IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 3, 2020, accepted September 1, 2020, date of publication September 18, 2020,
date of current version September 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024698

Integer Programming Techniques for Static Scheduling of Hard Real-Time Systems

ANA GUASQUE[1], HOSSEIN TOHIDI[2], PATRICIA BALBASTRE[1], JOSÉ MARÍA ACEITUNO[1], JOSÉ SIMÓ[1], AND ALFONS CRESPO[1]

[1]Instituto de Automática e Informática Industrial, Universitat Politècnica de València, 46022 Valencia, Spain
[2]Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC 27607, USA

Corresponding author: Ana Guasque (anguaor@ai2.upv.es)

This work was supported by the Spanish Science and Innovation Ministry (predictable and dependable computer systems for Industry 4.0) under Grant MICINN: CICYT project PRECON-I4 and Grant TIN2017-86520-C3-1-R.

ABSTRACT Hard real-time systems focus on obtaining a feasible schedule while satisfying different temporal requirements. In safety-critical applications, this schedule is generated offline. This article explores different integer linear programming techniques (ILP) to schedule uniprocessor hard real-time systems. The goal is to efficiently obtain a static schedule for periodic tasks and partitioned systems where temporal and spatial isolation is crucial. The advantage of the proposed ILP techniques is the possibility of choosing the optimization criteria so that deadlines are met and better performance quality is achieved. The drawback is the time spent finding an optimal solution. We propose an ILP method that reduces by 70% the time needed to obtain an optimal solution compared to basic approaches. This method is called the rolling task MILP approach and the optimization problem is addressed task by task. Experimental results show that our approach also achieves better results than heuristics when trying to reduce temporal parameters such as response times, context switches, and jitter. This makes our solution suitable for control systems and other applications.

INDEX TERMS Integer linear programming, optimization, partitioned systems, real-time systems, static scheduling.

I. INTRODUCTION

Modern real-time embedded systems comprise many applications, often of different criticalities, executing on the same computing platform. If a hard real-time task misses any temporal constraint in high criticality applications, it may suppose catastrophic results. As defined in [1], criticality is a designation of the level of assurance against failure needed for a system component. It is necessary to certify the system's safety and security to avoid any catastrophic or hazardous failure. Independent organizations certify the system if the collected evidence proves that it is behaving as expected. Conformance to a safety standard is of great help, or even required, for certification. There are several such standards for different domains, such as electronic systems (IEC 61508), airborne civil avionics (DO-178B), nuclear power plants (IEC 880), medical systems (IEC 601-4), European railways (EN 50128]), European space (ECSS), etc. Temporal and spatial partitioning (TSP) is often a requirement to isolate faults and avoid recertifying the whole system when the requirements change. Partitioned systems are a way to ensure TSP in which applications with different criticalities are grouped into partitions that run in isolation from each other. This isolation prevents a failure in a low criticality application to propagate in more critical applications.

Both static and dynamic allocations might be used to obtain a feasible schedule of a hard real-time task set. Static allocation of resources is a major requirement for fulfilling certification requirements, for example, in standards such as ARINC-653 [2]. Static scheduling requires a priori knowledge of the characteristics of the tasks. The static scheduler generates an offline plan, a sequence of task executions, for a group of available tasks. This scheme is known as cyclic executive [3], [4]. This static plan is saved in a table and indicates when each task should be executed. The verification

The associate editor coordinating the review of this manuscript and approving it for publication was Kai Li.

of the schedulability using this strategy must be carried out during the construction of the plan. Due to this determinism in the execution, static scheduling is widely used in real-time high integrity systems. Static scheduling presents advantages such as low cost at run time, and disadvantages, such as a lack of flexibility, and good knowledge of the task set is required.

Preemptive and non-preemptive periodic tasks scheduling is NP-hard in the strong sense [5]. Commonly used heuristics are mainly focused on obtaining a feasible schedule. However, optimizing certain temporal parameters, such as the best case response time, might be of interest, especially when timing constraints impose lower bounds on response times to events. For example, upon a collision, an airbag has to be inflated neither too early nor too late [6]. Jitter is the other critical parameter that measures the variation in response time. Especially in control systems, any delays can cause the system instability since these delays are not taken into account in the control law [7].

Exact optimization methods, such as integer linear programming (ILP) have been mostly used for scheduling tasks on multiprocessor systems rather than uniprocessor systems, as there are heuristics that can optimally schedule tasks on an uniprocessor system in polynomial time. By formulating the problem as an ILP, we can theoretically determine the optimal schedule in any system considering different objectives and constraints. The possibility of customizing an objective function that satisfies the needs of the designers is an interesting proposition that can be applied to different fields with different goals. However, solution times might not always remain tractable. Recent progress in optimization solvers diminishes this issue as they can effectively solve practical size instances.

In this work, we explore ILP for generating static schedules on uniprocessor applications for periodic and partitioned systems. Our goal is to find optimal static schedules with a customized objective rather than focusing on obtaining a feasible schedule. We formulate the problem for the periodic task model, both preemptive and non-preemptive, and propose an alternative ILP formulation that significantly improves the simple model performance. This new approach can be used to schedule a partitioned system to obtain an optimal solution in a reasonable time. The objective function can be changed to optimize any combination of temporal parameters. Response times (worst and best), jitter and context switches can be minimized in order to obtain better a performance in a variety of applications. The context switch is one of the most significant attributes of a multi-task operating system. It occurs when the CPU switches from one task to another, and requires saving the context of the current task so that the CPU can restore and complete it later. If not properly controlled, context switches may lead to reduced responsiveness, unnecessary delays, energy wastage, and extra memory requirements. These can lead to a high overhead in real-time embedded systems. Therefore, this work provides a highly versatile scheduling technique, with customized objectives and feasible results.

The rest of the paper is organized as follows: Section II describes the related work. Section III defines the model used. Section IV formulates the scheduling problem and presents the ILP approaches for both periodic tasks systems and partitioned systems. We evaluate the proposals in Section V.

II. RELATED WORK

Initially, static scheduling was the most used method in real-time systems, with the cyclic executive approach being the most common according to [3]. The idea is based on building a cyclic system that is practically made to measure. Static planning is a widely used planning method in real-time high integrity systems according to [8], since determinism is the most valued characteristic as a safety feature.

In contrast, dynamic scheduling offers greater flexibility. However, static scheduling with cyclic executives continues to exist. Several certification standards, like those listed in the previous section, include a requirement for the use of static allocation of resources. Even in the proposed safety-critical Java specification (SCJ) [9], the most constrained level, which should be especially suited for certification, prescribes the use of a cyclic executive.

As mentioned, there are heuristic algorithms that can find the optimal real-time schedule for simple uniprocessor architectures. However, many researchers are recently converting the classical scheduling problem into an ILP problem in order to optimize more complex models.

The works that obtain feasible schedules with ILP techniques in complex real-time models includes, among others: multiprocessor systems [10]; power consumption optimization [11]; consideration of architectures with local instruction or data caches [12]; weakly hard real-time systems [13]; mixed criticality [14]; distributed systems [15]; etc. This is due to the absence of heuristic optimal algorithms for more complex models than the typical periodic task model.

In [16], the precedence relation between tasks is considered. This work uses relaxed ILP techniques to obtain an optimal priority/deadline assignment for preemptive dynamic priority scheduling under precedence constraints. Also, the response time calculation and priority assignment problem with an ILP is presented in [17], where the ceiling of the response time equation is reformulated as an ILP problem. An improved real-time schedulability test that allows an exact and efficient definition of the feasible region by fewer binary variables is provided in [18]. Our goal is not only to find a feasible schedulable plan, but we are seeking to find the optimal schedule considering different objectives.

Regarding optimal strategies but not based on ILP, in [19] offline scheduling strategies for non-preemptive real-time tasks on uniprocessors are proposed. Using formal approaches such as supervisory control theory (SCT) or time discrete event systems (TDES), authors present an optimal scheduler for aperiodic and sporadic tasks. In [20], similar techniques have been used to reclaim unused task computation times.

III. MODEL DEFINITION

A. PERIODIC TASK MODEL

In a hard real-time system, there is a set of n independent real-time tasks $\tau = [\tau_1, \ldots, \tau_n]$, where each task generates a set of infinite jobs $(\tau_{ij}, j \geq 0)$ that must be completed before the arrival of the due time. This work considers that all tasks are periodic, i.e., composed by a activations every specified time. Each task is characterized as $\tau_i = (C_i, D_i, T_i)$, where T_i is the period, D_i is the deadline and C_i is the worst case execution time. From now on, D_i is relative deadline, in contrast with absolute deadline, which is $d_{ij} = j \cdot T_i + D_i$ for activation j. The utilization of a task τ_i is calculated as the relation between the computation time and the period, $U_i = \frac{C_i}{T_i}$.

For notational convenience and without loss of generality, we assume that the tasks are given in order of increasing deadline. Under the same deadline conditions, an order of increasing period is assumed.

The hyperperiod of the task set H, is the smallest interval of time after which the periodic patterns of all the tasks are repeated, and it is calculated as the least common multiple of the periods of the tasks.

The response time (w_{ij}) of task τ_i in activation j is the time between an activation being released and the end of its execution. The worst case response time [21], [22] is the maximum time interval between arrival and finish instants for each task ($WCRT_i = \max\{w_{ij}\}$). Similar to the worst case response time, the best case response time of a task is defined as the minimum time between any release of a task and its corresponding completion ($BCRT_i = \min\{w_{ij}\}$), where the minimum is taken over all executions of the tasks and all possible phasing of the task respect to each other. If, for each task, $BCRT_i \leq WCRT_i \leq D_i$, the task set will be then schedulable.

