Scheduling Video Transcoding Jobs in the Cloud

Panagiotis Oikonomou
Computer Science
Univ. of Thessaly
Lamia, Greece
paikonom@uth.gr

Maria G. Koziri
Computer Science
Univ. of Thessaly
Lamia, Greece
mkoziri@uth.gr

Nikos Tziritas
Computer Science
Univ. of Thessaly
Lamia, Greece
nitzirit@uth.gr

Antonios N. Dadaliaris
Computer Science
Univ. of Thessaly
Lamia, Greece
dadaliaris@uth.gr

Thanasis Loukopoulos
Computer Science and Biomedical Informatics
Univ. of Thessaly
Lamia, Greece
luke@uth.gr

Georgios I. Stamoulis
Electrical and Computer Eng.
Univ. of Thessaly
Volos, Greece
georges@uth.gr

Samee U. Khan
Electrical and Computer Eng.
North Dakota State Univ.
Fargo, ND, USA
samee.khan@ndsu.edu

***Abstract*—Video transcoding is the process of producing from an original (already encoded) input video sequence, multiple output sequences, each at potentially different bitrate, resolution and/or format. Transcoding is essential to support video delivery towards clients that use different players and have different network access capabilities. In the most basic scheme the input sequence is decoded and then re-encoded at the desired levels. Although significant research on fast transcoding schemes exists, the transcoding process is still computationally intensive. For this reason efficient scheduling methods that allocate resources to transcoding jobs are necessary in order to achieve good overall performance. Such policies usually aim at allocating transcoding jobs over co-located servers, thus, they typically overlook parameters such as network traffic. Motivated by the case of transcoding in the Cloud, in this paper we investigate the problem of scheduling transcoding jobs over a distributed system comprising of processing nodes that are geographically dispersed and might be whole clusters or even separate data centers. We propose algorithms to minimize both the inter-node network traffic and the intra-node energy consumption, while meeting the deadlines and quality requirements. Through simulation experiments we conclude on the best alternatives.**

Keywords—video coding, transcoding, scheduling, Cloud, energy efficiency

I. INTRODUCTION

Video uploading and downloading is one of the most popular activities among social media users. Cisco reported in [1] that in 2015 a year whereby mobile Internet traffic experienced a rise by 74%, 55% of it accounted for video. These trends will likely continue, even intensify, as modern smart devices with 4K cameras are established in the market and 8K TV screens replace older ones. To cope with an everlasting demand for higher resolution, new video coding standards such as VP9 [2] and HEVC [3] were introduced in recent years, while the quest for future generation standards have already begun, e.g., AV1 [4], JVET [5]. All these new standards aim at replacing the long standing H.264/AVC standard [6], by offering increased compression ratios without sacrificing quality.

At the same time, the delivery of video streams towards end devices of different screen and processing capabilities that use different players and reside on networks of various bandwidth capacities, poses a major challenge. To cope with the problem, transcoding is used whereby from the initial video sequence, a set of output sequences is produced, each potentially at different resolution, transmission rate and quality level. On top, transcoding might also involve the change of the coding standard, e.g., from H.264/AVC to HEVC [7]. The various rates and qualities on which an original sequence is transcoded (also called the encoding ladder) can be of large number. For instance, in Netflix a ladder of at least 10 different levels (probably more) is advocated in [8]. It is evident that even with such a conservative ladder, the processing demands posed by the transcoding needs of large media providers cannot be met without taking advantage of Cloud resources. For this reason a number of Cloud service providers, include nowadays video coding services, e.g., Amazon Elastic Transcoder, or transcoding services as part of their live casting service, e.g., Wowza Streaming Cloud.

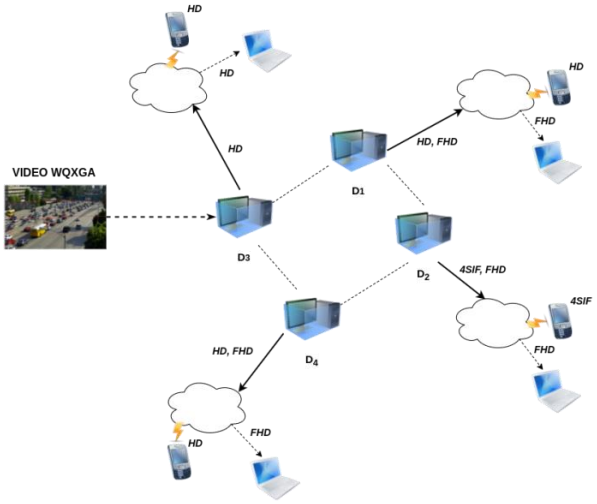

Fig. 1. An example of a Cloud transcoding service.

In this paper we consider the case of a Cloud service provider offering video coding and transcoding services and tackle the related scheduling problem. In particular we discuss the case whereby the provider owns a number of geographically distributed data centers and coding jobs arrive from different locations. An example of such a system is given in Fig. 1 that shows an arriving video at data center D_3, being transcoded at various targeted resolutions and delivered in multiple end points. We focus on investigating global scheduling approaches whereby it is decided in which data center to assign each transcoding job so as to optimize completion time, energy efficiency and network traffic. Our contributions include the following:

- We use a system model that captures the case of Cloud transcoding service providers operating over different data centers. The model is suitable for capturing both the case of live transcoding tasks and offline video sequence coding;
- We develop dynamic global scheduling algorithms that have different optimization goals, i.e., processing time, energy and network overhead;
- We conducted two sets of simulation experiments, based on realistic live casting scenario (Twitch) and offline video coding one (Facebook).

