Resource Management for Power-Constrained HEVC Transcoding Using Reinforcement Learning

Luis Costero, Arman Iranfar *Student Member, IEEE*, Marina Zapater *Member, IEEE*, Francisco D. Igual, Katzalin Olcoz and David Atienza *Fellow, IEEE*

Abstract—The advent of online video streaming applications and services along with the users' demand for high-quality contents require High Efficiency Video Coding (HEVC), which provides higher video quality and more compression at the cost of increased complexity. On one hand, HEVC exposes a set of dynamically tunable parameters to provide trade-offs among Quality-of-Service (QoS), performance, and power consumption of multi-core servers on the video providers' data center. On the other hand, resource management of modern multi-core servers is in charge of adapting system-level parameters, such as operating frequency and multithreading, to deal with concurrent applications and their requirements. Therefore, efficient multi-user HEVC streaming necessitates joint adaptation of application- and system-level parameters. Nonetheless, dealing with such a large and dynamic design space is challenging and difficult to address through conventional resource management strategies. Thus, in this work, we develop a multi-agent Reinforcement Learning framework to jointly adjust application- and system-level parameters at runtime to satisfy the QoS of multi-user HEVC streaming in power-constrained servers. In particular, the design space, composed of all design parameters, is split into smaller independent sub-spaces. Each design sub-space is assigned to a particular agent so that it can explore it faster, yet accurately. The benefits of our approach are revealed in terms of adaptability and quality (with up to to $4\times$ improvements in terms of QoS when compared to a static resource management scheme), and learning time ($6\times$ faster than an equivalent mono-agent implementation). Finally, we show that the power-capping techniques formulated outperform the hardware-based power capping with respect to quality.

Index Terms—Resource Management, DVFS, Power Capping, Reinforcement learning, Q-Learning, HEVC, self-adaptation.

1 Introduction

The emergence of massively parallel and heterogeneous architectures forces the co-location of applications in order to exploit the potential underlying performance, usually under tight system-level (for example in terms of maximum power performance) or application-level limits (minimum Quality of Service). The development of holistic and autonomous resource management schemes to simultaneously adapt application- and system-wide knobs with real-time requirements is far from being a trivial task, but it will become mandatory for such systems. Fortunately, Artificial Intelligence (AI) techniques can provide great help in this type of scenarios. Actually, the development of agents that optimally learn and improve their behaviour in an autonomous fashion has traditionally been one of the paramount goals of AI. *Responsiveness* and *self-adaptation* to the environment transform AI systems into appealing pieces of software that can sense, interact and react to environmental changes without human intervention. Specifically, Reinforcement Learning (RL) [1] is a field of AI that aims at learning by interaction, featuring a so-called reward-driven behavior. In RL, autonomous agents proceed by interacting with their environment, progressively altering their behavior by observing the consequences of their actions in form of state-action-reward tuples. RL agents are in general well suited to problems appearing in complex scenarios featuring large state spaces, highly dynamic and without any pre-existing knowledge.

In this paper, we integrate RL techniques into a self-adaptive resource manager to tackle the problem of automatic and dynamic application- and system-wide knob adaptation for multi-user video transcoding scenarios on modern multi-core servers. Then, we demonstrate that RL is an effective and efficient technique to automatically extract and apply policies that simultaneously fulfill performance, quality, and power restrictions when targeting resource management on multiple application instances. Our proposal is based on a design space decomposition into sub-spaces, facilitating the coverage of a large design space. Thus, each agent independently explores a particular sub-space to attain sufficient knowledge about the environment faster. Once the design space is fully explored by all the agents, each agent exploits its internal knowledge jointly with others' knowledge in a cooperative manner to optimally behave in the environment.

Our proposal is based on a specific use case of wide appeal nowadays: multi-user video transcoding via a highly tuned HEVC encoder (Kvazaar [2]) modified to expose dy-

- *Luis Costero, Francisco D. Igual and Katzalin Olcoz are with the Departamento de Arquitectura de Computadores y Automática at the Universidad Complutense de Madrid, Spain. E-mail: {lcostero,figual,katzalin}@ucm.es*
- *Arman Iranfar, Marina Zapater and David Atienza are with the Embedded Systems Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. E-mail:{arman.iranfar,marina.zapater,david.atienza}@epfl.ch*
- *This work has been supported by the EU (FEDER) and Spanish MINECO (TIN2015-65277-R, RTI2018-093684-B-I00), and MECD (FPU15/02050), by Spanish CM (S2018/TCS-4423), the ERC Consolidator Grant COMPUSAPIEN (725657), and H2020 RECIPE (801137)*

namic application-level knobs. High Efficiency Video Coding (HEVC) has emerged as a feasible solution to alleviate the exponential growth in network traffic originated both for live streaming and video on demand, because it provides up to 50% better compression compared with their predecessors keeping video quality [3]. This reduction in bitrate comes at the cost of an increase in computing demands on the server side. As a consequence, the burden is shifting from network pressure to compute requirements in terms of both *performance* and *energy consumption* [4]. Moreover, video streams often have to be converted to match the requirements of different clients by means of online video transcoding, which is a very resource intensive process [5], [6]. In multi-user scenarios, video providers' servers receive multiple simultaneous transcoding requests, each with different quality or throughput restrictions. Many modern video transcoding systems expose a number of dynamically tunable knobs with direct implications in resource usage and attainable quality of service (QoS); similarly, modern techniques exposed by hardware also have considerable impact on application throughput and power consumption. Hence, a proper selection of application-side and system-level parameters to simultaneously fulfill QoS requirements of clients while optimizing resource usage becomes a challenging task to increase the productivity of the underlying computing platforms.

Our main contributions are:

- We present a detailed analysis of parameters of a real-time encoding process (*dynamic knobs*) and their impact in throughput, power consumption, and video quality for a modern, multi-threaded video encoder. We show how application- and system-level knobs can be decomposed and policies to apply on them can be automatically learned through RL to provide fast and efficient run-time management of a highly tuned HEVC encoder on a modern multi-core server.
- We integrate turbo frequency management and power capping abilities (software and hardware), and demonstrate that application-aware power capping can reach equal or better results than traditional hardware-based techniques [7], both in terms of QoS and power consumption.
- Targeting multi-user real-time HEVC transcoding, we validate our approach on a server with state-of-the-art power-management capabilities. Compared with an equivalent *static* knob selection approach, we achieve a reduction in the number of QoS violations up to $2\times$ for a single video, and up to $4\times$ for a fully loaded server.
- Our insights reveal the advantages of a multi-agent over a mono-agent implementation in learning speed ($6\times$ faster) and policies (12% improvement in QoS). In terms of QoE, the multi-agent approach achieves a reduction of $7\times$ in the number of changes in QP, and a 30% reduction on average QP distance, ultimately meaning a better user experience.
- Comparing with state-of-the-art heuristics for resource management on similar scenarios, we achieve a maximum improvement of $14\times$ in the number of QoS violations –$5.7\times$ on average–.

The rest of the paper is structured as follows: Section 2 provides a comprehensive study of HEVC as a case of study, and motivates the necessity of machine learning techniques to tackle the resource management problem. In Section 3 and 4, we provide a deep description of our proposal towards the formulation of the resource management problem in terms of RL in general, and as a Q-learning problem in particular, respectively. Section 5 describes the main caveats in the deployment of a multi-agent design within the framework. Sections 6 and 7 provide detailed experimental results for a comprehensive set of scenarios. Section 8 closes the paper with a summary of the main conclusions of this work.

2 Multi-user transcoding: resource management

HEVC transcoding involves decoding a video to encode it again with the requested features. The main complexity of online transcoding comes from the high computational complexity of the encoder [3]. Thus, we exclusively focus on resource management to HEVC encoders in order to optimize performance and energy of the servers while providing the required per-user QoS and quality of experience (QoE) [8].

2.1 Application- and system-wide knobs on multi-core servers

Modern video encoders implementing high performance versions of the HEVC standard (and similarly of other standard specifications) are composed of a set of basic building blocks each frame has to pass through to be encoded. Each basic building block is parameterized by means of input *knobs* with application-wide impact (e.g., throughput or encoding quality) and/or system-wide effects (e.g., power consumption). Some of them must be decided and statically fixed a priori with no option for runtime modification (for example the *preset* selection in many HEVC encoders, that ultimately tunes a number of static knobs before execution); others can be modified, with different granularity, at runtime (e.g., quantization parameter -QP- or number of threads). We will refer to them as *static* and *dynamic knobs*, respectively, following the idea proposed in [9]. Moreover, it is not necessary or feasible to tune all the available dynamic knobs to significantly affect throughput, quality, bit rate and/or power consumption. In this work, we limit our study to those with the largest impact on the output quality and performance, and we employ a particular high-performance implementation of HEVC, Kvazaar, to illustrate how our proposal can be integrated into a state-of-the-art HEVC encoder. Note, however, that the ideas presented in this paper can be mapped to other knobs within the HEVC standard with minimal changes, and to other high performance implementations (e.g. x265[1]), provided they provide similar selectable knobs associated to their building blocks.

1) Quantization Parameter (QP): in charge of the quantization degree per frame [10], plays a significant role in output responses and can be tuned on a frame-to-frame basis at runtime. Among others, Huang et al. [11], and Biatek et al. [12] have worked on proper QP estimation and selection. QP values in the range of 22 to 37 are suggested by JCT-VC [13] to yield desirable quality. However, even with

1. https://x265.org

this knowledge, the optimal value is still unknown to meet the required PSNR, bitrate, throughput, and power budget at runtime due to inter- and intra-video variations, and its adaptive selection is still a subject of detailed study in the literature [14], [15].

2) Thread parallelism in HEVC: A key feature of Kvazaar that enables real-time encoding is Wavefront Parallel Processing (WPP) [10]. Each frame is divided into different rows and blocks which are processed in raster-scan order. A pool of *worker threads* is deployed upon initialization. When a thread finishes processing a block, it checks whether there is a new block with no dependencies to be processed. Following this paradigm allows to dynamically modify the number of active worker threads at each moment of the execution.

Modern multi-core architectures also expose a number of *system-level* dynamic knobs. Its proper selection offer trade-offs between power consumption and performance, and a proper adaptation to variable workloads with imposed resource limits. These system-wide knobs include *(i)* DVFS (Dynamic Voltage-Frequency Scaling [16]) to dynamically select operating frequency of the processor, with direct implications on performance and energy consumption; *(ii) dynamic power capping* selection to limit the energy consumption based on internal hardware events to estimate instantaneous power draw; and *(iii) turbo management*, exposing extra turbo frequency ranges that can be leveraged to boost performance under specific application restrictions [17]. This extra power budget depends on both the number of active cores and the type of vector instructions delivered [18]. Hence, applications need to use turbo frequencies in a restricted and intelligent way, considering a full view of the system usage. This situation adds an extra challenge in runtime frequency management.

