[1]Miguel Garcia-Pineda

# Towards GPU-enabled Serverless Cloud Edge Platforms for Accelerating HEVC Video Coding

Andoni Salcedo-Navarro[1], Raúl Peña-Ortiz[1], José M. Claver[1], *[1], and Juan Gutiérrez-Aguado[1]

[1]Departament d'Informàtica, Universitat de València, Avda. de la Universitat, s/n, Burjassot, 4610, Valencia, Spain

**Abstract**

Multimedia streaming has become integral to modern living, reshaping entertainment consumption, information access, and global engagement. The ascent of cloud computing, particularly serverless architectures, plays a pivotal role in this transformation, offering dynamic resource allocation, parallel execution, and automatic scaling—especially beneficial in HTTP Adaptive Streaming (HAS) applications. This study presents an event-driven serverless cloud edge platform with graphics processing units (GPUs), managed by Knative, tailored for video encoding. Two implementations of the High Efficiency Video Coding (HEVC) codec have been encapsulated in the functions: HEVC NVENC, that uses GPU acceleration, and x265 that only uses CPUs. Experiments focused on measuring the impact of replica requested resources on cold start, scalability and resource consumption with different allocated resources on slim and fat virtual machines (VMs). The best results are obtained when four slim replicas of the functions are deployed on a fat VM with a 8,4% reduction in encoding time for x265 and a 15,2% improvement for HEVC NVENC compared with other deployment scenarios. Comparatively, HEVC NVENC encoding is 8,3 times faster than x265. In multiresolution scenarios, GPU encoding drastically reduces segment encoding time by a factor of 12,4 between non-GPU and GPU-accelerated. These findings are important for live streaming applications where low latency is critical at all stages of the streaming process.

**Keywords**: Serverless, FaaS, GPU, Video encoding, Cloud edge

## 1 Introduction

In the digital age, multimedia streaming has become an integral part of our daily lives, fundamentally transforming the way we consume and share content. This technology allows

us to access a wide range of audio and video content, from music and movies to live sports events and educational materials, either on-demand or live. The importance of multimedia streaming in today's world cannot be overstated, as it has revolutionised the entertainment industry, education, communication, and business in countless ways [1]. For example, one of the most popular forms of multimedia entertainment today is through online streaming platforms becoming serious rivals to the broadcasting and television industry. The demand for regular television broadcasting is declining as these platforms provide high-quality original productions at the convenience of the customer's home. The shift from linear TV to on-demand viewing, the rise in binge-watching, and the increase in diversity of content due to original programming are some of the changes brought about by streaming services.

In terms of streaming technology, HTTP Adaptive Streaming (HAS) has become the dominant video delivery technology over the Internet. In HAS, videos are split into short intervals called segments, and each segment is encoded at various qualities/bitrates to adapt to the available bandwidth. These streaming systems necessitate efficient video encoding systems capable of generating content in a multitude of resolutions, bit rates, and codecs, with the aim of providing an optimal Quality of Experience (QoE) to end-users [2].

One of the key factors that has facilitated this transformation is the seamless integration of multimedia streaming with cloud computing [3]. Cloud computing has emerged as a game-changer for multimedia streaming, serving as a robust and scalable infrastructure for delivering content to users across the globe [4]. This technology has allowed individuals and organisations to exploit the power of the cloud to store, process, and deliver multimedia content efficiently and cost-effectively by harnessing:

- Scalability: The demand for multimedia content can vary dramatically, with spikes during popular events or viral content releases. Cloud computing enables content providers to dynamically scale their infrastructure up or down to accommodate these fluctuations in demand.
- Global Reach: Cloud-based multimedia streaming services can easily reach a worldwide audience by leveraging the distributed nature of cloud data centers. This allows content providers to minimise latency and deliver high-quality streaming experiences to users in various regions.
- Cost Efficiency: The cloud computing model offers cost benefits to multimedia streaming providers. Instead of investing in and maintaining their own hardware, they can rent resources on a pay-as-you-go basis, reducing capital expenditures and optimising operational costs.
- Flexibility: Cloud-based multimedia streaming services provide the flexibility to adapt to evolving industry standards and user preferences. Providers can quickly update their services, add new features, or support various devices without significant disruptions.

- Data Security: Cloud computing providers often invest heavily in security measures to protect data, making them a reliable choice for storing and transmitting multimedia content securely. This is particularly important in the context of copyrighted materials and user data protection.
- Content Delivery Networks (CDNs): Many multimedia streaming services leverage CDNs, which are a distributed network of servers in various data centers. These CDNs are often managed by cloud providers and are strategically positioned to cache and deliver content as close to the end-users as possible, further enhancing the streaming experience.
- Big Data Analytics: Cloud computing enables content providers to analyse user data and preferences, which can be used to personalise content recommendations and improve the overall user experience.

Within the various paradigms encompassed by the Cloud Computing concept, like Infrastructure-as-a- Service (IaaS), Platform-as-a-Service (PaaS), Software-as-a-Service (SaaS), etc., serverless computing is a cloud computing paradigm that shifts the responsibility of managing servers from the user to the cloud provider. In Function-as-a-Service (FaaS), one subset in the serverless ecosystem, users only need to focus on writing and deploying individual functions or units of code. The cloud provider dynamically manages the execution environment and automatically scales resources up or down based on the actual demand [5]. The event-driven model of FaaS, coupled with its statelessness and automatic scaling, distinguishes it from traditional cloud computing models. Each function execution is isolated and stateless, allowing for efficient resource management.

Cost efficiency stands out as a primary benefit of serverless computing, thanks to its pay-as-you-go model. Users are charged solely for the resources consumed during function execution, eliminating costs associated with idle time [6]. Automatic scaling ensures optimal resource utilization, making serverless computing a cost-effective choice, especially for unpredictable workloads. Reduced operational complexity is another significant advantage, as serverless computing abstracts infrastructure management tasks. Cloud providers handle server provisioning, maintenance, and scaling, enabling developers to concentrate on code development rather than infrastructure upkeep. Developers can efficiently deploy and update functions without managing the intricacies of server infrastructure, leading to accelerated development cycles and quicker time-to-market for applications and features.

Moreover, serverless computing supports microservices architecture, enabling developers to build and deploy independent functions. This aligns seamlessly with a modular and scalable application design, contributing to the overall flexibility and adaptability of the architecture.

A promising application of serverless architectures is found in HAS-based video coding systems designed for video streaming (VoD and live). In these scenarios, the original video

is segmented typically into short chunks lasting 1, 2, 4 seconds that are encoded for various bitrates and/or resolutions. For every encoding or set of encodings per chunk, a serverless platform can execute a code fragment encapsulated in a function, facilitating dynamic resource allocation. The serverless infrastructure enables parallel execution of encoding for each chunk. Moreover, it offers automatic scaling of replicas running the function to meet increased requirements, thereby potentially reducing the overall encoding time.

GPUs have been used in the video encoding process to improve performance and efficiency [7]. GPUs excel in handling intensive video tasks like encoding, thanks to their parallel processing. This means faster video processing, crucial for real-time streaming and broadcasting. GPUs' adaptability, support for multiple streams, and hardware-accelerated codecs make them vital for top-notch video encoding results.

For these reasons, our work presents and analyses a GPU video coding system built upon the serverless architecture paradigm characterised by streamlined automated or semi-automated deployment processes. Our proposal leverages the Knative open-source serverless platform [8] deployed on a resource-constrained infrastructure. One of the key differences between the cloud edge and the cloud core is that in edge scenarios the nodes possess significantly fewer capabilities compared to powerful servers located in distant data centers. In addition, the number of available nodes is much lower in the edge. Therefore, with the use of the FaaS model in the cloud edge we are not taking advantage of the scaling capabilities but we are using the event management capabilities and concurrency control offered by Knative. The implemented system underwent testing across various scenarios to analyse the performance and resource consumption when encoding videos under real-world conditions, such as the dynamic adaptive streaming over HTTP (DASH) environment. To the best of our knowledge, there is a lack of studies that analyse the behaviour of encoders in serverless platforms with replicas of the function that make use of GPUs.

The main contributions of this work are:

- Proposal of a GPU serverless deployment for video encoding: architectural needs, configuration required, and development of serverless functions.
- Evaluation of different virtualisation scenarios to deploy serverless video encoding with and without the use of GPUs.
- Comparison of cold starts based on the computational resources requested by the function's replicas.
- Analysis of the behaviour of replicas under different scenarios and contrasting resource consumption and time taken for encoding a video.
- Analysis of the performance when multiple representations of each video chunk are generated in a single call.

This paper is organised as follows. Section 2 presents previous approaches related to the topics of this work. Section 3 presents the main components of the deployed serverless infrastructure. Section 4 describes the testbed, the six scenarios considered (with and without GPU), and the results obtained for cold start, encoding times and resource consumption. Then the best scenario is selected to analyse multiresolution encoding with a single event. Section 5 performs a comparison of this proposal with other previous works considering 12 aspects. Finally, Section 6 presents the conclusions and further work.

## 2 Related work

Proposed cloud-based solutions for distributed video coding and transcoding services have recently evolved to be adapted to the more demanding requirements and improve their performance [9].

The need of dynamic instantiation of the number of encoders has been crucial for the new video coding services, characterized by high demand and adaptive streaming applications. For this purpose, new cloud-based solutions adapting the number of encoders to the needs of video streaming, such as high video resolutions with bandwidth constraints and quality requirements have been proposed. Thus, Gutiérrez-Aguado *et al.* [10] proposed a cloud-based distributed architecture tuned for video encoding (HEVC, VP9 and AV1 encoders). Based on an elastic pool of workers and media servers that provide fault tolerance, this solution allows the adaptation of the resources to the workload, offers high availability, and provides ubiquitous access.

Enabling GPUs in cloud-based architectures have been proposed in different works. Some examples of using GPUs in cloud environments can be seen in Yang *et al.* [11], which use PCI pass-through technology to access the NVIDIA graphics card and show that the GPU performance is similar in native and virtual machines. The next advancement in the use of GPUs in cloud environments can be seen in Gutiérrez-Aguado *et al.* [12], which introduce a new architecture to cloudify existing GPUs, which do not support native virtualization.

Video coding has also benefited from the use of GPUs to speed up the increasing complexity and high time cost of new encoders. Thus, the use of GPUs has been proposed to accelerate video coding in standards such as H.264/AVC [13] , HEVC [14], VP9 [15] and others. Moreover, Comi *et al.* [16] proposed using GPUs for video coding in cloud environments in the form of Network Function Virtualization (NFV), which facilitate the management and orchestration of network services. Thus, the GPU acceleration combined with NFV enhance the computing performance of commodity servers with the advantages offered by NFV in service management.

Nevertheless, these cloud-based solutions use virtual machines (VMs) to deploy and

destroy the dynamic encoding workers used for the distributed video encoding, with a non-negligible time for this purpose. Linux containers, and specially Docker containers, as a lightweight process, have been an alternative to VMs, reducing the deploying, and obtaining higher efficiency than with traditional VMs [17].

Following this approach, several architectures of containerized multimedia processing system for video transcoding based on Docker have been proposed (see subsection 4.3. in [9]).But the use of containers adds complexity due to orchestration and programming, hence container orchestration platforms (COPs) such as Docker Swarm, Nomad or Kubernetes [18] are used in cloud computing environments, being the latter the currently standard COP.