Results indicate that a global scheduler that decides based on Pareto optimality upon the three parameters: time, energy, network offers the best approach, while incurring small coordination overhead. Quite surprisingly, many Cloud service providers, e.g., Amazon Elastic Transcoder, leave location selection as a manual decision to the end user. To the best of our knowledge this is the first paper tackling the scheduling of transcoding jobs over different data center locations.

The rest of the paper is organized as follows. Section II discusses related work. The system model is illustrated in Section III. Our proposed scheduling schemes are presented in Section IV and evaluated in Section V. Finally, Section VI illustrates our concluding remarks.

II. Related Work

A. Energy Efficiency in Datacenters

Building energy efficient data centers has attracted much research effort recently, with the scope varying all the way from the case of optimizing single cluster to holistic data center design. In [9] energy efficient scheduling algorithms for heterogeneous clusters are proposed, while [10] and [11] characterize the energy efficiency of Hadoop clusters. In [12] a comparative study between 13 scheduling algorithms is done with an interest on energy consumption, while in [13] applications of DVFS for Hadoop clusters are also studied (among others). Finally, [14] discusses energy efficient in Hadoop clusters both from the standpoint of minimizing the total consumed energy and the peak power.

As far as data center level optimizations are concerned, [15] provides a thorough survey on modeling energy consumption, while [16] focuses on design principles for energy efficiency with a special interest on data centers providing video related services. A key component to reducing data center power consumption is the successful implementation of server consolidation. Server consolidation is the process of maintaining in active state the minimum set of servers that meet workload demands, placing the rest in hibernating mode or completely turning them off in order to save energy. Surveys on the topic can be found in [17] and [18].

In this paper we do not aim to model the effects of particular strategies within a data center, but we are rather interested in proposing global scheduling policies that are applicable regardless of the local strategies followed.

B. Video Coding and Trancoding

A lot of research efforts were devoted into reducing the computational burden imposed by video codecs, usually by taking advantage of parallelism. Examples on the category include for instance [19], [20], [21], [22] and [23] each tackling parallelism at a different level. Slice level parallelism was studied for instance in [19] whereby a frame is split into independent regions (slices) that can be encoded separately and each slice is assigned to a separate CPU core. Tile level parallelism, another method to split a frame into independent regions introduced by HEVC, was discussed in [20] and [21]. These methods are rather coarse grained since they provide parallelism at the level of group of Macroblocks (H.264/AVC) or CTUs (blocks in HEVC). Finer grained approaches rely to SIMD instructions in order to parallelize the various components of the codec (motion estimation, compensation and transform are usual candidates). Example works include [22] where SIMD parallelism is discussed for HEVC encoding and [23] where the focus was to parallelize the various parts of an AVS decoder.

The aforementioned works aim at speeding up the coding process when the input is a raw video sequence. Efficient transcoding solutions that operate over already coded sequences were also proposed. Examples include [24] where the transcoding from H.264/AVC to HEVC is discussed and [2] where an HEVC to VP9 transcoder is proposed. Research in this area usually aims at reusing the prediction information from the input sequence, in order to speedup the encoding

process in the targeted standard. Thus, impressive speedups (of 7x, even more) can be achieved compared to the process of decoding the original sequence and re-encoding it to the required target standard.

Since the focus of the paper is not to optimize a particular transcoding task, but rather to schedule transcoding jobs efficiently at a higher level, we do not take any particular assumptions on the optimizations that take place at a video coding level. In the experiments, we used the very popular `x264` and `x265` codecs in order to obtain realistic values for our simulations.

C. Cloud Trancoding

Contrary to video delivery that has been the target of extensive research effort, see [25] for a recent survey, few works relatively exist on Cloud provisioning for video coding jobs. In [26] the parallelization of an encoding task over a Hadoop cluster is proposed at a Group of Pictures (GOP) level. GOPs are mapped to the various processing nodes and are independently encoded. The reduction phase includes synthesizing the final coded sequence from its sub-parts. Although this approach promises a high parallelization degree for a standalone video sequence, its performance is questionable in the presence of high load due to the overheads incurred with Map-Reduce. For this reason, in the paper we assume that parallelization (if any) occurs within a single server. In other words a transcoding task is executed by a single server and not multiple ones.

Closer to this paper are the works in [27], [28] and [29]. In [27] the authors propose an online algorithm to decide about the server that will host a transcoding job. Parameters such as video resolution, server power and queue lengths are taken into account in order to optimize both processing delay and energy consumption. In [28] the case of live video streaming is tackled. The authors proposed an ILP formulation to decide about the transcoding rates of each stream that could be achieved based on the available processing capacity, having as a final goal the optimization of user experience. In [29] the focus is to decide on which video parts to transcode and cache them, based on the observation that users rarely view a video sequence completely.

