Journal of Systems Architecture 130 (2022) 102671

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Partition window assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows

Andoni Amurrio [a,b,*], J. Javier Gutiérrez [b], Mario Aldea [b], Ekain Azketa [a]

[a] IKERLAN Research Centre - Basque Research & Technology Alliance, Arrasate, Spain
[b] Software Engineering and Real-Time Group, Universidad de Cantabria, Santander, Spain

ARTICLE INFO

MSC:
00-01
99-00

Keywords:
Real-time
Partition windows
Schedulability
Time-partitioning
Multipath flows

ABSTRACT

Time and space partitioning techniques are implemented in the development of safety-critical applications to ensure isolation among components. A suitable arrangement of the execution of such partitions is a key challenge so that applications meet the timing requirements imposed to software. In this work, the effect of window sizes and context switch overheads in the partition window configuration is studied, with the aim of analyzing their impact when the response-time analysis and priority assignment techniques are applied. Then, a heuristic algorithm is proposed, in order to obtain a partition window configuration that enables the schedulability of partition-based safety critical systems. This algorithm is evaluated in synthetic test scenarios and it is also applied to a safety-critical use-case in the railway domain.

1. Introduction

1.1. Context and objective

Modern cyber–physical systems must meet both sophisticated functional and non-functional requirements, which leads to the increase of their complexity. Railway companies are following the path marked out by the avionics domain [1], whose architectural designs have evolved towards time and space partitioned models [2,3], where applications can execute in isolation. Each partition can have distinct non-functional requirements to guarantee response times, safety, confidentiality, etc., which make up the so-called mixed-criticality systems [4]. In [5], a detailed review of the work done in the last decade on this type of systems is carried out, ranging from the most theoretical scheduling or design aspects, to the basic implementation mechanisms. In mixed-criticality systems it is necessary to have total independence among partitions with the aim that the processes of specification, design, implementation, safety certification (in those systems that require it) and execution are totally independent throughout the different system components [6–9]. In the European project MultiPARTES [10] for example, a set of tools was proposed for the development of mixed criticality systems based on partitioning, from hardware and software architectures to partition management tools.

The ARINC 653 standard defines temporal partitioning, where partitions are executed periodically. Each partition can be composed of one or more partition windows, which are in essence time intervals during which processing resources are assigned for the execution of a certain partition. The standard defines a hierarchical scheduling policy, where partitions are executed sequentially in a pre-defined order within a major frame (MAF) and, within each partition, there is a secondary scheduler based on preemptive fixed priorities. The scheduling optimization of such systems has traditionally been solved at the partition level, by considering synchronous approaches where tasks' activations are referred to the beginning of the MAF. Therefore, the MAF is usually defined as the least common multiple of tasks' activation periods (also known as hyper-period), and most solutions rely on obtaining a table-driven scheduler at this partition level designed to fit each task in its corresponding partition. However, in this work we consider hierarchical scheduling as well as tasks that do not have to be necessarily synchronized with the MAF (events can arrive at any time within the MAF), making the worst-case response time analysis and optimization non-trivial. Hence, we aim at proposing an efficient algorithm to design the scheduling of tasks by combining partition window assignment stages with priority assignment heuristics, in such a way that all tasks can meet their timing requirements.

Safety and application-specific standards mandate adherence to complex architectural patterns. A paradigmatic example can be found in the railway domain too: in order to maintain a certain integrity level, manufacturers implement redundant architectures where processing activities are performed in separated instances and their results are voted [11,12]. From a real-time perspective, these architectures are

* Corresponding author at: IKERLAN Research Centre - Basque Research & Technology Alliance, Arrasate, Spain.
E-mail address: aamurrio@ikerlan.es (A. Amurrio).

https://doi.org/10.1016/j.sysarc.2022.102671
Received 21 January 2022; Received in revised form 23 May 2022; Accepted 14 July 2022
Available online 20 July 2022

modeled as multipath flows composed of fork and join structures that enable complex execution flows to be modeled. These architectures can also be analyzed and optimized through the priority assignment in each partition as presented in [13,14]. Correct functioning of safety critical applications depends not only on results being right, but also on them being produced in time. It is usual that timing requirements are imposed to software, which must not be exceeded even in the worst case scenario, and therefore adequately setting all the schedule-related parameters gains paramount importance.

As a step forward on re-factoring railway signaling applications, the aim of this work is to define an adequate partition window configuration that enables meeting the deadlines imposed to a partition-based safety critical application. To tackle this complex problem, we aim to perform a study on the effect of the size and number of partition windows in the schedulability of this kind of applications, and based on the knowledge acquired, a heuristic partition window assignment algorithm is developed. This algorithm will be characterized by means of a set of synthetic experiments, and then it will be applied to a railway use-case that motivates this work.

1.2. Related work

Research works addressing the scheduling optimization of time-partitioned real-time system have recently been attracting growing interest. Some of them make use of meta-heuristic approaches, such as [15] where an evolutionary algorithm [16] is implemented in the field of aerospace domain to create hierarchical schedulers, applied to multi-core architectures. They consider implicit-deadline independent tasks, in contrast to the multipath e2e flow model composed of tasks with arbitrary deadlines considered in this work. In [17,18] the authors propose a Tabu Search algorithm [19] to allocate tasks to partitions and to generate table-driven partition schedulers, with the aim of minimizing several aspects such as the certification costs. However, applications are modeled as a set of tasks with precedence relationships that are not allowed to cross different partitions, which is a key feature of safety critical applications, represented by the industrial use-case addressed in this work. There are other approaches that implement dedicated heuristic algorithms to tackle the partition scheduling problem. For instance, in [20] partitions are scheduled through a MILP method [21], which is later refined by means of a Game Theory algorithm [22]. However, their system model does not address any tasking model, as the work is limited to partition scheduling. This is also the case of [23], where partitions are allocated to processors by performing partial analysis in each stage of the algorithm proposed by the authors. Finally, in [24] tasks are allocated to partitions and then a TDMA partition schedule is created. This work is the only one so far that considers context switch overheads at partition level. However, their application model is based on independent tasks with implicit deadlines.

