An hypervisor approach for mixed critical real-time UAV applications

Tristan Fautrel
Université Paris-Est
LIGM, UMR CNRS 8049
tristan.fautrel@univ-paris-est.fr

Laurent George
ESIEE Paris
LIGM, UMR CNRS 8049
laurent.george@esiee.fr

Frédéric Fauberteau
Léonard de Vinci Pôle Universitaire
Research Center
frederic.fauberteau@devinci.fr

Thierry Grandpierre
ESIEE Paris
LIGM, UMR CNRS 8049
thierry.grandpierre@esiee.fr

***Abstract*—In the context of smart cities, with important population density, Unmanned Aerial Vehicles (UAV) should be certified for flying. Virtualization techniques allow us to run several applications of different criticalities on the same hardware architecture in a spatially and temporally isolated manner. This makes it possible to adapt the certification cost of an application to its criticality level. We consider the case where each application is composed of a set of sporadic tasks that are executed into a dedicated Virtual Machine (VM). The set of VMs is monitored by a hypervisor that provides them an abstraction of the underlying hardware resources. The use of an hypervisor to isolate the applications hosted into their VMs results in a two-level hierarchical system. The VMs are scheduled at the first level then the tasks of each application are scheduled at the second level. The parameters of the tasks composing the different applications are known a priori and the main contribution of this paper is to provide a sizing method for the VMs. It consists in computing both period and slot duration for each VM such that its assigned tasks are schedulable (satisfy all the deadlines of the tasks).**

I. Introduction

Existing professional Unmanned Aerial Vehicles (UAVs) embed several applications of different criticalities. The control command (CC) application is responsible for the flight of the UAV. It has strong real-time constraints and the highest criticality level, and is often developed using bare-metal software. The missions of the UAV refers to a set of applications that usually have lower criticality level (*e.g.* video application, communication). In the context of smart city and high civil density, drones should be certified at the highest criticality level to be allowed to fly (CC application has high safety criticality constraint). If all applications are run on the same platform with no logical and temporal isolation among them, the highest criticality application propagates its criticality to all other applications (less critical applications should then also be certified at the highest level). This can lead to over-dimension computing resources. Hence it might have impacts on the cost, weight and autonomy of the drone. This problem can be mitigated using Operating Systems (OSs) supporting mixed criticality scheduling [1] where each application is associated a worst case criticality level. This requires a specific OS supporting mixed criticality scheduling and that all applications are executed on the same OS. Container approaches like Docker or LinuX Container (LXC) supported by Control groups (cgroups) can also be considered for UAV systems [2]. Each application is then assigned to a specific container and is executed by a Linux host OS. Some patch supporting real-time scheduling for Linux have been recently proposed for real-time LXC [3] and docker containers [4]. LXCs have also be proposed in the context of UAVs [5] to limit and isolate the access to resources on a Linux system. The LXC is also proposed in [6] for the CommUnicationS-control distributed simulator of UAVs (CUSCUS). However, CC application in charge of UAV fly should be certified at the highest criticality level by certification authorities especially when flying in civil areas. Certifying a Linux system (with real-time extensions) is still not an easy task.

Furthermore, applications are often developed by distinct specialized teams and even by different service providers having their own runtime environment requirements. The integration phase of software components developed by heterogeneous teams is then particularly complex.

The development cost of this phase can be reduced by a last approach based on VMs assigned to the different teams. VMs scheduling is done by a hypervisor, a software layer in interaction with hardware platforms that can run several applications in dedicated VMs having their own OS and their own scheduler.

Hypervisor can be of type 1 or type 2. With type 1 hypervisor approach, the hypervisor is implemented bare metal, on top of hardware (see figure 1). The VMs are managed by the hypervisor. With type 2 hypervisor approach, a host operating is in charge of running the hypervisor which adds for embedded applications a significant overhead on VM scheduling and certification. Type 1 approach has been considered for UAV systems with the Xen Hypervisor in [7] or in [8]. This solution is however not easy to certify at a high criticality level.

In this paper, we propose a certified type 1 hypervisor approach for the dimensioning of an UAV system that allows us to certify applications with a specific certification tool according to its criticality level. This approach is currently considered in the CEOS [9] project for the conception of a certified UAV for surveillance and inspection applications. A certified hypervisor grants temporal and logical isolation between applications (see for *e.g.* PikeOS from Sysgo [10] that can be certified DO 178B/C for flying systems).

