MAMUT: Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-User Video Transcoding

Luis Costero*, Arman Iranfar†, Marina Zapater†, Francisco D. Igual*, Katzalin Olcoz* and David Atienza†
* Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Spain
† Embedded Systems Laboratory (ESL), Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
* {lcostero, figual, katzalin}@ucm.es † {arman.iranfar, marina.zapater, david.atienza}@epfl.ch

***Abstract*—Real-time video transcoding has recently raised as a valid alternative to address the ever-increasing demand for video contents in servers' infrastructures in current multi-user environments. High Efficiency Video Coding (HEVC) makes efficient online transcoding feasible as it enhances user experience by providing the adequate video configuration, reduces pressure on the network, and minimizes inefficient and costly video storage. However, the computational complexity of HEVC, together with its myriad of configuration parameters, raises challenges for power management, throughput control, and Quality of Service (QoS) satisfaction. This is particularly challenging in multi-user environments where multiple users with different resolution demands and bandwidth constraints need to be served simultaneously. In this work, we present** MAMUT**, a multi-agent machine learning approach to tackle these challenges. Our proposal breaks the design space composed of run-time adaptation of the transcoder and system parameters into smaller sub-spaces that can be explored in a reasonable time by individual agents. While working cooperatively, each agent is in charge of learning and applying the optimal values for internal HEVC and system-wide parameters. In particular,** MAMUT **dynamically tunes Quantization Parameter, selects number of threads per video, and sets the operating frequency with throughput and video quality objectives under compression and power consumption constraints. We implement** MAMUT **on an enterprise multicore server and compare equivalent scenarios to state-of-the-art alternative approaches. The obtained results reveal that** MAMUT **consistently attains up to 8x improvement in terms of FPS violations (and thus Quality of Service), 24% power reduction, as well as faster and more accurate adaptation both to the video contents and available resources.**

I. Introduction

In 2015, real-time entertainment already accounted for more than 74% of downstream network traffic in North America, with streaming services, including Netflix, YouTube, and Amazon Video, accounting for 57% of the global share [1]. Moreover, video streaming services continue to grow, and users are shifting towards the use of emerging video technologies, such as 4K video resolution; thus North America is expected to be the first region surpassing the 80% downstream streaming traffic threshold by the end of 2020 [1]. As a result of the network pressure posed by video streaming services, a shift to next generation video encoding standards, such as High Efficiency Video Coding (HEVC), is vital. HEVC provides twice the compression as of its predecessor, while keeping the same video quality [2].

However, such a considerable reduction in bandwidth requirement is accompanied by a 10x higher computational complexity. This fact poses challenges of time and energy consumption on the video providers' servers. As a result of diversity in video formats, users' devices, and network bandwidth, media adaptation is required to transcode the original encoded videos to a new version in order to satisfy the resource constraints (e.g., bandwidth and resolution) and users' preference. A video transcoder consist of a decoder followed by an encoder that changes a bitstream from one format to another. Today, multiple versions of the same video are stored in different formats and only the best one based on the user's demand is delivered. However, the fact that users daily upload more than 65 years of content to YouTube servers [3] whereas on average only the first 23 seconds of a video are watched (by Delmondo.co), implies inefficient and costly storage usage of such an approach. A promising solution is real-time video transcoding, which re-encodes the original video on the fly.

The bottleneck for achieving real-time HEVC transcoding is the encoder complexity, which is approximately 100x higher than that of the decoder [4]. Moreover, the numerous parameters available for adjusting the output quality and throughput add extra complexity. Finally, dealing with multi-user environments, where multiple different encoding requests have to be fulfilled simultaneously, poses other challenges on video providers' servers. While several works have tried to address efficient HEVC encoding through heuristics, machine learning, and model-based dynamic programming algorithms, none of them provide real-time HEVC transcoding in a multi-user environment.

In this work, we present MAMUT, a multi-agent Q-Learning (QL)-based run-time management strategy for QoS-aware real-time video transcoding in multi-user environments. In our approach, we decompose the design space into simpler sub-spaces, which lets us explore a larger design space. Each agent is able to independently explore a particular design sub-space to attain sufficient knowledge about the environment faster. Each agent, however, exploits its own knowledge jointly with the others to behave optimally along with all agents in the environment. In particular, we consider video quality and throughput as objectives, and power consumption and video compression as constraints. Our proposed approach consist of three agents for adapting the Quantization Parameter (QP) of the HEVC encoder, deciding for number of threads to encode each frame, and setting the processor frequency via Dynamic Voltage and Frequency Scaling (DVFS).