In the last decade, serverless computing has witnessed a significant upsurge primarily due to the cloud provider's management of dynamically allocated computing resources, coupled with a granular pay-per-use billing approach. It makes easier to write robust, large-scale web services [19], opening a new era in cloud computing. Amazon Web Services introduced AWS Lambda, the first serverless platform, on 2014, and similar abstractions are now available on most cloud computing platforms [20]. Moreover, several frameworks adopting the serverless approach have emerged to present open-source options as alternatives to what public cloud providers offer. Among these, notable platforms include OpenFaaS [21], Knative [22] , Apache OpenWhisk [23], IronFunctions [24], among others. Furthermore, using on-premise serverless infrastructures as OSCAR [25], which is based on OpenFaaS, combined with the use of public cloud serverless as AWS Lambda, has advantages. But in some cases, as in edge computing, the on-premises solution can be better when a minimalist Kubernetes distribution and a fast even-driven functionality are required.

There are few, but interesting works, where video coding systems have been developed adopting the serverless approach. Fouladi *et al.* [26] present a video encoder (for VP8) aimed at fine-grained parallelism using a functional programming style that achieves fine-grained parallelism based on AWS Lambda and the Mu framework [27]. Sprocket [28] is a serverless-based video processing framework that extends the Mu framework to support dynamic creation of tasks during processing. It enables developers to program a series of operations over video content in a modular and extensible manner. More recently, Moina-Rivera *et al.* [29] presents an event-driven serverless functions to encode video (for H.264 and AV1). The authors deployed event-driven serverless pipelines for video coding and quality metrics (VMAF) on a local serverless platform based on Knative.

As we have seen in this section, and to the best of our knowledge, there are few works that use GPUs in serverless frameworks [30] and there is a lack of works that provide architectures and analysis of GPU integration in serverless environments and none for video encoding. For this reason, in the following sections, we will present our proposal and evaluate it to validate its correct operation. In section 5, our proposal is compared with previous works related to video encoding on serverless platforms to evaluate their differences

and similarities.

# 3 Proposed architecture and serverless functions

This section presents the essential features of the Knative serverless platform, a pivotal element in our design strategy. We describe the platform's architecture and core components and the implementation of the event-driven serverless functions.

## 3.1 Serverless Platform: Knative

Among the various open-source serverless platforms, Knative stands out for its unique capabilities, as highlighted in []li2021analyzing. A standout feature of Knative is its ability to dynamically scale replicas based on concurrency, which aligns perfectly with our application's requirement of limiting each function replica to a single concurrent execution due to resource constraints. Knative further excels with its support for cold/warm starts, zero-scaling, and compatibility with CloudEvents [46], making it the ideal choice for our needs.

CloudEvents is a framework for streamlining event description and delivery across different services and platforms. This specification, under the Cloud Native Computing Foundation, is widely adopted by major cloud and SaaS providers. It details how to encapsulate CloudEvents in various protocols like AMQP, HTTP, Kafka, MQTT, websockets, Protobuf, etc. In our setup, CloudEvents are embedded in HTTP messages, with metadata in HTTP headers and event data in JSON format in the HTTP message body.

To invoke event-driven functions, Knative uses CloudEvents. Knative offer two elements that allow the routing of a CloudEvent from a source of events to the consumers of the events. The first element is the broker that receives the CloudEvents from the source. The second element is the trigger that is bound to a broker and specifies filters that the CloudEvent must satisfy in order to deliver the event to a consumer. This process is shown in Figure 1. These consumers can be standard Kubernetes resources or Knative services. The latter option offers automatic scalability and concurrency control.

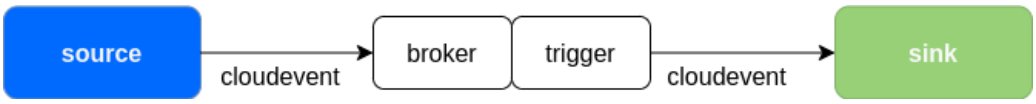



Figure 1: A source sends a CloudEvent to the broker, then the trigger sends the event to a sink depending on the attributes of the event

A network layer is necessary for installing Knative, with options like Kourier, Istio, or Contour. We chose Contour, as it recognizes Knative properties governing function time-

outs, set in the ConfigMap config-defaults in the knative-serving namespace. We adjusted these default values for longer function execution as shown in Listing 1.

**Listing 1** Modified timeout properties used to set timeouts of functions

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-defaults
  namespace: knative-serving
data:
  revision-timeout-seconds: "3600"
  max-revision-timeout-seconds: "7200"
  scale-to-zero-grace-period: "15s"
```

The deployed infrastructure, shown in Figure 2, consists of Knative deployed on a Kubernetes cluster and various serverless event-driven functions. Video chunks are stored on a download server (Nginx container), and a private registry is used for container images, accessible to Kubernetes through credentials. Processed video chunks are uploaded to a server running an upload service (developed in Tomcat). To encode a video, a CloudEvent is generated for each chunk and sent to the network layer gateway, then delivered to a function replica.

### 3.2 Function Implementation

Knative imposes no restrictions on function implementation. We used Java, based on our prior work, to develop the function with frameworks like Tomcat, Quarkus, Spring, or Micronaut, each capable of exposing an HTTP endpoint.

The functions are implemented using a modified version of embedded Tomcat, which offers CloudEvents abstractions [47]. This leads to smaller images due to fewer dependencies. To develop a event-driven function, the `CloudEventListener` abstract class must be extended to implement the abstract method `consumeEvent(...)` that receives the unmarshalled `CloudEvent` object and the two standard objects `HttpServletRequest` and `HttpServletResponse`. Using this application runtime we developed two serverless functions:

- Encoding video with GPU supporting HEVC using Nvidia NVENC codec, and
- Encoding video without GPU supporting libx265 codec.

These functions, being ephemeral, download, process, and upload video chunks for each invocation. The functions are deployed to respond to HTTP requests that encapsulate

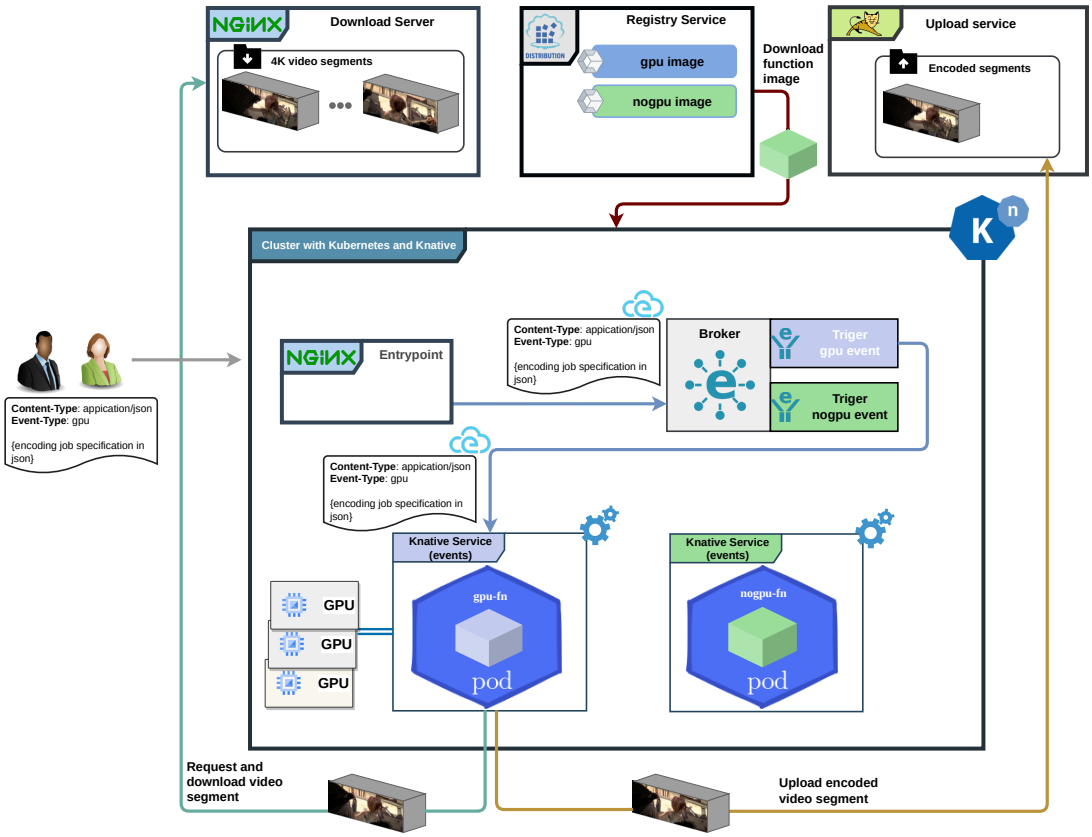



Figure 2: A cluster with Kubernetes and Knative where functions are deployed. A download server with the videos, an upload server to upload the results, and a container image registry are connected to the cluster. The kubernetes cluster is configured to have access to GPU resources and the function replicas can request GPU acceleration.

CloudEvents. For instance, a sample call to activate the functions via the broker through the Envoy gateway is detailed in Listing 2. In this call we can see the parameters passed to the `ffmpeg` process and the information about the data being downloaded and uploaded.

## 4 Experiments and results

This section describes the testbed used for the experiments and the different scenarios considered: three to be used with the codecs that do no utilise the GPU and other three with the codecs that do use the GPU. Then we analyse the impact of the requested replica resources in the cold start time. The resource consumption in terms of RAM, CPU and

**Listing 2**