A solution to improve the reliability and the safety of such a system is to assign distinct hardware resources to applications

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

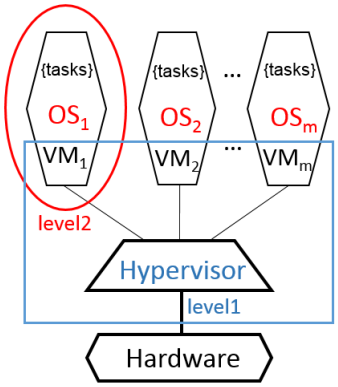

Fig. 1. Our type 1 two-level hypervisor approach

using a hypervisor. This approach consists in abstracting the hardware platform by running applications in dedicated VMs.

This lead to a system that can be seen as a two-level hierarchical system [11] (see figure 1).

The first level consists in scheduling VMs that are managed by a hypervisor. At the second level, each VM embeds an application and the tasks that compose it are scheduled by a specific Operating System (OS) in the slots allocated to the VM.

Several papers introduced hierarchical scheduling to schedule both periodic and sporadic (or aperiodic) tasks [12],[13].

In [11], a schedulability analysis is provided for this scheduling scheme in the context of hierarchical fixed priority preemptive tasks. In [14], a response time analysis is provided for hierarchical systems using Deferrable Servers [15] or Sporadic Servers [16]. In [17], an exact response time analysis is provided to improve the success ratio of the schedulability test.

In this work, we focus on a hierarchical scheduling and we consider that tasks composing an application are scheduled by a Fixed-Task-Priority (FTP) preemptive scheduling in a VM instead of a server ([18] introduced EDF policy).

Our hypothesis is that only the parameters of the tasks (*i.e.* WCET, period and deadline) at the second level are known but not the ones of the VMs (*i.e.* slot duration and VM period of activation). The problem we address is to determine VMs parameters, to obtain a schedulable system at the second level. The scheduling of the VMs implies to derive their parameters from the tasks parameters. In [19], such a sizing is provided in the case of servers at the first level. This server sizing is performed according to a FTP policy for the second level and a fair scheduling algorithm at the first level. In [20] we consider FTP policy and show that a fair scheduling at the first level leads to higher number of schedulable tasksets.

The concept of Mixed-Criticality (MC) has been introduced in [21] in a view to adapt the certification process to the criticality level of the tasks. The more critical task, the higher considered execution time. In this paper we consider that criticality is defined at the level of VMs. All tasks in one VM have the same criticality level but VMs can have different criticalities.

Our first contribution is to provide conditions to decide on the schedulability of the tasks in their dedicated VM. The conditions derive minimum and maximum acceptable VM periods. Then, we provide an algorithmic approach to compute slot durations. Our particular method first computes these parameters for the most critical VM. Then, it tries to schedule remaining VMs with harmonic periods multiple of the most critical VM period. The resulting VM schedule is strictly periodic (*i.e.* the difference between the start of the execution of the VM and the release time of the VM is constant for a given VM).

This paper is organized as follows: in Section II we introduce the model and notations used in this paper. In Section III we provide different task schedulability conditions for a given VM. In Section IV we propose an algorithm to compute the parameters of the most critical VM and in Section V we show how to find the parameters of the following VMs when they have harmonic periods.

Finally, we validate our work through a real-life use-case in Section VI and we give some conclusion of our work.

II. Model and Notations

As stated in Section I, the scheduling model considered in this paper is a two-level hierarchical level model. At the first level, a set ρ of m VMs is scheduled. The s^{th} VM $\rho^s \in \rho$ is characterized by its slot duration C^s, its period T^s and its worst case context switch time C^s_o. With this model, a VM is active during C^s every T^s. These scheduling parameters are not known a priori and we focus on a sizing approach to compute them. The set ρ is sorted in decreasing criticality order. That is, ρ^1 is the most critical VM and ρ^m is the least critical one.

The whole system consists of a set τ of n sporadic tasks. Each task τ_i is characterized by its Worst Case Execution Time (WCET) C_i, its minimum inter-arrival time T_i and its relative deadline D_i. The utilization of the task τ_i is denoted U_i and defined by $U_i = \frac{C_i}{T_i}$. The parameters of these tasks are known a priori. At the second level of our hierarchical scheduling model, each VM ρ^s hosts an application consisting of a subset $\tau(\rho^s)$ of τ. The subsets are disjoint and a task τ_i can be scheduled by only one VM. The utilization of the tasks of $\tau(\rho^s)$ is defined by $U^s = \sum_{\tau_i \in \tau(\rho^s)} \frac{C_i}{T_i}$. It corresponds to the utilization of the VM ρ^s without taking into account the overhead due to VM context switches.

We later characterize T^s_{min} (respectively T^s_{max}) the minimum (respectively the maximum) period of VM ρ^s such that any VMs smaller (respectively higher) period leads to non schedulable VMs. These bounds on the VM periods are used to find feasible VMs periods.

