Audience and complexity aware live video encoders orchestration

Abdelmajid Moussaoui
Thomas Guionnet
Mickaël Raulet

CITE THIS ARTICLE

Moussaoui, Abdelmajid; Guionnet,Thomas; Raulet, Mickaël; 2022. Audience and complexity aware live video encoders orchestration. SET INTERNATIONAL JOURNAL OF BROADCAST ENGINEERING. ISSN Print: 2446-9246 ISSN Online: 2446-9432. doi: 10.18580/setijbe.2022.5. Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

Audience and complexity aware live video encoders orchestration

Abdelmajid Moussaoui, Thomas Guionnet, and Mickaël Raulet.

Ateme {a.moussaoui, t.guionnet, m.raulet}@ateme.com

***Abstract*—Video encoding services are known to be computationally intensive. In a software environment, it is desirable to be able to adapt to the available computing resources. Therefore, modern live video encoders have the "elasticity" feature. That is, their algorithmic complexity adapts automatically to the number and capabilities of available CPU cores. In other words, the more CPU are allocated to a live video encoder, the higher the encoding performance. Until recently, the elasticity feature was used as an ad-hoc adaptation to uncontrollably varying conditions. In this paper, mechanisms allowing to take control of the computing resource are presented. Two real-time resource optimizations strategies are then proposed. The first one is based on video content complexity and manages the video head-end costs, while the second relates to audience measurements and targets network bandwidth usage optimization.**

***Index Terms*—Video compression, live encoding, Kubernetes, orchestration**

I. Introduction

In the field of video encoding, microservices architecture is becoming more and more beneficial over monolithic applications. The concept of microservices [1][2] allows a dramatic reduction of the design and implementation cycles durations and simplifies support and update of the applications. The virtualization concept on the other hand, allows being highly flexible and independent of the hardware. In the case of video compression, where performance is critical, the optimal granularity of the microservices must be optimized under constraints of real-time, low-latency, efficient data flow and availability. Practically, microservices must be stored in containers. The high number of containers requires orchestration. Among many available solutions [4][5][6], the work presented in this paper relies on Docker [7] for containerization and Kubernetes [5] for orchestration.

The video encoding solution considered in this paper is composed of several independent services which are thus managed by Kubernetes. However, the performance of a practical implementation of a video encoder is a trade-off between bitrate, perceived video quality, computing resource and architecture design. Kubernetes allows controlling the number of resources dedicated to each microservice. Thus, in the video compression context, one may consider allocating the resource non uniformly to different video services, depending on the desired trade-off for each video service. This must be carried out explicitly by the user though, since Kubernetes, as an orchestrator, is blind to the specifics of each application.

The proposed allocation solution will leverage previously introduced method [8] to seamlessly update the CPU for a service running on Kubernetes without service interruption. A full experimental system is demonstrated, applying the proposed dynamic resource allocation to a set of live encoders deployed in a Kubernetes environment. The rest of this paper is organized as follows: first, some elements of context and preliminary results are provided. Then two versions of the custom-orchestrator are detailed, complexity-based and audience-based. Finally experimental results are provided for each mode before conclusion.

II. Context, elasticity and CPU allocation

A given video encoder implementation can provide several trade-offs between resource consumption and video quality. This is the case, for example, with the High Efficiency Video Coding (HEVC) implementation x265 [9]. The tuning parameter (`-preset`) allows choosing a speed/coding performance trade-off in a range of predetermined settings. In this paper, the considered encoder adapts automatically to the available computing resources. That is, given the real-time constraint, the encoder chooses its parameters automatically depending on the platform capacity and current load. This tuning is updated dynamically. If the overall load of the platform changes, the tuning changes accordingly. The more computing resources available, the better the delivered coding efficiency. This concept is called video encoder elasticity [14].

As an illustration of the elasticity concept, example experiments have been conducted using the HEVC codec in its default configuration. All the considered video sequences have a 1080p (high definition, HD) resolution. Fig. 1 presents rate-distortion curves [12] for several encodings of the same 12 minutes movie extract. Each encoding is performed in real-time, with a fixed number of central processing unit (CPU) cores allocated to the corresponding microservice. In the video compression context, a rate-distortion curve illustrates the trade-off between bitrate and distortion (or quality) achieved by an encoder implementation or configuration. A configuration is found to be better than a reference configuration if its rate-distortion curve is above the reference rate-distortion curve. That is, for a given distortion, the bitrate is found to be lower, or conversely, for a given bitrate, the quality is found to be higher. The experimental observations confirm that the encoder adapts to the available

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

computing resource. Indeed, all the curves of Fig. 1 have been generated with strictly the same configuration, except for the number of CPU allocated. Thus, the rate-distortion performance improves as the CPU number increases.

