ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

QoS-aware Video Transcoding Service Composition Process in a Distributed Cloud Environment

Nawaf O. Alsrehin

Graduate Teaching Assistant, Dept. of Computer Science, College of Engineering, Utah State University, Logan, Utah, USA

ABSTRACT: In this paper, we address the problem of selecting and composing video transcoding services in a distributed cloud environment. One of the challenging issues for video transcoding service composition is how to find the best transcoding path to route the data flow through while satisfying the viewer requirements and specifications. In a cloud environment, video transcoding service providers provide different video transcoding services that have similar functionality (i.e., format conversion), but with different Quality of Services (QoS) specifications. Since the combination of the QoS specifications, such as frame size, frame rate, video bit rate, and transcoding delay might affect the end user's experience in non-intuitive and subjective way and also might affect the delivering of a high quality video content over any type of network, we propose a QoS-aware model to select and compose the best video transcoding services to satisfy hard constraints on the input and output video formats and comes as close as possible to satisfying soft constraints on the QoS. This model uses an aggregate function to evaluate the QoS for each transcoding service and for each viewer request to explore the best composition path. In this paper, we adapt the Simulated Annealing (SA) algorithm and the Genetic Algorithm (GA) as candidate solutions to help in the composition process. The SA/GA algorithms provide multi-constraints QoS assurance for video transcoding service composition. They also support directed acyclic graph composition topology. We have implemented a prototype of the proposed algorithms and conducted experiments using small-, medium-, and large-scale graphs of video transcoders and sample viewer requests to measure the performance and the quality of the results. The experimental results show that the SA outperforms the GA in terms of performance and success ratio for small-scale graph, while GA outperforms the SA algorithm in terms of performance for medium- and large-scale graphs. The success ratio for the SA and GA algorithms are close to each other for medium- and large-scale graphs. At the end, we provide several directions and suggestions for future work.

KEYWORDS: video transcoding services; quality of service; service selection; service discovery; video delivery system, cloud computing

I. Introduction and Background

In recent years, accessing video content has become more difficult for several reasons. First, the number of end-user wire and wireless devices, such as desktop computers, smart phones, tablets, laptops, …, etc, has increased dramatically, putting increased demands on video delivery systems. Cisco® predicts that there will be 50 billion devices connected to the Internet of Things by 2020 [1]. Second, the end-user devices themselves have better displays characteristics, computational power, and storage, enabling higher video quality, which in turn increases the demand and performance expectations for content delivery. For example, iPhone 6 supports 1080p HD video at 60 frames per second (fps) and SLO-MO video at 720p [2]. Third, as the video delivery systems grow, they tend to diversify and introduce more heterogeneity among the components. Fourth, the end-users demand more control of the Quality of Service (QoS), to accommodate different uses of videos. Finally, there has been an unprecedented explosion in the availability of video content and varieties of video formats. By 2018, Cisco® predicts that every second, nearly a million minutes of video content will cross the network and the IP video traffic will grow from 66% in 2013 of all the internet traffic consumers to 79% by 2018 [3]. This is just the beginning of a video flood that will inundate the internet.

To allow any user to watch any video from any kind of display device over any type of network, any video delivery system must be able to transcode the original video content into a compatible format (i.e., codec) that meets bandwidth

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

limitations, device-specific requirements, and end-user preferences. Video transcoding services are components within a delivery system that transform or convert video content from one format to another (e.g., from MPEG-2 to H.264). This transcoding process may also involve frame-size conversion, bit-rate conversion, or frame-rate conversion. These conversions are based on the end-user preference values. The transcoding process is usually done in cases where a target device (or playback software) does not support the original video's format or has limited storage capacity, or in order to convert incompatible or obsolete data to a better-supported or modern format [4]. Fig. 1 depicts the general video transcoding process.

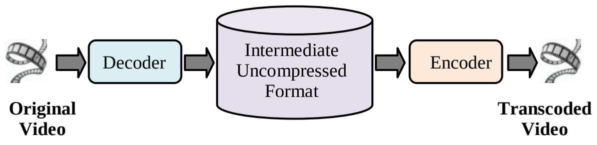

Fig. 1. Video transcoding process

When an end-user wants to watch a cloud-based video through a viewer (i.e., video playback software), the viewer needs to request a stream for that video. That request includes: a) the required video-coding format, also called the video-compression format, and b) a desired QoS that specifies a hoped-for video quality and a tolerable delay. A video coding format defines the structure of the video's image and audio data. Some popular formats are H.264/MPEG-4, MJPEG (Motion JPEG), WMV, and DivX, but there are over 20 in common use today [5]. A codec is a piece of software that can encode video data into a particular format or decode a video from that format. A codec's name is often used as a synonym for the format with which it works.

In general, frame size, frame rate, video bit rate, and audio sampling rate characterize the quality of a video and part of QoS specifications [6]. However, since the audio portion of the video accounts for only a small portion of the data, it is often not considered in QoS-related decisions. When dealing with streaming, a desired QoS may also include a constraint on the amount of delay that the viewer is willing to tolerate in the stream. Even though the delay does not directly characterize a video's quality, it does characterize a video stream and it affects the end user's experience. So, in this paper, we limit the QoS properties to: frame size, frame rate, video bit rate, and the average transcoding delay.

A viewer's desired QoS will be determined based on user preferences, viewer's window size, device capabilities, power-savings setting, network bandwidth, and other device-specific factors. For example, a 78.0" Diagonal UHD display device requires around 80Mbps video bitrate [7] while a 5.5" iPhone 6 plus requires around 18Mbps video bitrate [8]. Optimal utilization of the overall video delivery system resources comes from successfully calculating the desired QoS values for a specific device, playback software, or network connection. These resources are either device-specific resources, such as the computational power and the internal buffers, or network-specific resources, such as network bandwidth. Incorrect calculation of these values might negatively impact the end-user's subjective perception of the video. For example, asking for higher video bitrate for a device that has low computational power or low network bandwidth might make the device much slower than usual and the video might look jerky. In contrast, asking for low video bitrate while the device has a high computational power and network bandwidth may generate low video quality and negatively affect the user experience. Therefore, calculating the right QoS for a video will entirely change the user experience regarding video quality. How a viewer computes a desired QoS is an interesting human-factors and device-management problem, but it is outside the scope of this paper.

In general, this paper deals with the overall problem of delivering an original video in some format with a particular quality to a viewer who wants that video in a different format and with a specific QoS. There are two parts to this overall problem: a) transcoding or converting an original video to the desired format and quality while satisfying a delay constraint, and b) streaming that video to the viewer. The conversion may occur in its entirety before the streaming begins (e.g., Amazon Elastic Transcoder [9]), or it may be integrated into the streaming process (e.g., Akamai Media Content Delivery [10]). This paper focuses on the conversion process and, except for managing the delay; it doesn't matter whether the conversion occurs before streaming or in a pipeline with the streaming.

In a cloud-based video transcoding environment, the most important task is to transcode any video content in such a way that a) the quality of the transcoded video come as close as possible to the requested QoS quality level, b) the end-user can play the transcoded video smoothly without video freezes, c) the transcoded video can be played with the shortest start-up time [11]. A better video quality comes from satisfying the requested QoS level. Video freezes occur

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

due to unavailability of video frames while video startup time is the interval between the moment when the user selects the link and the moment when the video starts playing. The delay in start-up time is due to a delay in transcoding, streaming, or playing. In this paper, we focus on satisfying the requested QoS level while performing the transcoding process, which in turn helps in enhancing all the other factors.

Video transcoding for an on-demand video is a computer-intensive operation. Therefore, transcoding a large number of on-demand videos requires a large number of transcoding servers. Similarly, a large amount of disk space is required to store multiple transcoded versions for each source video [13]. Cloud computing provides computing and storage resources under the pay-per-use business model [12]. *Infrastructure as a Service (IaaS)* such as Amazon Elastic Computing Cloud (EC2)[1] provides the computing resources through Virtual Machines (VM) by dynamically creating scalable clusters of servers. Similarly, Amazon Simple Storage Service (S3)[2] provides the storage resources. EC2 can be used to virtually create scalable clusters of video transcoding servers that hold thousands of video transcoding services, and similarly S3 can be used to store both the original source video and the multiple transcoded versions. In a cloud environment, video transcoding can be performed in several different ways; for example, it is possible to map the entire video stream to a dedicated VM to transcode the entire stream, or split the video streams into smaller segments and independently transcode each of them in different VMs [13]. Regardless of which transcoding approach is used; guaranteeing the QoS level of the transcoded video requires selecting and composing the best video transcoding services from a pool of available ones, based on the viewer requirements and specifications to perform the transcoding process.

Software as a Service (SaaS) is a model where the customers can access the services via the internet without paying attention to how these services are maintained. The service providers are responsible for maintaining these services. In this paper, we assume that video transcoding functionalities are available as services, which are provided by service providers. A piece of software that converts a video from one coding format and quality to another format and quality is called a transcoding service or simply a transcoder. A cloud-based video delivery system may have thousands of different transcoders. If the delivery system has some transcoders that map from the requested video's original coding format to the viewer's desired format, which we called *compatible transcoders*, then the problem becomes one of best selection, where the system must choose a transcoder with a transcoding function whose output closely matches the desired QoS. If the delivery systems do not have any compatible transcoders, the system will need to convert the original video into an intermediate format(s) before converting into the required format. In this case, the problem becomes one of composing multiple best selections. In this paper we focus on a QoS-aware video transcoding service composition problem given input and output specifications.

Amazon Elastic Transcoder [9] is a video transcoding service provider that provides the transcoding functionality in the cloud. There are over 30 such providers today [14]. Many of the available video transcoding services provide the same functionality (i.e., format conversion), but with different QoS values. So, the challenge is how to select the best video transcoding services whose output closely matches the desired QoS, then compose these services together in a chain fashion, to satisfy the viewer requirements and specifications. Service composition is known to be an NP-hard problem and can be modeled as a multi-dimension, multi-choice, knapsack problem (MMKP) [15]. In this paper, we present a QoS-aware video transcoding service composition process in a distributed cloud environment where the video transcoding services are distributed in the cloud. In the composition process, we adapt the Simulated Annealing (SA) algorithm and the Genetic Algorithm (GA), which are generic probabilistic meta-heuristic learning algorithms to solve the combinatorial multi-objective optimization problem. They help in locating a very good state that is a good approximation to the global optimum of a given function in a large search space [16] [17].

Benefits of composing multiple transcoder in a chain fashion are: a) the final transcoded content would be exactly the same as if the transcoding had been done in one step, b) more complicated transcoding processes can be done even

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

with a limited number of transcoders, c) the computations could be distributed among different processing units, d) improved the client QoS level while properly scaling with number of clients, e) provide a clear separation between video contents and the transcoded ones, and f) generate more powerful applications with more complex functionalities because the functionality offered by individual services is limited [20].

Manual composition of services is time consuming, error prone, generally hard and not scalable. Therefore, many fully or partially automatic service composition approaches have been investigated [15]. Service composition involves creating composite services by combining different services to provide a new value-added service [15]. It does not only improve reusability of service components, but also enables rapid development of new complex applications and requirements [18] . Composing the best transcoders is an open problem to date; there are similar composition problems in other domains, like web-services, that have been heavily investigated [15]. Section II reviews this related work and categorize them into five different groups.

