I. J. Computer Network and Information Security, **2017, 9, 1-11**
Published Online September 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2017.09.01

CVSHR: Enchantment Cloud-based Video Streaming using the Heterogeneous Resource Allocation

Dr. Mohamed A. Elsharkawey[1], Dr. Hosam E. Refaat[2]
[1,2] Suez Canal University, Faculty of Computers & Informatics, Information System Department Ismailia 41522, Egypt
E-mail: [1]melshrkawey1964@yahoo.com, [2]hosam.refaat@ci.suez.edu.eg

Received: 22 June 2017; Accepted: 01 August 2017; Published: 08 September 2017

***Abstract*—**The Video requests can be streamed in two forms. They are the live streaming and the on-demand streaming. Both of them should be adapted (I.e., transcoded) to fit the characteristics (e.g., spatial resolution, bit rate… and the supported formats) of client devices. Therefore, many streaming service providers are presented the cloud services to be utilized in the video transcoding. But, the introducing of the cloud services for video transcoding is encountered by the contradiction between the deploying cloud resources in a cost-efficient without any major influence on the quality of video streams. In order to address this problem, this paper presents an Enchantment Cloud-based Video Streaming using the Heterogeneous Resource Allocation (CVSHR) to transcode the video streams on cloud resources in an efficient manner with the QoS of the requested video stream. The system architecture is elastic and based on multiple heterogeneous clusters that provide a great flexible resource allocation and De-allocation strategy. This strategy aims to assign a suitable VM with adequate resources based on the GOPs characteristic. Also, it can reassign the unused resources. In addition, the number of VMs can be extended as the system necessity. Finally, The CVSHR is simulated and evaluated on truthful cloud resources and various workload circumstances.

***Index Terms*—**Cloud system, video streams, elastic resource allocation, QoS.

I. Introduction

Nowadays, watching videos have intensely changed from traditional TV systems to video streaming on heterogeneous clients through the Internet. According to Cisco Systems, the anticipated streaming traffic will rise up to 80% of the entire Internet traffic by 2019 [1]. Video content has two system types, the on-demand streaming system (e.g., YouTube or Netflix) or live-streaming system which has multiple presentations on the internet. The heterogeneous clients include desktops, laptops, and smartphones. Each of these clients has its own resource capabilities. In general, the characteristics of the Video content should be adjusted to fit each of the client capabilities and network conditions. These characteristics may include a supported resolution, frame rate, video codec, and network bandwidth of each client device [2].

Of course, it is not accepted to store copies of the same video contents with different characteristics to serve different types of client requests. This approach needs huge storage resources and wasteful processing power. In addition, the infrastructures should be upgraded to achieve the explosive growth of video streaming demands on a large diversity of the client devices. So, this approach remains unachievable. In order to overcome these limitations, video streaming through the Internet is performed via a middle-layer transcoding system. The role of this transcoding system is to perform the mandatory adaptation in the contents of a video stream to fit each client capability and according to the change in network conditions [3].

On the other hand, video transcoding process may be performed via the client devices theoretically. But, practically transcoding has two limitations. Firstly, it requires heavy computation. Secondly, it is a time-consuming process. Due to these limitations, it is not practical to transcode videos on client devices, especially for the most portable clients [4]. So, the most acceptable approach is to perform the transcoding operation on a mid-network node (transcoder). The transcoding nodes should have enough processing power to do the required processing to achieve the needed adaptive. In this research, a proposed alternative approach is introduced to transcode on-demand video streaming using computing services presented by cloud providers.

However, the engagement of using the cloud resources for on-demand video transcoding is faced with a major challenge [5]. This challenge is how to accomplish this engagement in a cost efficient manner with the minimum effect on the QoS demands of video streams. Each Video stream client has its own QoS demands. Particularly, it needs to receive video streams at a specific time without any delay. A delay may occur as a result of two reasons. Firstly, delay due to an incomplete transcoding task by its former. Secondly, delays may occur in the starting of a video stream. In this paper, the former delay is defined as a deadline time that needed to represent the transcoded video stream. In addition, the second delay is defined as

the startup delay for a video stream. The startup delay is considered as a vital key measure the quality of the video streaming service given by the stream provided to the users. Thus, to maximize client's fulfillment, the QoS requirements of the video streaming can be viewed through three main objectives. The first objective is the minimization of the startup delay. Secondly, it is required to preserve the presentation deadline without missing. The third objective can be represented in minimizing the cloud resources allocated for the streaming service to satisfy the QoS demands for the streaming video. In [6], the authors present a CVSS model in order to deal with these objectives. However, this model has undergone some complications. Firstly, the model depends on the historical transcoding execution time the same video stream to predict the current transcoding time of the current execution. This assumption cannot be accepted as a result of different resources and the different load of each execution. In addition, the model assumes the most streamed videos do not be displayed to the end. But, this assumption is not always true. Furthermore, the model assigns the same resources for all used VMs in one cluster. This assumption means many video streams may be assigned to the VM that may have exceeded resources than the assigned task need.

Generally, based on the introduced descriptions and the investigation of the CVSS model, the whole research points can be summarized in satisfying the client request rate with regard to QoS requirements of the streaming video using the minimum cost of the utilized cloud resources. So, the addressed research in this paper can be specified in the following questions:

- How to enhance clients' QoS fulfillment by reducing the video stream startup delay and presentation deadline miss rate?
- How to build an adaptable cloud resource provisioning strategy that minimizes streaming service providers' deserved cost while respecting the demands of each client?

In order to overcome these research challenges, this paper presents an adaptive scheduling approach using heterogeneous resources (CVSHR) for transcoding on-demand video streams employing cloud resources. The heterogeneous resources of the cloud system architecture offer a highly flexible resource allocation to guarantee sufficient video streaming rate for each client request. Also, CVSHR approach introduces an optimized threshold based on the two types of delays mentioned before to promote a continuous video streaming regardless the size of the video stream. The CVSHR approach is implemented by assigning the appropriate VM resources for each type and architecture of video stream. In addition, the number of VMs can be extended in two different cases. The first case, if there is no one of the current VMs can be able to fit the required QoS of the video stream demand. The second case, when the time required for accomplishing the video stream transcoding process makes the deadline time. Moreover, the system efficiency is guaranteed by adding check algorithm to CVSHR to reevaluate the reserved resources periodically. This algorithm aims to release the outdo resources

The rest of this paper is organized as follows. Section II provides some background on video streaming and transcoding. In section III, the CVSHR architecture is presented. Scheduling methods and resource allocation policies will be discussed in section IV and V, respectively. In section VI, performance evaluations are performed. Finally, the Ssection VII concludes the paper and provides venues for the future work.

