

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

Abdelmajid Moussaoui  
a.moussaoui@ateme.com

Thomas Guionnet  
t.guionnet@ateme.com

Mickael Raulet  
m.raulet@ateme.com



18/11/2021

# Table of content

1. Introduction
2. Context and Preliminary Experiments
3. Proposed Solution
4. Experimental results
5. Conclusion and Future Work

# 1. Introduction

Use case:

- Live video encoding in the cloud using microservices applications.

Goal:

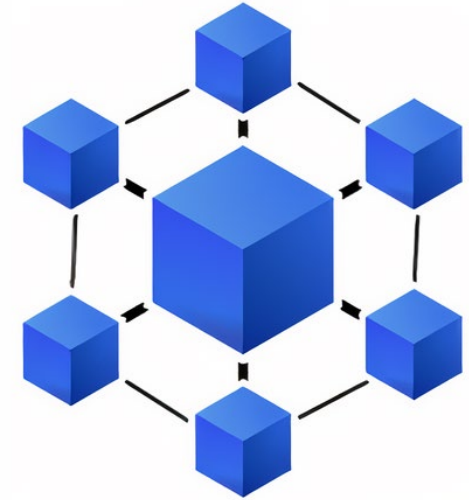
- Manage in real-time CPU resources allocated to a given encoder

# 1. Introduction: Microservices Applications Overview

Microservices architecture is an organizational method, where an application is composed of several independent services communicating via APIs. This architecture present various benefits:

- Design and implementation time reduction
- Simplifies application component's update
- Allows scalability and flexible implementation
- Virtualization native, independent of the hardware

Need for an orchestrator to manage the application and take advantage from all benefits.



# 1. Introduction: Kubernetes

The application services are Docker containers and they're orchestrated by Kubernetes:

## **Some of Kubernetes features:**

- Resource management for each service (Pod)
- Horizontal and vertical autoscaling, and load-balancing
- Automated deployment

## **Limitation:**

- Kubernetes updates resources by stopping and restarting the service

## **Proposed solution:**

- Use Linux system tools to update resources allocated to a service without interruption in transparent way to Kubernetes

## 2. Codec Elasticity

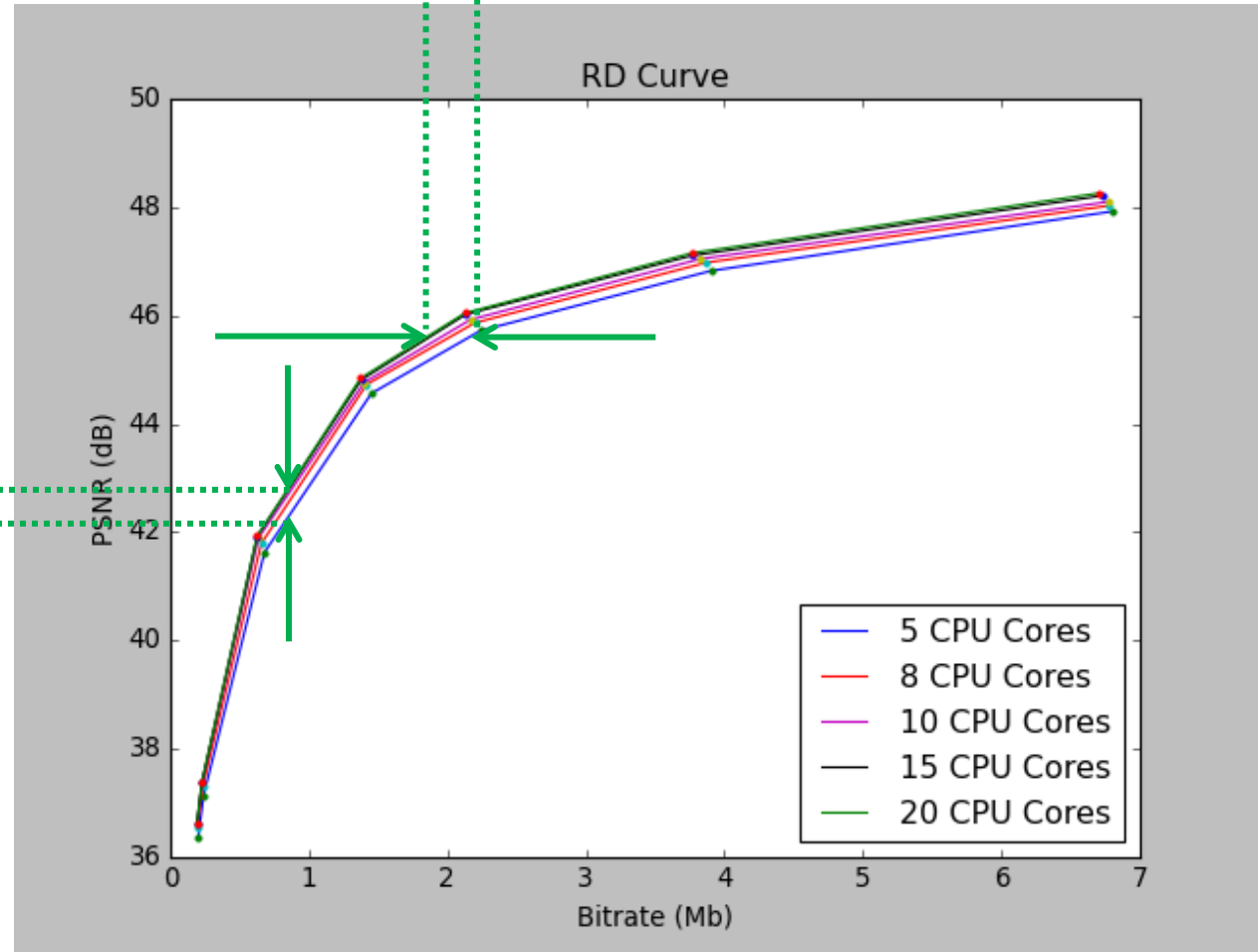
- A given video encoder implementation can provide several trade-offs between resource consumption and video quality
  - Fast mode / limited coding efficiency
  - Slow mode / best coding efficiency
- Live encoder implementation takes real-time constraint into account to adapt dynamically its video quality depending on the computational resources available.
- More CPU cores available means more parallel threads the encoder can launch and more search for better or optimal encoding point.