2.2 Motivation for dynamic resource and knob management

The development of a proactive, self-adaptive policy for resource management in a multi-user environment with multiple video requests is motivated by two main intrinsic characteristics of this kind of application: intra-video requirement variations and inter-video interactions. Let us illustrate this fact in terms of actual throughput (reported as frames per second or FPS), for different scenarios using a static resource assignment. The following results have been extracted on a real platform, later used to carry out all the experiments. The platform comprises a 20-core Intel Xeon server clocking from 1 GHz to 2 GHz and *turbo mode*. Frequency is selectable in 100MHz steps, and turbo frequency is limited up to 3.6 GHz as our encoder uses AVX2 instructions.

1) Intra-video requirements variability (due to content variation): Black line in Figure 1 reports a timeline of the observed throughput of a complete transcoding process of a high-resolution video (*QuarterBackSneak*, see Table 3) using static computing resources (3 threads at 1.5 GHz and $QP = 22$). Considering a target throughput of 24 FPS, video contents ultimately determine the instantaneous throughput attained, with areas in which the computing resources are sufficient (or even wasted) and others in which QoS violations appear frequently.

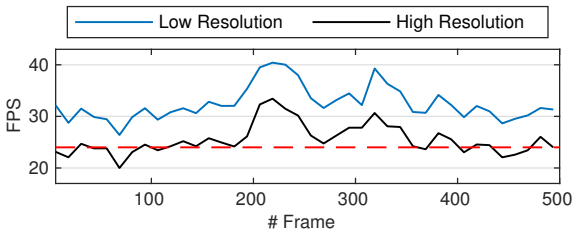


Figure 1: Timelines representing the throughput of the same sequence (*QuarterBackSneak*) with different resolutions: High (1280×720) and Low (832×480), when encoding with the same knobs (3 threads at 1.5GHz setting QP=22). The dotted line represents the real-time threshold (24 FPS).

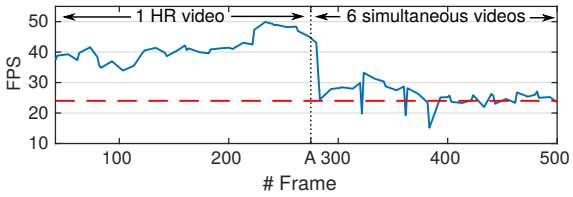


Figure 2: Timeline representing the instantaneous throughput of one High Resolution video (*QuarterBackSneak*) running with 3 threads and QP=22 at turbo frequency. Until point A, the video is encoded alone. From that point on, it is encoded simultaneously with other 5 videos (3 threads each).

2) Inter-video requirements variability (due to different resolution): Figure 1 shows the throughput obtained for the same video sequence with two different resolutions: High and Low resolution (black and blue lines) using the same values of knobs -number of threads, frequency and QP-. While the low resolution video transcoding process can achieve real time transcoding (i.e., 32.5 FPS on average), the high resolution video suffers from frequent QoS violations (i.e., 25.5 FPS).

3) Inter-video resource contention (due to changing number of requests): Figure 2 simulates a situation in which an ongoing isolated transcoding process with fixed assigned resources has its throughput diminished upon the appearance of other independent transcoding processes (point labelled as A in the Figure). The same sequence as that in Figure 1 is encoded with the same knobs except that in this case turbo frequency is used. When the process is executed in isolation, the throughput is much higher than required; but as soon as other transcoding processes appear in the server, and although each process is mapped to independent cores in the experiment, the achieved throughput is dramatically reduced. The attained throughput is thus highly sensitive to both general video characteristics, associated assigned resources and machine occupation. This type of scenario results in wasted resources when the process runs alone and constant quality of service violations when the server is loaded. In this situation, an increase in QP, for example, could balance throughput and quality. This combined decision, however, is not trivial.

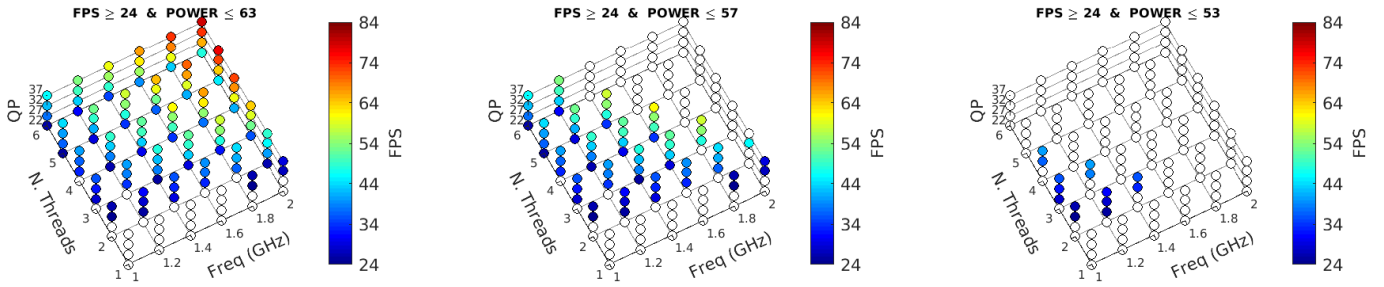


Figure 3: Average FPS for all combinations of QP, number of threads and frequency for three different values of power capping, when encoding the HR sequence *"FourPeople"*. Color-less points represent knob combinations not satisfying the constraints.

Table 1: Feasible knob combinations producing near-24 FPS encoding (on average), and impact on the other output metrics.

QP	N. ths	Freq (GHz)	FPS	Power (W)	PSNR
37	1	1.8	25.9	54.2	36.4
32	2	1.0	24.0	49.3	38.9
22	3	1.4	25.3	55.0	43.4
22	5	1.0	25.1	54.3	43.4

2.3 Necessity of ML for multi-user video transcoding

Figure 3 gives a quantitative and qualitative overview of the complexity and amplitude of the design space considering QP, number of threads and frequency for the encoding of a single video (*FourPeople*, see Table 3) under three different power caps. Each colored point in the figure represents a valid solution, with different throughput (from 24 FPS in dark blue to more than 80 FPS in red), power and quality. Areas in which both QoS and power restrictions are met ($FPS > 24$ and $power < power\_cap$) grow larger as power capping is relaxed, yielding more potential knob combinations. Considering only the dark blue points in Figure 3, which correspond to solutions close to 24 FPS on average, Table 1 summarizes different assigned resources and output metrics (averaged for the complete execution). Thus, for the best quality the chosen knobs would be $QP = 22$, 5 threads and 1 GHz, while the lowest power solution would be $QP = 32$, 2 threads and 1 GHz. Other combinations of knobs can also be the best option for different optimization goals.

The results above change for the different frames of the video, depending on video contents. Let us assume an encoding process with support for N_{enc} different encoding knobs values and N_{sys} different system parameters (e.g., frequency), which can be tuned at runtime, at a frame granularity. Consider now the encoding of a t-second video at a frame rate of F_r: in order to find the best encoding configuration for that frame, $N_{enc} \times N_{sys} \times t \times F_r$ profiles should be statically obtained. For instance, 4 QP values, 5 number of threads values, and 10 DVFS values, for a 10-second video at a frame rate of 24 FPS, would yield $48,000$ combinations to obtain the optimal encoding configuration and DVFS settings for the video, which gives a hint of the complexity of the brute-force approach.

In addition, configuring the encoding parameters is dramatically more challenging when considering multiple concurrent videos running on a server, in a so-called *multi-user video transcoding*. First, the inter-video resource contention impacts the throughput obtained by the independent encoding processes. Second, the power constraint may not let all encoding processes run at their own optimal configuration simultaneously. As a result, a joint profiling becomes mandatory.

Machine learning techniques are able to consider hidden and complicated interrelations of encoding parameters on any arbitrary platform along with video contents. This, however, requires a model-free learning algorithm, as provided by RL in general, and the *Q-Learning* (QL) algorithm in particular, as described in the following sections.

2.4 Related Work

From a generic perspective, our proposal gathers characteristics from resource managers and auto-tuning frameworks. Focusing on resource management, reinforcement learning has been previously applied both in embedded systems [19] and in large-scale distributed environments [20], [21] to take coarse-grain (heterogeneous or homogeneous) workload allocation. Multi-agent reinforcement learning has also been previously applied to address load balancing problems in grid computing resources for large-scale computing jobs [22]. None of the aforementioned efforts combines the fine granularity in knob selection proposed by our approach with multi-application management and contention/interaction considerations in intra-node resource management. Our results reveal that multi-agent RL is a valid technique for real-time fine-grained resource management.

Previous works on auto-tuning frameworks divide strategies into static (where knobs are predefined at installation time or at process level, see Active Harmony [23] or Autotune [24]), and dynamic (where knobs vary at runtime based on self-extracted knowledge, as in our case). In the framework of the ARGO project, mARGOt and a number of related efforts [25], [26] provide heuristics for dynamic knob selection at function level, without any support of AI, following reactive and proactive strategies to fulfill precision requirements. ADAPT [27] is based on compilation techniques to expose run-time opportunities for optimization, which are selected heuristically at execution time. PowerDial [9] proposes an integrated framework for

dynamic adaptation of application knobs to target load and power changes in the server. The concept of *dynamic knob* is introduced and exploited in the paper, but the runtime behaviour is reduced to heuristic knob tuning. Many of the auto-tuning frameworks consider dynamic knob adaptation by means of heuristic exploration; our proposal, on the contrary, demonstrates that RL can improve output application metrics without the need of complex heuristics.

Many of the aforementioned related efforts consider resource management and dynamic application auto-tuning as completely orthogonal efforts. On the contrary, we propose a *hybrid approach* between a resource manager and a dynamic auto-tuning framework. A detailed study of the strategies integrated in our framework reveal that resource management (e.g., frequency) and application auto-tuning (e.g., number of threads or QP) should be considered jointly to attain proper resource management and output application metrics.

