Dynamic Seamless Resource Allocation For Live Video Compression On A Kubernetes Cluster

Abdelmajid Moussaoui

ATEME, a.moussaoui@ateme.com.

Mickaël Raulet

ATEME, m.raulet@ateme.com.

Thomas Guionnet

ATEME, t.guionnet@ateme.com.

Written for presentation at the

SMPTE 2021 Annual Technical Conference & Exhibition

Abstract. *A solution is proposed on top of Kubernetes to dynamically allocate services resources without service interruption. It serves as the basis for optimizing a live video compression service. It is demonstrated that dynamic resource allocation can benefit to a video compression application, either by reducing the resource consumption, hence costs, or by enhancing delivered video quality. By combining the proposed solution with an elastic encoder and machine learning for content complexity estimation, a content and application aware dynamic resource orchestrator for real-time video compression is designed. Preliminary experimental results using ATEME Titan Live Micro-services [8] encoders demonstrate substantial bitrate reductions on even the most demanding channel.*

Keywords. Video compression, machine learning, AI, Cloud, Kubernetes, Video Coding, HEVC, Video quality, Content Adaptive.

Introduction

In the field of video encoding, a microservices architecture is becoming more and more beneficial for the advantages realized 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.

Kubernetes is an open-source containers orchestration platform, initially developed by Google [12] in 2014 and was later transferred to the Cloud Native Computing Foundation (CNCF). Kubernetes manages clusters of physical or virtual machines. It schedules containers to run on machines based on their available resources and the requirements of each container. It is widely used for management of public and private cloud services.

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 amount 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.

In this paper, it is proposed to complement Kubernetes with a custom-made orchestrator, with the goal of maximizing perceived video quality while minimizing the computing resources consumption, hence exploitation costs. It is demonstrated through experimental results that the full potential of the approach requires dynamic resource adaptation to be reached. However, to change the resources allocated to a service, Kubernetes requires to stop and restart the service, causing live service interruption. The main contribution of this paper is a new mechanism allowing seamless dynamic resource allocation in the Kubernetes context. 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. The proposed system is then outlined, and the dynamic allocation mechanism is presented. Finally, experimental results are provided before conclusion.

Context And Preliminary Experiments

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 [15].

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. Figure 1a presents rate-distortion curves [13] 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 computing resource. Indeed, all the curves of Figure 1a 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.

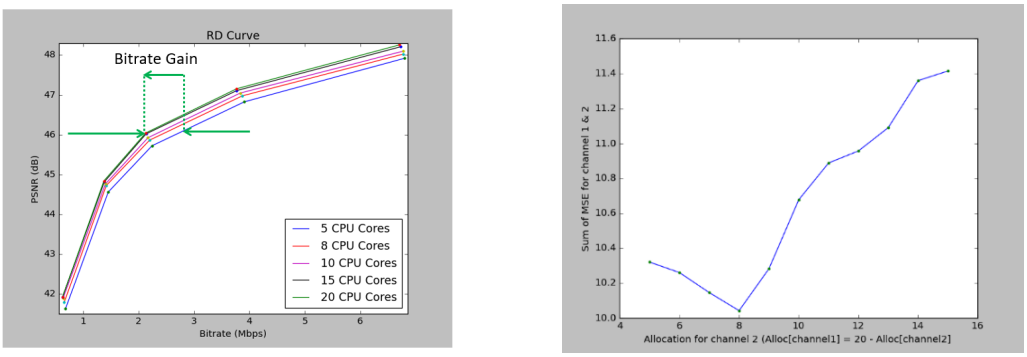

Figure 1 (a) Rate-Distortion curve for different CPU core allocations, (b) Sum of Mean Squared Errors (MSE) for different CPU allocations among two video channels.

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 Figure 1b. 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. Based on these two preliminary experiments, it is advocated that with a limited amount of resources, dynamic resource allocation can improve the overall compression efficiency of a set of live channels.

Proposed orchestration system

A general live video encoding orchestration system is outlined on Figure 2. A Kubernetes cluster manages the encoders deployed in Pods, the smallest Kubernetes manageable unit. In addition to this baseline structure, a “Pod Handler” module is responsible for seamless dynamic resource allocation, following the resource allocation instructions computed by the Orchestrator module. The latter computes optimal allocations based on the states of the encoders, which is updated in real-time. The dynamic allocation system is detailed in the next section, while other elements are outlined below.