1.3. Manuscript organization

The paper is organized as follows. We address a real-time scheduling problem whose main features have been described in Section 1. The system model, the schedulability analysis technique and priority assignment algorithms used in this work are described in Section 2, and a study of the effect that the size and number of partition windows has in the schedulability of hierarchically scheduled distributed real-time systems is performed in Section 3. Then, Section 4 describes the proposed heuristic algorithm that leverages the knowledge of the prior study, and in Section 5, the experimental evaluation is carried out by characterizing the algorithm through synthetic experiments and also by applying it to a railway use-case, which is also described in this section. Finally, Section 6 draws some conclusions and discusses future research lines.

2. Modeling, analysis and priority assignment

2.1. System model

Fig. 1 shows a simple model containing the elements that are used to describe the systems addressed in this work. The main element is the end-to-end (e2e) flow, which consists of a sequence of activities with precedence relations executed in response to a periodic or sporadic workload event, with a minimum inter-arrival time (T_i). The main component of an e2e flow is the event handler called *step*, which represents an operation being executed by a schedulable resource (a task or a message) in a processing resource (a computer or a network). Each step is activated by an input event, and after its execution it generates an output event. The jth step in the e2e flow Γ_i is denoted as τ_{ij}, and it has a worst-case and a best-case execution time, C_{ij} and C^b_{ij} respectively. In each e2e flow, steps are numbered in topological order in the range $[1..N_i]$. Workload events that activate e2e flows and also the internal events that activate handlers may exhibit a release jitter, so any step τ_{ij} may suffer a release jitter up to a maximum of J_{ij}. Steps can also have an initial offset Φ_{ij}, which is the minimum release time of the step relative to the nominal activation instant of the workload event, i.e. the event that activates the first step of the e2e flow. In Fig. 1, a workload event e_{in1}, which is represented by a down-pointing arrow, activates a step ($\tau_{1\ 1}$) and then its output event forks, activating two steps, whose output events combine to activate a final step ($\tau_{1\ 4}$). Horizontal blue arrows represent precedence relations among event handlers.

The response time of an instance of a step is the difference between its completion time and the nominal activation time of the workload event that triggered that instance of the e2e flow (which matches the activation of the first step of the e2e flow). The worst-case response time is denoted as R_{ij} and the best-case response time as R^b_{ij}, and both are obtained by schedulability analysis techniques. As mentioned before, deadlines are imposed to software in order to guarantee that applications complete their duties within a bounded time. In this paper, we only consider e2e deadlines, denoted as D_{ij}, which are the deadlines set at the output steps of e2e flows. Each step represents a utilization of the processing resource of $U_{ij} = C_{ij}/T_i$.

This model includes other event handlers that do not have runtime effects and enable the modeling of complex event combinations like the multipath e2e flows addressed here:

- Fork: It generates one event in each of its outputs each time an input event arrives.
- Join: It generates an output event when all of its input events have arrived.

In this work, we address a distributed architecture, where different processors, which provide hardware and software resources for task execution, are connected through one or more communication networks. It is assumed, without loss of generality, that the minimum and maximum latencies that messages undergo at network level can be obtained. Scheduling communications traffic is beyond the scope of this paper and it remains as future work. Processors host a real-time operating system that enables temporal partitioning, such as Integrity RTOS [25]. Therefore, hierarchically scheduled and time-partitioned systems are considered in this work.

Hierarchical schedulers are composed of a primary scheduler and a secondary scheduler. A table-driven scheduling policy is considered as primary scheduler in every processor, where temporal partitions are scheduled in a cyclic manner within a Major Frame (MAF). We define MAF_y as the MAF of CPU_y. A temporal partition P_x is composed of one or more partition windows Win_{xk}, defined as follows: $Win_{xk} = \{ S_{xk}, L_{xk} \}$ where S_{xk} is the start time relative to the start of the MAF, and L_{xk} is its length. The secondary scheduler is based on preemptive fixed priorities, where $Prio_{ij}$ is the priority of the step τ_{ij}, and where the

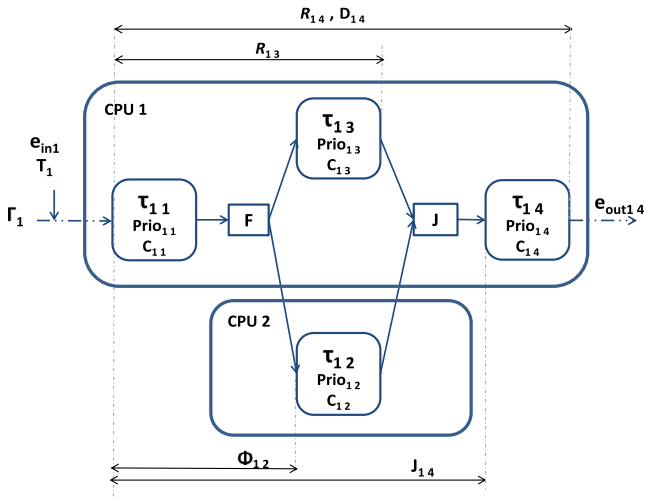

Fig. 1. Distributed multipath e2e flow. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

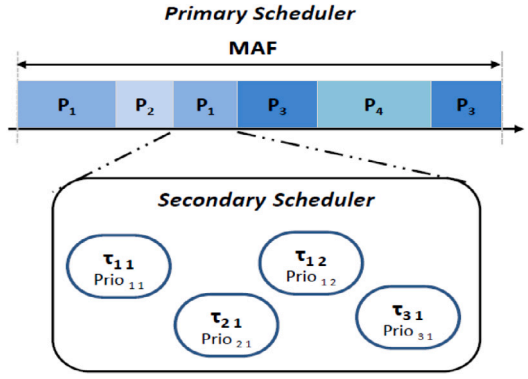

Fig. 2. Example of hierarchical scheduler.

highest number represents the highest priority. Priorities are valid in the context of each partition. Fig. 2 shows an example of a hierarchical scheduler composed of four temporal partitions, P_1 to P_4, where P_1 and P_3 are composed of two partition windows. Within Partition 1 there are four steps, executed according to their priorities by the secondary scheduler.

