

# NOVEL ARCHITECTURE FOR MULTIMEDIA HARDWARE ACCELERATION

Universidade de Vigo

AUTHOR: GHOFRANE EL HAJ AHMED

THESIS ADVISOR: FELIPE GIL CASTIÑEIRA

ENRIQUE COSTA MONTENEGRO



PhD Programme on Information and Communications Technology (Doc\_TIC)

## Motivation of the work

The technology related to the cloud computing and the GPUs has been evolving in the last years.

Containers are rapidly replacing Virtual Machines (VMs) as the compute instance of choice in cloud-based deployments.

The mechanism of container management and orchestration have to be developed in order to provide new services and guarantee the quality of experience for users and minimize the cost and the power consumption.

## Thesis Objectives

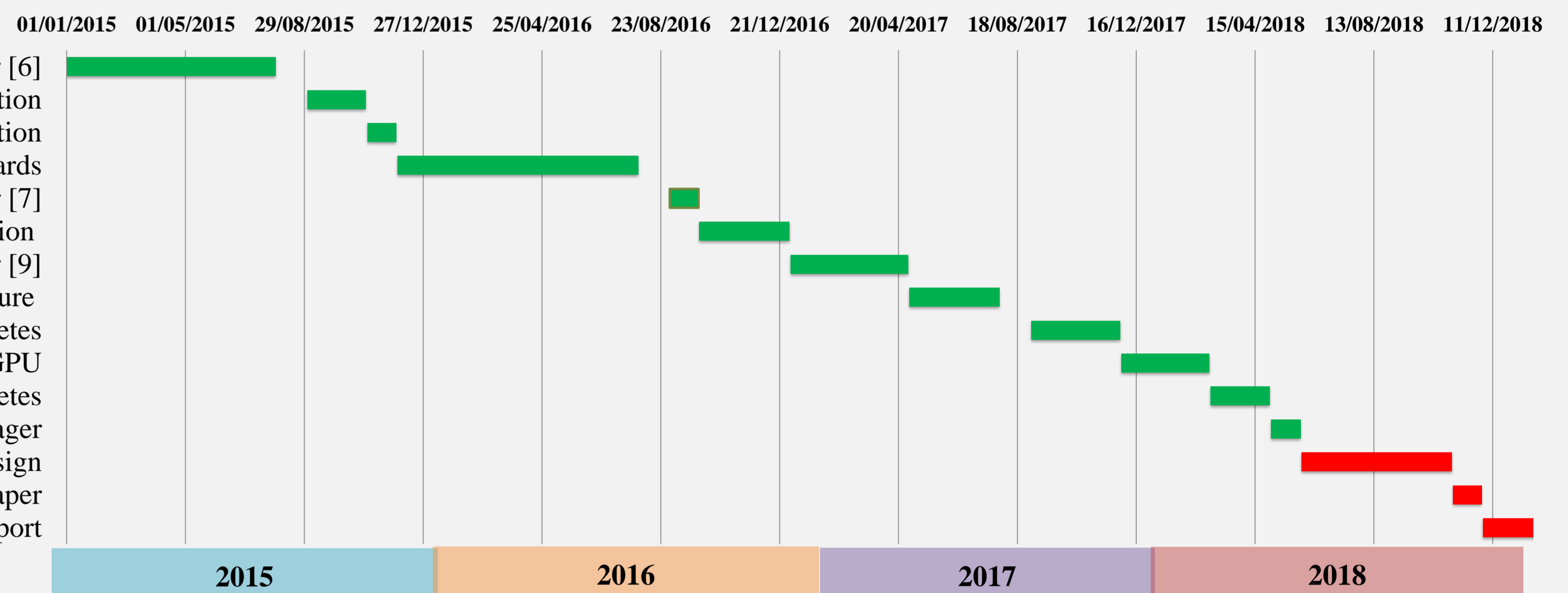
Test the performance of the GPUs on the embedded boards.

Design a new architecture of GPU manager and scheduler in cloud computing.

