(t)$	predicted transcoding rate of all servers at t
$RT(s,t)$	remaining renting time of server s 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
$jobCD$	job completion delay
MQL_{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 $N_{P_Q}(t)$
$calRT(s,t)$	calculate the value of $RT(s,t)$
$calcNAS()$	calculate the value of $NAS(t)$
$delay(d)$	delay for duration d
$getPR()$	get $PR(t)$ from video merger
$getTR(s)$	get transcoding rate of server s
$get\hat{T}R()$	get $\hat{TR}(t)$ from load predictor
$prov(n)$	provision n servers
$select(n)$	Select n servers for termination
$sort(S)$	sort servers S on remaining time
$term(S)$	terminate servers S

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

3.1 VM Allocation Algorithm

The VM allocation mechanism is described as Algorithm 1. In the algorithm, the play rate $PR(t)$ and total transcoding rate $TR(t)$ is computed. Then the predicted transcoding rate $\hat{TR}(t)$ of all streams is computed by the load predictor. It further checks the buffer size $B_S(t)$ where the transcoded video segments are stored.

Algorithm 1 VM allocation algorithm

1: **While true do**
2: $N_P(t) := 0, N_{P_Q}(t) := 0$
3: $PR(t) := getPR()$
4: $TR(t) := 0$
5:
6: $NAS(t) := calcNAS()$
for $s \in S(t)$ **do**
7: $TR(t) := TR(t) + getTR(s)$
8: **end for**
9: $\hat{TR}(t) := get\hat{TR}(TR(t))$
10: **if** $\hat{TR}(t) < PR(t) \wedge B_S(t) < B_L$ **then**
11: $N_P(t) := calcN_P()$
12: **end if**
13: **if** $avgQJ(t) > MQL_{UT}$ **then**
14: $N_{P_Q}(t) := calcQN_P()$
15: **end if**
16: $N_P(t) := N_P(t) + N_{P_Q}(t)$
17: **if** $len(S(t)) \cup N_P(t) > NAS(t)$ **then**
18: $N_P(t) := 0$
19: **end if**
20: **if** $N_P(t) \geq 1$ **then**
21: $S_p(t) := prov(N_P(t))$
22: $S(t) := S(t) \cup S_p(t)$
23: $delay(startUp)$
24: **end if**
25: **end while**

If the buffer size drops below the lower threshold B_L, the algorithms compute the number of VMs to provision $N_P(t)$ by using the following equation.

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

Here $|S(t)|$ is the number of transcoding servers at time t. In addition to the buffer underflow, the resource allocation algorithm takes into account the average queue length of all transcoding servers $avgQJ(t)$. If it exceeds the maximum upper threshold MQL_{UT}, then, a number of servers is provisioned $N_{P_Q}(t)$ by using the following equation.

$$N_{P_Q}(t) = \left\lceil \frac{avgQJ(t)}{MQL_{UT}} \right\rceil \quad (2)$$

The total number of servers to be provisioned is the sum of $N_P(t)$ and $N_{P_Q}(t)$. Since VM provisioning requires some time, to avoid unnecessary oscillations in the number of VMs, the algorithm adds some delay. In the above discussion, the VM allocation decisions are based on the accumulative play rate and the transcoding rate and are same as mentioned in [11]. However, resource allocation based on accumulative play rate and transcoding rate may have high provisioning cost. It does not take into account the current computation load and the system throughput. The required transcoding rate could be lower due to improper load balancing. Therefore, provisioning more number of servers will result in higher cost. Hence, we also take into account the computation load and the system throughput to provide an even more cost-efficient algorithm for resource allocation.

To determine the number of actual servers required, we first divide the transcoding requests into different classes according to the computation requirements as shown in Figure 2. Subsequently, we check how much throughput can a transcoding server provides for these different classes. For example, there could be following classes of streams according to the video transcoding rate.

i. High Definition (HD) streams with average transcoding rate of 60 frames per second (fps).
ii. Standard Definition (SD) streams with average transcoding rate of 132 fps.
iii. Mobile streams with average transcoding rate of 410 fps

When a new transcoding request arrives in the system, some segments of the stream are sent for transcoding.