II. Related Works

Originally, each captured video is stored on the streaming servers based on a specific structure and set of parameters known as transcoding parameters. When the video contents are requested by a specific client, the streaming server has to adapt the original structure and adjust the transcoding parameters of the original video to fit the client's ability. This modification is known as a transcoding process. In the following, the basic video stream structure that is used by the majority of codec video is offered and explained. In addition, the transcoding parameters are presented to illuminate its basic role.

A. Video Stream Structure

As shown in "Fig. 1," Video stream consists of many sequences. Each sequence has one header and several of Group of Pictures (GOPs). Each GOP structure is arranged in successive frames. The first frame in the GOP is known as an intra-frame (I-Frame). The remaining number of frames is a number of P (predicted) frames or B (bi-directional predicted) frames. Actually, there are two types of GOP: open-GOP and closed-GOP [7]. In open-GOP, there is inter-dependence between GOPs. In dissimilarity, each GOP in the closed-GOP is used in an independent manner to the other GOP. So, transcoding can be performed on each GOP independently. Generally, each frame consists of a number of macro-blocks (MB) which are the basic operational units for video encoding and decoding. Moreover, each macro-block contains a number of coefficients characterize the actual data that will be used in the reconstruction of the frame part represented by that macro-block. The transcoding process of video streams can be performed based on different levels, namely sequence level, GOP level, frame level, and the macro-block level. However, the bulky size of each sequence consumes a long transcoding time. On the other hand, the frames, and the macro-blocks have a temporal and spatial dependency that makes the transcoding process at that level complicated and slow [8]. So, the closed-GOP is selected as an optimum level that can be transcoded independently [9], and avoiding complicated and slow operations.

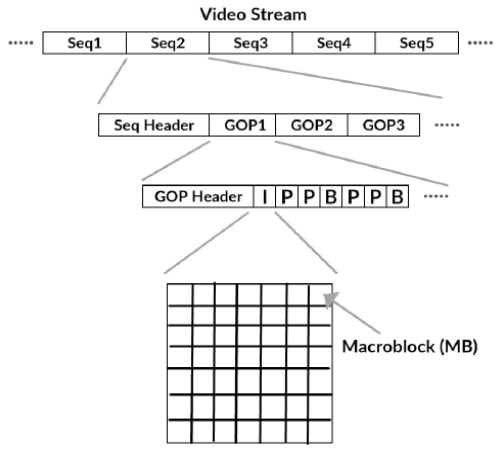


Fig.1. The Video Stream Structure That Consists of Several Sequences. Each Sequence Includes Several GOP. Each Frame of GOP Contains Several Macro-Blocks.

A. Video Transcoding

The transcoding parameters include a specific format, spatial resolution, frame rate, and bit rate. The transcoding is based on the modification of these parameters to fit the client request [10]. In the following, the parameters of transcoding operations are presented and clarified:

1) Bit Rate Adjustment:

In general, the video contents are represented by a massive amount of data. So, the high-quality video has to encode its contents with high bit rate. However, higher bit rate means the need to large network bandwidth for the video content transmission. For the diverse network conditions of clients, streaming, service providers ordinarily have to transcode the video stream's bit rate to ensure smooth streaming [11].

2) Spatial Resolution Reduction:

The spatial resolution specifies the dimension size of an encoded video. The dimension size does not automatically match to the screen size of client devices. Reduction of spatial resolution video is known as downscaling. It is performed by combining or removing the macro-blocks of an original video. Several algorithms are proposed and applied to reduce the spatial resolution without sacrificing video quality [12], [13].

3) Temporal Resolution Reduction:

When the client device doesn't support higher frame rate, the stream server has to drop some frames, and temporal resolution reduction becomes necessary. Accordingly, the motion vectors within the dropped frames are also lost. Due to this dependency, some of the incoming frames that based on lost motion vectors cannot be deduced. The reduction of the temporal resolution can be achieved using methods explained in [14].

4) Video Compression Standard Conversion:

Multiple video compression standards are introduced. They vary from MPEG2 to H.264, and to the latest one, HEVC. Actually, each client has a supported codec, so the video contents should be transcoded to that supported codec on the client to be revealed [15].

III. Proposed Model

The proposed model CVSHR is designed for on-demand video transcoding in cloud systems. An overview of the architecture is introduced in the "Fig. 2,". The CVSHR architecture aims to transcode variant video streaming using elastic and heterogeneous resource allocation of the cloud system. The proposed model consists of two main partitions. The first partition is known as video repository manager (VRM) and the second partition is known as Schedule and Elasticity Manager (SEM). In the following subsections, the architectural components of each partition and its function will be presented and explained.

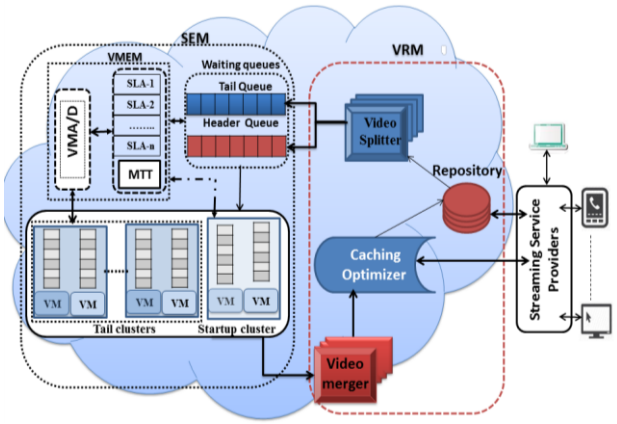


Fig.2. CVSHR Architecture

A. Video Repository Manager (VRM)

The VRM includes four main modules; they are video repository, video splitter, video merger and hot cashing video. The video repository contains all video streams that may be requested by all clients from the video stream provider. When the non-transcoded video stream is requested by a client, this video stream will be moved from video repository to the video splitter module to represent a new job. By Video Splitter, each job will be divided into a set of tasks. Each task is represented by GOPs that can be transcoded independently. The transcoding of GOPs will be performed based on the QoS of the client capabilities and network conditions.