```
curl -s -X POST KNATIVE_GATEWAY_URL \
    -H "Ce-createdtime: 1699607955259" \
    -H 'Content-Type: application/json' \
    -H 'Ce-Type: encoder' \
    -H 'Ce-Specversion: 1.0' \
    -H 'Ce-Source: /HttpEventSource' \
    -H "Ce-Id: 1" \
    -d "{
      "ffmpegParams": "-i chunk1.mp4 -c:v libx265 -b:v 11M
        -maxrate 11M -minrate 11M -preset medium echunk1.mp4",
      "datamesh": {
        "downloadUrl": "DOWNLOAD_SERVER_URL",
        "downloadFiles": [chunk1.mp4],
        "uploadFiles": [echunk1.mp4],
        "uploadUrl": "UPLOAD_SERVER_URL",
        "timesFile": "times.json"
    }
    }"
```

GPU usage is measured in all scenarios and codecs. Finally, the behaviour when encoding multiple representations in a single call is analysed.

### 4.1 Testbed and videos

The testbed used for the experiment consists of five physical machines interconnected via a 10 Gb/s switch. Three of them are commodity servers: one of the servers acts as a private registry that stores all the function images; other server has a Nginx service that allows to download all the video segments via HTTP; the third one is an upload service over HTTP, designed to store the encoded videos.

The Kubernetes cluster is deployed on the other two servers. One of them allocates the Kubernetes master node in a virtual machine with 16 GB RAM and 10 vCPUs. The other server has 128 GB RAM, 12 cores and 4 Nvidia Quadro M4000 GPUs, with 8 GB RAM each, that will allocate the Kubernetes worker VMs.

The virtual infrastructure has been designed carefully taking into account the performance. The SR-IOV NICs have been configured to expose virtual functions that are attached to the virtual machines. Besides, the GPUs will be assigned to the VMs using PCI passthrough. Finally, each VM will have a dedicated SSD to ensure fast data access. For Kubernetes, Docker was selected as the Container Runtime Interface (CRI) and Flannel as the Container Network Interface (CNI). Besides, we have deployed the Nvidia Operator (version v23.3.1) which facilitates a dynamic reconfiguration of the assignation of GPUs to VMs. The serverless platform used in this work is Knative (version 1.4.0).

The experimental phase employs two specific images. The `ffmpeg-fn-acc` image, with

604 MB of size, is built with Java, FFmpeg, CUDA GPU support, and incorporates all necessary Nvidia drivers. This image supports GPU acceleration and the codec `hevc_nvenc`. In contrast, the `ffmpeg-fn-no-acc` image, with 120 MB of size, is built with Java and FFmpeg, does not have GPU support but can handle codec `libx265`.

The 4K videos[1] used in the experiments are listed in Table 1. The videos have been downloaded, transcoded using x264 at very high quality (10 CRF), and finally split to obtain segments of 2 seconds. Table 1 also shows the final sizes after that process of transcoding and splitting.

Table 1: Videos used to perform the experiments.

| Video Name | Acronym | Resolution | Duration (seconds) | chunks | Size (GB) |
|---|---|---|---|---|---|
| Ancient Thought | ANC | 4K | 188 | 94 | 2,9 |
| Eldorado | ELD | 4K | 184 | 92 | 4,7 |
| Indoor Soccer | IND | 4K | 252 | 126 | 8,0 |
| Skateboarding | SKA | 4K | 276 | 138 | 7,7 |

## 4.2 Scenarios without and with GPUs

We have tested the performance on six different scenarios. In three of them, the replicas do not require GPU and use `libx265` to encode the segments. In the other three, the replicas request GPU and the encoder used is `hevc_nvenc`. Both encoders are different implementations of the same standard H.265/HEVC[2]. Meanwhile `hevc_nvenc` leverage GPU integration `libx265` only use CPUs

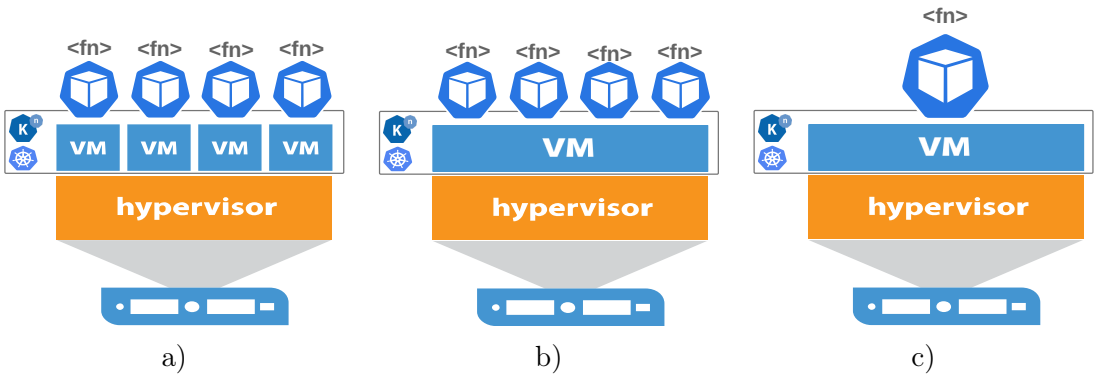
In all the scenarios, the computational resources assigned to each experiment are the same: 8 vCPUs and 96 GiB RAM. When GPUs are considered, the total number of GPUs assigned will be the same. However, these computational resources will be distributed differently in each scenario. The goal is to determine what scenario is the best (given these computational resources) in terms of video encoding time. The constraints encountered stem from a deficiency in resources, attributed to the scarcity of financial allocations.

### 4.2.1 Scenarios without GPU

Figure 3 depicts scenarios where GPU acceleration is not used for video encoding tasks. The three scenarios are:

**Scenario 1: four slim VMs with four slim replicas.** This configuration has four slim

[1] 4K videos available at: https://www.cablelabs.com/4k
[2] ITU Recommendation, available at: https://www.itu.int/rec/T-REC-H.265-202309-I/en




a) b) c)

Figure 3: Scenarios of replicas without GPUs: a) slim VMs with slim replicas, b) fat VM with slim replicas, and c) fat VM with fat replica.

worker nodes with 2 vCPUs and 24 GiB RAM. On each VM, a slim replica, that requests 2 vCPUs and 24 GiB RAM, has been provisioned, see Figure 3-a. In this scenario, the load is evenly distributed among the lightweight VMs and their corresponding lightweight replicas. Each replica is allowed to run only one concurrent encoding job.

**Scenario 2: one fat VM with four slim replicas.** This scenario employs a single fat VM with 8 vCPUs and 96 GiB RAM. This single VM allocates four slim replicas that are limited to 2 vCPUs and 24 GiB RAM, see Figure 3-b. Comparing this scenario with the previous one, the total amount of computational resources are the same, but their distribution is different. Each replica is allowed to run only one concurrent encoding job.

**Scenario 3: one fat VM with one fat replica.** This setup also uses a single fat VM with 8 vCPUs and 96 GiB of memory. However, instead of deploying four slim functions, we have a single fat function deployed on it, see Figure 3-c. The replica is allowed to run up to four concurrent encoding jobs.

#### 4.2.2 Scenarios with GPU

In this subsection, the scenarios described have access to GPUs to speedup the video encoding process. The four GPUs will be assigned in all the scenarios depicted in Figure 4 and described below.

**Scenario 4: four slim VMs with four GPU-assisted slim replicas.** This scenario uses four slim VMs, with 2 vCPUs and 24 GiB RAM each. Besides, each VM has a dedicated GPU. The replicas in this scenario will request 2 vCPUs and 24 GiB RAM and

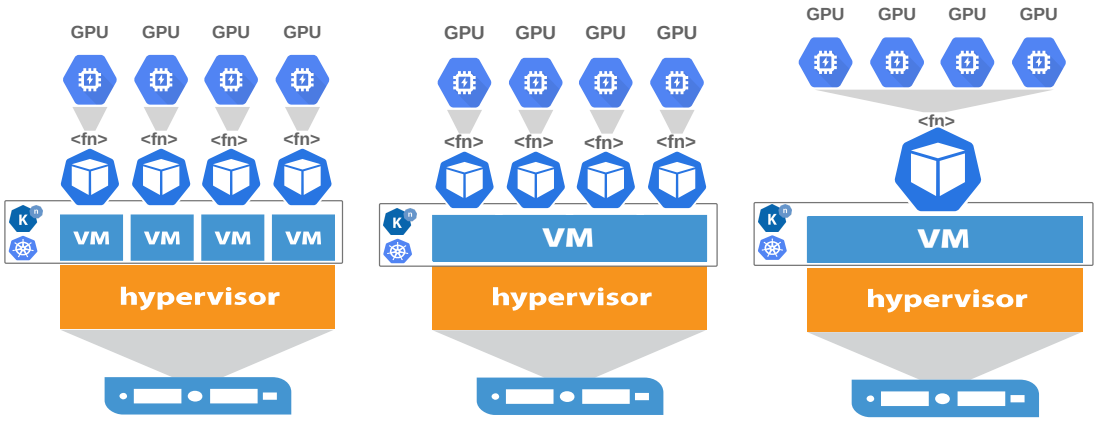



Figure 4: Scenarios with GPUs: a) slim VMs with slim replicas, b) fat VM with slim replicas, and c) fat VM with fat replica.

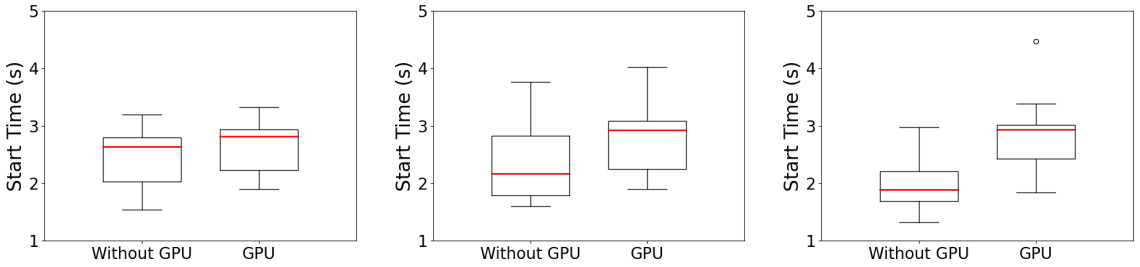
will have access to the GPU of the underlying VM, see Figure 4-a. Each replica is allowed to run only one concurrent encoding job.

**Scenario 5: one fat VM with four GPU-assisted slim replicas.** A single fat VM with 8 vCPUs, 96 GiB of memory, and the four GPUs allocates four replicas of the function. Each replica is limited to use 2 vCPUs, 24 GiB RAM, and one GPU, see Figure 4-b. Each replica is allowed to run only one concurrent encoding job.

**Scenario 6: one fat VM with one GPU-pool fat replica.** In this setup a single fat VM, with 8 vCPUs, 96 GiB of memory, and the four GPUs allocates a single replica of the function. The replica requests 8 vCPUs, 96 GiB RAM, and the four GPUs, see Figure 4-c. This replica is allowed to run up to four concurrent encoding jobs.

### 4.3 Impact of requested function resources on cold start

One of the main features that define serverless platforms is the ability to scale to zero the number of instances running the function if they are idle for a period of time. This is a convenient feature for users as entails a reduction in costs. However, this convenience has an impact in latency. If there are no running instances and the platform receives a request, one instance must be started to deal with the request. The container runtime must download the layers of the images from the registry if they are not in the file system of the node. Then the environment for the pod has to be created, the function must be started and, finally, the request can be delivered to the function. This elapsed time is named **cold start**. In




(a) Cold start in scenario 1 and 4

(b) Cold start in scenario 2 and 5

(c) Cold start in scenario 3 and 6

Figure 5: Cold start times in scenarios

this work, this time will be measured as the time elapsed from the creation of the event to the time when the event is received by the function.

The experiments described in this section are designed to evaluate the impact of the resources requested by the replica and the underlying VM configuration on the cold-start. We have performed 400 invocations in each scenario ensuring that between each invocation the platform has scaled to zero the number of instances.

Figure 5 shows the boxplots of the data obtained for the six scenarios. Each subfigure shows the boxplot of the cold start times when no GPU is requested (left) and the boxplot of the cold start times when the replicas request GPU (right). The pattern observed is that when the replicas request GPUs the required time to have the function running increases when compared with the setup where no GPU are requested. Specifically, the mean overhead is 10,47% in the first scenario, 17,44% in the second, and 36,50% in the third one. Looking at the results of the scenarios with GPUs, we can conclude that there are no big differences among them (the biggest difference in mean is just a 3,92%). On the contrary, for the scenarios without GPU we can observe more differences (up to 20,15%) being the scenario 3 (with only one fat replica) the fastest one.

### 4.4 Video encoding times and replica resource consumption

The video used for all the experiments in this section is SKA with 138 2-second segments, 7,7 GB of total size and a mean size per segment of 55,8 MB.

For each scenario, the total time required to encode all the segments was computed. When collecting this data, no monitoring of resources was performed (as the monitoring itself consumes part of the resources).

The experiment was repeated to collect resource consumption data. In this setup, PodInsights[31], a Kubernetes daemonset resource (that ensures that it will run one copy

at each worker node) has been deployed. This daemonset makes use of the `docker stats` command to recover information of every container running in the worker node, and stores the information in a MongoDB instance running inside Kubernetes.

After each experiment, the collected data is queried and stored in files for an ulterior analysis. It is important to highlight that the `docker stats` command can only be executed each second. For those scenarios where there are GPUs, the command `nvidia-smi` was executed every 200 milliseconds inside each replica to collect GPU usage.

The goal of this section is twofold. On the one hand, we want to compare the gain when using the GPUs to encode in a serverless platform compared with a baseline case, where GPUs are not available. On the other hand, we want to determine what is the best scenario, given the available resources, in terms of total time required to encode.

### 4.4.1 Encoding times and resource consumption in the scenarios without GPUs

As a baseline case, we present the data obtained when the encoding is performed in the scenarios that do not utilise GPUs. Table 2 describes the different scenarios studied regarding the resources allocated to the replicas, the number of worker nodes, and the concurrency (number of simultaneous encoding jobs) allowed in the replicas.

| Name | VMs | Instances | CPUs | RAM | Concurrency |
|---|---|---|---|---|---|
| $<\text{x265}>^{e_1}_{\text{slim}}$ | 4 | 4 | 2 | 24 | 1 |
| $<\text{x265}>^{e_2}_{\text{slim}}$ | 1 | 4 | 2 | 24 | 1 |
| $<\text{x265}>^{e_3}_{\text{fat}}$ | 1 | 1 | 8 | 96 | 4 |

Table 2: Different scenarios where the resources allocated to the replicas, the number of worker nodes, the encoder and the concurrency allowed in the replicas are varied.

Listing 3 presents an illustrative example of the ffmpeg commands executed inside the replicas to encode using `libx265` . The output resolution is 4K and the bitrates is 8 Mb/s for `libx265`, which are the minimum recommended bitrates for these codecs[3].

**Listing 3** Sample ffmpeg command executed inside the replicas to encode a video chunk using x265.