Our main contributions to the state of the art are as follows:

- we show how adaptation of application- and system-level parameters can be decomposed to provide fast and efficient run-time management of multi-user real-time

This work has been supported by the EU (FEDER) and Spanish MINECO (GA. No. TIN2015-65277-R), the Spanish MECD (GA. No. FPU15/02050), the EC H2020 MANGO (GA No. 671668), and RECIPE (GA No. 801137), and the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657).

HEVC transcoding.

- we implement MAMUT on a real multicore server, showing the behaviour of our approach when serving multiple simultaneous users while achieving real-time transcoding.
- we compare MAMUT against the state of the art through realistic scenarios. Our results show how a multi-agent reinforcement learning solution outperforms conventional monoagent machine learning and heuristic solutions, reducing power consumption up to 7% and 24%, and increasing the QoS achieved up to 5x and 8x, respectively.

II. Related Work

Several implementations of HEVC encoder exist, ranging from the non-real-time HM Test Model [5] as the reference software, to Kvazaar [6] and x265, both of which are able to provide real-time HEVC encoding through thread-based parallel processing. Real-time HEVC decoding has been studied and accomplished in the past [7], mainly through hardware acceleration.

HEVC encoders are composed of several processing blocks, each of which has multiple tuning parameters. Each parameter affects the encoder throughput (encoded frames per second, FPS), video quality (measured in Peak Signal-to-Noise Ratio, PSNR), video compression (bitrate), and power consumption [8]. In addition, video format and contents play a major role in throughput, video quality and compression, and chip power consumption. Since video contents may vary frame by frame, encoding parameters should be adapted on a frame basis for QoS optimization under limited power budget. Based on these facts, recent works [9], [10], [11] employed run-time adaptation of encoding parameters. These works, however, neither address QoS-aware real-time HEVC encoding nor consider a multi-user environment. Recently, a quality-aware power and thermal management approach has been proposed for non-real-time multistream HEVC encoding using machine learning techniques [8]. [12] proposes an online video transcoding framework targeting bio-medical videos in a multi-user environment. Nevertheless, inherent features of bio-medical videos make them a special case allowing for certain software simplifications that make online transcoding feasible. Moreover, as high-resolution videos were not evaluated, the functionality of this solution is not proven for these cases.

Finally, previous works (e.g. [10]) have modelled the output and complexity of the HEVC encoder as a function of a few relevant encoding parameters by exhaustive profiling of the application. However, these models are considerably platform-dependent and any change in the platform architecture may result in intolerable model error. A multi-user environment adds to the complexity and inefficiency of such models, since under limited resources and power budget, the parameters set for one video/user should be dynamically adjusted with respect to the encoding parameters set for other videos.

III. MAMUT: A Multi-Agent Machine Learning System for QoS-Aware Run-time Management

In MAL, multiple agents need to interact and behave cooperatively or competitively with some degree of autonomy.

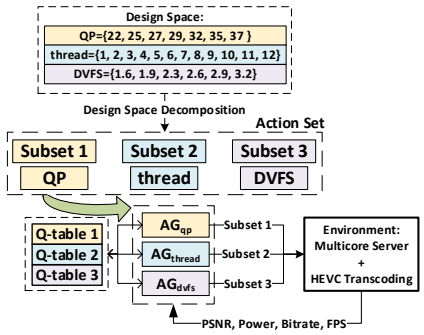

Figure 1. Proposed multi-agent RL approach (MAMUT).

The problem domain may be decomposed into smaller sub-problems and each agent takes charge of one of them, while communicating and interacting with other agents. As a result of such *cooperative* and *concurrent* learning, cooperative multi-agent learning is a promising solution to explore larger design spaces with less computational complexity, leading to a faster learning phase compared to mono-agent learning.

In this work, we leverage QL (*Q-Learning*) as a model-free RL (*Reinforcement Learning*) algorithm, since, compared to other RL algorithms, it is exploration-insensitive, thus, more suitable for practical problems [13]. Similar to conventional mono-agent learning, the QL algorithm in multi-agent learning is composed of a finite action set $\boldsymbol{A}$, and a finite state space $\boldsymbol{S}$. Each agent i is in charge of taking action a_t^i, and moves from its current state s_t to the next one s_{t+1}. Then, the corresponding Q-table is updated after each reward, indicating the value of applying a_t^i at s_t, is received.