Usually, it is desirable to reduce the overhead introduced by the context switch [23]. Multiple works address this topic. For example, [24] develops a new scheduling model, which unifies the concepts of preemptive and non-preemptive scheduling, and proposes algorithms for optimal assignment of priorities and preemption threshold. [23] presents a solution for reducing the number of context switches in multi-task scheduling, with task sets with limited hyperperiods.

B. PARTITIONED MODEL

Partitioned systems were developed to address security and safety problems and provide temporal and spatial isolation. A partition consists of an encapsulated group of applications that provide independent execution on a common platform. The operating system is in charge of supporting the execution of the applications. Partitions are executed independently on the top of the hardware, which could be virtual or otherwise.

In a partitioned system, tasks are organized in a set of p partitions $P_1, .., P_p$.

Therefore, we can extend our periodic task model to consider a partitioned model in the following way:

$$\tau_i = (C_i, D_i, T_i, \rho_i) \tag{1}$$

being ρ_i the identifier of the partition it belongs to.

An illustrative example of a partitioned system is presented in 2 in Section IV-C.

IV. MILP SCHEDULING APPROACHES

This section proposes a mixed-integer linear programming (MILP) formulation to determine the optimal offline schedule of periodic tasks on a uniprocessor hard real-time system. The following approaches can be taken to solve the proposed scheduling problem:

- A basic MILP formulation for periodic task systems that provides the optimal scheduling plan according to a certain optimization criteria based on reducing response times and context switches (see Section IV-A); This problem (feasibility of periodic tasks on uniprocessors) is co-NP-complete [25] and intractable in most cases, but it will serve as a reference for the rest of the approaches. Preemptive and non-preemptive scheduling will be considered.
- A MILP formulation with a rolling task approach. In this approach the MILP problem is decomposed into a set of MILP problems, one for each task. Each task activation is allocated in priority order.
- A MILP formulation for partitioned systems based on the two previous approaches (see Section IV-C).

Section III-A has introduced the general notations used throughout this article, and the rest of notations will be introduced when describing any particular approach.

A. MILP MODEL OF UNIPROCESSOR PERIODIC TASK SYSTEMS

We will call this model the *Simple MILP model*. Table 1 introduces the different indices, parameters, and variables used in the model.

According to the problem statement, the objective function is defined in Equation (2), which is minimizing the total number of context switches and the total response time for all tasks in all system executions. The problem is considered as multiobjective because it tries to reduce context switches and response times. The range of values and the units of context switches and response times are different. There is a need for scaling. Context switches are dimensionless. Thus, we need to normalize both context switches and response time to be of a similar scale. As context switching variable (s) is binary (0-1), we scale response times by dividing by deadlines for each activation and task, to have a fair trade-off between competing objectives. Later, in section IV-D, we define the weights using a goal programming approach presented in [26] in order to optimize each objective by minimizing its deviation from their target value.

Therefore, Equation (2) minimizes the sum of multiple (normalized) objectives, depending on the levels of

TABLE 1. MILP model notation.

Sets and indices	
i	Tasks $\tau_i \in \{1, 2, ..., n\}$
j	Activations of $\tau_i \in \{1, 2, ..., N_i\}$
Parameters	
C_i	Worst case computation time of τ_i
D_i	Relative deadline of τ_i
T_i	Period of τ_i
H	Hyperperiod of the task set
N_i	Number of activations of τ_i (H/T_i)
d_{ij}	Due date of τ_i in activation j
R_{ij}	$[j \cdot T_i, (j+1) \cdot T_i]$ Possible execution time interval for task i in activation j
Decision variables	
x_{ijt}	Task execution matrix. 1 if τ_i in activation j is executed at time t and 0 otherwise.
s_{ijt}	Context switch matrix. 1 if τ_i in activation j is active at time t and not at time $t + 1$ or viceversa and 0 otherwise.
w_{ij}	Response time matrix. Response time of τ_i in activation j

importance of each objective. Exact values for K_1 and K_2 will be provided in Section V to evaluate the proposed approaches.

$$\min \text{ Obj} = \sum_{\forall(i,j)} \sum_{\forall t} K_1 s_{ijt} + \sum_{\forall(i,j)} K_2 \frac{w_{ij}}{D_i} \tag{2}$$

$$\text{s.t: } x_{ijt} = 0 \quad \forall t, i, j | t \notin R_{ij} \tag{3}$$

$$\sum_{t \in R_{ij}} x_{ijt} = C_i \quad \forall i, j \tag{4}$$

$$t \cdot x_{ijt} \leq d_{ij} - 1 \quad \forall t \in R_{ij} \tag{5}$$

$$\sum_{(i,j)} x_{ijt} \leq 1 \quad \forall t \in \{0, 1, \ldots, H\} \tag{6}$$

$$s_{ijt} \leq x_{ijt} + x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\} \tag{7}$$

$$s_{ijt} \geq x_{ijt} - x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\} \tag{8}$$

$$s_{ijt} \geq x_{ij(t+1)} - x_{ijt} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\} \tag{9}$$

$$s_{ijt} \leq 2 - x_{ijt} - x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\} \tag{10}$$

$$w_{ij} \geq t \cdot x_{ijt} - j \cdot T_i + 1 \quad \forall t, i, j \tag{11}$$

$$x_{ijt}, s_{ijt} \in \{0, 1\} \tag{12}$$

$$w_{ij} \geq 0 \tag{13}$$

Constraints (3), (4), (5) and (6) are the key constraints and are the basic scheduling conditions. Constraint (3) does not allow a task to be executed outside its possible intervals of execution, $[j \cdot T_i, (j + 1) \cdot T_i]$, for all each activation a. Constraint (4) assures that the task completes all its execution time. Equation (5) ensures real-time requirements, as tasks must end before the arrival of their deadlines. Constraint 6 ensures that only one task is being executed at each point in time.

Constraints (7 - 10) determine if a context switch s_{ijt} happens or not, i.e., if a task is activated in a time t and not in time $t + 1$ or vice versa. This behaviour is treated as an exclusive OR (XOR), which gives true only if one, and only one, of the inputs is true, $x_{ij(t+1)} \oplus x_{ijt}$. As stated before, s_{ijt} takes value 1 if there is a context switch.

Constraint (11) calculates the response time of each activation of all tasks. Equations (12) and (13) represent the decision variable domains.

1) NON-PREEMPTIVE MILP MODEL

The previous approach provides an offline preemptive scheduling plan for a task set. It means that tasks can be interrupted by other tasks and restarted later. If non-preemptive behavior is desired, i.e., if a task has to be executed until it is completed without interruption, a new constraint will be added in the model (Equation (14)).

$$\sum_{\forall(i,j)} s_{ijt} \leq 2 - x_{ij0} \quad \forall t \in \{0, 1, \ldots, H-1\} \tag{14}$$

2) ENHANCED MILP MODEL OF UNIPROCESSOR PERIODIC TASK SYSTEMS

A possible way to reduce solution times of MILP problems is via a *warm start*, i.e., to provide manually starting solution vectors of the problem to the solver [27].

If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not feasible, the MILP solver tries to repair it. When it is difficult to find a good integer feasible solution for a problem, a warm start can significantly reduce the solution time.

The effectiveness of the warm start in MILP solvers depends on many factors. Sometimes, warm starts do not help the solver to find solutions more quickly, but authors may consider providing feasible starting points to fix an upper bound on the objective value (in case of minimization) and thus can be used to prune nodes during the search.

The feasible starting point considered in this work is the optimal scheduling policy known as Deadline Monotonic Scheduling (DM) [28]. DM is a fixed-priority scheduling algorithm that assigns the highest priority to the task with the shortest deadline.

Therefore, an offline feasible scheduling plan for the task set will be generated using DM, and this plan will be the starting point of the Simple MILP model described in Section IV-A.

3) DECISION VARIABLES SIZE

In some cases, the solution time of MILP problems grows exponentially with respect to the problem size. For the simple MILP model, the size of the decision variables is the number of elements in matrices x_{ijt}, s_{ijt} and w_{ij}. Let be S_x, S_s and S_w the number of elements that differ from zero in x_{ijt}, s_{ijt} and w_{ij}, respectively. For all these matrices, the number of rows is equal to the number of tasks in the system, n.

As far as w_{ij} matrix is concerned, the size is (rows x columns):

$$S_w = \sum_{i=1,\ldots,n} N_i \tag{15}$$

However, the number of zero elements in the matrices x_{ijt} and s_{ijt} is considerably large. Therefore, we only count the nonzero elements of those variables when computing the problem size.

For x_{ijt}, the number of nonzero elements in one row i and activation j is C_i. So, in row i for all activations, the number of nonzero elements is:

$$C_i \cdot N_i = C_i \frac{H}{T_i} = U_i \cdot H \tag{16}$$

As x_{ijt} has n rows, the set of points of x_{ijt} to calculate in the MILP problem is:

$$S_x = \sum_{i=1,\dots,n} U_i \cdot H = U \cdot H \tag{17}$$

In s_{ijt}, the same reasoning applies, but the maximum number of elements in a row is $C_i + 2$, considering the worst case in which a context switch occurs in each execution unit of C_i. Thus:

$$S_s = U \cdot H + 2 \cdot \sum_{i=1,\dots,n} N_i \tag{18}$$

The size S of the problem is then:

$$S = S_x + S_s + S_w = 2 \cdot U \cdot H + 3 \cdot \sum_{i=1,\dots,n} N_i \tag{19}$$

which is directly proportional to the number of tasks n, the total utilization U and the length of the hyperperiod H.

As it has been demonstrated, the simple MILP model is too computationally intensive to be used for large task sets with high utilization and long hyperperiods. For this reason, we propose an alternative MILP formulation in the next section.