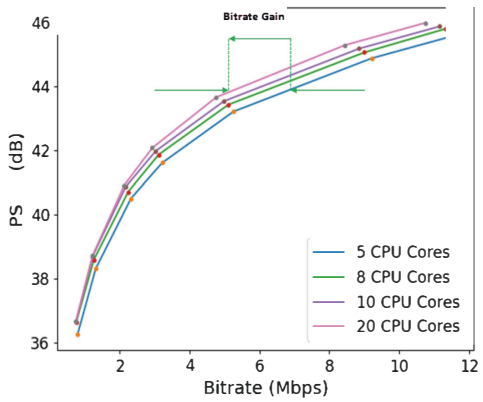

Fig. 1: Rate-Distortion curves for different CPU core allocations.

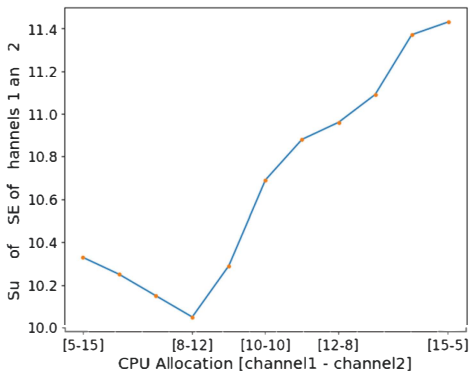

Fig. 2: Sum of Mean Squared Errors (MSE) for different CPU allocations among two video channels.

In a second experiment, the encodings of two different 12 minutes movie extracts are considered. The two contents have the same resolution and are both encoded using HEVC. An arbitrary fixed budget of 20 CPU cores is allocated to be shared between the two encoders. One must note that this fixed CPU budget is shared in a controlled manner between the two channels. A first part is allocated exclusively to the first channel, and the remaining part is allocated exclusively to the second channel. One may split it even and allocate 10 CPU cores to each channel or decide to allocate more CPU cores to one of the channels. The goal of this experiment was to find the optimal repartition of these 20 CPU cores between the two encoders, which minimizes the distortion for a given bitrate. The experiment showed that the allocation that maximizes the overall quality is not uniform, as illustrated on Fig. 2.

Both encoders have the same configuration, the difference is the encoded content itself. The channel 2 contains more complex content compared to channel 1. A video sequence is said to be more complex if it contains more information, like more motion or image texture, than the other sequence. The encoder must make more effort on a complex sequence to achieve the same coding efficiency as on a simple sequence.

III. Complexity Based Orchestration

A. *Dynamic CPU allocation*

The second experiment (Fig. 2) showed that for two channels with the same configuration, the allocation that minimized the distortion – thus maximizes the video quality – is not a uniform allocation, but rather a CPU cores distribution where the channel with high content complexity needs to be allocated more than the lower content complexity channel. Additionally, it is well known that the characteristics of contents are not constant in time. This is especially true for a 24/7 live channel. With a limited number of computational resources, dynamic resource allocation can improve the overall compression efficiency of a set of live channels.

The encoders run as part of a micro-services application in a Kubernetes cluster. All encoding services are running in Pods, the smallest Kubernetes manageable unit. A Pod contains one or several containers, and the hardware resources (CPU, memory, …) are managed at the container level. The native and supported way for Kubernetes to update the resources allocated to a container in a given Pod is to stop and restart the Pod with the desired resources allocation.

For a live video encoder, the reboot of the Pod even for milliseconds will lead to the loss of multiple video frames. However, service interruption of a live service is not acceptable. In a previous work [8] authors proposed a method for dynamic resource allocation for Kubernetes Pods with zero downtime.

The allocation system relies on an interaction between operating system features and Kubernetes device plugin feature [11]. It consists in updating the number of resources advertised to the Kubernetes scheduler and changing the current allocation using the Linux system tools in a way that is transparent to Kubernetes.

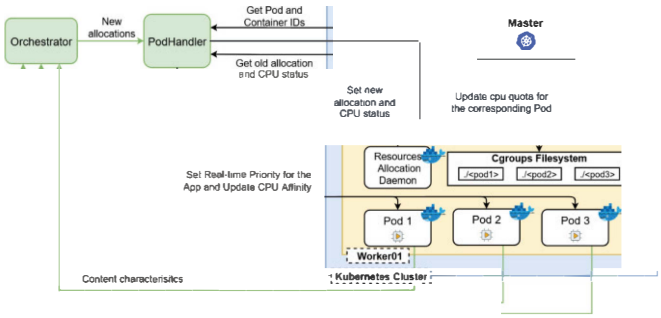

Fig. 3: Resources updating and orchestration process.

