MPEC2: Multilayer and Pipeline Video Encoding on the Computing Continuum

Samira Afzal, Zahra Najafabadi Samani, Narges Mehran, Christian Timmerer, and Radu Prodan
Institute of Information Technology (ITEC), Alpen-Adria-Universität Klagenfurt, Austria
Email: {name}.{surname}@aau.at

***Abstract*—Video streaming is the dominating traffic in today's data-sharing world. Media service providers stream video content for their viewers, while worldwide users create and distribute videos using mobile or video system applications that significantly increase the traffic share. We propose a multilayer and pipeline encoding on the computing continuum (MPEC2) method that addresses the key technical challenge of high-price and computational complexity of video encoding. MPEC2 splits the video encoding into several tasks scheduled on appropriately selected Cloud and Fog computing instance types that satisfy the media service provider and user priorities in terms of time and cost. In the first phase, MPEC2 uses a multilayer resource partitioning method to explore the instance types for encoding a video segment. In the second phase, it distributes the independent segment encoding tasks in a pipeline model on the underlying instances. We evaluate MPEC2 on a federated computing continuum encompassing Amazon Web Services (AWS) EC2 Cloud and Exoscale Fog instances distributed on seven geographical locations. Experimental results show that MPEC2 achieves $24\,\%$ faster completion time and $60\,\%$ lower cost for video encoding compared to resource allocation related methods. When compared with baseline methods, MPEC2 yields $40\,\%$–$50\,\%$ lower completion time and $5\,\%$–$60\,\%$ reduced total cost.**

***Index Terms*—Cloud, Fog, computing continuum, video encoding, multilayer partitioning, encoding pipeline.**

I. Introduction

The global Internet phenomena report [1] revealed that video streaming dominated $53.72\,\%$ of the overall Internet traffic during the first half of 2021. YouTube, Netflix, and Facebook are the three popular over-the-top media service providers that stream the services to their users through web video player or mobile applications. Worldwide users generate and distribute videos through video-oriented mobile applications (*i.e.*, live video sharing, online gaming) from a wide range of client devices, mostly tablets and smartphones. Video traffic will increase significantly as new multimedia applications, such as augmented and virtual reality, surveillance systems, autonomous cars, and online learning, become available. Furthermore, the media service providers encode an initial video sequence into varying sequences of different bitrates, resolutions, and codecs for viewers with a variety of devices and network access characteristics. Such an increase in video data volume requires intensive and costly use of computing resources for highly complex encoding tasks.

Cloud computing introduced a new type of value-added video encoding services varying in price and performance [2], used by service providers for the downlink streaming scenarios of video content to their viewers. Fog computing extends Cloud services with micro-data centers (or cloudlets) [3] located towards the edge of the network, closer to the end-users. Fog resources cache stream videos for the user devices with lower latency in the downlink, and transcode user-generated videos for efficient streaming to other viewers in the uplink [4].

Related methods tackle this challenging problem by dividing a video sequence into smaller tasks, and distributing them among instances for encoding [5], [6]. These methods only focus on Cloud infrastructures, follow one objective (*i.e.*, time or cost), and do not consider the transmission time of video data that significantly affects the total encoding time and cost.

In this work, we focus on the video encoding scheduling on the computing instances in the continuum. We propose a *multilayer and pipeline encoding on the computing continuum (MPEC2)* offering improved time and cost performance based on the following operational steps: 1) *identify* scenes of similar visual complexity in a video stream, *split* each scene into segments, and *select* a segment of each scene; 2) *create* a three-layer graph of resource, network channel and cost features, and *partition* the computing instances with similar features by the multilayer graph partitioning method [7]; 3) *manage* the segment encoding tasks among Cloud and Fog instances in two phases: a) *select* an appropriate instance type from proper partitions to encode one video segment task, and b) *distribute* all the scene's segment tasks over the selected instances by defining a pipeline of segment encoding [8].

The main contributions of this work are the following:

- We formulate the scheduling problem as a multilayer and pipeline video encoding model aiming at minimizing video encoding completion time and cost by selecting the appropriate instance types of the Cloud and Fog;
- We benchmark the video encoding performance of AWS EC2 [9] and Exoscale [10] instances, respectively for ten and five different types, using `H.265` codec and various constant quantization parameters, over 500 video segments with different content complexity;
- We demonstrate that MPEC2 reduces the completion time of video encoding by $24\,\%$ and the total cost by $60\,\%$ compared to related work.

This paper has eight sections. Section II surveys the related work. Section III describes the model, followed by the architectural design in Section IV. Section V presents the proposed MPEC2 stream encoding algorithm. Section VI defines the experimental design, while Section VII elaborates on the

evaluation scenarios and result analysis. Finally, Section VIII concludes the paper.

II. Related Work

This section reviews the state-of-the-art about the video streaming services on the Cloud and Fog computing infrastructures, classified into three categories.

A. Time reduction

Long et al. [11] presented an architecture of multiple Fog devices cooperatively preprocessing video chunks generated by cameras. The architecture divides delay-sensitive video processing tasks in several sub-tasks, preprocessed by multiple Fog devices in parallel aiming for lower completion time. Aral et al. [12] considered Fog computing characteristics to improve the user experience for latency-sensitive video encoding. The service placement considers the network connectivity path and available bandwidth between the user device and the Cloud or Fog instances. Ali et al. [13] proposed a real-time video stream processing method using Fog and in-transit computing. The algorithm allocates the video processing application by preferring the low-latency Fog devices while satisfying the deadlines. In contrast, MPEC2 also considers the cost objective from the perspective of media provider or user with limited budgets that affect the computing service selection.

B. Cost reduction

Barais et al. [14] designed a low-cost video-transcoding method as a microservice application. Their results show that parallel executions of video transcoding microservices on small Fog devices cost lower than on private Cloud servers. Mehrabi et al. [15] proposed a Fog-assisted and cost-efficient video streaming method and greedy-based scheduling algorithm to map users to Fog servers. The aim was maximizing the quality of experience and fairness among mobile users, and load balancing of the Fog data centers. In contrast, MPEC2 also considers the encoding time objective, besides the cost objective, with regard to the video encoding deadline required by media providers or users.

C. Time and cost reduction

Veillon et al. [16] proposed an architecture for a federated Fog content delivery network aiming to reduce the latency and cost of fetching video segments. The architecture uses cached video segments at the neighboring fog delivery networks for encoding the on-demand video segment. Erfanian et al. [17] proposed a mixed binary linear programming method for lightweight transcoding on the Fog with reduced transcoding time and computation cost. This method extracts the encoding metadata from the origin server and employs for transcoding at the Fog. Mehran et al. [18] proposed a multi-objective optimization-based model for placing an application on the computing continuum. This method aims at minimizing the completion time and total cost along with the energy consumption of application execution on the Cloud and Fog instances. MPEC2 considers reducing the transmission times in addition to these works.

D. Contribution

Related methods model the video encoding on the Cloud and Fog instances but neglect resource and network utilization along with the economic cost. We extend these methods by researching a network, resource, and cost multilayer partitioning model that minimizes completion time and cost. Moreover, we explore the scene scheduling consisting of multiple segments passing through encoding pipeline stages.

III. Model

This section presents the formal computing infrastructure, network, encoding application, segment, scene, and stream encoding models along with the problem definition.

A. Computing infrastructure model

The *computing infrastructure* consists of $\mathcal{N}_{\mathcal{I}}$ homogeneous *clusters*: $\mathcal{I} = \{I_j | 1 \leq j \leq \mathcal{N}_{\mathcal{I}}\}$, where a cluster consists of $\mathcal{N}_j$ virtual machine (VM) instances of type $I_j \in \mathcal{I}$ with the same resource capabilities (*e.g.*, AWS `c5.2xlarge`).

1) Coordinator: I_ζ is a special instance in the computing infrastructure holding the entire video stream, which orchestrates other instances during the encoding process.