Section III formally defines fundamental concepts that are related to this problem domain, such as video and transcoding service. In Section IV, we present a general model for cloud-based video delivery system independent of any particular video transcoding composition algorithm. We use this model as a framework to evaluate our adaptation to the SA and GA algorithms, which we introduce and describe them in sections V and VI, respectively.
We select and adapt the SA and GA algorithms to the video transcoding service composition problem domain due to several reasons: a) SA and GA have been widely used by researchers to solve many optimization problems, b) SA and GA algorithms have low initial cost, c) SA avoids getting stuck in local optima [16], d) GA has been widely and efficiently used in a cloud-based service composition research [19].

Section VII discusses the evaluation results, which include two kinds of experiments. The first experiment focuses on sensitivity analysis of a) the SA algorithm's parameters and b) the GA algorithm's parameters. The results of this experiment show that for small-scale scenarios, the SA algorithm finds optimal solutions, while for medium- and large-scale scenarios; the GA algorithm outperforms the SA algorithm in finding near optimal solutions. In the second experiment, we focus on evaluating the proposed algorithms in terms of success ratio. Basically, we focus in this experiment on the quality of the results (i.e., how well the proposed algorithms generate results). This experiment shows that the SA algorithm is better than GA for small-scale scenario in terms of success ratio, while for medium- and large-scale scenarios, both the SA and GA generate close results in terms of success ratio. Finally, we discuss these results along with some recommendations and future directions from this preliminary study.

II. RELATED WORK

Revising literature related to composition process, we can classify the composition related work (but not limited to) into five groups, a) web-service composition related work, b) multimedia service composition related work, c) multimedia transcoding service composition related work, d) cloud-based service composition related work, and e) SA and GA based service composition related work. After that we discuss the limitations of some of the revising approaches.

A. *Web-Service Composition Related Work*

Web service composition has received considerable attention in recent years [15]. The problem of web service composition shares many of the same concerns found in the video transcoding composition problem. However, it is not easy to directly apply web service composition approaches into multimedia domain due to several reasons: a) the rich semantic and the complex internal structure of multimedia content itself, which is a combination of different forms (e.g., video, audio, or images), b) the size of the multimedia content makes the process of store, transcode, transport, and receive them very expensive, c) the dynamic characteristics of the multimedia applications, such as the continuous flow of multimedia streams, and c) the real-time processing requirements and the required QoS level [20].
Optimization algorithms like Linear Programming, Dynamic Programming, and Dijkstra-based algorithm m have been proposed as solutions to the web service composition problem [20]. Yan Gao et al. applied Dynamic Programming to solve the web service selection problem based on interface matching, and to dynamically select the optimum Web services for composite services [21]. However, their approach has some limitations, like the complexity of runtime decision. Rathore and Suman proposed a Local Selection and Local Optimization (LSLO) approach based

on linear programming for optimal candidate service selection for composition [22]. In spite of the advantages of their approach, there are also some limitations. For example, they considered only the positive QoS properties. Avoiding negative QoS properties may result in inappropriate selections that violate viewer expectations. Tongguang proposed QoS-aware web service selection and composition approach based on Particle Swarm Optimization (PSO). The author provided a sequential QoS utility function to calculate the overall global best solution for finding the best service candidates for each service class [23]. The author also evaluated the performance of the proposed approach by experiments. In spite of the efficiency of the PSO approach, he did not evaluate the QoS assurance and user satisfaction rate.

B. *Multimedia Service Composition Related Work*

Xiaohui Gu and Klara Nahrstedt proposed a fully decentralized service composition framework called SpiderNet [24]. This framework supports a distributed multimedia service composition process with a statistical QoS assurance. The prototype implementation and the simulation results showed the feasibility and the efficiency of the SpiderNet. The QoS properties cover both the application and the network levels. However, video transcoding composing process requires a special handling due to the sequential dependency of the video transcoding services. Moreover, they do not consider frame rate and frame size as QoS properties.

Moissinac proposed a semantic-based automatic discovery and composition approach for multimedia adaptation service. The author focused on developing a semantic description of the basic adaptation services [25]. Service composition based on semantic description of multimedia services is a good approach. However, it needs to be extended by adding a complete description to all known categories of multimedia services.

Li et al. proposed a heuristic algorithm named Greedy-EF to solve the multimedia service composition problem in overlay networks [18]. This composition process finds the proper service paths and routes the data flows through, so that the resource requirements and the QoS constraints of the applications are satisfied. The simulation results showed that their proposed approach can achieve the desired QoS assurance as well as load balancing in multimedia service overlay networks. However, they considered only the response time and the availability of the services as QoS requirements. In addition, they assumed that the user knows the service classes he will request. In our proposed approach, the user has to know only the requested video file, his QoS specifications, and the required format or codec. After that, our proposed system will find the best service path and route the data flows through, so his/her QoS specifications are satisfied. Moreover, their QoS properties covered just only the application level.

Hossain proposed QoS-aware service composition approach for distributed video surveillance in which he used the ant-based algorithm to solve the multi-constraints QoS routing problem [26]. He validated his proposed approach through implementation and simulation. The implementation results showed the quality of the transcoded results in terms of Peak Signal-to-Noise Ratio (PSNR), while the simulation results also showed the performance and satisfaction rates.

C. *Multimedia Transcoding Service Composition Related Work*

For multimedia transcoding composition problem, Hossain and Saddik proposed QoS-aware multimedia transcoding service selection process that uses Ant Colony Optimization (ACO) algorithm for selecting the most suitable multimedia transcoding services for the desired composition process [20]. They used only average transcoding delay and frame rate as QoS properties for the selection process. In spite of the dynamicity of their proposed algorithm, it has more overhead than the genetic and traditional Dijkstra algorithms. Moreover, it has a long convergence time.

Alberto et al. introduced how the Service Oriented Architecture (SOA) paradigm can be applied to context-aware multimedia communications [27]. In addition, they presented a scoring function for selecting codec for a case of selecting transcoding functions taking into account different quality assessment metrics. They defined a new quality analyzer model to assign a score to each transcoding service. We think that they have some limitations: a) their evaluation focused just only on the audio codecs, b) their composition process based on the quality or the compression ratio for each codec, while the general video transcoding composition process handle the selection of the best

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

implementation from different implementations of the same codec, c) their evaluation results are based on a single video/audio source with a specific configurations, evaluating their approach based on a set of video/audio sources with different configurations might help in generating more general results.

D. *Cloud-based Service Composition Related Work.*

Vaidas Giedrimas and Leonidas Sakalauskas proposed a simulated annealing and variable neighborhood search algorithms for automated software services composition in the cloud [28]. The experimental results showed that their proposed algorithm is able to approximate to best know solution in relatively short time. However, we think that their experiment is not enough to evaluate the efficiency and the effectiveness of the proposed approach. Moreover, they do not calculate the execution time.

Zhen et al. proposed an extensible QoS model to calculate the QoS values of service in cloud computing and a genetic-algorithm-based approach to compose services in cloud computing [29]. The experimental results showed that the proposed approach finds optimal solutions for small-scale scenarios. For larger-scale problems, it outperforms the integer programming approach. However, calculating the QoS values is done offline and the penalty factor in the fitness function is static.

E. *SA and GA-based Service Composition Related Work*

SA has been applied in many problem domains, such as the traveling salesman problem, job shop scheduling, multicast routing, service selection for composite web services, and many more [30]. To date, a few others have attempted to use simulated annealing in selecting multimedia transcoding services. G. Zhi-peng, *et al*. shows one of these applications [31]. They applied the simulated annealing-based genetic algorithm (QQDSGA) to efficiently select web services with excellent QoE. They defined a set of QoS criteria, which includes: service cost, execution time, availability and reliability. In addition, they defined a calculation model for each one of these criteria that considers the inconsistency of the attributes and the target direction. They also defined an objective function that makes the global QoS value for service composition the key element of the evaluation. For QoE, they have used five customer satisfaction degrees that indicate the general acceptability of the composite services based on customer expectation and environment. L. Arockiam and N. Sasikaladevi developed and compared the simulated annealing with the genetic algorithms as service selection algorithms [32]. They also concluded that the simulated annealing algorithm outperforms the genetic algorithm in selecting reliable services. Arockiam and Sasikaladevi developed a simulated annealing as a service selection algorithm for composite web services. Arockiam and Sasikaladevi considered reliability as a QoS parameter to maximize the non-functional characteristics of composite web services [33].

Amiri and Serajzadeh applied GA for QoS-aware web service composition; they considered the response time, execution cost, reputation, availability, and successful execution rate as QoS properties. They increased the performance of the algorithm and to escape from local optimum, they enhance the selection and crossover functions. The experiments showed the computation time of the algorithm is very low.

F. *Limitations of Existing Approaches*

Here we want to discuss the limitations of some of the existing composition approaches. *First*, most of the aforementioned approaches considered multimedia services in general without focusing on video transcoding services in particular. *Second*, some of the aforementioned approaches considered just only the QoS properties that are related to the application itself, or that are related to the network itself. However, few of them directly focus on the QoS properties that are related to video transcoding services such as video bit rate, frame size, frame rate, video transcoding delay, and aspect ratio. *Third*, some of the existing solutions are not readily applicable to the video transcoding composition problem due to the transcoding requirements and the inter-service dependency constraints between them. *Fourth*, to the best of our knowledge, there is no previous work for composing different video transcoding services in a distributed cloud environment based on application-specific QoS requirements using SA or GA algorithms.

III. PRELIMINARY DEFINITIONS

In this section we formalize the QoS-aware video transcoding service composition problem by introducing some definitions for the basic concepts that are related to the problem domain. Let’s start by defining the video.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

Definition 1: **Video**. A video $\boldsymbol{v}$ represents a media file, which is a combination of audio and image streams. A video $\boldsymbol{v}$ has a format *v.f* and video characteristics, and represented by $\boldsymbol{v.c}$. A video's *v.c* includes the frame size ($\boldsymbol{v.c.fs}$), frame rate ($\boldsymbol{v.c.fr}$), and video bit rate ($\boldsymbol{v.c.br}$). Frame size can be further described in term of width *(v.c.fs.w)* and height (*v.c.fs.h*). When considering video quality, it is useful to consider a frame size's aspect ratio, which is the width divided by the height. For convenience, we will reference a video's aspect ratio as (*v.c.fs.ar*). Technically, a video's characteristics also include an audio bit rate or audio sample rate and size, but we exclude them from our discussion here because they do not contribute significantly to the composition problem.

Definition 2: Quality of Service (QoS). A QoS, denoted in formulas as *qos*, is a specification consisting of frame size (*qos.fs*), frame rate (*qos.fr*), bit rate (*qos.br*) and a tolerated delay (*qos.d*). We say that a *v.c* matches a *qos*, written as *v.c* ≈ *qos*, if and only if *v.c.fs* = *qos.fs*, *v.c.fr* = *qos.fr*, and *v.c.br* = *qos.br*. As with video characteristics, we can further describe the desired frame size in a QoS in terms of width (*qos.fs.w*), height (*qos.fs.h*), and aspect ratio (*qos.fs.ar*). We scope our algorithms to these QoS properties. However, other parameters can be added such as energy or computation consumption.

Definition 3: Video Quality and Degradation. Video quality is a measurement of how a transcoded video looks compared to the original one or compared to a standard if the original video is unavailable. This measurement can be evaluated either objectively or subjectively [34]. Objective evaluation techniques are mathematical models that a computer program can automatically calculate using a predefined metrics such as Mean Square Error (MSE). Subjective evaluations, on the other hand, require expert judgments. We represent a video's quality as *v.quality*. For any original video, *v*, its *v.quality* will be 100%. Conversely, we represent a video's degradation as *v.degradation*, with 0% meaning no lost in quality. For any original video, *v*, its *v.degradation* will be 0%.