Fig. 3 present the interaction between Kubernetes cluster and the dynamic allocation service (PodHandler). The PodHandler gets the new allocation computed by the orchestrator, then interacts with the device plugin to update the number of custom resources advertised to Kubernetes Scheduler, the next step is to update the Pod's Cgroups [10] Completely Fair Scheduler Quota (CFS Quota) that controls the Pod's CPU usage limit. Linux tool taskset is used to change CPU affinity to meet the new allocation. Finally, the resource state is updated for every server in a database managed by the Resource Allocation Daemon service.

B. Complexity based orchestration

The orchestrator computes optimal CPU resource allocation and relies on the PodHandler to apply this allocation. The orchestration algorithm is organized in two steps:

- Predicting the bitrate gain with the help of a machine learning algorithm
- Computing the optimal allocation that minimizes the function (1), based on the bitrate gain predictions:

$$J = \sum_{i=1}^{N} b_i + \lambda * d_i , \quad (1)$$

where, N is the number of channels b_i is the bitrate of the channel i, d_i is its video distortion and λ is the Lagrange multiplier.

For the first step, the orchestrator uses a trained machine learning model that predicts for every channel the possible gain of a given CPU allocation with respect to a reference allocation, the model takes as input several parameters:

- Video Codec (HEVC, AVC, AV1…)
- Channel configuration (Frame rate, resolution, bit depth…)
- Video quality (PSNR)
- Number of CPU cores
- Channel's complexity estimation

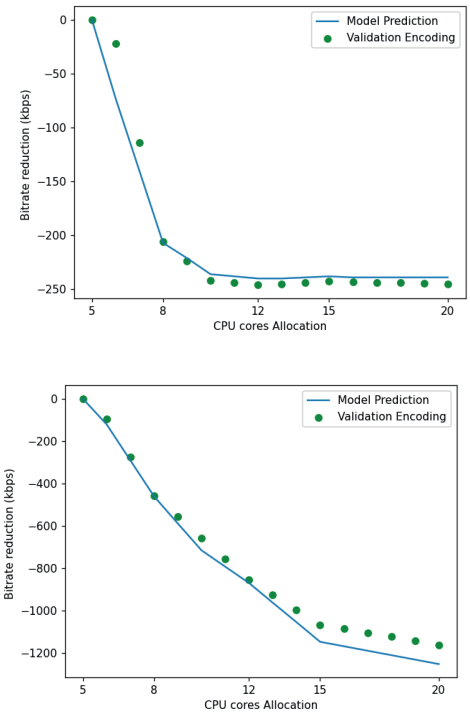

Fig. 4: Bitrate gain predicted by KNR model for, (a) low complexity channel, (b) high complexity channel.

A K-Neighbors Regressor (KNR) [18] algorithm is used for this task. The dataset is composed of various encodings with different configuration and content complexity. Fig. 4 illustrates the predictions made by the model for two different video sequences. The first sequence exhibits low complexity content and the second one high complexity. Both contents are encoded by an HEVC encoder and have the same configuration (25 fps, 1080p resolution, 8-bit depth…). The graphs in Fig. 4 show the KNR model estimation of the bitrate gain for a given CPU allocation with respect to the minimum allocation (5 CPU cores here), the video quality remaining constant. The complex channel takes better advantage of any additional CPU core. Additionally, from a CPU cores number threshold, additional CPU core will no longer provide bitrates reduction. This threshold is much higher for the high complexity channel than for the low complexity channel. Finally, the reliability of the prediction is assessed by comparing it to the actual encoder behavior.

In a second step, the result of gain estimation is used to compute the optimal allocation that will minimize the cost function. The algorithm proposed is a greedy algorithm, and since all curves predicted by the KNR model are strictly decreasing, it is guaranteed that it will return the optimal solution.

Let N be the number of channels in a given server, minAllocation returns the minimum allocation possible for a video channel to operate normally and M is the disputed CPU cores given in function (2):

$$M = totalCPUs - \sum_{i=1}^{N} minAllocation_i \quad (2)$$

Allocation Algorithm

Initialize with the minimum allocation
Output: allocation
For channel = 1 to N **do**:
  allocation[channel] <-- minAllocation[config]
End

For cpu <-- 1 **to** M **do**:
  For channel = 1 to N **do**:
    CurrentAlloc <-- allocation[channel]
    $gain[channel] <-- \boldsymbol{KNR}(currentAlloc, complexity, quality, config) + \lambda * d_i$
  End
ChosenChannel <-- argmax(gain)

allocation[ChosenChannel] = allocation[ChosenChannel] +1
End

Fig. 5: Allocation algorithm for complexity-based orchestration.

