MATHEMATICS AND COMPUTATIONAL SCIENCES

Vol 3(4), 2022, pp: 33-45
DOI:10.30511/mcs.2022.562401.1084
ISSN:2717-2708
http://mcs.qut.ac.ir/

Qom University of Technology

ILP Approach for Periodic Dependent Tasks in Multicore Mixed-criticality systems

Fatemeh Azad[1], Morteza Mohajjel Kafshdooz [2*]

Abstract: Today, the desire to use mixed-critical systems in the industry is increasing. In order to provide the processing power required by mixed-critical systems, multi-core architectures are considered a suitable option. One of the main challenges in mixed-critical systems is task scheduling, which is even more challenging in multi-core architectures. Many studies of task scheduling in mixed-critical multi-core systems have dealt with the scheduling of independent tasks. But in many real systems, tasks are dependent on each other. In this research, we will deal with the scheduling of dependent periodic tasks in mixed-critical multi-core systems in such a way that the presented schedule satisfies the system constraints. The proposed algorithm provides the best possible schedule using linear programming. The results of the experiments showed that the presented method has been able to significantly reduce the number of preemptions while maintaining the scheduling capability.

Keywords: Embedded, Real-time, Scheduling, Mixed-criticality, Integer linear programming
2020 Mathematics Subject Classification: 15XX, 49XX
Receive: 19 September 2022, **Accepted:** 12 October 2022

1 Introduction

As embedded real-time systems become more complex and the number of their tasks increases, the need for these systems for processing resources has increased. Many of these systems are made up of several tasks with different criticality. Usually in these systems to ensure the correct execution of more critical tasks, tasks with different criticality levels are executed on separate hardware. Today, for various reasons, such as reducing the cost of the hardware, the desire to integrate these systems has increased and hence the concept of mixed-critical systems has been created [25]. In mixed-critical systems, different parts of the system with different criticalities are integrated with a common hardware platform. The criticality of a task or system part is the degree of guarantee against failure that should be given to it. Tasks with a higher level of criticality need more guarantees for their correct execution [9].

In mixed-critical systems, two criticality levels are usually considered. In systems with two criticality levels,

[1] Department of Computer Engineering, Qom University of Technology, Qom, Iran, Email: azad.f@qut.ac.ir
[2*]Corresponding author: Department of Computer Engineering, Qom University of Technology, Qom, Iran, Email: mohajjel@qut.ac.ir

the worst-case execution time (WCET) for high-critical tasks is calculated by both the certification authority and the system designer, while for low-critical tasks WCET is calculated only by the system designer. The certification authority considers a conservative condition to evaluate WCET, that's why the value of WCET evaluated by the certification authority is higher than the execution time calculated by the system designer. In these systems, when a task cannot complete its execution within a low-criticality budget (i.e. WCET calculated by the system designer), the system changes its criticality mode to a high-critical mode. In this mode, high-critical tasks are executed with their high-criticality budget (i.e. WCET evaluated by the certification authority). In fact, in this mode, the system changes from an optimistic mode to a pessimistic and conservative one.

Task scheduling in these systems faces two important challenges: 1) ensuring correctness and 2) efficiency [1]. Correctness in these systems means that tasks have a sufficient guarantee to be executed within their given deadline according to their criticality level (the higher the criticality level of the task, the greater the task's need to meet its deadline). Efficiency in these systems means that the existing hardware resources are used efficiently in order to establish the correctness of the system.

Due to the dependency between tasks, it is desirable the scheduling algorithm considers the cost of communication between dependent tasks when mapping tasks to cores. Because, when two dependent tasks are executed on different cores, the communication cost is usually higher than when they are executed on the same core. Less attention has been paid to this cost in the previous related work. In this research, we have also paid attention to the communication cost between tasks.

In summary, our contributions is ***Presenting an optimal scheduling algorithm for periodic dependent tasks in multi-core mixed-critical systems which is aware of preemption and communication costs.***
The rest of this paper is organized as follows: In section 2, related works are reviewed. In section 3, assumptions and problem statements are explained. In section 4, the proposed methods are presented. In section 5, the proposed scheduling method is examined on the software of a drone. Finally, in section 6, the paper is concluded by stating possible future works.

2 Related Work

In this section, we review studies related to dependent task scheduling in mix-critical systems. It should be noted that in these studies to represent the dependent task, a directed acyclic graph (DAG) is used in which, nodes and edges represent tasks and their dependencies respectively.

In [2] and [3] the scheduling of dependent tasks in a single core mix-critical system is investigated. First, the nodes (tasks) are sorted in topological order, and nodes that do not have any predecessor or their predecessors have been executed, are added to the list of ready-task. In this topological order, high-critical tasks always have a higher priority than low-critical tasks. The algorithm presented in these articles is optimal, but can only be used for single-processor architectures.