Definition 4: Viewer's Request. A viewer's request, $\boldsymbol{Q}$, consists of a required video format, $\boldsymbol{Q.f}$, and a desired QoS, $\boldsymbol{Q.qos}$. Note $\boldsymbol{Q.f}$ is a hard constraint that must be satisfied, whereas $\boldsymbol{Q.qos}$ is a soft constraint. In this paper, we will use the viewer requirement to represent the required video format and the viewer specification to represent the desired QoS level. The video delivery system must provide a video in the $\boldsymbol{Q.f}$ format and should come as close to meeting the $\boldsymbol{Q.qos}$ as possible. Each viewer request has a cost associated with it and can be represented as $\boldsymbol{Q.cost}$.

Definition 5: Video Transcoding Function. We define a video transcoding function as a data processing function $t(x \xrightarrow{f_1,f_2c_1,c_2} y)$ that converts input video x, where $x.c = t.c_1$ and $x.f = t.f_1$, to generate an output video y, where $y.c = t.c_2$ and $y.f = t.f_2$. A video transcoding function takes time to execute and therefore introduces a delay into the streaming of a video. We represent the anticipated delay of *t* as *t.d*. For convenience of comparing the expected output of a transcoding function *t* with a requested QoS (*Q.qos*), we combine the *t.c*$_2$ and *t.d* properties into an aggregate property, called *t.qos*. Each video transcoding function t has a cost associated with it, $t.cost$. This cost represents the cost that we need to pay in terms of QoS properties when the transcoding system uses this function. It is obvious that the transcoding system should select the transcoding function that has a cost value (i.e., $t.cost$) that is close to the cost value of the viewer request (i.e., $Q.cost$).

Definition 6: Video Transcoder, aka, transcoding service. As mentioned earlier, a video transcoder is a piece of software that implements one or more video conversions, i.e., video transcoding functions. We restrict a transcoder such that all of the functions it implements have the same output format (*f*) and the same video characteristics (*c*). This restriction simplifies the discussion to follow without loss of generality. Furthermore, to simplify the selection and the composition of transcoders, we assume that a transcoder has to handle all possible input video characteristics (*c*) for the format that accepts. This assumption is actually consistent with most transcoder implementations. Given the above simplification and assumption, we can formally define a transcoder *T* as the implementation of a family of functions: $t_1, t_2, \ldots t_n$, where $t_i \in T, (1 \leq i \leq n)$ with an input format (*T.fin*), an output format (*T.fout*), and output video characteristics (*T.c*).

Definition 7: Compatible Transcoders. Given an original video *v* and viewer request *Q*, a transcoder *T* is a compatible transcoder for *v* and *Q*, which we can represent it as $comp(v, Q, T)$, if and only if *v.f* = *T.fin* and *Q.f* = *T.fout*. For example, assume that delivery system has 1000 video transcoding functions that convert from H.264 to

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

WMV1, but with different output video characteristics. If an original video had a H.264 format and the viewer wanted a WMV1 format, all 1000 video transcoding functions would be compatible transcoders.

IV. General Cloud-based Video Delivery System

Fig. 2 shows a use case for cloud-based video delivery system where user A can upload any video to the cloud while user B can request and play that video using any device regardless of the original video format. If the original video format is not supported at the user B's device or playback software, this video must be transcoded to a new format that the user B's device supports. To allow any user to watch any cloud-based video from any kind of display device over any type of network, the cloud-based video delivery system must be able to transcode any video. Video transcoding is a computationally intensive operation and due to the limited computational power for the end-user's devices; it may not be suitable to perform the transcoding process at the end-user side. Therefore, it is usually performed at the server-side [13].

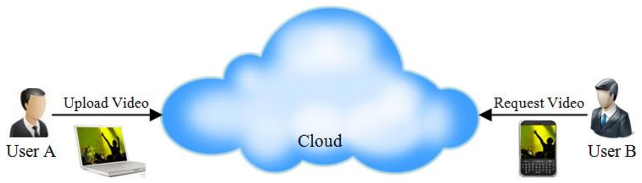

Fig. 2. A use case for cloud-base video delivery system

On-demand video transcoding has many challenges: 1) it must be done on-the-fly in the real-time, 2) it must meet the viewer's requirement and specification. In this paper, we will focus on the second challenge. Guarantee the viewer's requirement and specification requires selecting the best video transcoding services, from a pool of available ones that satisfy the required QoS level to perform the transcoding process.

If the requested video is not available in the requested format, the cloud-based video delivery system should transcode the original video to the requested format and then streams the transcoded video to the end-user. If the cloud-based video delivery system does not have any compatible transcoders, the system will need to convert the original video into an intermediate format(s) before converting it into the required format. In this case, discovering these multiple transcoders and composing them into a chain fashion to meet the viewers' requirement and specification is the challenge. We refer to this chain as a composition chain and we can formally define the composition chain as follows:

Definition 8: Composition Chain (CC). Given a viewer request, Q and a requested video, v. The Composition Chain is a set of transcoders $CC = \{T_1, T_2, \dots, T_i\}$, such that the first transcoder in this chain accepts the requested original video as an input and the last transcoder in this chain convert the requested video to the requested format as an output. In other words, $T_1.fin = v.f$ and $T_2.fin = T_1.fout$ and so on until $T_i.fin = T_{i-1}.fout$ and $T_i.fout = Q.f$.

The sequential format dependency in this composition chain should be satisfied and meets the viewer's requirement. To simplify this process, we consider each conversion or transcoding process as a level of transcoding, so, when converting from MPEG-2 to H.264, we consider this as a one-level conversion or one-level transcoding. Two-level transcoding requires transcoding the video into an intermediate format before transcoding it into the requested format. To meet the real-time video transcoding requirements, finding the composition chain should be done on-the-fly in the real-time in addition to the transcoding and streaming processes. Fig. 3 shows an example of a three-level video transcoding chain. Deciding the maximum number of levels the transcoding system might have (i.e., the chain's length) is an interesting cost-efficiency trade off problem, but is outside the scope of this paper and we will not consider it as a bottleneck.

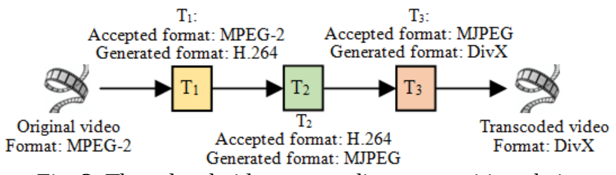

Fig. 3. Three-level video transcoding composition chain

Based on the example given in Fig. 3, if the format of the original video is MPEG-2 and the end-user wants this video in DivX format and assume that the cloud-based video delivery system has just the following transcoders T_1, T_2, and T_3 as follows: T_1 transcodes from MPEG-2 to H.264, T_2 transcodes from H.264 to MJPEG, and T_3 transcodes from MJPEG to DivX. Each one of these transcoders (i.e., T_1, T_2, and T_3) might have hundreds or thousands of video transcoding functions with different QoS values. Because the format of the requested video (i.e., MPEG-2) is different than the requested format (i.e., DivX), the cloud-based video delivery system must transcode the original video to the requested format. It is obvious that the cloud-based video delivery system that we provide in Fig. 3 has no compatible transcoder for the requested video and the requested format (i.e., no transcoder that directly convert from MPEG-2 to DivX). Therefore, the cloud-based video delivery system should discover the transcoders that meet the viewer requirements (i.e., the hard constraints). Based on the discovered transcoders, there will be three transcoders that participate in the composition chain in a sequential order, which means that there will be a three-level transcoding process as shown in Fig. 3.

In the three-level video transcoding composition chain that Fig. 3 presents, the original video is send to the T_1 in which it transcodes the original video to generate a new video in a H.264 format, while T_1 performs the transcoding process, it starts sending the transcoded frames to T_2 to transcode these frames to MJPEG format, while T_2 performs the transcoding process, it starts sending the transcoded frames to T_3. Finally, when T_3 starts receiving the transcoded frames, it performs the transcoding process to convert these frames to DivX, while T_3 performs the transcoding process, it starts sending the transcoded frames to the streaming system that streams these frames to the end-user’s device. This process should be done on-the-fly in real-time. For best utilizing, each transcoder save its transcoded streams into a virtual storage device for later on requests. Assume that the cloud-based video delivery system has hundreds or thousands of each one of the above transcoders (i.e., T_1, T_2, and T_3) and each one has different QoS specifications. The problem now is how we can select and compose these transcoding functions together in a way that guarantee the required QoS level during the transcoding process and guarantee the required QoS level of the video that is send to the end-user’s device. We will start describing the solution by first, describing a general model for on-demand cloud-based video distribution system in the following paragraphs.

Fig. 4 shows a general model for on-demand cloud-based video distribution system, which contains cloud-based video delivery system. Cloud-based video delivery system consists of three main sub-systems: a) cloud-based service management system, b) cloud-based video transcoding system, and c) cloud-based video streaming system. In this paper, we focus on the cloud-based service management system and, specifically, on the service composition process.

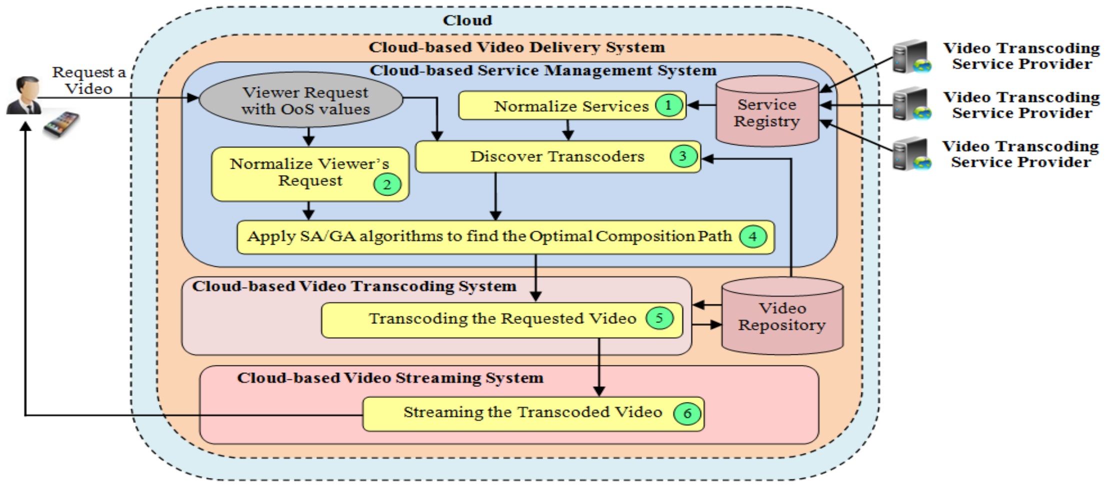
Cloud
Cloud-based Video Delivery System
Cloud-based Service Management System
Request a Video
Viewer Request with QoS values
Normalize Services 1
Service Registry
Video Transcoding Service Provider
Video Transcoding Service Provider
Video Transcoding Service Provider
Normalize Viewer's Request 2
Discover Transcoders 3
Apply SA/GA algorithms to find the Optimal Composition Path 4
Cloud-based Video Transcoding System
Transcoding the Requested Video 5
Video Repository
Cloud-based Video Streaming System
Streaming the Transcoded Video 6