Specifically targeting on video transcoding, previous works such as [28], [29], [30], have modeled the output and complexity of an HEVC encoder as a function of a few encoding parameters by exhaustive application profiling, considering a few encoding parameters due to the exponential complexity of dealing with a higher number. But these models are considerably platform-dependent and any change in the target architecture may result in an intolerable model error. Our previous work [31] proposes a mono-agent implementation of Q-Learning with a similar design for states, actions and reward functions, following different objectives (temperature and power control) without real-time restrictions and using an unoptimized (and sequential) HEVC implementation. While similar in philosophy, in this work we focus on the challenges imposed by designing and managing a multi-agent implementation following our previous effort in MAMUT [32] with tight restrictions in the target throughput, quality and system-wide power, on a fully-optimized HEVC implementation. Specifically, in this work we deeply extend the ideas presented in MAMUT, including a comprehensive section motivating the use of Reinforcement Learning techniques to deal with big design spaces, and a more detailed study of other similar approaches in the literature. We incorporate Turbo Frequency management into our formulation, forcing the redesign of the states to consider core occupation, and demonstrating that this complexity can be efficiently handled by an actual centralized resource manager with a negligible overhead. In addition, a detailed description of the design of the system is exposed, focusing on how the system deals with system-wide metrics and noisy measurements. Finally, we compare against a new mono-agent approach and a state-of-the-art heuristic in multiple scenarios, adding power capping capabilities to our system, thus illustrating how system-wide restrictions can also be handled by our RL formulation.

3 Reinforcement learning as a solution to dynamic resource allocation and self-adaptation

3.1 Reinforcement Learning. Mono-Agent Q-Learning

RL is appropriate for problem domains where reinforced information is available after a sequence of actions are performed in the environment. RL is able to deal with environment-dependent problems through dynamic optimization programming. Q-Learning, as a model-free algorithm of reinforcement learning, is able to cope with more sophisticated industrial problems. In addition, it is exploration-insensitive, thus, more suitable for practical problems [1].

A (mono-agent) Q-Learning model is composed of an agent (*learner*) able to select and take actions from a finite action set, **A**, and observing (*sensing*) its current state from a finite state space, **S**. The agent applies actions starting from an initial state and moving to a new one. Applying particular actions in particular states is encouraged or discouraged based on a *reward* received after moving to the new state. Starting from a usually random policy to select actions, the agent is ultimately able to follow a learned policy, π, which is a mapping from the state space to the action set. This mapping simply implies whether action a_t in state s_t is worthwhile to be applied.

a) Learning process – method: To learn the best policy, the agent maximizes the reward by storing a *Q-value* per state-action pair as $Q^{\pi}(s, a)$ indicates the quality of applying action a in state s. In other words, the *Q-value* represents the most probable long-term reward, provided the agent starts from state s, applies action a, and follows the policy π. All the different *Q-values* are stored in a *Q-table*, which is updated as follows [1]:

$$\begin{aligned} Q_{t+1}(s_t, a_t) = [1 - \alpha(s_t, a_t)] \times Q_t(s_t, a_t) + \\ \alpha(s_t, a_t) \times [R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)] \end{aligned} \quad (1)$$

where $Q_t(s_t, a_t)$ and $Q_{t+1}(s_t, a_t)$ are, respectively, the current and updated *Q-values* corresponding to the current taken action a_t at the current state s_t, R_{t+1} is the immediate reward after next state s_{t+1} is observed, $\alpha(s_t, a_t)$ determines the *learning rate*, and γ is the discount factor and controls the significance of the history of the *Q-values* against the recently obtained reward. The learning rate defined in Q-learning depends on the state-action pair. Then, through the learning rate definition, we ensure that a specific state-action pair has been observed a sufficiently large number of times.

In stochastic environments where action a_t at state s_t does not always result in a particular next state s_{t+1}, the learning rate is critical to ensure a fast and flawless learning phase. If the learning rate is assumed constant and set to 1, the previous reinforced information is overridden every time the state-action pair of (s_t, a_t) is observed. If the learning rate is constant and set to zero, there is no learning. For fully deterministic environments, $\alpha(s_t, a_t) = 1$ provides optimal learning. However, for stochastic problems [1], a decreasing-to-zero function for learning rate is able to provide optimal learning phase. A common definition used [33] is:

$$\alpha(s_t, a_t) = \beta / Num(s_t, a_t) \quad (2)$$

where β is a constant and $Num(s_t, a_t)$ is the number of observations of the state-action pair (s_t, a_t).

b) Learning process – phases: We consider three phases for the learning process similar to the literature [34], [35]. In this approach each pair state-action is updated based on the value of its learning rate. In the first phase, called *exploration* ($\alpha < \alpha_{th1}$), actions are taken randomly trying to explore all

the states and actions as quickly as possible. Once a state is explored enough times ($\alpha_{th1} < \alpha < \alpha_{th2}$), it moves to the *exploration-exploitation* phase, where now the taken action is the one that maximizes the Q-values learned in the previous phase ($a = \arg\max_{a \in \mathbf{A}} Q_t(s_t, a)$), at the same time the Q-tables are still updated. In the last phase, called *exploitation* ($\alpha_{th2} < \alpha$), the agent has already learned the final policy π, thus it takes the actions that maximize the Q-values similar to the previous phase, but not updating these values anymore.

3.2 Multi-Agent Q-Learning

Multi-agent learning (MAL) tackles a particular category of learning problems where multiple agents need to interact and behave cooperatively or competitively with some degree of autonomy. The problem domain may be decomposed into smaller sub-problems and each agent takes charge of one of them independently while communicating and interacting with other agents. As a result of such *cooperative* and *concurrent* learning, it is feasible to deal with a considerably large search space, as it can be split into smaller sub-spaces. Therefore, if complexities arising from interactions between agents are managed well, cooperative multi-agent learning is promising to explore larger design spaces with less computational complexity leading to a faster learning phase compared to mono-agent learning. However, the main challenge of cooperative concurrent learning is that each learner (agent) needs to adjust its behavior according to the others.

In this paper, we propose the use of Multi-Agent RL for our particular problem. The next two sections develop the formulation of the multi-user video transcoding scenario as a generic Q-Learning problem (Section 4) and the necessary adaptations applied to transform this generic formulation into a multi-agent approach (Section 5).

4 Modelling multi-user video transcoding as a Q-Learning problem

Our framework is deployed considering a multi-core server in which a number of transcoding requests from users arrive at random points in time, each one possibly of a different video type (resolution and contents). The framework responds to this requirement by assigning a variable amount of system resources by tuning application-wide dynamic knobs (QP and number of threads) of each concurrent encoding instance and system-wide knobs (core frequency) to meet joint *restrictions* in terms of performance (THROUGHPUT), quality (PSNR) and power (applying a power cap, P_{cap}). The scenario includes two extra conditions, namely: *(i)* resource usage should be minimized provided THROUGHPUT is met, and *(ii)* quality should be maximized if there are enough available resources.

Mapping this problem statement into an actual Q-Learning process requires a correct definition of states, actions and reward functions. The number, granularity and value distribution of states and actions is ultimately a trade-off between learning time and control degree on the accuracy of output metrics, in which both expert knowledge and application specifics play an important role. Hence, a correct definition of the reward function ultimately determines the success in maximizing/minimizing output metrics, optimality in resource usage and the compliance with the restrictions imposed.

4.1 State definition

Since in this work we aim at QoS-aware real-time transcoding under power budget constraints, the agents constantly sense output metrics (PSNR, THROUGHPUT, and POWER) on a frame-to-frame basis, mapping each particular system observation into a particular state. For 8-bit-depth videos and lossy compression, PSNR should range between 30 dB and 50 dB for acceptable human vision [36]. We divide this range into the following intervals to constitute PSNR states (S_{psnr}): $PSNR \leq 30$, ≤ 35, ≤ 40, ≤ 45, ≤ 50, and > 50 dB. The POWER state (S_{power}) is defined based on the power consumption constraints of the running server: $power < P_{cap}$ and $power \geq P_{cap}$. THROUGHPUT (measured in FPS) is divided into the following states, since the target frame rate is 24 FPS (the one defined in the NTSC standard [37]): FPS $< 24, < 28, < 32, < 35, < 40, < 50$ and ≥ 50. This non-regular formulation of the states is a trade-off between quality of the obtained policy and learning time. On one side, having smaller intervals near the threshold allows the system to distinguish the effects of similar actions and to have a more precise control on the applications. On the other side, having larger intervals far from the threshold reduces learning times, and it has a minimum impact on the quality of the obtained policy.

Additionally, as detailed in Section 6, the use of turbo frequency management into our framework motivates the introduction of buckets of *level of occupation* (number of cores occupied at a given execution point by all simultaneous transcoding processes) as an additional state (S_{occ}), as the actual processor frequency directly depends on this value. In our platform, S_{occ} can take 6 different values.

4.2 Reward Function

To provide suitable feedback to each agent, the selected reward function is a linear combination of three different rewards, one per state, all of them normalized:

a) THROUGHPUT *(minimize output metric under one lower limit):* based on the target frame rate (24 FPS):

$$R(FPS) = \begin{cases} -4 & FPS < 24 \\ a * FPS + b & 24 \leq FPS < 50 \\ 0 & FPS \geq 50 \end{cases} \quad (3)$$

This function provides negative values if the throughput is smaller than the target frame rate. A value of -4 was chosen to penalize those states below the constraint even if the other functions give the maximum reward, ensuring the system will learn not to visit those states in the future. The a and b parameters are adjusted to produce a maximum reward of 1.0 if FPS meets the target of 24, and a decreasing reward down to 0 for larger FPS ($a = -1/26$, $b = 25/13$). The reason is that when $FPS > 24$, 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. Nevertheless, achieving larger FPS

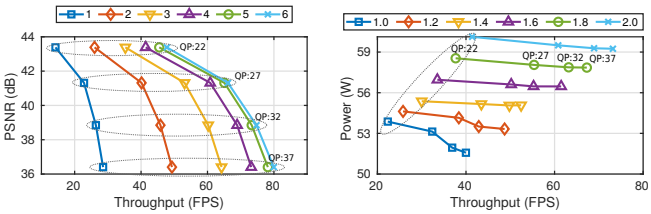


Figure 4: Metrics obtained when setting 2.0 GHz and varying the number of threads (left), and setting 4 threads and varying the frequency (right), while encoding a 1280x720 video.

may result in wasting resources, which ultimately means fewer users can be served. Therefore, this reward function directs the agents to save resources, provided the QoS is met.

b) PSNR *(maximize output metric between two limits):* As explained before, 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:

$$R(PSNR) = \begin{cases} -4 & \text{PSNR} < 30 \text{ or PSNR} > 50 \\ c \times e^{\text{PSNR}/50} + d & \text{otherwise} \end{cases} \quad (4)$$

where c and d are set to give a maximum reward of 1.0 when PSNR=50, and a reward of 0 when PSNR=30, thus maximizing quality ($c \approx 1.12$, $d \approx -2.03$).

c) POWER *(apply tight limit in output metric):* Power consumption is limited to a value defined by the server administrator (P_{cap}). If the constraint is violated, our reward function gives a value of -4 to cancel out positive values obtained from other reward (if any). Otherwise, 0.0 is obtained and no agent is mistakenly promoted for an action with a safe power consumption, but not a desirable PSNR or FPS.