In [4] the scheduling of dependent tasks in mix-critical systems in a multiprocessor architecture is examined and it is assumed that all tasks have a common deadline. The proposed method in this paper, first, by using the list scheduling (LS) algorithm [17] makes a scheduling table for high-critical mode. In this mode scheduling is non-preemptive. Then, for low-critical mode, the scheduling table is made also by using the LS algorithm. However, in low-critical mode, scheduling is preemptive. The proposed method in [4] is not optimal. Also, the

higher priority of high-critical tasks than the low-critical tasks reduces the schedulability of the low-critical tasks in low-critical mode.

In [18], by defining and using a concept called Latest Safe Activation Instant (LSAI), it has been able to give higher priority to low-critical tasks when needed and reduce the limitations of the scheduling in low-critical mode. Indeed, LSAIs determine the moments when high-critical tasks should be given higher priority than low-critical tasks. Therefore, they are considered a virtual deadline when scheduling in the low-critical mode of the system. This paper does not consider the systems which are consisted of multiple DAGs of tasks with different deadlines and periods.

In [5] the scheduling of multiple DAGs with different periods for a multicore mixed-critical system is considered. In this paper, it is assumed that each DAG is a task with its period, and its nodes are the jobs of the task which are dependent on each other. Also, the criticality level is assigned to jobs instead of tasks. A federated approach is used to map tasks to cores. In this method, tasks are divided into light and heavy tasks based on their utilization. For a light task, all its jobs will run on one core, and for heavy tasks on several cores. In this article, as in [4], high-level jobs are always given a higher priority, which reduces the schedulability of the system in the low-critical mode. Also, the federal method is not efficient in terms of resource utilization because the cores are dedicated to DAGs and it will use a large number of cores in order to have successful scheduling. It also eliminates the possibility of parallelizing jobs in light tasks.

In [26] a Semi-Federated method is used. The main purpose of this article is to prevent low utilization of the cores in the federated method. However, like the federated method, it reduces the possibility of scheduling low-critical tasks due to the absolute high priority given to high-critical jobs.

In [23] and [16], also the federated method is used to allocate cores to multiple DAGs with different periods. In their considered model, each DAG is equivalent to a task with a high or low-criticality level, and the nodes in the DAG are the jobs of that task between which there is a dependency. Therefore, a DAG and all of its nodes will have the same criticality level, and hence there is no relationship between tasks with different criticality levels. However, this assumption restricts the usage of this method in real applications.

In [19], a global method is used to schedule dependent tasks for mixed-critical systems with multicore architecture. In global methods, there is no restriction on task migration between cores. In the considered model of this paper, each mixed-critical system is represented by several DAGs, each of which has a different period. Furthermore, in this model, the high-critical task cannot be dependent on low-critical tasks. In this paper, schedule tables are made statically for high and low-critical modes. First, the schedule table of the high-critical mode is made by the G-LLF method and in ALAP format. It then made the schedule table of the low-critical mode using the G-LLF method and in ASAP format. A safe transfer condition is also defined to ensure safe transfer from low-critical to high-critical mode and to ensure that the tasks meet their deadlines. The method presented in this paper uses the minimum number of cores and the possibility of scheduling is higher than the previous methods. However, in this method, the preemption and communication cost between related tasks are not considered and there is no control over the number of preemptions although many of the preemptions are unnecessary.

In [20] the method presented in [19] is generalized for a mixed-critical system with an arbitrary number of criticality levels. The proposed method in this paper is based on three methods G-LLF, G-EDF, and G-EDZL. It starts from the highest criticality mode and makes the scheduling table in that criticality mode with one of the

above methods and the form of ALAP. This also is done for lower criticality mode in decent order except for the lowest criticality mode. For the lowest criticality mode, it makes the schedule table according to one of the same three methods and in the form of ASAP. In this article, unlike other research in this field, the dependence of high-critical tasks on low-critical tasks is allowed, but when scheduling, it is assumed that such tasks can be executed without complete execution of their prerequisite tasks, and hence this method violates the dependency among tasks. The method presented in this paper also produces a lot of preemptions due to the fact that it uses a global scheduling approach.

3 Assumptions and problem statement

The mixed-criticality system S that we assume in this study, consists of two criticality levels high and low. The system starts in low-critical mode and in this mode all tasks use their low-criticality budget (Clow). Whenever a task cannot finish its execution within the allocated budget, a time failure event (TFE) occurs and the system switches to the high-critical mode. In this mode, tasks use their high-critical budget (Chigh). We assume the system switches back to the low-critical mode at the first idle time and if no idle time occurs before the end of the current hyper-period (least common multiplier of tasks periods), the system automatically switches back to the low-critical mode at the end of this hyper-period. In line with some previous work, we also assume in the high-critical mode all low-critical tasks are dropped. It should be noted that although this is an aggressive method, however as switching to the high-critical mode is usually very rare, it is acceptable for many mixed-critical systems to simply drop low-critical tasks in high-critical mode.

The considered system S is defined as $S = (\Pi, \ \Gamma)$. Π denotes the system architecture and contains m homogeneous cores and Γ denotes the system software and contains n mixed-criticality DAGs (MC-DAG) represented by G_i ($i \in [1, n]$). Each MC-DAG G_i has the following specifications:

V_i denotes the set of vertexes in G_i . Each vertex v_j in this set represents a task τ_j. We assume all tasks are released at the same time at $t = 0$.