Fig. 4. A general model for on-demand cloud-based video distribution system

In this model, all the available transcoders and their transcoding functions, which are provided from the service providers, are captured in a Service Registry. Step 1, which only needs to occur once after the Service Registry is

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

loaded, normalizes $t_j.qos.q_i$ for each video transcoding function, t_j, in each transcoder, T, in the Service Registry, using the standard deviation normalization, $norm(t_j.qos.q_i)$ as in [35] as follows:

$$norm(t_j.qos.q_i) = \begin{cases} 2, \ if \ (t.qos.q_i - \mu(T.c.q_i)) > 2 * \delta(T.c.q_i)) \\ 0, \ if \ ((t.qos.q_i - \mu(T.c.q_i)) < -2 * \delta(T.c.q_i)) \\ \left(\frac{t.qos.q_i - \mu(T.c.q_i)}{2*\delta(T.c.q_i)}\right) + 1, \qquad otherwise \end{cases} \tag{1}$$

where $qos.q_i$ is the value of the QoS property i, $q_i \in qos$. The QoS properties are *qos.fs.w*, *qos.fs.h*, *qos.fs.ar*, *qos.fr*, *qos.br*, and *qos.d*. $\mu(T.c.q_i)$ and $\delta(T.c.q_i)$ are the mean and the standard deviation of the QoS property q_i for a transcoder T, respectively. Standard deviation normalization transforms data in more efficient way than decimal normalization, using means and standard deviation by picking up the most balanced instance instead of one that is better in one QoS, e.g., delay, and weak in another, e.g., frame rate [35]. We calculate the mean value for each QoS property i for each video transcoder T that contains m video transcoding functions as follows:

$$\mu(T.c.q_i) = \frac{1}{m} * \sum_{j=1}^{m} (t_j.qos.q_i) \tag{2}$$

Also, we calculate the standard deviation value for each QoS property i for each video transcoder T that contains m video transcoding functions as follows:

$$\delta(T.c.q_i) = \sqrt{\frac{1}{m} \sum_{j=1}^{m} ((t_j.qos.q_i) - \mu(T.c.q_i))^2} \tag{3}$$

For QoS property that are better with smaller values (e.g., delay), $norm(t_j.qos.q_i)$ is further transformed into $norm'(t_j.qos.q_i)$ as follows [35]:

$$norm'(t_j.qos.q_i) = 2 - \ norm(t_j.qos.q_i) \tag{4}$$

After calculating the normalized value for all the QoS properties for each video transcoding function t_j that has n QoS properties, we calculate the cost for each video transcoding function as follows:

$$t_j.cost = (\tfrac{1}{n} * \textstyle\sum_{i=1}^{n} norm(t_j.qos.q_i))) \tag{5}$$

It is obvious that we complete this step before the viewer submits a request. So, we decoupled its computation time from the computation time of the proposed algorithms.

The normalization step is a very important step for two reasons: a) each QoS property has different unit; for example, delay in millisecond while bit-rate in kilobits per second, and b) some properties are better with smaller value while others are better with bigger values.

Every time a viewer submits a request, Q, for specific video v, the system needs to normalize $Q.qos.q_i$ using (1). This process is Step 2 in Fig. 4. We calculate the viewer request's cost, $Q.cost$, as follows:

$$Q.cost = (\frac{1}{n} * \sum_{i=1}^{n} norm(Q.qos.q_i))) \tag{6}$$

where n is the total number of QoS that are available from the viewer request.

Step 3 uses $Q.f$ and the original requested video, $v.f$ to discover a set of transcoders that are available in the Service Registry and satisfy the viewer's requirements. In other words, service discovery step discovers the composition chain.

Discovering the composition chain does not mean finding the actual video transcoding functions. Basically, it helps us in finding a set of transcoders that meet the viewer requirement. In addition satisfying the viewer specification is very important step and we have to guarantee this. Finding the best composition of the discovered video transcoding functions to satisfy the viewer specifications requires discovering all the possible combinations of these candidate video transcoding functions, which can be very expensive in terms of computation time. Therefore, we propose the SA and GA algorithms as candidate solutions to solve this problem. Step 4 is the most relevant to this paper; in which we apply the SA or GA algorithms to find the best set of video transcoding functions that satisfy the viewer specifications.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

Once the best video transcoding functions that satisfy the viewer requirement and specifications have been discovered and identified, then the processes, represented by Steps 5 and 6, perform the actual transcoding and then streaming the transcoded video to the end-user's device.

V. Video Transcoding Composition Process

In order to support real-time multimedia applications such as video/audio streaming, video-on-demand or video conferencing over any type of network; the desired QoS should be satisfied in addition to the basic video format conversion. Notice that Step 4 shown in Fig. 4 does not completely describe the composition process. In this section, before start explaining the composition process, we want to assume that the cloud-based video delivery system is running in the cloud. Hence, the cloud-based video transcoding system that holds hundreds or thousands video transcoding functions are running in the cloud as well. In addition, we assume that these transcoding functions are services running in a number of virtual machines in the cloud.

Let us consider a video transcoding function network as a directed acyclic graph, $\mathrm{G(T, L)}$; so, we can formally define this graph as follows.

Definition 9: Directed Acyclic Video Transcoding Function Graph $\boldsymbol{G(T, L)}$. We can consider a video transcoding network as a directed acyclic graph $G(T, L)$, where $G.T = \{t_1, t_{2,} \dots, t_n\}$ denotes a set of nodes that represent the video transcoding functions, and $G.L = \{l_1, l_2, \dots l_m\}$ denotes a set of links that connect these video transcoding function nodes. We also can define the total number of video transcoding functions nodes that are available in the graph G by $|T| = n$, while the total number of links that are available in the graph G is $|L| = m$. Each link, l_k where $(1 \geq k \geq m)$ can be indicated by an ordered pair of video transcoding function nodes (i, j) and can be represented as l_{ij} where l_{ij} is said to originate from node i and terminated at node j. Therefore, for l_k or l_{ij}, $l_{ij}.source = i$ and $l_{ij}.destination = j$. We will use l_k and l_{ij} interchangeably if there is no confusion. Each link l_k is associated with a positive real number $l_k.cost$ that represents the cost of the desired video transcoding function node. Therefore, we can calculate the cost of the link l_k as follows:

$$l_k.cost \text{ } or \text{ } l_{ij}.cost = t_j.cost \tag{7}$$

Fig. 5 shows a sample directed acyclic graph of the video transcoding functions with four transcoders $T = \{T_1, T_2, T_3, T_4\}$ that have $\{\mathrm{a, b, c, d}\}$ video transcoding functions, respectively. Fig. 5 also shows three different views of the proposed QoS-aware video transcoding service composition process: a) the abstract view, b) the graph view, and c) the composition view. The abstract view shows all the available video transcoders that are available in the system. This view represents the functional sequence dependency between all these transcoders. It also shows the video transcoding composition chain.

The graph view represents a directed acyclic graph that consists of distributed video transcoding functions. Each node in this graph represent a video transcoding function which has several input and output ports for receiving input messages and sending output messages, these messages represent the video streams. Based on Fig. 5 and assume that the cloud-based video delivery system has just these four transcoders (i.e., $\{T_1, T_2, T_3, T_4\}$). T_1 converts from MPEG-2 to H.264, T_2 converts from H.264 to MJPEG, T_3 converts from MJPEG to DivX, and T_4 coverts from DivX to WMV1. Each transcoder contains a set of video transcoding functions that have different QoS values; T_1 has a video transcoding functions, T_2 has b video transcoding functions, T_3 has c video transcoding functions, and T_4 has d video transcoding functions. S and D represent the source and the destination nodes, which represent the original video content and the transcoded one based on the viewer request, respectively. All of the links in Fig. 5 have cost that is calculated using (5). For simplicity and presentation perspective, we assign the cost for few of them to keep the graph simple to read and follow. For example, the cost for transcoding the original video using $t_{1.1}$ transcoding function in level 1 is 0.7 and the cost of transcoding the original video using $t_{1.1}$ transcoding function in level 1 and $t_{2.1}$ transcoding function in level 2 is (0.7 + 0.4) =1.1 and the cost of transcoding the original video using $t_{1.1}$ transcoding function in level 1 and $t_{2.1}$ transcoding function in level 2 and $t_{3.2}$ transcoding function in level 3 is (0.7 + 0.4 + 0.9) = 2.0.

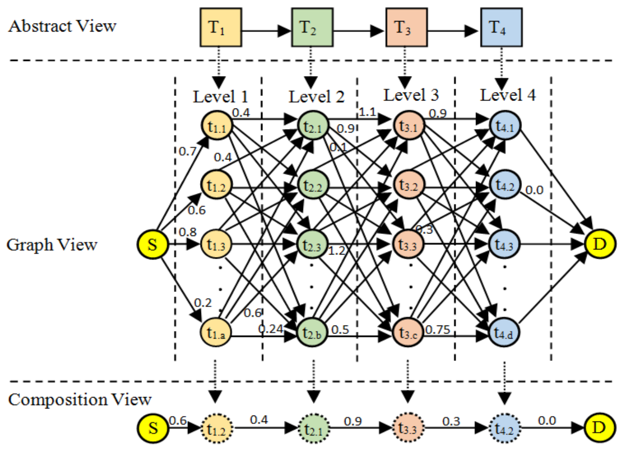

Fig. 5. Different views of the QoS-aware video transcoding composition process.

The last view is the composition view, which represents the video transcoding composition path from S to D that includes a set of video transcoding functions in between. These transcoding functions perform the real transcoding in a chained fashion as we described above. Finding the composite view refers to finding a path in a graph from the source node to the destination node based on viewer requirements. Based on the above example, the cost of the path in the composition view: $\{S, t_{1.2}, t_{2.1}, t_{3.3}, t_{4.2}, D\}$ is (0.6+0.4+0.1+0.3+0.0) = 1.4. We can formally define the video transcoding composition path as follows:

Definition 10: Video Transcoding Composition Path $\boldsymbol{p(S, D, G)}$. Video transcoding composition path $p(S, D, G)$ in a directed acyclic graph G from the source node S to the destination node D is a set of links L represented by $p.L$ where $L = \{l_1, l_2, \dots, l_r\} \in G.L$, and a set of nodes represented by $p.N$ where $N = \{t_1, t_2, \dots t_n\} \in G.T$ and $l_1.source = S$ and $l_r.destination = D$ and $(r > 1)$. Each path has a cost, $p(S, D, G).cost$. For simplicity, we can represent the cost by $p.cost$ that represents the aggregate or the objective function of the QoS values and we can define it as follows:

$$p(S, D, G).cost = p.cost = \sum_{i=1}^{r} l_i.cost \tag{8}$$

Based on the above graph view, we apply one of the proposed algorithms (SA or GA) to find the best composite view. Finding the best composition path refer to finding the best path that meets the viewer QoS requirements and specification. This path should minimize the absolute difference between the viewer QoS values and the cost of all the discovered paths. We can formally define the best QoS-aware video transcoding functions composition path as follows:

Definition 11: The Best QoS-aware Video Transcoding Functions Composition Path $\boldsymbol{p^{'}(S, D, G)}$. Given a viewer request Q and a set of possible video transcoding function paths $P = \{p_1, p_2, \dots, p_s\}$ from source node S to destination nodes D, we can define the best QoS-aware video transcoding functions composition path as a path that has the lowest absolute difference value, among all other possible paths, between its cost value and the cost value of the viewer request. Formally, we can define the best video transcoding composition path $p' \in P$ as follows:

$$\begin{gathered} p^{'}(S, D, G) \rightarrow \forall p_i \; where \; p_i \in P \; and \; (1 \le i \le s), \\ \exists p_j \; where \; (1 \le j \le s) \; and \; p_j \in P \; such \; that \\ abs(p_j.cost - Q.cost) \le abs(p_i.cost - Q.cost)^{\ddagger} \end{gathered} \tag{9}$$

Table 1 shows examples of different viewer requests and the composition chain for each one of them based on the transcoders that are available in Fig. 5. We assume that the original video format is MPEG-2. In this paper, we focus on a composition chain that contains more than one transcoder.

