2018 IEEE Real-Time Systems Symposium

Scheduling Multi-Periodic Mixed-Criticality DAGs on Multi-Core Architectures

Roberto Medina, Etienne Borde, Laurent Pautet
LTCI, Télécom ParisTech, Paris, France
firstname.lastname@telecom-paristech.fr

Abstract—**Thanks to Mixed-Criticality (MC) scheduling, high and low-criticality tasks can share the same execution platform, improving considerably the usage of computation resources. Even if the execution platform is shared with low-criticality tasks, deadlines of high-criticality tasks must be respected. This is usually enforced thanks to operational modes of the system: if necessary, a high-criticality execution mode allocates more time to high-criticality tasks at the expense of low-criticality tasks' execution. Nonetheless, most MC scheduling policies in the literature have only considered independent task sets. For safety-critical real-time systems, this is a strong limitation: models used to describe reactive safety-critical software often consider dependencies among tasks or jobs.**

In this paper, we define a meta-heuristic to schedule multi-processor systems composed of multi-periodic Directed Acyclic Graphs of MC tasks. This meta-heuristic computes the scheduling of the system in the high-criticality mode first. The computation of the low-criticality scheduling respects a condition on high-criticality tasks' jobs, ensuring that high-criticality tasks never miss their deadlines. Two implementations of this meta-heuristic are presented. In high-criticality mode, high-criticality tasks are scheduled as late as possible. Then two global scheduling tables are produced, one per criticality mode. Experimental results demonstrate our method outperforms approaches of the literature in terms of acceptance rate for randomly generated systems.

I. Introduction

With the adoption of multi-core architectures in safety-critical real-time systems, more and more functionalities share a common execution platform. In practice, due to safety requirements, only functionalities with the same level of *criticality* should share resources. This constraint limits the deployment of functionalities on multi-core architectures, leading to a potential waste of computation power.

To solve this problem, the Mixed-Criticality (MC) model [1] has proposed an approach to execute high and low-criticality tasks in a single platform. Thanks to operational modes, MC systems are capable of executing tasks with different criticalities in the same architecture. When the system is in a nominal mode: tasks are executed with an "optimistic" timing budget (*e.g.* a WCET estimated by the system designer). At the same time, MC systems are also capable of limiting low-criticality tasks' execution in favor of high-criticality tasks to ensure safety of the system. When a Timing Failure Event (TFE) is detected, (*i.e.* a task did not complete its execution within its low-criticality timing budget) the system switches to a high-criticality mode and high criticality tasks are executed with a "pessimistic" timing budget (*e.g.* a WCET enforced by a certification authority).

Many contributions to schedule these type of systems with real-time constraints (*i.e.* respecting deadlines in all operational modes) have been proposed in the literature [2]. However, most scheduling policies assume independent task sets: no synchronization (*e.g.* lock-based communication, precedence constraints) among tasks is considered by the schedulers. This assumption is very restrictive: methods used to develop reactive safety-critical systems often model such systems as data-flow graphs. Some examples of widely used tools that follow graph representations for their systems are SCADE from Esterel Technologies[1] and Simulink from MathWorks[2]. In practice, tasks have data dependencies and can only start their execution when all their predecessors have completed their execution. Such precedence constraints are usually modelled with Directed Acyclic Graphs (DAGs) of tasks. Cyclic dependencies among multi-periodic tasks can be resolved thanks to deterministic periodic-delayed communications like the one presented in [3].

In this paper, we aim at addressing this problem: *how to efficiently schedule multi-periodic DAGs of MC tasks on multi-core architectures?* Efficiency here refers to the capacity to find a feasible scheduler on a given number of cores.

Scheduling DAGs on multi-core architectures is a difficult problem [4] that becomes even more difficult when considering MC tasks. Precedence constraints, combined with the extension of timing budgets, induce a cascade effect propagating potential time drifts to dependent tasks. As a consequence, special care must be taken to ensure schedulability in both modes, but also in case of a mode switch : when a TFE occurs, enough processing time must be available for the remaining tasks to be executed in high-criticality mode.

To tackle this problem, we present in this paper a sufficient property to guarantee correct switches from low (LO)-criticality mode to high (HI)-criticality mode in a multi-core MC system with multi-periodic DAGs. Building on this property, we propose MH-MCDAG, a meta-heuristic to produce MC-correct schedules (as formally defined in [5]). We also provide an efficient implementation of MH-MCDAG, called G-ALAP-LLF. Based on Global-Least Laxity First (G-LLF), G-ALAP-LLF executes HI-criticality tasks as late as possible,

[1]SCADE - http://www.esterel-technologies.com/products/scade-suite/
[2]Simulink - https://www.mathworks.com/products/simulink.html

2576-3172/18/$31.00 ©2018 IEEE
DOI 10.1109/RTSS.2018.00042

giving more flexibility for the execution of LO-criticality tasks. Last but not least, we provide an evaluation framework to generate randomly configured multi-periodic MC-DAGs. Experimental results we provide show that our scheduling strategy has a promising acceptance rate when scheduling randomly generated systems. We also provide evidence that our method outperforms existing approaches of the literature [5].

To the best of our knowledge, the problem we address in this paper has received very few contributions so far (*i.e.* [5], [6] and [7]). MH-MCDAG, the meta-heuristic we propose, paves the way for new contributions on this topic. In addition, our evaluation framework could be reused and extended to evaluate future contributions on this topic.

The remainder of this paper is organized as follows. Our task model is presented in Section II. In Section III, we define and prove a sufficient condition to guarantee correct mode switches in the MC scheduling of our task model. We also present MH-MCDAG, our meta-heuristic to schedule DAGs of MC tasks. In Section IV we present G-ALAP-LLF, an efficient implementation of MH-MCDAG, based on G-LLF. We also describe how to adapt it to define an implementation based on Global-Earliest Deadline First (G-EDF). Our experimental results are given in Section V. In Section VI we present related works, to finally conclude in Section VII.

II. Dual-criticality Multi-Periodic MC-DAG Model

We present in this section the notations of the system model we consider throughout our contribution. A motivating example representing an Unmanned Aerial Vehicle (UAV) for field exploration is also presented in this section.

A. Notations

We define a MC system (MCS) $\mathcal{S} = (\mathcal{G}, \Pi)$, as a set $\mathcal{G}$ of Mixed-Criticality DAGs (MC-DAGs) and an architecture Π with m identical processors. We consider dual-criticality systems: the system has two execution modes HI and LO. We adopt the *discard* approach [8], *i.e.* if a TFE occurs, LO-criticality tasks are no longer executed and HI-criticality tasks have an extended timing budget.

A MC-DAG $G_j \in \mathcal{G}$, is defined as follows:

- $T_j \in \mathbb{N}$ and $D_j \in \mathbb{N}$ are respectively the period and deadline of the DAG G_j. Without loss of generality, we consider implicit deadlines (*i.e.* $D_j = T_j$) for the remainder of this paper.
- V_j is the set of vertices of G_j. Each vertex $\tau_i \in V_j$ is a MC task, characterized by two WCET parameters $C_i(LO)$ and $C_i(HI)$. Each task is also characterized with a criticality level $\chi_i \in \{\text{HI, LO}\}$. LO-criticality tasks have a $C_i(HI) = 0$, since we adopt the discard approach. In addition, note that a task $\tau_i \in V_j$ has the same period (T_j) and deadline (D_j) as its DAG G_j.
- $E_j \in (V_j \times V_j)$ is the set of edges between vertices. If $(\tau_i, \tau_j) \in E_j$, then task τ_i must finish its execution before task τ_j can begin. A vertex is **ready** to be executed as soon as all of its predecessors have been executed.