The definition of harmonic taskset comes from [22] and [23]. A harmonic taskset is defined by:

Definition 1 (Harmonic taskset [23])**.** *A set of tasks is harmonic if and only if:*

$$\forall \tau_i, \tau_j \in \{\tau\}^2, (T_i \bmod T_j = 0) \vee (T_j \bmod T_i = 0) \quad (1)$$

where $\mod$ *is modulo operator and* $\vee$ *is logical OR operator.*

We then adapt this definition to our context by applying it to the set of VMs instead of the the set of tasks. The equation is then transformed in:

Definition 2 (Harmonic set of VMs)**.** *A set of VMs is harmonic if and only if:*

$$\forall \rho^i, \rho^j \in \{\rho\}^2, \left(T^i \bmod T^j = 0\right) \vee \left(T^j \bmod T^i = 0\right) \quad (2)$$

This definition is used to determine if VMs are schedulable with a strictly periodic scheduling.

III. Conditions of schedulability

In this section we provide several conditions of schedulability which take into account the overhead of the context switching between the different VMs.

A necessary and sufficient condition to schedule harmonic tasks is given in [22] if and only if the tasks have distinct periods. This condition ensures that task executions are strictly periodic (the time between the release time and the start of the execution is constant). We extend this condition to the context of VM scheduling and determine if the VMs are schedulable by a specific scheduling algorithm when the periods of VMs are harmonic. As the VMs do not have direct temporal constraints, a schedulable VM ρ^s is a VM with parameters such that the scheduling algorithm used to schedule ρ^s satisfies its properties (*e.g.* strict periodicity in our case).

Theorem 1 (Schedulability condition of harmonic taskset with distinct periods [22])**.** *Let* τ *be an harmonic taskset such as* $\forall \tau_i \in \{\tau \setminus \{\tau_1\}\}, T_i > T_{i-1}$*. The set* τ *is schedulable if and only if:*

$$\forall \tau_i \in \{\tau \setminus \{\tau_1\}\}, C_i \leq T_1 - C_1 \quad (3)$$

We adapt the previous theorem to our context and model by applying it to the different VMs (*i.e.* by replacing the WCET of a task by the slot durations of a VM), taking into account the overheads of VM context switch. We obtain:

Theorem 2 (Schedulability condition of harmonic VMs set distinct periods with overheads)**.** *Let* ρ *be an harmonic set of VMs such as* $\forall \rho^s \in \{\rho \setminus \{\rho^1\}\}, T^i > T^{i-1}$*. The set* ρ *is schedulable if and only if:*

$$\forall \rho^s \in \{\rho \setminus \{\rho^1\}\}, U^s \times T^s + C_o^s \leq T^1 - C^1 \quad (4)$$

With the two-level hierarchical model, a necessary condition must be satisfied. This condition provides a bound on the utilization of the tasks in the whole system plus the context switch overhead of each VM: $\sum_{\rho^s \in \rho} \left(U^s + \frac{C_o^s}{T^s}\right) \leq 1$.

If VMs are harmonic, we deduce a bound $T_{\beta^1}^s$ for the minimum period of any VM ρ^s as follows.

$$\forall \rho^s \in \rho, T_{\beta^1}^s \geq C^s = C_o^s + U^s \times T_{\beta^1}^s \quad (5)$$

$$\Leftrightarrow T_{\beta^1}^s \geq \frac{C_o^s}{1 - U^s} \quad (6)$$

The bound $T_{\beta^1}^s$ is given from (6). The last term $U^s \times T_{\beta^1}^s$ represents the execution time of the tasks for this specific VM period when the scheduling is non-preemptive. We obtain (6) which provides a safe minimum period for a given VM ρ^s.

But it can be too small. For instance, consider that the overhead $C_o^s = 1$ unit of time (ut). For an utilization U^s ranging from 0.1 to 0.5 ut, this equation always gives the same minimum value of 2 ut for $T_{\beta^1}^s$. Then, if the VM period is set to 2 ut, only 1 ut remains for the VM. With such a VM period the only possible slot duration of the VM is necessarily 2 ut. In that case, this VM uses all system resources, even if task utilization is equal to 0.1, which is not acceptable for real systems. This is due to the integers parameters of the VMs which lead to rounding approximations. In order to avoid this drawback, we take into account the ratio α ($0 \leq \alpha \leq 1$) between the overhead and the VM period. We provide a new bound $T_{\beta^2}^s$ on the minimum VM period:

$$T_{\beta^2}^s \geq \frac{C_o^s}{\alpha} \quad (7)$$

For instance, if the user limits the overhead to 5% of the VM period (i.e. $\alpha = 0.05$), and the overhead is 1 ut, we get $T_{min}^s = \frac{1}{0.05} = 20$ while it was only of 2 ut using the previous equation.