2) Instance type: $I_j = (\mathtt{CORE}_j, \mathtt{CPU}_j, \mathtt{CP}_j, \mathtt{MEM}_j, \mathtt{STOR}_j, \mathtt{CS}_j)$ is a vector representing the resource capabilities: 1) the number of the processing cores $\mathtt{CORE}_j$ with 2) the processing speed $\mathtt{CPU}_j$ measured in millions of instructions per second (MIPS), 3) available at a processing cost $\mathtt{CP}_j$ per hour of instance use, 4) the memory $\mathtt{MEM}_j$ size (in GB), 5) the storage $\mathtt{STOR}_j$ size (in GB) available at a storage cost $\mathtt{CS}_j$ per MB of use.

3) Network channels: $\mathcal{H} = \{h_{kj} \,|1 \leq j, k \leq \mathcal{N}_{\mathcal{I}}\}$ connect the homogeneous clusters, where $h_{kj} = (\mathtt{LAT}_{kj}, \mathtt{BW}_{kj}, \mathtt{CR}_{kj})$ models the round-trip latency $\mathtt{LAT}_{kj}$, bandwidth $\mathtt{BW}_{kj}$, and cost $\mathtt{CR}_{kj}$ for transmitting 1 MB of video stream between two instances from the clusters of instance types I_k and I_j grouped based on their geographical locations.

B. Encoding application model

We represent an *encoding application* $\mathcal{A} = (\mathcal{CD}, \mathcal{BR}, \mathcal{R}, \mathcal{PR}, \mathcal{U})$ with a codec $\mathcal{CD}$, bitrate $\mathcal{BR}$, resolution $\mathcal{R}$, preset $\mathcal{PR}$, and media provider's or user's time and cost requirements $\mathcal{U} = \{\theta, \alpha\}$. The user requires a deadline θs, or time priority of $\alpha = 1$ or cost priority of $\alpha = 0$.

C. Video stream model

Video stream $\mathtt{V} = (\mathtt{V}_x, \mathcal{N}_{\mathtt{V}})$ represents a sequence of $\mathcal{N}_{\mathtt{V}}$ scenes $\mathtt{V}_x$ $(1 \leq x \leq \mathcal{N}_{\mathtt{V}})$. A *video scene* $\mathtt{V}_x = (\mathtt{V}_{xy}, \mathcal{D}, \mathcal{N}_{\mathtt{V}_x})$ merges $\mathcal{N}_{\mathtt{V}_x}$ *video segments* $\mathtt{V}_{xy}$ $(1 \leq y \leq \mathcal{N}_{\mathtt{V}_x})$ with the same duration $\mathcal{D}$, as typically performed by HAS streaming services [19], [20].

D. Segment encoding model

We model in this section the time and cost of encoding a video segment $\mathtt{V}_{xy}$ requested by a coordinator I_ζ.

a) Completion time: $T(\mathcal{A}, \mathtt{V}_{xy}, I_j)$ of encoding a segment $\mathtt{V}_{xy}$ by an application $\mathcal{A}$ on an instance I_j is the sum of the transmission and encoding times:

$$T\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right) = Tr\left(I_\zeta, \mathtt{V}_{xy}, I_j\right) + Te\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right).$$

b) Transmission time: $Tr\left(I_\zeta, \mathtt{V}_{xy}, I_j\right)$ is the ratio between the size of a segment $\mathtt{V}_{xy}$ and the network bandwidth $\mathtt{BW}_{\zeta j}$, plus the round-trip time $\mathtt{LAT}_{\zeta j}$ from the coordinator I_ζ to $I_j = \mu\left(\mathcal{A}, \mathtt{V}_{xy}\right)$:

$$Tr\left(I_\zeta, \mathtt{V}_{xy}, I_j\right) = \frac{\mathtt{sizeof}(\mathtt{V}_{xy})}{\mathtt{BW}_{\zeta j}} + \mathtt{LAT}_{\zeta j}.$$

c) Encoding time: $Te\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right)$ of a video segment $\mathtt{V}_{xy}$ on an instance I_j is:

$$Te\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right) = \frac{\mathtt{CPU}\left(\mathcal{A}, \mathtt{V}_{xy}\right)}{\mathtt{CPU}_\zeta} \cdot \mathtt{RS}_{\zeta j},$$

where $\mathtt{CPU}\left(\mathcal{A}, \mathtt{V}_{xy}\right)$ is the computational requirement (in million of instructions (MI)) to encode a segment $\mathtt{V}_{xy}$ by an application $\mathcal{A}$ on the coordinator I_ζ with the processing speed $\mathtt{CPU}_\zeta$, and $\mathtt{RS}_{\zeta j}$ is its *relative processing speed* to type I_j:

$$\mathtt{RS}_{\zeta j} = \frac{\mathtt{CPU}_\zeta}{\mathtt{CPU}_j}.$$

d) Encoding cost: $C\left(\mathcal{A}, \mathtt{V}_{xy}, I_j, I_\zeta\right)$ of a segment $\mathtt{V}_{xy}$ by an application $\mathcal{A}$ on an instance I_j is the sum of its processing $\mathtt{CP}_j$, storage $\mathtt{CS}_j$, and transmission costs $\mathtt{CR}_{kj}$ [21]:

$$C\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right) = Te \cdot \mathtt{CP}_j + \mathtt{sizeof}(\mathtt{V}_{xy}) \cdot \mathtt{CS}_j + Tr \cdot \mathtt{CR}_{\zeta j}.$$

E. Scene encoding model

We model, in this section, the time and cost of encoding a video scene consisting of multiple segments.

1) Completion time: $T\left(\mathcal{A}, \mathtt{V}_x, I_j\right)$ of encoding a scene $\mathtt{V}_x$ is the sum of the waiting, transmission, and encoding times of all its segments on a set of instances I_j:

$$T\left(\mathcal{A}, \mathtt{V}_x, I_j\right) = Tw\left(\mathcal{A}, \mathtt{V}_x, I_j\right) + Tr\left(I_\zeta, \mathtt{V}_x, I_j\right) + \\ + Te\left(\mathcal{A}, \mathtt{V}_x, I_j\right).$$

a) Waiting time: $Tw\left(\mathcal{A}, \mathtt{V}_x, I_j\right)$ required by a scene $\mathtt{V}_x$ is the transmission time of all its previous video scenes $\mathtt{V}_z$:

$$Tw\left(\mathcal{A}, \mathtt{V}_x, I_j\right) = \begin{cases} 0, & x = 1; \\ \sum_{z=1}^{x-1} Tr\left(I_\zeta, \mathtt{V}_z, I_j\right), & x \neq 1. \end{cases}$$

b) Transmission time: $Tr\left(I_\zeta, \mathtt{V}_x, I_j\right)$ of a video scene $\mathtt{V}_x$ from the coordinator I_ζ to a set of instances I_j takes place in multiple pipeline stages, as illustrated in Figure 1. Each stage fully utilizes the bandwidth to each instance by maximizing the number of transmitted segments per second (where $\mathtt{V}_{xy} \in \mathtt{V}_x$):

$$\mathcal{N}_{\zeta j} = \frac{\mathtt{BW}_{\zeta j}}{\mathtt{sizeof}(\mathtt{V}_{xy})}.$$

The scene transmission time considers the segment transmission to the $\left\lceil \frac{\mathcal{N}_j}{\mathcal{N}_{\zeta j}} \right\rceil$ homogeneous groups of $\mathcal{N}_{\zeta j}$ instances:

$$Tr\left(I_\zeta, \mathtt{V}_x, I_j\right) = Tr\left(I_\zeta, \mathtt{V}_{xy}, I_j\right) \cdot \left\lceil \frac{\mathcal{N}_j}{\mathcal{N}_{\zeta j}} \right\rceil,$$

where it is important to note that while the transmission time of segments to the first stages of instances of each group is considered in this formula, the segment transmission time to the following stages is ignorable due to their overlap with their previous segment encoding stages.

c) Encoding time: of a video scene $\mathtt{V}_x$ on the set of instance types I_j is the total segment encoding times along the pipeline stages:

$$Te\left(\mathcal{A}, \mathtt{V}_x, I_j\right) = Te\left(\mathcal{A}, \mathtt{V}_{xy}, I_j\right) \cdot \left\lceil \frac{\mathcal{N}_{\mathtt{V}_x}}{\mathcal{N}_j} \right\rceil,$$

where $\left\lceil \frac{\mathcal{N}_{\mathtt{V}_x}}{\mathcal{N}_j} \right\rceil$ is number of stages in an encoding pipeline.