Our work differs from the aforementioned papers in scope. Namely, whereas the focus of related work was on optimization at the level of a single cluster or (at most) on a single data center, we aim at investigating the performance of global scheduling schemes that distribute jobs over different distributed data centers. Nevertheless, we use the same dataset as in [28] to simulate the case of live transcoding jobs.

III. System Model

We assume a Cloud transcoding service with computational resources spanning across different geographically distributed data centers. In order to facilitate description throughout this section we introduce some notations that are summarized in Table I. Let D be the total number of available data centers and D_i denote the i^{th} such assuming a total ordering of them ($1 \leq i \leq D$). Each data center contains a number of servers. Such servers might in fact be virtual machines running over physical ones. As already mentioned we refrained from tackling the implementation particulars of data centers, e.g., VM allocation and migration policies. Therefore, for all purposes, the servers are assumed to be dedicated for transcoding jobs only. Let S be the total number of servers across all the available data centers dedicated for transcoding. We denote by S_k the k^{th} such server assuming a total ordering of them ($1 \leq k \leq S$).

Each server potentially has its own characteristics concerning processing capacity and power consumption. We denote server processing capacity by C_j. It is straightforward that this capacity can be measured as CPU speed, FLOPS or any other similar metric in the literature. However, in order to both simplify the model but also be able to capture the required performance characteristics without introducing additional parameters such as memory or number of cores, we measure processing capacity as the number of baseline video sequences that can be handled concurrently by the server. The baseline is assumed to be a raw 240p video that must be encoded in 30 frames per second using H.264/AVC (the particular coding settings used are detailed in Section V).

Each server is also characterized by its energy efficiency (let E_k). As already stated we do not consider DVFS since we focus on a higher than a cluster scheduling level. Instead we measure energy efficiency at each server based on its maximum processing capacity. Assuming PW_k to be the corresponding power consumption rate (e.g., Watts) the energy efficiency of each server is given by:

$$E_k = C_k / PW_k \tag{1}$$

We assume that the data centers are connected with high speed network links, effectively forming a network graph. Among others we are interested in reducing the network overhead due to transcoding jobs. We measure this overhead as the mere amount of bytes passing through each link (totaled for all links). We decided to follow a conservative estimation whereby the communication between D_i and D_j always occurs along the shortest path of h_{ij} hops.

Coding job requests arrive (presumably at some edge point) and are directed towards the closest data center. Let J_x denote the x^{th} coding job assuming a total order over the J total jobs that exist. Depending on whether the job represents a standalone file transcoding or a live casting event, it might result into one or more transcoding tasks respectively. Let JT_x be the number of transcoding tasks associated with J_x and T_{xy} be the y^{th} such transcoding task ($1 \leq y \leq JT_x$).

All the tasks of J_x are associated with the same input file or stream but with different outputs. Depending on the type (raw, H.264/AVC etc.) and the resolution of the input file, as well as the resolution and the type of the required output by a particular task, different processing capacity is required to achieve real time performance (e.g., 30 fps). Let W_{xy} denote the processing capacity required by T_{xy} measured as the ratio of load incurred by this particular transcoding versus the baseline case. Both the input sequence of J_x and all output sequences of the associated tasks have the same length (in secs), denoted by l_x. Nevertheless, they have different total sizes (in bytes). Let $b(J_x)$ denote the size of the input in J_x and $b(T_{xy})$ the size of the output of T_{xy}.

TABLE I. NOTATION USED IN THE PAPER

Symbol	Meaning
D	Total number of data centers
D_i	The i^{th} data center
S	Total number of servers in all D data centers
S_k	The k^{th} server
C_k	Processing capacity of S_k
PW_k	Power consumption of S_k
E_k	Energy efficiency of S_k
h_{ij}	Distance between D_i and D_j (in hops)
J	Total number of jobs
J_x	The x^{th} job
JT_x	The number of tasks associated with J_x
T_{xy}	The y^{th} task of J_x
W_{xy}	Processing load incurred by T_{xy}
l_x	Time length of input file/stream of J_x
$b(J_x)$	Total size length of input file/stream of J_x
$b(T_{xy})$	Total size length of output file/stream of T_{xy}
q_{xyk}	Service quality of live transcoding T_{xy} at S_k
AJ_x	Arrival time of J_x
ET_{xyk}	End time of T_{xy} assigned to S_k
A_{ix}	Boolean variable denoting whether J_x first arrives at D_i
O_{ixy}	Boolean variable denoting whether the output of T_{xy} is required by end users in the region of D_i
P_{xyk}	Boolean variable denoting whether T_{xy} is assigned for execution in S_k
IN_{ix}	Boolean denoting whether D_i must get the input stream of
B_{ik}	Boolean variable denoting whether S_k belongs to D_i.