The final equation for the minimum VM period is then:

$$T_{min}^s \geq \max\left(T_{\beta^1}^s, T_{\beta^2}^s\right) \quad (8)$$

We now define the maximum possible period of a VM. From the following equations and Figure 2 we deduce a bound T_{max}^s for the maximum period of VMs.

$$\forall \tau_i \in \tau(\rho^s), D_i \geq T^s - U^s \times T^s + C_i \Leftrightarrow \quad (9)$$

$$T_{max}^s \leq \frac{D_i - C_i}{(1 - U^s)} \quad (10)$$

Equation (9) is depicted in Figure 2. The deadline of every task should be greater than or equal to the period of the VM minus the slot duration of the VM plus the WCET of the task under study. If the deadline is lower than this value, the system becomes unschedulable. This condition is a necessary condition. The next steps are the same as for (6). The critical instant of a task is when its activation is synchronous with the tasks of higher priorities (see [20]). In Figure 2 the blue area corresponds to the time when the tasks of the VMs may be scheduled. At the end of the blue area, no tasks of the studied VM can be scheduled. This may be seen as the scheduling of a task of higher priority. The critical instant is therefore on this figure at the end of the blue area.

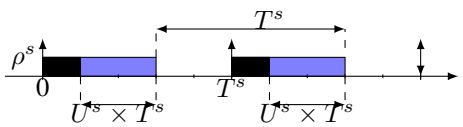

Fig. 2. Representation of the maximum period Equation (9).

IV. Parameters of the most critical VM

In this section, we provide an interval of periods for the first VM ρ^1 based on necessary schedulability conditions. This interval of VM periods is bounded by the minimum and the maximum VM periods found using equations of Section III. From this interval we deduce a set of parameters for the VM that guarantees the schedulability of the tasks of ρ^1. From these parameters, we define in Section V the parameters and the schedule of the remaining VMs.

A. Maximum Period

The equation for the maximum VM period is defined by Equation (10). It depends on the taskset $\tau(\rho^1)$ of VM ρ^1. It must be applied to every task of $\tau(\rho^1)$ then the maximum period T^1_{max} is given by the smallest obtained value.

The results of equation (10) are real numbers. As we consider integer VM parameters, the floor function is used to obtain a bound on the VM period from (10). This induces approximation on the bound of the maximum VM period, but is mandatory to find the parameters and the slot duration as seen in Section IV-C.

B. Minimum Period

As explained in Section III, (8) is used to compute the minimum utilization. This equation uses (6) and (7).

The minimum VM period is the largest value between the result of these two equations. We need to apply the ceil function to obtain a valid integer number for the minimum period.

According to equations (10) and (8) the VM may have a minimum period greater than or equal to the maximum period: that case corresponds to a schedulable problem. It may be caused by the overhead resulting from C^s_o which is not obviously taken into account during the creation of the VMs.

C. Computation of C^1 and T^1

From the minimum period T^1_{min} and the maximum period T^1_{max}, we can compute the parameters C^1 and T^1 of the most critical VM ρ^1. This is done by Algorithm 1. It uses the Worst Case Response Time (WCRT) computed using (11) that is adapted for sporadic harmonic periods from the equation given in [20]. The WCRT is computed from the classical equation given in [24] to which we add the time when ρ^1 is not scheduled and the duration of the overhead C^1_o. The WCRT is given by the following equation:

$$r_i = \max\left(w_{i,q} - qT_i\right) \tag{11}$$

Where $q = 0 \dots Q$ and Q is the minimum value such that $w_{i,q} \leq (q+1)T_i$ and $w_{i,q}$ is recursively computed by the following equation [20]:

$$w^{m+1}_{i,q} = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w^m_{i,q}}{T_j} \right\rceil C_j + \left\lceil \frac{w^m_{i,q}}{T^s} \right\rceil (T^s - C^s + C^s_o) \tag{12}$$

Algorithm 1 uses a binary search to compute the smallest T^1 and the corresponding C^1 of ρ^1 such that $\tau(\rho^1)$ is schedulable. It first tests for $T^1 = T^1_{min}$. If $\tau(\rho^1)$ is not schedulable, T^1 is binary searched between T^1_{min} and T^1_{max}.

The steps of Algorithm 1 are as follows. After initializing T^1 at L1 (i.e. Line 1), it loops from L2 to L14 to (i) binary search a value of T^1, (ii) compute the value of C^1 *w.r.t.* T^1 (L3) and (iii) test if $\tau(\rho^1)$ is schedulable. If $\tau(\rho^1)$ is schedulable according to (11), either $T^1 = T^1_{min}$ and (C^1, T^1) is returned, or $T^1 > T^1_{min}$ and the upper bound T^1_{max} is decreased to the value of T^1. Otherwise $\tau(\rho^1$ is not schedulable and the lower bound T^1_{min} is increased to the value of T^1. T^1 is then updated to the median value of the two bounds T^1_{min} and T^1_{max}, and the iterations continue. The algorithm either returns false if no feasible schedule could be found or true else. In the latter case, valid VM parameters are also returned.