F. Stream encoding model

We model in this section the completion time and total cost of an application $\mathcal{A}$ for encoding a video stream $\mathtt{V}$ consisting of multiple scenes $\mathtt{V}_x \in \mathtt{V}$ on a set of instances $\mathcal{I}$.

1) Completion time: $T\left(\mathcal{A}, \mathtt{V}, \mathcal{I}\right)$ is the latest completion time of all the scenes $\mathtt{V}_x \in \mathtt{V}$:

$$T\left(\mathcal{A}, \mathtt{V}, \mathcal{I}\right) = \max_{\forall \mathtt{V}_x \in \mathtt{V}} \left\{T\left(\mathcal{A}, \mathtt{V}_x, I_j\right)\right\}.$$

2) Total cost: $C\left(\mathcal{A}, \mathtt{V}, \mathcal{I}\right)$ aggregates the transmission costs from the coordinator with the storage and processing costs on the encoding instances of its scenes $\mathtt{V}_x \in \mathtt{V}$:

$$C\left(\mathcal{A}, \mathtt{V}, \mathcal{I}\right) = \sum_{\forall \mathtt{V}_x \in \mathtt{V}} \{Tr\left(I_\zeta, \mathtt{V}_x, I_j\right) \cdot \mathtt{CR}_{\zeta j} + \\ + \mathtt{sizeof}(\mathtt{V}_x) \cdot \mathtt{CS}_j + Te\left(\mathcal{A}, \mathtt{V}_x, I_j\right) \cdot \mathtt{CP}_j\}.$$

G. Example of scene encoding pipelines

Figure 1 depicts an example encoding pipeline of a video stream with two scenes $\mathtt{V} = \{\mathtt{V}_1, \mathtt{V}_2\}$ comprising $\mathcal{N}_{\mathtt{V}_1} = 8$ and $\mathcal{N}_{\mathtt{V}_2} = 4$ segments, respectively. We consider an infrastructure with two instance types $\mathcal{I} = \{I_1,\ I_2\}$, with four instances of the first type $\mathcal{N}_1 = 4$ and two of the second $\mathcal{N}_2 = 2$. We assume that the maximum number of segments per second transmitted from I_ζ to both instance types are $\mathcal{N}_{\zeta 1}$=2 and $\mathcal{N}_{\zeta 2}$=1, respectively. Thus, we have $\left\lceil \frac{\mathcal{N}_1}{\mathcal{N}_{\zeta 1}} \right\rceil$=2 and $\left\lceil \frac{\mathcal{N}_2}{\mathcal{N}_{\zeta 2}} \right\rceil$=2 homogeneous groups of instances receiving the segments through the first stages. Thus, encoding the scene $\mathtt{V}_1$ needs two stages on the four instances of type I_1, while the scene $\mathtt{V}_2$ needs two stages on the two instances of type I_2.

H. Problem definition

We define in this section the problem of scheduling an encoding application.

a) Segment encoding schedule: is a function $\mu : (\mathcal{A}, \mathtt{V}_{xy}) \rightarrow \mathcal{I}$ that assigns a segment $\mathtt{V}_{xy}$ of an encoding application $\mathcal{A}$ to an instance type $I_j = \mu\left(\mathcal{A}, \mathtt{V}_{xy}\right)$.

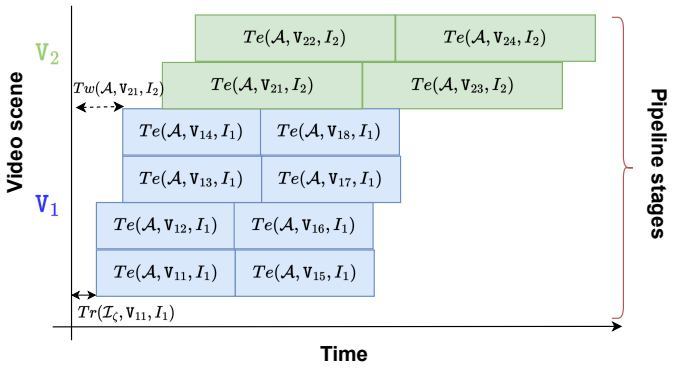


Fig. 1: Scene encoding pipeline example.

b) Segment scheduling objective: minimizes the aggregation of the completion time and cost of the segment encoding:

$$O_1(\mathcal{A}, \mathtt{V}_{xy}) = \min_{\mu(\mathcal{A}, \mathtt{V}_{xy}) = I_j} \{\alpha \cdot T(\mathcal{A}, \mathtt{V}_{xy}, I_j) + \\ + (1-\alpha) \cdot C(\mathcal{A}, \mathtt{V}_{xy}, I_j)\},$$

where $\alpha \in [0, 1]$ is a media service or user provider tradeoff. We instantiate this objective following particular problems:

1) *time-optimized* scheduling when $\alpha = 1$;
2) *cost-optimized* scheduling when $\alpha = 0$;
3) *deadline-constrained* scheduling with minimal cost tradeoff: $T(\mathcal{A}, \mathtt{V}_{xy}, I_j) \leq \theta \ \wedge \ \min(\alpha)$.

c) Stream scheduling objective: minimizes the completion time by execution of the scenes in pipeline stages:

$$O_2(\mathcal{A}, \mathtt{V}) = \min_{\mu(\mathcal{A}, \mathtt{V}) = \mathcal{I}} T(\mathcal{A}, \mathtt{V}, \mathcal{I}).$$

IV. MPEC2 Architecture

Figure 2 represents an overview of the MPEC2 architecture for encoding a video using different codecs or/and resolutions with reduced completion time and cost. MPEC2 runs on a coordinator I_ζ that accesses a video stream, makes scheduling decisions, and distributes segments for encoding on appropriate computing instances. MPEC2 has five components, described as follows.

A. Scene detector

The scene detector identifies scenes of similar visual complexity in an input video, and splits each scene into segments for encoding, based on a default segment duration $\mathcal{D}$. We select the middle segment $\mathtt{V}_{xm}$ of a scene $\mathtt{V}_x \in \mathtt{V}$ to represent all the segments of that scene. The reason is that all the segments of each scene have similar content complexity, encoding time, and size [5]. The selected segments $\mathtt{V}_{xm}$ are then sent to other components for further processing.

B. Multilayer graph generator

To handle the instance diversities, we model the computing infrastructure as totally interconnected *multilayer graph* $G = (\mathcal{I}, \mathcal{E}, \mathcal{L})$, where $\mathcal{I}$ is the set of instances and $\mathcal{E}$ is the set of interconnections between different resources, network and cost layers $\mathcal{L} = \{l_i | 0 \leq i < 3\}$:

a) Network layer (l_0): captures the network interconnections between instances using the round-trip network latency $\mathtt{LAT}_{kj}$ and bandwidth $\mathtt{BW}_{kj}$, modeled in Section III-A.

b) Resource layer (l_1): indicates the similarity between instances with respect to their processing speed $\mathtt{CPU}_j$, number of cores $\mathtt{CORE}_j$, memory $\mathtt{MEM}_j$, and storage $\mathtt{STOR}_j$, modeled in Section III-A.

c) Cost layer (l_2): defines the similarity between instances based on their processing $\mathtt{CP}_j$, storage $\mathtt{CS}_j$, and transmission costs $\mathtt{CR}_j$, modelled in Section III-D0d.

The *edges* $\mathcal{E}$ in the multilayer graph are of two types:

d) Intra-layer edges: $\mathcal{E}_{ll} = \{(I_k, I_j) \in \mathcal{I} \times \mathcal{I} | k \neq j\}$ connect two instances in one layer $l \in \mathcal{L}$ with a weight w^l_{kj} modelled according to the similarity score in each layer $l \geq 1$:

$$w^l_{kj} = \frac{1}{1 + d_l(I_k, I_j)}.$$

where $d_l(I_k, I_j)$ is the Euclidean distance between instance types I_k and I_j in layer $l \geq 1$.

e) Inter-layer edges: $\mathcal{E}_{ll'} = \{(I_j, I_j) \in \mathcal{I} \times \mathcal{I}\}$ connect an instance I_j in the layer $l \in \mathcal{L}$ with the corresponding instance I_j in all the other layers $l' \in \mathcal{L}$, where $l \neq l'$.