```
ffmpeg -i chunk1.mp4 -c:v libx265 -b:v 8M -maxrate 8M -minrate 8M
    -preset medium echunk1.mp4
```

[3]See for instance https://support.google.com/youtube/answer/2853702

Table 3 summarises the times obtained for the encoder `libx265` , and the three scenarios. Download, encoding, and upload columns show the mean time (in milliseconds) required to download, encode, and upload the 138 segments of the video. The best scenario is scenario 2 (with one fat VM and four replicas of the function) and the worst setup is scenario 1. The encoding time for x265 in the scenario 2 improves the encoding time by 8,42%.

The mean upload times are sort for x265 due to the size of the videos are smaller (the bitrate is 8 Mb). The loss in performance in scenario 1 can be attributed to the consumption of resources by the Knative service mesh in each worker node, which is around 480 milicores per node. In this setup, each VM has 2 vCPUs (or 2000 millicores), so the service mesh consumes a 24% of the available resources. Therefore, with 4 worker nodes the service mesh consumes almost 2 vCPUs. In the other two scenarios, with one worker node, the service mesh resource consumption represents only the 6% (480 millicores of the 8000 millicores available).

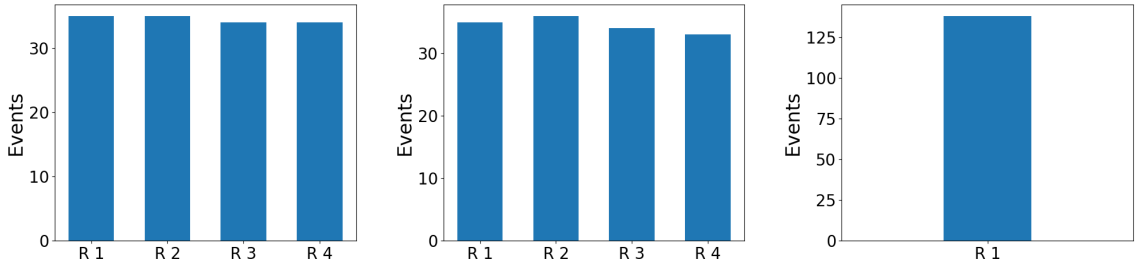
The total time in seconds required to encode all the segments of the video is shown in the last column of Table 3. This is in line with the known efficiency of `libx265` in achieving greater compression at the cost of more complexity.

| Name | Dowload(ms) | Encoding(ms) | Upload(ms) | Total(s) |
|---|---|---|---|---|
| $<\text{x265}>^{e_1}_{\text{slim}}$ | 175 | 71938 | 50 | 2518,78 |
| $<\text{x265}>^{e_2}_{\text{slim}}$ | 186 | 66351 | 57 | 2320,10 |
| $<\text{x265}>^{e_3}_{\text{fat}}$ | 160 | 67251 | 52 | 2332,38 |

Table 3: Mean times in milliseconds (per segment) to download, encode and upload. Last column shows the total seconds to encode all the chunks of the video.

Figure 6 shows the job distribution per replica during the encoding process. It is important to highlight that the computational burden of the encoding depends on the contents of the segments, therefore it is possible for a replica to execute some more jobs. However, as a new job is delivered to a replica only when it notifies the completion of the task, all the replicas spend a similar total amount of time encoding (in this experiment the maximum difference among all replicas was only 0,6%). In scenario 3 with a fat VM with one fat function, all encoding tasks are handled by the only available replica, but in this case up to four jobs can be executed concurrently.

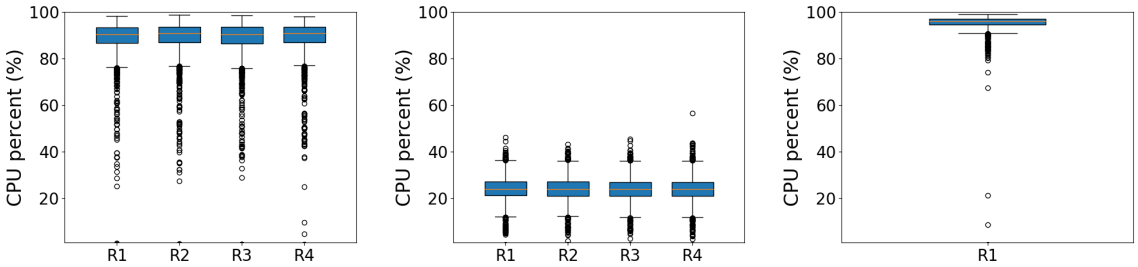
Turning the attention to resource consumption, Figure 7 shows the boxplot of the CPU consumption, obtained from the `percentageCPU` value from `docker stats`, of each replica involved in the encoding process. Scenario 1 with four VMs and four slim replicas each encoding consumed nearly all the CPU resources of the underlying VM. In scenario 3 with




(a) $<\text{x265}>^{e_1}_{\text{slim}}$ (b) $<\text{x265}>^{e_2}_{\text{slim}}$ (c) $<\text{x265}>^{e_3}_{\text{fat}}$

Figure 6: Number of events (encoding jobs) managed by each replica when encoding the 138 chunks of SKA video using `libx265` in the considered scenarios.

one fat VM that allocates one fat replica, the same behaviour is obtained. In scenario 2, with one fat VM and four slim replicas, each replica consumes one fourth of the available CPU resources, therefore the aggregated effect of the four replicas also saturates the CPU resources.

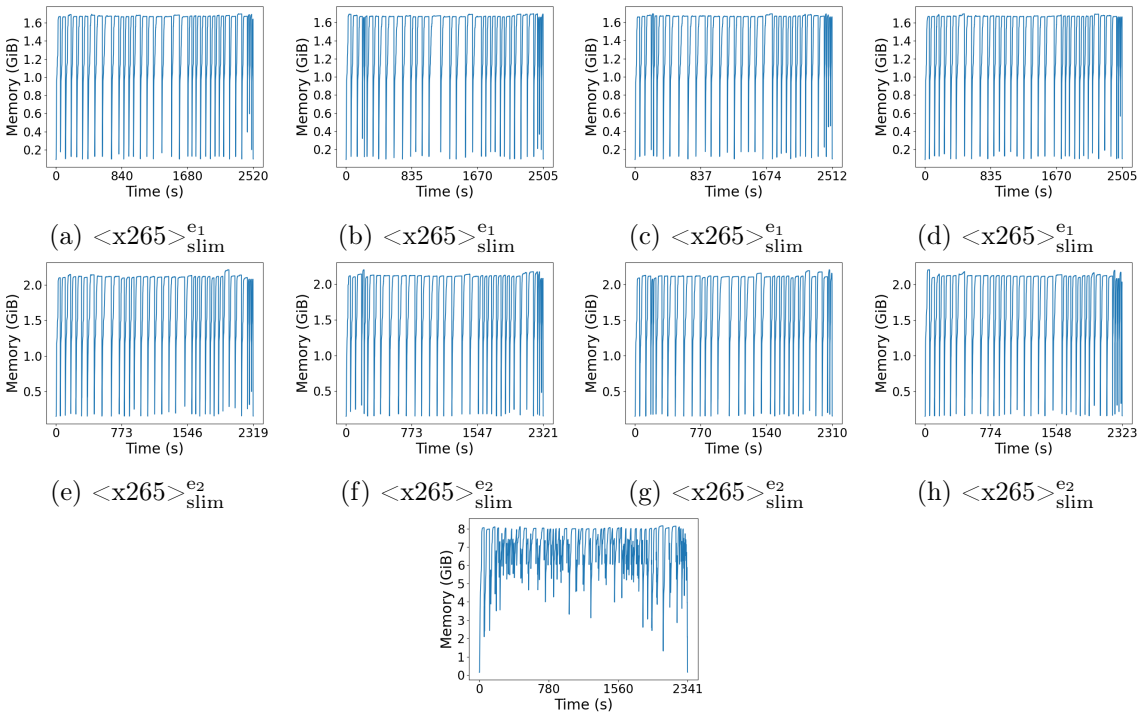



(a) $<\text{x265}>^{e_1}_{\text{slim}}$ (b) $<\text{x265}>^{e_2}_{\text{slim}}$ (c) $<\text{x265}>^{e_3}_{\text{fat}}$

Figure 7: CPU consumption(in %) per scenario for each replica of the encoder functions: `libx264` (top) `libx265` (bottom)

Figure 8 show the memory consumption of each replica during the encoding process, offering insight of the different memory requirements for replicas. The top row shows the data for scenario 1 (four each slim replicas in its own slim VM) where the maximum memory consumption is 1,7 GB and this maximum is fairly stable among jobs. The central row shows the data for the scenario 2 (four slim replicas in a fat VM) where we can observe peaks at 2,2 GB, but most jobs consume 2,1 GB.

In scenario 2 there is an increase of 0,4 GB of RAM per replica compared to the RAM consumption in scenario 1. In scenario 1 the replica only sees the 2 vCPUs of the VM, but in scenario 2 the replica sees the 8 vCPUs, even though the function is limited to use 2000 millicores. The `ffmpeg` call used starts as many threads as vCPUs are available. Therefore,




(a) $<\text{x265}>^{e_1}_{\text{slim}}$ (b) $<\text{x265}>^{e_1}_{\text{slim}}$ (c) $<\text{x265}>^{e_1}_{\text{slim}}$ (d) $<\text{x265}>^{e_1}_{\text{slim}}$

(e) $<\text{x265}>^{e_2}_{\text{slim}}$ (f) $<\text{x265}>^{e_2}_{\text{slim}}$ (g) $<\text{x265}>^{e_2}_{\text{slim}}$ (h) $<\text{x265}>^{e_2}_{\text{slim}}$

(i) $<\text{x265}>^{e_3}_{\text{fat}}$

Figure 8: Memory consumption (in GiB) vs time (in seconds) per replica of the encoder function `libx265`. Top row show results for scenario 1, central row for scenario 2, and bottom row for scenario 3.

when one replica runs `ffmpeg` in a slim VM, that has 2 vCPUS, two threads are started. When four replicas running `ffmpeg` are executed in a fat VM, with 8 vCPUS, each `ffmpeg` process launches eight threads. This increase in the number of threads accounts for the increase of 0,4 GB of RAM consumption per replica.

Finally, the bottom row is devoted to scenario 3 (one fat replica in a fat VM with up to four concurrent jobs) the maximum memory is four-fold those of scenario 2. In scenario 3 we observe that, due to the concurrency of tasks, the memory usage does not drop as there is always at least one `ffmpeg` process actively processing a video.

### 4.4.2 Encoding times and resource consumption in the scenarios with GPUs

This subsection presents the data obtained when the encoding is performed in the three scenarios that utilise GPUs and compares these results with those presented in the previous section. Table 4 describes the different scenarios studied, the resources allocated to the replicas, the number of worker nodes, the number of GPUs used per replica, and the concurrency (number of simultaneous encoding jobs) allowed in the replicas.

| Name | VMs | Replicas | GPUs per replica | CPUs | RAM | Concurrency |
|---|---|---|---|---|---|---|
| $<\text{hevc\_nvenc}>^{e_4}_{\text{slim}}$ | 4 | 4 | 1 | 2 | 24 | 1 |
| $<\text{hevc\_nvenc}>^{e_5}_{\text{slim}}$ | 1 | 4 | 1 | 2 | 24 | 1 |
| $<\text{hevc\_nvenc}>^{e_6}_{\text{fat}}$ | 1 | 1 | 4 | 8 | 96 | 4 |

Table 4: Different scenarios where the resources allocated to the replicas, the number of worker nodes, the number of GPUs per replica and the concurrency allowed in the replicas are varied.

Listing 4 presents an illustrative example of the ffmpeg commands executed inside the replicas to encode one video chunk with `hevc_nvenc` using the GPU.

**Listing 4** Sample ffmpeg commands executed inside the replicas to encode a video chunk with `hevc_nvenc` using the GPU.