In this work, we propose a concurrent cooperative multi-agent approach for run-time adaptation of HEVC encoding configuration and system parameters to achieve QoS-aware real-time HEVC transcoding under power budget constraints. Fig. 1 shows an overview of the proposed approach where three agents cooperate with each other. The environment is composed of two parts: application (HEVC transcoder) and platform (i.e., server). The action set $\boldsymbol{A}$ is split to three subsets A_1, A_2, A_3 such that $\forall i \neq j$, $A_i \cap A_j = \emptyset$, and $\bigcup_{i=1}^{3} A_i = \boldsymbol{A}$. Agents can send messages such that each agent accesses the Q-table of the others. In addition, states and rewards resulting from one agent's action are observable to all agents.

A. Agents

In MAMUT, we consider three different agents for tuning QP (AG_{qp}), deciding the number of threads used to encode a frame (AG_{thread}) through Wavefront Parallel Processing (WPP) [14], and per-core DVFS, (AG_{dvfs}).

B. Actions

a) QP: QP is one of the most important encoding parameters, as it affects FPS, PSNR, and bitrate [12], [9]. Although QP can take a wide range of values , we use QP values of 22, 25, 27, 29, 32, 35, and 37 based on our observation on the output PSNR, bitrate, and throughput.

b) Number of Threads: While HEVC encoding can always benefit from multithreading to increase FPS, Fig. 2 shows that throughput saturates above a certain number of

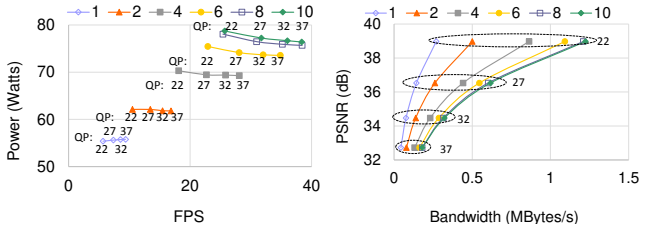

Figure 2. RD-curves, power, and throughput with respect to number of threads: 1, 2, 4, 6, 8, and 10 and QP values: 22, 27, 32, and 37 while encoding a 1080p-video at 3.2GHz using Kvazaar with the *ultrafast* configuration.

threads. Based on this observations, we consider a limited number of threads, as described in Section V.

c) DVFS: Our specific platform (see Section V) supports frequencies from 1.20 GHz to 3.2GHz. However, frequencies below 1.6 GHz can not provide real-time HEVC transcoding even if all constraints such as bandwidth and PSNR are released, and are therefore discarded.

d) Agents Sequence: We experimentally determine how frequently each agent should act, based on overhead, impact on our target objectives, and the number of parameter values to be explored as it is desirable that all agents finish the exploration phase at the same time. For our setup, AG_{qp} acts every 24 frames. With one frame as the offset, AG_{thread} takes action every 12 frames. AG_{dvfs} takes action every 6 frames with an offset of 2 frames. Since AG_{dvfs} and AG_{thread} act after AG_{qp}, they can modify the output throughput if it is degraded (or above the required constraints) because of AG_{qp} taking an action to increase (decrease) the video quality. In addition, as AG_{dvfs} takes actions more frequently, it can take charge of content variations and tune the throughput to the desired FPS. Figure 3 shows the proposed sequence for the agents.

C. States

Agents observe the output bitrate, PSNR, throughput, and power as states. Since for 8-bit-depth videos and lossy compression PSNR should range from 30 dB to 50 dB for acceptable human vision, we divide this range in the following intervals to constitute PSNR states (S_{psnr}): $PSNR \leq 30$, ≤ 35, ≤ 40, ≤ 45, ≤ 50, and > 50 dB. Power state (S_{power}) is defined based on the power consumption constraints of the running server: $power < P_{cap}$ and $power \geq P_{cap}$. The user's available bandwidth is highly affected by different parameters such as the contract, location, etc. In order to take into account these parameters, we consider three different bitrate states (S_{br}) based on the usual bandwidth provided by a 3G network [15]: $bitrate <$ 3Mb/s, 3Mb/s $\leq bitrate \leq$ 6Mb/s and $bitrate >$ 6Mb/s. Finally, the throughput (measured in FPS) is divided into the following states, since the target frame rate is 24: $fps < 24, < 26, < 28, < 30$ and ≥ 30.