E_i denotes the set of edges in G_i and it is a subset of $V_i \times V_i$. Each edge (τ_a, τ_b) in this set shows the dependency of τ_b on τ_a.

$pred(\tau_j)$ and $succ(\tau_j)$ are the set of predecessors and successors of the task τ_j respectively. Each task can be executed only when the execution of all of its predecessors is finished. It should be noted that high-critical tasks cannot depend on low-critical tasks. This is because low-critical tasks are dropped when the system switches to high-critical mode.

T_i is a natural number and denotes the period of G_i. We assume the period of all tasks in G_i are same and are equal to T_i. However, the period of each MC-DAG can differ from other MC-DAGs, and hence the system is multi-periodic. A multi-periodic system is schedulable when it can be scheduled for one hyper-period [10].

Each mixed-critical task $\tau_j \in V_i$ has the following specifications:

L_j is the criticality level of τ_j and as we assume two criticality levels, it is equal to Low or $High$.

$C_j(Low)$ is a non-negative number and denotes $WCET$ or the time budget of τ_j in the low-criticality mode of the system.

$C_j(High)$ is a non-negative number and denotes $WCET$ or the time budget of τ_j in the high-criticality mode of the system. For high-critical tasks, this number is greater than or equal to $C_j(Low)$ and for low-critical tasks $C_j(High) = 0$. This is because low-critical tasks are dropped into high-critical mode.

T_j denotes the period of τ_j and as we mentioned previously it is equal to the period of MC-DAG that contains τ_j ($T_j = T_i$).
The kth job of τ_j is represented by $J_{j,k}$ and it inherits the all specification of τ_j. The release time of $J_{j,k}$ is obtained from $r_{j,k}$ = *(k-1)* $\times$ T_j. We assume an implicit-deadline for each task and hence its job's absolute deadline is obtained from $d_{j,k} = k \times T_j$. Obviously for valid scheduling, the execution of $J_{j,k}$ must be between $r_{j,k}$ and $d_{j,k}$.
The utilization of G_i in criticality mode, θ is defined as $U_\theta(G_i) = \sum_{\tau_j \in G_i} C_j(\theta)/T_i$ and the total utilization of the system in criticality mode of θ is obtained from $U_\theta(S) = \sum_{G_i \in \Gamma} U_\theta(G_i)$. Obviously, $U_\theta(S)$ for each criticality mode, θ must be less than or equal to the number of processing cores.

3.1 Schedulability test

To have correct scheduling for the considered system S, we must ensure the correct scheduling in low and high-criticality mode and also in the transitions from the low to the high mode [3],[5]:

- **Correct scheduling in low-critical mode**: as we mentioned previously, in low-critical mode the execution time of each task τ_j is at most $C_i(Low)$ and to have correct scheduling in this mode, the allocated time for each job task τ_j between its release time and deadline must be greater than or equal to $C_j(Low)$.
- **Correct scheduling in high-critical mode**: to have correct scheduling in this mode, the allocated time for each job of a high-critical task τ_j between its release time and deadline must be greater than or equal to $C_j(High)$.
- **Safe transition from high to low-critical mode**: This condition is also related to correct scheduling in high-critical mode. In particular, this condition ensures that at each point in time, the transition from low to high-critical mode is safe and does not cause the non-schedulability of the system in high-critical mode. According to this condition, whenever the system switches to the high-critical mode, there must be enough time to execute high-critical tasks. Indeed, this is needed to be checked because high-critical tasks' execution time may increase in high-critical mode and the jobs of high-critical tasks that have been started before the transition time but do not yet complete their execution must have enough time to complete their increased remained execution time. According to [5] to have a safe transition, for each point in time t and each kth job of task j eq (3.1) must be satisfied:

$$\forall \tau_j \, , et_{\tau_j}^{Low}(r_{j,k}, t) < C_j(Low) : et_{\tau_j}^{Low}(r_{j,k}, t) \geq et_{\tau_j}^{High}(r_{j,k}, t) \tag{3.2}$$

In this equation, t denotes the time at which a time failure event (TFE) occurs. $et_{j,k}^{\theta}(t_1, t_2)$ denotes the allocated time for the job $J_{j,k}$ from time t_1 to time t_2 in θ criticality mode. Because the occurrence time of TFE is not known priory this condition must be checked for time $t \in [r_{j,k}, d_{j,k}]$. According to eq (1) for the high-critical tasks that their execution does not finish before time t in low-critical mode, the allocated execution time to them before time t in low-critical mode must be greater than or equal to the allocated time to them before time t in high-critical mode.

3.2 Preemption and Communication cost:

In system S, tasks can be preempted and continue their execution in another core or at another time. These preemptions can impose considerable overhead on the system. To consider this overhead, for each task τ_j, the

cost for preemption in criticality mode θ is estimated by $preemption_{cost}^{\theta}(\tau_j) = \lfloor PF \times C_j^{\theta} \rfloor$ [27]. In this equation, PF is a constant number in the range [0, 0.5).