```
ffmpeg -i chunk1.mp4 -c:v hevc_nvenc -b:v 8M -maxrate 8M -minrate 8M
    -rc vbr -preset medium echunk1.mp4
```

Table 5 summarises the results for the three scenarios that entail the use of GPUs. Download, encoding, and upload columns shows the mean time (in milliseconds) required to download, encode, and upload the 138 segments of the video respectively.

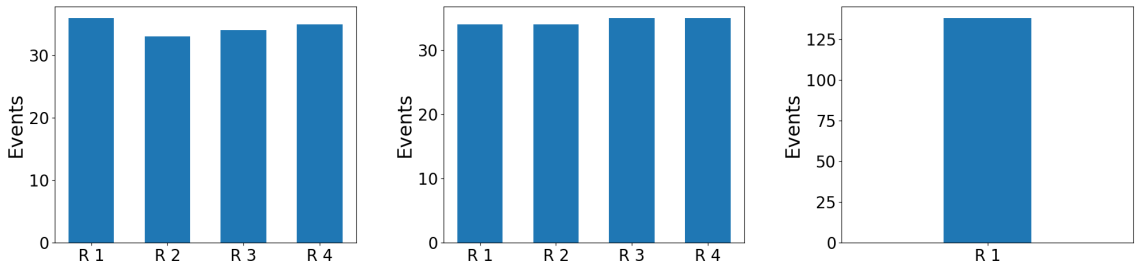
| Name | Download(ms) | Encode(ms) | Upload(ms) | Total(s) |
|---|---|---|---|---|
| $<\text{hevc\_nvenc}>^{e_4}_{slim}$ | 153 | 8771 | 50 | 315,77 |
| $<\text{hevc\_nvenc}>^{e_5}_{slim}$ | 164 | 7613 | 59 | 278,37 |
| $<\text{hevc\_nvenc}>^{e_6}_{fat}$ | 141 | 7948 | 52 | 286,23 |

Table 5: Mean times (per segment) to download, encode and upload. Last column shows the total time to encode all the chunks of the video.

We can draw similar conclusions to those obtained in the previous section. The scenario does not have a noticeable impact in the download nor in the upload times, but it has an impact in the encoding times. Similarly to the scenarios without GPUs, the best scenario for both codecs is scenario 5 (with one fat VM and four slim replicas of the function) and the worst setup is scenario 4. The mean encoding time per segment for `hevc_nvenc` in the scenario 5 improves a 15,21%. In contrast with the scenarios that do not require GPU, the implementation that uses GPUs produces similar encoding times.

The total time (in seconds) required to encode all the segments of the video is shown in the last column of Table 5. Comparing the data for scenario 5 with the last column of scenario 2 in Table 3, we can see the benefits of using the GPU. Specifically, `hevc_nvenc` is 8,33 times faster than `libx265`.

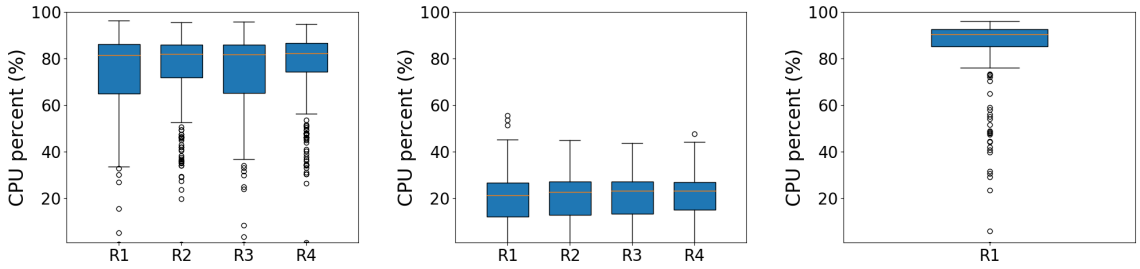
As illustrated in Figure 9, during the encoding process utilising GPUs, the workload is distributed among all the replicas in scenarios 4 and 5. Even though the number of jobs completed at each replica is slightly different, the maximum difference between completion times is less than 1,12% in all cases. The single replica in scenario 6 deals with all the jobs, running up to four encoding jobs concurrently.




(a) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (b) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (c) $<\text{hevc\_nvenc}>^{e_6}_{fat}$

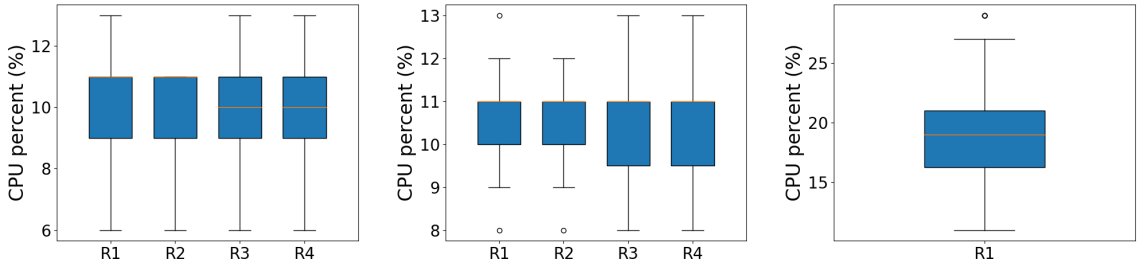
Figure 9: Number of events (encoding jobs) managed per replica to encode the 138 chunks of SKA video using `hevc_nvenc` (bottom) in the three scenarios with GPU.

Finally, we examine the resource utilization of each GPU replica, considering resources from the VM (CPU and RAM) and resources from the GPU (GPU utilization and GPU memory usage) during the encoding process. Figures 10 and 11 show how `hevc_nvenc` influence CPU and GPU utilization respectively across different scenarios.




(a) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (b) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (c) $<\text{hevc\_nvenc}>^{e_6}_{fat}$

Figure 10: CPU consumption (in %) per scenario for each replica of the encoder functions using `hevc_nvenc`



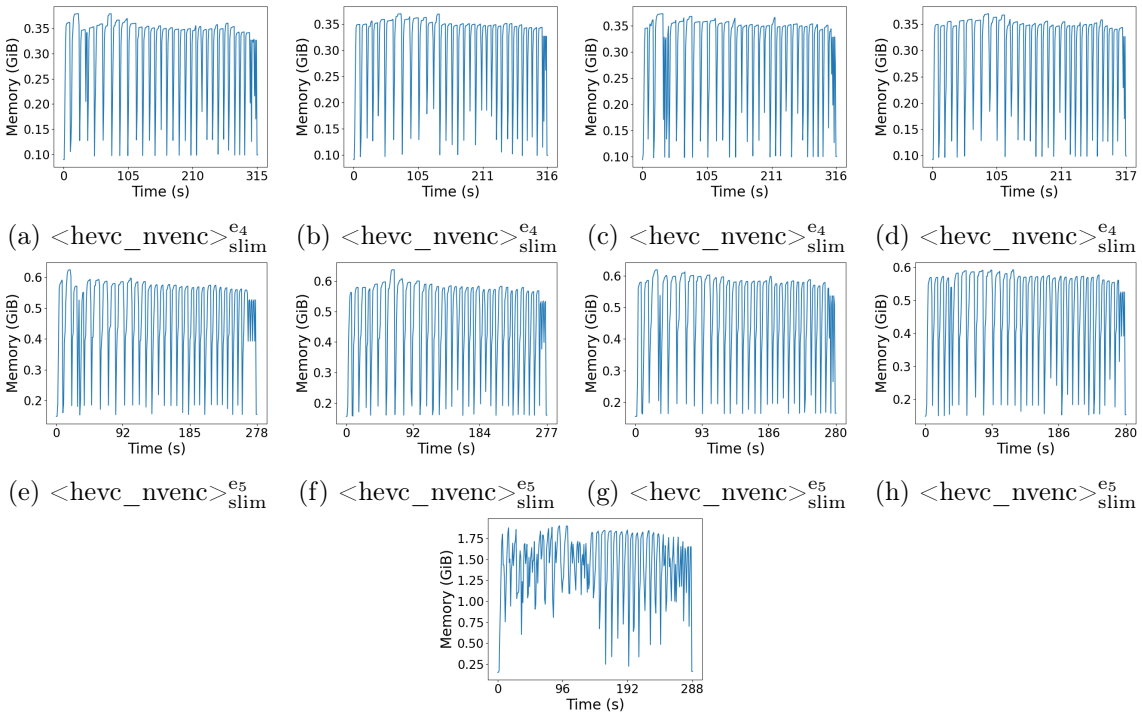

(a) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (b) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (c) $<\text{hevc\_nvenc}>^{e_6}_{fat}$

Figure 11: GPU utilization (in %) per scenario for each replica of the encoder functions using `h264_nvenc` (top) and `hevc_nvenc` (bottom)

Comparing Figure 7 with Figure 10 we can observe a reduction in CPU consumption when GPUs are used. As an example, in scenario 4 using `hevc_nvenc` the mean CPU consumption is around 80% while in scenario 1 using `hevc_nvenc` the mean CPU consumption is around 85%.

From Figure 11 we can see that in scenario 4 using `hevc_nvenc` the mean GPU utilization is 10,5%. The last column in Figure 11 shows the aggregated GPU utilization of the four GPUs assigned to the fat replica.

The RAM consumption for `hevc_nvenc` is shown in Figure 12. Comparing this data with Figure 8 we observe the following reductions in RAM usage comparing `hevc_nvenc` with `libx265` : 77,7% (scenarios 4 and 1); 71,2% (scenarios 5 and 2); and 76,7% (scenarios




(a) <hevc_nvenc>$^{e_4}_{slim}$ (b) <hevc_nvenc>$^{e_4}_{slim}$ (c) <hevc_nvenc>$^{e_4}_{slim}$ (d) <hevc_nvenc>$^{e_4}_{slim}$

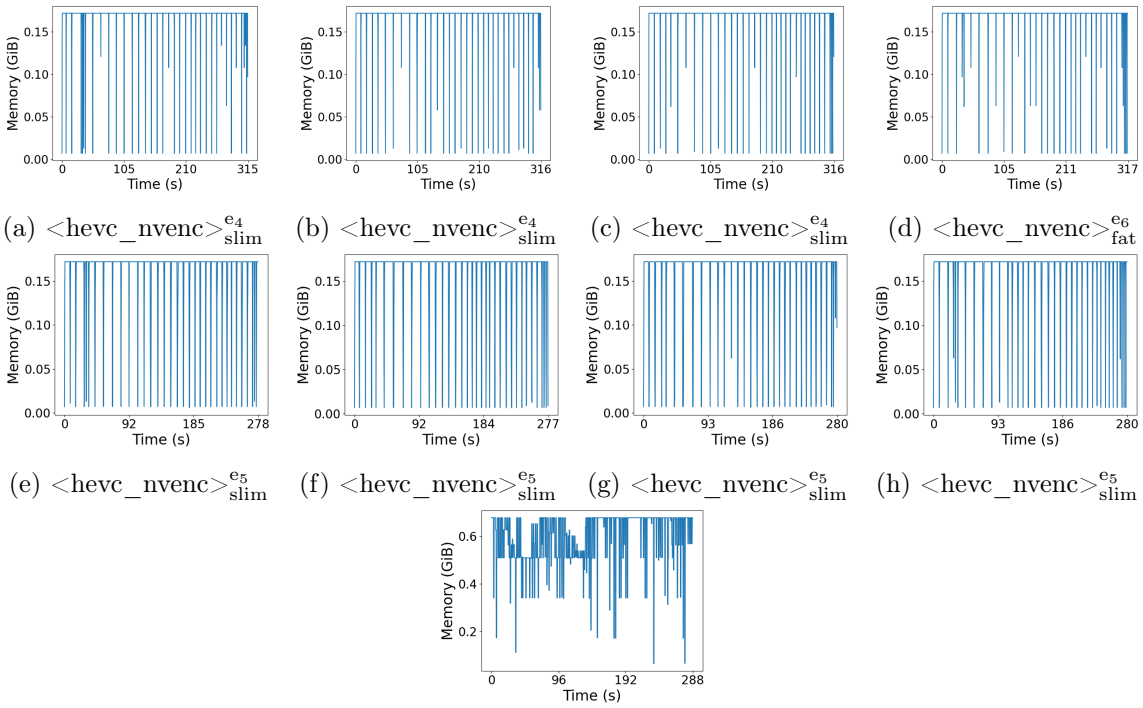
(e) <hevc_nvenc>$^{e_5}_{slim}$ (f) <hevc_nvenc>$^{e_5}_{slim}$ (g) <hevc_nvenc>$^{e_5}_{slim}$ (h) <hevc_nvenc>$^{e_5}_{slim}$

(i) <hevc_nvenc>$^{e_6}_{fat}$

Figure 12: Memory consumption (in GiB) vs time (in seconds) per replica of the encoder function `hevc_nvenc` for scenario 4 (top row), scenario 5 (middle row), and scenario 6 (bottom row)

6 and 3).

Finally, Figure 13 show the GPU memory usage for `hevc_nvenc` . The `hevc_nvenc` encoder consumes low memory in the GPU: 0,17 GB for scenario 4, 0,17 GB for scenario 5, and 0,67 GB for scenario 6. Besides, adding the RAM consumption and the memory usage in the GPU, the total memory is lower than the RAM required for the encoder that do not use GPU.




(a) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (b) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (c) $<\text{hevc\_nvenc}>^{e_4}_{slim}$ (d) $<\text{hevc\_nvenc}>^{e_6}_{fat}$

(e) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (f) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (g) $<\text{hevc\_nvenc}>^{e_5}_{slim}$ (h) $<\text{hevc\_nvenc}>^{e_5}_{slim}$

(i) $<\text{hevc\_nvenc}>^{e_6}_{fat}$

Figure 13: GPU memory usage (in MiB) vs time (in seconds) per replica of the encoder function `hevc_nvenc` for scenario 4(top row), scenario 5 (middle row), and scenario 6 (bottom row).

Summarising, there is a noticeable reduction in both CPU and RAM usage when GPUs are employed. This reduction is expected, as the GPU takes on the bulk of the computational load, allowing the machine's other resources to operate with less strain.

## 4.5 Function performance in multiresolution video encoding

From the data obtained in the previous subsections, scenario 2 (without GPUs) and scenario 5 (with GPUs) provided the best results in terms of coding time. In this section, we will

select those scenarios to perform the encoding of all the segments of the four videos detailed in Table 1. Besides, each segment will be encoded in different representations in a single call so that each segment will be transferred to the GPU only once. The output of each function will be five representations with different resolution and bitrates.

### 4.5.1 Multiresolution encoding without GPU

Listing 5 shows a sample command used to encode a segment, in a single ffmpeg call, at five different resolutions.

**Listing 5** Sample calls to perform multiresolution encoding: using `libx265` .