D. Reward Function

In order to provide suitable feedback to each of the agents, we need to define four reward functions, one for each state:

a) Throughput: We define the following reward function based on the target frame rate (FPS_{target}):

$$R_{FPS} = \begin{cases} -4 & FPS < FPS_{target} \\ \frac{1}{FPS-(FPS_{target}-1)} & \text{otherwise} \end{cases} \quad (1)$$

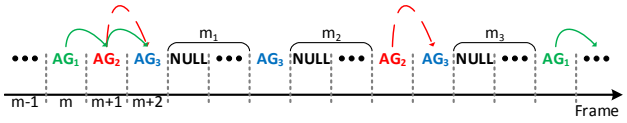

Figure 3. Agent sequence. Colored arrows show which agents need to look at the Q-table of the next agent.

This reward function provides negative values if the throughput is smaller than the target frame rate. The highest reward function is achieved if FPS exactly meets the target, however, if it is larger than FPS_{target} a smaller yet positive reward is provided. The reason is that achieving larger FPS may result in wasting resources, which ultimately means fewer users can be served. In the case where $FPS > FPS_{target}$, spare encoded frames can be buffered. Buffered frames can be used to compensate the overall framerate if, at some points, FPS temporarily drops below the target.

b) PSNR: As explained in Section III-C, a minimum PSNR of 30 dB is required. However, the goal of this work is to achieve higher video quality if there are enough resources. Hence, a higher reward is given when the agent moves to a state with larger PSNR, as follows:

$$R_{PSNR} = \begin{cases} -4 & \text{PSNR} < 30 \text{ or PSNR} > 50 \\ a \times e^{\text{PSNR}/50} - b & \text{otherwise} \end{cases} \quad (2)$$

where a and b are set to give a maximum reward of 1.0 when PSNR=50, and a reward of 0 when PSNR=30.

c) Bitrate and Power: The bitrate and power consumption are limited by the user's bandwidth and a power cap defined by the server manager (P_{cap}), respectively. Thus, we propose a reward function where a value of -4 is given if the constraint is violated, and 0.0 otherwise.

IV. Learning Phases and Learning Rate Function

A. Exploration and Exploration-Exploitation Phases

Since each agent has its own action set, we let the agents explore only state-action pairs corresponding to their own actions. As we need to deal with a stochastic environment, applying action a_t^i by AG_i at state s_t may not always result in a particular s_{t+1}. The reason lies in the fact that 1) contents of a video can change from one frame to another, 2) other agents taking charge of a single video may apply an action that alters the next expected state to a different one, and 3) other videos existing in a multi-user platform with their corresponding contents and agents can change the state unexpectedly. Thus, once a_t^i is taken at state s_t, all state transitions to new states need to be recorded during the exploration phase. Assume that $Num(s_t \xrightarrow{a_t^i} s_{t+1})$ shows number of times that applying a_t^i at s_t resulted in s_{t+1}, and $Num(s_t, a_t^i)$ represents total number of times that a_t^i was taken at state s_t. Then, the probability by which, after taking a_t^i at s_t, the agent observes s_{t+1} is $P(s_t \xrightarrow{a_t^i} s_{t+1}) = Num(s_t \xrightarrow{a_t^i} s_{t+1})/Num(s_t, a_t^i)$. This probability is updated throughout the learning process.

Whenever agent AG_i takes an action right before a frame starts, the next state (s_{t+1}) is observable right at the end of the frame by all agents. However, the immediate reward is only used to update the Q-value corresponding to (s_t, a_t^i). Then, the

following agent takes a random action in s_{t+1} and the same procedure in observing states and updating Q-table is followed. However, when an agent is followed by no other agents (shown as NULL in Fig. 3) the next observable state is the average of states containing the NULL action. This approach leads the agents to learn more about each others' behavior rather than about rapid video content variation, which can be regarded as noise in this case.

When the learning rate for each state-action pair drops below a threshold, α_{th1}, the agents start exploration-exploitation for that particular state. In this phase, agents do not take random actions, though after applying this particular action the Q-table is updated.

B. Learning Rate

Each agent must have its own learning rate for each state-action pair. The proposed learning rate function is a decreasing function of the number of state-action observations, differently from those proposed by the literature [8], [16], [17]. The reason is that if a learning rate function similar to the literature is considered, it is likely that an agent claims the end of the exploration phase even if other agents have not taken enough different actions. This issue ultimately makes one or more agents behave sub-optimally as taking action a_t in state s_t may not move the agent to state s_{t+1} as it expects. Thus, the agent cannot maximize the reward by following the Q-table.