4.3 Actions definition

The definition of actions taken by agents (both in terms of number and distribution within a range), similarly to states, must be chosen based on expertise (problem knowledge) and requirements of learning time and accuracy. Figure 4 provides Pareto curves that relate different output metrics and actions for a transcoding process on the target architecture. In the following, we use it as a baseline to justify the selected actions.

a) QP: QP variations affect FPS, PSNR, power consumption and bitrate [38]. We use QP values of 22, 27, 32 and 37 based on our observation and [13].

b) Number of Threads: While HEVC encoding can benefit from multithreading to increase FPS, Figure 4 shows that throughput saturates above a certain number of threads because there is not enough work for more threads. In our target platform, this limit appears for 5 threads in the case of a HR video, and 3 threads in the case of a LR video.

c) DVFS: Our platform supports frequencies from 1.00 GHz to 2.00 GHz and Turbo mode, selectable on demand and in a core-by-core basis in steps of 100 MHz (with Turbo enabled, this range extends up to 3.7 GHz, but is not under direct control of the user, as it depends on the current core occupation). Therefore, in this work we consider a selection of frequencies within this range as actions for the DVFS agent.

5 Integrating MAL into the Problem

Similar to conventional mono-agent learning, the QL algorithm in multi-agent learning is composed of a finite action set $\boldsymbol{A}$ split in multiple independent subspaces $\boldsymbol{A}_i$, 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 [1] is updated after each reward, indicating the value of applying a_t^i at s_t, is received. This formulation allows the system to deal with larger design spaces as it is split in multiple Q-tables, each one explored concurrently by a different agent. Specifically, 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. 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 and both states and rewards resulting from one agent's action are observable to all agents. In the following subsections, we further discuss on the design of agents, action set, state space, learning phases, and learning rate function.

5.1 Agent design and activation sequence

In MAL, we consider three different agents, each in charge of a different knob. We define agents for tuning QP (AG_{qp}), deciding the number of threads used to encode a frame (AG_{thread}) through Wavefront Parallel Processing (WPP) [10], and per-core DVFS (AG_{dvfs}).

Action granularity is also directly related to the agent activation sequence, and to the relative effects on output metrics of a single step variation in each action. Hence, one step in terms of QP implies large modifications in both quality and throughput (see Figure 4). For the latter, wrong actions taken by the QP agent can be solved or alleviated, with more detail, by subsequent application of actions by the threads or DVFS agents, each one with progressively finer granularity.

We experimentally determine how frequently each agent should act, based on system overhead, the 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 roughly 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 5 shows the proposed sequence for the agents.

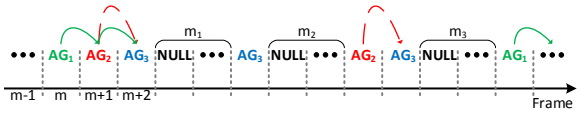


Figure 5: Agent sequence. Different arrows show which agents need to look at the Q-table of the next agent.

5.2 Learning process – dealing with power measurements

Metrics relative to each encoding request are application-wide; on the contrary, power consumption is a system-wide metric, which yields some additional considerations. This fact implies that changes observed in power consumption cannot be directly related to modifications applied to a single application instance, as they can be potentially caused by a combination of multiple changes applied to different concurrent instances. In our formulation, this fact can lead to incorrect learning policies from the agents, updating the Q-tables based on observations that are not consequences of their own actions, but consequences of actions applied by other agents at the same time. This problem only arises during the learning period, while Q-tables are being filled and incorrect observations can lead to low quality policies.

To tackle this problem, we propose a modified learning process, where only one sequence is encoded through our system. With only one video encoded at a time, the agent is able to relate changes in the observed metrics to its own actions. Obviously, as the final goal of the system is to run on a scenario with multiple instances running concurrently, multiple codification processes are executed in the background using a *static knob configuration*, pursuing a dual purpose: *(i)* Agents can observe the dependencies between encoding processes, adding this information to their Q-tables and therefore, obtaining better policies in the future; and *(ii)* The system can visit different state values that an individual encoding process cannot visits by its own (for example, some S_{occ} and S_{power} states can be only visited if almost all the cores are used simultaneously by multiple application instances).

5.3 General system overview

Figure 6 depicts a general diagram of our formulation, and shows how actions are chosen and applied. The system is continuously sensing the environment and receiving different (application- and system-wide) metric values frame-to-frame basis, storing and processing them. Upon each agent activation, the state is built from the current and stored metrics, discretizing the continuous values according to the states defined in the previous section (Step 1 in the diagram). This state is used to update the Q-table of the previous agent (Step 2, Equation 1), and to determine the action of the next agent (Step 3). As the learning rate is defined for each *state/action* pair, different pairs can belong to different learning phases –*exploration*, *exploration-exploitation* or *exploitation*– for the same state. In Step 4, the system determines the phase of the next action and then which action belonging to that phase applies (Step 5). How the phase is chosen requires an additional explanation, shown next. Finally, the system applies the chosen action in step 6.

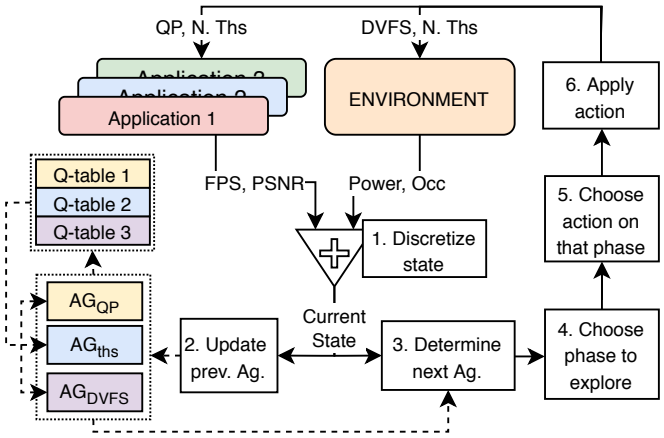


Figure 6: General system overview.

The fact that different learning rates exist for each *state/action* pair allows a more precise control of the learning progress of each state. However, it remains unclear which action an agent should choose among those available, as this decision ultimately depends on the learning phase. To solve this problem, we follow a two-step approach: *(i)* In Step 4, the agent selects the phase of the next action to choose, but not yet a specific action, as the action selection strategy depends on the phase just selected; and *(ii)* in Step 5, the agent chooses, between the actions belonging to the previously selected phase the next action to take. If the selected phase is *exploration*, the next action will be chosen randomly between those pairs belonging to the exploration phase. In the other phases, the action will be chosen in a co-operative way, as described in detail in Section 5.4.

Algorithm 1: Learning phase selection (Step 4 in Figure 6)

```
Data: number of actions at each phase (nExploration, nExplor_Exploi, ...)
      and number of available actions (|A_i|).
Result: chosen ← phase of the next action to choose.
1  begin
       /* =A=: All the pairs are in the same phase:        */
2      if nExploration = |A_i| then chosen ← exploration;
3      else if nExplor_Exploi = |A_i| then chosen ← explor_exploi;
4      else if nExploitation = |A_i| then chosen ← exploitation;
       /* =B=: Pairs mixed with different phases           */
5      else if nExploration > 0 then
6          if rand() ≥ 0.5 then                          // P=0.5
7              chosen ← exploration;
8          else                                          // P=0.25
9              chosen ← randomSel(explor_exploi, exploitation);
10     else
11         chosen ← randomSel(explor_exploi, exploitation);   // P=0.5
12 Function randomSel (phase1, phase2) is:
13     Selects randomly between both phases.
14     Checks if there is at least one action in each phase before choosing it.
```

Algorithm 1 shows the pseudo-code followed by an agent to choose the phase of the next action to apply (Step 4). If all actions belong to the same phase, the decision of which phase to select next is trivial (lines 1–5). However in the case where, in a specific state, there are actions belonging to different phases, the agent will try to progress the learning process by taking those actions that are still in the *exploration* phase first. To do that, the agent randomly chooses the next

phase giving the double of probability to the *exploration* phase than the other two phases. To do that, in the case there is at least one pair in exploration phase, the agent will give a probability of $1/2$ to select that phase, and a probability of $1/4$ to the other phases (lines 5 to 9). The decision between *exploration-exploitation* and *exploitation* phases is taken with the same probabilities.

5.4 Improvements over the Mono-Agent approach

1) Learning Rate Function: Each agent must have its own learning rate for each state-action pair because of the different number of actions and activation frequency. The proposed learning rate function is a decreasing function of the number of state-action observations, defined differently from those proposed in the literature [31], [34], [39]: if these functions were 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. 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, as follows:

$$\alpha^{(i)}(s_t, a_t^i) = \frac{\beta_i}{Num(s_t, a_t^i)} + \frac{\beta_i^{'}}{1 + \sum_{j \neq i} \left(\min_{a \in A_j} (Num(a)) \right)} \quad (5)$$

Here, the first term is taken from literature [39], 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: *(a)* (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, *(b)* other agents have tried all their actions (at least once).

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 $\gamma = 0.6$.

2) Dealing with a stochastic environment: 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 *(i)* contents of a video can change from one frame to another, *(ii)* other agents taking charge of a single video may apply an action that alters the next expected state to a different one, and *(iii)* other videos existing in a multi-user platform with their corresponding contents and agents can change the state unexpectedly.

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 the number of times that applying a_t^i at s_t resulted in s_{t+1}, and $Num(s_t, a_t^i)$ represents the 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, and used to cooperatively choose the best action, as seen next.

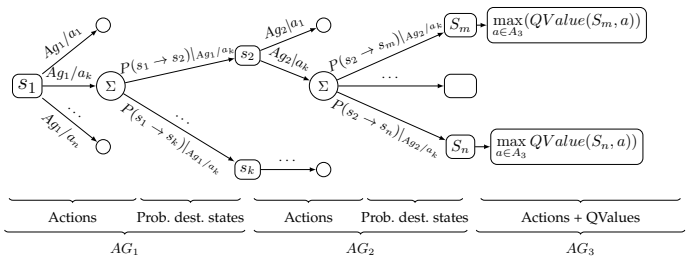


Figure 7: Different paths to explore before taking an action.

3) Cooperation process: Similarly to the mono-agent approach, the learning process of the different agents follows a three-phase approach; however, while in *exploration* phase the agents still explore the different actions randomly and independently, in the following phases they start to cooperate.

Algorithm 2: 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

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

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

3

4 **if** *(AG.next() == NULL)* **then**

5 **return** $\max_{a \in A_{AG}^*} \left(Q^{AG}(s, a) \right)$

6 **else**

7 **return** $\left(\max_{a \in A_{AG}^*} \left(\sum_P P^{AG}(s \xrightarrow{a} s') \times \boldsymbol{E}[Q_{Value}(AG.next(), s')] \right) \right)$

Although each agent explores the design space separately and has its own Q-table, it needs to act in the exploitation phase cooperatively. Consequently, the goal of each agent is not just to maximize the Q-value attainable from its own Q-table, but rather, maximizing the expected Q-value after a sequence of actions taken by all agents. Consider, for example, the sequence of agents shown as in Figure 5. 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 consider 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 2 and Figure 7. Even though the number of different paths to explore can grow exponentially, most of them can be pruned.