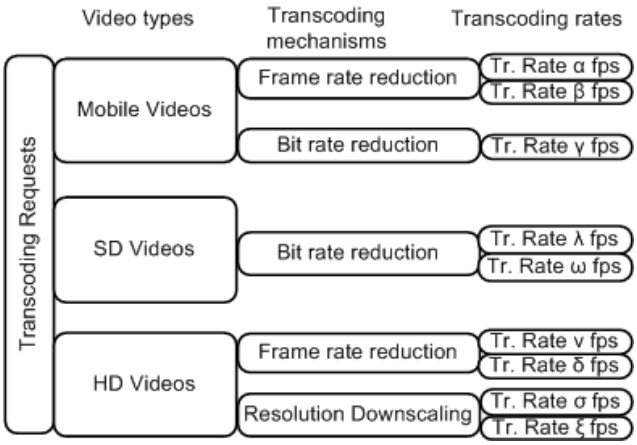

Figure 2: Classification of transcoding requests

Based on the transcoding rate of these segments, the entire video stream is placed in a certain class. The transcoding rate may fluctuate for a video steam. However, if it is within upper and lower bounds of that class, the stream will remain in the same class. In case if the transcoding rate crosses the bounds of that class, it is placed into another class. It means that the runtime class migration of video streams is also possible.

After the classification of video streams, we compute the total number of transcoding requests in each class. For example, there can be 20 requests in the first class, 70 requests in the second class and so on. Since we know the approximate transcoding rate of each class and also the number of streams in each class, we compute the number of servers required.

Let us assume there are n video transcoding requests in the system, denoted by req_1, req_2, ... req_n. These requests can be classified into m different classes C_1, C_2, ..., C_m. The streams in the same class should have similar play rate and similar average transcoding rate. The number of transcoding servers for each class is computed as follows.

$$NSC_k(t) = \left\lceil \frac{NStrmC_k(t) * PRC_k(t)}{avgTRC_k(t)} \right\rceil \quad (3)$$

Finally the total number of actual servers required for all classes of video streams is calculated by the summation of the number of actual servers required for individual classes, as given by equation 4.

$$NAS(t) = \sum_{k=1}^{m} NSC_k(t) \quad (4)$$

We check if the actual number of servers required at this time is less than the number of servers which are already provisioned, then there is need to provision more servers. However, if the number of already provisioned servers is higher than the actual, then there is no need to provision any more servers.

3.2 VM De-allocation Algorithm

The resource de-allocation mechanism is described in Algorithm 2. This algorithm de-allocates virtual machines if the number of provisioned servers $S(t)$ is more than the total number of servers $NAS(t)$ required to perform video transcoding of different classes of video streams and the buffer size $B_S(t)$ exceeds its upper threshold B_U for a predetermined amount of time.

While performing the resource de-allocation, the algorithm computes the remaining renting time of each transcoding server $RT(s,t)$. If the algorithm determines that there are some transocoding servers $S_c(t)$ whose remaining times are in the range of the pre-determined upper threshold of remaining time RT_U and the lower threshold of remaining time RT_L, then it calculates the number of servers to terminate $N_T(t)$ as follows.

$$N_T(t) = \lceil len(S(t)) - NAS(t) \rceil - N_B \quad (5)$$

The $N_T(t)$ servers having lowest remaining renting times are selected for termination. Once a server is selected for termination, it stops accepting new transcoding tasks. As the transcoding time of the jobs is relatively less, the selected servers complete all jobs in queue before termination.