B. ROLLING TASK MILP MODEL

In the rolling task scheme, the MILP problem consists of scheduling only one task in the hyperperiod. Once all activations of this task are allocated, the next task is chosen. The previous solution is an input to the new problem as a new constraint. This constraint implies that the new task can not be allocated in the same instants of time that the first one. This process is repeated until all tasks are allocated. The algorithm is explained in Fig. 1. For instance, let us define a set of three tasks $\tau_i = (C_i, T_i)$, with implicit deadlines ($D_i = T_i \ \forall i$), being $\tau_0 = (2, 5)$, $\tau_1 = (3, 9)$ and $\tau_2 = (3, 18)$. Note that tasks are ordered by increasing deadlines. First, τ_0 is allocated with the objective of minimizing its response time and context switches. As a result, plan σ_0 is defined. τ_1 is then allocated in the idle slots of time of plan σ_0. Note that, among all possibilities of allocation, the algorithm selects the one with the most balanced weighted sum of context switches and response times. As a result, plan σ_1 is defined. Finally, τ_2 is allocated in the idle slots of time of plan σ_1, following the same criteria.

If tasks are chosen in a priority scheme, the result is similar to the fixed priority non-preemptive scheduling algorithm with priorities equal to deadlines. Note that the reduction in time in rolling MILP approach comes from the a priori determined priority ordering.

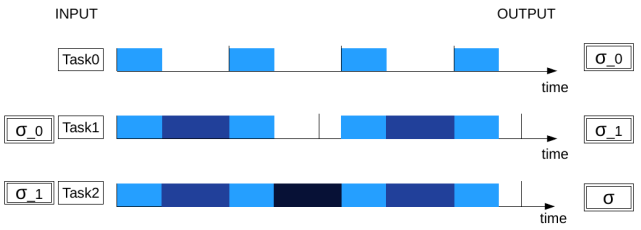


FIGURE 1. Functioning of the rolling task algorithm.

The optimization criteria are maintained as in the simple MILP problem, minimizing both context switch and response time.

With this purpose, variations over the previous approach are done and are explained in Algorithm 1. The algorithm works as follows: the system receives the task set ordered by increasing deadlines. The first task, $i = 0$, is selected in order to be optimized, with the objective of reducing its context switch and response time. To avoid that other tasks are now included in the model, we force that all $x_{ijt} = 0$ for $i > 0$ (line 8). Once this is done, the variable x_{00t} is saved in a global auxiliary variable that contains the plan, σ_{ijt} (line 14). Previous constraints are then removed (line 18) and the model is updated (line 19). The algorithm moves to next task, $i = 1$. To maintain the previous assigned values of x, we force $x_{ijt} = \sigma_{ijt}$ for those tasks $i < 1$ and $x_{ijt} = 0$ for $i > 1$ (line 10). x_{11t} is then optimized. This process is repeated for all tasks and the optimized scheduling plan is saved in the variable σ_{ijt}.[1]

The previous algorithm is translated into an MILP problem, which is similar to that stated in Section IV-A. For each task, the value of the objective function is calculated in Equation (20). The difference between this equation and the one in the simple MILP is that Equation (20) evaluates the objective function once for each task, while Equation (2) evaluates it for all the tasks in one step.

$$\min \text{Obj} = \sum_{\forall j \in N_i} \sum_{\forall t} K_1 s_{ijt} + \sum_{\forall j \in N_i} K_2 \frac{w_{ij}}{D_i} \tag{20}$$

$$\text{s.t: } x_{kjt} = \sigma_{kjt} \quad \forall k, t | j \in [0, 1, \dots, i-1], \quad t \in \{0, 1, \dots, H\} \tag{21}$$

$$x_{kjt} = 0 \quad \forall k, t | j \in I - [0, 1, \dots, i], \quad t \in \{0, 1, \dots, H\} \tag{22}$$

$$x_{ijt} = 0 \quad \forall t, j | t \notin R_{ij}, j \in N_i \tag{23}$$

$$\sum_{t \in R_{ij}} x_{ijt} = C_i \quad \forall j \in N_i \tag{24}$$

$$t \cdot x_{ijt} \leq d_{ij} - 1 \quad \forall t, j | t \in R_{ij}, j \in N_i \tag{25}$$

[1] In Fig. 1, variable σ_{ijt} omits subindexes j and t for improved clarity and only refers to the task i.

Algorithm 1 Rolling Task MILP Algorithm

```
INPUT: Task set with n tasks ordered by increasing deadlines
OUTPUT: Scheduling plan, σ
procedure ROLLING TASK MILP ALGORITHM(τ)
    Calculate H, d_ij, R_ij
    Plan σ_ijt → 0
    for τ_i ∈ τ do
        if i < n then
            x_kjt = 0    i < k ≤ n
        if i > 0 then
            x_kjt = σ_kjt
        add Constraints for task i
        set Objective and optimize
        if feasible then
            σ_ijt = x_ijt
        else
            Exit
        if i < n-1 then
            Remove constraints of the model
            Update the model
```

$$\sum_{l} x_{ijt} \leq 1 \quad \forall t \in \{0, 1, \ldots, H\} \tag{26}$$

$$s_{ijt} \leq x_{ijt} + x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\}, j \in N_i \tag{27}$$

$$s_{ijt} \geq x_{ijt} - x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\}, j \in N_i \tag{28}$$

$$s_{ijt} \geq x_{ij(t+1)} - x_{ijt} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\}, j \in N_i \tag{29}$$

$$s_{ijt} \leq 2 - x_{ijt} - x_{ij(t+1)} \quad \forall t, i, j | t \in \{0, 1, \ldots, H-1\}, j \in N_i \tag{30}$$

$$w_{ij} \geq t \cdot x_{ijt} - j \cdot T_i + 1 \quad \forall t, j | j \in N_i \tag{31}$$

$$x_{ijt}, s_{ijt} \in \{0, 1\} \tag{32}$$

$$w_{ij} \geq 0 \tag{33}$$

In each pass of the algorithm (i.e., for each task) only that task is evaluated. Therefore, it inherits the execution plan of previous tasks (Constraint 21) and ignores next tasks (Constraint 22). The other constraints are similar to the ones presented in Section IV-A, considering only a single task rather than a set of tasks. Note that the non-preemptive version of the rolling task MILP model is easy to obtain by adding constraint 14 to the previous model.

C. PARTITIONED SYSTEMS MILP MODEL

The model presented in Section IV-B can be used to schedule a partitioned system, as defined in Section III-B. The idea is to make a hybrid approach between the periodic and the rolling task model.

Given a task model where each task is defined as $\tau_i = (C_i, D_i, T_i, \rho_i)$, the rolling partition approach selects the first partition and schedules all tasks within this partition, following the rolling task approach defined in Section IV-B. The next partition is then selected and all its tasks are scheduled. The algorithm ends when all partitions are allocated. It is explained in Algorithm 2.

Algorithm 2 Rolling Partition MILP Algorithm

```
INPUT: Task set with n tasks ordered by increasing deadlines
OUTPUT: Scheduling plan, σ
procedure ROLLING PARTITION MILP ALGORITHM(τ)
    Calculate H, d_ij, R_ij
    Order tasks by number of partition → Partitions[ ]
    for ρ_i ∈ Partitions do
        for τ_i ∈ ρ_i do
            Rolling task MILP algorithm(τ_i)
```

We have made numerous simulations to investigate the feasibility of this method; however, we found some counter examples that make it necessary to propose another approach to solve the partitioned MILP scheduling problem. First, we are going to show one of these counter examples.

1) COUNTER EXAMPLE

Given a task set as shown in Table 2, we can schedule this system following a flat scheduling approach. But first, the hierarchical scheduling is presented. Partitioned software architectures define two levels of hierarchy: the global level, with a scheduler that allocates CPU time to partitions; and a local scheduler per partition, which schedules the tasks using the available time per partition. Flat scheduling is considered [29] among the strategies that can be followed to achieve hierarchical scheduling. This approach consists in removing all the barriers defined by the partitioned system and schedule and considering all the tasks at once. We then suppose that a single global scheduler is in charge of managing all the tasks and conducts the corresponding scheduling algorithm. The last step is to adapt the solution back to the partitioned system by grouping. With this, the final schedule will be very efficient and, sometimes, optimal. In this strategy, the timing knowledge of tasks should be previously detailed in depth in order to analyze the overall system and optimize the solution.

TABLE 2. Task set parameters τ.

	C_i	D_i	T_i	ρ_i
τ_0	2	5	5	0
τ_1	3	10	10	1
τ_2	4	20	20	0

Fig. 2 shows the scheduling plan obtained with this approach. To schedule the flat system, we can use both the DM or the rolling task MILP algorithm. As seen in the temporal plan, all deadlines are met, so the system is schedulable.

We are now applying the rolling partition MILP algorithm to the same set. According to Algorithm 2, partition 0 is

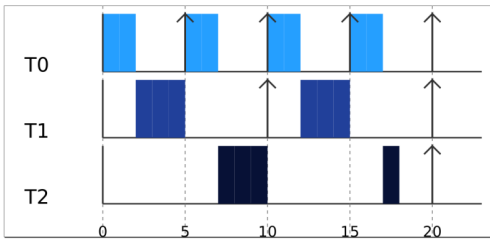
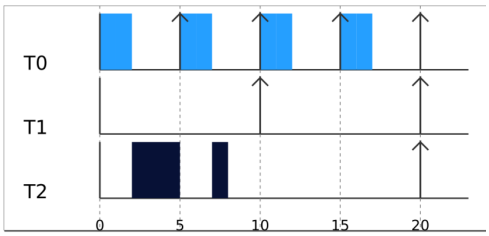
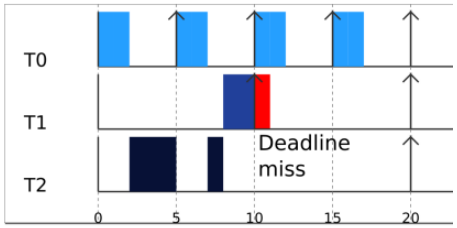


FIGURE 2. Task set τ scheduled with DM.

initially selected. All tasks within that partition (τ_0 and τ_2) are then allocated in the core, using the rolling task algorithm. The result is shown in Fig. 3a. The algorithm then moves to partition 1 and tries to allocate the tasks within that partition (τ_1) into the available slots of the previous scheduling plan. As shown in Fig. 3b, it is impossible to execute τ_1 entirely, and the task misses the deadline in its first execution. Therefore, the system is not schedulable.