4) Dealing with Sensing Noise: When dealing with a non-deterministic problem, it is common that all the states

are not visited uniformly. This fact can be caused by wrong/noisy measurements from the system (e.g., power consumption measurements can be affected by other processes, or precision problems can arise when measuring the frame processing time). This problem cannot always be fixed, leading to the need to adapt the learning process to deal with this issue. In the mono-agent approach described before, this behaviour has no impact on the learning process as each pair state/action can evolve independently of the other pairs, however, in the multi-agent approach, the learning rate definition depends not only on the number of times one pair state/action has been visited, but also on the minimum number of times other agent has visited each action (second term of Equation 5). This definition affects not only the learned policy, but also the evolution of learning phases, in the worst case, making the system stall in the exploration phase, not allowing to progress to the next phases. To tackle this problem, we propose the use of a slightly modified version of the classical Z-score algorithm [40] to detect outliers. Our proposal, shown in Equation 6, identifies state/action pairs that have not been visited enough times compared to the other pairs. When the learning rate function has to be calculated, the states detected as *bottom-outliers* are discarded. However, these states are not removed from the Q-tables since they can be potentially visited in the future enough number of times to have an impact on the learning function. Indeed, we have experimentally determined that discarding states with z-score$' \leq -3.5$ filters the desired states.

$$\text{z-score}'^{(i)}(s_t) = \frac{Num(s_t, a_t^i) - \mu^{(i)}}{\sigma'^{(i)}} \tag{6}$$

where:

$$\mu^{(i)} = \frac{\sum_{s_t} Num(s_t, a_t^{(i)})}{Num^{(i)}(s_t)}, \quad \mathbb{S}^i = \left\{ s_t \mid Num(s_t, a_t^{(i)}) \leq \mu^{(i)} \right\}$$

$$\mu'^{(i)} = \frac{\sum_{\hat{s_t} \in \mathbb{S}} Num(\hat{s_t}, a_t^{(i)})}{|\mathbb{S}^{(i)}|}, \quad \sigma' = \sqrt{\frac{\sum\limits_{\hat{s_t} \in \mathbb{S}} \left(Num(\hat{s_t}, a_t) - \mu'^{(i)} \right)^2}{|\mathbb{S}^{(i)}|}}$$

6 Experimental Setup

The described framework and techniques have been implemented in a real server using a centralized runtime resource manager written from scratch, that integrates the multi-agent Q-Learning logic and allocates the registered applications (in our case, multiple instrumented Kvazaar instances using the predefined *ultrafast* configuration). The resource manager is a client/server infrastructure coded in C++, in charge of four main actions, namely:

(1) Metric Reception Module: receives application-specific metrics on a configurable regular basis (in our case bitrate, PSNR and throughput) from registered applications. System-V message queues are employed to communicate the applications with the centralized server, due to their low latency and the ability to define custom message formats. On the client side, communications are encapsulated into an external C library. Then, the integration of the communication patterns between client and server is straightforward and non-intrusive. As an example, for Kvazaar, this instrumentation only added 37 new lines to the original source code.

(2) Sensing Module: performs a periodic and configurable *sensing* of the underlying system-wide metrics. In our formulation, the module is configured to sense power consumption via the `RAPL` component in `PAPI` [41], although other alternative mechanisms can be easily integrated.

(3) Decision Module: selects the most appropriate knob combination to be applied to each registered application based on both application metrics and platform status gathered by the previous modules, applying predefined techniques (e.g., multi-agent Q-Learning or mono-agent Q-Learning, in our specific case, or others, including ad-hoc heuristics). Concurrent multi-application support is implemented via ISO C++ threads. In the case of Q-Learning, the discretization of metrics and sensing parameters, according to the state definitions detailed in Section 4.1, is also a task performed by the module.

(4) Resource Management Module: it is in charge of the underlying shared resource management, performing an efficient distribution across concurrent clients. In our formulation, it sets application-to-core affinity and according to the Decision Module outputs, manages per-core frequency values. Application affinity is managed by means of `POSIX Thread Affinity API` calls; frequency variations are carried out via the `libcpufreq` library.

6.1 Experimental testbed

Our experimental testbed consists of a 20-core server with an Intel Xeon Gold 6138 CPU and 128 GB of DDR4 RAM. *Hyperthreading* was disabled as Kvazaar does not benefit from it. Thanks to the use of System-V message queues between applications and server, the measured overhead introduced by the centralized resource manager has been found to be negligible (i.e., less than 0.05% of the total encoding time). Per-core DVFS ranges from 1.00 GHz to 2.00 GHz, selected by the resource manager. TDP (Thermal Design Power) is 125W, and the maximum transition latency for DVFS is 10.0 μs, which is small enough for real-time DVFS application. Hardware power capping, when necessary, has been carried out through Intel RAPL.

Turbo frequency management in the Intel Xeon Gold family requires a specific explanation. In these processors, when turbo mode is enabled and selected by the DVFS agent, the maximum frequency for each core depends on *(a)* the specific vector instructions issued by each core (Normal, AVX-2 –active in our setup– and AVX-512), and *(b)* the number of active cores at each moment, as shown in Table 2 and [18].

6.2 Dataset definition

We consider videos with two different resolutions: Low Resolution (LR) videos, which is the default resolution provided by Youtube (832×480 pixels), and High Resolution (HR) videos which is the resolution considered as High Definition (720p/HD videos, 1280×720 pixels). HR videos have been extracted from those proposed by the JCT-VC [13]; LR videos are re-scaled versions of their HR counterparts. As shown in Table 3, the chosen videos cover different scenarios. On one hand, videos like HR6 or HR1 produce high variability on the obtained FPS, but with low resource requirements (e.g., $30.6(\pm 6.5)$ FPS with fixed resources for

Table 2: AVX2 turbo frequencies for the target architecture depending on the number of active cores.

Active cores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Frequency (GHz)	3.6	3.6	3.4	3.4	3.2	3.2	3.2	3.2	2.7	2.7	2.7	2.7	2.5	2.5	2.5	2.5	2.3	2.3	2.3	2.3
MAL state (S_{occ})	1		2		3				4				5				6			

Table 3: Video characterization using static encoding resources: 3 threads, 1.5GHz, QP=22.

Id	Name	μ_{FPS}	σ_{FPS}	Id	Name	μ_{FPS}	σ_{FPS}
HR1	*FourPeople*	30.6	6.5	HR4	*QuarterBackSneak1*	26.1	2.9
HR2	*KristenAndSara*	30.2	4.1	HR5	*BT709Parakeets*	29.1	5.1
HR3	*OldTownCross*	15.4	1.5	HR6	*Johnny*	31.7	5.7
				HR7	*ThreePeople*	29.7	6.3

HR1). On the other hand, HR3 or HR4 are sequences with low variability but higher resource demands due to their contents ($26.1(\pm 2.9)$ FPS when encoded with the same knob values in HR4). In our experiments, the first three videos were used for training, while the rest were used for testing the obtained results.

7 Results

7.1 MAL evaluation: adaptability and resource usage

For comparison purposes, we show next the behavior of our approach compared to a STATIC knob selection strategy, where knob values are fixed to a constant value during the whole execution. For the sake of fairness, knob values in STATIC correspond to the average values learned by the MAL system for the training videos (i.e., 3 threads, 1.5 GHz and $QP = 32$ for HR videos, and 3 threads, 1.4 GHz and $QP = 22$ for LR videos). We report results for five executions of each video when running isolated, and average values obtained for random sets of more than one video being encoded simultaneously. For both MAL and STATIC, Table 4 reports QoS metrics in terms of *(a)* QoS violations (percentage of time the video is encoded below the predefined real-time threshold, represented as -Δ- in the table), and *(b)* attained quality (measured in PSNR) for the different videos in high and low resolution. The tables also show the average knob value (QP, number of threads and frequency) for the MAL approach.

7.1.1 Single video behavior

Focusing on observations for isolated videos, we can distinguish two types of sequences, with different learned policies depending on their characteristics: a first group (comprising sequences HR6 and HR7) that is encoded with a low number of resources; and a second group which needs more resources to be encoded (HR4 and HR5). For the latter, the system learns to increase the frequency and average thread number (more evident for HR4).

In Figure 8 we report the throughput of the STATIC and MAL approaches when encoding the high-demanding HR4 sequence. The first two plots report the instantaneous FPS obtained by the STATIC and MAL approaches respectively, and show how the MAL approach is able to restrict the attained throughput to the real time threshold. On the contrary, the percentage of time the STATIC strategy is not able to fulfill this restriction is larger even though the selected

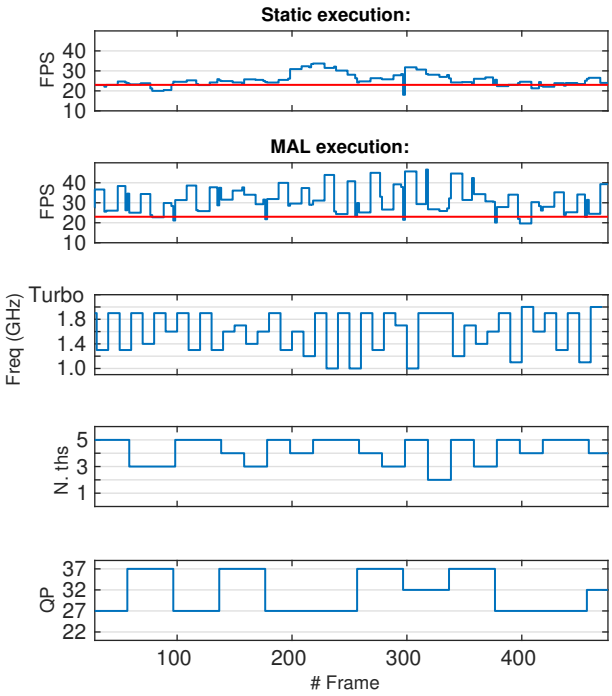


Figure 8: STATIC vs MAL when encoding a HR4 sequence.

(and fixed a priori) values for each knob are those learned by our system. This is a clear sign of the benefits of the dynamic knob tuning.