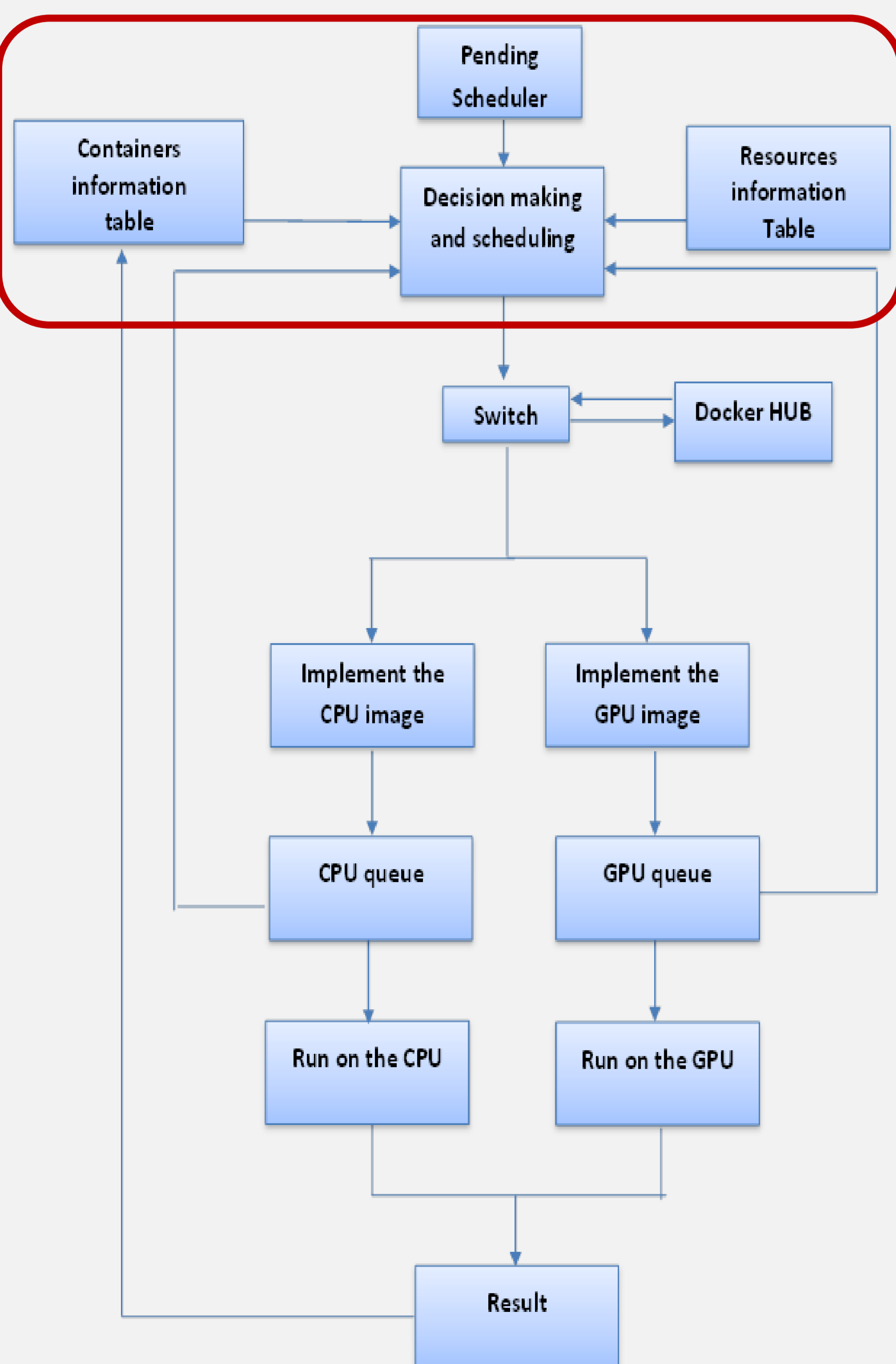
Modeling the GPU task on Docker [1] and Kubernetes [2] platform.