We define the Available Utilization of the partition P_x, AU_{P_x}, as the processing time allocated to P_x in its processor CPU_y, which is in essence, following the terminology just presented, the sum of the utilization of all the temporal windows within the MAF, so:

$$AU_{P_x} = \sum_{\forall Win_{xk} \in P_x} L_{xk}/MAF_y \quad (1)$$

Following the description given for the utilization represented by each step, the Partition Utilization of P_x, U_{P_x} is defined as the sum of the utilization of all the steps contained in P_x:

$$U_{P_x} = \sum_{\forall \tau_{ij} \in P_x} U_{ij} \quad (2)$$

The overheads provoked by context switches at the primary scheduler are taken into account. This overhead is the time CS_y that CPU_y needs to load a partition context at the beginning of a partition window and to save it after execution finishes. In other words, it can be understood as a non-available CPU time whenever a partition window executes. For response-time analysis purposes, this effect can be modeled by recording this unavailable time at the beginning of every partition window and subtracting this amount from the available CPU-time for that window, as shown in the example in Fig. 3, where P_x

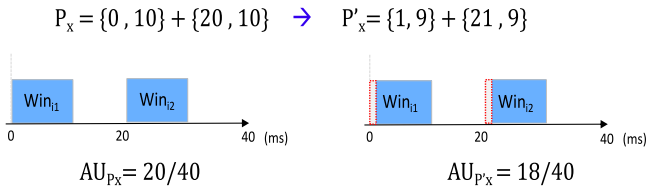

Fig. 3. Partition and effective partition with $CS_y = 1$ ms.

executes on CPU_y within a $MAF_y = 40$ ms. In this example, a time partition is composed of two partition windows, and the effect of the context switch time of CS_y=1 ms at each window provokes that the total available time of the effective partition (P'_x) is 2 ms less than the original P_x. Therefore, the effective partition window is defined as follows: $Win'_{xk} = \{ S_{xk} + CS_y, L_{xk} - CS_y \}$.

The system model derived in this work complies with the MAST model [26]. MAST (Modeling and Analysis Suite for Real-Time Applications) is a GPL open source model and also a set of tools developed by the University of Cantabria, which is aligned with the OMG's MARTE standard [27]. It enables the description of the temporal behavior of computing systems, and includes a bunch of scheduling parameter assignment algorithms, different schedulability analysis techniques and simulation tools. The second version of its metamodel, MAST2 [28], adds several novel scheduling policies and modeling elements, such as time partitioning.

2.2. Response-time analysis and priority assignment

In order to conduct the experimental evaluation of this work, we need to calculate the steps' worst-case response times. To do so, we will make use of the technique developed in [13]. This is an offset-based schedulability analysis technique [29,30], which was extended to support multipath e2e flows. It is the most accurate technique available to calculate worst-case response times, as it is demonstrated that the results of the holistic analysis [31] for multipath flows are notably improved. Readers are encouraged to read the aforementioned references for a deeper understanding of the schedulability analysis.

As said before, the secondary scheduler is based on the FP policy, and therefore several priority assignment algorithms can be used within the proposed partition window assignment algorithm. We will rely on the priority assignment techniques developed in [14], which can be applied to multipath e2e flows and time-partitioned distributed real-time systems. These algorithms produce a single solution by distributing the end-to-end deadline through all the steps in the flow (called Virtual Deadlines or VDs), and then transforming such deadlines into priorities following the deadline monotonic algorithm. The main conclusion in [14] is that there is no algorithm that stands out from the others in the tested scenarios, which reinforces the idea of evaluating all of them and choosing the one that produces the best solution. Here is a brief introduction to each algorithm:

- Ultimate Deadline (UD): It is the simplest scheduling-parameter assignment algorithm, where the e2e deadline is assigned to all steps composing the e2e flow [32].
- Effective Deadline (ED): The VD of a step according to the ED algorithm is the e2e deadline minus the sum of the worst-case execution times of its successor steps [32].
- Proportional Deadline (PD): The e2e deadline is distributed among all the steps in the flow proportionally to their worst-case execution times and the sum of the worst-case execution times of all the steps of the flow [32]. If the produced VDs refer to the activation event of the e2e-flow, they are called global deadlines, and if they refer to the activation of each step, they are called local deadlines. Therefore, two variants of the PD algorithm are distinguished: PD_Global and PD_Local.

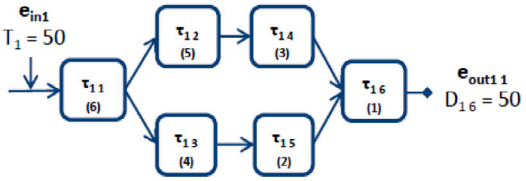

Fig. 4. Guiding application example.

- Normalized Proportional Deadline (NPD): This algorithm is similar to PD, but it also considers the utilization of the processing element where it is hosted [32]. In this case the global and local versions are also distinguished: NPD_Global and NPD_Local.
- Equal Slack (EQS): This algorithm was proposed for on-line deadline assignment in soft real-time distributed systems, based on EDF schedulers [33]. Deadline assignment is performed by equally dividing the slack, defined as the difference between the deadline and the worst-case response time.
- Equal Flexibility (EQF): This algorithm was also originally proposed for on-line EDF scheduling [33], and it is also based on dividing the slack (considering the previously given definition), while the proportionality with respect to the execution times of the steps is maintained. To do so, the concept flexibility is defined as the ratio between the slack and the worst-case response time.