Since the model returns the bitrate gain for a constant PSNR, the distortion term in the algorithm is also constant, thus, the Lagrange multiplier can be put to 0.

The algorithm provided on Fig. 5 will return the allocation that minimizes the total bitrate of all channels while keeping the same video quality. Note that the number of allocated CPU cores to the channels is an integer number, in order to ensure optimal usage of threading and CPU cache memory. After receiving the number of cores, the PodHandler will take care of finding the best CPU affinity considering the NUMA architecture [13] of the physical processor.

C. Complexity-based orchestration experimental results

1) Bitrate minimization

Many tests have been conducted with various configurations and repeated to validate the stability of the

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

system running live. From a large set of varied sequences, several subsets have been selected to perform our experiments. Little variation in the results has been observed, as long as the subsets are heterogeneous. In a sense, the behavior of the system is comparable to a statistical multiplexer (statmux), as an allocation for a set of sequences having all the same characteristics brings little to no gain. The following example has been kept as a meaningful representative of these experiments.

Four 1080p channels of 5 minutes duration, encoded using an HEVC encoder, configured in constant quality mode and targeting the same video quality. This mode delivers variable bitrate (VBR) streams. Therefore, the performance at a given quality is measured by the bitrate. The better the allocation, the lower the bitrate. An arbitrary number of 28 CPU cores is available to be shared between the 4 channels. These channels have all different content complexity levels.

Two allocation scenarios are run and compared. The first is a uniform allocation, where every channel gets a fixed 7 CPU cores no matter its content. The second is dynamic allocation; in this mode, the orchestrator will compute the optimal allocation periodically based on the content complexity.

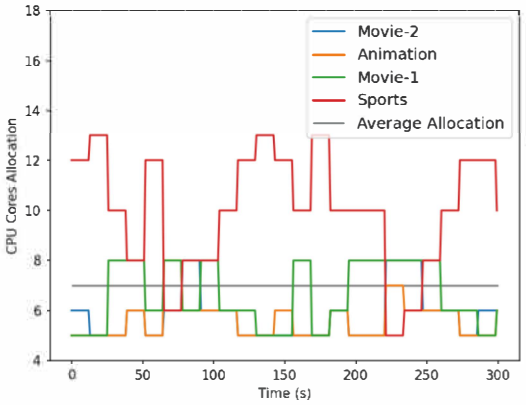

Fig. 6: Dynamic allocation of 28 CPU cores among 4 channels encodings.

Fig. 6 illustrates the changes of the allocation over time for the four channels depending on their respective complexities. As one may expect, the sports content is more complex than the others, hence a larger allocation has been granted to it almost all the time.

TABLE I: BDRATE GAINS COMPARED TO UNIFORM ALLOCATION.

	Movie-1	Movie-2	Sports	Animation	Mean
BDRate	2.93%	2.63%	-8.85%	1.23%	-0.51%

For the proposed combination of sequence and settings, rate distortion curves are derived from which Bjøntegaard Delta Rates (BDRates) [3] can be computed. Table I presents the BDRate gains relative to the uniform allocation. Negatives values indicate a gain (bitrate reduction), and positives values a loss. The first observation is that resource augmentation for one channel implies resource reduction for at least one other channel, leading to BDRate losses. Still, with the proposed dynamic allocation, an overall BDRate gain is achievable.

However, the BDRate is a relative performance metric especially when comparing sequences with different content types. The actual bitrates are provided in Table . The overall performance is measured by the sum of the bitrates for the 4 channels, with a lower total bitrate indicating better performance.

TABLE II: BITRATES IN MBPS FOR ALL RUNS AT THE SAME QUALITY.

	Uniform	Dynamic	Gain
Movie-1	0.686	0.709	3.35%
Movie-2	0.245	0.250	2.04%
Sports	4.015	3.469	- 13.6%
Animation	0.195	0.198	1.54%
Total	5.141	4.626	- 10%

Compared to uniform allocation, dynamic allocation reduces the bitrate by 10%. For the highest bitrate sequence, Sports, the required bitrate is reduced by 13.6% thanks to dynamic allocation, which represents more than 0.5 Mbps on a very demanding content. The absolute bitrate increase on the other channels is comparatively negligible. Gaining more than 0.5 Mbps on a channel is an opportunity to reach more users with the full resolution quality. For the content provider, it also translates into cost control. With uniform allocation, more CPU cores would be necessary to reach the same bitrate as the proposed solution, hence a higher cost. In a summary, this experiment showed 10% overall bitrate gain in dynamic allocation mode while using the same CPU budget and achieving the same video quality.