Algorithm 1: Parameters of the most critical VM

```
   Input: (T^1_min, T^1_max)
   Result: (C^1, T^1, sched)
1  T^1 = T^1_min;
2  while T^1_min + 1 ≠ T^1_max do
3  |  C^1 = ⌈U^1 × T^1 + C^1_o⌉;
4  |  if τ(ρ^1) is schedulable using (11) then
5  |  |  sched = true;
6  |  else
7  |  |  sched = false;
8  |  if sched = true and T^1 = T^1_min then
9  |  |  return (C^1, T^1, sched);
10 |  else if sched = true then
11 |  |  T^1_max = T^1;
12 |  else
13 |  |  T^1_min = T^1;
14 |  T^1 = (T^1_min + T^1_max)/2;
15 return (C^1, T^1, sched);
```

V. Parameters and scheduling of harmonic VMs

In this section we provide a method to compute the parameters of VMs other than ρ^1 (once the parameters of ρ^1 have been defined). Since the problem of sizing VMs is complex, we provide a heuristic approach. We compute harmonic VM periods in order to produce higher number of schedulable tasksets [20].

A. Parameters

The parameters of the most critical VM ρ^1 are used to find the parameters of the other VMs. These parameters are the input of Algorithm 2. We compute the parameters one by one starting from the most critical to the least one. This algorithm uses the condition of schedulability defined in Equation (4). This algorithm computes and tests a set of parameters for each remaining VM and checks if they are schedulable according to Equation (4). The schedulability of the tasks in each VM is not tested at this point.

The necessary and sufficient condition of Equation (4) is valid if and only if all the VMs are harmonic and have distinct periods. As the parameters of the first VM are already set, the next VM periods are found iteratively from the last VM period found (starting by the VM period of ρ^1).

Algorithm 2 loops on every remaining VM (L2) to find valid parameters if possible. L3 finds the maximum VM period of the current tested VM. Then L4 tests a VM period for the current VM. To create distinct harmonic periods, we multiply by 2 the VM period of the last VM set. It assures the harmonicity without heavily increasing the period of the VMs. After finding a VM period to test, we can check if the VM period found is not higher than the maximum VM period of the VM (L5). As in Algorithm 1 (see Section IV-C), L6 computes the VM slot duration. The remaining lines apply equation (4) to know if the VM tested is schedulable using these parameters. It adds the parameters to the VM when succeded or ends the algorithm otherwise.

Algorithm 2: Parameters of harmonic VMs

Input: ρ, ρ^1, C^1, T^1
Result: Parameters of the VMs scheduled
1 lastVM = 1;
2 **foreach** ρ^s *in* ρ **do**
3 Set T^s_{max} according to (10);
4 $T^s_{test} = T^{lastVM} \times 2$;
5 **if** $T^s_{test} \leq T^s_{max}$ **then**
6 $C^s_{test} = \lceil U^s \times T^s_{test} + C^s_o \rceil$;
7 **if** $C^s_{test} \leq T^1 - C^1$ **then**
8 $C^s = C^s_{test}$; $T^s = T^s_{test}$; lastVm = s;
9 **else**
10 break;

B. Scheduling

Once the parameters of each harmonic VMs are known, their scheduling must be defined. Our scheduling algorithm is an adaptation of [22]. We split it in Algorithm 3 and 4. These algorithms always find an available solution without failing. This is due to the subset of VMs of ρ given to the algorithms. This subset has parameters which satisfies schedulability condition of equation (4). This allows us to simplify the algorithm to schedule the VMs and assures that the algorithms will end with a solution.

Algorithm 3 has one input-element corresponding to the subset of VMs previously defined. This set is ρ_{sub} with $\rho_{sub} \in \rho$. The algorithm returns the set of the VM start times in one hyper-period, the Least Common Multiple (LCM) of the periods of the VMs. This information is stored in the variable $VMxList$. Each VM has a list. The x^{th} list $VMxList$ corresponds to the x^{th} VM. L1 of the algorithm defines the number of activations of the first VM during the scheduling defined on the hyper-period. As the VMs have distinct and harmonic periods, the hyper-period is equal to the highest VM period.