(a) Partition 0 allocation.



(b) Partition 1 allocation.

FIGURE 3. Task set τ scheduled with Rolling partition algorithm.

As shown in this example, the rolling task approach performs well in terms of feasibility, but its extension to the hybrid rolling partition algorithm does not provide a feasible result. Therefore, another approach for partitioned systems must be proposed.

2) HIERARCHICAL MILP MODEL FOR PARTITIONED SYSTEMS

As mentioned in the previous section, we will consider a hierarchical system with global and local levels. We consider a periodic server at the global level and the rolling task MILP approach at the local level.

At the global level, the scheduler periodically assigns the CPU to the partition, i.e., $\Gamma = (\theta, \pi)$ provides θ units of CPU every π units of time. Once time has been assigned for each partition, the local scheduler allocates the tasks that belong to the partition in their corresponding time slots, following the rolling task algorithm.

The method for calculating the global level periodic resource is explained hereafter: the first step consists of calculating the amount of CPU required for each partition P_i as the sum of the utilizations required for its tasks:

$$U_{P_i} = \sum_{\forall \tau_m \in P_i} \frac{C_m}{T_m} \tag{34}$$

We now have to fix θ_{P_i} or π_{P_i} and calculate the other using U_{P_i}. We will set the period of the partition to the shortest period of the tasks that belong to this partition. The credit of the periodic server for each partition θ_{P_i} is then calculated as:

$$\theta_{P_i} = U_{P_i} \cdot min(T_m) \quad \forall \tau_m \in P_i \tag{35}$$

Algorithm 3 explains the hierarchical MILP model. First, the global level scheduling is evaluated (line 3). For each partition, the algorithm calculates the periodic resource through the parameters of the tasks that belong to that partition (lines 6-7). The output of this function is the periodic resource for each partition (line 8). The local level scheduling is then calculated for each partition (line 9). The function uses as an input parameter the periodic resource of each partition and schedules the tasks in this resource (line 13) to generate the temporal plan. If the partition does not completely use its credit to schedule tasks, the remaining time will be transferred to the next partitions (lines 14-15). As a result, the local plan for each partition is calculated (line 16).

Algorithm 3 Hierarchical MILP Algorithm

1: INPUT: Task set with n tasks ordered by increasing deadlines
2: OUTPUT: Scheduling plan, σ
3: **procedure** GLOBAL-LEVEL-SCHEDULING(τ)
4: Group tasks by partition index $\rightarrow$ *Partitions*[ ]
5: **for** $\rho_i \in$ Partitions **do** ▷ Calculate the periodic resource for each partition
6: $\pi_{\rho_i} = \min(T_j) \quad \forall$ task$_j \in \rho_i$
7: $\theta_{\rho_i} = \text{int}\left(\pi_{\rho_i} \cdot \sum_{task_j \in \rho_i} U_j\right)$
8: **return** $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i}) \quad \forall \rho_i \in$ *Partitions*[ ];
9: **procedure** LOCAL-LEVEL-SCHEDULING($\Gamma(\pi, \theta)$, τ, *Partitions*[ ])
10: **for** $\rho_i \in$ Partitions **do**
11: Initialize plan σ
12: **for** $\tau_j \in \rho_i$ **do** ▷ Application of rolling task MILP approach
13: σ += Schedule τ_j in $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i})$
14: Idle-time$_i$ = $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i})$ - σ
15: $\sigma \quad \forall i \in$ *Partitions*[ ]
16: **return** $\Gamma_{\rho_i}(\pi_{\rho_i}, \theta_{\rho_i}) \quad \forall \rho_i \in$ *Partitions*[ ];

For the task set defined in Table 2, the global level periodic resource is calculated as:

$$U_{P_0} = \frac{2}{5} + \frac{4}{20} = 0.60$$

$$U_{P_1} = \frac{3}{10} = 0.30$$
$$\theta_{P_0} = 0.60 \cdot 5 = 3$$
$$\theta_{P_1} = 0.3 \cdot 10 = 3$$

Therefore, the periodic resources for the partitions are modelled as: $\Gamma_{\rho_0} = (3, 5)$ and $\Gamma_{\rho_1} = (3, 10)$, assuming integer numbers for all parameters. The global scheduling plan is depicted in Fig. 4.

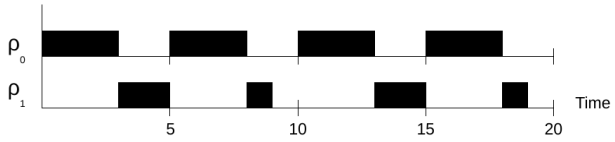


FIGURE 4. Periodic server in the global level.

We followed the same approach for a small instance of the problem presented in Table 2. As shown in Fig. 5, the plan is schedulable as all the tasks at different activations are completed before their deadlines. Moreover, the obtained schedule is very efficient in terms of the context switch.

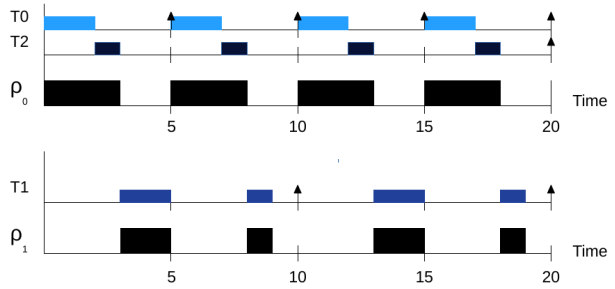


FIGURE 5. Rolling task MILP in the local level.

D. SELECTION OF THE WEIGHTS IN THE MULTIOBJECTIVE FUNCTION

A method to select the weights K_1 and K_2 in order to optimize both objectives is presented in [26] using a goal programming objective method. The procedure consists of solving the MILP problem twice, once for each objective and then obtain a function that closely approximates both optimal values. For this purpose, the task set defined in Table 2 is used and the target values are obtained:

TABLE 3. Target values for single objectives.

Objective	Response Time	Context switch
Minimize Response time	3.1	14
Minimize Context Switches	4.95	12

We then minimize the percentage deviations from the target values to optimize the response time and context switch, simultaneously (Equation (36)):

$$\min K_1 \sum_{\forall(i,j)} \sum_{\forall t} \frac{s_{ijt} - 12}{12} + K_2 \sum_{\forall(i,j)} \frac{1}{D_i} \frac{WCRT_i - 3.1}{3.1} \quad (36)$$

Depending on each objective's relative importance, the value of K_1 and K_2 might vary in the range of [0, 1] such that $K_1 + K_2 = 1$ (Fig. 6). It is the decision maker's responsibility to choose the weights that best fit the requirements.

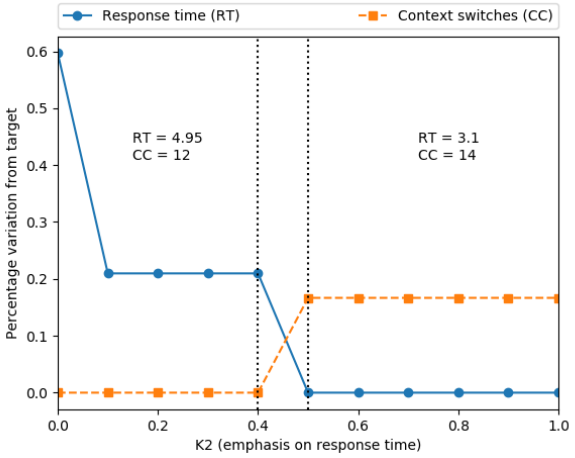


FIGURE 6. Optimal solutions for minimizing the weighted sum of percentage variations of the objectives, with different values for K_1 and K_2.

V. EXPERIMENTAL EVALUATION

This section evaluates both the quality of generated schedules and the required time for obtaining them. The assessed MILP models are:

- Simple (GS) presented in Section IV-A1.
- Simple with start solution (GS2) presented in Section IV-A2.
- Rolling task (GR) presented in Section IV-B.

As far as weights K_1 and K_2 is concerned, we have considered $K_1 = 1$ and $K_2 = 1$ for GR, GS and GS2. These weights are set in this way in order to provide a balanced solution in which context switch and WCRT are reduced equally. The relation between the response time of a task and its deadline is always less than or equal to one (a task cannot end after its deadline). Therefore, the weight of the context switch is usually slightly greater than the weight of the response time. Finally, we evaluate the preemptive versions of all MILP models.

A. EXPERIMENTAL CONDITIONS

The simulation scenario developed for this work is depicted in Fig. 7. It is divided into three steps:

- Generation of the load (see Section V-A1).
- Execution of MILP approaches and DM (see Section IV).
- Validation (see Section V-A2).

The automatic load generator generates a task set and the associated parameters with the process described in Section V-A1. This task set is the input for the 3 MILP approaches and the DM scheduler that generates the corresponding scheduling plans. If the obtained plans are then valid, their performance parameters will be stored. This

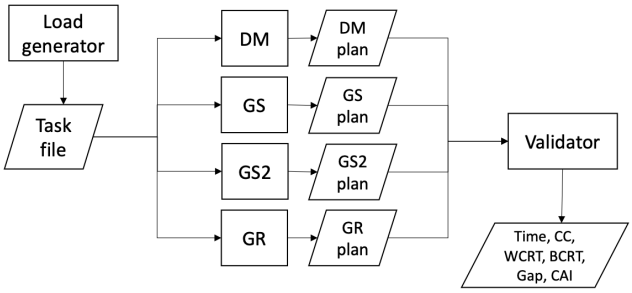


FIGURE 7. Experimental evaluation overview.

sequence is repeated to complete a sufficient number of simulations.