2) CPU Usage Optimization

In the previous experiment the goal was to allocate the available CPU cores in order to reduce the required bitrate at a given video quality. In a case where the aim is to minimize the encoding cost, i.e., to use less CPU cores (e.g., when using public cloud) or increase the channels density (have more channels in the same server), dynamic allocation allows reducing the total CPU cores required for a set of channels compared to the uniform allocation mode while achieving the same bitrate for the same video quality.

TABLE III: BITRATES (MBPS) FOR UNIFORM AND DYNAMIC ALLOCATION WITH DIFFERENT CPU BUDGET.

	Uniform 44 CPU Cores	Dynamic 28 CPU Cores	Gain
Movie-1	0.679	0.709	4.42%
Movie-2	0.246	0.250	1.63%
Sports	3.546	3.469	-2.17%
Animation	0.2	0.198	-1%
Total	4.671	4.626	-0.96%

The experiment setup is the same as the previous one, four live HD channels are encoded with an HEVC encoder in constant quality mode. For the uniform allocation, 44 CPU cores are allocated to the channels (11 cores for each). The Table III shows the bitrates achieved for a given video quality in the uniform and dynamic CPU allocation modes. For the dynamic allocation, a total budget of 28 CPU cores is allocated which is 36% less than the 40 CPU cores of the uniform allocation. Yet, a gain of 0,96% of required bitrate is achieved compared to the uniform allocation mode. In

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

summary, this experiment shows that one can save up to 36% of CPU cores when applying a content complexity aware dynamic allocation on a set of live channels in a public or private cloud.

IV. Audience and Complexity Based Orchestration

The complexity-based allocation method optimizes the video encoding performance in the video head-end. The result was a lower overall bitrate compared to a uniform allocation. However, some channels have seen their required bitrates increased because they are less complex. In a real use-case, some channels may be more popular than the others, so the bitrate gain, or loss, of a channel affects the video traffic on the network and is eventually multiplied by the number of viewers that are watching the channel.

In this chapter, a new method is introduced to take the channel audience into account in addition to the content complexity when computing the allocation. Just like complexity, the number of viewers of a live channel can change over time, therefore dynamic allocation is the more suitable allocation mode.

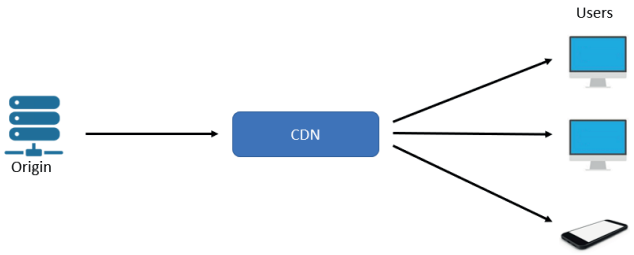

Fig. 7: Video streaming over CDN.

A. Video distribution network

Live Video streaming can be performed through different set-ups, either a digital video broadcast or an OTT (Over the Top) media streaming. A typical example is OTT streaming over a content delivery network (CDN) as presented in Fig. 7, which is one of the most used set-ups for live and VOD streaming (Video on Demand).

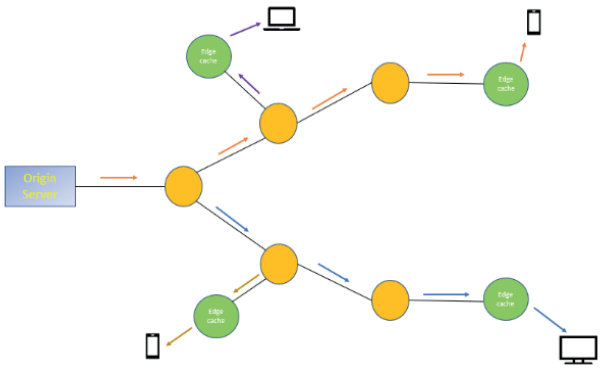

Fig. 8: Example of CDN architecture, with Origin server in blue, nodes in yellow and edge caches in green.

A CDN is a group of geographically distributed and interconnected servers, it provides cached content to the end users. In the field of video streaming, CDN is an essential component in the distribution scheme. Video channels require a high bandwidth to be transmitted, some channels can have thousands or possibly millions of viewers watching at the same time. The origin server usually has limited capabilities, thus cannot serve all the viewers, even if it can, the viewers could be distributed all over the world, the stream then should travel a long distance for every single viewer. Using a CDN, the stream is provided once by the origin and delivered to the edge cache servers in the viewers' regions, so all the viewers in the same geographic area can fetch the content from the nearest CDN edge cache (Fig. 8).