# 2. Preliminary Experiments: Codec Elasticity

- Several encodings of the same channel, with different CPU cores allocations
- The more CPU cores are allocated, the better the coding

PSNR / MSE gain between 5 and 20 CPU cores

Bitrate gain between 5 and 20 CPU cores



## 2. Preliminary Experiments: Optimal Allocation

When running several channels with different contents. An optimal CPU allocation exists and not necessarily a uniform allocation.

Two channels with different content complexity:

**Channel 1:** High complexity movie

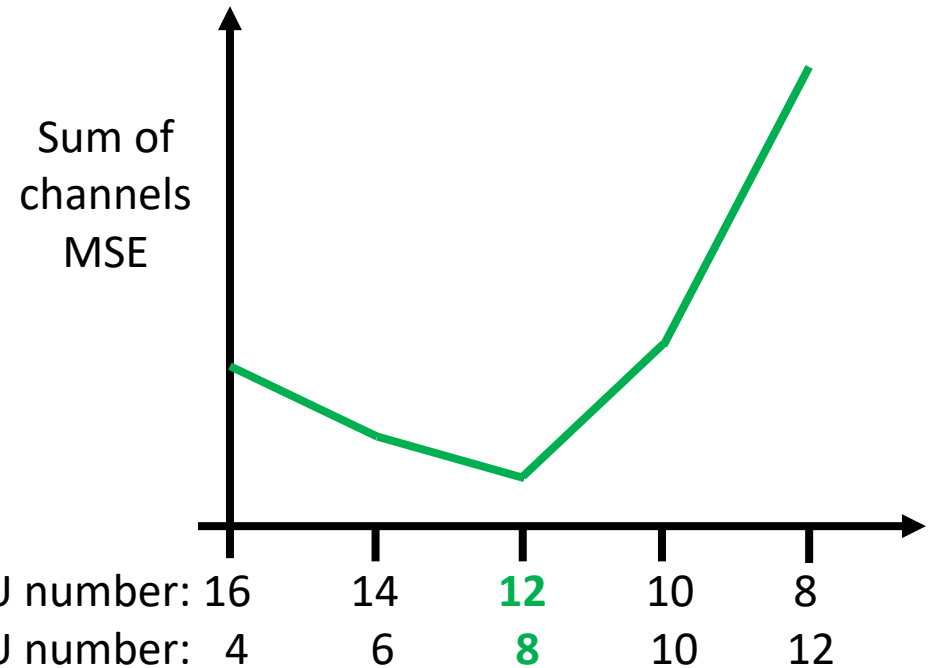
**Channel 2:** Low complexity movie

**Total CPUs Cores:** 20

**Criteria:** Minimize the distortion (MSE)

**Optimal allocation:**

- Channel 1 : 12 CPU Cores
- Channel 2 : 8 CPU Cores





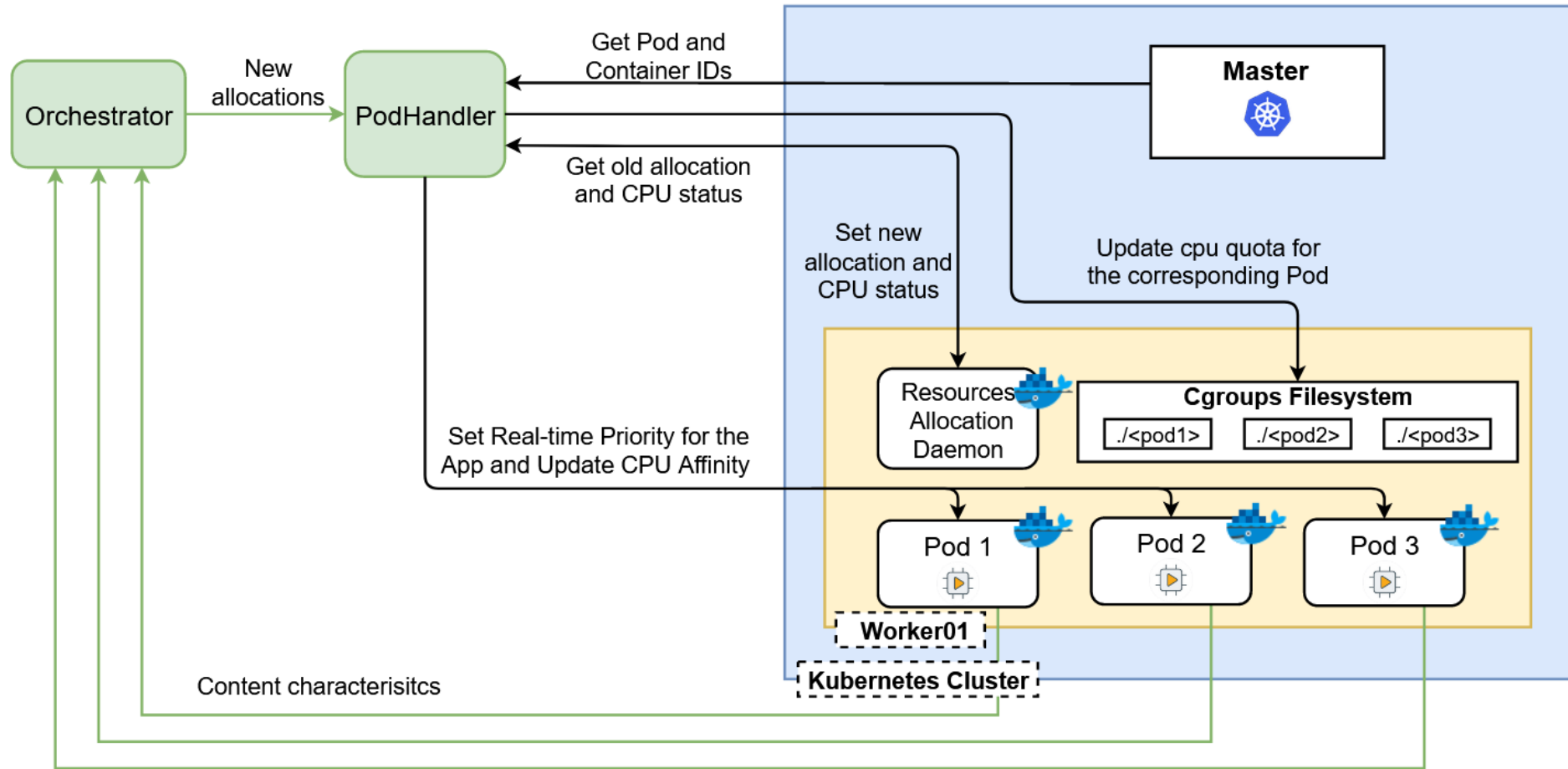
# 3. Proposed Solution: Dynamic Resource Allocation

Live video changes complexity over time, hence the need for an alternative to Kubernetes native process for updating the resources, allowing seamless resources updates, i.e. without restarting.