In system S, if there is an edge (τ_a, τ_b) in the system, then task τ_b depends on the output data of the task τ_a, and hence the task τ_b is not allowed to start until the task τ_a is completed. Depending on the cores to which these tasks are mapped, two modes may occur [17]. If these two tasks (the beginning of the task τ_b and the end of the task τ_a) are scheduled on the same core, and data transfer between them takes place in zero-time units ($communication_{cost}\ (\tau_a, \tau_b) = 0$). If these two tasks are scheduled on different cores, the task τ_b must wait for a while before execution until the data transfer from task τ_a finishes. This data transfer time in criticality mode θ is equal to $communication_{cost}\ (\tau_a, \tau_b) = \lfloor CF \times C_b^{\theta} \rfloor$. In this equation, CF is a constant number in the whole system and the range [0,0.5). Now if the task τ_b depends on a set of tasks, it must wait for $\max_{\tau_p \in pred(\tau_b)} communication_{cost}(\tau_p, \tau_b)$ before starting its execution.

4 Proposed methods

In this section, we represent an optimal method based on integer linear programming (ILP) to schedule tasks in system S aimed at reducing the number of preemptions. In ILP, the problem is modeled as some linear statements and then it is solved by an ILP solver such as CPLEX. In the following text, we first introduce the used notation, and then we present the proposed ILP model.

Parameters:

As mentioned previously to ensure the schedulability of a periodic system it is sufficient to check its scheduling for one hyper-period. Therefore, in the following, we only consider the scheduling of the system in the first hyper-period $[0, HP(S)]$.

for each task τ_j of the system there is $\frac{HP(S)}{T_{\tau_j}}$ jobs that are included in the model. For each job i we have the following parameters:

- c_i^{θ}: a non-negative value that denotes the WCET of job i in criticality mode $\theta \in \{Low, High\}$
- r_i : a non-negative value that denotes the release time of job i.
- d_i : a non-negative value that denotes the relative deadline of job i.
- $E_{i,i'}$: a binary value that denotes the existence of an edge between job i and job i'

In addition to the mentioned parameters, the following parameters are also required in the model:

- HP: hyper period
- M: the total number of cores
- N: the total number of jobs
- CF: communication factor
- PF: preemption factor

Also, we use index i for jobs $(1 \leq i \leq N)$, index t for times $(0 \leq t \leq HP)$, and index k for processing cores $(1 \leq k \leq M)$

Decision variables:

To represents the objective function and the constraints of the problem, the following variable is used:

- $x_{i,k,t}^{\theta}$: a binary variable to indicate whether or not job i is mapped to core k at time instance t in criticality mode θ.

- $et_{i,t}^{\theta}$: a non-negative integer variable to indicate the amount of time that job j has been executed before time t in criticality mode θ.
- $s_{i,k,t}^{\theta}$: a binary variable to indicate whether or not job i starts its execution in core k at time instance t in criticality mode θ.
- $f_{i,k,t}^{\theta}$: a binary variable to indicate whether or not job i finishes its execution in core k at time instance t in criticality mode θ.
- $p_{i,k,t}^{\theta}$: a binary variable to indicate if job j has been preempted. This variable is 1 if job j has been preempted at criticality mode θ before time t and is resumed in core k at time t.
- $u_{i,i',k}^{\theta}$: a binary variable which indicates at criticality mode θ whether or not job i has finished its execution in core k and job i' has started its execution at the same core.
- $c_{i,i'}^{\theta}$ a binary variable that indicates whether or not to impose the communication cost from task i to task i'in criticality mode θ.

Constraints:
To obtain a correct solution some constraints must be satisfied which we introduce in the following text.

- For each job at each time in criticality mode θ there must be at most one allocated core:

$$\forall \boldsymbol{i}, \forall \boldsymbol{t}: \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} x_{i,k,t}^{\theta} \leq \mathbf{1} \tag{4.1}$$

- Each core at each time in criticality mode θ must be allocated to at most one job:

$$\forall \boldsymbol{k}, \forall \boldsymbol{t}: \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} x_{i,k,t}^{\theta} \leq \mathbf{1} \tag{4.2}$$

- To ensure each task completes its execution in criticality mode θ, its total allocated execution time must be C_i^{θ}:

$$\forall \boldsymbol{i}: \sum_{\boldsymbol{t=0}}^{\boldsymbol{HP}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} x_{i,k,t}^{\theta} = C_i^{\theta} \tag{4.3}$$

- Each job's allocated time in criticality mode θ must be from its release time to its deadline:

$$\forall \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t} (\boldsymbol{t} < r_i \ \boldsymbol{and}\ \boldsymbol{t} \geq d_i): x_{i,k,t}^{\theta} = \mathbf{0} \tag{4.4}$$

- For each job in criticality mode θ that its execution time is greater than 0, there must be exactly one starting point and one end point:

$$\forall \boldsymbol{i} (C_i^{\theta} > \mathbf{0}): \sum_{\boldsymbol{t=0}}^{\boldsymbol{HP}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} s_{i,k,t}^{\theta} = \mathbf{1} \tag{4.5}$$

$$\forall \boldsymbol{i} \left(C_i^{\theta} > \mathbf{0}\right): \sum_{\boldsymbol{t=0}}^{\boldsymbol{HP}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} f_{i,k,t}^{\theta} = \mathbf{1} \tag{4.6}$$