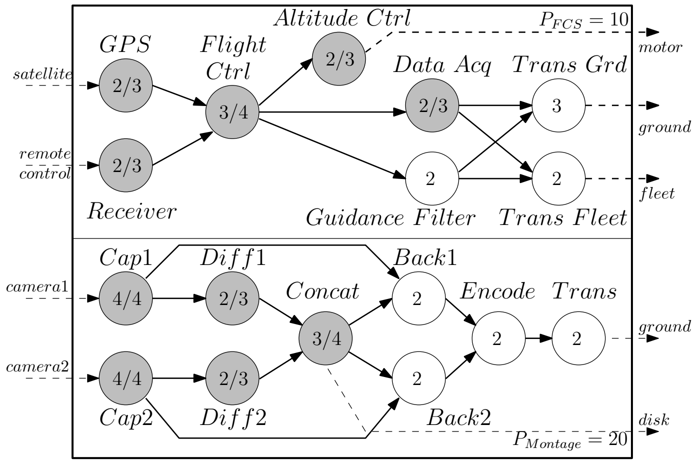


Fig. 1: UAV system $\mathcal{S}$ with two MC-DAGs: a Flight control system FCS and an image processing workload $Montage$.

We define $succ(\tau_i)$ (resp. $pred(\tau_i)$) the set of successors (resp. predecessors) of a task. The predecessors of a HI-criticality task cannot contain a LO-criticality task: for safety reasons, a HI-criticality task cannot depend on the execution of a LO-criticality task. If a mode switch occurs, the LO-criticality task might never be executed and the precedence constraints would never be satisfied.

Task jobs refer to the activations of a periodic tasks of a given DAG. $j_{i,k}$ denotes k-th job of a task τ_i. A task job is said to be released at time instant $r_{i,k} = k \times T_j$ and has to complete its execution before $d_{i,k} = (k+1) \times D_j$: the release time (respectively deadline) of a task job is also the release time (resp. deadline) of a DAG job. We define the utilization rate for a MC-DAG (G_j) in the execution mode χ: $U_\chi(G_j) = \sum_{\tau_i \in G_j} C_i(\chi)/T_j$. For a system $\mathcal{S}$ with multi-periodic MC-DAGs, the utilization rate of $\mathcal{S}$ in mode χ is given by: $U_\chi(\mathcal{S}) = \sum_{G_j \in \mathcal{S}} U_\chi(G_j)$. To have a schedulable system, we know that the condition $m \geq \lceil U_\chi(\mathcal{S}) \rceil$ must be satisfied in HI and LO-criticality modes.

B. Motivating Example: Unmanned Aerial Vehicle

In the remainder of this paper we describe our contribution using a motivating example of an UAV for field exploration. Its graphical representation is presented in Fig. 1.

The UAV is composed of two MC-DAGs: the first one takes care of the Flight Control System (FCS), noted G_{FCS}, the representation is extracted from the services presented in [9]. The second MC-DAG represents a scientific workflow used for image processing [10], noted $G_{Montage}$. Vertices in gray represent HI-criticality tasks, while white vertices are LO-criticality tasks. Vertices are annotated with their timing budgets: a single value is given for LO-criticality tasks since they are not executed in the HI-criticality mode. Full edges represent data dependencies between tasks, while dashed edges represent where data is coming from/being sent to. The idea behind this motivating example is to demonstrate that the FCS could be executed next to an image processing workflow on a tri-core architecture, the lower bound of required processors ($\lceil U_{max\{HI,LO\}}(\mathcal{S})\rceil = 3$).

This motivating example is interesting for several reasons. First, embedded systems like UAVs have limited processing power due to energy consumption and weight constraints. Deploying a large number of distinct processors has an important impact in terms of energy required to power the architecture and lift the vehicle from the ground. Yet, nowadays, an UAV like the one illustrated in Fig. 1 would include two distinct processors: one for the FCS and another for the image processing. By incorporating a multi-core architecture we gain in energy consumption and weight. Second, the HI-criticality mode of the FCS ensures the safety of the UAV by keeping it on the air. At last, by having HI-criticality tasks in the image processing MC-DAG, we are capable of delivering a degraded version of its application.

III. MC-CORRECT SCHEDULING

An MC-correct scheduling for DAGs was defined in [5]:

Definition 1. *A **MC-correct** schedule is one which guarantees*

1) ***Condition LO-Mode****: If no vertex of any MC-DAG in $\mathcal{G}$ executes beyond its $C_i(LO)$ then all the vertices complete execution by the deadlines; and*
2) ***Condition HI-Mode****: If no vertex of any MC-DAG in $\mathcal{G}$ executes beyond its $C_i(HI)$ then all the vertices that are designated as being of HI-criticality complete execution by their deadlines.*

Condition LO-Mode in Definition 1 ensures the schedulability in LO mode: as long as LO tasks execute within their $C_i(LO)$ (*i.e.* no TFE occurs), a MC-correct scheduling satisfies task deadlines and precedence constraints. **Condition HI-Mode** in Definition 1 states that when HI-criticality vertices need to execute until their $C_i(HI)$ (*i.e.* when a TFE occurs), a MC-correct scheduling satisfies deadlines and precedence constraints of HI tasks.

In this section, we present and prove a sufficient condition that guarantees **Condition HI-Mode** of MC-correct scheduling. Building on this result, we define a meta-heuristic for MC-correct scheduling of DAGs.

A. Safe mode transition, a sufficient property

For each task τ_i executing in mode χ, we define the function ψ_i^{χ} as follows:

$$\psi_i^{\chi}(t_1, t_2) = \sum_{s=t_1}^{t_2} \delta_i^{\chi}(s). \qquad (1)$$

where

$$\delta_i^{\chi}(s) = \begin{cases} 1 & \text{if } \tau_i \text{ is running at time } s \text{ in mode } \chi, \\ 0 & \text{otherwise} \end{cases}.$$

This function defines the execution time allocated to task τ_i in mode χ from time t_1 to time t_2.

Definition 2. ***Safe Transition Property***

$$\psi_i^{LO}(r_{i,k}, t) < C_i(LO) \Rightarrow \psi_i^{LO}(r_{i,k}, t) \geq \psi_i^{HI}(r_{i,k}, t). \quad (2)$$

Safe Trans. Prop. states that, while the k-th activation of HI task τ_i has not been fully allocated in LO mode, the budget allocated to this job in LO mode must be greater than the one allocated to it in HI mode. Intuitively, this guarantees that whenever a TFE occurs, the final budget allocated to the job of τ_i is at least equal to its WCET in HI mode.

Definition 3. *A **correct** schedule in a given mode χ, is a schedule that respects the deadline and precedence constraints on all task jobs of the MCS considering their $C_i(\chi)$.*

We propose the following sufficient condition to ensure **Condition HI-Mode** of MC-correct scheduling : the scheduling of tasks is correct in HI mode and **Safe Trans. Prop.** is enforced in LO mode (Definition 3). This result is formalized in Theorem 1 and proved in the remainder of this subsection.

Theorem 1. *To ensure **Condition HI-Mode** of MC-correct scheduling (Definition 1), it is sufficient to define a correct schedule in HI mode and from this, define a correct schedule in LO mode respecting **Safe Trans. Prop.***

Proof. Assume a TFE occurs at time t, and consider the job $j_{i,k}$ of any HI task τ_i. At time t, $j_{i,k}$ has been executed for $\psi_i^{LO}(r_{i,k}, t)$ (see Eq. 1).

Case 1. If $\psi_i^{LO}(r_{i,k}, t) = C_i(LO)$, $j_{i,k}$ has completed its execution at time t. τ_i was completely executed in LO mode, and met its deadline. Indeed, the scheduling in LO mode is correct and ensures that all tasks meet their deadlines if they all execute within their $C_i(LO)$.