Based on a pre-specified percentage of all GOPs, the first set of GOPs that represents this percentage will be delivered to the header queue in the SEM unit. In addition, the remaining GOPs of the requested video stream will be delivered to the tail queue of the VWQ in the SEM unit. However, the functions of the two remaining modules of the VRM, namely the video merger and hot chasing video are taking place after completing all processes of the SEM unit. The function of the video merger is represented in integrating all the transcoded GOPs in their order to form a completed transcoded video.

Moreover, it passes the completed transcoded video headlong to the video repository to be accessed by clients. The last module of the VRM is the hot caching video. Its role is the monitoring of the access rate of all video streams. There are a few videos that are accessed very repeatedly while many others are infrequently streamed by the clients.

Thus, to avoid excessive and repeated transcoding operations of the frequently accessed videos, the VRM architecture offers a hot caching video to decide whether a transcoded video should be cached or not. However, if the video is barely requested by clients, there is no need to be stored (*i.e.,* cached). The major gain of hot caching video is reducing the start-up time of the high accessed video and saving the cloud resources consummations due to the repeating the same transcoding operations for the same video.

B. Schedule and Elasticity Manager (SEM).

The main role of SEM is to monitor and perform the operations on the stream tasks (GOPs) on the transcoding VMs within the CVSHR architecture. The SEM provides the VMs with adequate capabilities to be dealing with the QoS of the client demands and reducing the deserved cost to the stream provider. Thus, the SEM operations are managed based on an elastic resource provisioning policy. The SEM consists of three main modules. Namely, "the video waiting queues (VWQ)", the Virtual Machines Elasticity Manager (VMEM) and the Virtual Machines clusters.

As mentioned before, the VWQ includes two types of queues. These queues are used for each video stream. The header queues are used to save the first set of the GOPs (tasks) whiles the tail queue is used to save the remaining GOPs sets. Actually, multiple GOPs sets of different video streams may be presented simultaneously in the both queues of the VWQ.

The second module VMEM contains two main components. They are the lists of the service level agreements (SLAs) that are supplied with the placeholder for the computed value of the mean transcoding time (MTT) and the Virtual Machine Allocation/DE-allocation (VMA/D). For the SLAs, each list in the service level agreement (SLA) is formed to work as an interface between one of the tail heterogeneous clusters and the required QoS for the arrival of the video streams. The selection of the VMs cluster is performed based by comparing the QoS of the client demands with the most proper-matched service level agreements (SLA) and the computed MTT that will be clarified later in this section. In addition, the VMA/D module is used to perform two main functions. They are the allocation function and the De-allocation function. The allocation function is executed regularly and in an event-based monitoring the available resources of the activated VMs in all clusters. If the monitoring process notifies that one of the activated VMs in the clusters has an adequate resource to perform the required tasks according to the dedicated SLA, it is selected. Otherwise, the VMA/D tries to find the most proper VM to perform the required tasks with missing rate restrictions. If the both tries are failed, the VMA/D informs the cloud provider to create a new VM that has the mandatory resources of a specified cluster.

In the proposed system, the creation command decides the resources of the new VM. In addition, the grouping of the homogeneous VMs within each cluster introduces the regularity in the streaming process and saves in the resource consumption. On the other hand, the DE-allocation process is executed in heartbeat manner and also in an event-based by the VMA/D. It aims to monitor the resources of the used VMs to distinguish the excess and unused resources. The resources assigned to any VM are removed or reassigned in two cases. The first case, when the transcoding operations for all the GOPs in the local queue of a specified VM have been completed. Secondly, when a client cancels its job, the VMA/D may remove or reassign the unused resources.

The third module includes set of clusters. These clusters are classified into two basic types. They are the start-up type that includes only one cluster and the tail type that may include multiple clusters. The start-up cluster may include multiple VMs with dissimilar resources for each of them. Each cluster of the tail type is heterogeneous. Namely, each cluster has different resources than another one. However, all VMs are homogenous within each cluster of the tail type. Namely, all VMs belong to the same tail cluster have the same resource capabilities. Hence, the VMs that are allocated from cloud provider are assigned according to the specifications of each cluster. Additionally, a local queue is assigned to each VM to receive its allocated GOPs. The pre-loading of the GOPs in the local queue will minimize an execution delay of the tasks. The tail clusters classifications are performed to be matched to the QoS, MTT and demand rates specified in the one of the SLAs to deal with the demanding QoS of the demanded video streams. So, a novel cluster may be created when the required QoS of novel video streams are not adapted to one of the current existing clusters. In addition, an extra VM may be added to the same cluster when the scheduling video streams are overwhelmed the existing VMs within that cluster. More details concerning the SEM resource provisioning policies are discussed in the following section.

The operations of each video stream are performed on two consecutive and separated processes. In the first, the GOPs are assigned to the header queue of the start-up cluster with the appropriate SLA. The fitting VM of this cluster is used to accomplish two main objectives. Firstly, it minimizes the start-up time of each video stream by transcoding its header GOP on one of the available VMs of the start-up cluster. Secondly, it is used to compute the value of the mean transcoding time (MTT) for each GOPs set of each video stream and return the computed value to the SLAs module. Secondly, the tail GOPs is allocated by using VMA/D to the cluster that preserves the required resources for transcoding the tasks based on the SLA and MTT. Hence, the VMA/D will pass the tail GOPs to the local queue of the intended VM in one of the tail clusters at the same time with the MTT.

The MTT is very important and vital value in the proposed model. Instead of depending on the historical values of the previous transcoding of that stream as in the CVSS model. The computed MTT is used as an estimated transcoding time for all GOPs (tasks) that still remain in the tail queue and belong to the same video stream. So, as the MTT for each set of GOPs for a specified video stream is evaluated, the VMA/D module uses it as a predicted transcoding time. Based on this predicted transcoding time and dedicated SLA, an appropriate cluster from the tail cluster is allocated. The appropriate cluster is the cluster that has the VMs of adequate resources to perform the remaining GOPs (tasks) for the stream stored in the tail queue.