C. Partitioner

Classifying the instances based on their features narrows the search space to fewer partitions, decreases the search complexity, and improves the scheduling decision time [7].

1) Features: We define three features for classification:

a) Network channel features: are network latency $\mathtt{LAT}_{\zeta j}$ and bandwidth $\mathtt{BW}_{\zeta j}$ between the coordinator I_ζ and the instance type I_j.

b) Resource features: are processing speed $\mathtt{CPU}_j$, memory $\mathtt{MEM}_j$ and storage sizes $\mathtt{STOR}_j$ for an instance type I_j.

c) Cost features: are the costs of encoding $\mathtt{CP}_j$, storage $\mathtt{CR}_j$ and transmission $\mathtt{CR}_j$ of an instance type I_j.

2) Instance classification and partitioning: has three steps.

a) Layer partitioning: defines a set of disjoint partitions for each layer using the Louvain clustering algorithm [22]. Each partition contains instances with similar network (latency and bandwidth), resources (processing speed, memory and storage) or cost (encoding, storage, transmission) features. Layer partitioning targets a single objective, such as transmission time, processing speed, or cost.

b) Graph compression: merges each partition containing instances with similar features (*e.g.*, network, resources, costs) into a single node. In addition, it merges the edges between the partitions into a single edge.

c) Compressed graph classification: defines a set of partitions $\mathcal{P} = \{p_z | 1 \leq z \leq \mathcal{N}_p\}$ with similar network, resources and cost features across all layers using the Louvain clustering algorithm and aiming for multiple objectives, such as data transmission time, processing speed, and cost tradeoffs. Each partition has aggregated network channel, resource and cost features $f_p = (\mathtt{LAT}_p, \mathtt{BW}_p, \mathtt{CPU}_p, \mathtt{MEM}_p, \mathtt{STOR}_p, \mathtt{CP}_p, \mathtt{CS}_p, \mathtt{CR}_p)$, calculated as their average across all its instances.

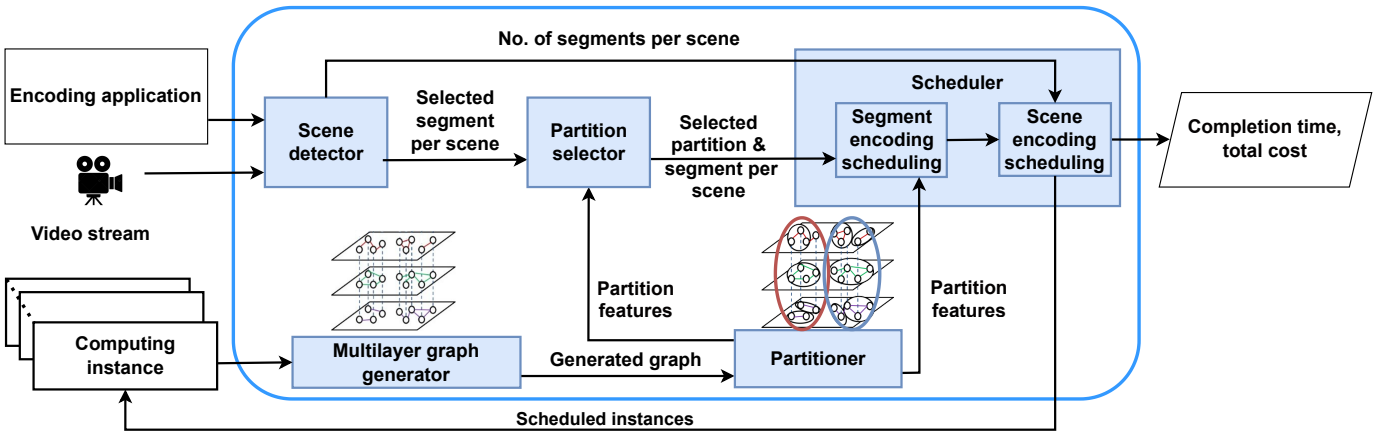


Fig. 2: MPEC2 architecture overview.

D. Partition selector

This component selects one partition $p \in \mathcal{P}$ for placing an application according to the media provider or user deadline, time or cost priority:

$$p = \begin{cases} \max\limits_{\forall p \in \mathcal{P}} \{\texttt{SIM}(\mathcal{A}, \texttt{V}_{xm}, f_p)\}, & T \leq \theta; \\ \min\limits_{\forall p \in \mathcal{P}} \{T(\mathcal{A}, \texttt{V}_{xm}, f_p)\}, & \alpha = 1; \\ \min\limits_{\forall p \in \mathcal{P}} \{C(\mathcal{A}, \texttt{V}_{xm}, f_p)\}, & \alpha = 0; \end{cases}$$

a) $\max\limits_{\forall p \in \mathcal{P}} \{\texttt{SIM}(\mathcal{A}, \texttt{V}_{xm}, f_p)\}$*:* selects a partition p with average completion time of segment encoding that has maximum similarity [7] to the deadline θ. The similarity function SIM relies on the Euclidean distance between partition features, and provider or user priority α to meet the deadline θ with minimum cost [7].

b) $\min\limits_{\forall p \in \mathcal{P}} \{T(\mathcal{A}, \texttt{V}_{xm}, f_p)\}$*:* selects a partition p with minimum average completion time for encoding the segment $\texttt{V}_{xm} \in \texttt{V}$ regarding the user's time or cost priority α.

c) $\min\limits_{\forall p \in \mathcal{P}} \{C(\mathcal{A}, \texttt{V}_{xm}, f_p)\}$*:* selects a partition p with minimum cost for encoding a video segment $\texttt{V}_{xm} \in \texttt{V}$ regarding the user's time or cost priority α.

E. Segment and scene encoding scheduler

1) Segment encoding scheduling: selects an instance type $I_j \in p$ that optimizes the objective O_1 defined Section III-H, representing the tradeoff between the completion time (aggregating the transmission and encoding times, explained in Section III-D) and the processing cost based on the user or provider time and cost priority α, as follows.

a) Cost-optimized scheduling: sorts all instances in the partition in ascending order based on their estimated costs, and selects the cheapest one. In case of more instances with equal cost, it considers the completion time as the second priority.

b) Time-optimized scheduling: sorts all instances in the partition in ascending order based on their completion times and selects the one with the fastest one. In case of more instance selections, it considers the cost as the second priority.

Algorithm 1 MPEC2 scheduler.

Input: $\mathcal{A} = (\mathcal{CD}, \mathcal{BR}, \mathcal{R}, \mathcal{PR}, \mathcal{U})$ ▷ Encoding application
$\texttt{V}$ ▷ Video stream
$\mathcal{I} = \{I_j | 1 \leq j \leq \mathcal{N}_\mathcal{I}\}$ ▷ Cloud and Fog homogeneous *clusters* set
$\mathcal{H} = \{h_{kj} | 1 \leq j, k \leq \mathcal{N}_\mathcal{I}\}$ ▷ Network channel set
$\mathcal{N}_j;\ \forall I_j \in \mathcal{I}$ ▷ Number of instances of each type
Output: $(T, C, \mu(\mathcal{A}, \texttt{V}_{xy}));\forall \texttt{V}_{xy} \in \texttt{V}$ ▷ Time, cost, and schedule of all segments

1: **function** MPEC2($\mathcal{A}, \texttt{V}, \mathcal{I}, \mathcal{H}, \mathcal{N}_j$)
2: $\mathcal{G} \leftarrow$ GENERATEML($\mathcal{I}, \mathcal{H}$) ▷ Generate a multilayer graph
3: $\mathcal{P} \leftarrow$ PARTITION($\mathcal{G}$) ▷ Partition the multilayer graph
4: $(\texttt{V}_x, \mathcal{N}_\texttt{V}) \leftarrow$ DETECTSCENES($\texttt{V}$), $\forall x \in [1, \mathcal{N}_\texttt{V}]$ ▷ Detect scene
5: **for all** $\texttt{V}_x \in \texttt{V}$ **do** ▷ Iterate scenes
6: **for all** $\texttt{V}_{xm} \in \texttt{V}_x$ **do** ▷ Iterate selected segments
7: $f_p \leftarrow (\texttt{LAT}_p, \texttt{BW}_p, \texttt{CPU}_p, \texttt{MEM}_p, \texttt{STOR}_p, \texttt{CP}_p, \texttt{CS}_p, \texttt{CR}_p)$; $\forall p \in \mathcal{P}$
8: $p \leftarrow$ SELECTPARTITION($\mathcal{A}, \texttt{V}_{xm}, f_p$) ▷ Select partition
9: $\mu(\mathcal{A}, \texttt{V}_{xm}) \leftarrow$ SCHEDULESEGMENT($\mathcal{A}, \texttt{V}_{xm}, p$) ↪ ▷ Schedule segment encoding
10: **end for**
11: $T, C, \mu \leftarrow$ SCHEDULESCENE($\mathcal{A}, \texttt{V}_x, \mu, \mathcal{N}_j$) ▷ Schedule scene encoding
12: **end for**
13: **return** (T, C, μ);
14: **end function**

c) Deadline-constrained scheduling: selects an instance that satisfies the deadline with the lowest cost.