There are various solutions to get the audience measurements, from the CDN itself like Wowza Media System, which provides a near real-time API [16] to query the number of viewers for a given channel, or, directly from the players, as for example, Smart Sight API [17] by Media Melon, which gives real-time analytics collected from the players.

Allocation Algorithm

Initialize with the minimum allocation
Output: allocation
For channel = 1 to N **do**:
  allocation[channel] <-- minAllocation[config]
End

For cpu <-- 1 **to** M **do**:
  For channel = 1 to N **do**:
    CurrentAlloc <-- allocation[channel]
    $gain[channel] <-- (\mathbf{1} + v_i) * \boldsymbol{KNR}(currentAlloc, complexity, quality, config) + \lambda * v_i * d_i$
  End
ChosenChannel <-- argmax(gain)

allocation[ChosenChannel] = allocation[ChosenChannel] +1
End

Fig. 9: Allocation algorithm for audience and complexity-based orchestration.

B. Cost function optimization

The goal is to minimize the overall distributed data over the network by the streaming, going from the origin server where the channels are encoded, to the end users passing through the CDN edge cache servers, under the constraint of the video head end limited CPU resource. The cost function to minimize is given as (3):

$$\boldsymbol{J} = \sum_{i=1}^{N} \boldsymbol{b_i}\,(\mathbf{1} + \boldsymbol{v_i}) \; + \; \boldsymbol{\lambda} * \; \boldsymbol{v_i} * \boldsymbol{d_i} \qquad (3)$$

Where, $\boldsymbol{N}$ is the number of channels, $\boldsymbol{b_i}$ is the bitrate of the channel $\boldsymbol{i}$, $\boldsymbol{d_i}$ is its video distortion, $\boldsymbol{v_i}$ the number of viewers and $\boldsymbol{\lambda}$ is the Lagrange multiplier. The $\mathbf{1}$ in the term $(\mathbf{1} + \boldsymbol{v_i})$ corresponds to the stream distributed from the origin server to the CDN.

To optimize the function, the previously trained KNR model is still applicable. The greedy algorithm however

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

needs to consider the number of viewers of the channels. The updated algorithm is presented in Fig. 9.

TABLE IV: OVERALL BITRATES GENERATED BY TWO CHANNELS, FOR DIFFERENT VIEWERS DISTRIBUTIONS.

Sports-1 Viewers	Movie-3 viewers	Uniform (Gbps)	Dynamic (Gbps)	Gain (%)
0%	**100%**	602.51	577.02	-4.23
5%	**95%**	617.41	596.0	-3.47
25%	**75%**	676.99	651.5	-3.77
50%	**50%**	752.09	751.46	-0.08
75%	**25%**	825.94	790.15	-4.33
95%	**5%**	885.52	820.1	-7.39
100%	**0%**	900.41	829.24	-7.9

C. *Primary experiments*

Several experiments have been conducted to emphasize the importance of including the audience measurements along with the complexity to maximize the encoding performance in a cost-effective manner. In the first experiment, two HD channels that have a nearly equivalent complexity levels are encoded with an HEVC encoder in various scenarios. The first channel Sports-1 is a 5-minutes rugby match sequence, and Movie-3 is an action movie with the same duration. An arbitrary total number of viewers is taken as 100000 viewers for both channels. The considered scenario is that X% (percent) of the viewers are watching Sports-1 and (100 - X) % are watching Movie-3.

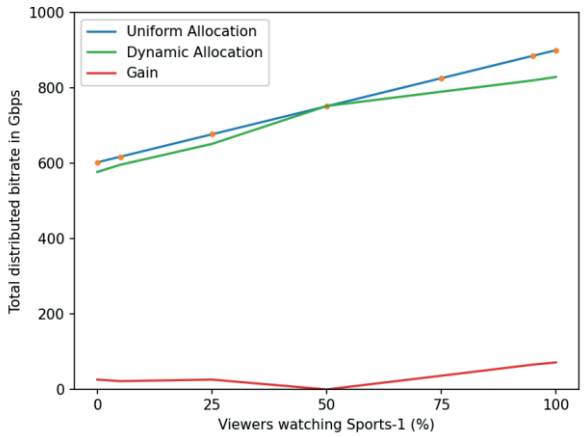

Fig. 10: Bitrate over the distribution network of two similar complexity channels with uniform and dynamic CPU allocation.