Case 2. If $\psi_i^{LO}(r_{i,k}, t) < C_i(LO)$, as a TFE occurs, the scheduling strategy triggers the HI mode. Basically, it stops the LO-criticality scheduling to start the HI-criticality one at time instant t. The WCET of $j_{i,k}$ is also updated to $C_i(HI)$. At time instant t, job $j_{i,k}$ in LO mode has already been executed for $\psi_i^{LO}(r_{i,k}, t)$. At the time instant t, when the TFE occurs, the job $j_{i,k}$ has $C_i(HI) - \psi_i^{HI}(r_{i,k}, t)$ of execution time available to complete its execution in HI mode. We want to know if the allocated budget is large enough to respect the deadline $d_{i,k}$ after the mode switch.

We define $B_{i,k}(t)$ as the budget allocated to job $j_{i,k}$ in LO mode in time interval $[r_{i,k}, t]$, plus the budget allocated to it in HI mode in time interval $[t, d_{i,k}]$ (remember that t is the time the TFE occurred). More formally,

$$B_{i,k}(t) = \psi_i^{LO}(r_{i,k}, t) + \psi_i^{HI}(t, d_{i,k})$$

Since $C_i(HI) = \psi_i^{HI}(r_{i,k}, t) + \psi_i^{HI}(t, d_{i,k})$, we have:

$$B_{i,k}(t) = \psi_i^{LO}(r_{i,k}, t) + C_i(HI) - \psi_i^{HI}(r_{i,k}, t)$$

Enforcing **Safe Trans. Prop.** in LO mode, we know that $\psi_i^{LO}(r_{i,k}, t) \geq \psi_i^{HI}(r_{i,k}, t)$. Therefore:

$$\begin{aligned} B_{i,k}(t) &\geq \psi_i^{HI}(r_{i,k}, t) + C_i(HI) - \psi_i^{HI}(r_{i,k}, t) \\ &\geq C_i(HI). \end{aligned}$$

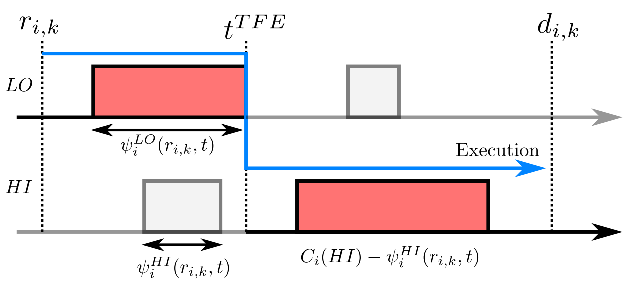


Fig. 2: Illustration of case 2: $\psi_i^{LO}(r_{i,k},t) < C_i(LO)$.

We conclude that the budget allocated to job $j_{i,k}$ when a TFE occurs, is large enough to complete its execution within its HI-criticality WCET.

To better understand the proof, Fig. 2 illustrates the behavior of the scheduling strategy respecting **Safe Trans. Prop.** and being schedulable in LO and HI modes. The red rectangles represent the available computation time for a job $j_{i,k}$ when a TFE occurs. It is clear from the figure that this computation time is large enough for the job to complete its execution within its $C_i(HI)$.

$\square$

B. Meta-heuristic for multi-periodic MC-DAG scheduling

In this subsection we define MH-MCDAG, a meta-heuristic to schedule multiprocessor systems executing multi-periodic MC-DAGs. The meta-heuristic computes two *time-triggered (TT) scheduling tables* [11], S^{LO} and S^{HI}, one for each criticality mode. These tables are computed *off-line* to ease the calculus of Eq. 1 (*i.e.* $\psi_i^{\chi}(t_1,t_2)$). In order to enforce **Safe Trans. Prop.**, we have to schedule HI-criticality tasks in LO mode by taking into account how these tasks are scheduled in HI mode. The first step calclates the schedule of the system in HI mode by applying a suitable scheduling algorithm off-line, G-EDF for instance [12]. If the schedule is correct in HI mode, we then compute the schedule in LO mode (the same scheduling algorithm can be used again) but we enforce **Safe Trans. Prop.** to guarantee **Condition HI-Mode** of MC-correctness (Definition 1). If the resulting schedule is correct in LO mode, **Condition LO-Mode** of MC-correctness is also satisfied and therefore the schedule is MC-correct. Thanks to the TT scheduling tables, when a TFE occurs, the mode switch to the HI-criticality mode just consists in changing scheduling tables from S^{LO} to S^{HI}. Note that, for safety reasons, reactive and safety-critical systems like the UAV illustrated in Fig. 1 are often scheduled with TT tables.

Definition 4. MH-MCDAG *Meta-heuristic for multi-periodic MC-DAG scheduling*

1) *Compute the scheduling table in HI mode and check its correctness.*
2) *Compute the scheduling table in LO mode, enforcing **Safe Trans. Prop.** Check correctness of the schedule in LO mode.*

To the best of our knowledge, [5], [6] and [7] are the only works considering multi-periodic MC-DAGs. These approaches, called federated scheduling, create clusters (*i.e.* set of cores) in the multi-core architecture and allocate an exclusive cluster to each *heavy* DAG (*i.e.* a MC-DAG with a utilization greater than 1, Definition 2 of [5]). A heuristic to schedule *heavy* DAGs on their clusters is then applied. The remaining DAGs, with an utilization rate inferior to 1, are allocated to the remaining cores scheduled as a set of independent tasks: all tasks of a DAG are executed sequentially and are considered to be part of a single bigger task. These bigger tasks are then scheduled using existing methods for independent MC tasks [2].

As a matter of fact, the federated approach presented in [5] fits in our meta-heuristic: it first schedules tasks in HI mode with a non-preemptive List Scheduling (LS), and checks its correctness. Then tasks are scheduled in LO mode with a Preemptive LS. The HI-criticality jobs in LO mode have the priority ordering they had in HI mode and all HI-criticality jobs have a higher priority than LO-criticality jobs. Thus, HI-criticality jobs always execute earlier in LO mode than in HI mode and **Safe Trans. Prop.** is implicitly verified.

In this section, we have proposed a meta-heuristic to schedule multi-periodic MC DAGs. This approach requires a scheduling algorithm for periodic tasks and task precedences as well as the enforcement of a specific condition to guarantee a safe switch from LO mode to HI mode. In the next section, we propose to instanciate this meta-heuristic with a specific scheduling algorithm in order to optimise the acceptance ratio.

IV. G-ALAP-LLF, AN ALAP INSTANCE OF MH-MCDAG

Like we mentioned in the previous section, the federated approach [5] fits the definition of MH-MCDAG. However, while this approach has the benefit of being quite simple, it leads to poor resource usage: for example, if a system is composed of a *heavy* DAG with a utilization rate of 3.1 and a DAG with a utilization of 0.2, 5 cores will be needed (a cluster of 4 cores for the *heavy* DAG and 1 core for the other DAG), while we can expect to find a MC-correct schedule on 4 cores. For this reason, we aim at defining more efficient scheduling methods in terms of usage of multi-core architectures.

In this section, we present G-ALAP-LLF, an implementation of MH-MCDAG, which aims at optimizing the acceptance rate for MCS. This implementation is based on G-LLF, and we explain in this section how it can be adapted to an implementation of MH-MCDAGbased on G-EDF. As opposed to the federated techniques [5], [6], [7], we adopt a *global* scheduling approach.

A. Motivation and overview

To design an efficient scheduling strategy for multi-processor MCS composed of multi-periodic MC-DAGs, we instantiate the meta-heuristic described in the previous section.

In order to have a good acceptance rate for the implementation of MH-MCDAG, we have to:

1) Adapt an efficient scheduling algorithm, in terms of acceptance rate, for periodic tasks with precedence constraints. The scheduling policy is used off-line to obtain

the scheduling tables. The scheduling policy is applied to each of the criticality modes;

2) Enforce **Safe Trans. Prop.** without reducing the exploration space of the possible schedules in LO mode; for instance not limiting ourselves to schedules where HI-criticality jobs have priority over all LO-criticality jobs.