Assuming T_{xy} is assigned to an initially empty S_k, we distinguish two cases:

(a) T_{xy} corresponds to live transcoding in which case for a duration of l_x, and assuming $W_{xy} \leq C_k$, W_{xy} out of C_k total processing capacity will be reserved by T_{xy}, leaving a remaining capacity of C_k-W_{xy} to be assigned to other tasks. Also we will denote that the quality (let q_{xyk}) at which T_{xy} was served equaled 1. In case $W_{xy} > C_k$ all the server capacity will be occupied and the corresponding service quality will be:

$$q_{xyk} = W_{xy}/C_k \tag{2}$$

(b) T_{xy} is a standalone file transcoding in which case all C_k will be reserved and T_{xy} that arrived at time AJ_x (all the tasks of a job are assumed to arrive at the same time) will have an end time (let ET_{xyk}) equaling:

$$ET_{xyk} = AJ_x + W_{xy}l_x/C_k \tag{3}$$

We should notice that (2) and (3) hold for tasks arriving at empty servers. In case a server is assigned more than one task then it is assumed that its processing power C_k is evenly split among the hosted tasks.

An arriving request for J_x might not necessarily be satisfied by the data center it first arrives. Furthermore, it is not necessary that all its tasks will be assigned to the same data center for processing. To calculate the network overhead incurred by J_x one must add the overhead for fetching the input stream to the position(s) where task processing will occur and the overhead for sending the outputs of the transcoding tasks to the necessary destinations.

Let A_{ix} be a boolean variable, with A_{ix}=1 iff J_x first arrives in D_i and 0 otherwise. Let O_{ixy} be a boolean variable, whereby O_{ixy}=1 iff the output sequence of T_{xy} must be sent to satisfy viewers that are located in the geographic neighbor of D_i and 0 otherwise. Let P_{xyk} be a boolean variable depicting whether T_{xy} is assigned to S_k (P_{xyk}=1) or not (P_{xyk}=0). Furthermore, let IN_{ix}=1, if the input stream of J_x must be made available at D_i and 0 otherwise. Obviously, IN_{ix}=1 iff at least one of the tasks of J_x are assigned for processing at a server located in D_i. Last, let B_{ik}=1 iff S_k belongs to D_i and 0 otherwise. Then we can typically define the network overhead (let N_x) incurred by J_x as:

$$N1_x = \sum_{i=1}^{D}\sum_{j=1}^{D} A_{ix} IN_{jx}\, h_{ij}\, b(J_x) \tag{4}$$

$$N2_x = \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{ik}\, O_{jxy} h_{ij}\, b(T_{xy}) \tag{5}$$

$$N_x = N1_x + N2_x \tag{6}$$

subject to the constraints:

$$\left(1 - IN_{jx}\right)\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{jk} = 0\,, \quad \forall 1 \leq j \leq D \tag{7}$$

$$\sum_{y=1}^{JT_x}\sum_{k=1}^{S} P_{xyk} B_{jk} \geq IN_{jx}\,, \quad \forall 1 \leq j \leq D \tag{8}$$

In (4) the overhead incurred by the input stream is captured; in (5) the overhead of all output streams, while (6) gives the total overhead for J_x as the summation of the two. (7) and (8) are necessary to ensure IN_{jx} takes valid values. Specifically (7) states that if at least one task of J_x is assigned for processing to one of the servers of D_j then IN_{jx} will equal 1. (8) forbids the case of having IN_{jx}=1 without assigning any of J_x's tasks at D_j.

We further illustrate network overhead calculation through Fig. 2 which is similar to Fig.1 but simplified. The figure shows a transcoding job arriving at D_3 that consists of two tasks, one to transcode from WQXGA to FullHD and another one to HD. Assume for the sake of the example that WQXGA size is b bytes, FullHD output is b/2 and HD b/4. By assigning the FullHD transcoding to D_2 and the HD transcoding to D_1, network overhead is calculated as follows. The input overhead will be 3b (2b to send to D_2 and b to send to D_1). The output overhead will be 0 for HD (only needed by the region of D_1) and b/2 for FullHD (must be sent from D_2 to the region of D_1). The total network overhead in this case equals 7b/2. On the other hand if all tasks were assigned to D_4, the input sequence overhead would be b (D_3 to D_4 transfer), the FullHD output (2+1)b/2 and the HD output 2b/4, for a total overhead of 3b.

The system wise requirement is to assign transcoding jobs to the available servers so that: (i) in case of live transcoding the QoS level as expressed in (2) is maximized; (ii) in case of non-live transcoding tasks the completion time as per (3) is

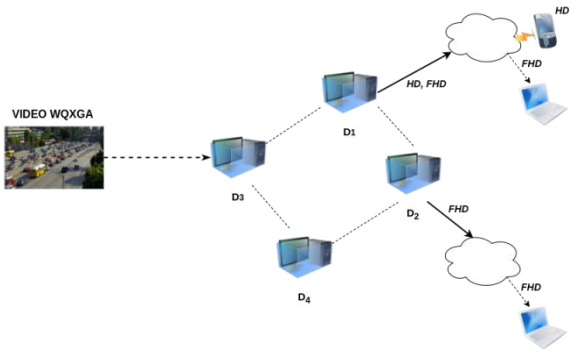

Fig. 2. An example transcoding job with two corresponding tasks.

minimized (or a deadline is met thereof); (iii) the total energy consumed by the servers is minimized (a single server S_k consumes energy PW_k multiplied by the amount of time it remains active computing at least one task) and (iv) the total network overhead as rigorously expressed in (4)-(8) for the case of a single job J_x is minimized for all arriving jobs. As it is evident, the system optimization targets are conflicting with each other. Next, we discuss scheduling policies that are either biased towards one criterion or favor Pareto optimal solutions.