2) Scene encoding scheduling: allocates the segments of the video scenes that compose a stream to the instances in the computing continuum (see Section III-B), optimizing the objective O_2 defined in Section III-H. The scene scheduling distributes the independent segments in a pipeline model on the instances, where each pipeline comprises a number of encoding stages [8].

V. MPEC2 Scheduling Algorithm

Algorithm 1 describes the MPEC2 pseudocode for encoding an application $\mathcal{A}$ with an input video stream $\texttt{V}$ on a set of instance types $\mathcal{I}$ interconnected through a set of network channels $\mathcal{H}$. There are $\mathcal{N}_j$ instances of each type $I_j \in \mathcal{I}$. Firstly, the algorithm applies the Louvain clustering and calculates the multilayer graph of the computing infrastructure (line 2). Line 3 splits the multilayer graph into partitions based on the instances, network channels, and their cost similarity. Line 4 detects the scenes, splits them into segments of the duration requested by the media provider or user, and selects a segment

of each video scene. The algorithm loops between lines 6 – 10 until it schedules all segment encoding tasks on the instances. In particular, it selects a partition including an instance for the segment encoding, and sets the scheduling function (lines 8 – 9). Line 11 schedules the pipeline of the video scene's segments based on the number of instances of each selected instance type. Finally, line 13 returns the completion time and total cost along with the scheduled instances.

VI. Experimental Setup

We implemented the MPEC2 in `Python 3.10.5`. We used the `PySceneDetect` library to detect the video scenes in a stream. Moreover, we used the `NetworkX` Python package to implement the Cloud and Fog multilayer graph along with the partitioning model. We configured the network bandwidth by measuring maximum achievable throughput with `iPerf3` [23] and the round-trip latency with the `ICMP` echo request and reply tool.

A. Experimental infrastructure testbed

We run the experiments on the Carinthian Computing Continuum (C^3) testbed [24] with instances from two providers distributed across the Cloud and Fog computing layers (see Table I). The instance types are heterogeneous in number and resource capabilities, such as number of cores, processing speed, and memory size.

AWS: We utilized compute-optimized, general-purpose, and memory-optimized AWS instances in Frankfurt and London. We shared a multi-attached general-purpose solid state disk volume (500 GB) for storing video dataset among the AWS instances in the same availability zone.

Exoscale: We used compute-optimized Exoscale [10] instances located in Geneva, Zurich, Munich, Frankfurt, Vienna, and Klagenfurt. We attached a single 400 GB storage volume to each instance for the experiments on the Exoscale.

Encoding costs: We estimated the costs for Cloud and Fog instances and of transmitting video streams through the bandwidth allocated to the shared or isolated storage using the Exoscale [25] and AWS [26] calculators. Aside from the compute cost calculations [27], for the local storage cost estimations, we used the information provided by AWS [28] and Exoscale [29]. We relied on the AWS calculator [30] for the data transfer cost estimation and the Exoscale outbound prices [31].

B. Encoding benchmark

We prepared the test video streams using `FFmpeg v3.4.11` from the publicly available video complexity dataset [32], [33]. It comprises 47 video sequences of 4 k ultra high definition (UHD) resolution with $\mathcal{R}$=3840×2160 pixels in YUV format. We divided the raw videos into 500 segments of $\mathcal{D} = 5$ s, each with the same 23.98 frame/s, 4 : 2 : 0 YUV format, and 8 bit/pixel. We encoded each YUV segment using the $\mathcal{CD}$=`H.265` video codec with the lossless method that generates the highest possible quality compared to the original videos. We used the `libx265` library with a

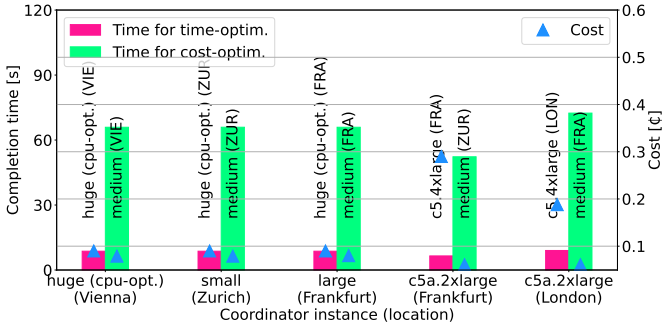


Fig. 3: Time-optimized and cost-optimized segment encoding results.

preset $\mathcal{PR}$ of *medium* and a constant quantization parameter of 37, controlling the amount of compression. Finally, we benchmarked the encoding of the 47 video streams on ten commonly used AWS EC2 Cloud and six Exoscale Fog instances (see Table I), created the instance encoding dataset, and extracted the encoding times in the 4 s–75 s range. **Results**: Table II shows the average encoding time of all segments on each instance relative to the coordinator instances `c5.2xlarge` and `large`. As observed, the relative processing speed `RS` of instances is, on average, ∼6 % different from the relative processing speed benchmarks [5]. The reason is that the `CPU` speed directly impacts on the video encoding time [34]. Therefore, this observation validates the use of instance relative speed `RS` for the time and cost estimations (see Section III-D).

VII. Evaluation

We selected a video stream with three different scenes that vary in content complexity, with different number of segments and sizes presented in Table III. We define three experiments to evaluate MPEC2 on our computing continuum testbed.

- *Segment encoding scheduling* considering various provider or user priorities (*i.e.*, time, cost, deadline) and coordinator locations.
- *Scene encoding scheduling* with various number of segments per scene in time- and cost-optimized scenarios.
- *Related work comparison* with *MAPO* [18], *random*, *fastest*, and *cheapest* instance selection methods.

A. Segment encoding scheduling

We investigate the impact of the media service provider or user priorities on the completion time and cost of one segment encoding. The coordinator can be a Cloud instance for media service providers or a Fog instance in surveillance systems, for example, where users are close to the edge of the network. In addition, we validate MPEC2 with the actual values (see Section VI-B) in time- and cost-optimized, along with deadline-constrained scenarios. The selected segment of the scene $\mathtt{V}_1$ with $\mathtt{sizeof}(\mathtt{V}_{1m}) = 330$ MB is evaluated in the following scenarios.

TABLE I: Experimental Cloud and Fog infrastructure testbed.

Provider	*AWS*										*Exoscale*					
Location	Frankfurt, London										Geneva, Zurich, Munich, Frankfurt, Vienna					Klagenfurt
Instance	compute-optimized				general-purpose			memory-optimized			CPU-optimized	standard				standard
type	c5.2xlarge	c5.4xlarge	c5.9xlarge	c5a.2xlarge	m5.xlarge	m5.2xlarge	m5.4xlarge	r5.xlarge	r5.2xlarge	r5.4xlarge	huge (cpu-opt)	huge	large	medium	small	large
No. instances ($\mathcal{N}_j$)	10	10	5	10	15	10	5	15	10	5	10	5	10	10	10	10
CORE (cores)	8	16	36	8	4	8	16	4	8	16	16	8	4	2	2	4
CPU (MIPS)	21 100	44 200	90 700	23 000	10 200	19 700	38 400	10 250	19 700	38 900	48 000	25 000	14 000	7200	7100	12 000
MEM (GB)	16	32	72	16	16	32	64	32	64	128	32	32	8	4	2	8
STOR (GB)	500										400					
BW (Mbit s^{-1})	748–5089	748–5089	748–5089	748–5089	748–5089	748–5089	748–5089	748–5089	748–5089	748–5089	480–13 000	480–13 000	480–13 000	480–13 000	480–13 000	500–13 000
LAT (ms)	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–28	0.5–26	0.5–26	0.5–26	0.5–26	0.5–26	7–12
CP$_j$ (¢/h)	0.388	0.776	1.746	0.348	0.23	0.46	0.92	0.304	0.608	1.216	0.44	0.36	0.089	0.044	0.022	0.089
monthly CS$_j$ (¢/GB)	0.053–0.0952										0.0432					
CR$_{kj}$(¢/GB)	0.05–0.09										0.02					

TABLE II: MPEC2 vs. benchmark relative processing speeds of Cloud and Fog instances.