```
#x265 without gpu
ffmpeg -i chunk1.mp4
    -filter_complex '[0:v]yadif,split=5[out1][out2][out3][out4][out5]'
    -map '[out1]' -c:v libx265 -s 854x480 -b:v 1350k -maxrate 1500k
        -bufsize 3M -preset medium echunk1_480p.mp4
    -map '[out2]' -c:v libx265 -s 1280x720 -b:v 2750k -maxrate 4M
        -bufsize 8M -preset medium echunk1_720p.mp4
    -map '[out3]' -c:v libx265 -s 1920x1080 -b:v 6M -maxrate 8M
        -bufsize 16M -preset medium echunk1_1080p.mp4
    -map '[out4]' -c:v libx265 -s 2560x1440 -b:v 8M -maxrate 11M
        -bufsize 22M -preset medium echunk1_1440p.mp4
    -map '[out5]' -c:v libx265 -s 3840x2160 -b:v 11M -maxrate 14M
        -bufsize 28M -preset medium echunk1_2160p.mp4
```

Detailed times during the encoding of all the segments of the four videos are provided in Table 6. The first column represents the video, the second one the mean time to download a segment of the video, the third one the mean time to encode a video chunk, the fourth is the mean time to upload the encoded video chunk, the fifth is the total size of the video, and finally, the last column shows the total time to encode all the chunks of the video.

If we compare the mean time to encode each segment of SKA with `libx265` in Table 6 and that of the scenario 2 in Table 3, we see that it takes around 128 seconds to encode the five representations in a single ffmpeg call while it took around 72 seconds to encode a single representation. The mean encoding time, when performing multiresolution, with this encoder increments by a 1,78x factor.

### 4.5.2 Multiresolution encoding with GPU

Listing 6 shows a sample commands executed inside the replicas with GPUs to encode a segment, in a single ffmpeg call, at five different resolutions.

Table 7 presents the times obtained when encoding all the segments of the four videos detailed in Table 1. The first column represents the video, the second one the mean time

| Video | Download(ms) | Encode(ms) | Upload(ms) | Size(MB) | Total(s) |
|---|---|---|---|---|---|
| **Results for libx265** | | | | | |
| ANC | 102 | 136536 | 197 | 496 | 3264,97 |
| ELD | 166 | 134347 | 201 | 545 | 3139,70 |
| IND | 207 | 166397 | 221 | 775 | 5307,64 |
| SKA | 183 | 128225 | 190 | 748 | 4480,60 |

Table 6: Mean times (per segment) to download, encode and upload. Last column shows the total time to encode all the chunks of the video. The two encoders have been evaluated without GPU for each video.

**Listing 6** Sample call to perform multiresolution encoding with GPUs using `hevc_nvenc` .