IV. Scheduling Algorithms

We consider a two level scheduling scheme. In the top level a global scheduler intercepts all job requests and assigns the corresponding tasks to data centers. At every data center a local scheduler is responsible to assign the tasks allocated by the global scheduler onto the available servers. Before proceeding with the description of local and global scheduling policies, we would like to mention that in practice a global scheduler is not required to really intercept all job requests submitted by end users. Rather a policy can be implemented whereby the local scheduler of the data center a request arrives at, asks the global scheduler whether it should redirect the request or handle it itself, thus, realizing the global scheduler in an indirect manner. We refrain from discussing such details, instead focusing on the questions of what information should be used by the global scheduler and what selection policies to follow.

A. Local Scheduling

We experimented with HEFT-like local schedulers working as follows. Tasks are considered for scheduling one by one in arrival order. To perform task-server assignment, all servers are examined with the aim of finding one that satisfies the processing time criterion. In case more than one exists, the more energy efficient server according to (1) is selected. For live transcoding jobs the processing time criterion dictates finding a server such that q_{xyk} is maximized, while for non-live tasks we consider a deadline that must be met. It is worth noting that as far as the global scheduler is concerned, local schedulers are mere black boxes. Thus, the particular selection (HEFT-based) made in the paper is for simulation experiments purpose and does not restrict global scheduling choices.

B. Global Scheduling

To be feasible, global schedulers must work without having full knowledge of data center details. For instance knowing server queues would require an unrealistically high monitoring effort. In the paper we only consider the following information to be available at the global scheduling level: (a) the hop distance between any two data centers, i.e., h_{ij} values; (b) the total number of servers at each data center dedicated to the transcoding service; (c) the average energy efficiency and processing capacity of these servers; (d) for each job J_x its characteristics and (e) for each data center, the tasks currently assigned to it that have not finished yet. Information (a), (b) and (c), can be made available during a preprocessing step and does not change often throughout system's operation time. Knowledge of (d) is straightforward requiring only the information available with each job request. Finally, (e) can be made available by having each local scheduler notifying the global scheduler upon a task's completion.

Using the above information, the global scheduler estimates for each data center three values ($e1$, $e2$ and $e3$) each for the criteria of processing time, energy efficiency and network cost, measuring how beneficial it would be to assign a particular task there. For $e1$ and $e2$ the following estimations are used:

$$e1(D_i) = \sum_{T_{xy}|\exists k: P_{xyk}B_{ik}=1} W_{xy} \Big/ \sum_{k=1}^{S} C_k B_{ik} \tag{9}$$

$$e2(D_i) = e1(D_i)\left(\frac{\sum_{k=1}^{S} B_{ik}}{\sum_{k=1}^{S} E_k B_{ik}}\right) \tag{10}$$

The third estimator $e3$, i.e., the one for network, is actually calculated in an exact manner using (4)-(8) since the necessary information to do so is available by (a) and (d).

The global scheduler works in the following manner. Upon a job's arrival, and for every possible task-data center assignment, it calculates estimators $e1$, $e2$ and $e3$. The smaller the value of an estimator is, the better the assignment for this particular performance metric. Various policies can then be acted upon. For instance, the scheduler might choose based on $e1$ only, $e2$, or $e3$, termed the resulting scheme accordingly as P (processing), E (energy), N (network). Another policy is to rate the best solution across all three metrics. For this reason data centers are sorted three times (for $e1$, $e2$ and $e3$) and a Pareto optimal candidate is defined as the one with the minimum cumulative rank across all the three lists. Hence forth for sort, by Pareto optimal we will denote a solution with minimum cumulative ranking. We term this policy as PEN. Once a data center is selected, the global scheduler informs the relevant local scheduler upon its decision.

V. Experiments

A. Simulation Setup

Datacenters: To generate the network of datacenters we used the 30 regions where Azure was available as of November 2016 assuming the existence of a single data center at each of them. Due to absence of relevant connectivity information, we created the network as follows. First, we calculated for every Azure region pair, its geographic distance. Then at the resulting graph we ran MST.

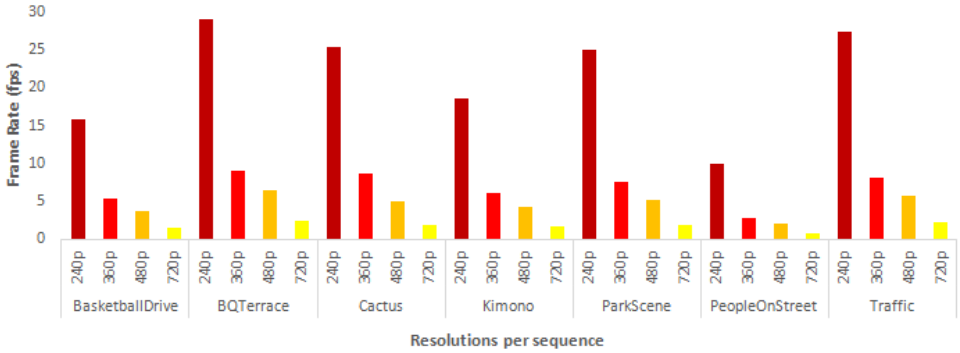

Fig. 3. Frame rate coding times for common test video sequences in various resolutions.