Table 1. Examples of some viewer requests

Original Video	Requested Format	Composition Chain
MPEG-2	WMV1	$\{T_1, T_2, T_3, T_4\}$
	DivX	$\{T_1, T_2, T_3\}$
	MJPEG	$\{T_1, T_2\}$

In general, we can summarize the video transcoding service composition process into the following steps:

Step 1: Calculate the cost of each video transcoding function using (5) and calculate the viewer request cost using (6).

Step 2: Analyze the viewer request and discover the m transcoders as shown in the abstract view in Fig. 5.

Step 3: Discover the video transcoding functions for each transcoder. Each transcoder T_i might have n_i transcoding functions.

Step 4: From the transcoding function graph, we have $n_1 * n_2 * n_3 * \ldots * n_m$ available composition paths.

Step 5: Apply one of the proposed algorithms (SA or GA) to find the optimal composition path. This done by discovering the possible paths and calculate the cost of each discovered path using (8). Then, calculate the difference between the cost of each discovered path and the viewer's cost and then select the path that has the minimum difference value using (9) as the optimal composition path.

VI. Simulated Annealing Algorithm for Video Transcoding Composition Process

In this section, we discuss how we adapt the SA algorithm into the video transcoding service composition problem domain. SA algorithm models the annealing process in metallurgy. It involves heating up the material and then cooling it in a controlled way that helps in increasing the size of its crystals and reducing their defects, thus reaching a state with lower system energy [16].

We divide the SA algorithm into six parts: a) parameters configuration, b) the initial solution, c) the cost function, d) the neighboring function, e) the acceptance probability, and f) the cooling schedule. The following sub-sections describe these parts in more details.

A. *Parameters configuration*

In this part, we configure the parameters of the SA algorithm, mainly, the temperature and the cooling rate. Usually, the algorithm starts with a high temperature value, and then this value starts decreasing based on the cooling schedule to reach the lowest temperature. The cooling schedule is user defined and it is usually low enough to reflect the slow cooling. Function 1 describes this configuration process.

B. *The Initial solution*

After configured the parameters, we get to the initial solution. In the initial solution, we randomly select a video transcoding function from each level in the directed video transcoding function graph. We ensure the feasibility of the initial solution by adding the source and the destination to it. Function 2 describes the initialization process. A feasible solution is a solution or a composition path that satisfies the hard constraints based on the viewer requirements.

C. *The cost function*

Function 3 describes how we calculate the cost of any path. Basically, the cost of any path is the summation of cost of the links that are participating in that path. In essence, we use (7) to calculate the cost of any path.

D. *Generating the neighboring path*

Generating the neighboring solution or path based on the current one is one of the main parts in the SA algorithm. Function 4 shows the neighbor function in which we randomly select a neighbor path to the current one from the directed acyclic video transcoding function graph. This function randomly selects a video transcoding function (i.e., node) from each level in the graph and then composing them together to generate a feasible path. We ensure the feasibility of the new neighboring path by adding the source and the destination nodes to it. Therefore, the new neighboring path satisfies the viewer request's hard constraints.

E. *The acceptance probability*

After generating the neighboring path, we use the acceptance probability function to calculate the acceptance probability based on the combination of the currentCost, newCost, userCost, and the $temp$. This function compares the cost of the two paths with respect to the userCost. We described above how we calculate these costs. The acceptance probability function helps the SA algorithm to intelligently accept worse solutions and escape from the local optima that are worse than the global one [36]. The acceptance probability is a non-negative real number between 0 and 1. Function 5 describes the acceptance probability function.

F. *The cooling schedule*

The cooling schedule is a schedule that represents the decreasing value of the temperature. The temperature value should be decreased slowly and it depends on a pre-defined user value. Line 16 in Algorithm 1 shows the cooling schedule that we used.

Now, we want to combine all the above six parts together in one part to describe the SA algorithm. Algorithm 1 describes the overall SA algorithm. At the first step, we call the conFig.(temp$_0$, γ) function (see Function 1) to assign an initial values for the temperature and the cooling rate. This represents part 1 in the Algorithm 1. After that, we initialize the initial path by calling the Initialization(G, $Q.f$, $v.f$) function (see Function 2). Then, assign the initialPath to the currentPath. After that we calculate the currentCost by using the getCost(currentPath) function (see Function 3). Then, we declared the bestPath and assign the currentPath to the bestPath and the currentCost to the bestCost. This represents part 2 in Algorithm 1. In part 3 in Algorithm1, the loop part, we use the temp as a loop condition. Inside the loop, first we use neighbor(G, p) (see Function 4) to find a neighbor path and assign it to newPath. Then we calculate its cost and assign it to the newCost. Then we check if the value that is returned by calling the $acceptProb(temp, currentCost, userCost, newCost)$ function is greater than a random number between 0 and 1, if so, we accept the newPath as the currentPath. Then we compare the currentCost with the bestCost, if the absolute difference between the currentCost and the userCost is less than the absolute difference between the bestCost and the userCost, we assign the currentPath to the bestPath and the currentCost to the bestCost. The last step in the loop part is the cooling step in which we decrement the temperature according to the cooling rate. Finally, we repeat the third part until the loop's condition becomes false, and then, at the end, we return the bestPath as an optimal path.

Function 1. Parameters configuration: $config(temp_0, \gamma_0)$

Output: initial value for the temperature, $Temp$ and the initial value of the cooling rate, γ.

```
1.   $temp$= $temp_0$;
2.   $\gamma = \gamma_0$;
```

Function 2. Initial solution: Initialization($G, Q.f, v.f$)

Input:

- Complete Video transcoding function graph, $G(T, L)$.
- Viewer requested format (i.e., $Q.f$), to capture the destination node, D.
- Original Video format (i.e., $v.f$), to capture the source node, S.

Output: An initial path, p that connects source with destination,.

```
1.   initial path $p$ = null;
2.   video transcoding function $t$ = null;
```

Function 5. The acceptance probability: $acceptProb(temp, currentCost, userCost, newCost)$

Input:

- Temperature, temp.
- Current cost, currentCost.
- User cost, userCost.
- New cost, newCost.

Output: random value between 0 and 1 represent the acceptance probability, accept.

```
1.   accept = 0;
2.   if (|newCost - userCost |<| currentCost - userCost |)
3.       accept = 1.0;
4.   else
5.       accept = $\exp^{((currentCost - userCost) - (newCost - userCost)/temp)}$
6.   end if
7.   return accept;
```

```
3. Add $S$ to $p$;        // based on $v.f$
4. for (i←1 to G.numberOfLevels) do
5.     $t$ = randomly select t from $G.T(i)$
6.     add $t$ to $p$
7. end for
8. add $D$ to $p$        // based on $Q.f$
9. return $p$
```

Function 3. Getting path cost: getCost (p)

Input: Current path, p.
Output: non-negative real number shows the cost of the current path.

```
1. cost = 0;
2. for (i←1 to p.size()) do   //p.size = number of links in p path
3.     cost +=p.getLink(i).cost;        // see (7)
4. end for
5. return cost;
```

Function 4. Generating neighbor path: neighbor(G, p)

Input:
- Complete video transcoding function graph, $G(T, L)$.
- Current path, p.

Output: neighbor path, neigborePath.

```
1. neigborePath = null;
2. video transcoding function t = null;
3. Add p. S to neigborePath
4. for (i←1 to G.numberOfLevels) do
5.     t = randomly select t from G.T(i)
6.     add t to neigborePath;
7. end for
8. add p. D to neigborePath;
9. return neigborePath;
```

Algorithm 1: Simulated Annealing Composition Algorithm: $SA(G, Q.cost, Q.f, v.f)$

Input:
- Directed acyclic video transcoding function graph, $G(T, L)$.
- User cost, $Q.cost$, and the requested format, $Q.f$.
- Original Video format, $v.f$.

Output: Near optimal video transcoding composition path.

```
Part 1: Configuration:
1. calling config(temp$_0$, $\gamma_0$)          /* see Function 1 */
Part 2: Initialization
   /* see Function 2 */
2. currentPath = Initialization($G, Q.f, v.f$)
3. currentCost = getCost(currentPath);/* see Function 3 */
4. bestPath = currentPath;
5. bestCost = currentCost;
Part 3: Loop
   while (temp > 1)
6.      newPath = neighbor($G, p$);    /* see Function 4 */
7.      newCost = getCost(newPath);
8.      if ( $acceptProb(temp, currentCost, Q.cost, newCost)$
                 > rand ())        /* see Function 5 */
9.          currentPath = newPath;
10.         currentCost = newCost;
11.     end if
12.     if (|currentCost – Q.cost | < | bestCost - Q.cost |)
13.         bestPath = currentPath;
14.         bestCost = currentCost;
15.     end if
16.     temp = temp * (1- $\gamma$);
17. end while
18. return bestPath;
```

VII. GENETIC ALGORITHM FOR VIDEO TRANSCODING COMPOSITION PROCESS

In this section, we discuss how we adapt the GA algorithm into the video transcoding service composition problem domain. GA is a meta-heuristic searching algorithm used to generate useful solutions to the optimization problems in a large search space. It depicts the biological evaluation of genes and chromosomes. GA works on a search space called population that contains a set of randomly generated solutions called chromosomes. After £ iterations, this population is evolved toward a better state. Each chromosome has a fitness value that defines the quality of the solution. GA uses adaptive search strategy in which it evolves the population by finding a set of best solutions, and then performs the crossover operation to create a new generation or population. The weaker candidates get less chance to be in the new generation while the best candidates from the previous generation are moving to the new generation. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached.

In video transcoding service composition problem domain we used the graph to represent the search space. The population represents a set of composition paths. Each chromosome represents a composition path. We calculate the fitness of each chromosome by using (7). Similarly, we can divide the GA algorithm into three parts: a) configuration part, b) initialization, and c) the loop part, in the following sub-sections, we will describe each part in details.

A. *Parameters configuration*

In this part, we configure the parameters of the GA algorithm, mainly, the population size and the number of iterations. These parameters are user defined and their values depend on the problem domain and size. Function 6 describes this configuration process.

B. *The Initialization*

After configured the parameters, we get to the initialization step. In this step, we create a population which contains a set of video transcoding composition paths that are stored in an array structure. The size of this array depends on the population size which is a user defined value from the previous configuration step. Each element in this array represents a feasible video transcoding composition path. Function 7 describes the initialization process.