IV. QoS-Evaluation for Transcoding Scheduling

For scheduling, the first set of GOPs of the requested video stream is placed in a header queue upon arrival, while remaining GOPs sets are passed to the tail queue. In order to minimize the start-up delay, a separate header cluster is implemented to be permanently having a free spot in any of the local queues of its VMs. This permanent free spot makes the start-up cluster is ready to receive any new set of GOP. Based on the monitoring operations of the VMA, the VM that has the shortest length local queue compared to the all local queues of the VMs in the startup cluster is selected. For this VM, the SLA that matched the required QoS of the loaded video stream is submitted. In concurrence, the header GOPs of the video streams are passed to its local queue. As an opposite to all previous research deal with transcoding of the on-demand video [16], the proposed model CVSHR separates the VMs that transcode the header GOPs from the VMs that are assigned to transcode the remaining GOPs of the video stream. This separation leads to a reduction in startup delay due to the VMs of the startup cluster that is assigned only for this purpose. This assignment reduces the amount of the waiting time in the local queues of the VMs in the startup cluster. On the other hand, each GOP is treated as a separate task with an individual deadline. So, the deadline of a GOP is re-defined as the presentation time of the first frame in that GOP. According to this vision, the entire deadline for each GOP can be evaluated as follows:

For a GOP number m in the video stream, n, D_{mn} that is denoted as the deadline time can be evaluated according to the equation:

$$D_{mn} = s_n + \rho_m \rightarrow \quad (1)$$

Where s_n : is the startup time of the video stream, ρ_m is the presentation time of the all previous GOPs of the video stream.

However, the prediction of the transcoding time represents a vital problem for the on-demand video streaming. In the previous research, it is evaluated from the historic transcoding times of the different clients for each video stream [17]. This method assumed the homogeneous of the applied VMs in each time of the transcoding. Actually, this assumption is impossible to be satisfied due to the difference in the available resources that may be assigned each time to the homogeneous VMs. Each time, the available resources are changed due to the circumstance that the VMs can be assigned to different physical machines on the cloud.

Therefore, the GOPs that are assigned to the header queue are used to evaluate the MTT. The MTT is computed as an average of the actual transcoding times of the GOPs presented in the header queue for each video stream. This computational method is performed based on currently available resources assigned to the applied VMs to overcome the resources problem in the previous researches. This computed MTT will be used as an estimated transcoding time for the remaining GOPs of the whole video stream. So, the MTT is computed according to the following equation:

$$MTT_n = \frac{\sum_{i=1}^{h} GT_i}{h} \rightarrow \quad (2)$$

Where h is the selected number represents the set of the GOP executed in the startup cluster belong to the same video stream n and GT_i is the actual time consumed to transcode the GOP_i.

However, the predicting transcoding time may be affected due to the randomness when the transcoding operations are performed on the VMS for the remaining GOPs that are executed in the tail cluster. So, the absolute transcoding time is proposed to acquire the worst-case analysis of transcoding time estimation that will be denoted as τ_n and defined by the equation

$$\tau_n = MTT_n + \sigma_n . \rightarrow \quad (3)$$

σ_n : is the standard deviation of the header transcoding time for GOPs of the same video stream.

For simplicity, to evaluate the predicted completion time that denoted as ψ , the loading time of the video stream from hard disk is neglected.

For the GOPs number I in the video stream, n, the task completion time $\psi_{n,i}$ is driven based on the following equation:

$$\psi_{n,i} = \tau_c + \tau_r + \omega + \tau_n \rightarrow \quad (4)$$

Where, τ_c is the current time, τ_r is the remaining executing time of the current task spend in the VM.

ω : is the scheduled waiting time of the tasks reserved in the local queue for execution, and τ_n is the estimated transcoding time that is evaluated from the first equation.

The scheduled waiting time ω is evaluated from the equation

$$\omega = \sum_{k=1}^{n} m_k \tau_k \ . \rightarrow \quad (5)$$

Where, τ_k is the absolute transcoding time computed from the first equation of the k[th] video stream and m_k is the number of GOPs for that video in that local queue.

As mentioned above, the deadline time of any GOP_x in the tail queue of the remaining video stream is equivalent to the startup time plus the presentation time of all previous GOPs. Hence, the deadline time can be deduced from the following equation:

$$D_{n,x} = s_n + \sum_{i=1}^{x-1} p(G_i) \rightarrow \quad (6)$$

Where s_n : is the startup time of the header GOPs of the video n and $p(G_i)$ is the presentation time of the GOP number i .

Hence, the following equation is used to decide if that GOP_i can wait in this VM queue or not.

$$\psi_{n,i} \leq D_{n,x} \rightarrow \quad (7)$$

This equation is used to select the appropriate VM within the anticipated cluster. The VM is selected such that the completion time of each transcoded GOP must be less or equal to the predicted deadline time for that GOP. This condition is an obligatory condition to avoid the missing of the GOP when actually transcoded to be revealed in the scheduling presentation time.

V. Dynamic Resource Provisioning Policy

In order to maximize the benefits of the system resources, two functions are presented. They are the allocation and De-allocation functions. The allocation function aims to find the most proper and adequate resources to perform the required tasks. The allocation is achieved through three levels. It starts by observing the available resources of the activated VMs in all clusters. If adequate resources are not available, it tries to find the most proper VM to perform the required tasks with missing rate restrictions. Otherwise, it informs the cloud provider to create a new VM with the required resources in a specified cluster. On the other hand, the de-allocation function aims to detect the excess and unused resources. These resources are removed or re-assigned to another VM. In the following, the operations and the pseudo codes of the both functions are offered and explained.