Servers: We performed simulation experiments on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz. Since our server setting is not of generic use, we translated results into one of the Amazon EC2 instances to make our simulation setting more applicable. Specifically, we consider the C3 instances which are recommended for video coding. Comparing the processor passmark ratios between the CPU of our server and the one used in the C3 instances, i.e., Intel Xeon E5-2680 v2 (Ivy Bridge) it can be estimated that the instance c3.4xlarge will account for a speedup of 2.2x compared to our server. Thus, we estimated the coding times over C3 instances using the actual values obtained by our server factored by the aforementioned speedup.

Live casting dataset: To simulate broadcasting activity, we used the same dataset from Twitch as the one described in [28]. We kept the portion of the dataset representing one day activity (Jan. 6th, 2014). We then filtered it by deleting entries with broadcasts having no viewers and the broadcasts of resolution less than 220p. To keep the simulation time manageable we consider the following 5 resolutions: 240p, 360p, 480p, 720p and 1080p. In case a broadcast in the trace does not follow one of the previously mentioned resolutions, we cluster it to its closest matching. We assume that a broadcast must be transcoded to all the resolutions that are lower than the one it uses. Clearly, with this setting the maximum number of transcoding tasks incurred by a broadcast is 4, corresponding to a 1080p stream that must be downscaled to 720p, 480p, 360p and 240p. Upscaling, is not considered in the experiments. For simulation purposes we assumed that all videos used 30 fps.

In the live casting scenario all arriving sequences were in H.264/AVC and should be transmitted in the same standard but at different resolution/rate. Since in the dataset of [28] no relevant information existed concerning transcoding times, we resorted to estimating them as follows. We used Class A and B common test video sequences Using `ffmpeg` we adjusted the resolution to the desired ones and encoded them using `x264` with parameters suggested for PSNR tailored performance in [30]. Fig. 3 plots the frame rate achieved by the encoder using one core. We then used the average value for its resolution case as a baseline estimator of the actual transcoding time. This baseline refers to the process of completely decoding the arriving sequence and re-encoding it from scratch, a strategy that is less than optimal as described in Section II(B). To capture the case where a more efficient transcoder would be available we considered a speedup of 7x in the process similarly to the performance achieved in [24]. Finally we factored in the number of available cores in a C3 instance and the relevant speedup against our server.

File transcoding dataset: We consider the case where a large social media provider wants to translate a portion of its videos from H.264/AVC into HEVC as motivated in [7]. In lack of a suitable dataset we created one as follows. We selected one of the most popular video publishers in Facebook, NowThis, which accounted during a 6 month time period (July 2016 –Dec. 2016) for roughly 6 billion views according to [31]. We random sampled 20 videos that accounted for roughly 200M views and downloaded them in both available resolutions, 240p and 720p (2 of them were 1080p instead of 720p). We then transcoded them from H.264/AVC into HEVC using `x265` with the PSNR tailored settings suggested in [30] and recorded the transcoding time for the two resolution levels. In the relevant experiment we assume a dataset size of multiple of these 40 files (20 videos in 2 resolutions) arrive in the system for processing.

B. Results

First we evaluated the performance of the algorithms in the live casting scenario. Broadcasters and viewers were randomly distributed over the available datacenters for the purpose of calculating network overhead. We measured the performance of the algorithms as the number of servers per datacenter increases from a baseline value obtained uniformly between 50 and 200 up to a 2x, 4x and 8x case.

Fig. 4 shows the percentage of live casting streams that were processed successfully, i.e., at their nominal rate of 30fps. As expected with increased processing power per data center the acceptance rate increases as well reaching 100% for all heuristics in the 8x case. Among single criterion options, the algorithm that favors processing time (P) over all other metrics achieves the best performance, followed by the network only metric (N) and the energy (E). It is worth noting, that the PEN algorithm that is based on identifying Pareto-optimality achieves comparable to P results in terms of accepted jobs. However, the merits of PEN are clearly shown in Figs. 5 and 6 that plot network overhead and energy consumption respectively. PEN achieves almost in par performance concerning the energy criterion against P while as far as network overhead is concerned, it accounts for a dramatic

Fig. 4. Percentage of accepted jobs as the number of servers increases (live casting scenario).

Fig. 5. Network cost as the number of servers increases (live casting scenario).

Fig. 6. Energy consumption as the number of servers increases (live casting scenario).

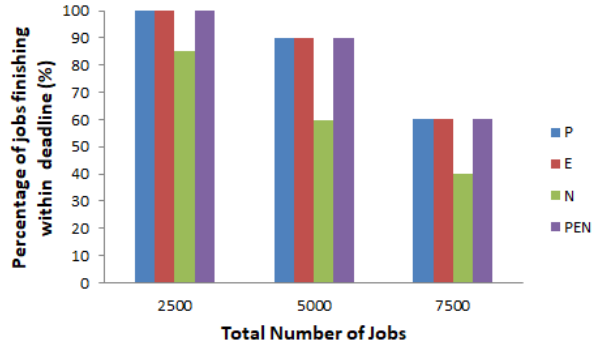

Fig. 7. Percentage of jobs finishing within their deadlines as the number of arrived jobs increases (Facebook scenario).