We use Gurobi optimizer [30], from Gurobi Optimization, Inc., which is a powerful optimizer designed from scratch to run in multi-core and with capability to run in parallel mode. It has achieved performance improvement with each version and provides a Python interface. Version 9.0 [30] is selected for solving our proposed MILP formulations. All MILP approaches described in previous sections are executed on an Intel Core i7 CPU with 16GB of RAM.

1) LOAD GENERATOR

The load is generated using a synthetic task generator. Given the system utilization value and a number of tasks for each set, the utilization is shared among the tasks using the UUniFast discard algorithm [31]. Computation times are generated randomly in [2,10] and periods are deduced from the system utilization. We restricted the hyperperiod to be no larger than 5000, so we discard the sets that exceed this limit, repeating the generation again until enough feasible instances are found.

2) VALIDATION OF THE RESULTS

The validation consists of two steps. Firstly, we must check feasibility to ensure all deadlines are met. Secondly, several performance parameters are obtained to compare different methods. Specifically, these are the parameters obtained for each set:

- Response times: We calculate best and worst case response times as defined in Section III-A. In order to evaluate WCRT and BCRT of the task set, they will be calculated as the average between $WCRT_i/D_i$ and $BCRT_i/D_i$ for all tasks in the set, respectively.
- Number of context switches (CC): This parameter is computed by counting the number of times tasks are preempted by others.
- Solution time: For MILP approaches, we calculate the time spent to obtain a solution. As simple MILP models are too computationally-intensive for some task sets, the solver is limited to a certain time limit of solution time. After this time, the solution is stored.
  To pick the best time limit, we vary the GS algorithm time limits to between 100 and 3000 s to solve different instances of the problem. We then plot the percentage of the cases that have been solved to optimality (optimality gap <= 0.001) vs. the time limit. As depicted in Figure 8, up to 500 s, the percentage of optimal cases increases significantly. However, the improvement is negligible after that. Therefore, we conclude that 500 s is the right choice for the time limit.

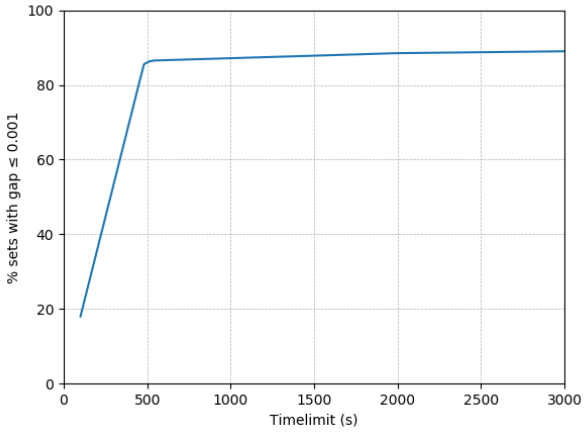


FIGURE 8. Experiment to select the timelimit parameter.

- Optimality gap: If the previously chosen time limit is reached, the solution is not optimal. Therefore, the distance to the optimal solution should be analyzed. This distance is called the *gap* hereafter. It is the distance between the estimated lower bound (for minimization problem) and the best feasible solution that has been found so far. As looking for proven optimal solutions takes a long time to compute, a common practice is to look for a solution that guarantees not worse than x% (gap) from the optimal solution. A significant advantage is that most of the gap is often reduced quite early.
- Control action interval (CAI): It is the percentage variation of the control action delivery of a task relative to its period [7]. This parameter allows us to compare the performances of a control system. The CAI of a task i is calculated as:

$$CAI_i(\%) = \frac{WCRT_i - BCRT_i}{T_i} \cdot 100 \qquad (37)$$

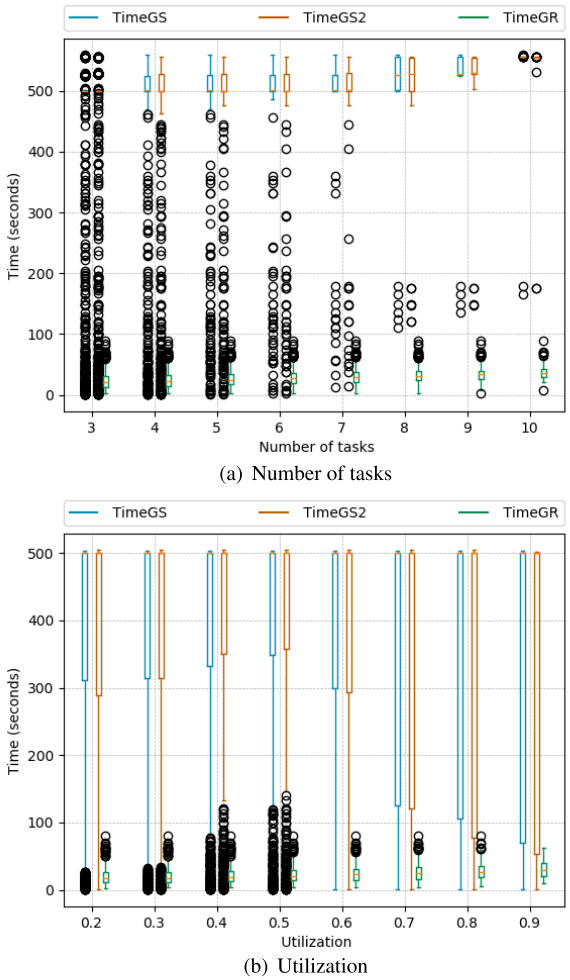
B. EXPERIMENTAL RESULTS

This subsection evaluates the performance of the proposed MILP approaches and the scheduling policy DM using the performance parameters (i.e., quality of the solution and computation intensity) that are obtained in the evaluation process.

1) COMPARISON OF ILP APPROACHES

a: SOLUTION TIME

The results show that both the simple model and the simple model with a warm start take a long time to run. In fact, the median of all the simulations with GS and GS2 approximates the time limit (500 seconds), as depicted in Fig. 9. When the time limit is usually reached, only good solutions



(a) Number of tasks

(b) Utilization

FIGURE 9. Solution times depending on the utilizations and the number of tasks of the sets.

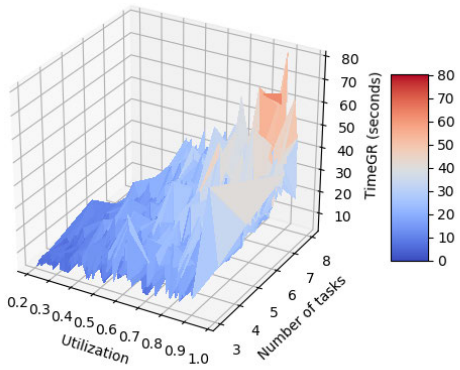


FIGURE 10. Influence of the number of tasks and system utilization in the solution time of the GR algorithm.

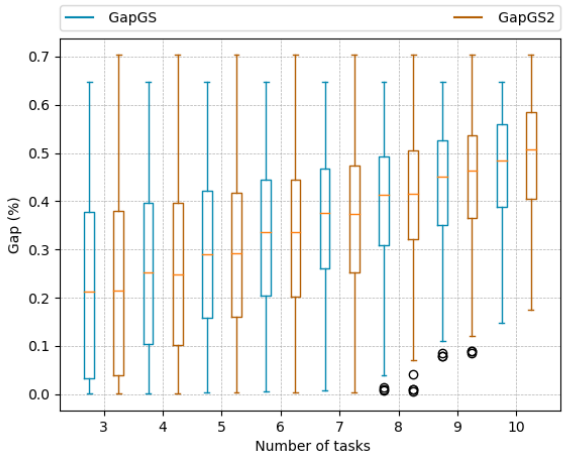


FIGURE 11. Influence of the number of tasks in the percentage of optimality gap.

are found and not the optimal solution. In the case of a rolling approach, promising results are achieved. GR needs five times less solution time than GS and GS2.

In terms of solution time and computation intensity, it is clear that the best approach is GR. In Fig.10, we can see the relationship between utilization, the number of tasks, and the solution time for GR. As shown in Section IV-A3, the solution time increases with the number of tasks and utilization.

We then compare the rest of the performance parameters to check if it is worth spending computation time to obtain more significant results.

b: GAP

This subsection studies the GS and GS2 optimality gaps as they could not find the optimal solution in the selected time limit. As observed in Fig. 11, the optimality gap is strikingly similar in GS and GS2. Therefore, providing a first solution (warm start) did not improve the solution time nor the solution quality. It is further observed that the gap increases with the number of tasks in the system, which is in line with the results in the previous section.

c: WORST CASE RESPONSE TIME

Regarding the WCRT, we can see in Fig. 12 that DM achieves the best result. This is expected as DM schedules tasks with shorter deadlines first. But regarding MILP approaches, GR achieves better results than GS or GS2. This is because GR obtains an optimal solution while GS and GS2 reach the time limit without finding an optimal solution in the major part of the task sets.

d: CONTEXT SWITCHES

Regarding the number of context switches, as depicted in Fig. 13, GS and GS2 approaches produced fewer context switches than GR and DM. It should be noted that GS and GS2 coincide in this figure.