The last three plots in Figure 8 show the actual modifications carried out in knobs to adapt to the video content, and gives an overview of the general strategy learned by the agents: varying system frequency more often than number of threads, and modifying number of threads more often than QP. This strategy does actually have an explanation. As introduced in Section 5, one-step frequency modifications has a relatively small impact on the encoding process, exposing a finer-grained control over throughput. If the system needs to perform larger changes in the encoding process (e.g., due to large changes in video contents), it will first modify the number of threads (*N. ths* in the figure) or QP. Thus, it will need to adjust frequency immediately after to tune the execution in a finer-grained way. This high-level behavior exhibiting the impact of each knob (in the order of frequency, number of threads, and QP) has been extracted automatically after the learning process, and is one of the main benefits of the RL-based approach.

To sum up, MAL reduces the number of QoS violations when compared to the STATIC approach, yielding $1.6\times$ and $2\times$ better results for the HR and LR sequences. respectively.

7.1.2 Turbo behavior and occupation level

Next, we show that the occupation of the cores has a strong influence on the turbo behavior. Consider the traces in Figure 9. The first two plots represent an encoding process

Table 4: Output metrics and resource usage for the MAL approach compared with the STATIC assignment.

HR	*Output metrics*				*Avg. Knob values*		
	−Δ−		**PSNR** (dB)		**Freq**	**N. Ths**	**QP**
	MAL	Static	MAL	Static	MAL		
1×HR4	7.8	14.6	41.1	40.8	1.6	4.0	31.1
1×HR5	0.9	0.4	40.2	39.9	1.4	3.8	31.4
1×HR6	1.0	0.4	40.7	40.1	1.2	3.6	31.3
1×HR7	0.5	0.4	39.3	39.2	1.3	3.5	31.9
1×HR *Avg.*	2.5	4.0	40.3	40.0	1.4	3.7	31.4
2×HR *Avg.*	3.1	5.4	40.3	40.1	1.3	3.3	31.8
3×HR *Avg.*	4.5	8.5	39.4	40.1	1.3	3.3	33.7
4×HR *Avg.*	7.1	9.9	39.0	40.1	1.4	3.1	34.8

LR	*Output metrics*				*Avg. Knob values*		
	−Δ−		**PSNR** (dB)		**Freq**	**N. Ths**	**QP**
	MAL	Static	MAL	Static	MAL		
1×LR4	27.9	71.7	44.0	44.7	1.7	2.5	23.7
1×LR5	7.4	0.5	44.4	44.5	1.5	2.8	22.4
1×LR6	0.9	0.4	44.8	44.8	1.3	3.0	22.0
1×LR7	0.5	0.5	44.0	44.0	1.4	3.0	22.0
1×LR *Avg.*	9.2	18.3	44.5	44.6	1.5	2.8	22.5
2×LR Avg.	11.1	11.1	44.0	44.6	1.5	2.6	23.6
3×LR Avg.	12.7	16.1	43.7	44.6	1.5	2.6	24.1
4×LR Avg.	13.1	19.9	43.9	44.6	1.6	2.5	23.7
5×LR Avg.	15.0	22.7	43.3	44.6	1.5	2.6	25.2
6×LR Avg.	13.8	35.0	41.7	44.6	1.5	2.6	28.6

of a single high-demanding LR5 video (as described before), while the last four plots show the behavior when 5 LR5 videos are encoded simultaneously (only the traces of one of them shown). When only a single video is encoded, the MAL system sets the number of threads to 3 and QP value to 22 (same values used in the STATIC approach), adapting itself to the video content changing only the frequency between 1.3 GHz and turbo frequency (3.4GHz due to the low occupation). However, as the occupation of the machine increases, the turbo frequency decreases (down to 2.5 GHz when there are 15 cores occupied) and, thus, not sufficient to adapt to content variation. Consequently, the system learns how to adapt to the content by properly modifying the other knobs.

7.1.3 Concurrent sequences behavior

a) Scenario 1. Homogeneous-resolution videos: When there is more than one video being encoded simultaneously, the results clearly show how our approach is consistently able to encode the different workloads with a low number of QoS violations, adapting to different occupation levels and video resolutions (mainly increasing QP, that is, decreasing quality to compensate the lower turbo frequency). While the STATIC approach works relatively well for a low number of videos, its QoS decreases when the computational demands increase, as shown in the second half of Table 4.

As shown in the previous section, when there is only one video running, the STATIC approach behaves relatively well for most of the sequences. However, when the number of concurrent videos increases, so does the number of QoS violations. The flexibility of the MAL approach allows a dynamic adaptation of knobs during the execution to satisfy the different demands of the different videos on different server loads, obtaining better results than the STATIC approach. In the case of HR sequences, the degradation of the QoS ranges from 3.1% to 7.1% when the MAL approach encodes 2 to 4 simultaneous videos respectively, and from 5.4% to 9.9% when the videos are encoded using the STATIC approach.When LR sequences are encoded, it ranges from 11.1% when 2 videos are encoded simultaneously (for both approaches), up to 13.8% in the MAL approach and 35.0% in the STATIC approach when 6 videos are considered. Although the MAL approach decreases the PSNR when the number of videos increases, the loss in quality is only

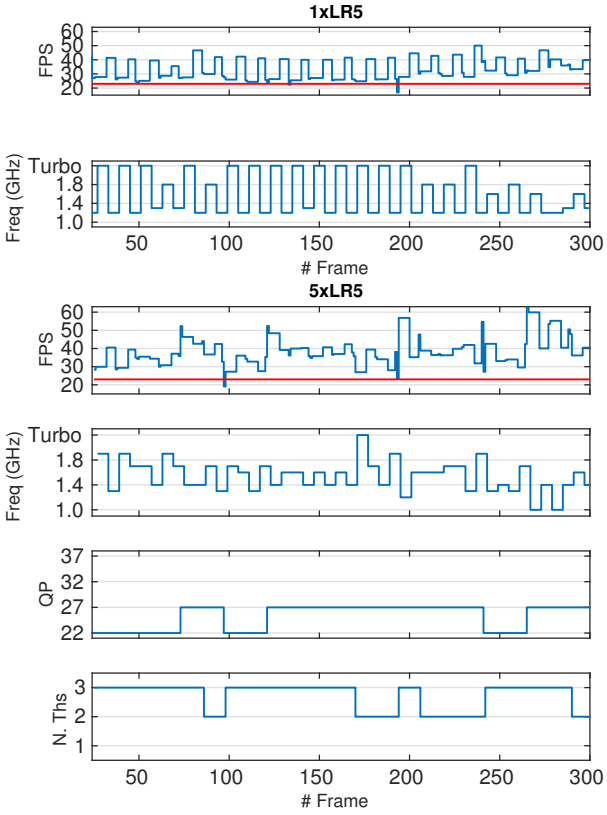


Figure 9: Encoding timeline for a single LR5 sequence is encoded (top) vs. 5 LR5 simultaneous sequences (bottom). For a single sequence, the system sets the number of threads to 3 and the QP value to 22 (not shown in the plots).

of 2.9dB in the worst case, thanks to the QP adaptation previously described.

To recap, the percentage of time in which the QoS restrictions are violated increases with the number of simultaneous videos. However, the QoS degradation is greater for STATIC than for MAL. Hence, our proposed solution is able to adapt to different levels of occupation, which results in more desirable outcomes (1.7× and 2.5× for HR and LR videos respectively).

b) Scenario 2. Behavior under video combinations: Table 5 shows the results obtained for a more realistic scenario, where different number of videos of different resolutions

Table 5: Output metrics and number of threads (MAL vs STATIC) for different combinations of mixed videos.

	PSNR (dB)		–Δ–		**N. Threads**	
	MAL	STATIC	MAL	STATIC	MAL	STATIC
1HR+1LR	41.5	42.0	1.2	0.3	5.7	6
1HR+3LR	42.8	43.4	7.6	9.9	10.8	12
1HR+5LR	42.0	43.8	11.6	10.7	14.8	18
2HR+1LR	41.0	41.5	2.8	2.3	8.6	9
2HR+3LR	39.6	42.6	6.3	3.8	13.3	15
2HR+5LR	39.6	43.3	10.0	20.7	18.6	21
3HR+1LR	39.4	40.9	2.3	3.7	10.8	12
3HR+3LR	39.8	42.3	10.0	14.4	16.8	18
3HR+5LR	38.6	42.7	11.3	26.6	20.0	24
4HR+1LR	39.1	40.8	3.6	3.9	14.0	15
4HR+3LR	38.1	41.8	7.1	16.4	19.0	21
4HR+5LR	38.1	42.6	9.0	35.5	20.0	27

are simultaneously encoded. Qualitatively, the behavior is similar to the previous experiments: as the occupation of the server increases, the QoS violations increase too. In addition, in the extreme cases where the STATIC approach assigns more threads than available cores (20 in our platform) arising oversubscription (i.e., two active threads are executed on the same physical core), our approach is able to adapt the quality and number of threads between the videos in order not to exceed the available physical cores and to obtain maximum QoS. MAL obtains 2× improvements in QoS when compared against STATIC assigning 21 threads (experiment 2HR +5LR), and up to 4× when 27 threads are used by STATIC (experiment 4HR +5LR). The loss in quality is minimum, i.e., encoding always sequences with PSNR above 38 dB.

7.2 Comparison with state-of-the-art heuristics

To compare our proposal with an existing state-of-the-art approach, we have implemented the ARGO heuristic described in [25], [26]. ARGO bases its decisions on an internal database storing all feasible knob configurations (called *Operating Points* -OP-), and an estimation of the output metrics obtained for each OP. ARGO maintains a set of constraints that the system should never violate. At runtime it chooses, between all the promising OPS that do not violate the constraints, the configuration that maximizes a user-defined rank function. To provide self-adaptation, ARGO computes on the fly different coefficients to relate the obtained measurements to those expected and stored.

To compare our proposal with ARGO, we generate its internal database by profiling the same videos used in our system; we set QP and number of threads as the dynamic knobs managed by the heuristic, delegating the frequency adjustment to the *ondemand* governor of the OS. The heuristic acts every 6 frames (matching the minimum frequency in our approach), and the rank function used to evaluate OPs matches the reward function used in the MAL approach.

Table 6 shows the results obtained when executing the same combination of videos used in the previous section. For the sake of clarity, we report the MAL results again in this table. Although ARGO obtains higher PSNR in all the tested scenarios, the amount of time the system violates the throughput constraint is systematically larger than in our proposal. This behavior is explained in terms of the changes in the content of the video. While our proposal is able to perform a fine-tuning process by adjusting the processors' frequency, ARGO delegates this to the governor of the operating system. This delegation results on setting the frequency at turbo in all scenarios, thus, it needs to increment the number of threads to compensate a violation in the QoS constraint. However, although a change in the number of threads can have a huge impact on the instantaneous throughput, the increment in FPS is slower than the one obtained when changing the frequency. The later is the policy followed by the MAL approach.

Table 6: MAL compared to the ARGO approach.