A. Allocation

The pseudo code of the allocation algorithm is shown in the "Fig. 3". Generally, each VM in the start-up cluster has an upper bound of the load (β) to minimize the start-up delay. So, the allocation algorithm tries to find a start-up VM which has sufficient free slots and adequate resources to receive all the GOPs assigned to the header set of the arrived video stream. If there is no VM in the start-up cluster has these adequate slots and resources, the allocation algorithm asks the cloud provider to create a new VM. After allocation and based on the transcoding parameters specified in the SLA, the algorithm computes the mean transcoding time using the equation number 3. As shown in line 7, four parameters are involved in determining which cluster of a specified VMs type that will be able to perform the required transcoding. These parameters include the computed MTT, the submitted SLA that includes the Upper QoS threshold and the Lower accepted QoS threshold. Once the cluster type is determined, the algorithm starts to compute the predicted deadline time for each GOP. As mentioned before, the

Allocation Algorithm

Input

$h_i = \{G_{i,1},.....,G_{i,l}\}$ /* The GOPs sets that transferred from the video splitter to a HQ in the WFQ*/

$t_i = \{G_{i,h+1},.....,G_{i,n}\}$ /* The GOPs sets that transferred from the video splitter to a TQ in the WFQ*/

υ // Upper QoS threshold

ε // Lower QoS threshold

β /* The maximum number of header GOPs presented in the local queues of the startup VM.*/

1. VMSrtQue = FindStrtQue(l) /* Fined free slots in startup queue slots $\geq l$ */
2. If (VMSrtQue = null)
3. VMSrtQue = CreateStartupVM(β) /* create new startup VM with maximum length of l=local queue β */
4. End if
5. Enqueue(VMSrtQue, h_i) //Enroll the header GOP into a retrieved startup VM queue
6. MTT=MTT(h_i)+ σ // retrieve the mean transcoding time modified by standard deviation
7. VMType = SelVMType(MTT, SLA, υ , ε)
8. For each task in $G \in t_i$
9. Deadline= calculateDeadline(G)
10. E = FindQueue (Deadline, VMType) /*return a VM queue that fit the predicted deadline time of GOP */
11. If $E = null$
12. E = FindHigherQueue (Deadline, VMType)
13. End If
14. If $E = null$
15. E =CreateVM(VMType) /* add a new VM the cluster that satisfy the required condition*/
16. End If
17. Enqueue(G , E)
18. End for

Fig.3. Pseudo Code of the Allocation Algorithm

predicted deadline time for each GOP is evaluated as the summation of the start-up delay plus the presentation time of the all previous GOPs. Finally, based on the VM type and the value of the predicted deadline time, the allocating algorithm determines precisely which VM within the specified cluster will be used to perform the required transcoding for each GOP in the tail queue. In other words, the algorithm for each GOP has to find a VM with a specific type and its waiting queue will not miss the deadline. If there is no VM within the specified cluster type will fit these requirements, the algorithm tries to find another type cluster to accomplish the required transcoding without missing the predicted deadline. If there is no current VM will achieve the required conditions, the allocation algorithm will ask the cloud provider to add a new VM with intended specifications.

B. De-allocation

The pseudo code of the de-allocation algorithm is shown in "Fig. 4," It is implemented based on the following two principles:

- Releasing the VMs of an idle status that has empty local queues.
- Transform the status of some VMs from a busy state to an idle state by reassigning the tasks in their local queues to another local queue of VMS without affecting the deadline of these tasks

De-allocation:(unused or extra-reserved resources)Algorithm

Input

Q = $\{q_1, q_2, \ldots, q_n\}$ //The system has VM queues for the same cluster

1. Sort(Q) // sorted Q by the length of queues in ascending order
2. For (i=1,…,n) //take VM form the shortest queue to the longest one
3. For (j=n,…,i-1) //take VM form the shortest queue to the longest one
4. For each task $t \in q_i$
5. E = FindQueue(t, $t.deadline$, υ, ε) /*return another queue in the same Cluster which will not missing t deadline*/
6. If $E \neq null$
7. migrateTask(t, E) // move the task to the new queue
8. End if
9. End for
10. If $|q_i| = 0$ // the queue become empty
11. Release VM_i // free the VM for thr queue q_i
12. End if
13. End for
14. End for

Fig.4. Pseudo Code of the De-Allocation Algorithm

In the implementation of this algorithm, the local queues of VMs within each cluster are scanned periodically to reevaluate the existing load within each queue. After each scanning, two operations are performed. Firstly, the VMs of empty queues are released by the algorithm. Second, the local queues within each cluster are sorted in ascending order. The sorting is performed according to the number of busy slots. In addition, the VM that has the smallest number of tasks occupied in the slots of its local queue is selected. The algorithm tries to find another local queue in another VM to perform the required transcoding for these tasks without missing the deadline using the FindQueue method, as shown in line (5). Hence, the algorithm starts to re-distribute the load of the shortest queue to longer queue. If there is no another queue in that cluster can handle the intended tasks without missing the deadline, the algorithm tries to find a queue in the higher cluster to perform the required transcoding for these tasks without missing the deadline. When the shortest queue becomes empty, the engaged VM must be released, and the same operations are repeated for the next shortest busy queue.

VI. Performance Evaluation

The performance of the proposed model is measured by implementing its simulation on the WorkflowSim [18]. The WorkflowSim is an open source workflow simulator extends the CloudSim [19]. So, it will be used to evaluate the performance of scheduling methods and resource provisioning policies. The system evolution is performed by selecting a variety of video streaming requests from a set of benchmark videos in the range between 10 and 600 seconds. Approximately, we try to model the simulation of the proposed system under the same conditions used for simulating the CVSS system to accomplish a perfect comparison. Therefore, the characteristics of the used system resources are based on the characteristics of VMs in Amazon EC2. So, the transcoding execution times of the selected videos are performed by transcoding them on T2.Micro instances of Amazon EC2 that are available for free. Since the proposed model is interested in the cost of VMs allocated for transcoding operations, the calculations of the repository costs are neglected. In order to achieve the randomness, in the execution time of transcoding tasks on cloud VMs, the GOPs of each benchmark video is transcoded for 20 times. In addition, the proposed system is evaluated under different amount of workload. Hence, the performances of the requests of the video stream are measured by changing the range of requests from 100 to 1000 within the same period of time. However, for standardization of the comparison of both systems, the experiments were performed using a static number of 9 VMs. For the CVSS system, the resources assigned to the 9 VMs are equally distributed. Namely, all VMs will be homogenous having equivalent resources. On the other hand, the 9 VMs for the CVSHR will be distributed into three clusters. Each cluster will have three VMs. The same total amount of resources assigned to the 9 VMs of the CVSS system will be unequally distributed to the 9 VMs of three clusters of the proposed System CVSHR. So, theses cluster can be classified into three categories. The high cluster has high resources for each VM than the VMs in other clusters. Also, the middle

cluster has resources for each VM less than the VMs of the high cluster, but greater than those of the low cluster. However, the VMs within each cluster are homogeneous but different from the VMs of another cluster.