It is clear that, as far as CC is concerned, DM will obtain the worst results, since it is not a scheduling algorithm focused on reducing CC. Regarding GS, GS2 and GR, all have CC minimization as objective. However, the differences between GR and GS or GS2 are related to the way in which GR performs the optimization. As GR optimizes by following a rolling task approach, it means that tasks are scheduled in lower deadline order. This is very similar to DM and this

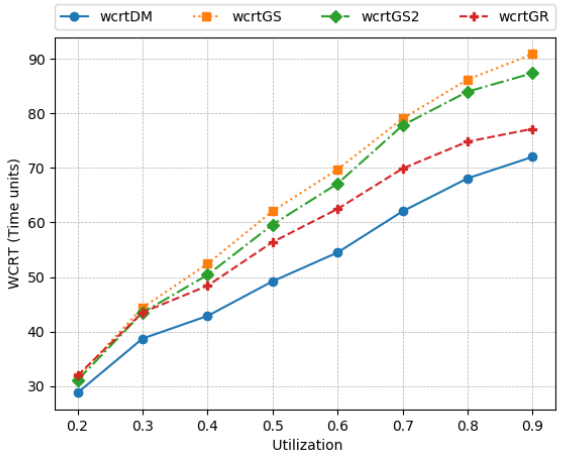


FIGURE 12. Influence of the system utilization in the worst case response times.

is why GR achieves worse results in terms of CC than GS and GS2. Nonetheless, as GR has CC minimization in its objective, it achieves better results than DM.

As a reminder, the authors consider reducing the response time and the number of context switches as the main objectives of the problem. DM is the approach that generates a plan with the shortest response times and, in contrast, more context switches. GS is completely the opposite. This is due to the weights of the objectives in the multiobjective function. Depending on these coefficients, the results may change, as explained later in Section V-B2.

e: BEST CASE RESPONSE TIME

In the case of the best case response time, GS is the approach with the shortest BCRT than other approaches. The ratio between BCRT and deadlines increases with the number of tasks and system utilizations, as seen in Fig. 14.

f: CONTROL ACTIVATION INTERVAL

Finally, regarding to the percentage of CAI, DM is the approach that implies a better system performance, as observed in Fig. 15. This may be logical since GR and GS do not explicitly use a criterion to minimize CAI.

As a conclusion, we consider that GR the best approach in terms of time and performance and it will be studied in depth hereafter. Moreover, through an in-depth study, the GR approach is extended and applied to different objectives. This leads us to the next section, where we will investigate the effect of varying the constants K1 and K2.

2) INFLUENCE OF THE WEIGHTS OF THE OBJECTIVES IN THE OBJECTIVE FUNCTION

As stated in Section IV-A, the objective function covers the total number of context switches, s, and the total response time, w, for all tasks.

By changing the weights K_1 and K_2, we can derive as many versions of GR as we want, depending on the system requirements. For example, a particular case of GR is GR2,

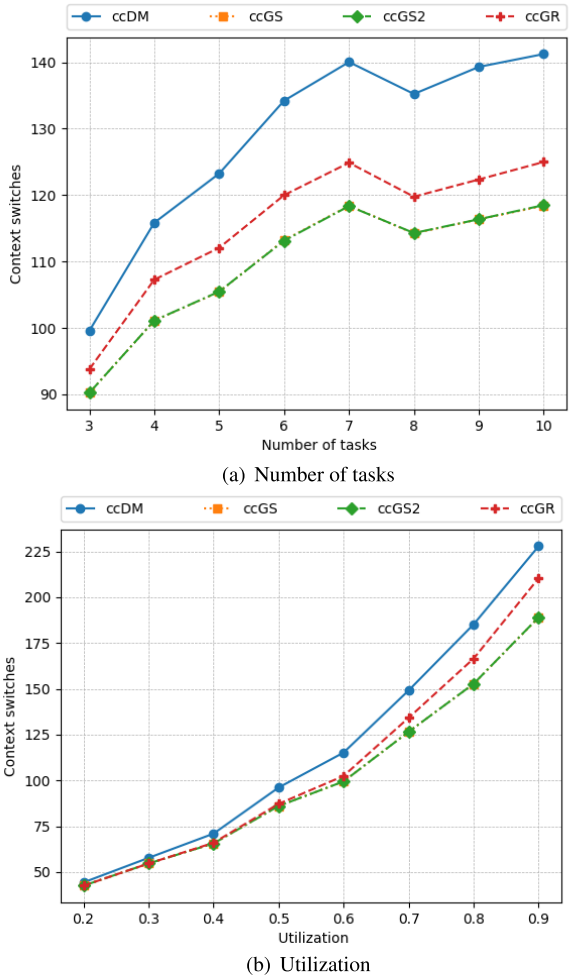


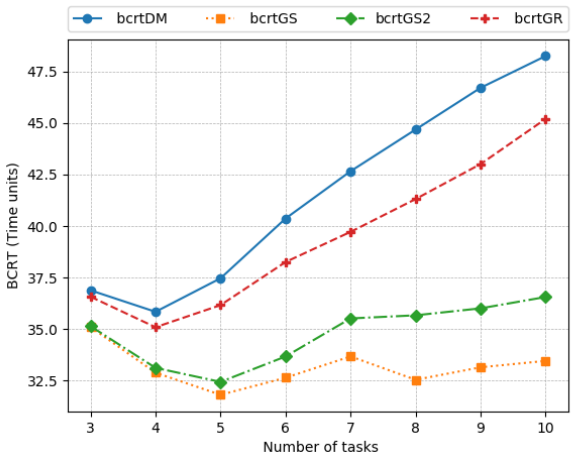
FIGURE 13. Impact of the system utilization and number of tasks in the context switches.

which corresponds to the GR approach with a single objective: to reduce the response time of the system. Therefore, the context switch is removed from the objective function, i.e., $K_1 = 0$ and $K_2 = 1$ in Equation (2).

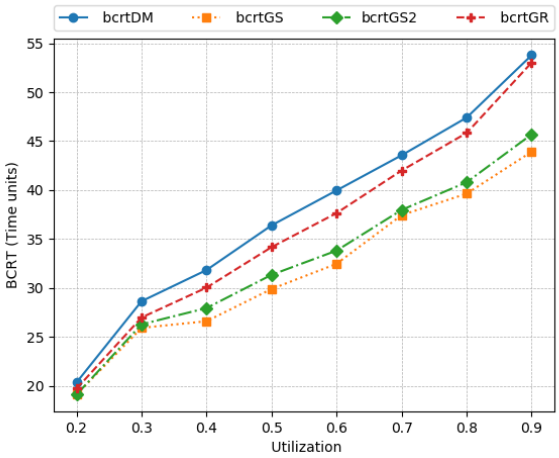
In this subsection, GR and GR2 approaches are compared, and DM is used as a reference in the evaluation of the results.

For this purpose, simulations with hyperperiods in [0, 3500] are made and the solution time is limited to 1800 s. Furthermore, utilization factors are defined in range [0.1, 0.8] and the number of tasks in [2, 10] per set.

As seen in Fig. 16, the solution time grows exponentially for GR and more slowly for GR2 as the objective function has been simplified. Reducing the number of objectives in the objective function significantly reduces the solution time. Looking into Fig. 17, we can conclude that the GR2 approach is the most similar approach to the DM approach in terms of system performance. Although GR slightly reduces the number of context switches and BCRT compared to GR2 and DM, it implies worse performance in WCRT and CAI. It is possible to conclude that regarding WCRT, the GR2 approach reduces the response times even below the DM approach, and this is a favorable result.

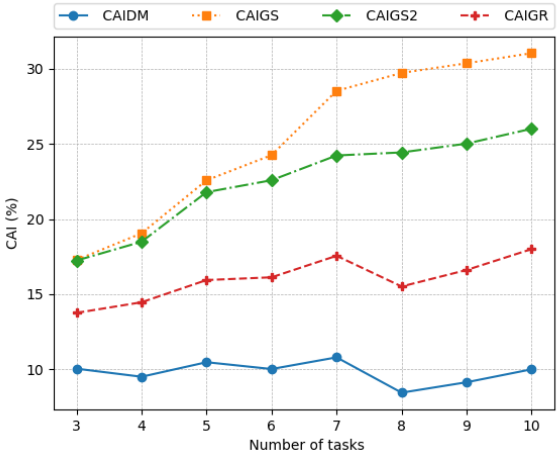


(a) Number of tasks

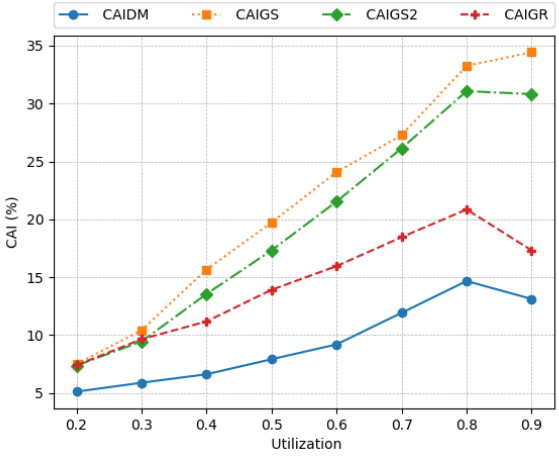


(b) Utilization

FIGURE 14. Influence of the system utilization and number of tasks in the best case response times.



(a) Number of tasks



(b) Utilization

FIGURE 15. Impact of the system utilization and number of tasks in the control activation interval.

As discussed, the system's performance parameters change when the objective function varies i.e., considering different terms in the objective function or changing the weights associated with each objective. Obviously, the weights of the objective function should be customized for the needs of the problem. For example, in control applications, it is desirable to minimize the computational delay in the controller, as well as the sampling jitter and the control jitter [32]. The implementation of the minimization of computational delay is explained in the next subsection.