- The elapsed execution time of each job until time t is equal to the sum of times that it has been allocated to different cores:

$$\forall \boldsymbol{i}, \forall \boldsymbol{t}: \sum_{\boldsymbol{t'=0}}^{\boldsymbol{t-1}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} x_{i,k,t'}^{\theta} = et_{i,t}^{\theta} \tag{4.7}$$

- In criticality mode θ, if job i starts its execution at time t then its elapsed time up to time t must be 0. This can be linearized as follows:

$$\forall \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}: et_{i,t}^{\theta} \leq \left(\mathbf{1} - s_{i,k,t}^{\theta}\right) \times C_i^{\theta} \tag{4.8}$$

- In each criticality mode θ, to ensure consistency of $s^{\theta}_{i,k,t}$ and $x^{\theta}_{i,k,t}$ at the start time of each job i, if it starts its execution at time t on core k ($s^{\theta}_{i,k,t} = 1$) then it must be allocated to the same core k at time t $\left(x^{\theta}_{i,k,t} = 1\right)$. This can be linearized as follows:

$$\forall\, \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ s^{\theta}_{i,k,t} \le x^{\theta}_{i,k,t} \tag{4.9}$$

- In each criticality mode θ, if job i finishes its execution at time t then its elapsed time up to time t must be equal to its total execution time. This can be linearized as follows:

$$\forall\, \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ \left(C^{\theta}_{i} - et^{\theta}_{i,t}\right) \le \left(\mathbf{1} - f^{\theta}_{i,k,t}\right) \times C^{\theta}_{i} \tag{4.10}$$

- In each criticality mode θ, to ensure consistency of $f^{\theta}_{i,k,t}$ and $x^{\theta}_{i,k,t}$, if job i finishes its execution at time t on core k ($f^{\theta}_{i,k,t} = 1$) then it must be allocated to the core k at time $t-1$ $\left(x^{\theta}_{i,k,t-1} = 1\right)$. This can be linearized as follows:

$$\forall\, \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ f^{\theta}_{i,k,t} \le x^{\theta}_{i,k,t-1} \tag{4.11}$$

- In each criticality mode θ, to indicate time instances that job i is preempted, we use a binary variable $p^{\theta}_{i,k,t}$. This variable is one whenever at time t job i is allocated to core k () and at $t-1$ it is allocated to other core and also it is not started at time t $\left(p^{\theta}_{i,k,t} = \left(\mathbf{1} - x^{\theta}_{i,k,t-1}\right) \times x^{\theta}_{i,k,t} \times \left(\mathbf{1} - s^{\theta}_{i,k,t}\right)\right)$. However, this equation is not linear. Its linearized form is as follows:

$$\begin{gathered}\forall\, \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ p^{\theta}_{i,k,t} \ge \left(-x^{\theta}_{i,k,t-1}\right) + x^{\theta}_{i,k,t} + \left(-s^{\theta}_{i,k,t}\right) \\ \forall \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ p^{\theta}_{i,k,t} \le \left(\mathbf{1} - x^{\theta}_{i,k,t-1}\right) \\ \forall \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ p^{\theta}_{i,k,t} \le x^{\theta}_{i,k,t} \\ \forall \boldsymbol{i}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ p^{\theta}_{i,k,t} \le \left(\mathbf{1} - s^{\theta}_{i,k,t}\right)\end{gathered} \tag{4.12}$$

- In each criticality mode θ, if job i resumes its execution on core k at time t (i.e., $p^{\theta}_{i,k,t} = 1$) then it must wait $\lfloor PF \times C^{\theta}_{i} \rfloor$ time units before time t to complete its context switching phase. In this phase, no other job must be in execution mode on the same core and also, this job must not be allocated to other cores. This can be formulated as follows:

$$\forall \boldsymbol{i}, \forall \boldsymbol{i'}, \forall \boldsymbol{k}, \forall \boldsymbol{k'}, \forall \boldsymbol{t}, \forall \boldsymbol{t'}\left(\boldsymbol{t} - \lfloor PF \times C^{\theta}_{i} \rfloor \le \boldsymbol{t'} < \boldsymbol{t}\right):\ x^{\theta}_{i',k,t'} + x^{\theta}_{i,k',t'} \le \mathbf{2}\left(\mathbf{1} - p^{\theta}_{i,k,t}\right) \tag{4.13}$$

- In each criticality mode θ, if the job i' depends on job i and job i finishes its execution at time t then the elapsed time of the job i' up to time, t must be 0:

$$\forall \boldsymbol{i}, \forall \boldsymbol{i'}, \forall \boldsymbol{k}, \forall \boldsymbol{t}:\ E_{i,i'} \times et^{\theta}_{i',t} \le \left(\mathbf{1} - f^{\theta}_{i,k,t}\right) \times C^{\theta}_{i'} \tag{4.14}$$