In our implementation, we consider a global scheduling algorithm for each operational mode. In general, to adapt a scheduling algorithm to fit in our meta-heuristic, the algorithm needs to be applied off-line in order to obtain scheduling tables. It also needs to be compatible with the enforcement of **Safe Trans. Prop.**, *i.e.* we need to be capable of counting time slots allocated to tasks' jobs in each operational mode. G-LLF and G-EDF are global algorithms respecting these conditions.

To ease the computation of the LO scheduling table, we want to ease the enforcement of **Safe Trans. Prop.** To do so, $\psi_i^{HI}(r_{i,k}, t)$ should be kept minimal as long as $\psi_i^{LO}(r_{i,k}, t) < C_i(LO)$. In other words, HI-criticality tasks in HI mode should be scheduled as late as possible (ALAP) and not as soon as possible (ASAP). This gives a good scheduling flexibility for HI-criticality tasks in LO mode and as a consequence, a good scheduling flexibility for LO-criticality tasks in LO mode.

To obtain such a behavior on HI-criticality tasks in HI mode (*i.e.* the HI-criticality task is ready to be executed but waits until the last possible instant to be allocated) we perform a common transformation in task graph theory [4] on our MCS. To schedule the HI mode task set with a ALAP strategy, **we produce the dual task graph** and schedule this new task set with an ASAP strategy. This transformation consists in substituting ALAP deadlines (resp. arrival times) as ASAP arrival times (resp. deadlines) and **inverting the precedence constraints on all MC-DAGs**.

In order to compute scheduling tables in each mode using G-LLF, we have to define a laxity formula in the context of tasks with precedence constraints. Let us first consider the common concept in task graph theory, the **critical path**. The critical path (CP_i^χ) of a vertex τ_i in mode χ is the longest path, considering the $C_i(\chi)$ of vertices, to reach an exit vertex (*i.e.* a vertex with no successors). Let now define the laxity of task τ_i, at time instant t, in the criticality mode χ as follows:

$$L_{i,k}^\chi(t) = d_{i,k} - t - (CP_i^\chi + R_{i,k}^\chi) \tag{3}$$

$d_{i,k}$ is the deadline of the job $j_{i,k}$. $R_{i,k}^\chi$ is the remaining execution time of the job $j_{i,k}$, *i.e.* $R_{i,k}^\chi = C_i(\chi) - \psi_i^\chi(r_{i,k}, t-1)$.

A similar approach is used to compute scheduling tables using G-EDF. We define the priority for each task job $P_{i,k}$ as follows, where d_i is the *relative* deadline for the job $j_{i,k}$:

$$P_{i,k}^\chi = d_i - CP_i^\chi. \tag{4}$$

For space limitation reasons, we only develop the G-LLF version in the remainder of the paper. Theoretically, this version is expected to dominate the G-EDF version in terms of acceptance rate [13]. Here, theoretically means that preemption and migration costs are ignored.

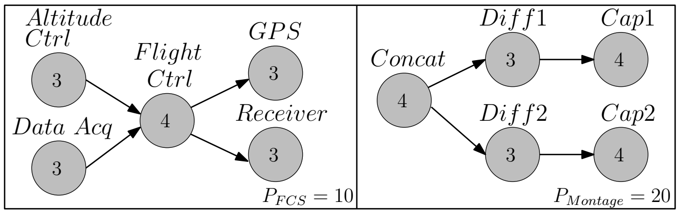


Fig. 3: Transformation of $\mathcal{S}$ to its dual $\mathcal{S}^\star$

Algorithm 1 Computation of the HI scheduling table

```
1: function CalcSHI($\mathcal{S}$: MCS to schedule)
2:     Transform system $\mathcal{S}$ to $\mathcal{S}^\star$
3:     $R_{list} \leftarrow \emptyset$                                   ▷ List of ready tasks
4:     for all HI tasks $\tau_i$ do
5:         $R_i^{HI} \leftarrow C_i(HI)$              ▷ Remaining execution time
6:         $R_{list} \leftarrow R_{list} \cup \{\tau_i \mid pred(\tau_i) = \emptyset\}$
7:     end for

8:     for all timeslots $t < H$ do
9:         SortHI($R_{list}, t$)
10:        if VerifyConstraints($R_{list}, t$) = $\perp$ then
11:            return Not Schedulable
12:        end if
13:        for all cores $c \in \Pi$ do
14:            $\tau_i \leftarrow$ head of $R_{list}$ not being allocated
15:            $S^{HI}[t][c] \leftarrow \tau_i$                       ▷ Allocate task $\tau_i$
16:            $R_i^{HI} \leftarrow R_i^{HI} - 1$
17:        end for
18:        $R_{list} \leftarrow R_{list} \cup$
               $\{succ(\tau_i) \mid \forall \tau_j \in pred(succ(\tau_i)), R_j^{HI} = 0\}$
19:        $R_{list} \leftarrow R_{list} \setminus \{\tau_i \in R_{list} \mid R_i^{HI} = 0\}$
20:        for all $G_j \in \mathcal{G}$ do                    ▷ Reactivation of $G_j$
21:            if $t + 1 \mod T_j = 0$ then
22:                $\forall \tau_i \in G_j,\ R_i^{HI} \leftarrow C_i(HI)$
23:                $R_{list} \leftarrow R_{list} \cup \{\tau_i \in G_j \mid pred(\tau_i) = \emptyset\}$
24:            end if
25:        end for
26:    end for
27:    Reverse the scheduling table $S_{HI}$
28:    return $S^{HI}$ scheduling table
29: end function
```

B. Computation of the HI scheduling table

The first step to obtain a HI scheduling table consists in transforming the MCS $\mathcal{S}$ into its dual $\mathcal{S}^\star$, as illustrated on Fig. 3 for our UAV use-case. Only HI-criticality tasks are represented with their respective timing budget $C_i(HI)$. A scheduling table is computed on $\mathcal{S}^\star$ using G-LLF. We then obtain the scheduling table of $\mathcal{S}$ by reversing the table obtained for $\mathcal{S}^\star$ (*i.e.* we perform a horizontal flip). The algorithm used to compute the S^{HI} table is presented by Alg. 1. This algorithm uses a **ready list** of jobs and orders its element based on their laxity computed as described in Formula 3.

The ready list is initialized (lines 3-7 in Alg. 1) with source vertices (*i.e.* vertices with no entry edges) of $\mathcal{S}^\star$:

Altitude Ctrl, *Data Acq* and *Concat* in Fig. 3. We then proceed to allocate tasks at each time slot until we reach the hyper-period ($H = 20$ for $\mathcal{S}^\star$). For each time slot t, we call the function SORTHI that considers the results of Eq. 3 to sort the list (l. 9). Once the tasks have been ordered, the function VERIFYCONSTRAINTS is called. This function checks if obtaining a feasible schedule is still possible. Constraints that need to be respected are: (i) vertices do not have negative laxities, (ii) no more than m vertices have zero laxities, and (iii) the sum of remaining times for all jobs needs to be inferior to the number of available slots remaining in the table.

In order to obtain another implementation of MH-MCDAG, we just need to redefine SORTHI and VERIFYCONSTRAINTS. For a G-EDF based implementation, SORTHI uses Formula 4 and VERIFYCONSTRAINTS only checks that jobs' deadlines are respected since there are no laxities.