Proposed solution based on Linux Kernel tools:

- Cgroups Filesystem
  - Stores resources limits and status associated to processes
- CPU Affinity
  - Bind a process to a set of CPU Cores
- Linux real-time priority
  - Gives the process exclusive access to the CPU Cores
- And Kubernetes feature Device Plugin
  - Advertises to Kubernetes Scheduler the number of CPU Cores available

# 3. Proposed Solution: Dynamic Resource Allocation



# 3. Proposed Solution: Dynamic Resource Allocation

Updating the allocation for a Pod by the PodHandler goes through different steps:

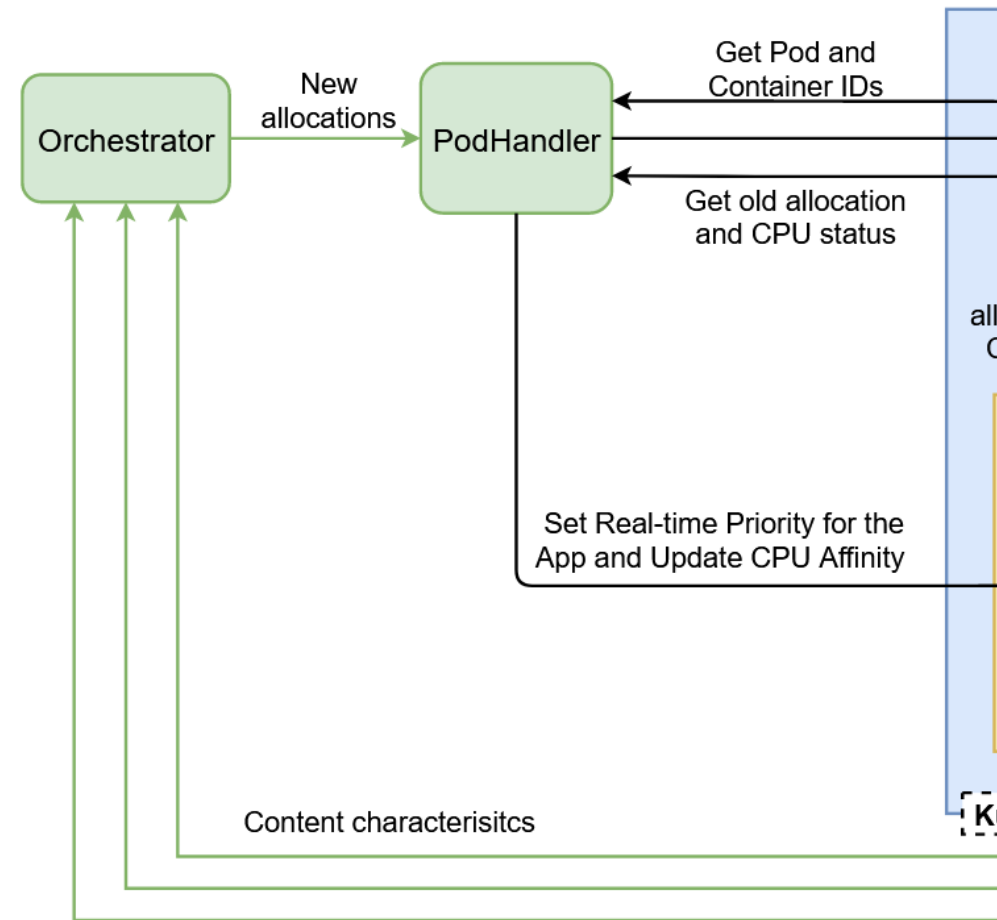
- Receiving the new allocation computed by the orchestrator
- Get the pod and container IDs to locate the corresponding files in the Cgroups
- Check if the new allocation is possible, using the custom service Resource Allocation Daemon
- If So, update the CPU usage limit in the Cgroups Filesystem
- Keep Kubernetes Scheduler informed by updating the CPU number advertised by the Device Plugin
- Change the CPU Cores Affinity and give the application exclusive access to these Cores by setting the Linux real-time priority.