- In each criticality mode θ, if job i finishes its execution on core k and job i' starts its execution on the same core k then $u^{\theta}_{i,i',k} = 1$. This can be formalized as $u^{\theta}_{i,i',k} = \sum_{t=0}^{HP} f^{\theta}_{i,k,t} \times \sum_{t=0}^{HP} s^{\theta}_{i',k,t}$. The linearized form is as the three following inequalities:

$$\forall\, \boldsymbol{i}\,, \forall \boldsymbol{i'}, \forall \boldsymbol{k}:\ u^{\theta}_{i,i',k} \geq \sum_{t=0}^{HP} f^{\theta}_{i,k,t} + \sum_{t=0}^{HP} s^{\theta}_{i',k,t} - 1 \tag{4.15}$$

$$\forall \boldsymbol{i}\,, \forall \boldsymbol{i'}, \forall \boldsymbol{k}:\ u^{\theta}_{i,i',k} \leq \sum_{t=0}^{HP} f^{\theta}_{i,k,t}$$

$$\forall \boldsymbol{i}, \forall \boldsymbol{i'}, \forall \boldsymbol{k}:\ u^{\theta}_{i,i',k} \leq \sum_{t=0}^{HP} s^{\theta}_{i',k,t}$$

- In each criticality mode θ, if the job i' depends on job i ($E_{i,i'} = 1$) and job i' starts its execution on a different core than the core that job i finishes its execution, then the communication cost from job i to job i' must be imposed:

$$\forall \boldsymbol{i}\,, \forall \boldsymbol{i'}:\ c^{\theta}_{i,i'} = \left(\mathbf{1} - \sum_{\boldsymbol{k=1}}^{\boldsymbol{M}} u^{\theta}_{i,i',k}\right) \times E_{i,i'} \tag{4.16}$$

- In each criticality mode θ, if at least one of the prerequisite jobs of job i, imposes a communication cost to it, then before it can start its execution on core k at time t, it must wait $\lfloor CF \times C^{\theta}_{i} \rfloor$ time units before time t to complete its communication phase. In this phase, no other job must be in execution mode on the same core and also, this job must not be allocated to other cores. This can be formulated as follows:

$$\forall \boldsymbol{i}\,, \forall \boldsymbol{i'}, \forall \boldsymbol{i''}, \forall \boldsymbol{k}, \forall \boldsymbol{k'}, \forall \boldsymbol{t}, \forall \boldsymbol{t'}\left(\boldsymbol{t} - \lfloor CF \times C^{\theta}_{i} \rfloor \leq \boldsymbol{t'} < \boldsymbol{t}\right):\ x^{\theta}_{i',k,t'} + (x^{\theta}_{i',k',t'} \times E_{i',i}) \leq \tag{4.17}$$
$$\mathbf{2}\left(\mathbf{1} - s^{\theta}_{i,k,t}\right) + \mathbf{2}\left(\mathbf{1} - c^{\theta}_{i'',i}\right)$$

- To ensure the safe transition, as we mentioned previously for each job i that has not finished its execution before time t in low-criticality mode ($\sum_{t'=0}^{t}\sum_{k=1}^{M} f^{Low}_{i,k,t'} = 0$), its elapsed execution time until time t in high-criticality mode must be less than or equal to its elapsed time in low-criticality mode this can be represented as follows:

$$\forall \boldsymbol{i}\ \,, \forall \boldsymbol{k}\,, \forall \boldsymbol{t}:\ et^{High}_{i,t} \leq et^{Low}_{i,t} + \left(C^{High}_{i} \times \sum_{t'=0}^{t}\sum_{k=1}^{M} f^{Low}_{i,k,t'}\right) \tag{4.18}$$

Objective: As we mentioned previously, the objective is to reduce the number of preemptions. Therefore, the objective is defined as minimizing the total number of preemptions in high and low-criticality modes:

$$\boldsymbol{Min} \sum_{\boldsymbol{i,k,t}} (p^{Low}_{i,k,t} + p^{High}_{i,k,t}) \tag{4.19}$$

5 Results

In this section, with an example, we will show the effectiveness of the proposed methods. The intended system as a motivational example is an Unmanned Arial Vehicle (UAV). The graphical representation of the software of this system is shown in Figure (1) [19]. The system consists of two parts. The first part (top of the figure) represents the Flight Control System (FCS) [24], the tasks of this part are displayed in a DAG and its period is $T_{FCS} = 12$. The second part (the bottom of the figure) is the image processing system called Montage [7], the tasks of this part are displayed in a DAG and its period is $T_{Montage} = 24$.

Each of these DAGs consists of several nodes that represent tasks. White nodes indicate low-critical tasks and gray nodes indicate high-critical tasks. Inside each node, the budget of the corresponding task in low and high-critical mode is shown (The first number from the left is equal to C(Low) and the other is equal to C (High)). Each DAG, in addition to nodes, contains several edges that indicate the data dependency between tasks.