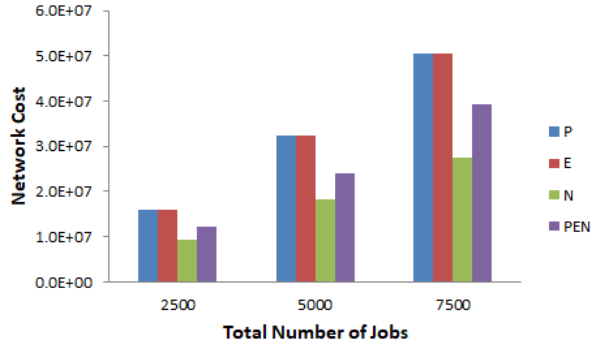

Fig. 8. Network cost as the number of arrived jobs increases (Facebook scenario).

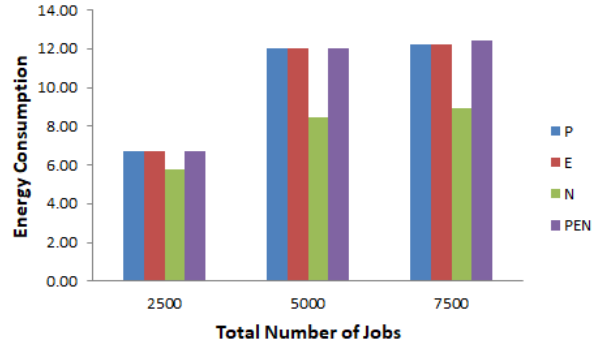

Fig. 9. Energy consumption as the number of arrived jobs increases (Facebook scenario).

improvement over both P and E heuristics (about 4x less overhead). Compared to N that assigns jobs to the optimal (network-wise) datacenters, PEN incurs roughly between 2x and 2.5x network cost (Fig. 5), but accounts for better acceptance rates as shown in Fig. 4. As far as the energy criterion is concerned, Fig. 6 shows that especially when computational resources are limited (e.g., 1x), E achieves the best performance with no clear winner for the second place.

Viewing the first results in retrospect, we can state that the merits of a scheduler operating with PEN are demonstrated by the fact that maximum job acceptance rate was achievable while cutting down significantly network overhead and without affecting at large energy consumption. In the second experiment we evaluated the algorithms for the file transcoding Facebook scenario. We considered 20 servers per datacenter assigned for the transcoding service and plotted results as the number of arrived jobs increases. For each job we assumed a deadline equaling the sequence duration plus 10 minutes. Figs. 7, 8 and 9, depict the percentage of jobs that meet their deadlines, the network and energy overhead respectively. Again PEN achieves top performance as far as processing time and deadlines are concerned (Fig. 7) but at a smaller network overhead (Fig. 8) compared to the rest.

VI. Conclusions

In this paper we tackled the problem of scheduling video transcoding jobs over a Cloud based service spanning multiple datacenters. The proposed schemes were evaluated through

simulation experiments with two realistic setups, one involving live broadcasts and one involving batch file transcoding for a social media provider. Results show that a heuristic (PEN) that tackles all three optimization targets, i.e., processing time, energy and network efficiency, using a Pareto-optimal strategy bears the largest merits.

ACKNOWLEDGMENT

Nikos Tziritas' Post-doctoral research was carried out with an IKY scholarship funded by the Action "Supporting Post-doctoral Researchers" from EP resources "Development of Human Resources, Education and Lifelong Learning" with priority axes 6,8,9 and co-funded by the European Social Fund - ESF and the Greek government

Samee U. Khan's work was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Cisco Systems Inc. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 (White Paper). Available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[2] E. de la Torre, R. Rodriguez-Sanchez and J. L. Martínez, "Fast video transcoding from HEVC to VP9," IEEE Trans. on Consumer Electronics, vol. 61, no. 3, pp. 336-343, Aug. 2015.

[3] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[4] Alliance for Open Media. Available at: http://aomedia.org

[5] Future Video Coding (JVET). Available at: https://jvet.hhi.fraunhofer.de

[6] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560–576, Jul. 2003.

[7] M. Koziri, P. Papadopoulos, N. Tziritas, A.N. Dadaliaris, T. Loukopoulos, and G.I. Stamoulis, "A framework for scheduling the encoding of multiple smart user videos," in Proc. SMAP 2016, pp. 29-34.

[8] A. Aaron, Z. Li, M. Manohara, J. De Cock, and D. Ronca, "Per Title Encode-Optimization", Netflix Tech blog, 14 Dec. 2015. Available at: http://techblog.netflix.com/2015/12/per-title-encode-optimization.html

[9] G. Terzopoulos, and H.D. Karatza, "Power-aware bag-of-tasks scheduling on heterogeneous platforms," Cluster Computing, vol. 19, no. 2, pp. 615-631, 2016.