# 3. Proposed Solution: Custom Orchestrator

The uniform allocation is not the optimal one, hence the introduction of a custom orchestrator that is content aware. It computes the allocation dynamically for all the Pods (Encoders) based on:

- Video sequences complexity
- Total number of CPU Cores available

For live encoding, the content complexity is computed in the encoder lookahead.

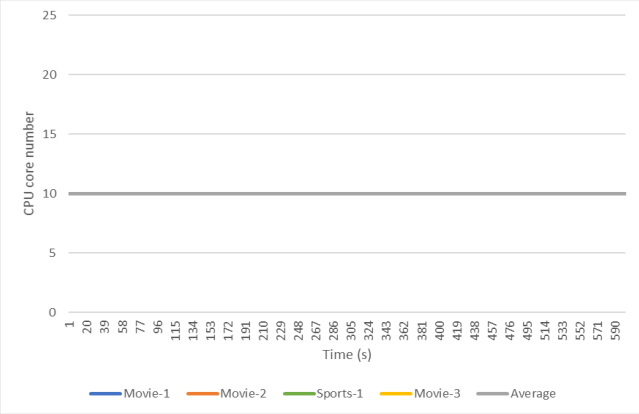
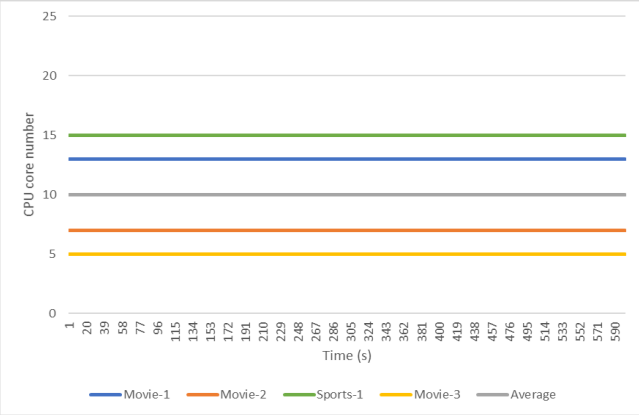
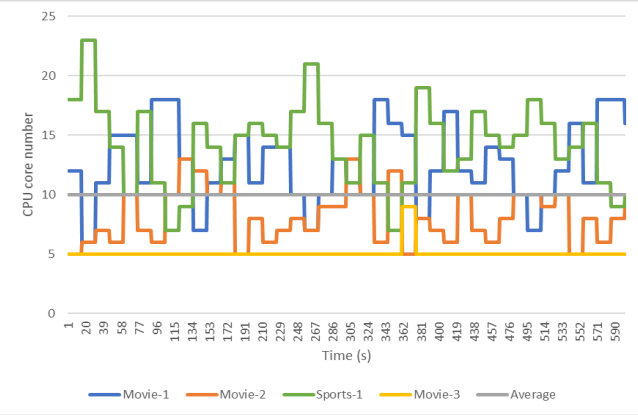


# 4. Experimental settings

- Titan Live Micro Services (MS)
- HEVC
- Constant Quality (CQ)
  - All channels targeting the same visual quality
  - Hence VBR
  - The better the encoding, the lower the bitrate
- Four 1080p channels
  - Movie-1 (Action)
  - Movie-2 (Historical fiction)
  - Sports-1 (Rugby)
  - Movie-3 (Drama)
- Total number of CPU Cores: 40
- **Goal: Demonstrate an overall bitrate gain**
  - At the same perceived video quality



# 4. Experimental settings

CPU cores allocation	Static	Dynamic
Uniform	<p>Reference</p> 	<p>N/A</p>
Non-uniform	<p>Hand-tuned, for Experimentation only</p> 	<p>Proposed method</p> 

# 4. Experimental results

BITRATES (MBPS)	UNIFORM	NON-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

Slight bitrate increase on already cheap channels (60kbps)

12,8% bitrate reduction on the costliest channel (almost 1 Mbps)

8,2% overall bitrate reduction

9,4% overall bitrate reduction

# 5. Conclusion and Future Work

- Seamless dynamic resource allocation for live video channels
- Proposed orchestrator and pod-handler going beyond Kubernetes capabilities
- Up to 12,8% bitrate reduction on lab experiment
- **Visual inspection of the channels confirms perceived quality stability**
- Future-works
  - Demonstration in production set-ups
  - Demonstration of multiple optimization use cases
  - Refinement of orchestration algorithm



THANK YOU