## Research Plan & Next Year Planning

Establish an essential knowledge of Gstreamer [3], the hardware acceleration [4][5] and the media server [6]  
 Review the data sheets of different manufactures of hardware video acceleration  
 Initial design of architecture for multimedia hardware acceleration  
 Test video processing in different embedded boards  
 Write a conference paper [7]  
 Establish an essential knowledge of the GPU virtualization  
 Establish an essential knowledge of Cuda [8], Docker and Nvidia-Docker [9]  
 Design a "Virtualized Media Server" Architecture  
 Establish an essential knowledge of container orchestration and Kubernetes  
 Test and analyse the performance of Nvidia-Docker and Kubernetes with the GPU  
 Design a mathematical model for GPU job on Docker and Kubernetes  
 Design a new Kubernetes scheduler and manager  
 Implement the new design  
 Write and publish a journal paper  
 Write the final report

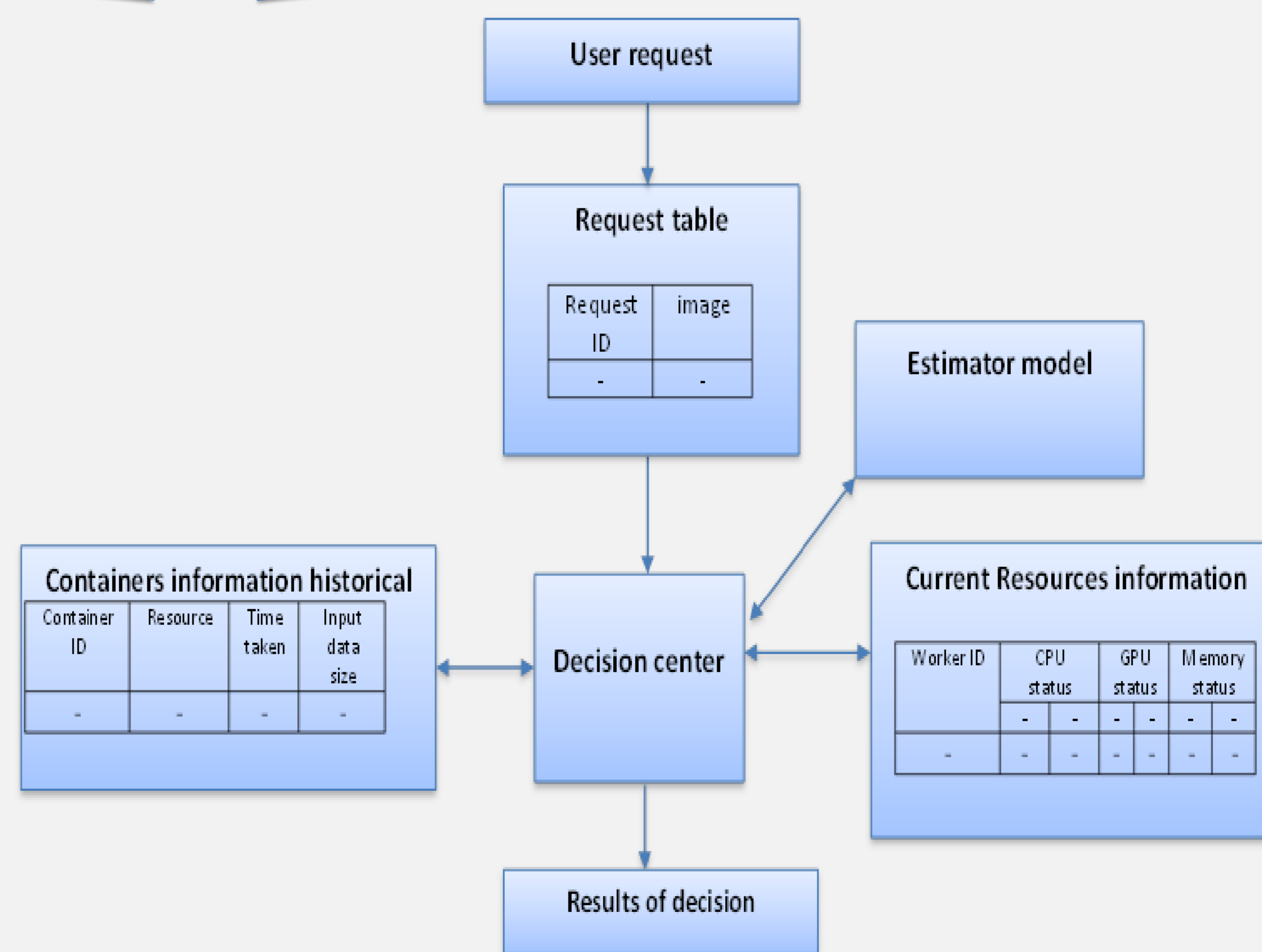


## Results



The new Kubernetes manager and scheduler uses containers information historical to decide which the best node where the pod should be deployed and it can choose automatically between the CPU node and the GPU node.