Algorithm 3: Harmonic VMs scheduling algorithm

Input: Set ρ_{sub} for which the parameters are found
Input: p the number of VMs in ρ
Result: List of start of execution for each VM
1 repetitionsFirstVM = $\frac{T^p}{T^1}$;
2 boolean isFree[repetitionsFirstVM] = true;
3 **for** $i \leftarrow 0$ **to** *repetitionsFirstVM* **do**
4 VM1List += $i \times T^1$;
5 **foreach** ρ^j *in* ρ_{sub} **do**
6 repetitionsVM = $\frac{T^p}{T^j}$;
7 **for** $i \leftarrow 0$ **to** *repetitionVM* **do**
8 index = $\frac{i \times T^j}{T^1}$;
9 firstFree = findFirstFree(index);
10 isFree[firstFree] = false;
11 VMjList = VMjList $\bigcup \{firstFree \times T^1 + C^1\}$;

L2 defines an array of booleans (initially set to true) which has a size equal to the number of activations of the first VM. The VM with the highest number of activations is the most critical one. The scheduling algorithm uses the time between each activation of the first VM to schedule the other VMs. Each VM is scheduled right after the execution of the first VM (see [22] and [23] for more details). This boolean array defines if the space right after the i^{th} activation of the first VM is available or not (used by another VM or free).

The first loop (L3) is to define the start times of the first VM. As nothing is scheduled yet, the result of this loop is the start of slot executions at each period for the first VM. The second loop (L5) defines the start times of all the remaining VMs. L6 defines the number of activations of each VM in the hyper-period.

L8 set the variable $index$. This variable is the minimum index in the boolean array $isFree$ which corresponds to the i^{th} repetition of the VM ρ^j. L9 set the $firstFree$ variable. This variable corresponds to the first free index in the array $isFree$. In other words function $findFirstFree$ (algorithm 4) finds the first index in the array $isFree$ where no VMs is scheduled (except the first one). L10 sets the index of $isFree$ array as not available. L11 updates the list of VM j start-times.

Algorithm 4: findFirstFree function

Input: isFree[] a boolean array of size $\frac{T^p}{T^1}$
Input: index the minimum index to find
Result: The index of the first free space in the scheduling
1 **for** $i \leftarrow index$ **to** p **do**
2 **if** *isFree[i]* **then**
3 **return** i;

VI. Example

We illustrate our approach by presenting the use case of the CEOS project [9]: an industrial project that develops reliable and secure inspection drone systems. We derive an example

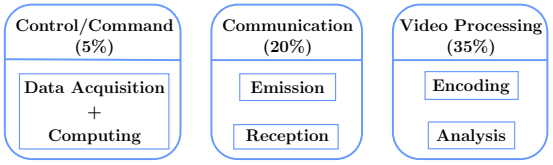

Fig. 3. The three VMs of CEOS use-case and their associated tasks.

	τ_i	C_i	T_i	D_i
ρ^1	τ_1	1	20	15
ρ^2	τ_2	1	15	20
	τ_3	2	15	30
ρ^3	τ_4	2	30	40
	τ_5	4	40	80

Fig. 4. Global taskset

from this use case to illustrate the algorithms and conditions previously presented.

The CEOS use case is composed of three applications developed by independent teams and have different requirements in terms of memory usage, real-time constraints, computing power and criticality. Each application is associated to a dedicated VM respectively (1) VM_1 of High criticality: a control-command application in charge of the flight of the drone, (2) VM_2 of Medium criticality: a communication application which allows the drone to receive command from a ground station, but also to send drone's parameters and (3) VM_3 of Low criticality: a video application which stores, analyzes and compress images from embedded cameras.

The control-command application is implemented bare-metal, the communication application is running on OpenWRT and the video application on a Linux Debian operating system.

In the context of the CEOS project, we rely on the the PikeOS [10] hypervisor to isolate spatially and temporally each application in a dedicated VM. This level 1 hypervisor has been chosen as it can be certified DO 178B/C for flying systems. Certification can mandatory when flying in civil areas (authorization for flying is given by DGAC in France depending on the weight of the UAV).

We now give an example of scheduling which uses three VMs and takes as input the taskset τ defined in Figure 4. The first VM contains one task τ_1, the second one two tasks (τ_2 and τ_3) and the third one two tasks (τ_4 and τ_5). For each VM ρ^i the overhead C_o^i is set equal to 1.

Our example takes into account the criticality of the different VMs. As none of the parameters of the VMs is known, the first step is to find the parameters of the most critical VM, namely ρ^1. As explained in Section IV, the minimum and maximum VM periods for this VM must be computed with (8) and (10).