C. *The Loop*

In this part, we used a for-loop structure that iterate £ times, the value of the £ is a user-defined value based on the previous configuration step. For each iteration, we evolve the population by generating a new population from the previous one. Function 8 describes the evolving process. Evolving the population requires randomly selecting two parents and performs the crossover operation to generate a new child. Each one of the parents and the child represents a composition path. Selecting each parent requires generating a new population that contains n randomly selecting composition paths (n is 5 in our proposed algorithm), and then finds the path that has the highest fitness value (i.e., the path that has the lowest cost value). Function 10 describes the process of how the GA randomly creates a path. We calculate the fitness value for each composition path by taking the absolute difference value between the existing path's cost and the viewer request's cost. Function 9 describes how we calculate the fitness value.

After generating new parents, we perform the crossover process. This process requires selecting a random number between 0 and the path size, then dividing each parent into two parts based on the selected random number. After that, merges the first part of parent$_1$ with the second part of parent$_2$ to generate a new child composition path. Function 11 describes this crossover method and Fig. 6 depicts the crossover process. After generating a new child from his parents, then we save this child in the current population and then repeat the whole process $popSize$ times. The crossover process helps in creating a new path that might have a better fitness value than his parents.

Algorithm 2 shows our adaptation to the GA algorithm in which it iterate £ times to find a near-optimal path. As we mentioned above and in each iteration, we evolve the current population until the termination condition is satisfied, finally, we return the best video transcoding path as an output from the GA algorithm.

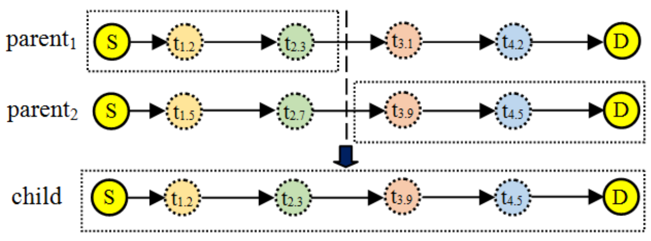

Fig. 6. Crossover process

Function 6. Parameters configuration: config($popSize_0$, £$_0$)

Output: initial value for the population size, $popSize$ and the number of iterations, £.

```
1.   $popSize = popSize_0$;
2.   £ = £$_0$;
```

Function 7. Initialize population: InitiaPop($popSize$)

Function 10. Get random composition path: $getRandomPath(pop)$

Input: population, pop.
Output: Random composition path.

```
1.   create new population, $newPop$;
2.   create an empty array of size 5;
3.   fill this array with randomly 5 composition paths
     selected from $pop$;
1.   return $getFittest(newPop)$;        /* see Function 9 */
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

```
Input: population size, $popSize$.
Output: instance of population.
1. Create new Population;
2. Create an empty array, $path$ of size popSize.
3. Fill this array with popSize random composition paths;
```

```
Function 8. Update population: $evolvePop(pop)$
Input: population, $pop$.
Output: population, $pop$.
1. for (i←1 to $pop.size$) do
               /* See Function 10 */
2.     $parent_1 = getRandomPath(pop)$;
               /* See Function 10 */
3.     $parent_2 = getRandomPath(pop)$;
               /* See Function 11 */
4.     $child$ = crossover($parent_1, parent_2$);
5.     $pop.path[i] = child$;
6. end for
7. return $pop$;
```

```
Function 11. Crossover operation: crossover($parent_1$, $parent_2$)
Input: random paths: $parent_1$ and $parent_2$.
Output: A new path builds from these two input paths.
1.  create an empty child path;
2.  $int\ position = rand(0, parent_1.size - 1)$;
3.  if ($position < child.size$)
4.      for ($i \leftarrow 0\ to\ position$) do
5.          $child.path[i] = parent_1.getNode(i)$;
6.      end for
7.      for ($i \leftarrow position\ to\ child.size$) do
8.          $child.path[i] = parent_2.getNode(i)$;
9.      end for
10. end if
11. return $child$;
```

```
Function 9. Get the best composition path: $getFittest(pop, Q.cost)$
Input: population, $pop$.
Output: The best composition path.
1. $best = pop.paths[0]$;
2. double bcost =getCost(best);        /* See Function 3 */
3. for (i←1 to $pop.size$) do
4.     if ((getCost($pop.paths[i]$) $- Q.cost$)| < |(bcost $- Q.cost$)|)
5.         $best = pop.paths[i]$;
6.     end if
7. end for
8. return $best$;
```