In the following, the simulation of the proposed system is evaluated according to the three different circumstances. Firstly, measuring the performance of the proposed system (CVSHR) compared with the performance of the CVSS system. The comparison is measured when it is required to accomplish a specified QoS. Secondly, the impacts of the queuing policies are implemented to measure their effects on the both models. Finally, the implementation of the dynamic resource provisioning policy is compared with the static resource provisioning policy for the proposed model.

A. Influence of the QoS-aware Scheduling Method

"Fig. 5," demonstrates the changes in the average start-up delay of video streams when the proposed QoS-aware scheduling method is applied for both of the proposed (CVSHR) and the (CVSS) systems. The experiments were performed with different numbers of video stream requests arriving during the same time interval to reveal the influence of the different amount of workload (horizontal axis in "Fig. 5,"). Moreover, Shortest Job First (SJF) is used for the queuing policy in the waiting queues.

As shown in "Fig. 5A," the average start-up delays of both curves are less than1 second when video requests are less than 500. But, as the number of video requests is increased, the average start-up delays are increased rapidly for the CVSS and their values are moved near to 2.5 seconds. On the other hand, the CVSHR curve is keeping the average start-up delays less than 1 second even with the higher requests. These results have been accepted due to the start-up cluster that has VMS only assigned for the transcoding of the header GOPs set of the video streams. On the CVSS, the same VMs are assigned for both types of the GOP.

"Fig. 5B," illustrates a big a difference between both systems, especially when the numbers of the video stream requests are highly increased. For the CVSHR, the average deadline misses rate is almost keeping its values near to the 4%. On the other hand, these values will increase more than 10% for the CVSS. The comparisons of these results are seemed logic results due to the advanced distribution of the available resources into different proficiencies within different clusters of the CVSHR system. The advanced distribution makes each task is assigned to the VM of the suitable resource. These outcomes can be assured in "Fig. 5C," that discloses the difference between the both systems (CVSHR and CVSS) for the incurred cost. In spite of all tasks have been completed in both systems, the heterogeneous adaptation of the VM resources gives CVSHR an apparent enhancement in the incurred cost.

B. Influence of the Queuing Policy

In "Fig. 6," the different queuing methods are applied to the queues of both systems. They aim to measure the policy effect of each queuing method with the intended QoS scheduling method to accomplish the minimum of the start-up delay, deadline miss rate, and the incurred cost. Two different queuing methods are applied. Namely, first come first serve (FCFS) and shortest job first (SJF). The performance of each queuing method is measured using the same static resource provisioning policies used in the previous test. The results of these experiments are shown in "Fig. 6".

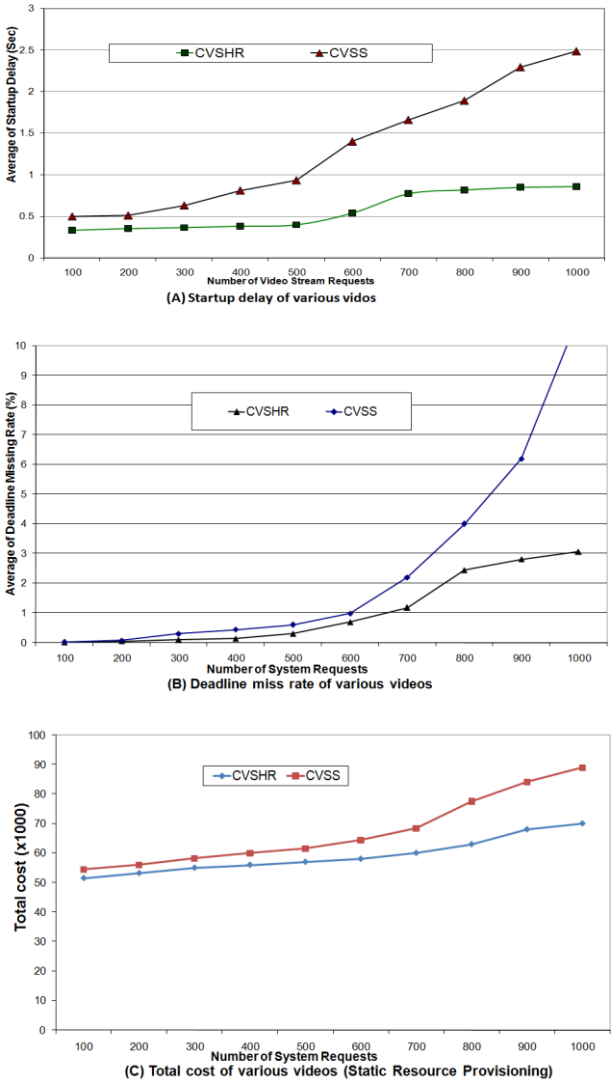


Fig.5. Performance Comparisons between CVSHR and CVSS Are Plotted with Varying Number of Video Requests.

"Fig. 6A," and "Fig. 6B," illustrates as the number of video requests increased, the values of the average start-up delay and average deadline miss rate will remain low and more stable with the implementation of the SJF for both systems. These results have been accepted since the implementation of the SJF policies will rapidly reduce the amount of GOPs waited to be transcoded than in the FCFS policy implementation. Therefore, the start-up delays will be reduced with the SJF and become more adopted with the static provisioning policy ."Fig. 6A," and "Fig. 6B," demonstrates clear enhancements in the average start-up delay and average deadline miss rate

results for the CVSHR especially when a number of video requests are greater than 500. These results can be recognized since the resources implemented through the cluster categories. The different resource capabilities for each cluster lead to the employing of each job to each tone cluster. I.e. approximately, no job will be assigned to a VM that has extra resources than it will need. Finally, the "Fig. 6C," illustrates that there are little bit enhancements by using the CVSHR system than the CVSS system with respect to the total cost that refers to the consumed resources.