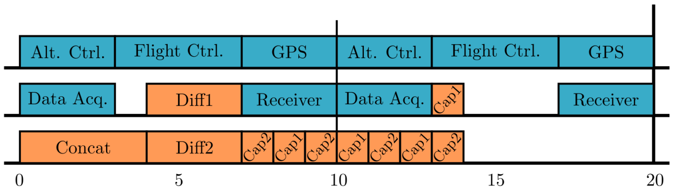
After performing the sorting in the list and checking that the scheduling tables are still obtainable, the heuristic grabs the m-first ready jobs of the list and allocates the tasks to the multi-core architecture (l. 13-17). Once the allocation for this timeslot has been performed, the ready list can be updated in two ways: (i) new jobs can become ready to execute because their precedence constraints are satisfied (l. 18) and (ii) jobs that finished their execution are removed from the list (l. 19).

We illustrate the algorithm on our motivating example. At time slot 0, for example, $L^{HI}_{Altitude}(0) = 3 - 0 - 3 = 0$, $L^{HI}_{Data}(0) = 3 - 0 - 3 = 0$ and $L^{HI}_{Concat}(0) = 13 - 0 - 4 = 9$. They are the only three ready tasks until time slot 3, so *Altitude Ctrl* and *Data Acq* are fully allocated. At time slot 3, *Altitude Ctrl* and *Data Acq* have finished their executions, activating the *Flight Ctrl* task. At time slot 4, the *Concat* task finishes its execution, activating $Diff1$ and $Diff2$. We have enough cores to execute all active tasks until time slot 7. At time slot 7, the *Receiver* and *GPS* tasks from the *FCS* become active, as well as the $Cap1$ and $Cap2$ tasks from the *Montage* MC-DAG. We have the following laxities $L^{HI}_{Receiver}(7) = 10 - 7 - 3 = 0$, $L^{HI}_{GPS}(7) = 10 - 7 - 3 = 0$, $L^{HI}_{Cap1}(7) = 20 - 7 - 4 = 9$ and $L^{HI}_{Cap2}(7) = 20 - 7 - 4 = 9$. *Receiver*, *GPS* are going to run since they have the lower laxities. The tie is broken arbitrarily for $Cap2$ and $Cap1$, so $Cap2$ will execute first.

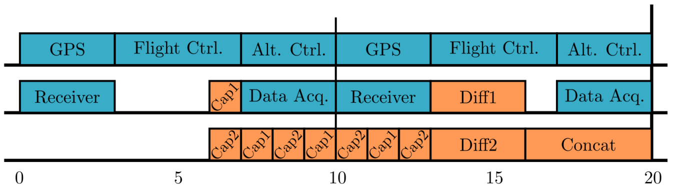
MC-DAGs can have new activations during the computation of the tables: if the slot considered corresponds to the period of a MC-DAGs, its sources are added to the list once again and all R^{HI}_i are reset to $C_i(HI)$ (l. 20-25). At time slot 10, MC-DAG G_{FCS} starts a new execution since it is executed twice during the hyper-period. The final HI scheduling table is presented in Fig 4a. Tasks belonging to G_{FCS} are illustrated in blue, while tasks belonging to $G_{Montage}$ are illustrated in orange. The last step (l. 27) performs the horizontal flip on the the scheduling table as illustrated in Fig 4b.

C. Computation of the LO scheduling table

The algorithm used to compute the S^{LO} table is presented by Alg. 2. It is very similar to the algorithm used to schedule the dual system $\mathcal{S}^\star$. The main difference relies on the fact that



(a) HI scheduling on the dual system $\mathcal{S}^\star$



(b) HI scheduling of $\mathcal{S}$ with maximized response time

Fig. 4: HI scheduling table inversion

Algorithm 2 Computation of the LO scheduling table

```
1: function CALCSLO($\mathcal{S}$: MCS to schedule)
2:     $R_{list} \leftarrow \emptyset$                         ▷ List of ready tasks
3:     for all $\tau_i$ do
4:         $R_i^{LO} \leftarrow C_i(LO)$         ▷ Remaining execution time
5:         $R_{list} \leftarrow R_{list} \cup \{\tau_i \mid pred(\tau_i) = \emptyset\}$
6:     end for

7:     for all timeslots $t < H$ do
8:         SORTLO($R_{list}, t$)
9:         if VERIFYCONSTRAINTS($R_{list}, t$) $= \perp$ then
10:            return NOT SCHEDULABLE
11:        end if
12:        for all cores $c \in \Pi$ do
13:            $\tau_i \leftarrow$ head of $R_{list}$ not being allocated
14:            $S^{LO}[t][c] \leftarrow \tau_i$                 ▷ Allocate task $\tau_i$
15:            $R_i^{LO} \leftarrow R_i^{LO} - 1$
16:        end for
17:        $R_{list} \leftarrow R_{list} \cup$
               $\{succ(\tau_i) \mid \forall \tau_j \in pred(succ(\tau_i)), R_j^{LO} = 0\}$
18:        $R_{list} \leftarrow R_{list} \setminus \{\tau_i \in R_{list} \mid R_i^{LO} = 0\}$
19:        for all $G_j \in \mathcal{G}$ do              ▷ Reactivation of $G_j$
20:            if $t + 1 \mod T_j = 0$ then
21:                $\forall \tau_i \in G_j,\ R_i^{LO} \leftarrow C_i(LO)$
22:                $R_{list} \leftarrow R_{list} \cup \{\tau_i \in G_j \mid pred(\tau_i) = \emptyset\}$
23:            end if
24:        end for
25:    end for
26:    return $S^{LO}$ scheduling table
27: end function
```

Safe Trans. Prop. must be ensured to follow the definition of our meta-heuristic (Definition 4).

To respect **Safe Trans. Prop.** during the computation of our scheduling table, we define SORTLO, the sorting function for jobs in the LO-criticality mode. It defines the following priority ordering:

1) HI-criticality tasks that would invalidate **Safe Trans.**

Prop. if they are not allocated at time slot t, are considered as tasks with zero laxity.

2) The rest of ready tasks are sorted according to their laxity (Formula 3).

Once the sorting is obtained for the ready list, like for the HI-criticality mode, we call the VERIFYCONSTRAINTS function to check if a feasible schedule can be obtained, constraints are the same than in the HI-criticality mode. To obtain the G-EDF implementation we just need to redefine SORTLO to consider Formula 4 when HI-criticality tasks do not risk to invalidate **Safe Trans. Prop.** VERIFYCONSTRAINTS is also adapted to check if deadlines are respected.

We illustrate the computation of the S^{LO} applied to system of Fig. 1. At time slot 0, we have the following laxities: $L^{LO}_{GPS}(0) = 2 - 0 - 2 = 0$, $L^{LO}_{Receiver}(0) = 2 - 0 - 2 = 0$, $L^{LO}_{Cap1}(0) = 9 - 0 - 4 = 5$ and $L^{LO}_{Cap2}(0) = 9 - 0 - 4 = 5$. Tasks of G_{FCS} have the lowest laxity and they are allocated first. The third processor is shared by tasks $Cap1$ and $Cap2$ until time slot 2, when GPS and $Receiver$ have completed their execution. The algorithm continues to allocate tasks in function of their laxities until slot 17. At this time slot, tasks $Altitude\ Ctrl$, $Trans\ Grd$, $Trans\ Fleet$ and $Trans$ are in the ready list. However, $Altitude\ Ctrl$ in the HI scheduling table is allocated at this time slot and this job has not started to be allocated. The application of SORTLO puts $Altitude$ at the beginning of the list.

The final scheduling table is presented in Fig 5. As we can see, our scheduling strategy respects the deadline of the system in both modes, and satisfies **Safe Trans. Prop.** during the computation of the LO scheduling table. The computed scheduling tables are therefore considered as MC-correct in a tri-core architecture (the minimal number of cores required to schedule the system).