Alternatively, we use the following learning rate function for each agent, AG_i, which allows each agent to monitor the number and variety of actions taken by other agents:

$$\alpha^{(i)}(s_t, a_t^i) = \frac{\beta_i}{Num(s_t, a_t^i)} + \frac{\beta_i^{'}}{1 + \sum_{j \neq i} (\min_{a \in A_j} (Num(a)))} \quad (3)$$

Here, the first term is taken from the literature [17], while in the second one $Num(a)$ is the number of times agent A_j has taken action a. Then, $\min_{a \in A_j}(Num(a))$ gives the minimum number of times that all actions available to AG_j have been selected. Subsequently, constants β_i and $\beta_i^{'}$ need to be set such that the exploration phase for (s_t, a_t^i) cannot finish until the following two conditions are satisfied: 1) (s_t, a_t^i) is observed so many times that $\frac{\beta_i}{Num(s_t, a_t^i)}$ can drop below a threshold and, 2) other agents have tried all their actions (at least once).

Due to the different frequencies at which each agent takes an action, in addition to the different sizes of the sub-spaces each agent has to explore, the learning rate parameters can vary from one agent to the other. In this work, we experimentally set $\beta_i = 0.3$ and $\beta_i^{'} = 0.2$, $\alpha_{th1} = 0.1$ and $\alpha_{th2} = 0.05$, and $\gamma = 0.6$. γ is the discount factor, that controls the significance of the history of Q-values vs. recently obtained rewards.

C. Exploitation Phase

The exploitation phase starts when the learning rate drops below a threshold, α_{th2}. Entering the exploitation phase, however, does not mean that exploration is finished. In fact, whenever a new state is observed by one agent, exploration phase starts for this particular state.

Algorithm 1: Exploitation phase

Input : Q^i, $P(s_t \xrightarrow{a_t^i} s_{t+1})$, $\mathbf{A}$; // $i \in \{1, \ldots, N\}$
Output: a_t^{i*} ; // current action taken by the i^{th} agent

$a_t^{i*} \leftarrow \underset{a \in A_i^*}{\arg\max} \left(\sum P(s_t \xrightarrow{a} s'_{t+1}) \times \boldsymbol{E}[Q_{Value}(AG_i.next(), s'_{t+1})] \right)$

function $\boldsymbol{E}[Q_{Value}(AG, s)]$**:** // list of agents, state

  if *(AG.next() == NULL)* **then**
    return $\max_{a \in A_{AG}^*} \left(Q^{AG}(s, a) \right)$
  else
    $a \leftarrow \underset{a \in A_{AG}^*}{\arg\max} \left(Q^{AG}(s, a) \right)$
    return $\left(\sum P^{AG}(s \xrightarrow{a} s') \times \boldsymbol{E}[Q_{Value}(AG.next(), s')] \right)$

Although each agent learns separately and has its own Q-table, it needs to act in the exploitation phase cooperatively and, as explained in Section III-B, in sequence. Consequently, the goal of each agent is not simply maximizing the Q-value attainable for its own Q-table, but rather, maximizing the expected Q-value after a sequence of actions taken by all agents. Imagine the sequence of agents shown as in Fig. 3. Starting from the m^{th} frame, the first agent, AG_1, is followed by two different agents, AG_2 and AG_3. Thus, the action taken by AG_1 should consider the probable transitions from one state to the other throughout the entire chain, composed of these three agents, in order to maximize the Q-value. Indeed, AG_1 should select an action which ultimately moves the entire system to a state in which an action taken by AG_3 is capable of providing the highest Q-value. This is equivalent to considering the expected Q-value given that a particular action is selected by AG_1. Hence, the conditional expected Q-values should be computed for all available actions in the current state s_t, in the chain of $AG_1 \rightarrow AG_2 \rightarrow AG_3$, as shown in Algorithm 1.

Moreover, it is possible that an agent moves to the exploitation phase earlier than the others since the number of actions that belongs to each agent is different. In such a case, the first agent in the sequence cannot rely on the behavior of the following agents. Hence, it only follows its own Q-table regardless of the expected Q-value that is achievable at the end of the sequence. Clearly, this behavior is not optimal as the whole system is not in the exploitation phase yet.