Instance type	c5.2xlarge	c5.4xlarge	c5.9xlarge	c5a.2xlarge	m5.xlarge	m5.2xlarge	m5.4xlarge	r5.xlarge	r5.2xlarge	r5.4xlarge	huge (cpu-opt)	huge	large	medium	small	large
RS to c5.2xlarge	1	2.1	4.3	1.1	0.48	0.93	1.82	0.46	0.93	1.84	2.27	1.18	0.66	0.34	0.34	0.57
MPEC2 processing speed relative to c5.2xlarge	1	1.96	3.4	1.2	0.43	0.88	1.72	0.44	0.88	1.69	2.4	1.26	0.65	0.32	0.32	0.58
RS to *large*	1.51	3.16	6.48	1.64	0.73	1.41	2.74	0.73	1.41	2.78	3.43	1.79	1.00	0.51	0.51	0.86
Benchmark processing speed relative to large	1.54	2.99	5.18	1.63	0.67	1.36	2.65	0.67	1.36	2.62	3.65	1.95	1.00	0.52	0.52	0.88

TABLE III: Video stream characteristics.

Scene	*Number of segments*	*Selected segment size* (MB)
V_1	25	330
V_3	20	475
V_2	15	240

1) Time versus cost-optimized scenarios: We configured the media provider or user priority to $\alpha = 0$ for cost-optimized and $\alpha = 1$ for time-optimized scenarios to investigate the impact of user priorities on the completion time and cost of segment encoding. MPEC2 selects instances at the same location with the coordinator in all time-optimized scenarios to reduce the transmission time. Figure 3 shows that the completion time of encoding one segment for time-optimized scenario when coordinator is `huge (cpu-opt)` (Vienna) (8.69 s) is approximately 7.35 times faster than its cost-optimized one (65.91 s). Likewise, the cost-optimized segment encoding (0.079 ¢) is about eight times cheaper than time-optimized (0.1 ¢). Overall, the time-optimized scenarios are 86 % faster and 77 % more expensive than the cost-optimized ones.

a) Coordinator in Vienna: MPEC2 does not select the fastest `c5.9xlarge` instance in Frankfurt (with 36 cores and 90 700 MIPS processing speed), since the transmission time (13.34 s) is an order of magnitude longer compared to the `huge (cpu-opt)` instance at the same location (0.519 s).

b) Coordinator in London: The cost-optimized scenario selects a `medium` instance in Zurich. Interestingly, MPEC2 does not select the Exoscale `small` instance with two cores, 7100 MIPS speed and the lowest cost (*i.e.*, 0.08 ¢ versus 0.079 ¢) due to its longer encoding time (*i.e.*, 131.16 s versus 65.91 s) at a slightly higher price. Although the difference in the price in negligible for one segment, it imposes higher price for encoding tens of segments.

2) Deadline-constrained scenario: We configured a constant deadline of $\theta = 10$ s, based on the actual transcoding time presented in [35] to investigate the completion time and cost of segment encoding. We selected `c5a.2xlarge` (Frankfurt) as coordinator.

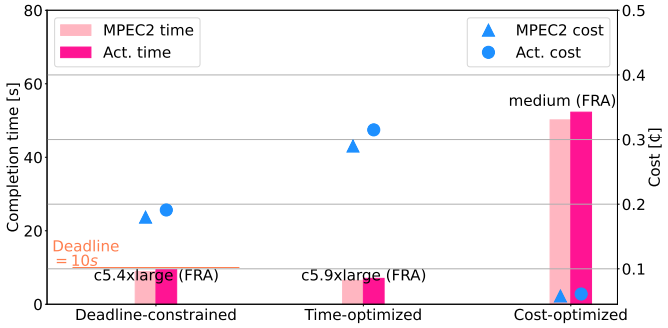


Fig. 4: Segment encoding completion time and cost in deadline-constrained, time-optimized, and cost-optimized scenarios with `c5a.2xlarge` (Frankfurt) coordinator.

Figure 4 shows that, in the deadline-constrained scenario, MPEC2 selects `c5.4xlarge` (Frankfurt) to encode a segment with the completion time of 8.93 s that, as expected, is less than 10 s. This completion time (8.93 s) is higher than the time-optimized result (6.51 s) that selects `c5.9xlarge` (Frankfurt). Particularly, in a deadline-constrained scenario, MPEC2 allocates the instance type with the lowest cost among the instances that satisfy the deadline (see Figure 5). Therefore, in this experiment, MPEC2 selects `c5.4xlarge` with the lowest cost (0.18 ¢), while two other instances `m5.4xlarge` (0.238 ¢) and `c5.9xlarge` (0.246 ¢) are also able to satisfy the deadline. On the other hand, the total cost for the deadline-constrained scenario is more expensive (0.18 ¢) than the cost-optimized one (0.058 ¢) with the medium instance.

We analyze the MPEC2 scheduling for four coordinator locations in Figure 5 as follows:

a) Exoscale `huge (cpu-opt)` (Vienna) and `small` (Zurich): MPEC2 selects `huge (cpu-opt)` instances for both coordinator locations since they provide completion times lower than the deadline.

b) Exoscale `large` (Frankfurt): MPEC2 first selects the Exoscale `huge (cpu-opt)` (Frankfurt) and AWS `c5.9xlarge` (Frankfurt) instance types with completion

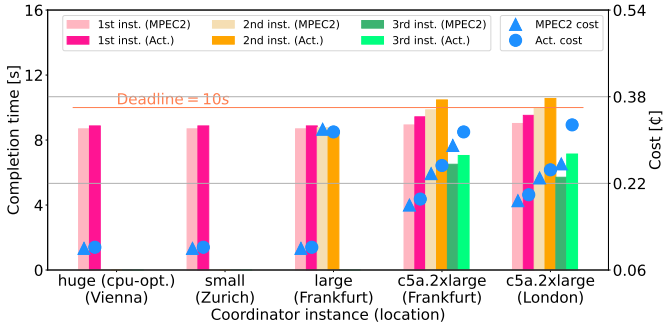
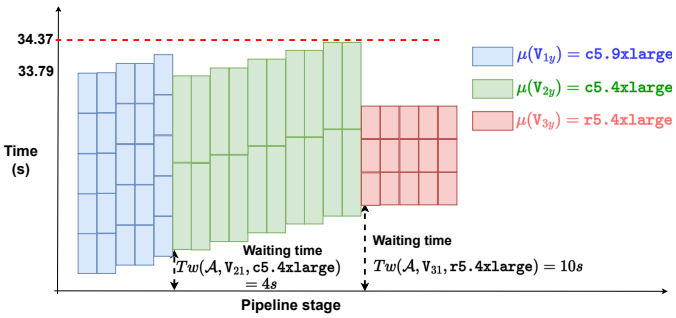
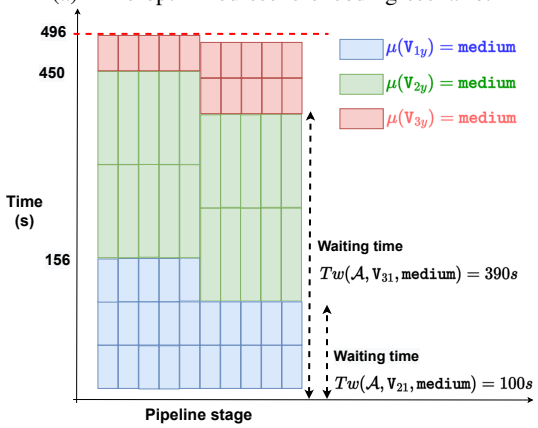


Fig. 5: Deadline-constrained encoding completion time and cost for different coordinator locations.