```
Algorithm 2: Genetic Composition Algorithm: $GA(G, Q.cost, Q.f, v.f)$
Input:
  • Directed acyclic video transcoding graph, $G(T, L)$.
  • User cost, $Q.cost$, and the requested format, $Q.f$.
  • Original Video format, $v.f$.
Output: Near optimal video transcoding composition path.
Part 1: Configuration:
1. config($popSize_0$, £$_0$) ;              /* see Function 6 */
Part 2: Initialization
2. Population $pop$ = InitiaPop($popSize$)
       /* see Function 7 */
Part 3: Loop
3. for (i←1 to £) do
4.     $pop$ = evolvePop($pop$);       /* see Function 8 */
5. end for
6. return $getFittest(pop)$;           /* see Function 9 */
```

VIII. Evaluations and Discussion

In this section, we evaluate our adaptation to the SA/GA algorithms through experiments. In the first experiment we focus on: a) the sensitivity analysis of the parameters of the SA algorithm, b) the sensitivity analysis of the parameters of the GA, and c) analyzing the execution time for both of them. Then we provide a discussion and analysis that compare these evaluation results.

In the second experiment, we focus on the quality of the results. In other words, how well the proposed algorithms generate results. To measure their quality results, we calculate the success ratio of the SA algorithm and compare it with the GA algorithm's success ratio. Finally, we provide a discussion regarding the limitations and some possible extensions of our work.

A. *Configuration*

For these experiments, we use Java JDK 1.8 via IDE Eclipse Kepler service release 2 to implement our adaptation to the SA and GA algorithms and other utility classes. The evaluation is done using a desktop computer that has an Intel Core i5 CPU 3.30 GHz; 8 GB RAM, and runs Windows 7 Professional.

B. *Setup*

To evaluate our adaptation to the SA/GA algorithms, we create the following groups of video transcoding functions; these video transcoding functions cover some of the common video characteristics, Table 2 shows a number of them.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

The descriptions of these groups are presented in Table 3. Each group represents a graph which contains set of nodes and links as we described above. The nodes are the video transcoding functions and the links are the edges that connect these nodes. Each group has a set of levels. Each level represents a transcoder (i.e., set of video transcoding functions that convert from one codec to another). For example, group A has two levels (i.e., the first level converts from MPEG-2 codec to H.264 codec and the second level converts from H.264 codec to MJPEG codec). Group A contains two transcoders; each one has 36 video transcoding functions. Therefore, this group contains 72 video transcoding functions. Each group (i.e., graph) also contains two other nodes, the source and the destination. The source node represents the original video content and the destination node represents the transcoded video based on the viewer requested format.

Table 2. A number of video transcoding functions

t_i	From	To	Frame rate	Bit rate	Frame size			delay
					width	height	Aspect ratio	
t_1	MPEG-2	H.264	25	1075	640	480	1.33	5
t_2	H.264	MJPEG	20	215	320	240	1.33	3

Table 3. Group description

Group ID	Nodes in each level	Number of levels	Total number of nodes	Total number of links
A	36	2	2+(36*2)= 74	36^2+2*36 = **1368**
B	72	3	2+(72*3)= 218	72+(72*72) +(72*72) +72 = **10512**
C	90	4	2+(90*4) = 362	90+(90*90) +(90*90) + (90*90) + 90 = **24480**

C. *Sensitivity analysis of the parameters*

We perform the sensitivity analysis of the parameters of the proposed algorithms to determine the performance and the quality of the results. In this experiment we focus on measuring how much the results are close to each other and close to the viewer request. We evaluate the mean and the standard deviation of the solution, s, which represents a positive value that is calculated by taking the absolute difference between the cost of the QoS-aware video transcoding composition path that is returned from applying the SA/GA algorithms, $s_k.cost$, and the viewer request's cost, $Q.cost$, using (10). Also, we evaluate the average run time in millisecond for each algorithm based on 10 different viewer requests, and run each request 10 times.

$$s = abs(s_k.cost - Q.cost) \tag{10}$$

Table 4 and Table 5 show the sensitivity analysis of the parameters (i.e., the temperature and the cooling rate) of the SA algorithm. T and γ are the temperature and the cooling rate, respectively. avg(s) std(s) are the average of the mean and the average of the standard deviation of the solution, s, avg(t) is the average execution time in millisecond that is needed to find the composition path, s_k for request i. We compute the values of the above as follows where $n = 10$, total number of runs for each viewer request and $m = 10$, total number of viewer requests:

$$avg(s) = \frac{1}{m}\sum_{i=1}^{m}(AVG_i) \tag{11}$$

$$AVG_i = \frac{1}{n}\sum_{k=1}^{n}(s_{k_i}.cost - Q_i.cost)) \tag{12}$$

$$std(s) = \frac{1}{m}\sum_{i=1}^{m}(STD(s_i)) \tag{13}$$

$$STD(s_i) = \sqrt{\frac{1}{n}\sum\nolimits_{k=1}^{n}(((s_{k_i}.cost - Q_i.cost) - avg(s))^2)} \tag{14}$$

$$avg(t) = \frac{1}{m}\sum_{i=1}^{m}(\frac{1}{n}\sum_{k=1}^{n}(s_{k_i}.t)) \tag{15}$$

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

where $s_{k_i}.t$ is the execution time in millisecond needed to find the composition path k for request i. For $avg(s)$, $std(s)$ and $avg(t)$, the lower the value, the better it is.

We can notice from Table 4 that, for example, for group A, when T = 100000 and γ = 0.002, avg(s) and std(s) are 0, which means that the SA generates the same path and the cost of this path is the same as the viewer's requested cost during all the runs. Therefore, the SA successfully finds the optimal path that is the closest path to the viewer request among all the discovered paths. So, we can conclude that, based on this case, the SA finds the optimal QoS-aware video transcoding composition path. When T= 1000 and γ = 0.002, we got almost the same results with less execution time. Notice that when the value of T increased and the value of γ decreased, we got better results in terms of $avg(s)$ and $std(s)$. The best solutions obtained are **boldfaced**.

For group B and C, bigger groups, the situation is almost the same, the results getting better when the value of T increased and the value of γ decreased, which means that when the graph getting bigger, we need more time to get better results. For example, for group B, the best values of $avg(s)$ and $std(s)$ are when T =100000 and γ = 0.002, which takes 1242.38 ms. While for group C, it takes 6782.49 ms when T = 100000 and γ = 0.001.

Table 4. The Sensitivity Analysis of the SA Algorithm's parameters (i.e., Temperature, T, and Cooling rate, γ (10 runs)).

Group ID	γ	T=1000		T=5000		T=10000		T=20000		T=50000		T=100000	
		$avg(s)$	$std(s)$	$avg(s)$	$std(s)$	$avg(s)$	$std(s)$	$avg(s)$	$std(s)$	$avg(s)$	$std(s)$	$avg(s)$	$std(s)$
A	0.002	0.0000	0.0001	0.0012	0.0037	0.0013	0.0041	0.0013	0.0041	0.0002	0.0005	**0.0000**	**0.0000**
	0.004	0.0056	0.0073	0.0005	0.0009	0.0027	0.0057	0.0027	0.0056	0.0002	0.0006	0.0015	0.0045
	0.006	0.0051	0.0070	0.0048	0.0076	0.0031	0.0059	0.0052	0.0069	0.0020	0.0049	0.0038	0.0063
	0.008	0.0062	0.0082	0.0092	0.0090	0.0023	0.0051	0.0030	0.0063	0.0067	0.0071	0.0042	0.0088
B	0.002	0.0315	0.0156	0.0274	0.0184	0.0289	0.0128	0.0236	0.0119	0.0233	0.0135	**0.0195**	**0.0114**
	0.004	0.0450	0.0212	0.0376	0.0128	0.0356	0.0169	0.0326	0.0175	0.0406	0.0121	0.0364	0.0153
	0.006	0.0534	0.0236	0.0428	0.0201	0.0444	0.0247	0.0427	0.0199	0.0447	0.0187	0.0375	0.0214
	0.008	0.0458	0.0186	0.0528	0.0248	0.0408	0.0280	0.0479	0.0230	0.0359	0.0202	0.0473	0.0206
C	0.001	0.0669	0.0209	0.0672	0.0293	0.0752	0.0166	0.0738	0.0259	0.0673	0.0193	**0.0610**	**0.0209**
	0.002	0.0945	0.0278	0.0848	0.0223	0.0828	0.0294	0.0863	0.0236	0.0778	0.0165	0.0676	0.0252
	0.003	0.1054	0.0353	0.1052	0.0299	0.1017	0.0230	0.0775	0.0235	0.0815	0.0270	0.0849	0.0287
	0.004	0.1083	0.0330	0.1102	0.0295	0.0878	0.0333	0.0986	0.0259	0.0912	0.0274	0.0856	0.0294

Table 5 shows the average execution time for SA algorithm. It is obvious that when the value of the T increased and the value of γ decreased, the execution time will be increased. In addition, when the number of video transcoding function nodes in the graph increased, the execution time will be increased as well. However, the average execution time for SA is lower than the average execution time for GA as we will show next.

Table 5. The average execution time for the SA algorithm based on different temperature, T, and cooling rate, γ values (10 runs).

Group ID	γ	T = 1000	T = 5000	T = 10000	T = 20000	T = 50000	T = 100000
		$avg(t)\ (ms)$	$avg(t)\ (ms)$	$avg(t)\ (ms)$	$avg(t)\ (ms)$	$avg(t)\ (ms)$	$avg(t)\ (ms)$
A	0.002	102.95	133	146.86	159.96	178.02	190.56
	0.004	43.24	55.62	61.16	66.66	74.03	79.59
	0.006	26.7	34.16	37.57	40.99	45.57	48.82
	0.008	18.62	24.02	26.49	28.95	32.53	34.93
B	0.002	676.86	857.01	942.67	1031.15	1151.52	1242.38
	0.004	310.41	387.83	424.06	460.65	510.16	549.98
	0.006	199.31	250.16	272.68	294.03	325.03	348.52
	0.008	147.98	184.34	199.2	215.31	237.28	252.73
C	0.001	3884	4856.82	5289.36	5796.66	6333.57	6782.49
	0.002	1788.12	2221.84	2434.45	2637.85	2872.17	3065.89
	0.003	1168.39	1449.13	1565.71	1690.82	1844.48	1967.4
	0.004	862.71	1067.3	1159.96	1241.7	1358.8	1458.37

Table 6 shows the sensitivity analysis of the GA's parameters (i.e., population size and number of iterations). P and £ are the population size and the number of iterations, respectively. $avg(s)$, $std(s)$, and $avg(t)$ mean the same as above and we used (10) to (15) to calculate their values. As we mentioned above, For $avg(s)$, $std(s)$ and $avg(t)$, the lower the value, the better it is. The best solutions obtained are **boldfaced**.

We can notice from Table 6 that, for example, for group A and B, we get the best values of $avg(s)$ and $std(s)$ when P = 250 and £ = 30, while for group C, we get the best values of $avg(s)$ and $std(s)$ when P = 250 and £ = 100. Also, we can notice that GA finds better results than SA in terms of $avg(s)$ and $std(s)$. However, GA takes very long time than SA. For example, for groups B, GA takes 3816.38 ms to find the best result while for group C, it takes 32894.68 ms. In general, increasing the value of P and £ does not mean always getting much better results.

Table 6. The sensitivity analysis of the GA's parameters, i.e., population size, P and number of iterations £ (10 runs)

Group ID	£	P = 50			P = 100			P = 150			P = 200			P = 250		
		$avg(s)$	$std(s)$	$avg(t)$ (ms)	$avg(s)$	$std(s)$	$avg(t)$ (ms)	$avg(s)$	$std(s)$	$avg(t)$ (ms)	$avg(s)$	$std(s)$	$avg(t)$ (ms)	$avg(s)$	$std(s)$	$avg(t)$ (ms)
A	5	0.0618	0.0799	13.77	0.1062	0.1090	26.52	0.1345	0.1092	39.34	0.1511	0.1218	52.56	0.1594	0.1069	66.19
	10	0.0204	0.0206	26.37	0.0119	0.0263	53.22	0.0087	0.0214	80.26	0.0099	0.0240	106.49	0.0163	0.0409	133.04
	15	0.0095	0.0171	39.26	0.0102	0.0228	78.38	0.0040	0.0099	121.48	0.0031	0.0083	161.13	0.0018	0.0045	200.1
	20	0.0165	0.0144	53.58	0.0018	0.0023	106.41	0.0011	0.0015	159.13	0.0010	0.0013	208.98	0.0033	0.0094	267.17
	30	0.0160	0.0191	80.37	0.0030	0.0057	160.63	0.0032	0.0062	239.18	0.0005	0.0006	321.25	**0.0004**	**0.0004**	**397.1**
	50	0.0193	0.0208	132.83	0.0068	0.0093	264.48	0.0020	0.0049	391.38	0.0015	0.0040	531.16	0.0004	0.0005	659.87
	100	0.0169	0.0167	267.55	0.0036	0.0062	532.82	0.0015	0.0016	796.22	0.0009	0.0013	1068.09	0.0017	0.0047	1335.07
	200	0.0126	0.0137	530.3	0.0047	0.0079	1050.54	0.0024	0.0051	1568.82	0.0018	0.0047	2121.29	0.0005	0.0005	2583.5
B	5	0.1866	0.1287	128.56	0.1837	0.1574	253.64	0.2050	0.1557	378.52	0.2395	0.1534	508.16	0.1500	0.1111	634.27
	10	0.0469	0.0477	258.44	0.0488	0.0687	509	0.0710	0.0784	758.89	0.0694	0.0894	1018.14	0.0790	0.1088	1285.11
	15	0.0300	0.0189	390.01	0.0150	0.0252	766.93	0.0144	0.0314	1145.73	0.0087	0.0190	1540.87	0.0297	0.0549	1910.53
	20	0.0332	0.0294	519.08	0.0135	0.0151	1018.07	0.0074	0.0121	1522.57	0.0057	0.0073	2034.79	0.0024	0.0045	2545.46
	30	0.0339	0.0240	775.19	0.0213	0.0240	1548.02	0.0078	0.0085	2270.72	0.0025	0.0029	3032.83	**0.0016**	**0.0018**	**3816.38**
	50	0.0465	0.0249	1285.08	0.0118	0.0127	2589.24	0.0056	0.0075	3827.55	0.0039	0.0060	5052.19	0.0023	0.0028	6332.7
	100	0.0281	0.0201	2555.32	0.0133	0.0126	5102.3	0.0063	0.0085	7697.33	0.0034	0.0047	10165.14	0.0032	0.0061	12639.56
	200	0.0220	0.0190	5146.5	0.0082	0.0096	10223.39	0.0048	0.0044	15417.62	0.0021	0.0028	20141.77	0.0019	0.0024	24990.13
C	5	0.2407	0.1312	331.89	0.2501	0.1349	660.65	0.2414	0.1453	987.42	0.2551	0.1616	1311.83	0.2556	0.1392	1650.16
	10	0.0873	0.0727	662.08	0.0864	0.0954	1314.33	0.0976	0.0922	1978.2	0.0996	0.0977	2629.76	0.0969	0.0822	3277.93
	15	0.0565	0.0362	989.3	0.0250	0.0335	1974.62	0.0355	0.0453	2959.37	0.0397	0.0667	3928.45	0.0354	0.0549	4924.83
	20	0.0512	0.0285	1320.05	0.0329	0.0213	2620.28	0.0168	0.0188	3950.66	0.0122	0.0149	5256.53	0.0124	0.0268	6544.6
	30	0.0582	0.0290	1960.09	0.0203	0.0159	3932.84	0.0142	0.0091	5924.82	0.0084	0.0079	7894.75	0.0078	0.0089	9855.14
	50	0.0451	0.0237	3288.79	0.0166	0.0119	6567.57	0.0170	0.0191	9917.35	0.0056	0.0068	13153.09	0.0065	0.0070	16343.35
	100	0.0564	0.0312	6565.13	0.0193	0.0133	13147.43	0.0138	0.0105	19635.23	0.0069	0.0080	26282.76	**0.0052**	**0.0048**	**32894.68**
	200	0.0552	0.0383	13083.79	0.0310	0.0241	26241.13	0.0106	0.0096	39249.11	0.0067	0.0065	52938.24	0.0058	0.0093	65453.51