Readers are encouraged to read the aforementioned references for further details on their design and implementation.

3. Study of the influence of partition windows in schedulability

In this section, a study of the influence of the of size and number of partition windows on the worst-case response times is presented, which was performed in [34] and which serves as the basis for the development of the heuristic partition window assignment algorithm presented in Section 4. To perform the proposed study, different partition scheduling schemes will be evaluated. Thus, a simple application example is built, which includes the most relevant features that characterize the motivating railway use-case. This example is composed of a single multipath e2e flow activated periodically every 50 ms which is composed of six steps ($\tau_{1\,1}$ to $\tau_{1\,6}$), as shown in Fig. 4. The e2e flow is mapped within a single partition (P_x) composed of a single partition window, and it is assumed, for the sake of simplicity, that the worst-case execution time of each step is fixed and equal 2 ms. The priority of each step is shown in brackets, and the MAF considered for the whole experiment set is 50 ms. When referring to the schedulability of the application, it relates to the worst-case response time of $\tau_{1\,6}$ ($R_{1\,6}$) in comparison with its deadline.

3.1. Available utilization

A very common early-design decision regards the CPU time allocated for the execution of each partition, i.e. AU_{P_x} in the example described here. Depending on this time, response times may vary significantly as shown in Fig. 5. Even if this effect seems obvious, it gives us an idea about the effect that not having all the processor time dedicated for the execution of applications produces on the worst-case response times calculated by the analysis technique. For a fixed MAF and considering a fixed number of windows, a lower available utilization generates longer gaps between partition windows, thus the longer these gaps are (where P_x is not allowed to be executed), the higher is the worst-case response time. In this example the partition utilization U_{P_x} represents 24% of the CPU time, and worst-case response times vary from 580 ms to 12 ms when the available CPU goes from the initial value of 24% to 100% respectively.

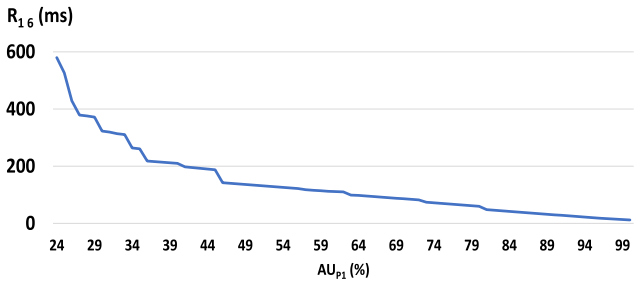

Fig. 5. Worst-case response time of $\tau_{1\,6}$ as a function of AU_{P_x}.

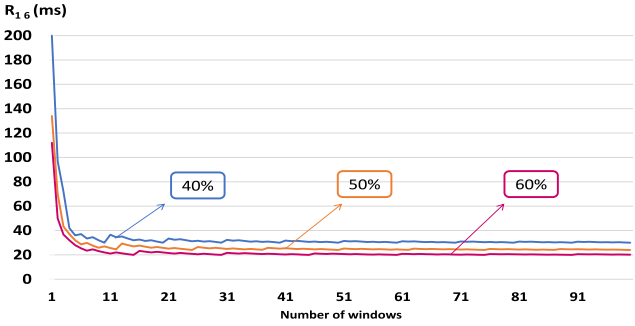

Fig. 6. Evolution of worst-case response time of $\tau_{1\,6}$ with the number of partition windows for different values of AU_{P_x} (in %).

3.2. Number of windows

Once the available time has been fixed for a partition, the next design decision to take is to distribute this given time in the MAF. As a first approach, a uniform distribution of partition windows will be considered, although this might be subject to optimization when more partitions make up the MAF. Fig. 6 shows the worst-case response times obtained for this example when the number of partition windows varies from 1 to 100, for three different values of AU_{P_x}. As can be seen, increasing the number of windows produces a reduction in the length of the unavailable gaps in the MAF, leading to a remarkable reduction of the obtained response times, up to a point where this reduction is no longer significant. Again, we can observe here how having larger gaps between windows leads to higher response times.

3.3. Context switch overheads

The experiments modeled in the previous section have not considered the effects of the context switch overheads that are present when a partition is activated. These context switch overheads have been modeled in the previous section. In general, they depend on the operating system/hypervisor where applications are executed. For instance, in [35] the measured context switch overhead in a hypervisor is 17 µs, and in [36], it is 27 µs. Therefore, in the experiments performed in this paper, a value of $CS = 20$ µs is considered, along with a higher order of magnitude representing a slower processor, i.e. $CS = 200$ µs. Formally, the maximum CPU time that can be spent in context switch overheads in processor CPU_y, is bounded by the difference between the available partition utilization (AU_{P_x}) and the partition utilization (U_{P_x}). Considering this, the limit of the number of partition windows that can be set for partition P_x without overloading the partition is defined as follows:

$$NW_{P_x} = \lfloor (AU_{P_x} - U_{P_x}) \cdot \frac{MAF_y}{CS_y} \rfloor \quad (3)$$

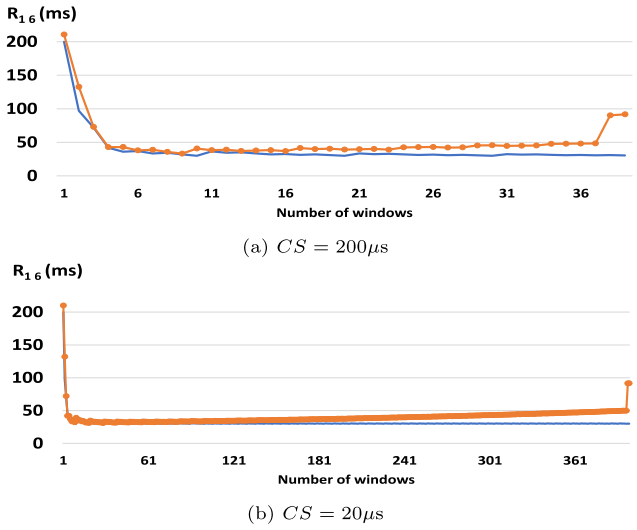

Fig. 7. Worst-case response time as a function of the number of partition windows — $AU_{P_x} = 40\%$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