V. Experimental Setup and Results

A. Experimental Platform

In our experiments, trying to cover the extreme cases of the design space, we consider videos with two very different resolutions: High Resolution (HR) (1080p/fullHD videos with resolution 1920×1080 pixels), and Low Resolution (LR) (832×480) videos. The specific video sequences have been extracted from the JCT-VC benchmark [18]. We perform our experiments on a 16-core (32-thread) server composed of two Intel Xeon E5-2667 v4 CPUs. The measured overhead introduced by the system is negligible (i.e, less than 0.05% of the encoding time). As explained in Section III-B, the number

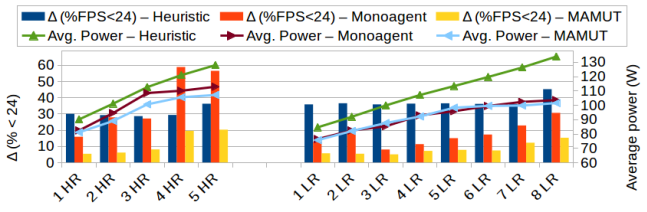

Figure 4. Δ - QoS (in terms of percentage of frames under QoS threshold) and power consumption for the heuristic, mono-agent and MAMUT encoding different combinations of HR and LR videos.

of threads that an encoding process can use before reaching saturation depends on its resolution. In our target platform, this limit appears for 12 threads in the case of an HR video, and 5 threads in the case of an LR video. Per-core DVFS is available with frequencies ranging from 1.2 GHz to 3.2 GHz. However, our observations reveal that frequencies below 1.6 GHz do not allow real-time transcoding.take.

For the sake of comparison, we have implemented two additional alternatives to our multi-agent approach. First, we have adapted the mono-agent-QL approach [8] such that the agent uses the entire action set, i.e., all possible combinations of all available actions in the design space. Because of the combinatorial explosion in the number of actions, which makes it unfeasible to train the system in a reasonable amount of time, a representative subset was chosen, ranging the same interval as the original actions, but with less granularity. Second, we have implemented a heuristic approach [19] that sets the number of threads (targeting FPS), adapts QP (targeting PSNR), and applies DVFS (for power management). Both systems act at the same frequency then the fastest agent in our system (i.e. every 6 frames). We use the Kvazaar open source encoder as the baseline of this work, using the default *ultrafast* configuration for HR videos, and the default *slow* configuration for the LR videos.

Finally, we consider two different scenarios to evaluate MAMUT and compare it against the mono-agent-QL and heuristic approaches described before. All reported results hereafter are extracted after five repetitions of the transcoding process under equal conditions, reporting the average values.

B. Scenario I: Serving Videos of Different Resolutions and Contents Dynamism

In the first scenario we assume that several different number of videos with different resolutions and contents need to be served simultaneously. The goal is to analyze videos separately by resolution to confirm that MAMUT behaves as expected and to extract remarks and insights by sweeping CPU usage spectrum from 1 to the maximum number of simultaneous videos.

Figure 4 shows the power consumption (in Watts) and the QoS violations –Δ– (in terms of percentage of frames processed below the 24 FPS target frame rate) of MAMUT, compared with the mono-agent and the heuristic implementations when running different workloads. The specific combinations of workloads include HR videos (from 1 to 5 processed simultaneously –1HR to 5HR in the Figure–) and LR videos (from 1 to 8 –1LR to 8LR–). The results show how, for the same workload, our approach is consistently able to reduce

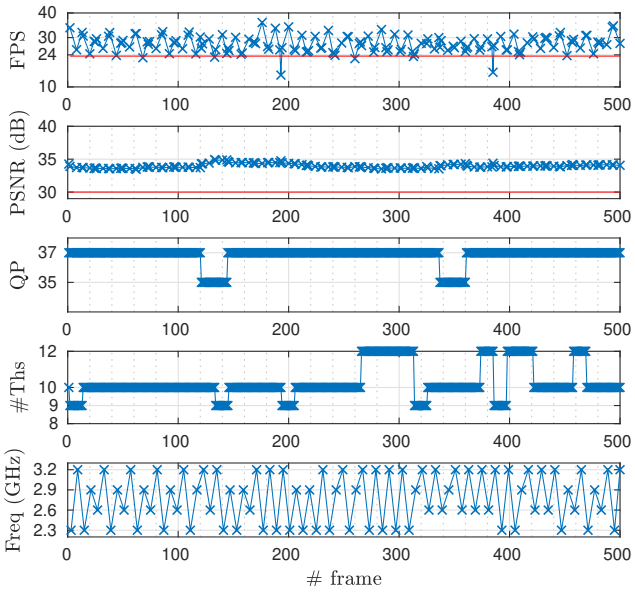

Figure 5. Detailed execution traces for MAMUT encoding an HR video.