(a) Time-optimized scene encoding scenario.



(b) Cost-optimized scene encoding scenario.

Fig. 6: Scheduling results of scene encoding pipeline.

times lower than 10 s. Among them, MPEC2 prefers the Exoscale `huge (cpu-opt)` (Frankfurt) instance that satisfies the deadline with the lowest cost of 0.1 ¢.

c) AWS `c5a.2xlarge` (Frankfurt): MPEC2 selects first three AWS instance types: `c5.4xlarge` (Frankfurt), `m5.4xlarge` (Frankfurt), and `c5.9xlarge` (Frankfurt) that satisfy the deadline. Among them, it prefers the `c5.4xlarge` (Frankfurt) instance with the lowest cost.

d) AWS `c5a.2xlarge` (London): MPEC2 selects the AWS `c5.4xlarge` (London) instance with 0.188 ¢ encoding cost, following the same strategy.

3) Accuracy analysis: Figure 4 further compares the MPEC2 estimated segment encoding completion time and cost with the actual measurements. The results show that MPEC2 estimations are close to the actual values with the accuracy of 95 % for deadline-constrained, 93 % for time-optimized, and 96 % for cost-optimized scenarios. Similarly, Figure 5 shows that the actual completion times and costs validate MPEC2 estimated values with an accuracy above 90 % in all scenarios.

B. Scene encoding scheduling

We evaluate the scene encoding scheduling using the completion time and total cost of encoding all video segments having `c5a.2xlarge` (Frankfurt) as the coordinator.

1) Time-optimized scenario: In the first phase, MPEC2 selects an instance type for encoding the segments of each video scene. In the second phase, it allocates the segments to the selected instance types by following a pipeline model.

a) Segment encoding scheduling: selects the instance type with the lowest completion time for segment encoding among the available instances displayed in Table IV. Thus, it selects the `c5.9xlarge` for the first scene, `c5.4xlarge` for the second scene, and `r5.4xlarge` for the last scene.

b) Scene encoding scheduling: depicted in Figure 6a, allocates the segments in an encoding pipeline of $\mathcal{N}_1$=5 `c5.9xlarge`, $\mathcal{N}_2$=10 of `c5.4xlarge`, and $\mathcal{N}_3$=5 of `r5.4xlarge` instance types (see Table I). The `c5a.2xlarge` (Frankfurt) coordinator transmits a maximum of two segments per second for the first two scenes, and fourteen segments per second for the third scene. Thus, we have $\left\lceil \frac{\mathcal{N}_1}{\mathcal{N}_{\zeta 1}} \right\rceil$=3, $\left\lceil \frac{\mathcal{N}_2}{\mathcal{N}_{\zeta 2}} \right\rceil$=5, and $\left\lceil \frac{\mathcal{N}_3}{\mathcal{N}_{\zeta 3}} \right\rceil$=1 homogeneous clusters providing the possibility of six, ten, and fourteen encoding pipelines. The number of instances restrains the number of encoding pipelines to five, ten, and five for each respective scene, which needs five, two, and three stages to complete the segments' encoding (the number of stages depends on the number of segments per scene). Finally, MPEC2 estimates 34.37 s for the completion time and 2.79 ¢ for the total cost of video segment encoding.

2) Cost-optimized scenario: We study the cost-optimized scenario similar to the time-optimized one.

a) Segment encoding scheduling: selects the low-cost `medium` instance for all the scenes, with the corresponding completion times displayed in Table IV. Although the `small` instance has the lowest cost (see Table I), it increases the segment encoding time and generates a higher cost.

b) Scene encoding scheduling: depicted in Figure 6b allocates the segments in an encoding pipeline on ten `medium` (Frankfurt) instances (see Table I). The `c5a.2xlarge` (Frankfurt) coordinator transmits a maximum of twelve segments per second for the first scene, eight for the second, and seventeen for the third, due to different segment sizes. Thus, we have $\left\lceil \frac{\mathcal{N}_1}{\mathcal{N}_{\zeta 1}} \right\rceil$=1, $\left\lceil \frac{\mathcal{N}_2}{\mathcal{N}_{\zeta 2}} \right\rceil$=2, and $\left\lceil \frac{\mathcal{N}_3}{\mathcal{N}_{\zeta 3}} \right\rceil$=1 homogeneous clusters providing the possibility of twelve, sixteen, and seventeen encoding pipelines. The number of instances restrains the number of encoding pipelines for all scenes, which need three, two, and again two stages to complete segment encoding. Although selecting the same `medium`

TABLE IV: Segment encoding results for scene scheduling.

Scene	*Time-optimized*					*Cost-optimized*				
	Instance type (Frankfurt)	*Completion time* (s)	*Cost* (¢)	*Accuracy*		*Instance type (Frankfurt)*	*Completion time* (s)	*Cost* (¢)	*Accuracy*	
				Time	*Cost*				*Time*	*Cost*
V_1	c5.9xlarge	6.51	0.29	92 %	93.5 %	medium	52.28	0.061	95 %	96.8 %
V_2	c5.4xlarge	15.38	0.32			medium	146.92	0.121		
V_3	r5.4xlarge	4.29	0.14			medium	24.38	0.041		

(Frankfurt) instance type for all video scenes increases the stream encoding completion time (494.2 s), it reduces the total cost (0.61 ¢).

3) Accuracy analysis: Table IV compares the MPEC2 completion time and cost estimations with the actual encoding benchmarks. Using the middle segment to estimate the scene encoding time produced an accuracy of 92 % in completion time and 93.5 % in cost for the time-optimized scenario. In the cost-optimized scenario, MPEC2 obtains an even better accuracy of 95 % for the completion time and 96.8 % for the total cost of video encoding.

C. Related work comparison

In this section, we compare MPEC2 with four related works, evaluated in time- and cost-optimized scenarios.

1) *MAPO* [18] provides the instance selections to the multi-objective optimization algorithm that searches for a Pareto set of tradeoff solutions considering completion time and cost objectives. MAPO utilizes non-dominated sorting genetic algorithm (NSGA-II) [36] to search for a set of instance types. Then, the method selects one instance type for encoding the video stream.

2) *Random* selects one arbitrary instance type per scene to encode the video stream;

3) *Fastest* selects the fastest instance type relying on the CPU speed (see Table I) to encode the video stream.

4) *Cheapest* selects the lowest cost instance type (see Table I) to encode the video stream.

a) Time-optimized scenario: Table V shows that MPEC2 reduces the completion time of the video stream encoding by 24 % compared to MAPO, which does not differentiate the video scenes and selects only one instance type c5.4xlarge for all scenes. In contrast, MPEC2 selects c5.9xlarge for the scene V_1, c5.4xlarge for the V_2, and r5.4xlarge for the V_3. MPEC2 further improves the completion time on average by 54 % compared to the random method that generates a load imbalance when allocating scenes to slow instances [5]. The fastest method schedules the encoding tasks on five c5.9xlarge instances with the highest CPU speed of 90 700 MIPS (see Table I). In contrast, MPEC2 selects different instance types and encodes the video scenes in a pipeline model reducing the completion time by 40 %.

b) Cost-optimized scenario: Table V shows that MPEC2 improves the video stream encoding cost by 60 % compared to MAPO, which selects just one instance type that is not the lowest cost one. MPEC2 reduces on average total cost by 50 % compared to the random method, which may select a costly instance type. Finally, the cheapest method does select the instance type small with the lowest cost for encoding all the scenes, increases the completion time that imposes a higher total cost. Therefore, the cheapest method performs 5 % worse in terms of total cost than MPEC2.

TABLE V: Comparative MPEC2 results in time and cost-optimized scenarios.