In all the different scenario presented in Table IV and Fig. 10, the proposed dynamic orchestrator managed to perform better than the uniform allocation, with a gain varying as a function of the viewers distribution. In the case where the two channels are equally popular, the number of viewers is the same and the complexity is equivalent, so the dynamic orchestrator will allocate approximately a uniform allocation and that explain why the gain is low. Also, the sequence Sports-1 is slightly more complex than Movie-3, that why the total bitrate and the gain are larger when the Sports-1 is the most viewed.

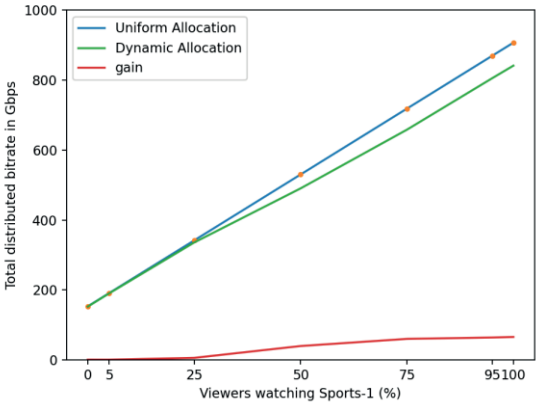

Fig. 11: Bitrate over the distribution network of two different complexity levels channels with uniform and dynamic CPU allocation.

The second experiment is conducted in the same configuration, two HD channels encoded with an HEVC encoder. However, the sequences in this experiment have different complexity levels. The first sequence is Sports-1, and the second is Movie-1, a historical movie. The results are shown in Fig. 11. In the case of equal popularity, the orchestrator still reduces the overall bitrate, because of the complexity difference. Moreover, when the less complex channel is the most viewed, there is always a gain with respect to uniform allocation even if it's smaller.

D. *Full-scale experimental results*

Here, a real use-case is simulated, where a content or service provider has a set of live channels distributed to its subscribers, each channel may be watched more or less than the others. In this experiment four HD channels are encoded with an HEVC encoder with the same configuration. The video sequences used have different intrinsic content complexity levels. The goal is to reduce the overall bitrate generated by the channels over the distribution network (CDN) when applying audience and complexity based dynamic allocation compared to a uniform static allocation.

Several scenarios are considered where the distribution of viewers is different. The total number of viewers is 100000, the different distributions of the viewers over the 4 channels are provided in Table V.

TABLE V: VIEWERS DISTRIBUTIONS SCENARIO DESCRIPTION.

	Sports	Movie-1	Movie-2	Animation
Scenario-1	70000	10000	10000	10000
Scenario-2	10000	70000	10000	10000
Scenario-3	10000	10000	70000	10000
Scenario-4	10000	10000	10000	70000

In the first scenario, the channel **Sports** is considered the most watched, with 70% of total number of viewers, while the other channels get each 10% of the total viewers. The results are presented in Table VI, corresponding to the total data generated by the four channels. The dynamic allocation mode reduced this number by 13.33%, it is 45.95 Gbps

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5

(Gigabit per second), and more than 14% of the most popular channel spared just by changing the way that the available CPU cores are allocated to the channel.

TABLE VI: TOTAL DATA GENERATED BY THE CHANNELS IN (MBPS).

	Uniform	Dynamic	Gain
Movie-1	7812	8386	7.35
Movie-2	2920	2998	2.66
Sports	331813	285137	- 14.07
Animation	2163	2237	3.42
Total	344708	298758	- 13.33

The other scenarios have been tested as well; the results are summarized in Table VII. In all the presented use cases, the orchestrator succeeds in finding the optimal allocation that reduces the total bitrate. The most popular channel will have a larger weight (3), and consequently may be allocated more CPU cores. The gain is maximal when the complex channel is the most viewed, this is the expected behavior as illustrated in Fig. 2, where the potential bitrate gain increases with the content complexity.

TABLE VII: PERCENTAGE OF TOTAL BITRATE REDUCTION MADE IN DYNAMIC CPU ALLOCATION COMPARED TO UNIFORM CPU ALLOCATION.

	Scenario-1	Scenario-2	Scenario-3	Scenario-4
Movie-1	7.35	-8.12	0.06	0.63
Movie-2	2.66	12.37	-8.84	7.19
Sports	- 14.07	-6.73	-6.46	-6.56
Animation	3.42	1.84	-0.34	-1.06
Total	- 13.33	-5.42	-7.41	-2.01

V. CONCLUSION

In this paper, a new method is introduced for computing the CPU allocation for live video encoders. It is demonstrated that dynamic allocation is more suitable and give a significant bitrate reduction compared to a uniform static allocation. In a first mode, complexity aware dynamic orchestration showed a gain up to 13.6% on a very demanding content, and an average of 10% reduction of the overall bitrate required by four channels. The second proposed mode considers the number of viewers for each channel in an OTT streaming environment. This method offers further gains, with more than 14% reduction of the overall bitrate distributed over the network by a complex channel, and an average of 13.33% compared to a uniform allocation.