Table I
NUMBER OF THREADS AND FREQUENCY USED IN AVERAGE

	MULTI-AGENT		MONO-AGENT		HEURISTIC	
	N_{th}	Freq.	N_{th}	Freq.	N_{th}	Freq.
HR	10.1	2.8	9.2	2.9	5.9	3.2
LR	3.7	2.8	3.2	2.7	2.6	3.2

power consumption between 10% to 24% when compared with the heuristic approach and up to 7% compared with the Monoagent implementation. A maximum improvement of 8x, and 5x in terms of FPS violations is achieved, when compared with the heuristic and the mono-agent implementation respectively.

Diving into details of the behaviour of each approach, we can extract a number of specific insights. First, when comparing MAMUT with the Heuristic approach, the runtime behaviour is inherently different. The Heuristic approach tries to achieve QoS requirements using maximum frequency and a low number of threads, whereas MAMUT encodes each frame using a higher number of threads, but lower frequency, as shown in Table I. This behaviour greatly improves power consumption.Figure 5 illustrates the behaviour of MAMUT when encoding an HR video. The number of threads does not change most of the time, while frequency changes continuously trying to keep the FPS close to 24, but never going below. Second, MAMUT is able to better use the available resources and adapt to different loads, as in situations in which the load is low it achieves much larger QoS with lower power, and under high load, it obtains slightly better QoS and saves power. Observe how, in the heuristic approach, QoS is almost constant and MAMUT manages to adapt the computation to the available resources (constantly achieving better results). Third, both ML-systems are able to learn a similar policy. However, since the number of actions in the mono-agent approach was limited, as described in Section V-A, it is not able to provide the same QoS measurements. Additionally,

Table II
SCENARIO II, AVERAGE RESULTS. EACH ROW REPORTS METRIC FOR A SEQUENCE OF A SPECIFIC COMBINATION OF VIDEOS.

	HEURISTIC				MONO-AGENT				MAMUT			
	Watts	N_{th}	FPS	Δ	Watts	N_{th}	FPS	Δ	Watts	N_{th}	FPS	Δ
1HR1LR	96.0	4.2	25.4	34.7	92.4	6.4	28.1	11.7	88.4	7.9	31.1	3.9
1HR2LR	106.3	4.1	24.6	32.6	96.7	5.6	26.4	22.7	93.4	6.5	31.3	6.2
2HR1LR	109.7	4.7	25.2	33.8	102.3	7.6	26.5	17.1	97.4	9.7	30.4	5.1
2HR2LR	114.5	4.5	25.0	33.7	105.4	6.6	25.8	25.9	100.3	7.6	29.5	11.0
2HR3LR	123.3	4.2	24.9	34.4	107.4	5.9	25.0	37.0	101.9	5.9	26.2	22.8
2HR4LR	124.5	4.4	25.1	33.4	108.5	5.8	24.7	44,6	100.9	6.3	27.7	24.1
3HR1LR	122.5	5.3	24.4	33.1	113.7	7.9	23.3	44.2	104.3	8.7	26.2	20.5
3HR2LR	129.9	5.4	24.5	34.0	110.9	7.0	21.7	65.9	105.2	7.0	25.3	31.2
3HR3LR	134.6	6.1	23.4	43.2	111.8	6.2	20.8	71.5	106.9	6.2	25.1	35.8

although the search space was reduced in our mono-agent implementation, the time taken to learn was 15 times larger, due to the combinatorial explosion in the number of state-action pairs to visit before the exploitation phase. The PSNR achieved for all the proposals is close to 34 dB in the case of HR videos, ranging from 36 dB when a LR video is encoded by a ML-system, to 41 dB when it is encoded by the heuristic approach. Finally, concerning bitrate and power, all the implementations met the constraints.

C. Scenario II: Serving Transcoding Requests Batches of Variable Resolution Requirements

In the second scenario, we assume that a set of transcoding requests is simultaneously received from different users with different resolution requirements. In contrast to the first scenario, here we consider sequences of random videos being simultaneously transcoded, simulating a real scenario where users are coming and going continuously. To restrict the extent of the experiment, we consider that each initial video is followed by a sequence of four different videos of the same resolution, randomly selected. With this scenario, we illustrate the capability of our approach to satisfy QoS requirements of different users with different demands during time. Also, given the random nature in video contents, we explore the capability of the approach in dealing with different video contents.