HR	*Output metrics*				*Knobs*	
	–Δ–		**PSNR** (dB)		**N. Ths**	**QP**
	MAL	ARGO	MAL	ARGO	ARGO	
1×HR4	7.8	9.6	41.1	43.6	2.7	25.0
1×HR5	0.9	8.5	40.2	42.5	1.5	26.7
1×HR6	1.0	9.9	40.7	42.9	1.1	25.2
1×HR7	0.5	7.0	39.3	41.7	1.0	26.4
1×HR *Avg.*	2.5	8.8	40.3	42.7	1.6	25.8
2×HR *Avg.*	3.1	9.4	40.3	42.7	1.6	25.8
3×HR *Avg.*	4.5	6.3	39.4	42.7	1.7	25.8
4×HR *Avg.*	7.1	7.1	39.0	42.7	1.9	25.9

Table 6 also shows that the number of threads used by ARGO increases with the number of videos. As shown in Table 2, the turbo frequency decreases as the number of active cores increases. Hence, an increase in the amount of threads is needed to compensate the loss in frequency.

7.3 Multi-agent vs. mono-agent

To show the advantages and disadvantages of our multi-agent proposal against other mono-agent-based Q-Learning approaches, we have implemented the general mono-agent approach described in Section 3.1. For the sake of fairness, the mono-agent takes an action every 6 frames (the minimum frequency in MAL), and considers all possible combinations of actions considered by MAL. All results hereafter correspond to a training process with 4 HR simultaneous transcoding processes, using the sequences for training and test described before. Only results with 4 simultaneous videos are reported due to the unfeasible time required to train the mono-agent system for different number of videos, as described next. The reported results correspond to the average values obtained when encoding 4 test sequences selected randomly at the same time (each executed 5 times), using the same combination of sequences for both approaches. Data measured confirm the weak points previously mentioned for the mono-agent approach and alleviated by the MAL implementation, namely: the learning time for the mono-agent is dramatically larger than that of the MAL approach (6 times longer in our experiments), and the mono-agent approach has less control on the consequences of each action as the decision space considers all the knobs together, hiding the relations between them.

a) Learning time: The two plots on the top of Figure 10 report the learning time for the mono-agent and multi-agent approach, respectively, under equivalent experimental scenarios. Each line represents the amount of state-action pairs

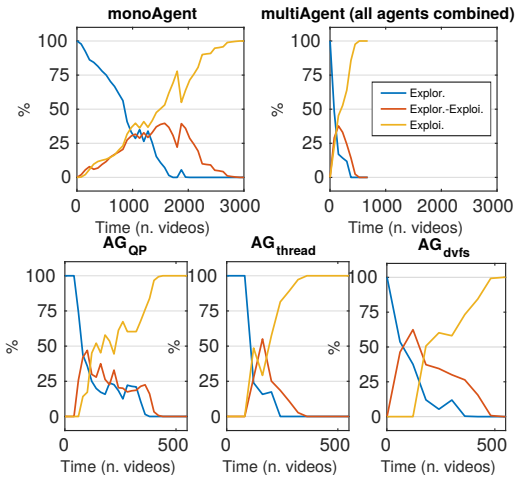
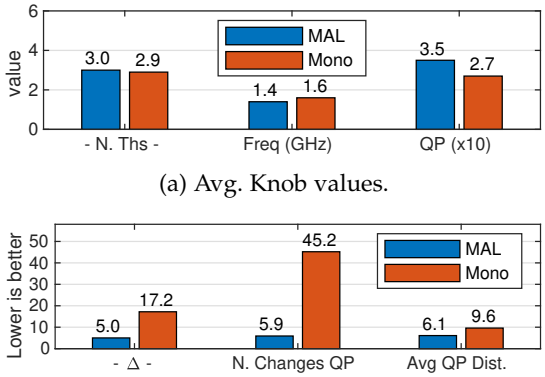


Figure 10: Learning evolution of the mono-agent approach vs multi-agent approach. Each line represents the percentage of state-action pairs that are in each phase. Top, the mono-agent approach vs the multi-agent approach (all agents combined). Bottom, a detailed view of the behavior of each agent.



(a) Avg. Knob values.

(b) QoE metrics.

Figure 11: Top: resource usage by the MAL and mono-agent approach. Bottom: QoS and QoE metrics obtained. The data represents average values for different combinations.

(in terms of percentage) at each phase of the learning process. In the case of the multi-agent approach, lines represent combined data for the three agents. Both x-axes are equally scaled for comparison purposes, and represent the status of the system after training with a certain number of sequences ($\approx 500 frames/sequence$). The results clearly show how all state-action pairs start in the *exploration* phase, and how they move to *exploration-exploitation* and then to *exploitation* phase while the pairs are being visited over time. As described in Section 5, running the system on a real platform produces noisy measurements, which forces the adoption of filtering techniques to remove these noisy data. Our filtering algorithm does not remove these measurements, but ignores them until it is sure that these measurements are correct (Equation 6). This leads to the lines in the plot not to be monotonic, but still convergent. As the decision space for the mono-agent considers all combinations for all actions (4 QP values $\times$ 5 different numbers of threads $\times$ 12 frequencies = 240 different actions), the convergence for the mono-agent is slower than for the multi-agent, which splits the decision space and explores them concurrently, yielding $6\times$ faster learning times.

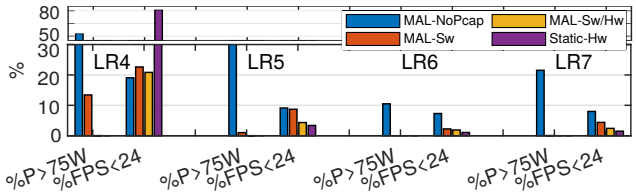
The plots at the bottom illustrate the learning process of each agent in MAL. Even though all of them show the same behavior, the convergence slightly varies among them, due to the different number of actions each agent needs to explore, as well as by the different frequency at which each agent acts.

b) Learned policies: Figure 11a shows how the number of threads and selected frequency are similar to those learned by MAL; QP values are considerably smaller (26.9 vs 34.6), yielding higher-quality videos while the computing resources increase. This implies an increase in the QoS violations shown in Figure 11, obtaining worse results than for MAL. The reason of this behavior is that, when considering simultaneous knob changes, the relation between them is hidden, obtaining at the end a more coarse-grained control than the MAL approach. Also, MAL does not only rely on the Q-values learned on the decision process, but also on the stored probabilities between states when applying each action. Thus, it has more information than the mono-agent to make right decisions.

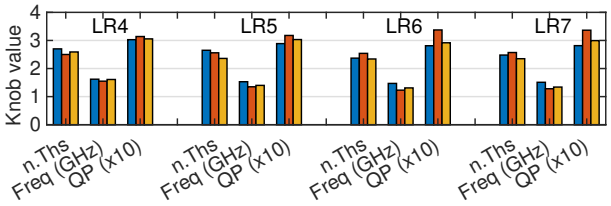
c) Actions variability (QoE): In addition to QoS, it is also crucial to consider QoE [8]. This is directly related to the perception of the user when visualizing the encoded sequence. For example, even if the frame rate is always above 24 FPS, continuous or abrupt changes in quality can damage QoE. To quantify these two metrics, Figure 11 reports the number of QP changes during the encoding process (i.e. quality changes), and average distance between the selected QP values (if the distance is large, the change in quality is significant). Besides the coarser-grained control of the mono-agent when knobs are considered jointly, it is impossible to set a different frequency (for agent activation) to modify each knob. This produces a higher number of changes in QP than MAL (in our case, every 6 frames vs every 24), and hence constant changes in quality. In addition, as the mono-agent learned to use a lower QP value with the same number of threads and frequency as MAL, when the system needs to adapt the QP value, it performs abrupt changes, damaging QoE. For the example shown in Figure 11, the benefits in terms of QoS and QoE can be summarized as a reduction of 12% in FPS violations, $7\times$ less QP changes, and 30% reduction on average QP distance.

7.4 Power capping

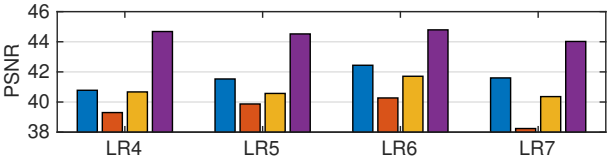
Finally, we investigate on the ability of MAL to apply tight power capping limits while adapting resources and keeping thresholds on THROUGHPUT and PSNR. The experiments are based on the execution of the maximum number of videos before oversubscription arises (in our case, 6 LR simultaneous videos), combined with a power capping limit (75W, which is 60% of TDP). We study three power capping mechanisms: a) software capping (MAL-SW), in which MAL autonomously learns the optimal knob combinations to achieve power capping following the ideas presented in Section 5. b) Hybrid software-hardware capping (MAL-SWHW), in which no power states are considered at learning



(a) QoS metrics.



(b) Resource usage.



(c) PSNR.

Figure 12: (a) QoS, (b) resource usage, and (c) quality for the same 6 videos simultaneously encoded under a power cap of 75 W, using the different policies described in the text.

time and there is no power reward, but instead hardware mechanisms are applied to maintain power under the cap. c) A STATIC-Hw implementation with hardware capping where the values for the different knobs are statically selected a priori and hardware capping is used. The values chosen for the STATIC-Hw approach correspond to the average values learned by MAL. We report results in terms of used resources (Figure 12b) and output metrics (Figures 12a and 12c) for the three power capping mechanisms and for MAL-NOPCAP, which means MAL with TDP as power cap.

First, consider the capabilities of the three methods to maintain power consumption below the cap. For reference, if none of the three previous mechanisms is used, the amount of time in which the cap is exceeded in a normal execution of the MAL system ranges from 52% (LR4) to 10% (LR6). By using any of the three approaches, power capping violations (labelled as $\%P > 75W$) reduce this range in all cases. When hardware capping is applied, obviously, the percentage of capping violations is reduced to a value close to 0%. In the case of pure software capping, it ranges from 13% (LR4) to 0% (LR6), which demonstrates the ability of MAL to dynamically adjust knobs to meet the power capping restrictions.

Regarding QoS, both software-hardware and software power capping are able to achieve similar FPS violations (columns labelled as $\%FPS < 24$) in all cases compared with situations in which no power capping is applied. Recall that, in those cases, power capping is simultaneously met. This achievement is possible due to a correct adaptation (reduction) of quality (PSNR). At a glance, MAL manages to slightly reduce average core frequency, from a range between 1.62 GHz (LR4) to 1.47 (LR6) to a range between 1.55 GHz (LR4) to 1.23 (LR6). This reduction is accompanied by an increase in QP, and hence a reduction in quality (PSNR), see Figure 12c. Note that, however, quality is maintained under acceptable limits. In summary, MAL achieves software power capping, for the tested video combinations, by means of reducing the frequency and trading off quality. Besides, if hardware capping mechanisms are available, MAL can cooperate with them.