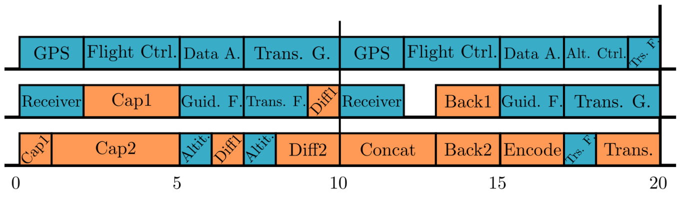


Fig. 5: LO scheduling table

D. Analysis of G-ALAP-LLF

In this subsection, we explain the expected advantages of G-ALAP-LLF over the federated approach [5]. Fig. 6 illustrates the scheduling tables obtained by the federated approach [5]. As we can see, this approach requires five cores to find a feasible schedule in the LO mode, whereas we found a MC-correct one using three cores.

First, in G-ALAP-LLF, MC-DAGs are scheduled using a *global* algorithm in both LO and HI modes. To use a different algorithm, like G-EDF, the only necessary change is to modify the functions SORTHI, SORTLO and VERIFYCONSTRAINTS of Algorithm 1 and 2 in order to use Formula 3 (G-LLF) or 4 (G-EDF). The fact that our scheduling is global, is one of the two major reasons for obtaining a better resource utilization.

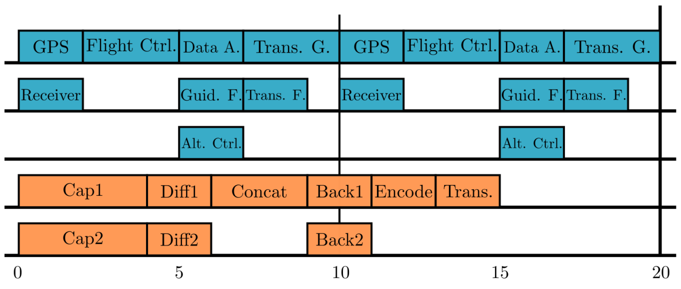


Fig. 6: LO scheduling table produced by the federated approach [5]

Indeed, the G-ALAP-LLF method can schedule any task on any core whereas the federated approach clusterizes cores for *heavy* DAGs and execute *light* DAGs as sequential tasks. The second major source of optimization comes from the computation of the HI scheduling table. Scheduling HI-criticality tasks in HI mode ALAP preserves the possibility to allocate LO-criticality tasks on more timeslots. More formally the $\psi^{HI}_i(r_{i;k}, t)$ of G-ALAP-LLF has higher chances to increase at a later point t, as opposed to the $\psi^{HI}_i(r_{i;k}, t)$ of the federated approach, which schedules HI-criticality tasks ASAP.

Nonetheless, an advantage of the federated approach over ours, is that if tasks complete their execution before their $C_i(\chi)$, their successors can start their execution sooner. This is possible thanks to the priority ordering that was computed. Our approach, on the other hand, executes the system in a TT fashion. As demonstrated in [14], if the TT approach is much more predictable, it offers a less efficient resource utilization compared to the event-triggered approach. In addition, the federated approach will generate less preemptions and migrations than G-ALAP-LLF.

V. EXPERIMENTAL RESULTS

In this section we present our experimental results. The objective is to evaluate the performance of G-ALAP-LLF. We proceed to this evaluation by (i) generating random sets of MC-DAGs and (ii) measuring the ratio of systems for which our method finds a MC-correct schedule.

Following this method, we also compare G-ALAP-LLF to the federated scheduling [5]. To our knowledge, it is the only contribution that considers a task model similar to ours, composed of (i) MC tasks, (ii) precedence constraints, and (iii) multiple deadlines for DAGs. The approaches of [6], [7] have very restrictive MC-DAGs were all vertices belong to the same criticality level and thus, do not correspond to the type of applications we consider.

A. Experimental setup - MC-DAG Framework

For our experiments, we implemented the G-ALAP-LLF algorithm in an open-sourced framework[3]. In addition, since works in [5] have only presented theoretical results, we also implemented the federated approach. Last but not least, we developed a MCS generator in order to produce many MCS with random properties. Thanks to these tools, we generated a

[3]MC-DAG Framework - https://github.com/robertoxmed/MC-DAG

set of MCSs and measured the ratio for which each scheduling method finds a MC-correct schedule.

The random generation needs to be **unbiased** and **uniformly cover** the possible timing configurations of MCS. To design this random generation, we first integrated existing methods to generate DAGs with unbiased topologies [15]. This is an important aspect, since certain DAG-shapes tend to be more schedulable than others. The distribution of execution time for tasks is not controlled by this DAG generation approach. Yet, the utilization of the system is the most important factor used to perform benchmarks on real-time scheduling techniques. To overcome this limitation, we have integrated existing methods achieving a uniform distribution of utilizations for tasks [16], [17].

Parameters for the generation of MCS are as follows: • U: Utilization of the system in both criticality modes. • $|\mathcal{G}|$: Fixed number of MC-DAGs per system. • $|V_j|$: Fixed number of vertices per MC-DAG, *i.e.* all MC-DAGs have the same number of vertices. • ρ: Ratio of HI criticality tasks. • f: Reduction factor for the utilization of HI tasks in LO mode. • e: Probability to have an edge between two vertices.

Once these parameters are set, we first distribute uniformly the utilization of the system to each MC-DAG. We use the uniform distribution described in [16] to assign a utilization for each MC-DAG. The period/deadline for each MC-DAG is then assigned randomly: this period is chosen from a predefined list of numbers in order to avoid prime numbers[4] (which are also avoided in the industrial context). With the assignment of the period and the utilization of the MC-DAG, we can distribute the utilization to tasks of the DAG. We use UUnifast-discard [17] in this case. As opposed to the utilization that can be given to DAGs, a vertex cannot have a utilization greater than 1 since it is a *sequential* task (parallel execution for a vertex is not possible). UUnifast-discard is therefore an apropriate method. The utilization available for LO-criticality tasks is given by the difference between the utilization of HI tasks in HI mode and the utilization of HI tasks in LO mode, the difference being controlled by parameter f.

Once the utilization of the system is distributed among MC-DAGs and the utilization of MC-DAGs is distributed among tasks, we start the generation of the topology for the MC-DAGs. We start by creating the HI-criticality vertices. These vertices are connected following the probability e given by the user and without creating cycles among vertices. After the HI-criticality tasks have been created, we create the LO-criticality tasks. Again vertices are connected following the probability e chosen by the user and without creating cycles. The higher the probability e, the more dense is the resulting graph: vertices have more precedence constraints to satisfy, making the scheduling of the system more difficult.

The benchmarks we perform to assess statistically G-ALAP-LLF compared to the federated approach [5] follow these two steps: first, we generate a large number of random MCSs. Second, we measure the percentage of systems for which G-ALAP-LLF and the federated approach find feasible schedules.

[4]Possible periods: $\{100, 120, 150, 180, 200, 220, 250, 300, 400, 500\}$.

B. Acceptance rate

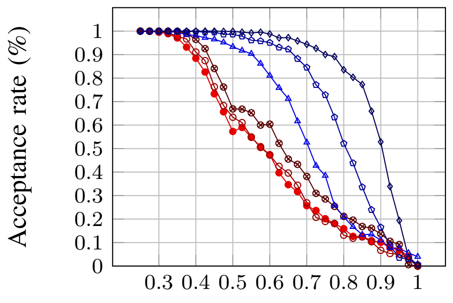
In the remainder of this section, we present and analyze our experimental results. The objective of these experiments is to evaluate the performance of our method in terms of acceptance rate, measured as the percentage of MCS for which our heuristic finds a MC-correct schedule.

Experimental parameters: We control the parameters of the MCS generator so as to measure their influence on the performance of our method. We expect the following parameters to make the scheduling problem more difficult: (i) the density of the graphs, (ii) the utilization of the system, (iii) the utilization per task of the system, (iv) the number of MC-DAGs. Our experiments aim at measuring the effect of these parameters on G-ALAP-LLF's performance.