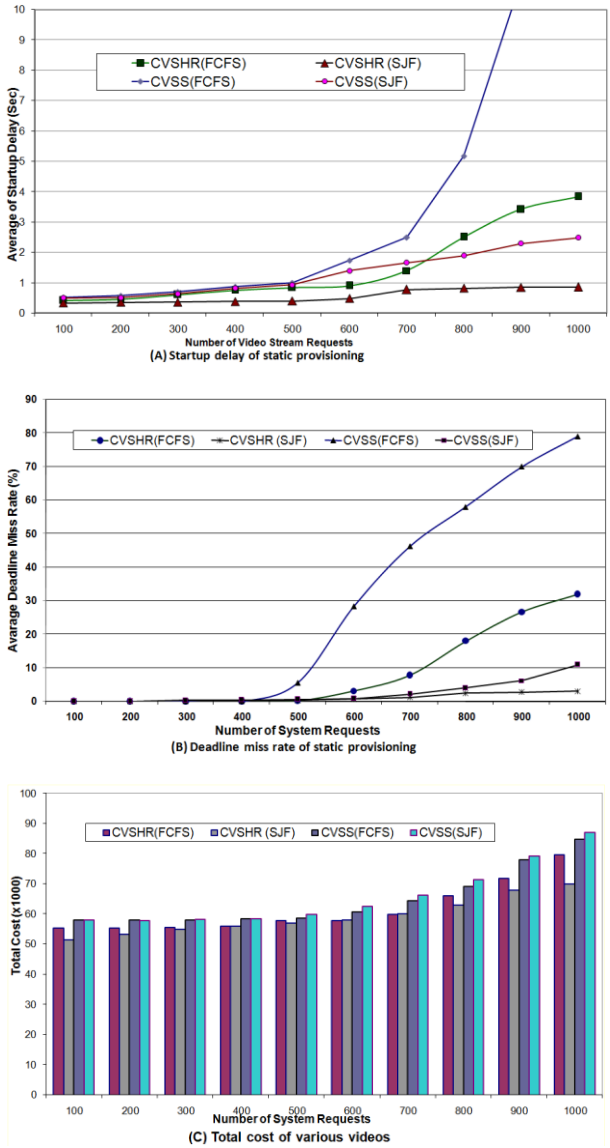


Fig.6. Performance Comparisons between CVSHR and CVSS for the Influence of Different Queuing Policies on the QoS-aware Scheduling

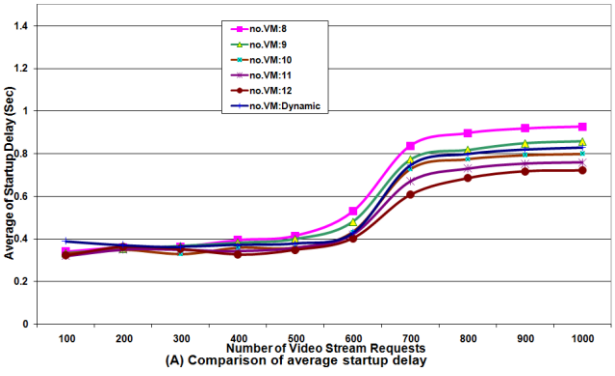
C. Dynamic versus Static Resource Provisioning Policy

In the following, the CVSHR system is evaluated by both of the dynamic resource provisioning policy and the static resource provisioning policy to compare their impact on the CVSHR performance. The comparison is performed by using the SJF that gives a better result to measure the QoS violation and the incurred cost of the previous test. The static provisioning is performed using a fixed number of VMs from 8 to12. Logically for both systems, the lack of VMs will increase the average start-up delay and average deadline miss rate, especially for the huge number of video requests.

"Fig. 7A," illustrates the result of the average start-up delays with the static and dynamic provisioning policies for both systems. These results introduce two outcomes. Firstly, for an increase in the number of video requests, a low and stable average startup delay is accomplished for the dynamic policy than the static policy. The enhancements of the start-up delay results satisfied with the CVSHR system can be clarified due to the using of the elasticity manager. It rapidly assigns each incoming video request to the VM with adequate resources in one of the heterogeneous clusters that pre-defined by one of the SLA. Secondly, a little enhanced is achieved in the result of the average start-up delays with the static policy than the dynamic policy for light loads. This outcome can be explained by the number of VMs used in both cases. The static policy usually starts with a large number of VMs. On the other hand, the dynamic policy mainly starts with a small number of VMs to reduce the incurred cost. So, the new tasks should have waited until the new VM is allocated.

"Fig. 7B, shows that the average deadline missing rate of the static reserving policy is increased rapidly when a little number of VMs will be used. On the other hand, the dynamic resource provisioning policy reduces the average of deadline missing rate when compared with the static policy. The enhancement that clearly realized by the CVSHR than the CVSS system can be clarified from the usage of SLA and the computed MTT. Both of them are used to speedily determine the characteristics of the required VM. This swift determination will reduce the amount of waiting time for each video stream and hence reducing the missing rate.

"Fig. 7C," demonstrates that for a light load of video requests the incurred cost will be reduced by more than (50%) for the dynamic VM provisioning compared with the static provisioning policy. In this case, the stream provider that uses a static policy will be paid for unused resources. On the other hand, as the video requests increased, more VMs are invoked and hence, the incurred cost of the dynamic policy will be increased matching the static one.