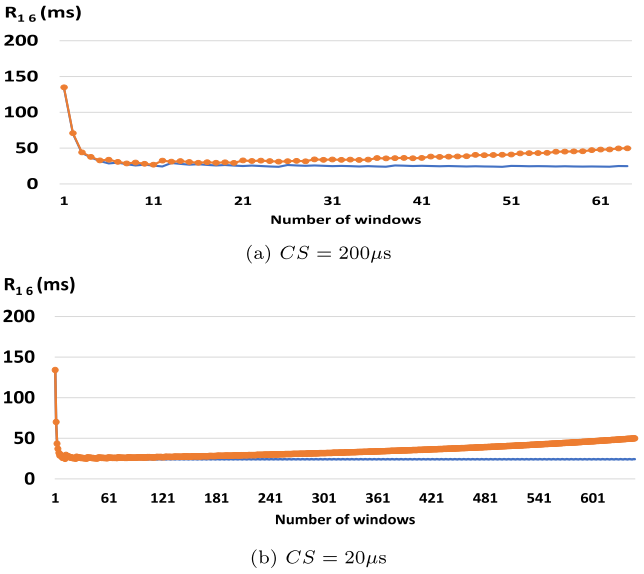

Fig. 8. Worst-case response time as a function of the number of partition windows — $AU_{P_x} = 50\%$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In other words, with the number obtained by this expression or a lower value of windows, enough time for the execution of partition P_x is guaranteed.

Figs. 7 to 9 show the worst-case response times obtained for different numbers of partition windows, available utilizations and context switch overhead values for this guiding example. For different AU_{P_x} values we calculate the response times when increasing the number of partition windows in the range 1 to NW_{P_x}. Qualitatively, worst-case response times vary in the same way regardless of the CPU availability, i.e. the maximum response times are obtained when P_x is scheduled in a single window, and as the number of windows increases response times reduce fast, up to a point. After that point, notice that in the ideal scenario, $CS_y = 0$ (blue plot), the response-time curve remains constant, while if $CS_y > 0$ (orange plot) the curve increases again.

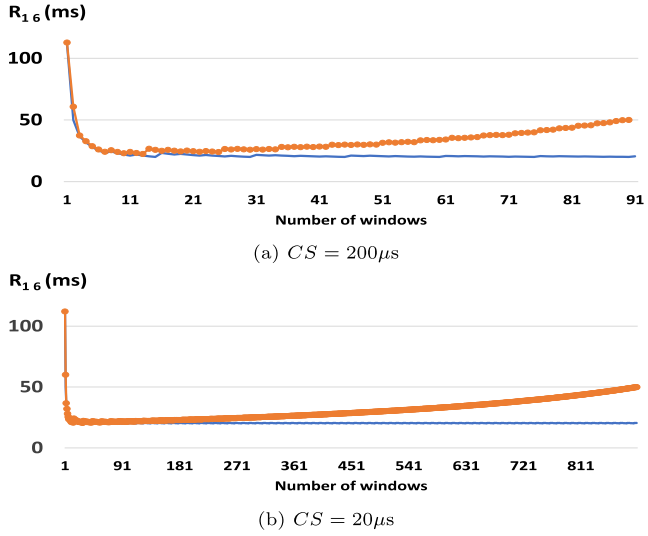

Fig. 9. Worst-case response time as a function of the number of partition windows — $AU_{P_x} = 60\%$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.4. Conclusions of the study

It has been observed that increasing the number of partition windows has a positive effect on the reduction of response times, which is outweighed by the negative effect of context switch overheads. Finding the window configuration that produces the turning point in response times is essential for a partition window optimization algorithm.

When the partition utilization is very low in comparison to the available utilization, and also when context switch overheads are very low, the range of windows to explore ($1..NW_x$ for P_x) is very large. It has been found that the schedulability analysis tool requires a long execution time to analyze systems with large numbers of windows ($NW_x > 500$). Therefore, the strategy to follow will be to start to explore from a minimum number of windows and then increase the number until the turning point is found.

4. Window Assignment (WinAs) algorithm

In the previous study, the effect of the configuration parameters (referred to the available utilization and the number of windows within the MAF) on the worst-case response times has been analyzed. Now, an algorithm called Window Assignment (WinAs) is proposed in order to leverage this knowledge, by searching for a partition window configuration that produces a schedulable solution for a fixed available utilization in each partition.

The WinAs algorithm tries to find a schedulable system configuration by increasing the number of windows for a fixed available utilization assigned to each partition. It produces, in each processor, an adjusted MAF where there is one partition window for each partition, as well as a priority assignment. This algorithm exploits the results of the study presented in the previous section, where the influence of the number of partition windows within the MAF and the gap between these windows on the worst-case response times is shown.

The rationale for the design of the algorithm is to gradually increment the number of partition windows, by always having a single window per partition within a diminishing MAF. Reducing the MAF is equivalent to increasing the number of windows, assuming a uniform window distribution, within a fixed MAF. For example: considering a 100 ms MAF, the window for a certain partition P_1 whose AU_{P1} is 40% will first be defined as follows: $Win_{11} = \{0, 40\}$. In order to increase the number of windows (assuming that the solution was not

schedulable), WinAs sets the new MAF in that processor to 50 ms. Therefore, the new partition window will be defined as: $Win_{11} = \{0, 20\}$, which is equivalent to dividing the previous 40 ms window into two uniformly distributed 20 ms windows within the prior 100 ms MAF. Performing a non-uniform window assignment remains as future research work, bearing in mind that, as noticed in the previous study, worst-case response times are highly influenced by the longest gap between windows within the MAF.