Mathematical model of the GPU is used to estimate the time required to run a GPU task inside a Docker container.



### Mathematical model of the GPU task on Docker

$$Comp_{SM} = (ld_0 + st_0) \cdot g_{SM}$$

$$Comp_{GM} = (ld_1 + st_1 - L1 - L2) \cdot g_{GM} + L1 \cdot g_{L1} + L2 \cdot g_{L2}$$

#### The first scenario

One container runs in a single GPU node.

$$T_1 = \begin{cases} \frac{t \cdot (Comp + Comp_{GM} + Comp_{SM})}{R \cdot P \cdot \lambda \cdot \alpha}, & \alpha = 1 \\ error, & \alpha = 0 \end{cases}$$

#### The second scenario

Multi-containers have the same number of threads runs simultaneously and start in the same time in a single GPU.

$$T_{i \in [1..N]} = \begin{cases} \left( \frac{t \cdot (Comp + Comp_{GM} + Comp_{SM})}{R \cdot P \cdot \lambda \cdot \alpha} \right) * N, & \alpha = 1 \\ error, & \alpha = 0 \end{cases}$$

#### The third scenario

Multi-Containers run simultaneously in a single GPU. However, we suppose in this scenario that one of those applications has an execution longer than the others.

On suppose that  $S_{i \in [1..N]} = \frac{t_i \cdot (Comp + Comp_{GM} + Comp_{SM})}{R \cdot P \cdot \lambda \cdot \alpha}$

$$T_N = \begin{cases} S_N + S_{i \in [1..N-1]} \cdot (N - 1), & \alpha = 1 \\ error, & \alpha = 0 \end{cases}$$

$Comp$  : The computational time used by each thread in a kernel.  
 $Comp_{GM}$  : The execution time for communication in global memory per thread.  
 $Comp_{SM}$  : The execution time for communication in global shared per thread.  
 $ld_0$  : The total number of load performed by all threads in the shared memory.  
 $st_0$  : The total number of stores performed by all threads in the shared memory.  
 $ld_1$  : The total number of load performed by all threads in the global memory.  
 $st_1$  : The total number of stores performed by all threads in the global memory.  
 $R$  : Clock rate.

$L1$  : The cache memory.  
 $L2$  : The cache memory.  
 $g_{SM}$  : The latency in communication over shared memory.  
 $g_{GM}$  : The latency in communication over global memory.  
 $g_{L1}$  : The latency in communication over cache memory.  
 $g_{L2}$  : The latency in communication over cache memory.  
 $\lambda$  : Application optimisations factor  
 $\alpha$  : If there is memory resource available  $\alpha = 1$  else  $\alpha = 0$ .

## References

- [1] Docker [Online]. Available: <https://www.docker.com>.
- [2] kubernetes [Online]. Available: <https://kubernetes.io/>
- [3] GStreamer Application Development Manual [Online]. Available: <http://gstreamer.freedesktop.org/>
- [4] X. Nui, L. Galarza, Y. Gao, J. Fan. "Software-hardware co-design for video coding acceleration" In Southeastern Symposium on System Theory (SSST), Jacksonville, FL, March 2012, pp. 57 – 60.
- [5] D. Min, Q. Rongcai, W. Ruiping, B. Sheng, C. Wenyi, X. Jiayi, "A new high-definition video player method based on GPU technology", In international Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), Bangkok, May 2012, pp.388 – 392.
- [6] H. Wang, J. Li, C. Zhao, Z. Ying, "Design of an Embedded Streaming Media Server in video monitoring" In International Conference on Natural Computation (ICNC), Shenyang, July 2013, pp. 1324 – 1328.
- [7] G. El Haj Ahmed, F.-G. Castiñeira, E.-C. Montenegro, P.-C. Soto, "System-on-Chip evaluation for the implementation of video processing servers", 5<sup>th</sup> World Conference on Information Systems and Technologies (WorldCIST), April 2017.
- [8] Cuda [Online]. Available: <https://developer.nvidia.com/cuda-toolkit>
- [9] Nvidia-Docker [Online]. Available: <https://devblogs.nvidia.com/parallelforall/nvidia-docker-gpu-server-application-deployment-made-easy/>