Also, we can notice that the GA needs more time to find the near optimal solutions for group B and C, while the SA finds good enough solutions for group B and C within less period of time. For this reason, we performed another experiment for the SA algorithm in which we give the SA much more opportunity to be able to find better results. Table 7 describes this experiment. This done by increasing the value of T and decreasing the value of γ. Table 7 shows that the SA generates the best result for group B when the value of T = 1E+20 and γ = 0.001. The best solutions obtained are **boldfaced**. However, it takes 26134.04 ms which is much higher than the time that the GA takes to find even a better result. For group C, the SA generate the best result when the value of T = 1E+20 and γ = 0.001. However, it takes 46777.47 ms which is much higher than the time that the GA takes to find even a better result.

Table 7. Second experiment for the SA algorithm

Group ID	T	γ	$avg(s)$	$std(s)$	$avg(t)$
B	1E+20	0.001	**0.0062**	**0.0069**	**26134.04**
	1E+20	0.002	0.0153	0.0089	8222.44
	1E+20	0.003	0.0133	0.0081	4462.95
	1E+20	0.004	0.0180	0.0096	2996.54
C	1E+20	0.001	0.0433	0.0158	46777.47
	1E+20	0.002	**0.0484**	**0.0117**	**16526.82**
	1E+20	0.003	0.0552	0.0188	9732.45
	1E+20	0.004	0.0693	0.0153	6840

Based on the results that are shown in Tables 4, 5, 6, and 7, we can conclude that:

- SA can find the optimal results for small groups or graphs and it can find good enough results for medium or bigger groups or graphs within a small duration of time.
- GA can find near-optimal results for medium or bigger groups. However, it takes very long time.

D. *Success ratio*

In this experiment, we focus on evaluating the proposed approaches in terms of success ratio. Then, we compare the success ration of the SA algorithm with the success ratio of the GA algorithm. Formally we can define the success ratio as follows:

$$S_r = \frac{\alpha}{\beta} \tag{16}$$

where S_r represents the success ratio, α represents the total number of QoS criteria that are satisfied, and β represents the total number of the QoS criteria that we measure (i.e., the total number of QoS criteria is 6). We provide a range of user satisfactions: 100%, 90%, and 80%. The 100% means that the end-user requests a 100% satisfaction on all the QoS criteria, which means that the end-user requests a video transcoding functions that are 100% compatible with his QoS requirements. For example, if the user wants a 25 fps as a frame rate and he requests a 100% satisfaction rate, we count this QoS as a satisfied QoS criterion if and only if the algorithm generates a path that has the last video transcoding function in this path generates exactly 25 fps as a frame rate. Similarly, the 90% satisfaction rate means we count this QoS criterion as a satisfied QoS criterion if and only if the algorithm generates a path that has the last video transcoding function in this path generates a frame rate between 22.5 and 27.5, which means if the viewer request a 90% satisfaction rate, he can accept any frame rate between the above range (i.e., 225 and 27.5).

Table 8 and Table 9 show the success ratio for SA algorithm and the GA algorithm, respectively. We calculate the $avg(all)$ by calculating the average of the success ratio after running 10 different viewer requests, and execute each request 10 times. Formally, we calculate $avg(all)$ as follows:

$$avg(all) = \frac{1}{i} * (\sum_{i=1}^{10} (\frac{1}{j} * (\sum_{j=1}^{10} VR_i . R_j . S_r)) \tag{17}$$

where VR_i is the viewer request i, R_j is the j run, and S_r is the success ratio for request i in run j.

These results that are shown in Table 8 and Table 9, proof that the SA algorithm is better than the GA for small groups or graphs while the results of both the SA and GA are close to each other for medium and large groups.

Table 8. Success ratio for the SA

Group ID	Satisfaction rate	$avg(All)$
A	100%	0.82
	90%	0.85
	80%	0.88
B	100%	0.34
	90%	0.46
	80%	0.59
C	100%	0.29
	90%	0.42
	80%	0.56

Table 9. Success ratio for the GA

Group ID	Satisfaction rate	$avg(All)$
A	100%	0.56
	90%	0.61
	80%	0.69
B	100%	0.33
	90%	0.45
	80%	0.57
C	100%	0.34
	90%	0.47
	80%	0.60

IX. Limitations and Future Direction

Real-time video transcoding for on-demand videos requires transcoding the video content on-the-fly and in real-time. Satisfying the requested QoS level during the transcoding process requires selecting and/or composing several video transcoding functions together in a chained fashion. Efficient selection/composition processes requires more efficient approaches. The proposed approaches still have some limitations, such as low performance and quality results. However, this is just the beginning to propose robust video transcoding service composition approaches in cloud computing. In addition to the current application specific QoS, future work may focus on adding the network QoS to the composition process. Also, we are planning to enhance the performance of these proposed algorithms by further implementation improvement. More possible future directions would be to build a complete cloud-based video delivery system, which includes both video transcoding and streaming subsystems, based on the viewer requirements and preferences.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 5, May 2015

X. Conclusion

Delivering real-time video content over any type of network is becoming a necessary requirement. During the delivery process, the original multimedia content might be transcoding to meet the different viewer's requirements and preferences. Guarantee QoS during the transcoding process requires selecting and composing the best video transcoding functions from a pool of transcoding functions. So, in this paper, we propose a video transcoding service selection and composition approach, which introduce two candidate algorithms based on the simulated annealing and the genetic algorithm, to satisfy the required QoS level and meet the viewer's requirements. We have implemented a prototype of the proposed algorithms and conducted experiments using small-, medium-, and large-scale graphs of video transcoders and sample viewer requests to measure the performance and the quality of the results. The experimental results show that the SA outperforms the GA in terms of performance and success ratio for small-scale graph, while GA outperforms the SA algorithm in terms of performance for medium- or large- scale graph. The success ratio for the SA and GA algorithms are close to each other for medium- or large-scale graph.

References

1. C. D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," April 2011. [Online]. Available: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. [Accessed 19 March 2015].
2. Apple, [Online]. Available: https://www.apple.com/iphone-6/. [Accessed 15 March 2015].
3. Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2013–2018," 10 June 2014. [Online]. Available: http://www.cisco.com. [Accessed 11 January 2015].
4. Wikipedia, "Transcoding," 6 January 2015. [Online]. Available: http://en.wikipedia.org/wiki/Transcoding. [Accessed 11 January 2015].
5. Wikipedia, "Video Codec," 1 July 2014. [Online]. Available: http://en.wikipedia.org/wiki/Video_codec. [Accessed 11 January 2015].
6. J. Cabasso, "Determining Video Quality," November 2008. [Online]. Available: http://www.aventuracctv.com/PDF/ATI_Video_Quality.pdf. [Accessed 24 January 2015].
7. Samsung, [Online]. Available: http://www.samsung.com/us/. [Accessed 21 March 2015].
8. Apple. [Online]. Available: https://www.apple.com. [Accessed 11 January 2015].
9. Amazon, "Amazon Elastic Transcoder," [Online]. Available: http://aws.amazon.com/elastictranscoder/. [Accessed 11 January 2015].
10. "Akamai," [Online]. Available: http://www.akamai.com/. [Accessed 24 January 2015].
11. W. Zhu, C. Luo, J. Wang and S. Li, "Multimedia Cloud Computing," *Signal Processing Magazine, IEEE,* Vol. 28, No. 3, pp. 59-69, 2011.
12. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM,* Vol. 53, No. 4, pp. 50-58, April 2010.
13. F. Jokhio, A. Ashraf, S. Lafond, I. Porres and J. Lilius, "Cost-Efficient Dynamically Scalable Video Transcoding in Cloud Computing," TUCS Technical Report, Finland, 2013.
14. Wikipedia, "List of video transcoding software," 31 July 2014. [Online]. Available: http://en.wikipedia.org/wiki/List_of_video_transcoding_software. [Accessed 15 March 2015].
15. D. G. Joseph and M. Moghaddam, "Service Selection in Web Service Composition: A Comparative Review of Existing Approaches," in *Web Services Foundations*, Springer New York, 2014, pp. 321-346.
16. "Wikipedia," 15 Fabruary 2015. [Online]. Available: http://en.wikipedia.org/wiki/Simulated_annealing. [Accessed 15 March 2015].
17. Wikipedia, "Genetic Algorithm," 6 April 2015. [Online]. Available: http://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed 14 April 2015].
18. W. Li, Y. Wang, C. Li, S. Lu and D. Chen, "A QoS-aware service selection algorithm for multimedia service overlay networks," in *Parallel and Distributed Systems, 2007 International Conference on*, Hsinchu, pp. 1-8, 2007.
19. A. Jula, E. Sundararajan and Z. Othman, "Cloud computing service composition: A systematic literature review," *Expert Systems with Applications,* Vol. 41, No. 8, pp. 3809–3824, 15 June 2014.
20. M. S. Hossain and A. E. Saddik, "QoS Requirement in the Multimedia Transcoding Service Selection Process," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT,* Vol. 59, No. 6, pp. 1498-1506, 2010.
21. Y. Gao, J. Na, B. Zhang, L. Yang and Q. Gong, "Optimal Web Services Selection Using Dynamic Programming," in *Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06)*, pp. 365-370, 2006.
22. R. Maya and S. Ugrasen, "Web Service Selection Algorithm for Dynamic Service Composition using LSLO Approach," in *IEEE*, Dhaka, Bangladesh, pp. 1-6, 2013.
23. T. Zhang, "QoS-aware Web Service Selection based on Particle Swarm Optimization," *Journal of Networks*, Vol. 9, No. 3, pp. 565-570, 2014.
24. X. Gu and K. Nahrstedt, "Distributed multimedia service composition with statistical QoS assurance," *IEEE TRANSACTIONS ON*

MULTIMEDIA, Vol. 8, No. 1, pp. 141-151, 2006.

25. J.-C. Moissinac, "Automatic Discovery and Composition of Multimedia Adaptation Services," in *The Fourth International Conferences on Advances in Multimedia*, Chamonix, France, April 29, pp. 155-160, 2012.
26. M. S. Hossain, "QoS-aware service composition for distributed video surveillance," *Multimedia Tools and Applications,* Vol. 73, No. 1, pp. 169-188, 2014.
27. A. J. Gonzalez, J. Alcober, R. M. d. Pozuelo, F. Pinyol and K. Z. Ghafoor, "Context-aware multimedia service composition using quality assessment," in *2011 IEEE International Conference on Multimedia and Expo (ICME)*, Barcelona, Spain, pp. 1-6, 11-15 July 2011.
28. V. G. a. L. Sakalauskas, "Simulated Annealing and Variable Neighborhood Search Algorithm for Automated Software Services Composition," in *35th International Convention on Information and Communication Technology, Electronics and Microelectronics* , Opatija, Croatia, pp. 395-399, May 21-25,2012.
29. Z. Ye, X. Zhou and A. Bouguettaya, "Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing," *Database Systems for Advanced Applications*, lecture Notes in Computer Science, Vol. 6588, pp 321-334, April, 2011.
30. B. S. a. P. Kumar, "A Survey of Simulated Annealing as a Tool for Single and Multiobjective Optimization," *The Journal of the Operational Research Society,* Vol. 57, No. 10, pp. 1143-1160, 2006.
31. Z.-p. GAO, J. CHEN, X.-s. QIU and L.-m. MENG, "QoE/QoS driven Simulated Annealing-based Genetic Algorithm for Web Service Selection," *The Journal of China Universities of Posts and Telecommunications,* Vol. 16, No. 1, pp. 102–107, September 2009.
32. L. Arockiam and N. Sasikaladevi, "Simulated Annealing Versus Genetic Based Service Selection Algorithms," *International Journal of U- & E-Service, Science & Technology,* Vol. 5, issue 1, p35, 2012.
33. L. Arockiam and N. Sasikaladevi, "Simulated Annealing Based Service Selection Algorithm for Composite Web Service," *International Journal of Advanced Research in Computer Science,* Vol. 3, No. 2, p.132, March 2012.
34. " Euclidean Distance," Wikipedia, 17 November 2014. [Online]. Available: http://en.wikipedia.org/wiki/Euclidean_distance#Squared_Euclidean_distance. [Accessed 6 11 2014].
35. Jinshan Liu, Val´erie Issarny, "QoS-aware Service Location in Mobile Ad-Hoc Networks," in 5th International Conference on Mobile Data Managment MDM, Berkeley, CA, United States. pp. 224-235, 2004.
36. Y. Xu, R. Qu and R. Li, "A simulated annealing based genetic local search algorithm for multi-objective multicast routing problems," *Annals of Operations Research*, Vol. 206, No. 1, pp. 527-555, July 2013.
37. Wikipedia, "Video Quality," 9 March 2015. [Online]. Available: http://en.wikipedia.org/wiki/Video_quality. [Accessed 21 March 2015].
38. Wikipedia, "Video Quality," 14 July 2014. [Online]. Available: http://en.wikipedia.org/wiki/Video_quality. [Accessed 12 January 2015].
39. W. D. J. C. Lianyong Qi, "Weighted principal component analysis-based service selection method for multimedia services in cloud," *Springer Computing - Springer-Verlag Wien,* 2014.

BIOGRAPHY

Nawaf Alsrehin is a PhD student-Graduate Teaching Assistant in the Computer Science department, School of Engineering, Utah State University, USA. He received a Bachelor degree of Computer Science (CS) and a Master degree of Computer Information System (CIS) in 2003 and 2006 from Yarmouk University, Jordan. His research interests are multimedia services, video transcoding, and cloud-based multimedia services.