The design of the algorithm is described in Algorithm 1. The first step of the algorithm consists of calculating the initial MAF values (line 2). It is calculated, in the context of each processor, as follows: the value of the MAF will be the most restrictive deadline (the lowest value) of all the steps present in that processor. With this choice, we make sure that all partitions will be executed at least once in the time lapse of the most restrictive deadline. In each iteration, a single window is defined for each partition within the MAF (line 8), its length being proportional to the available utilization of that partition, which for partition P_x is calculated as follows:

$$L_{x1} = MAF \cdot AU_{Px} \tag{4}$$

Since the schedulability analysis of each partition is performed independently [13], the worst-case response times do not depend on a specific partition ordering within the MAF. Therefore, the newly defined partition windows (one per partition) are placed one after another in an arbitrary ordering, based on the partition's index.

Algorithm 1 Window Assignment (WinAs) algorithm

```
1:  for each CPU_y do
2:      MAF_y = min(D_ij ∈ CPU_y)
3:  end for
4:  while Stopping criterion not met do
5:      for each CPU_y do
6:          Offset = 0
7:          for each P_x in CPU_y do
8:              Win_x1 = {Offset, L_x1}
9:              Offset = Offset + L_x1
10:         end for
11:     end for
12:     Perform priority assignment and response-time analysis
13:     if Schedulable then
14:         Return SUCCESS
15:     else
16:         for each CPU_y do
17:             MAF_y = MAF_y/Q
18:         end for
19:     end if
20: end while
21: Return FAIL
```

As shown in [14], priorities assigned to the steps within time partitions have a big impact on the system's schedulability. In this step (line 12), all the algorithms described in [14] are evaluated, and the best solution is selected at this stage. To determine the best priority assignment, a figure of merit is proposed for schedulable solutions: for each e2e flow the maximum value R_{ij}/D_{ij} among all output steps is calculated, so that the worst result per e2e flow is captured; then, the average R_{ij}/D_{ij} ratio of all e2e flows is calculated, so the algorithm's result that obtains the lowest average ratio is considered the best solution.

If the system is schedulable, the algorithm succeeds (line 14), whereas if it is not, the MAF is reduced by an adjustable factor (Q in line 17), which should be greater than 1, and Eq. (4) is applied to the single window of each partition in order to maintain the appropriate available utilization. In the previous example, where the rationale of this algorithm has been explained, the reduction factor applied to the MAF was $Q = 2$, as the MAF was reduced from 100 ms to 50 ms. This value will also be used for the performance evaluation in the next section. The adjustment of factor Q constitutes an optimization problem in itself, and it may be addressed in future work.

The stopping condition evaluated in line 4 of Algorithm 1 will be met when all partitions reach the maximum number of windows, which can be calculated directly by Eq. (3). If during this search a schedulable solution is not found, WinAs fails (line 21), and returns the MAF value that produces the highest System Slack Factor (SSF) value among all the explored values, where a single partition window is assigned to each partition according to its available utilization. The System Slack Factor is defined in the MAST toolset[1] in the following way: a Slack Factor (SF) is the factor by which the worst-case execution times of a step or a set of steps may be increased while keeping the system schedulable, or decreased, in order to make the system schedulable. This definition of SF can be applied to different sets of steps, therefore if all the steps that compose the system are considered, it is called System Slack Factor (SSF).

5. Performance evaluation

In this section the algorithm proposed in Section 4 is evaluated. First, the WinAs algorithm is characterized through a set of synthetic experiments, and then it is applied to a railway industrial use-case.

5.1. Design of the synthetic experiments

As seen in the study in Section 4, varying the number of windows within the MAF may be beneficial for some e2e flow's response times and detrimental for others, since the algorithm tries to schedule the most restrictive e2e flows by selecting an appropriate MAF for them. Therefore, the least restrictive ones suffer an increase in their response times, due to the large context switch overheads. As the study of Section 4 has been performed for a single e2e flow allocated to a single partition, it is necessary to analyze the behavior of WinAs, in order to assess its capacity to search for schedulable solutions when different e2e flows, in terms of deadline requirements and/or load, are part of the same target system. Moreover, the effect of considering some e2e flows that are allocated only to a subset of the processors needs to be evaluated, since, as shown in the description of the algorithm, the solution space may be different in the context of each processor.

With all these features in mind, a set of synthetic scenarios will be evaluated. To do so, a baseline synthetic scenario, referenced as **Scn1** from now on and depicted in Fig. 10, has been designed. It is composed of four e2e flows with different extensive deadline requirements, and they are allocated to four processors. Then, some modifications upon this base scenario will be performed, which include modifying the step-to-processor allocation in **Scn2** and **Scn3** (Figs. 11 and 12 respectively) and also modifying their workloads (**Scn4** and **Scn5**). Although the target systems of this work contain multipath e2e flows, only linear e2e flows are considered in these experiments, so that the results are not affected by other effects related to multipath e2e flows. This will provide a deep insight into the behavior of the algorithm.

In **Scn1**, four e2e flows ($\Gamma_1 - \Gamma_4$) are allocated to four processors. The periods of their activation events, which represent a wide range of values, are as follows: $T_1 = 40$ ms, $T_2 = 250$ ms, $T_3 = 750$ ms and $T_4 = 1000$ ms. It is assumed that all their deadlines are equal to their periods, although the response-time analysis technique would support arbitrary ones too. Therefore, Γ_1 is the most restrictive e2e flow and Γ_4 the least restrictive one. Table 1 shows, for each step of each e2e flow, its worst-case execution time. Best-case execution times are assumed to be the same as the worst-case ones.