```
#h265/HEVC with GPU
ffmpeg -i chunk1.mp4
    -filter_complex '[0:v]yadif,split=5[out1][out2][out3][out4][out5]'
    -map '[out1]' -c:v hevc_nvenc -s 854x480 -b:v 1350k -maxrate 1500k
        -bufsize 3M -b_ref_mode 0 -rc vbr -preset medium echunk1_480p.mp4
    -map '[out2]' -c:v hevc_nvenc -s 1280x720 -b:v 2750k -maxrate 4M
        -bufsize 8M -b_ref_mode 0 -rc vbr -preset medium echunk1_720p.mp4
    -map '[out3]' -c:v hevc_nvenc -s 1920x1080 -b:v 6M -maxrate 8M
        -bufsize 16M -b_ref_mode 0 -rc vbr -preset medium echunk1_1080p.mp4
    -map '[out4]' -c:v hevc_nvenc -s 2560x1440 -b:v 8M -maxrate 11M
        -bufsize 22M -b_ref_mode 0 -rc vbr -preset medium echunk1_1440p.mp4
    -map '[out5]' -c:v hevc_nvenc -s 3840x2160 -b:v 11M -maxrate 14M
        -bufsize 28M -b_ref_mode 0 -rc vbr -preset medium echunk1_2160p.mp4
```

to download a segment of the video, the third one the mean time to encode a segment, the fourth is the mean time to upload a segment, the fifth is the total size of the video and finally, the last column shows the total time to encode the video.

If we compare the mean time to encode each segment of SKA with `libx265` in Table 6 and that of the scenario 2 in Table 3, we see that it takes around 11,76 seconds to encode the five representations in a single ffmpeg call while it took around 7,6 seconds to encode a single representation. The mean encoding time, when performing multiresolution, with this encoder increments by a 1,54x factor.

Comparing the data for `libx265` without GPU and `hevc_nvenc` with GPU in Tables 6 and 7, taking the mean across all videos, the mean time to encode a segment using `libx265` without GPU is 141,37 seconds while the mean encoding time using `hevc_nvenc` with GPU is 11,37 seconds. Therefore, when encoding with the GPU, the mean time to encode a segment has been reduced drastically by a factor of 12,43.

| Video | Download(ms) | Encode(ms) | Upload(ms) | Size(MB) | Total(s) |
|---|---|---|---|---|---|
| **Results for hevc_nvenc** | | | | | |
| ANC | 82 | 10320 | 177 | 525 | 255,11 |
| ELD | 146 | 11165 | 236 | 648 | 272,20 |
| IND | 179 | 12251 | 232 | 904 | 405,71 |
| SKA | 158 | 11757 | 210 | 822 | 424,78 |

Table 7: Mean times (per segment) to download, encode and upload. Last column shows the total time to encode all the chunks of the video. The two encoders have been evaluated with GPU for each video.

## 5 Comparative analysis with related work

In this section, we present a comparative analysis between our proposed architecture and the existing works. As described in Section 2, the antecedent studies pertinent to the serverless video encoding paradigm markedly differ from the analysis performed in this work. To perform this comparison we have extracted 11 features from the most close papers and from the proposal as shown in Table 8. The cells marked as N/A denote that we were not able to obtain the information from the respective source papers.

As we have discussed in Section 2, we have not found studies that perform a detailed study of the behaviour of encoders in serverless platforms using GPUs. For this reasons, we

want to make a comparison with [26], [28], and [29]. Both [26] and [28] leverage Amazon's serverless platform, AWS Lambda, for the execution of their respective frameworks. This choice provides them with a virtually boundless pool of resources. In contrast, both [29] and our proposed solution adopt Knative as the serverless platform deployed on premises. This approach, while constrained in terms of resources, allows us comprehensive control over the system, emphasising a trade-off between resource limitations and enhanced system governance.

Table 8: Comparison of video encoding systems based on serverless architectures.

| ***Features*** | **Fouladi, S. et al. (2017) [26]** | **Ao, L. et al. (2018) [28]** | **Moina-Rivera, W. et al. (2022) [29]** | **Our proposal (2023)** |
|---|---|---|---|---|
| *Platform ownership* | Amazon | Amazon | Own Platform | Own Platform |
| *Serverless platform* | AWS lambda | AWS lambda | Knative<br>Open-source solution | Knative<br>Open-source solution |
| *Framework for managing parallel execution* | Mu | Mu | CloudEvents | CloudEvents |
| *Video Codecs Testing* | VP8 | N/A | x264 and AV1 | x265 and HEVC NVENC |
| *Video resolutions tested* | 4K | 1080p and 720p | 4K | 4K, 1440p, 1080p,<br>720p, 480p |
| *Other serverless functions* | No | Yes, it includes<br>facial recognition | Yes, it calculates<br>VMAF quality metric | No |
| *Video chunk size* | 4 seconds, divided into<br>16 microchunks of<br>6 frames each | 2 seconds | 2 seconds | 2 seconds |
| *Modified video codec* | Yes | No, it is based<br>on ffmpeg tool | No, it is based on<br>ffmpeg tool | No, it is based<br>on ffmpeg tool |
| *Resource constrained serverless platform* | No | No | Yes | Yes |
| *Scalability and performance study* | Low | Medium | High | High |
| *GPU video encoding support* | No | No | No | Yes |

Mu framework serves as a fundamental component in the orchestration of parallel executions, a role shared by [26] and [28]. In contrast, both [29] and our proposed solution leverage CloudEvents in conjunction with the Knative open-source serverless platform. It is noteworthy to mention that the application of the Mu framework is now considered quite limited, given its abandonment in 2018 and the subsequent absence of support. This renders our choice of CloudEvents and Knative more contemporary and applicable.

An examination of the video encoding systems presented reveals substantial disparities between the earlier proposals and our system. Notably, the solution proposed in [26] exclusively supports the VP8 codec, which has fallen out of favor in contemporary contexts due to the widespread compatibility offered by the H.264 (`libx264`) and VP9 (an evolution of VP8) codec family. Moreover, VP8 struggles to deliver optimal performance for ultra-high resolutions such as 4K, where alternatives like H.265 (`libx265`) and AV1 showcase superior efficiency.

In contrast, our proposal takes a comprehensive approach by considering two distinct codecs: `libx265`, and `hevc_nvenc`. The encoder `libx265` is preferred for ultra-high resolutions (4K and 8K).

In [29], the authors experimented with the AV1 codec. However, our proposal excluded AV1 due to the absence of GPU support for this codec in the available hardware used for our experiments. Consequently, a direct comparison with CPU-based implementations was not feasible.

Our proposal, along with [26] and [29], underwent testing using 4K videos, while [28] employed videos with resolutions of 1080p and 720p. This divergence in testing resolutions underscores the varying experimental setups and objectives across these studies.

Regarding chunk sizes, our analysis reveals a consistent pattern among [28], [29], and our proposed solution, wherein a 2-second chunking strategy is employed. Conversely, [26] opts for larger 4-second chunks, further subdivided into micro-chunks, each comprising six frames. The selection of chunk sizes, typically ranging between 2 to 4 seconds, represents a well-balanced compromise between encoding efficiency and adaptability to varying bandwidth conditions [76].

Some proposals require the adaptation of video codecs to exploit a finer parallelism. In the case of [26], adjustments to the VP8 codec were imperative to ensure the proper functioning of their proposal. In contrast, [28], [29], and our proposal relied on standard implementations of video codecs embedded within the versatile FFmpeg tool. This discrepancy underscores the diverse approaches taken by different frameworks, with each solution tailoring its codec strategy to align with the unique requirements of the serverless architecture.

Another feature presented in Table 8 is the incorporation of functions beyond video coding. In the case of [28], supplementary functions, such as face detection, augment

their system's capabilities. Conversely, [29] integrates a video quality function, calculating VMAF per chunk by comparing a reference video with an encoded counterpart. In this work, we focussed on video encoding. However, it is worth noting that the architecture is inherently extensible, allowing for the straightforward integration of additional functions if required. This intentional modularity ensures adaptability to diverse use cases and evolving requirements without compromising the core video encoding functionality.

Notably, [29] and our proposed solution are tailored for environments with constrained resources, presenting different scenarios and analysing the behaviour in these limited environments. Conversely, other proposals such as [26] and [28] were conceptualized within environments where the primary constraint is monetary cost rather than computational resources.

Furthermore, our proposal offers an in-depth exploration of system scalability and performance, specifically when GPUs are employed for the encoding process comparing the results with equivalent encoders that do not use the GPUs. While [29] conducted a similar study, however it did not account for GPU usage. In [28], though a performance study is conducted, the level of depth is not as extensive. On the other hand, [26] presents a more lightweight study regarding the performance and scalability of their proposal, lacking the depth found in subsequent works.

This comparative analysis illuminates the progressive evolution in the depth and breadth of performance studies across these works, with our proposal placing particular emphasis on performance and the impact of GPU utilization.

## 6 Conclusions and future work

Multimedia streaming has become an indispensable part of modern life, transforming how we consume entertainment, access information, and engage with the world around us. The rise of cloud computing has played a pivotal role in this evolution, providing the infrastructure and scalability necessary to deliver high-quality streaming experiences to a global audience. In particular, serverless architectures are well-suited for video coding systems in HAS applications, where video is divided into segments of 2 or 4 seconds typically, due to their dynamic resource allocation, parallel execution capabilities, and automatic scaling features. This approach can significantly reduce encoding time and improve overall streaming efficiency.

This study has developed and benchmarked two distinct event-driven serverless functions tailored for video encoding in the cloud edge, evaluating their performance both with and without GPU acceleration. These functions have been encapsulated within two containers, and their operational characteristics were analysed through deployment on an on-premises resource-constrained serverless platform managed by Knative. Our experiments

focused on assessing resource consumption under different configurations. The investigation involved varying the allocated resources, considering both fat and slim function replicas, deployed on both slim and fat VMs. This approach allowed for a thorough examination of the serverless functions' performance characteristics, offering insight on their behaviour in real-world deployment scenarios and contributing valuable data to the ongoing development on serverless computing and resource management.

The experiments demonstrate a balanced distribution of encoding tasks across all replicas. For both CPU and GPU codecs, the optimal setup was found to be a large VM hosting four smaller replicas of the function. In contrast, the least efficient configuration involved four small VMs, each running a single replica of the function.The GPU-accelerated version (`hevc_nvenc`) outperforms its CPU-only counterpart (`libx265`) significantly, being 8,33 times faster. This advantage is particularly valuable in live streaming applications, where reducing latency is crucial for achieving optimal performance. The benefits of using `hevc_nvenc` are even more evident in multiresolution encoding scenarios, which are common in HAS, where it reduces the mean time to encode a segment in five resolutions by a factor of 12,43.

As future work, we plan to adapt this platform to perform live multiresolution encoding in the cloud edge for HAS applications. Besides, we will explore new event-driven functions to support the execution of the encoder AV1 in the GPU. Additionally, our ongoing efforts are directed towards the development of a serverless platform designed for the field-of-view (FoV) based encoding of 360VR videos. This initiative aims to leverage serverless architecture to enhance the efficiency and adaptability of video encoding processes specifically targeted at immersive 360-degree virtual reality content. We seek to further advance the capabilities and versatility of our system, ensuring its relevance and applicability in emerging domains.

## Funding

Grant by PID2021-126209OB-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", by the "European Union".

## References

[1] S. Vijay, P. Mann, R. Chaudhary, and A. Rana, "Emerging Trends in Multimedia," in *Emerging Technologies in Data Mining and Information Security*, ser. Lecture Notes in Networks and Systems, vol. 491. Singapore: Springer, 2023, pp. 301—-311.

[2] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[3] M. Amini Salehi and X. Li, *Future of Multimedia Streaming and Cloud Technology*. Cham: Springer International Publishing, 2021, pp. 179–187.

[4] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[5] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What Serverless Computing is and Should Become: The next Phase of Cloud Computing," *Communications of the ACM*, vol. 64, no. 5, p. 76–84, 4 2021.

[6] D. Taibi, J. Spillner, and K. Wawruch, "Serverless Computing-Where Are We Now, and Where Are We Heading?" *IEEE Software*, vol. 38, no. 1, pp. 25–31, 2021.

[7] S. Katsigiannis, V. Dimitsas, and D. Maroulis, "A GPU vs CPU performance evaluation of an experimental video compression algorithm," in *Seventh International Workshop on Quality of Multimedia Experience*, ser. QoMEX'15, 5 2015, pp. 1–6.