The content characteristics are computed in the encoder lookahead. They are an estimate of the relationship between bitrate, CPU resource and video quality, obtained through a machine learning (ML) process. The orchestrator receives the content characteristics from the encoders and computes an optimal allocation depending on the total available computing resource and the optimization target. It is worth noting that several strategies are possible. One may minimize costs, while another may maximize video quality under resources constraint. The update frequency of the allocation is a tunable parameter.

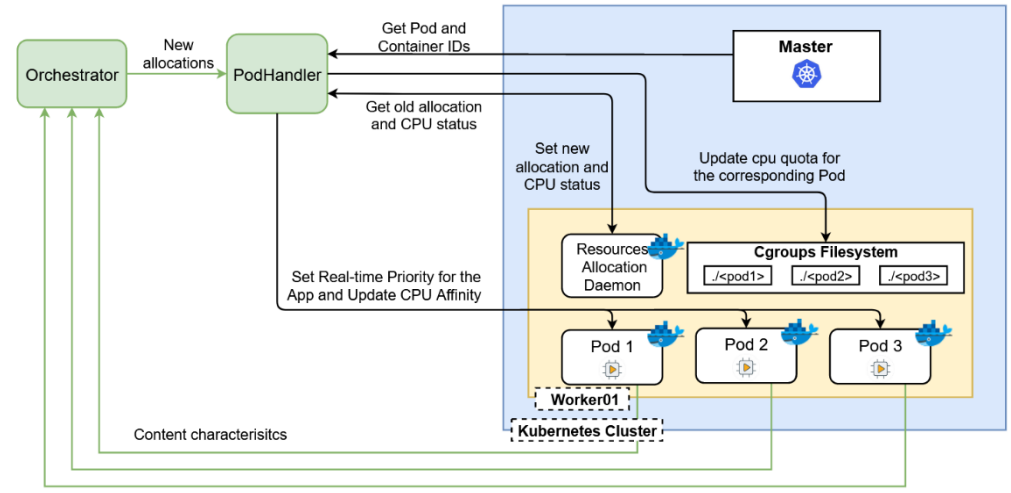

Figure 2: Resources updating and orchestration process.

Dynamic resource allocation

Seamless dynamic allocation is a key enabler for the proposed system. Kubernetes allows control of the allocation of Pod resources (each encoder runs in a Pod), but Kubernetes does not allow the system administrator to change them dynamically without service interruption. To overcome this problem, the proposed method relies on an interaction between operating system features and Kubernetes device plugin feature [11]. It consists of 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.

On Linux servers, for each instantiated Pod, Kubernetes creates several files in the control groups (cgroups) [10] file system to handle the resources allocated to the Pod. The file CPU Completely Fair Scheduler Quota (CFS Quota) is used to control the Pod’s CPU usage limit,

which represents the total amount of time the tasks of the Pod are allowed to run in a period. This period is defined in another file: cpu.cfs_period_us. For example, for a CFS period of 100 ms, if the limit is 50 ms, that means 50% of a CPU Core is available to the Pod. In this paper, only integer numbers of CPU Cores are considered, so CPU limit should be a multiple of the period.

The Linux kernel monitors and enforces the CPU usage limit. The CFS Quota file can be updated dynamically to change the allocation for a running Pod. The device plugin is used to keep the Kubernetes scheduler informed that the number of CPU available has changed. The device plugin is a feature implemented on top of Kubernetes to advertise custom resources, like GPUs or FPGAs, that Kubernetes does not support by default.

After changing the CPU limit to the desired number of CPU Cores, a set of CPU cores is selected by the PodHandler (Figure 2) for the Pod. These cores are chosen when it's possible to be in the same physical CPU socket (Non-Uniform Memory Access [14] - NUMA - aware affinity) to improve the performance of the encoder. The final step is to set the new affinity within the Pod to change the cores that the application is running on, and to set the Linux real time priority to the threads of the application for them to have exclusive access to that CPU cores.

Experimental results

The proposed method has been implemented on a private cloud in ATEME premises. Many tests have been conducted with various configurations and repeated to validate the stability of the system running live. From a large set of varied sequences, several subsets have been picked to perform our experiments. Little variation in the results has been found, 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 exactly the same characteristics brings little to no gain. The following example has been kept as a meaningful representative of these experiments. Four HD 1080p live channels are configured in constant quality (CQ) mode, all targeting the same visual quality. This mode delivers variable bit-rate (VBR) encodings. Therefore, the performance at a given quality is measured by the bitrate. The better the allocation, the lower the bitrate. An arbitrary number of 40 CPU cores is available to be shared between the 4 channels.