In **Scn2** steps from Γ_1 are removed from CPU1 and in **Scn3** steps from Γ_2 are also relocated, thus obtaining an unbalanced partition allocation. Due to the MAF selection strategy described in the previous

[1] https://mast.unican.es/

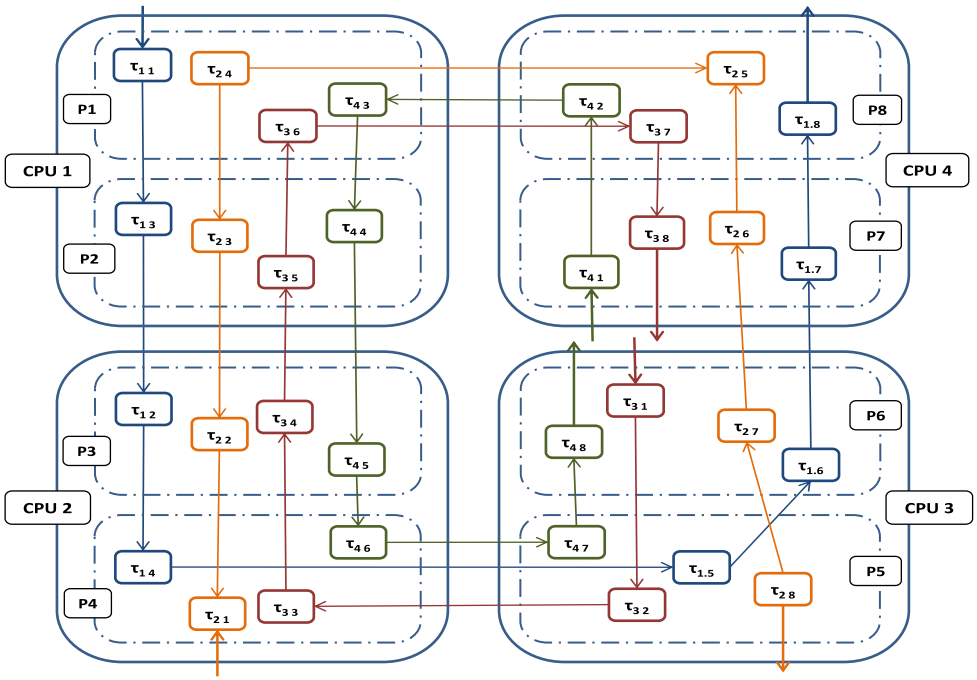

Fig. 10. Baseline synthetic scenario **Scn1**: 2 partitions hosted in each of the 4 processors.

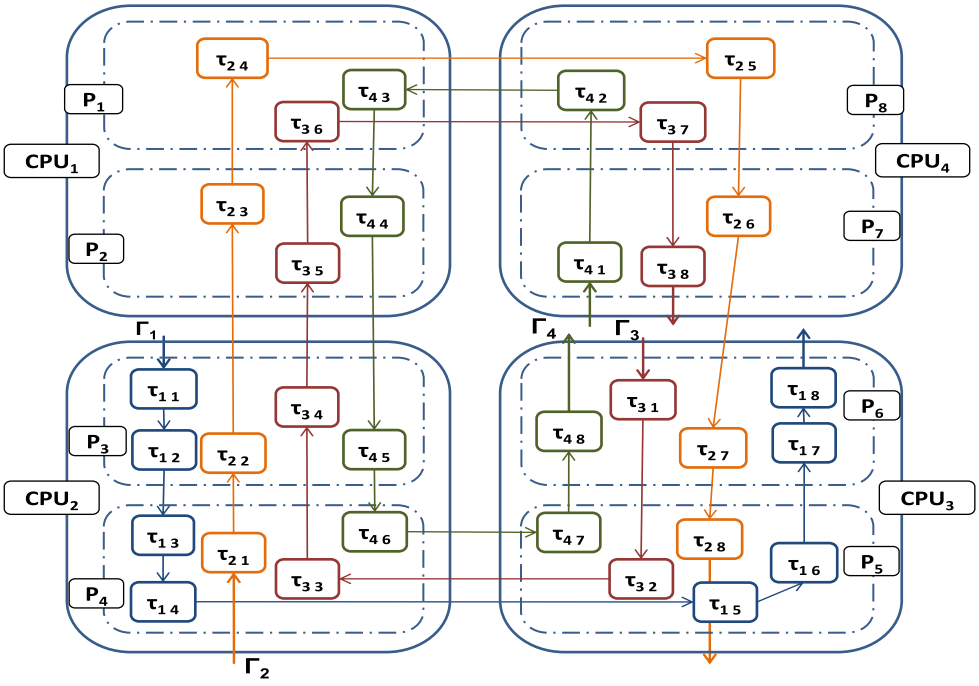

Fig. 11. Scn2: Γ_1 has been removed from CPU_1 and its steps relocated to partitions in the other processors.

Section, the solution space becomes independent in the context of each processor, depending on which is the most deadline-restrictive e2e flow they host. In **Scn4** and **Scn5** a scaling factor of 2 is applied to the worst-case execution times of the steps of the most restrictive and least restrictive e2e flows, Γ_1 and Γ_4 respectively, both considering the architecture of the baseline scenario. Since the algorithm considers the most restrictive flow's deadline to calculate the MAF in each processor, the effect of increasing the workload of Γ_1 has to be analyzed. Similarly, the effect of increasing the load in the least restrictive e2e flow is also important, as it has the least influence in the MAF calculation.

As the utilization of the partitions is in the range of 6% and 17%, the following available utilization for the partitions will be evaluated: 20%, 30%, 40% and 50% (note that 50% is the maximum available utilization possible as there are two partitions per processor). In these experiments, context switch overhead is assumed to be 1 μs.

5.2. WinAs algorithm characterization

After designing the experiments that will be carried out in this section, the WinAs algorithm will be applied on them in order to assess its performance in the different scenarios previously described.