Algorithm 1 may now be used to find the parameters of the first VM. Line 2 initializes T^1_{test} to 10. Line 4 with such values leads to $C^1_{test} = \lceil \frac{1}{20} \times 10 + 1 \rceil = 2$. The test VM $\rho^1_{test}(2, 10, 1, 1)$ is then created. The scheduling of the tasks in this VM is tested (line 6). Equation (11) leads to a WCRT

Equation (6)	$\lceil \frac{1}{1-\frac{1}{20}} \rceil = 2$
Equation (7)	$\lceil \frac{1}{0.1} \rceil = 10$
Minimum Period (Equation (8))	10
Maximum Period (Equation (10))	$\lfloor \frac{15-1}{1-\frac{1}{20}} \rfloor = 14$

TABLE I
MINIMUM AND MAXIMUM VM PERIODS FOR ρ^1

ρ^2	τ^2	$\lfloor \frac{20-1}{1-U^2} \rfloor = 23$	ρ^3	τ^4	$\lfloor \frac{40-2}{1-U^3} \rfloor = 45$
	τ^3	$\lfloor \frac{30-2}{1-U^2} \rfloor = 35$		τ^5	$\lfloor \frac{80-4}{1-U^3} \rfloor = 91$

TABLE II
COMPUTATION OF THE MAXIMUM PERIOD OF VM2 AN VM3.

equal to 10, so the VM is schedulable.

As all the tasks of the first VM are schedulable, these parameters (2 for the slot duration and 10 for the VM period) are valid. From the parameters of the first VM we can deduce the parameters of the following VMs if harmonic parameters may be found thanks to Algorithm 2. This algorithm uses the maximum VM period found in (10). The result of this equation for each task of each VM is given in Table II (where $U^2 = \frac{2}{10}$, $U^3 = \frac{1}{6}$.

From this approach we can deduce for each VM the maximum period. Respectively 23, 45 for ρ^2 and ρ^3.

The line 3 sets the maximum period T^s_{max} of all VMs except VM_1 from the values provided in Table II. Line 4 defines the current VM period to test. As the first one is 10, the second one will be 20, the third 40. Only three VMs may be scheduled according to line 5 and the maximum VM period. Line 6 initializes the slot duration to test for each VM. In that case, $C^2_{test} = \lceil \frac{2}{10} \times 20 + 1 \rceil = 5$, $C^3_{test} = \lceil \frac{1}{6} \times 40 + 1 \rceil = 8$ (and as the test line 5 fails for ρ^4, C^4_{test} is not computed).

Line 7 checks C^2_{test} and C^3_{test} to know if this slot duration satisfies the necessary and sufficient condition given by (4). In that case, both values fulfill this required condition and we now obtain the three following VMs: $\rho^1(2, 10, 1, 1)$, $\rho^2(5, 20, 1, 2)$ and $\rho^3(8, 40, 1, 3)$.

At this point we can check if the tasks within the VMs are schedulable according to (11). The result of the WCRT of the tasks are 17, 28, 35 and 76 for τ_1, τ_2, τ_3 and τ_4 respectively.

As the three first VMs have harmonic periods they may be scheduled by Algorithms 3 and 4. In our example $\rho_{sub} = \rho^1 \cup \rho^2 \cup \rho^3$ since only the first three VMs have parameters because of the previous algorithms.

In this algorithm we are sure that the system (at least the VMs) are schedulable because of the different methods to find the parameters. The main idea to schedule the VMs is that the exact number of repetitions of each VM period is known (usually $\frac{hyperperiod}{period}$ but here the hyper-period is equal to the greatest period of VMs).

The first VM has the greatest number of repetitions as it has the smallest VM period. The algorithm will schedule the other VMs right after the slot durations of the first VM.

Let us define the start of the execution of the first VM (lines 3-5). In this example, $repetitionsFirstVM = \frac{40}{10} = 4$. The

execution are then $0 \times 10 = 0$, $1 \times 10 = 10$, $2 \times 10 = 20$ and $3 \times 10 = 30$.

The goal is now to schedule the others VMs when ρ^1 is not active. This is done in the loop (lines 6-14) of Algorithm 3. The idea here is to find the first repetition of the first VM which has no other VM scheduled.

We develop the loop for ρ^2. Here $repetitionsVM = \frac{40}{20} = 2$. The loop from lines 8 to 13 has only two iterations. The two computations of the index gives respectively 0 and 2 for ρ^2. It corresponds to the first iteration of ρ^1 where the VM may be scheduled. A function to check if a VM is not already scheduled is given in Algorithm 4. This function will find the first available iteration. As ρ^2 is the first VM scheduled after ρ^1 this function will not iterate.

But after this, the iteration 0 and 2 of the first VM are already taken by ρ^2. We show through ρ^3 the importance of Algorithm 4. For this one, $repetitionsVM = \frac{40}{40} = 1$, so $index = 0$. The repetition 0 of ρ^1 is already used by ρ^2. This is where Algorithm 4 is useful to get the first available repetition. In this example it is the repetition 3.