Three runs are conducted:

- Uniform allocation: the same computing resource is statically allocated to each channel.
- Static allocation: resource allocation is static, though non uniform. Allocation is hand-tuned to provide optimal performance.
- Dynamic allocation: resource allocation is performed automatically and dynamically using the proposed orchestrator. The optimized resource allocation is shown in Figure 3.

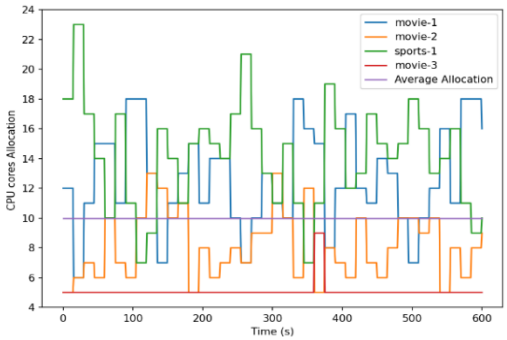

Figure 3: Dynamic allocation of 40 CPU cores among 4 channels encodings.

For each combination of sequence and settings, rate distortion curves are derived, from which Bjøntegaard Delta Rates (BDRates) [3] can be computed. Table 1a presents the BDRate gains relative to the uniform allocation. 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 which does not provide the full picture in this context. The bitrates for a same level of quality are presented in Table 1b. The overall performance is measured by the sum of the bitrates for the 4 sources, with a lower total bitrate indicating better performance. Compared to uniform allocation, static allocation decreases the total bitrate by 8.2%. The dynamic allocation further reduces the bitrate by 1.2%, that is, 9.4% reduction compared to uniform allocation. For the highest bitrate sequence, Sports-1, the bitrate is reduced by 12.8% thanks to dynamic allocation, which represents almost 1Mbps on a very demanding content. The absolute bitrate increase on the Movie-2 and Movie-3 contents is comparatively negligible. Gaining 1 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.

Table 1: (a) BDRate gains compared to uniform allocation, (b) Bitrates in Mbps for all runs at the same quality.

	STATIC	DYNAMIC
Movie -1	-9.09%	-8.49%
Movie-2	16.19%	5.99%
Sports -1	-9.65%	-9.38%
Movie -3	9.60%	8.58%
Mean	**1.76%**	**-1.07%**

	Uniform	Static	Dynamic
Movie-1	4.70	4.22	4.25
Movie-2	1.13	1.31	1.19
Sports-1	7.40	6.51	6.45
Movie-3	0.54	0.58	0.58
SUM	**13.76**	**12.63**	**12.47**

The difference between static and dynamic allocation may seem limited. However, for practical reasons, the experiment is conducted on 10 minutes long contents. On this short duration, the characteristics of each content varies, but remains in a limited interval, as illustrated by the corresponding CPU allocation (Figure 3). Therefore, a more dramatic difference is expected in production, but will need to be demonstrated empirically. Additionally, the channels contents complexity can change over time, hence hand tuned content aware static allocation is unusable in production.

Conclusion

A content and application aware, purpose-built dynamic resource orchestrator for real-time video compression in the cloud has been proposed and demonstrated. The key factor in the system is a method that allows changing the CPU resources allocated by a Kubernetes cluster to a video encoder without interrupting live transmission. The benefits of the system are demonstrated through experimental results. Up to 12.8% bitrate reduction is demonstrated on a complex sport channel, with an average bitrate reduction of 9.4% on a pool of 4 mixed characteristics channels.

The experiments have been conducted in a private cloud. Many runs with various parameters have been performed in order to assess the stability of the system and to provide meaningful performance measurement across a large range of rate/quality settings. On that basis, the next step is naturally to move on to real production set-ups. Future works also include refinement of the orchestration algorithm and refinement of the content characteristics estimation ML based tool, as well as the demonstration of multiple optimization use cases.

References

1. N. Dragoni et al, Microservices: yesterday, today, and tomorrow. https://arxiv.org/abs/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. ATEME , ATEME Virtualized Video Solution, https://www.ateme.com/wp-content/uploads/2021/02/ATEME-VIRTUALIZED-VIDEO-SOLUTION.pdf 2021.
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. Abhishek Verma et al, Large-scale cluster management at Google with Borg, https://arxiv.org/pdf/1901.07821.pdf
13. Yochai Blau, Rethinking Lossy Compression: The Rate-Distortion-Perception Tradeoff https://doi.org/10.1002/0471219282.eot142.
14. Linux Kernel Organization, NUMA https://www.kernel.org/doc/html/v4.18/vm/numa.html
15. 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