[10] W. Lang, and J.M. Patel, "Energy management for MapReduce clusters," PVLDB, vol.3, no. 1, pp. 129-139, 2010.

[11] J. Leverich, and C. Kozyrakis, "On the energy (in)efficiency of Hadoop clusters," Operating Systems Review, vol. 44, no.1, pp. 61-65, 2010.

[12] A.A. Chandio, K. Bilal, N. Tziritas, Z. Yu, Q. Jiang, S.U. Khan, and C.-Z. Xu, "A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems," Cluster Computing, vol. 17, no.4, pp. 1349-1367, 2014.

[13] S. Ibrahim, T.-D. Phan, A. Carpen-Amarie, H.-E. Chihoub, D. Moise, and G. Antoniu, "Governing energy consumption in Hadoop through CPU frequency scaling: An analysis," Future Generation Comp. Syst., vol. 54, pp. 219-232, 2016.

[14] N. Zhu, L. Rao, X. Liu, J. Liu, and H. Guan, "Taming power peaks in MapReduce clusters," SIGCOMM 2011, pp. 416-417.

[15] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," IEEE Communications Surveys and Tutorials, vol. 18, no.1, pp. 732-794, 2016.

[16] C. Delimitrou, and C. Kozyrakis, "The Netflix challenge: Datacenter edition," Computer Architecture Letters, vol. 12, no.1, pp. 29-32, 2013.

[17] A. Beloglazov, R. Buyya, Y.C. Lee, and A.Y. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," Advances in Computers, vol. 82, pp. 47-111, 2011.

[18] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P.P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q.M. Malluhi, N. Tziritas, A. Vishnu, S.U. Khan, and A.Y. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," Computing, vol. 98, no.7, pp. 751-774, 2016.

[19] M.G. Koziri, P. Papadopoulos, N. Tziritas, A.N. Dadaliaris, T. Loukopoulos, and S.U. Khan, "Slice-Based Parallelization in HEVC Encoding: Realizing the Potential through Efficient Load Balancing," in Proc. MMSP 2016, pp. 1-6.

[20] M. Koziri, P. K. Papadopoulos, N. Tziritas, N. Giachoudis, T. Loukopoulos, S. U. Khan, and G.I. Stamoulis, "Heuristics for Tile Parallelism in HEVC," Proc. 25th European Signal Processing Conf. (EUSIPCO 2017), IEEE, Kos, Greece, Aug. 2017, pp. 1514-1518.

[21] M. Shafique, M. U.K. Khan, and J. Henkel, "Power efficient and workload balanced tiling for parallelized high efficiency video coding," in Proc. ICIP 2014, pp. 1253-1257.

[22] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," EURASIP J. Image and Video Processing, vol. 16, 2014.

[23] M.G. Koziri, D. Zacharis, I. Katsavounidis, and N. Bellas, "Implementation of the AVS Video Decoder on a Heterogeneous Dual-Core SIMD Processor," IEEE Trans. Consumer Electronics, vol. 57, no. 2, pp. 673-681, May 2011.

[24] J. F. Franche and S. Coulombe, "Fast H.264 to HEVC transcoder based on post-order traversal of quadtree structure," in Proc. 2015 IEEE International Conference on Image Processing (ICIP), Quebec City, QC, 2015, pp. 477-481.

[25] M. Wang, P.P. Jayaraman, R. Ranjan, K. Mitra, M. Zhang, E. Li, S.U. Khan, M. Pathan, and D. Georgakopoulos, "An overview of Cloud based Content Delivery Networks: Research dimensions and state-of-the-art," Trans. Large-Scale Data and Knoledge Centered Systems XX, LNCS series, vol. 9070, pp.131-158, March 2015.

[26] M.R. Zakerinasab and M. Wang, "Dependency-aware distributed video transcoding in the cloud," in Proc. LCN 2015, pp. 245-252.

[27] W. Zhang, Y. Wen, J. Cai, D.O. Wu, "Toward Transcoding as a Service in a Multimedia Cloud: Energy-Efficient Job-Dispatching Algorithm," IEEE Trans. Vehicular Technology, vol. 63, no. 5, pp. 2002-2012, June 2014.

[28] R.A. Pardo, K. Pires, A. Blanc, and G. Simon, "Transcoding Live Adaptive Video Streams at a Massive Scale in the Cloud," in Proc. MMSys 2015, pp. 49-60.

[29] G. Gao, W. Zhang, Y. Wen, Z. Wang, and W. Zhu, "Towards Cost-Efficient Video Transcoding in Media Cloud: Insights Learned From User Viewing Patterns," IEEE Trans. Multimedia, vol. 17, no. 8, pp. 1286-1296, Aug. 2015.

[30] J. De Cock, A. Mavlankar, A. Moorthy, and A. Aaron, "A Large-Scale Video Codec Comparison of x264, x265 and libvpx for Practical VOD Applications," SPIE Applications of Digital Image Processing XXXIX, vol. 9971, 997116. Sept. 2016. doi: 10.1117/12.2238495.

[31] Tubular Labs Inc. Available at: https://tubularlabs.com/leaderboards?type=overall&platform=facebook