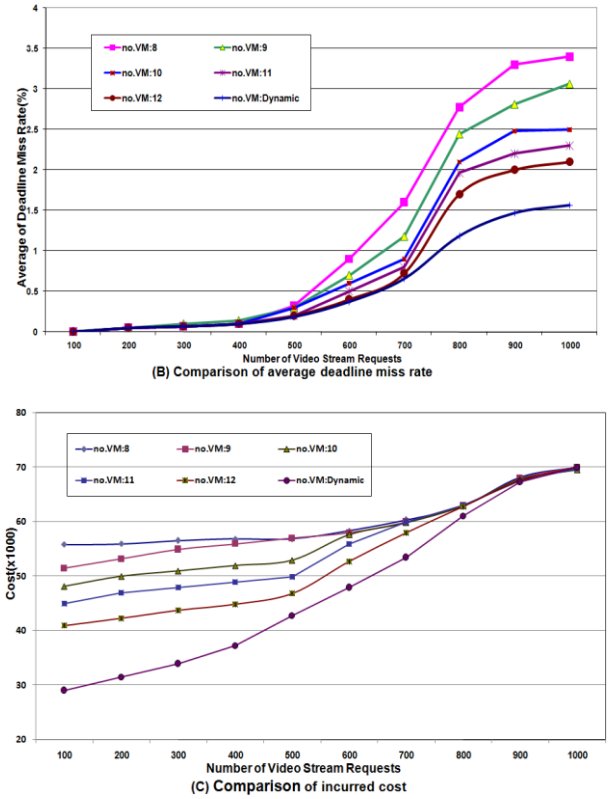


Fig.7. Comparing the Performance of the Static and Dynamic Provisioning Policies

VII. Conclusion And Future Work

In this paper, CVSHR model is proposed for transcoding video streams on heterogeneous cloud resources. The model designed for efficient reserving cloud resources under cost constraints without major influence on the quality of video streams. This model also provides elastic allocation and de-allocation policy to re-evaluate the necessary system resources. Moreover, the main objective of this model is providing cloud resources to fit the deadline of each GOP in the video stream. The system evaluation assures that, the CVSHR enhances the average deadline miss rate values when compared by the VCSS. Also, CVSHR gives breakthrough enhancement in the average startup delay values. In queuing policy, the experiments show the SJF gives better performance than the FCFS. In addition, by comparing the test of static and dynamic reserving policy, we concluded that; the static reserving will be paid for unused resources. On the other hand, as the video requests increased, more VMs are invoked and hence, the incurred cost of the dynamic policy will be increased matching the static one. In the future work, we have two targets. The first one is adding the edge computing to enhance the performance of the system. The second target is to re-engineer the system to be more suitable for live video.

References

[1] C. V. N. Index, “Forecast and methodology, 2014-2019,” 2015.

[2] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, “Video transcoding: an overview of various techniques and research issues,” IEEE Transactionson Multimedia, vol. 7, no. 5, pp. 793–804, 2005.

[3] Zhijun Lei and Nicolas D. GeorganasAdaptive video transcoding and streaming over wireless channels Journal of Systems and Software, Volume 75, Issue 3, pp 253-270, March 2005.

[4] X. Li, M. A. Salehi, and M. Bayoumi, “Cloud-based video streaming for energy- and compute-limited thin clients,” in the Stream2015 Workshopat Indiana University, Oct, 2015.

[5] Khosro Mogouie, Mostafa Ghobaei Arani, Mahboubeh Shamsi, “A Novel Approach for Optimization Auto-Scaling in Cloud Computing Environment”, PP.46-53, Pub. Date: 2015-10-8, DOI: 10.5815/ijcnis.2015.11.05

[6] Xiangbo Li, Mohsen AminiSalehi, Magdy Bayoumi, Rajkumar Buyya, "CVSS: A Cost-Efficient and QoS-Aware Video Streaming Using Cloud Services", in 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2016.

[7] M. A. Mesa, A. Ramirez, A. Azevedo, C. Meenderinck, B. Juurlink, and M. Valero, “Scalability of macroblock-level parallelism for h. 264 decoding,” in Proceedings of the 15th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 236–243, 2009.

[8] F. Lao, X. Zhang, and Z. Guo, “Parallelizing video transcoding using map-reduce-based cloud computing,” in Proceedings of IEEE InternationalSymposium on Circuits and Systems, pp. 2905–2908, 2012.

[9] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, “Analysis of video segmentation for spatial resolution reduction video transcoding,” in Proceedings of IEEE International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), pp. 1–6, 2011.

[10] A. Vetro, C. Christopoulos, and H. Sun, “Video transcoding architectures and techniques: an overview,” IEEE on Signal ProcessingMagazine, vol. 20, no. 2, pp. 18–29, 2003.

[11] ZhijunLei, and Nicolas D.Georganas "A rate adaptation transcoding scheme for real-time video transmission over wireless channels" Signal Processing: Image Communication, Volume 18, Issue 8, pp 641-658, September 2003.

[12] J. Xin, M.-T. Sun, K. Chun, and B. S. Choi, “Motion re-estimation for hdtv to sdtv transcoding,” in Proceedings of IEEE InternationalSymposium on Circuits and Systems (ISCAS), vol. 4, pp. IV–715, 2002.

[13] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, “A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud,” in Proceedings of the 39th IEEE Conference on Software Engineering and Advanced Applications (SEAA), pp. 365–372, 2013.

[14] Y. Ismail, J. B. McNeely, M. Shaaban, H. Mahmoud, M. Bayoumiet al., “Fast motion estimation system using dynamic models for h. 264/avcvideo coding,” IEEE Transactions on Circuits and Systems for VideoTechnology, vol. 22, no. 1, pp. 28–42, 2012.

[15] T. Shanableh, E. Peixoto, and E. Izquierdo, “Mpeg-2 to hevc video transcoding with content-based modeling”,IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, pp. 1191–1196, 2013.

[16] Deepika Saxena, R.K. Chauhan, Ramesh Kait "Dynamic Fair Priority Optimization Task Scheduling Algorithm in Cloud Computing: Concepts and Implementations", PP.41-48, Pub. Date: 2016-2-8, DOI: 10.5815/ijcnis.2016.02.05

[17] Mokhtar A. Alworafi, Atyaf Dhari, Asma A. Al-Hashmi,

Suresha, A. Basit Darem "Cost-Aware Task Scheduling in Cloud Computing Environment", PP.52-59, Pub. Date: 2017-5-8, DOI: 10.5815/ijcnis.2017.05.07

[18] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in 2012 IEEE 8th International Conference on E-Science, ser. eScience, 2012, pp. 1–8. [Online]. Available:https://github.com/WorkflowSim

[19] 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, 2011.

Authors' Profiles

Mohammed A. El-Shrkawey received his B.Sc. in Electrical engineering from the Military Technical Collage, Cairo in 1987. He received his M. Sc. in Computer Engineering from Faculty of Engineering, Al Azhar University, Cairo in 2002. He received his Ph. D. in Network Security from Faculty of Computers & Informatics, Cairo University in June 2007. He is currently a lecturer in Faculty of Computers & Informatics, Suez Canal University, Ismailia, Egypt. His current research interests are Networks, Modeling, simulation, and Image Processing

Hosam E Refaat: has graduated from the Faculty of Science, Assuit university, Egypt, in 1998. In October 2006, he finished his Master degree in the field of distributed systems from the faculty of Science, Cairo University, Egypt. Currently, he is a lecturer in Faculty of Computers & Informatics, Suez Canal University, Ismailia, Egypt. His current research interests are Parallel Systems, Cloud Computing, Edge Computing, and Data mining.

How to cite this paper: Mohamed A. Elsharkawey, Hosam E. Refaat,"CVSHR: Enchantment Cloud-based Video Streaming using the Heterogeneous Resource Allocation", International Journal of Computer Network and Information Security(IJCNIS), Vol.9, No.9, pp.1-11, 2017.DOI: 10.5815/ijcnis.2017.09.01