Method	*Time-optimized*		*Cost-optimized*	
	Improvement	*Instance type (Frankfurt)*	Improvement	*Instance type (Frankfurt)*
MPEC2	0%	c5.9xlarge (scene V_1) c5.4xlarge (scene V_2) r5.4xlarge (scene V_3)	0%	medium
MAPO	24 %	c5.4xlarge	60 %	c5.4xlarge
Random	54 %	Random instance type	50 %	Random instance type
Fastest	40 %	c5.9xlarge	–	–
Cheapest	–	–	5 %	small

VIII. Conclusion

We introduced MPEC2, a new scheduling method for video encoding application on the computing continuum. MPEC2 uses a multilayer graph partitioning model [7] to find an appropriate instance type based on its location to encode one video segment. Afterward, it proposes a pipeline model [8] to distribute the encoding of all the video scenes' segments on the selected instance types. We evaluated MPEC2 in a realistic Cloud and Fog testbed distributed across seven geographical locations outperforming related methods in terms of completion time and total cost of encoding a video stream. MPEC2 achieved 24 %, 54 %, and 40 % faster video encoding compared to *MAPO*, *random*, and *fastest* instance selection methods. Moreover, it lowered the total encoding cost by 60 %, 50 %, and 5 % compared to the *MAPO*, *random*, and *cheapest* methods.

Acknowledgment

This work received support from:

- Austrian Research Promotion Agency (FFG) under grant agreement 877503 (APOLLO) and grant agreement FO999897846 (GAIA);
- *European Union*'s Horizon 2020 research and innovation program, grant agreement 101016835 (DataCloud).

References

[1] Sandvine. The global internet phenomena report. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2021/Phenomena/MIPR%20Q1%202021%2020210510.pdf, 2022. [Online; accessed Oct.-2022].

[2] Csmsu graphics and medialab video group video transcoding clouds comparison 2019. http://compression.ru/video/codec_comparison/cloud_2019/, November 2019. [Online; accessed Oct.-2022].

[3] Ahmed Ali-Eldin, Bin Wang, and Prashant Shenoy. The hidden cost of the edge: a performance comparison of edge and cloud latencies. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2021.

[4] Behrouz Jedari, Gopika Premsankar, Gazi Illahi, Mario Di Francesco, Abbas Mehrabi, and Antti Ylä-Jääski. Video caching, analytics, and delivery at the wireless edge: a survey and future directions. *IEEE Communications Surveys & Tutorials*, 23(1):431–471, 2020.

[5] Anatoliy Zabrovskiy, Prateek Agrawal, Vladislav Kashansky, Roland Kersche, Christian Timmerer, and Radu Prodan. FSpot: fast and efficient video encoding workloads over amazon spot instances. *Computers, Materials and Continua*, 71(3):5677–5697, 2022.

[6] Juan Gutiérrez-Aguado, Raúl Peña-Ortiz, Miguel García-Pineda, and Jose M. Claver. Cloud-based elastic architecture for distributed video encoding: evaluating h.265, vp9, and av1. *Journal of Network and Computer Applications*, 171:102782, 2020.

[7] Zahra Najafabadi Samani, Nishant Saurabh, and Radu Prodan. Multilayer resource-aware partitioning for fog application placement. In *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, pages 9–18. IEEE, 2021.

[8] Tarek Elgamal, Atul Sandur, Phuong Nguyen, Klara Nahrstedt, and Gul Agha. DROPLET: distributed operator placement for iot applications spanning edge and cloud resources. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 1–8. IEEE, 2018.

[9] Amazon Web Services. AWS ec2. https://aws.amazon.com/ec2/. [Online; accessed Oct.-2022].

[10] Exoscale Corporation. European cloud hosting. https://www.exoscale.com/compute/. [Online; accessed Oct.-2022].

[11] Changchun Long, Yang Cao, Tao Jiang, and Qian Zhang. Edge computing framework for cooperative video processing in multimedia iot systems. *IEEE Transactions on Multimedia*, 20(5):1126–1139, 2017.

[12] Atakan Aral, Ivona Brandic, Rafael Brundo Uriarte, Rocco De Nicola, and Vincenzo Scoca. Addressing application latency requirements through edge scheduling. *Journal of Grid Computing*, pages 1–22, 2019.

[13] Muhammad Ali, Ashiq Anjum, Omer Rana, Ali Reza Zamani, Daniel Balouek-Thomert, and Manish Parashar. Res: Real-time video stream analytics using edge enhanced clouds. *IEEE Transactions on Cloud Computing*, 2020.

[14] Olivier Barais, Johann Bourcier, Yérom-David Bromberg, and Christophe Dion. Towards microservices architecture to transcode videos in the large at low costs. In *2016 International Conference on Telecommunications and Multimedia (TEMU)*, pages 1–6. IEEE, 2016.

[15] Abbas Mehrabi, Matti Siekkinen, and Antti Ylä-Jääski. Edge computing assisted adaptive mobile video streaming. *IEEE Transactions on Mobile Computing*, 18(4):787–800, 2018.

[16] Vaughan Veillon, Chavit Denninnart, and Mohsen Amini Salehi. F-FDN: federation of fog computing systems for low latency video streaming. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–9. IEEE, 2019.

[17] Alireza Erfanian, Hadi Amirpour, Farzad Tashtarian, Christian Timmerer, and Hermann Hellwagner. LwTE: light-weight transcoding at the edge. *IEEE Access*, 9:112276–112289, 2021.

[18] Narges Mehran, Dragi Kimovski, and Radu Prodan. MAPO: a multi-objective model for iot application placement in a fog environment. In *Proceedings of the 9th International Conference on the Internet of Things*, pages 1–8, 2019.

[19] Susanna Schwarzmann, Nick Hainke, Thomas Zinner, Christian Sieber, Werner Robitza, and Alexander Raake. Comparing fixed and variable segment durations for adaptive video streaming: A holistic analysis. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 38–53, 2020.

[20] Anatoliy Zabrovskiy, Prateek Agrawal, Christian Timmerer, and Radu Prodan. FAUST: fast per-scene encoding using entropy-based scene detection and machine learning. In *2021 30th Conference of Open Innovations Association FRUCT*, pages 292–302. IEEE, 2021.

[21] Juan J Durillo and Radu Prodan. Multi-objective workflow scheduling in amazon ec2. *Cluster computing*, 17(2):169–189, 2014.

[22] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[23] iperf - the ultimate speed test tool for tcp, udp and sctp. https://iperf.fr/. [Online; accessed Oct.-2022].

[24] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. Cloud, fog or edge: where to compute? *IEEE Internet Computing*, 2021.

[25] Exoscale Corporation. European cloud cost calculator. https://www.exoscale.com/calculator/. [Online; accessed Oct.-2022].

[26] Amazon Web Services. AWS cost calculator. https://calculator.aws/#/createCalculator. [Online; accessed Oct.-2022].

[27] Amazon Web Services. Amazon EC2 cost calculator. https://calculator.aws/#/createCalculator/EC2. [Online; accessed Oct.-2022].

[28] Amazon Web Services. Amazon EBS cost calculator. https://calculator.aws/#/addService/EBS. [Online; accessed Oct.-2022].

[29] Exoscale Corporation. European cloud storage cost calculator. https://www.exoscale.com/pricing/#storage-optimized-instances. [Online; accessed Oct.-2022].

[30] Amazon Web Services. AWS data transfer cost calculator. https://calculator.aws/#/addService/DataTransfer. [Online; accessed Oct.-2022].

[31] Exoscale Corporation. European cloud outbound cost calculator. https://www.exoscale.com/pricing/#additional-features. [Online; accessed Oct.-2022].

[32] Hadi Amirpour, Vignesh V Menon, Samira Afzal, Mohammad Ghanbari, and Christian Timmerer. VCD: video complexity dataset. In *Proceedings of the 13th ACM Multimedia Systems Conference*, pages 234–239, 2022.

[33] Institue of Information Technology. Video complexity datasets. https://ftp.itec.aau.at/datasets/video-complexity/. [Online; accessed Oct.-2022].

[34] Roland Mathá, Dragi Kimovski, Anatoliy Zabrovskiy, Christian Timmerer, and Radu Prodan. Where to encode: a performance analysis of x86 and arm-based amazon ec2 instances. In *17th IEEE International Conference on eScience*, 2021.

[35] Prateek Agrawal, Anatoliy Zabrovskiy, Adithyan Ilangovan, Christian Timmerer, and Radu Prodan. FastTTPS: fast approach for video transcoding time prediction and scheduling for http adaptive streaming videos. *Cluster Computing*, 24(3):1605–1621, 2021.

[36] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.