[8] The Knative Authors, "Knative," 2022. [Online]. Available: https://knative.dev/

[9] W. Moina-Rivera, M. Garcia-Pineda, J. Gutiérrez-Aguado, and J. M. Alcaraz-Calero, "Cloud media video encoding: review and challenges," *Multimedia Tools and Applications*, Mar 2024. [Online]. Available: https://doi.org/10.1007/s11042-024-18763-2

[10] J. Gutiérrez-Aguado, R. Peña-Ortiz, M. García-Pineda, and J. M. Claver, "Cloud-based elastic architecture for distributed video encoding: Evaluating H.265, VP9, and AV1," *Journal of Network and Computer Applications*, vol. 171, p. 102782, 12 2020.

[11] C.-T. Yang, H.-Y. Wang, W.-S. Ou, Y.-T. Liu, and C.-H. Hsu, "On implementation of gpu virtualization using pci pass-through," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, 2012, pp. 711–716.

[12] J. Gutiérrez-Aguado, J. M. Claver, and R. Peña-Ortiz, "Toward a transparent and efficient GPU cloudification architecture," *The Journal of Supercomputing*, vol. 75, no. 7, pp. 3640–3672, 7 2019.

[13] R. Rodríguez-Sánchez, J. L. Martínez, G. Fernández-Escribano, J. L. Sánchez, and J. M. Claver, "A Fast GPU-Based Motion Estimation Algorithm for H.264/AVC," in *Advances in Multimedia Modeling*, K. Schoeffmann, B. Merialdo, A. G. Hauptmann, C.-W. Ngo, Y. Andreopoulos, and C. Breiteneder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 551–562.

[14] W. Xiao, B. Li, J. Xu, G. Shi, and F. Wu, "HEVC Encoding Optimization Using Multicore CPUs and GPUs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 11, pp. 1830–1843, 2015.

[15] M. Srinivasan, "Vp9 encoder and decoders for next generation online video platforms and services," in *SMPTE 2016 Annual Technical Conference and Exhibition*, 2016, pp. 1–14.

[16] P. Comi, P. S. Crosta, M. Beccari, P. Paglierani, G. Grossi, F. Pedersini, and A. Petrini, "Hardware-accelerated high-resolution video coding in virtual network functions," in *2016 European Conference on Networks and Communications (EuCNC)*, 2016, pp. 32–36.

[17] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and Virtual Machines at Scale: A Comparative Study," in *17th International Middleware Conference*, ser. Middleware'16. Trento Italy: ACM, 11 2016.

[18] N. Marathe, A. Gandhi, and J. M. Shah, "Docker Swarm and Kubernetes in Cloud Computing Environment," in *3rd International Conference on Trends in Electronics and Informatics*, ser. ICOEI'19, 4 2019, pp. 179–184.

[19] A. Jangda, D. Pinckney, Y. Brun, and A. Guha, "Formal Foundations of Serverless Computing," *Proceedings of the ACM on Programming Languages*, vol. 3, p. 1–26, 10 2019.

[20] H. Martins, F. Araujo, and P. R. da Cunha, "Benchmarking Serverless Computing Platforms," *Journal of Grid Computing*, vol. 18, no. 4, pp. 691–709, 7 2020.

[21] OpenFaaS Ltd., "OpenFaas: Serverkess Functions, Made Simple," 2023. [Online]. Available: https://www.openfaas.com

[22] The Knative Authors, "Knative," 2022. [Online]. Available: https://knative.dev

[23] The Apache Software Foundation, "Apache OpenWhisk: Open Source Serverless Cloud Platform," 2023. [Online]. Available: https://openwhisk.apache.org

[24] Iron.io, "IronFunctions: Open Source Serverless Computing," 2016. [Online]. Available: https://open.iron.io

[25] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, "Serverless Workflows for Containerised Applications in the Cloud Continuum," *Journal of Grid Computing*, vol. 19, no. 3, p. 30, 7 2021.

[26] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, Fast and

Slow: Low-Latency Video Processing Using Thousands of Tiny Threads," in *14th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'17. Boston, MA, USA: USENIX Association, 2017, pp. 363–376. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi17/nsdi17-fouladi.pdf

[27] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," in *USENIX Annual Technical Conference*, ser. ATC'18. Boston, MA, USA: USENIX Association, 7 2018, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang

[28] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A Serverless Video Processing Framework," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC'18. Carlsbad, CA, USA: ACM, 10 2018, p. 263–274.

[29] W. Moina-Rivera, M. Garcia-Pineda, J. M. Claver, and J. Gutiérrez-Aguado, "Event-driven serverless pipelines for video coding and quality metrics," *Journal of Grid Computing*, vol. 21, no. 2, 3 2023.

[30] J. Wen, Z. Chen, X. Jin, and X. Liu, "Rise of the Planet of Serverless Computing: A Systematic Review," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, p. 1–61, 7 2023.

[31] A. Salcedo-Navarro, J. Gutiérrez-Aguado, and M. Garcia-Pineda, "Podinsights: a millisecond pod metric collector for kubernetes," in *Euro American Conference on Telematics and Information Systems - Proceedings of the 12th Euro American Conference on Telematics and Information Systems, EATIS 2024*, 2024, accepted: 15th March 2024.

[32] S. Iserte, R. Peña-Ortiz, J. Gutiérrez Aguado, J. M. Claver, and R. Mayo, "GSaaS: A service to cloudify and schedule GPUs," *IEEE Access*, vol. 6, pp. 39 762–39 774, 7 2018.

[33] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "GPU Virtualization and Scheduling Methods: A Comprehensive Survey," *ACM Computing Surveys*, vol. 50, no. 3, p. 1–37, 6 2017.

[34] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim, "GPU Enabled Serverless Computing Framework," in *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, ser. PDP'18. IEEE, 3 2018.

[35] K. Satzke, I. E. Akkus, R. Chen, I. Rimac, M. Stein, A. Beck, P. Aditya, M. Vanga, and V. Hilt, "Efficient GPU Sharing for Serverless Workflows," in *Proceedings of the 1st Workshop on High Performance Serverless Computing*, ser. HPDC'21. Sweden: ACM, 6 2020.

[36] D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, and G. Moltó, "Accelerated serverless computing based on GPU virtualization," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 32–42, 5 2020.

[37] D. M. Naranjo Delgado, M. Contreras, G. Moltó, S. Risco, I. Blanquer, J. Prades, and F. Silla, "On the acceleration of FaaS using remote GPU virtualization," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23 Companion. New York, NY, USA: Association for Computing Machinery, 2023, p. 157–164. [Online]. Available: https://doi.org/10.1145/3578245.3584933

[38] S. Risco and G. Moltó, "GPU-enabled serverless workflows for efficient multimedia processing," *Applied Sciences*, vol. 11, no. 4, p. 1438, 2 2021.

[39] K. Konstantoudakis, D. Breitgand, A. Doumanoglou, N. Zioulis, A. Weit, K. Christaki, P. Drakoulis, E. Christakis, D. Zarpalas, and P. Daras, "Serverless streaming for emerging media: towards 5G network-driven cost optimization," *Multimedia Tools and Applications*, vol. 81, no. 9, pp. 12 211–12 250, 3 2021.

[40] M. Zhang, Y. Zhu, C. Zhang, and J. Liu, "Video processing with serverless computing: a measurement study," in *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV'19. Amherst, MA, USA: ACM, 6 2019.

[41] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in *Proceedings of the 12th ACM Multimedia Systems Conference*, ser. MMSys'21. Istanbul, Turkey: ACM, 7 2021.

[42] M. Zhang, F. Wang, Y. Zhu, J. Liu, and B. Li, "Serverless Empowered Video Analytics for Ubiquitous Networked Cameras," *IEEE Network*, vol. 35, no. 6, p. 186–193, 11 2021.

[43] M. Zhang, Y. Zhu, J. Liu, F. Wang, and F. Wang, "CharmSeeker: Automated Pipeline Configuration for Serverless Video Processing," *Transactions on Networking*, vol. 30, no. 6, p. 2730–2743, 12 2022.

[44] M. Zhang, Y. Zhu, L. Shen, F. Wang, and J. Liu, "OmniSense: Towards Edge-Assisted Online Analytics for 360-Degree Videos," in *Conference on Computer Communications*, ser. INFOCOM'23. IEEE, 5 2023, pp. 1–10.

[45] J. Li, S. Kulkarni, K. Ramakrishnan, and D. Li, "Analyzing Open-Source Serverless Platforms: Characteristics and Performance," in *The Thirty Third International Conferences on Software Engineering and Knowledge Engineering*, ser. SEKE'21. Pittsburgh, USA: KSI Research Inc., 7 2021.

[46] CloudEvents Authors, "Cloudevents project," 2023. [Online]. Available: https://cloudevents.io/

[47] J. Gutiérrez-Aguado, "Adapting embeded Tomcat to develop event-driven serverless functions," in *JCIS2022, SISTEDES*, 2022. [Online]. Available: http://hdl.handle.net/11705/JCIS/2022/040

[48] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, "An Architecture for Distributed High Performance Video Processing in the Cloud," in *3rd International Conference on Cloud Computing*, ser. CLOUD'10. Miami, Florida, USA: IEEE, 7 2010, pp. 482–489.

[49] M. Jeon, N. Kim, and B.-D. Lee, "MapReduce-Based Distributed Video Encoding Using Content-Aware Video Segmentation and Scheduling," *IEEE Access*, vol. 4, pp. 6802–6815, 2016.

[50] C. Song, W. Shen, L. Sun, Z. Lei, and W. Xu, "Distributed video transcoding based on MapReduce," in *13th International Conference on Computer and Information Science*, ser. ICIS'14. Taiyuan, China: IEEE/ACIS, 6 2014, pp. 309–314.

[51] M. Kim, Y. Cui, S. Han, and H. Lee, "Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, pp. 213–224, 2013.

[52] S. Sameti, M. Wang, and D. Krishnamurthy, "Stride: Distributed Video Transcoding in Spark," in *37th International Performance Computing and Communications Conference*, ser. IPCCC'18. Orlando, Florida, USA: IEEE, 11 2018, pp. 1–8.

[53] ——, "CONTRAST: Container-based Transcoding for Interactive Video Streaming," in *Network Operations and Management Symposium*, ser. NOMS'20. Budapest, Hungary: IEEE/IFIP, 4 2020, pp. 1–9.

[54] J. Gutiérrez-Aguado, R. Peña-Ortiz, M. Garcia-Pineda, and J. M. Claver, "A Cloud-Based Distributed Architecture to Accelerate Video Encoders," *Applied Sciences*, vol. 10, no. 15, p. 5070, 7 2020.

[55] Y. Dong, X. Zhang, Y. Zhao, and L. Song, "A containerized media cloud for video transcoding service," in *International Conference on Consumer Electronics*, ser. ICCE'18. Las Vegas, Nevada, USA: IEEE, 1 2018, pp. 1–4.

[56] P. Pääkkönen, A. Heikkinen, and T. Aihkisalo, "Architecture for Predicting Live Video Transcoding Performance on Docker Containers," in *International Conference on Services Computing*, ser. SCC'18. San Francisco, California, USA: IEEE, 7 2018, pp. 65–72.

[57] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, NIST Publications SP 500-325, 9 2011.

[58] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox, "GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications," in *7th International Conference on Cloud Computing*. Anchorage, Alaska: IEEE, 6 2014.

[59] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: GPU Virtualization at the Hypervisor," *IEEE Transactions on Computers*, vol. 65, no. 9, p. 2752–2766, 9 2016.

[60] T. Xu, "GVT-g Setup Guide," 9 2021. [Online]. Available: https://github.com/intel/gvt-linux/wiki/GVTg_Setup_Guide

[61] NVIDIA Corporation, "NVIDIA virtual GPU technology," 2023. [Online]. Available: www.nvidia.com/virtualgpu/

[62] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato, "A complete and efficient CUDA-sharing solution for HPC clusters," *Parallel Computing*, vol. 40, no. 10, p. 574–588, 12 2014.

[63] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, and T. Narumi, "DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment," in *SC Companion: High Performance Computing, Networking Storage and Analysis*. Salt Lake City, Utah, USA: IEEE, 11 2012.

[64] T.-Y. Liang and Y.-W. Chang, "GridCuda: A Grid-Enabled CUDA Programming Toolkit," in *Workshops of International Conference on Advanced Information Networking and Applications*. Biopolis, Singapore: IEEE, 3 2011.

[65] L. Shi, H. Chen, and J. Sun, "vCUDA: GPU accelerated high performance computing in virtual machines," in *International Symposium on Parallel & Distributed Processing*. Chengdu, Sichuan, China: IEEE, 5 2009.

[66] R. Montella, G. Giunta, G. Laccetti, M. Lapegna, C. Palmieri, C. Ferraro, V. Pelliccia, C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "On the virtualization of cuda based gpu remoting on arm and x86 machines in the gvirtus framework," *International Journal of Parallel Programming*, vol. 45, no. 5, p. 1142–1163, 10 2016.

[67] Amazon Web Services, Inc, "Recommended GPU Instances," 2023. [Online]. Available: https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html

[68] Microsoft Azure, "GPU optimized virtual machine sizes," 6 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-gpu

[69] Google Cloud, "GPU platforms," 12 2023. [Online]. Available: https://cloud.google.com/compute/docs/gpus

[70] Alibaba Cloud, "Overview of GPU-accelerated compute-optimized and vGPU-accelerated instance families," 11 2023. [Online]. Available: https://www.alibabacloud.com/help/en/ecs/user-guide/gpu-accelerated-compute-optimized-and-vgpu-accelerated-instance-families-1

[71] R. Rodríguez-Sánchez, J. L. Martínez, G. Fernández-Escribano, J. L. Sánchez, and J. M. Claver, "A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 2012, pp. 166–173.

[72] S. Radicke, J. Hahn, C. Grecos, and Q. Wang, "A highly-parallel approach on motion estimation for high efficiency video coding (hevc)," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*, 2014, pp. 187–188.

[73] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim, "GPU Enabled Serverless Computing Framework," in *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 533–540.

[74] KNIX authors, "KNIX MicroFunctions," 12 2021. [Online]. Available: https://github.com/knix-microfunctions/knix

[75] T.-A. Yeh, H.-H. Chen, and J. Chou, "KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC'20. Stockholm, Sweden: ACM, 6 2020.

[76] S. Lederer, "Bitmovin: Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length," 4 2020. [Online]. Available: https://bitmovin.com/mpeg-dash-hls-segment-length/

[77] C. Cicconetti, M. Conti, A. Passarella, and D. Sabella, "Toward distributed computing environments with serverless solutions in edge systems," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 40–46, 2020. [Online]. Available: http://dx.doi.org/10.1109/mcom.001.1900498