VII. Conclusion

We consider UAV real-time dimensioning when the set of applications executed by the drone are run in dedicated VMs managed by an hypervisor. An application is composed of several sporadic tasks with arbitrary deadlines. The VMs are scheduled by an hypervisor according to a scheduling table that must be constructed to satisfy the real-time constraints of the tasks. We suppose that each VMs have different criticalities. Furthermore, all the tasks executed in one VM must have the same criticality. The problem we consider is as follows: knowing the task parameters, how to dimension the VMs so as to satisfy the real-time constraints of the tasks ? We adopt an iterative approach for the construction of the scheduling table of the VMs, starting from the most critical VM to the least critical. For each VM, we propose an algorithm to compute its parameters (slot duration and period) satisfying the real-time constraints of the tasks run by the VM. This algorithm results in harmonic VM periods that maximize the chance to obtain a feasible schedule. As a future works we plan to propose a linear programming approach to assign non assigned remaining slots to non-schedulable VMs by our algorithm, possibly leading to arbitray slot patterns, not strictly periodic. We will also experiment our solution with the Ceos project.

Acknowledgment

This research is partially funded by the FR FUI/FEDER22 Ceos project, contract agreement no. DOS0052555/00 [9].

References

[1] A. Burns and R. I. Davis, "Mixed criticality systems - a review," in *https://www-users.cs.york.ac.uk/burns/review.pdf*.

[2] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. P. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar, "Openuav: A uav testbed for the cps and robotics community," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, IEEE Press. Piscataway, NJ, USA: IEEE Press, 2018. [Online]. Available: https://doi.org/10.1109/ICCPS.2018.00021

[3] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," in *EWiLi'18, the embedded operating system workshop, Co-located with the Embedded Systems Week, Torino, Italy, Oct. 2016*, 2018.

[4] M. Cinque and D. Cotroneo, "Towards lightweight temporal and fault isolation in mixed-criticality systems with real-time containers," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, June 2018, pp. 59–60.

[5] K. J. S. White, E. Denney, M. D. Knudson, A. K. Mamerides, and D. P. Pezaros, "A programmable sdn+nfv-based architecture for uav telemetry monitoring," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2017, pp. 522–527.

[6] N. R. Zema, A. Trotta, G. Sanahuja, E. Natalizio, M. D. Felice, and L. Bononi, "Cuscus: Communications-control distributed simulator," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2017, pp. 601–602.

[7] C. Rametta and G. Schembra, "Designing a softwarized network deployed on a fleet of drones for rural zone monitoring," vol. 9, no. 1, 2017. [Online]. Available: http://www.mdpi.com/1999-5903/9/1/8

[8] A. D. Santangelo, "A low cost, secure radio communications system for uavs," in *34th AIAA International Communications Satellite Systems Conference*, 2016, p. 5758.

[9] "Ceos fui/feder22 project," *www.ceos-systems.com*, 2017-2020.

[10] R. Kaiser and S. Wagner, "Evolution of the pikeos microkernel," in *Proc. of MIKES*, 2007, pp. 50–57.

[11] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Proc. of RTSS*. IEEE, 1999, pp. 256–267.

[12] Z. Deng and J.-S. Liu, "Scheduling real-time applications in an open environment," in *Proc. of RTSS*. IEEE, 1997, pp. 308–319.

[13] Z. Deng, J.-S. Liu, and J. Sun, "A scheme for scheduling hard real-time applications in open system environment," in *Proc. of EWRTS*. IEEE, 1997, pp. 191–199.

[14] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarhical fixed-priority scheduling." in *Proc. of ECRTS*, 2002.

[15] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers*, vol. 44, pp. 73–91, 1995.

[16] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *RTS*, vol. 1, no. 1, pp. 27–60, 1989.

[17] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proc. of RTSS*. IEEE, 2005, pp. 10–pp.

[18] A. Guasque, P. Balbastre, and A. Crespo, "Real-time hierarchical systems with arbitrary scheduling at global level," *J. Syst. Softw.*, vol. 119, pp. 70–86, 2016.

[19] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proc. of ECRTS*. IEEE, 2003, pp. 151–158.

[20] T. Fautrel, L. George, and F. Fauberteau, "A hypervisor schedulability analysis for safety and security critical applications scheduled in arbitrary patterns of slots," *Proc. of JRWRTC*, 2017.

[21] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. of RTSS*, 2007.

[22] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," *Automata, languages and programming*, pp. 299–311, 2010.

[23] W. Wei and C. Liu, "On a periodic maintenance problem," *Operations Research Letters*, vol. 2, no. 2, pp. 90–93, 1983.

[24] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proc. of RTSS*. IEEE, 1990, pp. 201–209.