8 CONCLUSIONS

In this paper, we have proposed detailed guidelines and evidences that demonstrate the feasibility of integrating Reinforcement Learning techniques into an ad-hoc resource management system serving multiple (and heterogeneous) video transcoding requests on a modern multi-core server.

The results for a highly optimized HEVC implementation have demonstrated that our multi-agent approach adapts to changes in video contents and server occupation, achieving an improvement of $2\times/4\times$ when the occupation of the server is low/high, respectively. We have given evidences that reveal the appealing of a multi-agent approach in terms of learning time ($6\times$ reduction compared with a mono-agent approach) and quality of learned policies ($3.4\times$ improvements on QoS, and $7\times$ in QoE). We have shown how power capping capabilities can be incorporated to the resource manager obtaining competitive results when compared against hardware power capping mechanisms. The management of dynamic application- and system-level knobs in a holistic fashion is general enough to be extended with further parameters or output metrics, and to other applications, both in the multimedia area and in other fields; also, the architectural-related techniques applied to deal with system knobs are of wide appeal to be applied (isolated or in conjunction) to other present and future architectures.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[2] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source HEVC/h. 265 encoder," *ACM on Multimedia Conference*, pp. 1179–1182, 2016.

[3] L. Pham Van, J. De Praeter, G. Van Wallendael, S. Van Leuven, J. De Cock, and R. Van de Walle, "Efficient bit rate transcoding for high efficiency video coding," *IEEE Trans. on Multimedia*, vol. 18, no. 3, pp. 364–378, 2016.

[4] D. Silveira, M. Porto, and S. Bampi, "Performance and energy consumption analysis of the x265 video encoder," *25th European Signal Processing Conference*, pp. 1519–1523, 2017.

[5] Z. Wang, L. Sun, C. Wu, W. Zhu, Q. Zhuang, and S. Yang, "A joint online transcoding and delivery approach for dynamic adaptive streaming," *IEEE Trans. on Multimedia*, vol. 17, no. 6, pp. 867–879, June 2015.

[6] X. Li, M. A. Salehi, M. Bayoumi, N. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video transcoding using heterogeneous cloud services," *IEEE TPDS*, vol. 29(3), pp. 556–571, 2018.

[7] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 545–559, 2016.

[8] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck, "QoE-driven rate adaptation heuristic for fair sutton1998reinforcementadaptive video streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 2, pp. 28:1–28:24, 2015.

[9] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," *SIGPLAN*, vol. 46, no. 3, pp. 199–212, 2011.

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

[11] Y.-W. Huang, C.-Y. Chen, C.-M. Fu, C.-W. Hsu, Y.-L. Chang, T.-D. Chuang, and S.-M. Lei, "Method and apparatus of delta quantization parameter processing for high efficiency video coding," May 10 2012, uS Patent App. 13/018,431.

[12] T. Biatek, M. Raulet, J.-F. Travers, and O. Deforges, "Efficient quantization parameter estimation in HEVC based on ρ-domain," *Signal Procc. Conf.*, pp. 296–300, 2014.

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

[14] J. He, E. Yang, F. Yang, and K. Yang, "Adaptive quantization parameter selection for H.265/HEVC by employing inter-frame dependency," *IEEE Trans. on Circ. and Systems for Video Technology*, vol. 28, pp. 3424–3436, 2018.

[15] T. Zhao, Z. Wang, and C. W. Chen, "Adaptive quantization parameter cascading in HEVC hierarchical coding," *IEEE Trans. on Image Processing*, vol. 25, no. 7, pp. 2997–3009, 2016.

[16] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Evaluation of DVFS techniques on modern hpc processors and accelerators for energy-aware applications," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.

[17] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, pp. 20–27, 2012.

[18] I. Corporation. (2019, March) Intel xeon processor scalable family. specification update.

[19] B. Donyanavard, T. Mück, A. M. Rahmani, N. Dutt, A. Sadighi, F. Maurer, and A. Herkersdorf, "SOSA: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," *MICRO '52*, pp. 685–698, 2019.

[20] Z. Liu, H. Zhang, B. Rao, and L. Wang, "A reinforcement learning based resource management approach for time-critical workloads in distributed computing environment," *IEEE Int. Conf. on Big Data*, pp. 252–261, 2018.

[21] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[22] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in grid computing," *Future Gener. Comput. Syst.*, vol. 27, no. 5, pp. 430–439, May 2011.

[23] C. Ţăpuş, I.-H. Chung, and J. K. Hollingsworth, "Active harmony: Towards automated performance tuning," *ACM/IEEE Conf. on Supercomputing*, pp. 1–11, 2002.

[24] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin, "Autotune: A plugin-driven approach to the automatic tuning of parallel applications," *Applied Parallel and Scientific Computing*, pp. 328–342, 2013.

[25] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, "mARGOt: a Dynamic Autotuning Framework for Self-aware Approximate Computing," *IEEE Trans. on Computers*, vol. 68, no. 5, pp. 713–728, 2018.

[26] D. Gadioli, G. Palermo, and C. Silvano, "Application autotuning to support runtime adaptivity in multicore architectures," *Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 173–180, 2015.

[27] M. J. Voss and R. Eigenmann, "ADAPT: Automated de-coupled adaptive program transformation," *Int. Conf. on Parallel Processing*, 2000.

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

[29] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Complexity model based load-balancing algorithm for parallel tools of HEVC," *Visual Communications and Image Processing*, pp. 1–5, 2013.

[30] Y. Zhang, S. Kwong, and S. Wang, "Machine learning based video coding optimizations: A survey," *Information Sciences*, vol. 506, pp. 395–423, 2020.

[31] 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*, vol. 29, pp. 2268–2281, 2018.

[32] L. Costero, A. Iranfar, M. Zapater, F. D. Igual, K. Olcoz, and D. Atienza, "MAMUT: Multi-agent reinforcement learning for efficient real-time multi-user video transcoding," *Design, Automation & Test in Europe Conference*, pp. 558–563, 2019.

[33] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *Journal of Machine Learning Research*, vol. 5, no. Dec, pp. 1–25, 2003.

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

[35] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter-and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[36] S. T. Welstead, *Fractal and wavelet image compression techniques*. SPIE Optical Engineering Press, Bellingham, WA, 1999.

[37] N. T. S. Committee, *Report and Reports of Panel No. 11, 11-A, 12-19, with Some supplementary references cited in the Reports, and the Petition for adoption of transmission standards for color television before the Federal Communications Commission*. National Television System Committee, 1953.

[38] A. Iranfar, A. Pahlevan, M. Zapater, M. Žagar, M. Kovač, and D. Atienza, "Online efficient bio-medical video transcoding on MPSoCs through content-aware workload allocation," *Design, Automation & Test in Europe Conference*, pp. 949–954, 2018.

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

[40] C. Hayward and A. Madill, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[41] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing 2009*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173. [Online]. Available: http://link.springer.com/10.1007/978-3-642-11261-4_11

Luis Costero Luis M. Costero studied Mathematics and Computer Science at Universidad Complutense de Madrid (UCM) from 2010 to 2015, where he also obtained a Master's degree in Computer Science. His main research areas involve high performance computing, asymmetric processors, power consumption and resource management. He is currently pursuing a PhD related to the previously mentioned areas.

Arman Iranfar Arman Iranfar (S17) received the MS degree in electrical engineering, circuits and systems from the University of Tehran, Iran. He is currently working toward the PhD degree in electrical engineering in the Swiss Federal Institute of Technology Lausanne (EPFL). His research interest includes applied machine learning and reinforcement learning in multi-objective management of MPSoCs. He has published over 14 peer-reviewed papers in top-notch conferences and journals and served as reviewer in IEEE TC, and TSUSC. He is student member of the IEEE.

Marina Zapater Marina Zapater is Associate Professor in the School of Engineering and Management of Vaud (HEIG-VD), in the University of Applied Sciences Western Switzerland (HES-SO) since 2020, and Research Associate in the Embedded System Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland, since 2016. She received her Ph.D. degree in Electronic Engineering from Universidad Politcnica de Madrid, Spain, in 2015. Her research interests include thermal, power and performance design and optimization of complex heterogeneous architectures, from embedded edge devices to high-performance computing processors; and energy efficiency in servers and data centers. In these fields, she has co-authored more than 50 papers in top-notch conferences and journals, She is an IEEE and CEDA member, and has served as CEDA YP representative (2019-2020).

Francisco D. Igual Francisco D. Igual obtained the MS degree in Computer Engineering from University Jaume I de Castelln (Spain) in 2006, and the Ph.D. degree in Computer Science from the same University in 2011. In 2011, he joined the University of Texas at Austin as a post-doctoral researcher, and in 2012, he joined the Department of Computer Architecture from the University Complutense of Madrid as a Juan de la Cierva Fellow. Since 2019, he is an associate professor at the same University. His research interests include high-performance and energy-aware computing, dense linear algebra library development and optimization (collaborating with the SHPC group at the University of Texas at Austin), and runtime task scheduling on massively heterogeneous architectures. He has co-authored more than 50 papers on journals and conferences in the aforementioned fields.

Katzalin Olcoz Katzalin Olcoz is Associate Professor in the Department of Computer Architecture and System Engineering of the Complutense University. She received a Ph.D. degree in Physics in 1997 from the Complutense University (UCM) of Madrid. From 2012 to 2016 she served as head of the department of Computer Architecture and System Engineering of the same university. She was a visiting professor at EPFL (Lausanne, Switzerland) from April to June, 2018. Her research interests include high performance computing, resource management, energy efficiency and virtualization. Within the computer architecture group of the Complutense University, she has been involved in several projects in the field of computer architecture and design automation from high-level specifications, since 1992.

David Atienza David Atienza (M'05-SM'13-F'16) is associate professor of electrical and computer engineering, and heads the Embedded Systems Laboratory (ESL) at EPFL. He received his PhD degree in computer engineering from UCM, Spain, and IMEC, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MPSoC) and low-power Internet-of-Things (IoT) systems, including new thermal-aware design for MPSoCs and many-core servers, and ultra-low power edge AI architectures for IoT. He has co-authored more than 300 papers, several book chapters, and seven patents. He received the DAC Under-40 Innovators Award in 2018, IEEE TCCPS Mid-Career Award in 2018, an ERC Consolidator Grant in 2016, the IEEE CEDA Early Career Award in 2013, and the ACM SIGDA Outstanding New Faculty Award in 2012. He is an IEEE Fellow, an ACM Distinguished Member, and has served as IEEE CEDA President (period 2019-2020).