The scheduling of the system should be checked within a Hyper-Period. The hyper-Period of this system is 24. Therefore, we have two activations of the tasks in FCS and one activation of the tasks in Montage.

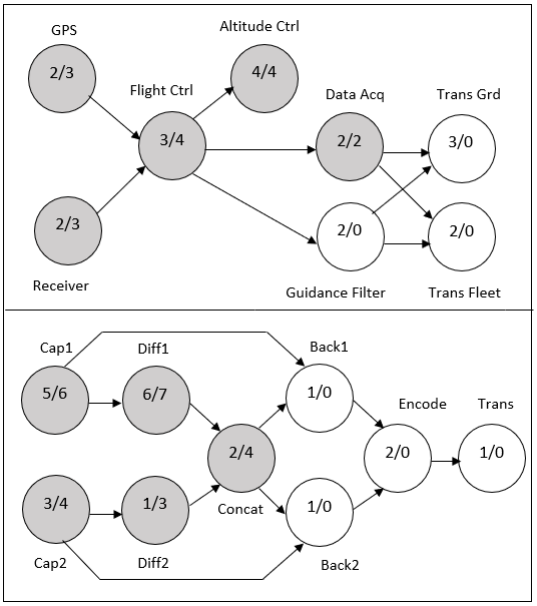

Figure 1: UAV system

First, the tasks of this system are scheduled by the limited preemption method, assuming a 3-core architecture and PF = CF = 0.4. The result of this scheduling is shown in Figure (2) in the low-critical mode and Figure (3) in the high-critical mode. In these figures, the parts marked with cc and pc that exists before the execution of some tasks indicate the cost of communication and preemption imposed on that tasks respectively. As the results show, the number of preemptions in high and low-critical modes is 1.

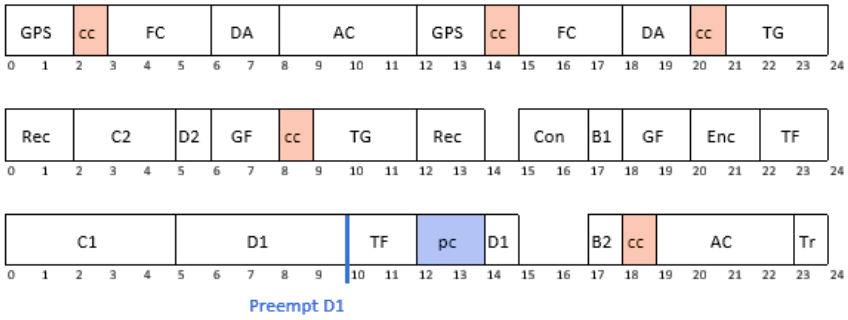

Figure 2: UAV's limited preemptive scheduling in low-critical mode

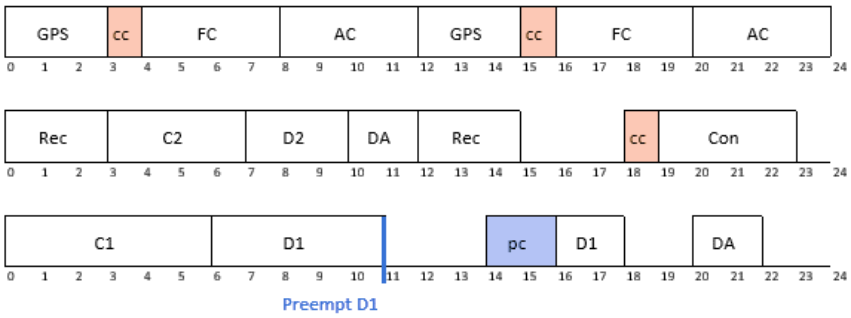

Figure 3: UAV's limited preemptive scheduling in high-critical mode

The result of optimal scheduling of this system in low-critical mode is shown in Figure (4) and the result of scheduling in high-critical mode is shown in Figure (5). As it is shown, in low-critical mode, the optimal method has one preemption which is equal to the result of the limited-preemption scheduling, but in the case of high system criticality, the result of the optimal method has no preemption.

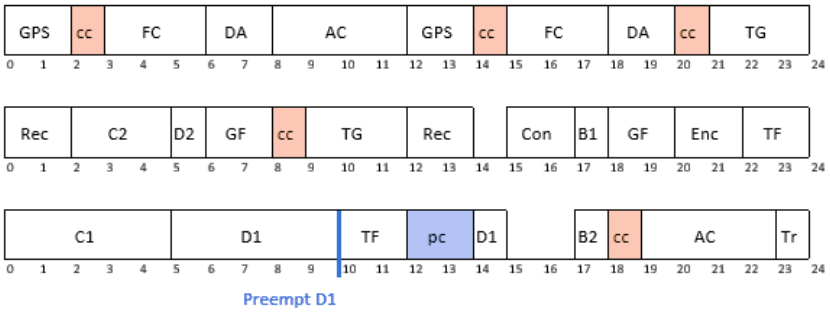

Figure 4: UAV's optimal scheduling in low-critical mode

Figure 5: UAV's optimal scheduling in high-critical mode

6 Conclusion

In this research, the scheduling of periodic dependent tasks in multi-core mixed-critical systems was investigated. In this article, unlike previous works, the cost of preemption and communication between tasks

were considered. The results of experiments showed that the presented method was able to significantly reduce the number of preemptions while maintaining the scheduling capability. It is worth mentioning that the presented method is offline and does not impose overhead on the system during execution. In addition, in this research, an optimal ILP method for the mentioned problem were presented. On the other hand, execution time is not economical and it is not able to solve the problem in a limited time in medium and large systems. In order to complete this article, future research can address issues such as increasing the QoS of low-critical tasks in the high-critical mode of the system or applying a transfer cost from the low-critical mode to the high-critical mode of the system.