Algorithm 2 VMs de-allocation algorithm

1: **While true do**
2: if $len(S(t)) > NAS(t) \wedge B_S(t) > B_U \wedge count_o(t) > C_T$ then
3: **for** $s \in S(t)$ **do**
4: $RT(s,t) := calRT(s,t)$
5: **end for**
6: $S_c(t) := \{\forall s \in S(t) \mid RT(s,t) < RT_U \wedge RT(s,t) > RT_L\}$
7: **if** $|S_c(t)| \geq 1$ **then**
8: $N_T(t) := calcN_T()$
9: $N_T(t) := min(N_T(t), |S_c(t)|)$
10: **if** $N_T(t) \geq 1$ **then**
11: $sort(S_c(t))$

12: $S_t(t) := select(N_T(t))$
13: $S(t) := S(t) \setminus S_t(t)$
14: $delay(jobCD)$
15: $term(S_t(t))$
16: **end if**
17: **end if**
18: **end if**
19: **end while**

4. SIMULATION RESULTS

In real systems involving complex environment, it is always possible to use software simulations to test and evaluate new algorithms [7]. In this regard, we have used Python programming language to perform simulations. Our discrete-event simulation is based on the SimPy simulation framework [13], which is a library for writing discrete-event simulations in Python.

4.1 Experimental Design and Setup

We use a computation cost model similar to Amazon EC2 in which the renting period of a VM is based on an hourly charge model. In our experiments, we use only small instances and its cost is assumed to be $0.06 per hour. The storage cost is based on a nonlinear model similar to the Amazon S3 cost model in which for the first Tera Byte (TB), the renting cost is $0.095 per Giga Byte (GB) per month. For the next 49 TB, the cost is $0.080 per GB per month. In the similar way, for next 450 TB, the cost is $0.070 per GB per month. For next 500 TB, it is $0.065 per GB, for next 4000 TB, it is $0.060 per GB and for over 5000 TB, the storage cost is $0.055 per GB per month. We use three different types of videos namely HD, SD and mobile video streams. Mostly SD videos are used for streaming over the internet. Therefore, we considered 50% SD videos, 20% HD videos and 30% mobile videos. The transcoding time of a GOP for different types of video varies. Therefore, the average size of a segment for HD video is 68 to 82 frames, while for mobile video and SD videos, it is from 230 frames to 270 frames. We consider two different load patterns in two separate experiments. Load pattern 1 in experiment 1 consists of 30 possible video formats for the transcoded videos. Whereas, load pattern 2 in experiment 2 has 45 possible video formats for the transcoded videos. The upper threshold RT_U of the remaining time is taked as 60 seconds, while the lower threshold RT_L of the remaining time is taked as 12 seconds. The total number of frames in a video stream is taken in the range of 15000 to 18000. The desired play rate for a video stream is often kept fixed, i.e., 30 fps for SD videos and mobile video streams, while 24 fps for HD videos. The transcoding rate for HD videos is assumed to be between 36 fps to 48 fps. For SD and mobile videos it is assumed to be 60 fps to 120 fps.

4.2 Results

In Figures 3, 4, 5, and 6 the plots of user requests show the number of video transcoding requests in the system. The cost plot shows the transcoding as well as the storage cost in US dollars. The videos plots indicate the number of source videos and transcoded videos. The load patterns used in both experiments are synthetic, involving two peaks and minor fluctuations in the load. Each transcoded video is also stored in the video repository. Therefore, for subsequent requests for the same transcoded video, it is not needed to perform the transcoding operation. We performed two experiments as follows.

1) Experiment 1 - 30 possible transcoded formats:

Figure 3 shows results from experiment 1. This experiments uses the algorithms described in [11]. In this experiment, a maximum of 212 transcoding servers are used. The average number of servers is 96 and the minimum number of servers is 2. The two-days renting cost of transcoding servers is 220 dollars and the storage cost is 180 dollars.