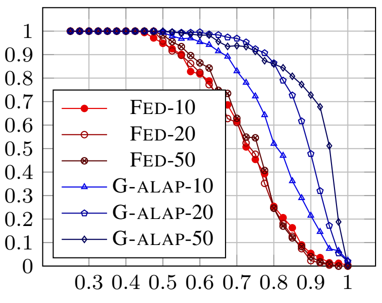
Our experimental results are presented in Fig. 7. The x-axis of the plots is the normalized utilization of the system ($U_{norm} = U(\mathcal{S})/m$), and the y-axis is the acceptance rate. For each point of the graphs, we tested 1000 randomly configured MCS. In our experiments, the ratio of HI tasks is set to $\rho = 50\%$ and reduction factor of HI tasks utilization in LO mode is set to $f = 2$. In addition, we generate MC-DAGs with three configurations in terms of number of tasks $|V_j|$ per MC-DAG: MC-DAGs are composed of 10, 20 and 50 tasks in each configuration. This setup allows us to see how the two scheduling approaches perform when a large number of tasks is considered (up to 200 when considering 4 MC-DAGs).

Influence of the number of vertices: Fig. 7a presents the results obtained with an edge probability $e = 20\%$, two MC-DAGs ($|\mathcal{G}| = 2$), and an architecture of four cores ($m = 4$). Note that a MC-DAG with $e = 20\%$ is already dense enough to represent industrial systems with their usual precedence constraints. This first experiment provides promising results: G-ALAP-LLF found MC-correct schedules for over 50% of systems with a utilization up to 70%. For the federated approach the utilization of the system needs to be below 55% to schedule over 50% of MCS. We also notice that, for G-ALAP-LLF, the acceptance rate increases with the number of tasks per MC-DAG (*e.g.* G-ALAP-50 is above G-ALAP-20). This is due to the fact that for a given system utilization and an increasing number of tasks, the uniform distribution tends to give less utilization to each task: tasks become smaller and therefore easier to schedule with G-ALAP-LLF. We can also notice that the performance of the federated approach is less sensitive to the number of tasks per DAG, which was expected since the federated approach first assigns DAGs to clusters of cores, and then schedules *heavy* DAGs on their clusters.

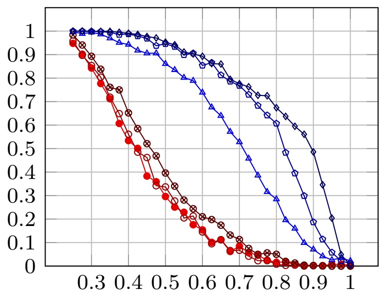
Influence of the number of MC-DAGs: To confirm these observations, we also increased the number of MC-DAGs. As a consequence, tasks become smaller but they are also scheduled with more heterogeneous periods. Fig. 7b provides the results obtained with four MC-DAGs (instead of two in the previous configuration). As we can see, the performances tend to improve for both methods, which tends to demonstrate that (i) the utilization per DAG has a strong influence on the



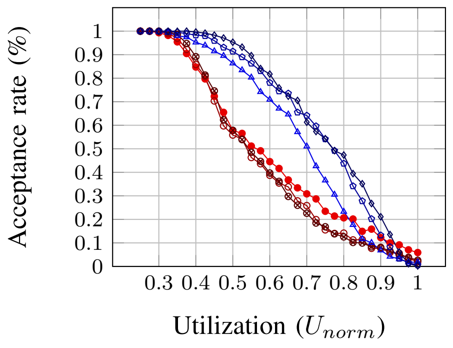
(a) $e = 20\%$, $|\mathcal{G}| = 2$ and $m = 4$.



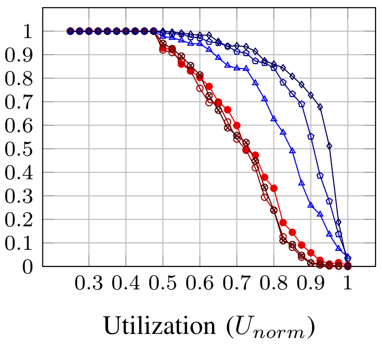
(b) $e = 20\%$, $|\mathcal{G}| = 4$ and $m = 4$.



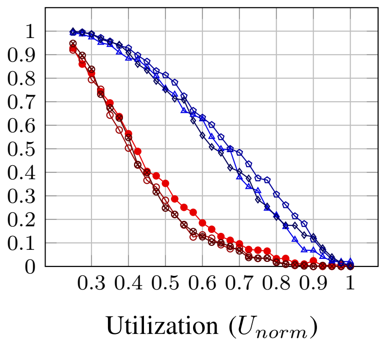
(c) $e = 20\%$, $|\mathcal{G}| = 4$ and $m = 8$.



(d) $e = 40\%$, $|\mathcal{G}| = 2$ and $m = 4$.



(e) $e = 40\%$, $|\mathcal{G}| = 4$ and $m = 4$.



(f) $e = 40\%$, $|\mathcal{G}| = 4$ and $m = 8$.

Fig. 7: Comparison between our scheduling strategy and the federated approach [5]

performance of the federated approach, and (ii) the utilization per task has a strong influence on performance of the G-ALAP method. Conversely, the number of MC-DAGs and thus the heterogeneity of periods seems to have a lower influence.

Influence of the number of cores: Keeping this last configuration, we increase the utilization per task by increasing the number of cores: we consider an architecture of eight cores ($m = 8$). Experimental results are reported in Fig. 7c. As expected, the scheduling problem is more difficult (the utilization per DAG and per task increases with respect to the previous configuration) and the measured performance is degraded. However, we remark that the measured performance is lower than for the initial configuration (Fig. 7a) whereas the utilization per DAG and per task is similar. This is due to the link between two parameters of our experimental setup: when increasing the number of tasks, we increase the number of edges as well. However, the number of potential edges grows quadratically with the number of tasks.

Going from results reported in Fig. 7a to those reported in 7b, the number of tasks increases linearly but the number of edges grows quadratically. Yet, the performances improve because the utilization per DAG and per task becomes smaller. Going from results reported in Fig. 7a to those reported in Fig. 7c, the utilization per DAG and per task remains the same, but the number of edges grows fast. As a consequence, the performance is degraded.

As a conclusion of these first experiments, we can see that the following parameters have a strong influence on the performances of both methods: the utilization of the system and the density of the graphs. In addition, the utilization per tasks influences the performance of the G-ALAP-LLF method while the utilization per DAG influences the performance of the federated approach.

Influence of the probability to have an edge between two vertices: In Fig. 7d, Fig. 7e, and Fig. 7f, we report the results using the same configurations as before, except for the edge probability: in these configurations, $e = 40\%$. These results confirm our previous conclusions: the density of the graph has a strong impact on the performance of the method, as well as the system utilization. Besides, the G-ALAP-LLF method continues to outperform the federated approach.

However, with these configurations, we can make the following observations. First, the number of tasks per DAG has less influence on the performance of G-ALAP-LLF than in the previous experiments (*e.g.* G-ALAP-50 and G-ALAP-10 get closer). This reduction is however less visible in Fig. 7e where the number of tasks is important and the utilization per task is small. In other configurations, the utilization per task is too high to compensate the difficulty to schedule DAGs with such a density ($e = 40\%$). Second, the gap of performances obtained with each method gets smaller (*e.g.* G-ALAP-10 and FED-10 get closer). This tends to show that the configurations considered push our method towards its limits: scheduling problems considered are very hard, which was expected since graphs generated with $e = 40\%$ are very dense. Still, with a high number of tasks and a low utilization per tasks, our results are very good: G-ALAP-50 on Fig. 7e exhibits an acceptance rate above 70% for a system utilization of 90%.