Reference

[1] S. Baruah, Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations, IEEE Des. Test, 35(2) 2018, 31-37.
[2] S. Baruah, Implementing mixed-criticality synchronous reactive programs upon uniprocessor platforms, Real-Time Systems, 50(3) 2014, 317-341.
[3] S. Baruah, Semantics-preserving implementation of multirate mixed-criticality synchronous programs, in Proceedings of the 20th International Conference on Real-Time and Network Systems, 2012, 11-19.
[4] S. Baruah, Implementing mixed criticality synchronous reactive systems upon multiprocessor platforms, The University of North Carolina at Chapel Hill, Tech. Rep, 2013.
[5] S. Baruah, The federated scheduling of systems of mixed-criticality sporadic DAG tasks, in 2016 IEEE Real-Time Systems Symposium (RTSS), 2016: IEEE, 227-236.
[6] S. Baruah et al., Scheduling real-time mixed-criticality jobs, IEEE Transactions on Computers, 61(8) 2011, 1140-1152.
[7] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.H. Su, K. Vahi, Characterization of scientific workflows, in 2008 third workshop on workflows in support of large-scale science, 2008: IEEE, pp. 1-10.
[8] E. Bini, G.C. Buttazzo, Measuring the performance of schedulability tests, Real-Time Systems, 30(1-2) 2005, 129-154.
[9] A. Burns and R. Davis, Mixed criticality systems-a review, Department of Computer Science, University of York, Tech. Rep, 2013, 1-69.
[10] G.C. Buttazzo, Hard real-time computing systems: predictable scheduling algorithms and applications. Springer Science & Business Media, 2011.
[11] F. Cadoret, T. Robert, E. Borde, L. Pautet, F. Singhoff, Deterministic implementation of periodic-delayed communications and experimentation in aadl, in 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013), 2013: IEEE, 1-8.
[12] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.M. Vincent, F. Wagner, Random graph generation for scheduling simulations, in 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), 2010, ICST,10.
[13] R. I. Davis, A. Burns, Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, in 2009 30th IEEE Real-Time Systems Symposium, 2009: IEEE, 398-409.
[14] J. Lee et al., MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors, in 2014 IEEE Real-Time Systems Symposium, 2014: IEEE, 41-52.
[15] H. Li, S. Baruah, Outstanding paper award: Global mixed-criticality scheduling on multiprocessors, 24th Euromicro Conference on Real-Time Systems, 2012: IEEE, 166-175.
[16] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, C. Lu, Mixed-criticality federated scheduling for parallel real-time tasks, Real-time systems, 53(5) 2017, 760-811.
[17] P. Marwedel, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things. Springer, 2017.
[18] R. Medina, E. Borde, L. Pautet, Directed acyclic graph scheduling for mixed-criticality systems, in Ada-Europe International Conference on Reliable Software Technologies, 2017: Springer, 217-232.
[19] R. Medina, E. Borde, and L. Pautet, Scheduling multi-periodic mixed-criticality dags on multi-core architectures, in 2018 IEEE Real-Time Systems Symposium (RTSS), 2018: IEEE, 254-264.

[20] R. Medina, E. Borde, L. Pautet, Generalized Mixed-Criticality Static Scheduling for Periodic Directed Acyclic Graphs on Multi-Core Processors, IEEE Transactions on Computers, 2020.
[21] M. Mitchell, An introduction to genetic algorithms. MIT Press, 1998.
[22] R. M. Pathan, Schedulability analysis of mixed-criticality systems on multiprocessors, in 2012 24th Euromicro Conference on Real-Time Systems, 2012: IEEE, 309-320.
[23] R. M. Pathan, Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors, in 30th Euromicro Conference on Real-Time Systems (ECRTS 2018), 2018: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[24] S. Siebert, J. Teizer, Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system, Automation in construction, 41 2014, 1-14.
[25] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in 28th IEEE international real-time systems symposium (RTSS 2007), 2007: IEEE, 239-243.
[26] T. Yang, Y. Tang, X. Jiang, Q. Deng, and N. Guan, Semi-Federated Scheduling of Mixed-Criticality System for Sporadic DAG Tasks, in 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), 2019: IEEE, 163-170.
[27] H.E. Zahaf, G. Lipari, S. Niar, Preemption-Aware Allocation, Deadline Assignment for Conditional DAGs on Partitioned EDF, in 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2020: IEEE, 1-10.