Figure 4 shows the results from experiment 1 using the improved algorithms proposed in this paper. In this experiment, a maximum of 114 transcoding servers are used. The average number of servers is 42 and the minimum number of servers is 2. The two-days renting cost of transcoding servers is 103 dollars and the storage cost is 180 dollars. Therefore, the results indicate that the storage cost for old and new algorithms is same as expected. Whereas, the transcoded cost with the improved algorithms is less than half of the old algorithms for two days of simulation results. Hence, it is evident that the existing algorithms take an aggressive decision while provisioning the servers and result in higher trasncoding cost.

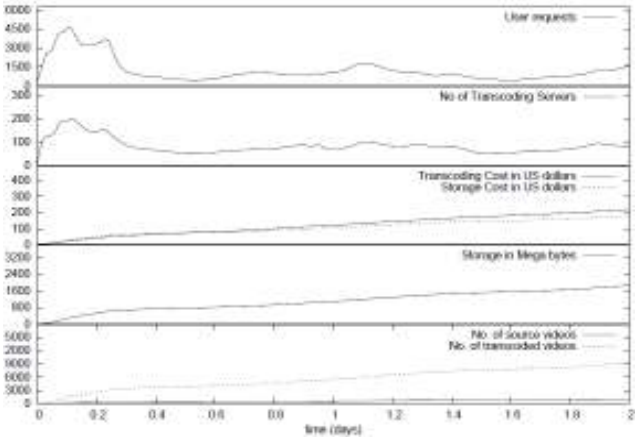

Figure 3: Experiment 1 results of old approach

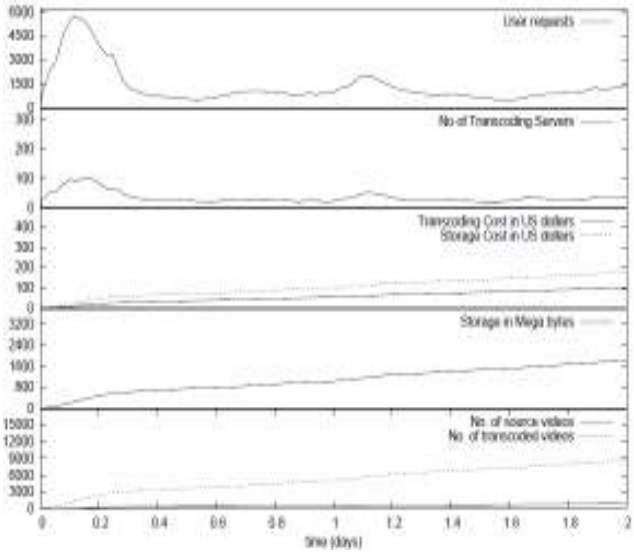

Figure 4: Experiment 1 results of new approach

2) Experiment 2 - 45 possible transcoded formats:

Figure 5 shows results from experiment 2. In this experiment, the possible number of transcoded videos for each source video is 45 which requires more transcoding operations as compared with the results in experiment 1. This experiment also uses the same algorithms as proposed in [11]. In this experiment, a maximum of 304 transcoding servers are used. The average number of servers is 138 and the minimum number of servers is 2. The two-days renting cost of transcoding servers is 301 dollars and the storage cost is 270 dollars. Figure 6 shows the results from experiment 2 using the improved algorithms proposed in this paper. In this experiment, a maximum of 182 transcoding servers are used. The average number of servers is 65 and the minimum number of servers is 2. The two-days renting cost of transcoding servers is 149 dollars and the storage cost is 270 dollars.

From the experiments, it is evident that the new algorithms have approximately half cost as compared with the previous algorithms.

5. CONCLUSION

In this paper, we presented virtual machine allocation algorithms for video transcoding in an IaaS cloud. The proposed algorithms are also compared with the existing, prediction based video transcoding algorithms in a cloud computing environment.

The proposed algorithms use accumulative play rate, accumulative transcoding rate, computation load and system throughput to take resource allocation and deallocation decisions. The proposed algorithms also use a two-step prediction method to predict the future load in order to allow proactive resource allocation. We used GOP level video segmentation to distribute transcoding load among different transcoding servers. The results in both experiments (exp.1 and exp.2) indicate that the proposed algorithms are more cost-efficient as compared to the existing algorithms. In both the experiments, the transcoding cost with the proposed algorithms is almost half as compared to the transcoding cost with traditional algorithms [11].