C. REDUCING CONTROL ACTIVATION INTERVAL

A variant form of the GR algorithm, GR3, is defined here with the CAI and response time in the objective function, i.e., Equation (38). The first part and the second parts of the equation minimize the CAI and the task's overall response time, respectively.

$$\min \text{Obj} = \frac{1}{T_i}(WCRT_i - BCRT_i) + \sum_{\forall j \in N_i} \frac{1}{D_i} w_{ij} \qquad (38)$$

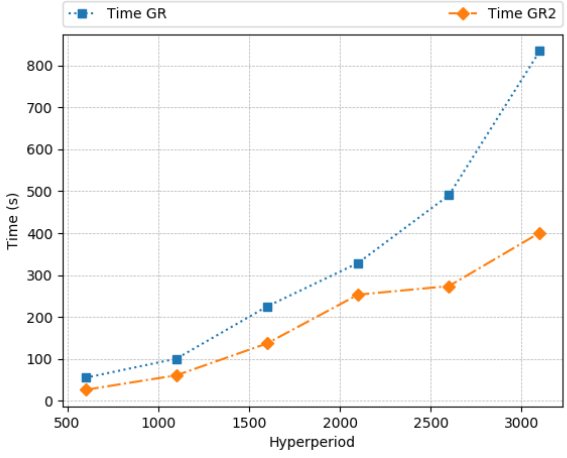
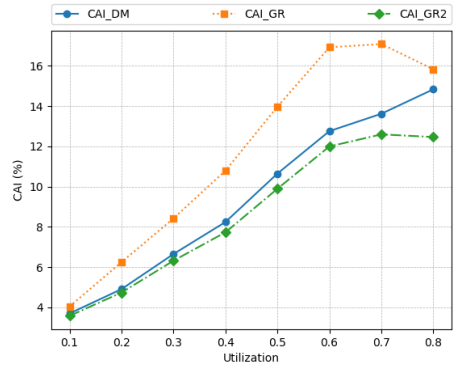


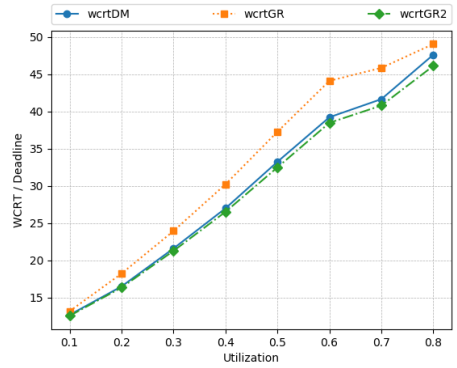
FIGURE 16. Solution time in GR and GR2 approaches depending on the system hyperperiod.

In this subsection, GR, GR2 and GR3 approaches are compared, and DM is used as a reference in the evaluation of the results. Table 4 resumes the objective of each approach.

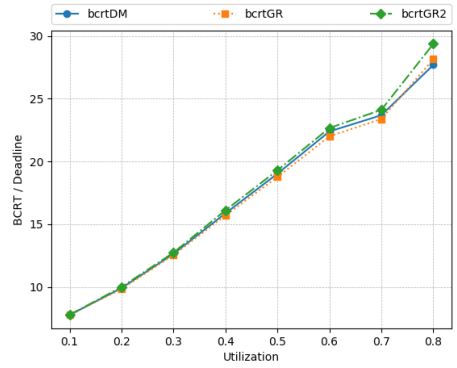
As seen in Fig. 18, minimizing the CAI results in the best results among all approaches. GR3 approach provides



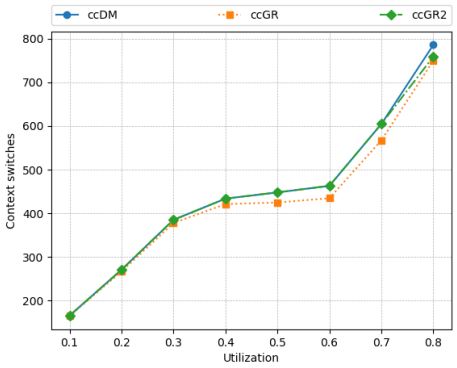
(a) CAI



(b) WCRT



(c) BCRT



(d) Context switches

FIGURE 17. Performance parameters in GR and GR2 approaches depending on the system utilization.

slightly better results than GR2 in terms of CAI and both, GR2 and GR3 improve the system's performance compared with the DM approach. This is a promising result of the work.

TABLE 4. Summary table of rolling task approaches.

Approach	Objective
GR	Minimize response time and context switches
GR2	Minimize response time
GR3	Minimize CAI and response time
DM	Deadline Monotonic Scheduling

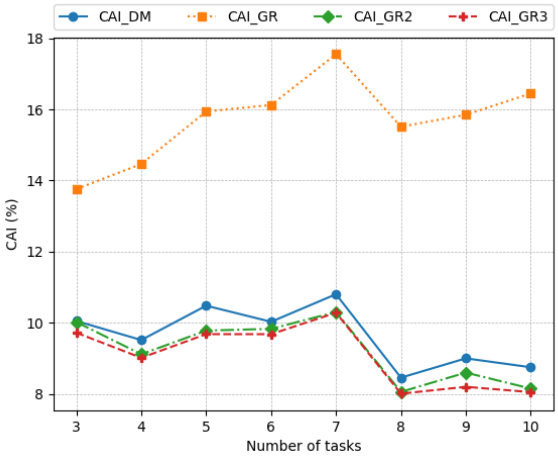


FIGURE 18. CAI for different approaches depending on the number of tasks of the system.

D. EVALUATION OF HIERARCHICAL MILP MODEL FOR PARTITIONED SYSTEMS

For the evaluation of partitioned systems, the proposed hierarchical MILP model has been applied to a real design case from the avionics domain [33], described in Table 5.

TABLE 5. Real design case from the avionics domain.

	C_i	D_i	T_i	ρ_i
τ_0	1	25	25	ρ_0
τ_1	3	50	50	ρ_0
τ_2	2	50	50	ρ_1
τ_3	1	50	50	ρ_2
τ_4	1	25	25	ρ_3
τ_5	1	50	50	ρ_3
τ_6	2	100	100	ρ_3
τ_7	5	200	200	ρ_3
τ_8	1	50	50	ρ_4
τ_9	1	50	50	ρ_4

Firstly, the global level periodic resource is calculated as:

$$U_{P_0} = \frac{1}{25} + \frac{3}{50} = 0.10$$
$$U_{P_1} = \frac{2}{50} = 0.04$$
$$U_{P_2} = \frac{1}{50} = 0.02$$
$$U_{P_3} = \frac{1}{25} + \frac{1}{50} + \frac{2}{100} + \frac{2}{100} = 0.105$$
$$U_{P_4} = \frac{1}{50} + \frac{1}{50} = 0.04$$
$$\theta_{P_0} = 0.10 \cdot 25 = 2.5$$
$$\theta_{P_1} = 0.04 \cdot 50 = 2$$

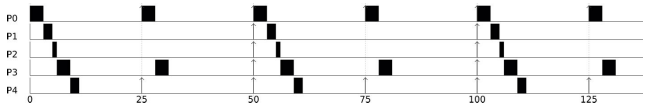


FIGURE 19. Periodic server in the global level for real design case described in Table 5.

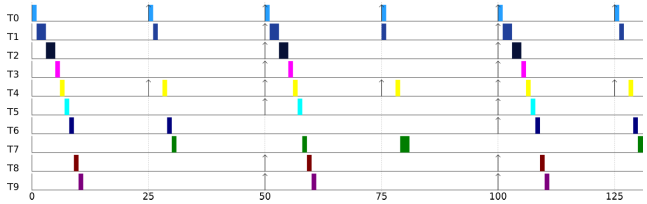


FIGURE 20. Rolling task MILP in the local level for real design case described in Table 5.

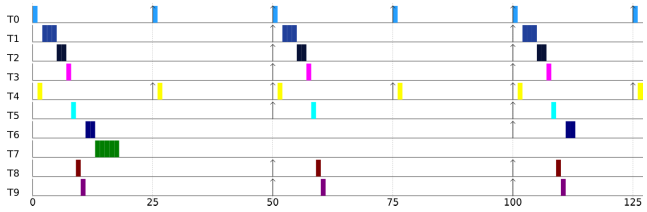


FIGURE 21. DM algorithm for real design case described in Table 5.

$$\theta_{P_2} = 0.02 \cdot 50 = 1$$
$$\theta_{P_3} = 0.105 \cdot 25 = 2.625$$
$$\theta_{P_4} = 0.04 \cdot 50 = 2$$

Therefore, the periodic resources for the partitions are modelled as: $\Gamma_{\rho_0} = (3, 25)$, $\Gamma_{\rho_1} = (2, 50)$, $\Gamma_{\rho_2} = (1, 50)$, $\Gamma_{\rho_3} = (3, 25)$ and $\Gamma_{\rho_4} = (2, 50)$, assuming integer numbers for all parameters. The global scheduling plan that fulfills with this periodic resources is depicted in Fig. 19. Note that because of space limitations, the scheduling plan is limited to 130 units of time instead of all the hyperperiod.

Fig. 20 shows the local scheduling plan according to the rolling MILP approach.

Now, we are going to compare this approach with flat scheduling with DM algorithm [29]. Fig. 21 depicts the scheduling plan for the case defined in Table 5 following DM algorithm.

If the executions of the tasks that belong to the same partition are grouped, the execution in the global level is depicted in Fig.22:

Looking at Figs. 19 and 22, it is obvious that at the global level, hierarchical allocation achieves fewer context switches than flat scheduling. In fact, it is reduced by 35%. This reduction is due to the fact that in the case of flat scheduling, tasks are scheduled independently of the partition to which they belong. The exact reduction will depend on how the credit θ and the period π in the global level are selected. The criteria to select these values are out of the scope of this article, but there are several resource-partitioned models such as the regularity-based resource partitioning (RRP) model [34], the periodic model [35], and the explicit deadline periodic model [36] that can be used to calculate θ and π.

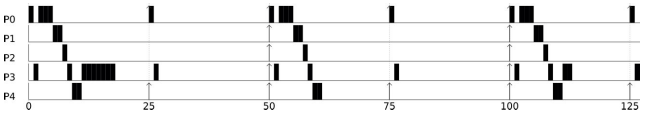


FIGURE 22. Periodic server in the global level for real design case described in Table 5.