Table II shows the average values for the main metrics for a specific combination of video types in scenario II. Qualitatively, the behaviour of all implementations is similar to that in the previous scenario, and all are able to satisfy QoS requirements if the workload is not close to the resource saturation point, achieving an average PSNR $\approx$ 36dB in all approaches. When the machine is fully utilized (e.g. when transcoding three HR videos simultaneously), MAMUT still achieves the best results in terms of QoS, whereas the mono-agent system cannot meet the QoS requirements. The reason lies in that the latter is restricted to fewer number of actions; thus, it cannot adapt to all situations. MAMUT consumes 8% to 20%, and 4% and 7% less power when compared the heuristic and mono-agent approaches, respectively. Finally, QoS violations are reduced up to 8x and 4x compared with the heuristic and mono-agent approaches, respectively.

VI. CONCLUSION

In this work we presented MAMUT, a novel multi-agent reinforcement learning solution for efficient real-time multi-user video transcoding. Our solution adapts better to the environment and different video contents than alternative approaches by dividing the design space in different subspaces, each of them explored by one different agent, while working cooperatively with the others to decide the next actions to take. In our design, agents tackle both intrinsic parameters of the transcoding process (e.g. Quantization Parameter) or architectural parameters (number of threads and processor frequency). Our solution exhibits better results both in energy consumption (up to 24% when compared with a heuristic approach, and 7% compared with a mono-agent approach) and less QoS violations (up to 8x and 5x, respectively), while satisfying restrictions in power and compression. In other words, MAMUT is able to simultaneously and transparently improve QoS and resource usage with no user intervention in real time scenarios, where multiple transcoding requests have to be served simultaneously.

REFERENCES

[1] I. SANDVINE, "Global internet phenomena report. 2016," 2015.

[2] B. Bross, "High efficiency video coding (HEVC) text specification draft 9 (sodis)," in *11th JCT-VC meeting*, Oct 2012.

[3] "DMR youtube report," 2017. [Online]. Available: http://expandedramblings.com/youtube-statistics

[4] F. Bossen, B. Bross *et al.*, "HEVC complexity and implementation analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, 2012.

[5] P. Bordes, P. Andrivon *et al.*, "Joint collaborative team on video coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11," 2016. [Online]. Available: https://HEVC.hhi.fraunhofer.de

[6] M. Viitanen, A. Koivula *et al.*, "Kvazaar: Open-source HEVC/h. 265 encoder," in *ACM on Multimedia Conference*, 2016, pp. 1179–1182.

[7] M. Abeydeera, M. Karunaratne *et al.*, "4K real-time HEVC decoder on an FPGA," *IEEE Trans. Circuits Syst. Video Tech.*, vol. 26, pp. 236–249, 2016.

[8] A. Iranfar, M. Zapater, and D. Atienza, "Machine learning-based quality-aware power and thermal management of multistream HEVC encoding on multicore servers," *IEEE TPDS*, 2018.

[9] D. Palomino, M. Shafique *et al.*, "TONE: Adaptive temperature optimization for the next generation video encoders," in *Int. Symposium on Low power electronics and design (ISLPED)*, 2014, pp. 33–38.

[10] M. U. K. Khan, M. Shafique, and J. Henkel, "Power-efficient workload balancing for video applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2089–2102, 2016.

[11] J.-H. Hu, W.-H. Peng, and C.-H. Chung, "Reinforcement learning for hevc/h. 265 intra-frame rate control," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[12] A. Iranfar, A. Pahlevan *et al.*, "Online efficient bio-medical video transcoding on MPSoCs through content-aware workload allocation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 949–954.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[14] G. J. Sullivan, J.-R. Ohm *et al.*, "Overview of the high efficiency video coding(HEVC) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[15] D. Belson, "Akamais state of the internet," 2017. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf

[16] A. Iranfar, S. N. Shahsavani *et al.*, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous MPSoCs," in *Int. Symp. on Low Power Electronics and Design*, 2015.

[17] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 96, 2014.

[18] F. Bossen and H. Common, "Test conditions and software reference configurations," *JCT-VC Doc*, 2013.

[19] M. Grellert, M. Shafique *et al.*, "An adaptive workload management scheme for HEVC encoding," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE, 2013, pp. 1850–1854.