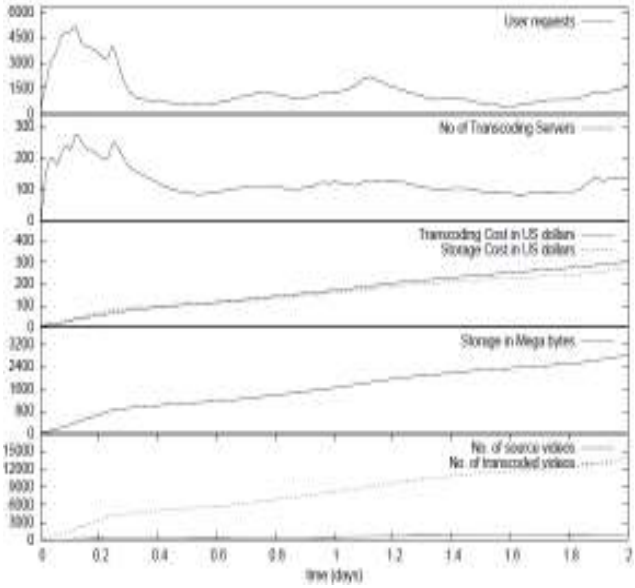

Figure 5: Experiment 2 old results

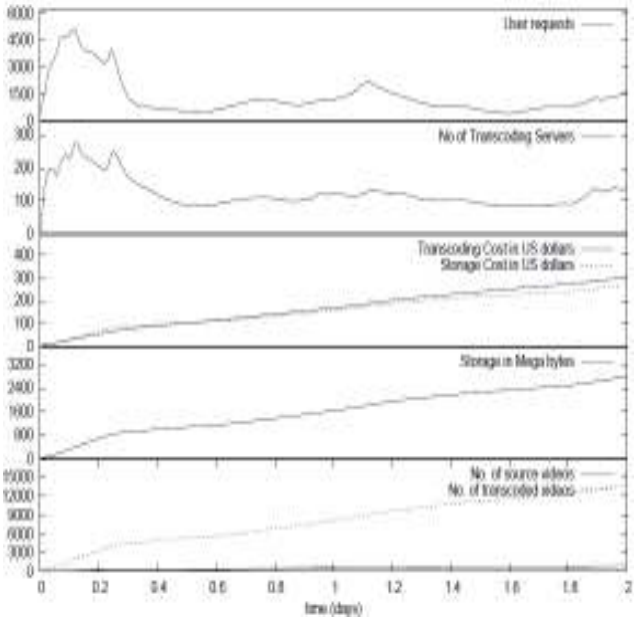

Figure 6: Experiment 2 new results

REFERENCES

[1] 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.

[2] Mauro Andreolini and Sara Casolari. Load prediction models in web-based systems. In Proceedings of the 1st international conference on Performance evaluation methodolgies and tools, valuetools '06, New York, NY, USA, 2006. ACM.

[3] 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.

[4] 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), 2012 38th EUROMICRO Conference on, pages 431 –438, sept. 2012.

[5] 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), 2013 13th IEEE/ACM International Symposium on, pages 482–489.

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

[7] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, C. 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.

[8] 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.

[9] 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 Symposiuml, page 6 pp., Dec 2011.

[10] Fareed Jokhio, Adnan Ashraf, Sébastien Lafond, and Johan Lilius. A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud. In 39th EUROMICRO Conference on Software Engineering and Advanced Applications, pages 365–372. IEEE Computer Society, 2013.

[11] 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.

[12] 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, feb. 2012.

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

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

[15] J. Rhoton. Cloud Computing Explained. Recursive Press, 2010.

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

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

[18] 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.

[19] 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.