Finally, the proposed method needs 36% less CPU cores compared to a uniform allocation to achieve the same video quality at the same bitrate, which could reduce the operational costs significantly.

REFERENCES

[1] N. Dragoni et al, Microservices: yesterday, today, and tomorrow 2016. https://doi.org/10.48550/arXiv.1606.04036
[2] Francesco et al, Architecting with microservices: A systematic mapping study. https://doi.org/10.1016/j.jss.2019.01.001
[3] G. Bjøntegaard, Calculation of average PSNR differences between RD-curves, Technical Report VCEG-M33, ITU-T SG16/Q6, Austin, Texas, USA, 2001.
[4] Docker Inc., Docker Swarm, Container's orchestrator. https://docs.docker.com/engine/swarm/key-concepts/
[5] The Linux Foundation , Kubernetes Platform Container's Orchestrator. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
[6] The Apache Software Foundation , Mesos A distributed systems kernel. http://mesos.apache.org/
[7] Docker Inc , Docker Software containerization platform. https://docs.docker.com/get-started/overview/
[8] A. Moussaoui, M. Raulet and T. Guionnet, "Dynamic Seamless Resource Allocation for Live Video Compression on a Kubernetes Cluster," in SMPTE Motion Imaging Journal, vol. 131, no. 4, pp. 45-49, May 2022, doi: 10.5594/JMI.2022.3160832.
[9] MulticoreWare Inc, https://x265.readthedocs.io , x265 HEVC implementation.
[10] Serge Hallyn, Linux Control Groups File System https://man7.org/linux/man-pages/man7/cgroups.7.html
[11] The Linux Foundation , Kubernetes device plugin https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
[12] Yochai Blau https://onlinelibrary.wiley.com/action/doSearch?ContribAuthorRaw=Berger%2C+Toby, Rethinking Lossy Compression: The Rate-Distortion-Perception Tradeoff https://doi.org/10.1002/0471219282.eot142.
[13] Linux Kernel Organization, NUMA https://www.kernel.org/doc/html/v4.18/vm/numa.html
[14] Herbest et al, Elasticity in Cloud Computing: What It Is, and What It Is Not. https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst
[15] J. Vanne et al. Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs
[16] Wowza Inc. Viewers data API https://www.wowza.com/docs/how-to-get-viewer-data-for-a-wowza-cdn-stream-target-by-using-the-wowza-streaming-cloud-rest-api
[17] MediaMelon Inc. Players QoE analytics API https://www.mediamelon.com/product-smartsight-qoe
[18] Kramer, O. (2013). K-Nearest Neighbors. In: Dimensionality Reduction with Unsupervised Nearest Neighbors. Intelligent Systems Reference Library, vol 51. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-38652-7_2

Abdelmajid Moussaoui received an engineering degree in multimedia networking from Telecom-Paris, Paris, France, in 2020. He is a research engineer in the Research and Innovation Department of ATEME. His current research areas are video encoding, codec orchestration in the cloud, and machine learning. He holds several pending patents related to the optimization of the orchestration of live video channels distribution in the cloud.

Thomas Guionnet is a fellow research engineer at ATEME, where he currently leads the innovation team's research on artificial intelligence applied to video compression. Beyond his work for ATEME, he has also contributed to the ISO/MPEG - ITU-T/VCEG - VVC, HEVC, and HEVC-3D standardization process; he teaches video compression at theESIR Engineering School, Rennes, France; and he has authored numerous publications including patents, international conference papers, and journal articles. Prior tojoining ATEME, he spent 10 years at

Envivio conducting research on real-time encoding, video-preprocessing, andvideo quality assessment. He holds a PhD from Rennes 1University, Rennes.

Mickaël Raulet is the chief technology officer at ATEME, where he drives research and innovation with various collaborative research and development projects. He represents ATEME in several standardization bodies: ATSC, DVB, 3GPP, ISO/IEC, ITU, MPEG, DASH-IF, CMAF-IF, SVA, and UHDForum. He is the author of numerous patents and more than 100 conference papers and journal scientific articles. In 2006, he received a PhD from INSA in electronic and signal processing, in collaboration with Mitsubishi Electric ITE, Rennes, France.

Received in 2022-07-08 | Approved in 2022-08-17

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.5 Web Link: http://dx.doi.org/10.18580/setijbe.2022.5