This evaluation concludes that the hierarchical MILP model can successfully schedule both levels with better results regarding context switches. This is especially important in partitioned systems where there are two types of context switches: between partitions (global level) and between tasks inside a partition (local level).

VI. CONCLUSION

We have explored different MILP techniques to schedule uniprocessor hard real-time systems. Our goal was to demonstrate that the MILP technique is an excellent alternative to heuristic scheduling algorithms, even in uniprocessor systems. We have proposed a MILP technique that achieves good performance in solution times and obtains better schedules than heuristics in terms of response times and jitter and generally in any desired performance parameter due to the possibility of customizing the optimization criteria. We have also proposed an MILP formulation for scheduling partitioned systems, considering them as two-level hierarchical systems. Further work involves evolving the approaches into more complex models and architectures, especially multicore architectures with more realistic models such as tasks with precedence relations, mixed-criticality systems, and power consumption reduction.

REFERENCES

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–37, Jan. 2018.
[2] *Avionics Application Software Standard Interface (ARINC-653). Part 1—Required Services*, Airlines Electron. Eng. Committee, Annapolis, MD, USA, Mar. 2006.
[3] T. P. Baker and A. Shaw, "The cyclic executive model and Ada," in *Proc. Real-Time Syst. Symp.*, Dec. 1988, pp. 120–129.
[4] C. D. Locke, "Software architecture for hard real-time applications: Cyclic executives vs. Fixed priority executives," *Real-Time Syst.*, vol. 4, no. 1, pp. 37–53, Mar. 1992.
[5] K. Jeffay, "Scheduling sporadic tasks with shared resources in hard-real-time systems," in *Proc. Real-Time Syst. Symp.*, Dec. 1992, pp. 89–99.
[6] H. J. Rivera-Verduzco and R. J. Bril, "Best-case response times of real-time tasks under fixed-priority scheduling with preemption thresholds," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.* New York, NY, USA: Association Computing Machinery, Oct. 2017, pp. 307–346.
[7] A. Crespo, I. Ripoll, and P. Albertos, "Reducing delays in Rt control: The control action interval," *IFAC Proc. Volumes*, vol. 32, no. 2, pp. 8527–8532, 1999.
[8] J. Zamorano, A. Alonso, and J. A. de la Puente, "Building safety-critical real-time systems with reusable cyclic executives," *Control Eng. Pract.*, vol. 5, no. 7, pp. 999–1005, Jul. 1997.
[9] T. Henties, J. J. Hunt, D. Locke, K. Nilsen, M. Schoeberl, and J. Vitek, "Java for safety-critical applications," in *2nd Int. Workshop Certification Saf.-Crit. Softw. Controlled Syst. (SafeCert)*, Jan. 2009.
[10] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *Proc. 25th IEEE Int. Real-Time Syst. Symp.*, Dec. 2004, pp. 37–46.
[11] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 12, pp. 1702–1714, Dec. 1999.

[12] V. A. Nguyen, D. Hardy, and I. Puaut, "Cache-conscious off-line real-time scheduling for multi-core platforms: Algorithms and implementation," *Real-Time Syst.*, vol. 55, no. 4, pp. 810–849, Oct. 2019.
[13] Y. Sun and M. D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, pp. 1–19, Oct. 2017.
[14] T. Fleming and A. Burns, "Investigating mixed criticality cyclic executive schedule generation," in *Proc. Workshop Mixed Criticality (WMC)*, 2015, pp. 42–47.
[15] J. Kim, H. Oh, H. Ha, S.-H. Kang, J. Choi, and S. Ha, "An ILP-based worst-case performance analysis technique for distributed real-time embedded systems," in *Proc. Real-Time Syst. Symp.*, Dec. 2012, pp. 363–372.
[16] L. Mangeruca, M. Baleani, A. Ferrari, and A. Sangiovanni-Vincentelli, "Uniprocessor scheduling under precedence constraints for embedded systems design," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 1, pp. 1–30, Dec. 2007.
[17] B. Lisper and P. Mellgren, "Response-time calculation and priority assignment with integer programming methods," in *Proc. Work–Progress Ind. Sessions, 13th Euromicro Conf. Real-Time Syst.*, E. Tovar and C. Norström, Eds. Jun. 2001, pp. 13–16. [Online]. Available: http://www.es.mdh.se/publications/282-
[18] H. Zeng and M. Di Natale, "An efficient formulation of the real-time feasibility region for design optimization," *IEEE Trans. Comput.*, vol. 62, no. 4, pp. 644–661, Apr. 2013.
[19] R. Devaraj, A. Sarkar, and S. Biswas, "Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using supervisory control of timed DES," in *Proc. Amer. Control Conf. (ACC)*, May 2017, pp. 3212–3217.
[20] R. Devaraj, A. Sarkar, and S. Biswas, "Exact task completion time aware real-time scheduling based on supervisory control theory of timed DES," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2018, pp. 1908–1913.
[21] P. K. Harter, "Response times in level-structured systems," *ACM Trans. Comput. Syst.*, vol. 5, no. 3, pp. 232–248, Aug. 1987.
[22] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986.
[23] A. A. Paul and B. A. S. Pillai, "Reducing the number of context switches in real time systems," in *Proc. Int. Conf. Process Autom., Control Comput.*, Jul. 2011, pp. 1–6.
[24] P. R. Nuth and W. J. Dally, "A mechanism for efficient context switching," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Processors*, Oct. 1991, pp. 301–304.
[25] S. K. Baruah, R. R. Howell, and L. E. Rosier, "Feasibility problems for recurring tasks on one processor," *Theor. Comput. Sci.*, vol. 118, no. 1, pp. 3–20, Sep. 1993.
[26] M. F. Tompkins, "Optimization techniques for task allocation and scheduling in distributed multi-agent operations," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2003.
[27] G. Ausiello, V. Bonifaci, and B. Escoffier, "Complexity and approximation in reoptimization," in *Computability in Context*. 2011, pp. 101–129. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9781848162778_0004, doi: 10.1142/9781848162778_0004.
[28] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Perform. Eval.*, vol. 2, no. 4, pp. 237–250, Dec. 1982.
[29] A. Crespo, A. Alonso, M. Marcos, J. A. de la Puente, and P. Balbastre, "Mixed criticality in control systems," *IFAC Proc. Volumes*, vol. 47, no. 3, pp. 12261–12271, 2014.
[30] *Gurobi Optimizer Reference Manual*, Gurobi Optimization, Houston, TX, USA, 2019.
[31] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 398–409.
[32] A. Cervin, "Improved scheduling of control tasks," in *Proc. 11th Euromicro Conf. Real-Time Syst. (Euromicro)*, Jun. 1999, pp. 4–10.
[33] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, "A compositional framework for avionics (ARINC-653) systems," Dept. Comput. Inf. Sci., Univ. Pennsylvania, Philadelphia, PA, USA, Tech. Rep. MS-CIS-09-04, 2009.
[34] A. K. Mok and X. Alex, "Towards compositionality in real-time resource partitioning based on regularity bounds," in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2001, pp. 129–138.
[35] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proc. Int. Symp. Syst.-on-Chip*, Dec. 2003, pp. 2–13.
[36] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 129–138.

ANA GUASQUE was born in Valencia, Spain, in 1987. She received the B.S. degree in industrial engineering, the M.S. degree in automation and industrial computing, and the Ph.D. degree in industrial engineering from the Universitat Politècnica de València (UPV), in 2013, 2015, and 2019, respectively.

She is currently working as a Researcher with UPV. Her main research interests include real-time operating systems, scheduling, and optimization algorithms and real-time control.

HOSSEIN TOHIDI was born in Tehran, Iran, in 1985. He received the B.S. degree in industrial engineering from the Iran University of Science and Technology, Tehran, in 2008, the M.S. degree in operations research from the University of Tehran, Tehran, in 2013, and the M.S. degree in engineering management from the University of Minnesota Duluth, Duluth, MN, USA, in 2016. He is currently pursuing the Ph.D. degree in industrial and systems engineering with North Carolina State University, Raleigh, NC, USA.

His research interests include stochastic optimization, large-scale optimization, network optimization, reinforcement learning, and machine learning.

PATRICIA BALBASTRE received the degree in electronic engineering from the Universitat Politècnica de València (UPV), in 1998, and the Ph.D. degree in computer science, in 2002. She is currently an Associate Professor of Computer Engineering with UPV. Her main research interests include real-time operating systems, dynamic scheduling algorithms, and real-time control.

JOSÉ MARÍA ACEITUNO was born in Valencia, Spain, in 1982. He received the B.S. degree in computer management from the University of Castellón, in 2012, and the M.S. degree in artificial intelligence from the Universitat Politècnica de València (UPV), in 2016, where he is currently pursuing the Ph.D. degree in distributed systems.

From 2016 to 2019, he was a Teacher of High-Level Web Applications and Multiplatforms at ILERNA Online, Spain.

JOSÉ SIMÓ received the M.S. degree in industrial engineering and the Ph.D. degree in computer science from the Universitat Politècnica de València (UPV), Spain, in 1990 and 1997, respectively. Since 1990, he has been involved in several Spanish and European research projects mainly related to real-time and embedded systems and industrial collaborations. He is currently a Professor with the Department of Computer Engineering, UPV. His current research interests include the development of real-time embedded systems, autonomous systems, and robotics.

ALFONS CRESPO received the Ph.D. degree in computer science from the Universitat Politècnica de València (UPV), in 1984. He is currently a Professor with the Department of Computer Engineering, UPV. He became an Associate Professor, in 1986, and a Full Professor, in 1991. He leads the Industrial Informatics Group and has also led several European and Spanish research projects. He has published more than 60 papers in specialized journals and conferences in the area of real-time systems. His main research interest includes aspects of the real-time systems, including scheduling, hardware support, and scheduling and control integration.