Other values were tested for the parameters of our experiments: number of MC-DAGs, cores, tasks, reduction factor and HI-tasks ratio. All these tests showed a general tendency: G-ALAP-LLF it is capable of scheduling more systems compared to FEDMCDAG [5]. However, we acknowledge that this result remains valid only if we do not account for preemption and migration costs. Indeed, a well-known weakness of laxity based algorithms is that they increase the number of preemptions and migrations. Nevertheless, in these experiments, it would have been difficult to define the timing overhead of preemptions or migrations. In future works we plan to evaluate an implementation of our meta-heuristic, MH-MCDAG, with another scheduling criteria than tasks laxity.

In conclusion, G-ALAP-LLF shows very good performance in terms of acceptance rate, even when comparing the results obtained with G-ALAP-LLF to results obtained on MC scheduling for ***independent*** task sets. Indeed, when $U_{norm} = 0.8$, G-EDF-VD applied on independent tasks has an acceptance rate of 80% [18], while our approach reaches 70% on dependent tasks with $e = 20\%$ and $|\mathcal{G}| = 4$. We expect to have a lesser schedulability rate but the difference is acceptable considering the utilization of the system and the fact that we have precedence constraints.

VI. RELATED WORKS

In this paper, we propose a multi-core scheduling heuristic for real-time systems using the mixed-criticality model. Our heuristic leverages a common limitation regarding MC scheduling [2]: most contributions only consider independent task sets. In our case, we considered applications modeled thanks to DAGs in order to represent data dependencies: a common practice for real-time systems. In the following section we present some relevant related works from the different scientific communities mentioned.

The only contributions considering MC tasks and multiple DAGs are [5], [6] and [7] which compute federated schedulers. The closest task model to ours is the one presented in [5] and as demonstrated in previous sections, our scheduling approach outperforms their method in terms of number of cores required to schedule a system and in schedulability when the number of cores is fixed. In [6], [7], the task model is very restrictive: each DAG belongs to a criticality level, *i.e.* all tasks of the DAG have the same criticality HI or LO. The approach calculates virtual deadlines for HI-criticality DAGs in order to respect mode switches. Nonetheless, our experimental results showed that a global approach tends to schedule more systems in a given number of cores.

Real-time scheduling for **multiple** DAGs has seen recent interesting contributions in the cloud computing domain. Workflows of tasks often in the form of DAGs are distributed into heterogeneous computing clusters. While the execution platforms are different than simple multi-core architectures, some heuristics take into account deadlines for DAGs. In [19] the authors propose a deadline distribution and planning algorithm that aims to minimize the execution time of a workflow. In [20], the authors calculate an urgency factor for each vertex of the DAG by taking into account the execution charge of its successors and the deadline of the DAG. An extension to the Heterogeneous Execution Time First heuristic is presented in [21], in which priorities are assigned to tasks in order to satisfy as many deadlines as possible. While these works consider deadlines for multiple DAGs, the execution budgets for each vertex cannot change during the execution of the system and no mode switches need to be ensured.

The construction of TT scheduling tables for MC systems has been studied in [22]: scheduling tables are constructed by exploring the solution space. This exploration is guided by a leeway parameter allowing the detection of unfeasible schedules to perform backtracking when needed. Nonetheless the approach presented was only tested for mono-processors and our experimentations with Constrained programing considering data-dependencies between tasks and multi-core architecture were not capable of scaling for randomly generated systems. An online mechanism to incorporate MC aspects into TT tables is presented in [23]. It relies on the fact that a scheduling table already exists for the LO-criticality mode and tries to use remaining time slots of the table to handle mode transitions and timing extensions for HI-criticality table. Since our approach computes two compatible scheduling tables, we do not rely on time slots that remained unused in the LO-criticality mode.

VII. CONCLUSION

In this paper, we have presented four contributions. First, we have defined a sufficient property to guarantee correct switches from LO mode to HI mode in a multi-core Mixed-Criticality scheduling for multi-periodic DAGs. Second, based on this property, we have defined and proved MH-MCDAG, a meta-heuristic produce MC-correct schedules for multi-periodic MC-DAGs. The computation of these schedules is performed off-line to ease the enforcement of the safe transition property. Third, we have defined and evaluated G-ALAP-LLF, an instantiation of MH-MCDAG, based on a global scheduling for multi-core execution platforms. Our experimental results have demonstrated that G-ALAP-LLF exhibits very good performances (in terms of acceptance rate) compared to the state-of-the-art. We also have explained how G-ALAP-LLF can be adapted to obtain an implementation of MH-MCDAG based on G-EDF. At last, we proposed an experimental platform to generate multi-periodic MC-DAGs with random configurations. This platform can be used to evaluate MC scheduling methods in terms of acceptance rate.

In the future we plan to extend MH-MCDAG to support an arbitrary number of criticality levels by applying our 2-levels approach by induction. We also plan to study the number of preemptions and migrations produced by G-ALAP-LLF and to use/evaluate other global algorithms entailing fewer preemptions and migrations. Indeed, solutions based on LLF scheduling are known to generate a lot of preemptions and migrations.

Acknowledgment

This research work has been funded by the academic and research chair Engineering of Complex Systems and the Smart, Safe and Secure Platform (S3P) project.

References

[1] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *Real-Time Systems Symposium*, 2007.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," *University of York, Tech. Rep*, 2013.

[3] F. Cadoret, T. Robert, E. Borde, L. Pautet, and F. Singhoff, "Deterministic implementation of periodic-delayed communications and experimentation in aadl," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*. IEEE, 2013, pp. 1–8.

[4] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, 1999.

[5] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *Real-Time Systems Symposium*. IEEE, 2016.

[6] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-Time Systems*, vol. 53, no. 5, 2017.

[7] R. M. Pathan, "Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 106. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[8] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 2010, pp. 90–101.

[9] S. Siebert and J. Teizer, "Mobile 3d mapping for surveying earthwork projects using an unmanned aerial vehicle (uav) system," *Automation in Construction*, vol. 41, 2014.

[10] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workshop on Workflows in Support of Large-Scale Science*. IEEE, 2008.

[11] H. Kopetz, "The time-triggered model of computation," in *rtss*. IEEE, 1998, p. 168.

[12] J. Li, K. Agrawal, C. Lu, and C. Gill, "Outstanding paper award: Analysis of global edf for parallel tasks," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*. IEEE, 2013, pp. 3–13.

[13] J. Lee, A. Easwaran, I. Shin, and I. Lee, "Zero-laxity based real-time multiprocessor scheduling," *Journal of Systems and Software*, 2011.

[14] H. Kopetz, "Event-triggered versus time-triggered real-time systems," in *Operating Systems of the 90s and Beyond*. Springer, 1991, pp. 86–101.

[15] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *International ICST conference on simulation tools and techniques*. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2010.

[16] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems Symposium*, vol. 30, no. 1, 2005.

[17] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Real-Time Systems Symposium*. IEEE, 2009, pp. 398–409.

[18] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. IEEE, 2012, pp. 166–175.

[19] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, no. 1, 2013.

[20] W. Wang, Q. Wu, Y. Yan, and F. Wu, "Maximize Throughput Scheduling and Cost-Fairness Optimization for Multiple DAGs with Deadline Constraint," *International Conference on Algorithms and Architectures for Parallel Processing*, vol. 7016, 2015.

[21] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "Mixed real-time scheduling of multiple DAGs-based applications on heterogeneous multi-core processors," *Microprocessors and Microsystems*, vol. 47, 2016.

[22] J. Theis, G. Fohler, and S. Baruah, "Schedule table generation for time-triggered mixed criticality systems," *Proc. WMC, RTSS*, pp. 79–84, 2013.

[23] J. Theis and G. Fohler, "Mixed criticality scheduling in time-triggered legacy systems," *Proc. WMC, RTSS*, pp. 73–78, 2013.