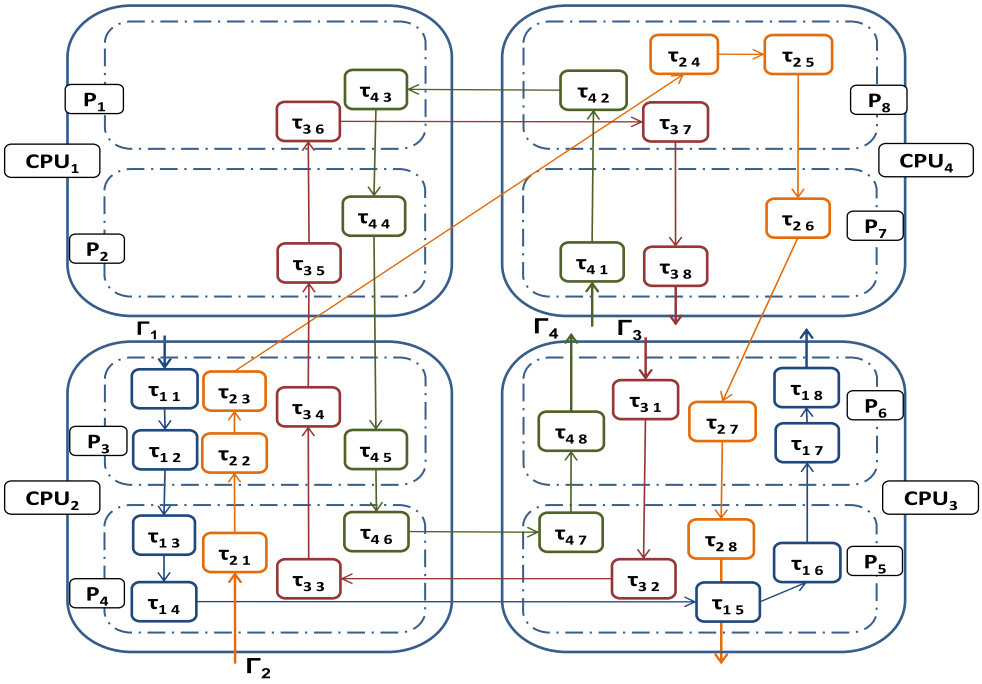

Fig. 12. Scn3: Γ_2 has also been removed from CPU_1 and its steps relocated to partitions in the other processors.

Table 1
Worst-case execution times for Scn1.

Scn1					
e2e	Step	C_{ij}	e2e	Step	C_{ij}
Γ_1	τ_{11}	1	Γ_2	τ_{21}	1
	τ_{12}	0.5		τ_{22}	2
	τ_{13}	0.5		τ_{23}	3
	τ_{14}	1		τ_{24}	2
	τ_{15}	1		τ_{25}	1
	τ_{16}	2		τ_{26}	4
	τ_{17}	2		τ_{27}	2
	τ_{18}	1		τ_{28}	1
Γ_3	τ_{31}	15	Γ_4	τ_{41}	20
	τ_{32}	12		τ_{42}	40
	τ_{33}	20		τ_{43}	6
	τ_{34}	40		τ_{44}	7.5
	τ_{35}	30		τ_{45}	30
	τ_{36}	16		τ_{46}	37.5
	τ_{37}	18		τ_{47}	6.5
	τ_{38}	22		τ_{48}	11

The worst-case response times of each e2e flow, as a function of the number of iterations that WinAs takes, have been plotted in Figs. 13 to 17. In each figure, the e2e flows have been separated into different graphs, and each colored plot represents a different available utilization assigned to each partition. The horizontal red line shows the deadline requirement of each e2e flow, and vertical axes have been represented in logarithmic scale for the sake of clarity. The first schedulable solution obtained by applying WinAs to the synthetic experiments in each scenario can be seen in Table 2. For each available utilization value, we show the MAF produced by WinAs in each processor, as well as the worst-case response time of the output step of each synthetic e2e flow.

Qualitative results are consistent with the ones obtained in the previous experiments, as for all e2e flows, increasing the number of windows is beneficial at the beginning of the search. However, and as was expected, it becomes detrimental (response times increase) earlier for the least restrictive e2e flows. Therefore, the number of window trade-offs must be carefully chosen in order to obtain a schedulable solution. In fact, notice that in Scn1, for 20% and 30% available utilizations, the algorithm fails to obtain a schedulable solution, as response-time curves of Γ_4 are always above the deadline plot. In contrast, when the available utilizations are 40% and 50%, the algorithm does find schedulable solutions.

Regarding Scn2 and Scn3, the proposed algorithm takes more iterations to finish the search. During the first 10 iterations, the number of windows is increased in all processors, while after that, they are increased only in a subset of processors. This is because, in processors where the most restrictive e2e flows are hosted, the limit of the number of windows is reached before it is reached in those where they are not present. However, as shown in Figs. 14 and 15, response times always increase after the 10th iteration, when the number of windows is only increased in a subset of processors. Finally, as shown in Figs. 16 and 17, which correspond to Scn4 and Scn5 respectively, although there are not significant qualitative differences in the behavior of the algorithm when the load of a certain e2e flow is increased, it fails in obtaining schedulable solutions for any available utilization at Scn5, and only when the available utilization is 50% there is a schedulable solution in Scn4. Generally, it can be stated that WinAs's behavior remains qualitatively consistent in all the scenarios described in this section.

5.3. Scheduling evaluation of a railway industrial use-case

The need of train manufacturers to re-factor safety critical applications has motivated the research into novel analysis and scheduling optimization techniques [13,14]. In this section we present the modeling of the real-time railway signaling application according to the system model previously described. Although some information has been omitted because of confidentiality agreements, we show real data provided by the application developers.

The targeted signaling application, depicted in Fig. 18, is in charge of supervising the driving and providing relevant information to the driver [37]. This supervision consists of executing several functionalities, which are activated when the train goes through a balise and receives a message with driving instructions. The event that triggers the execution is represented by the workload event e_{in1}, whose activation period is 1 s, and τ_{11} represents the capturing task. After capturing

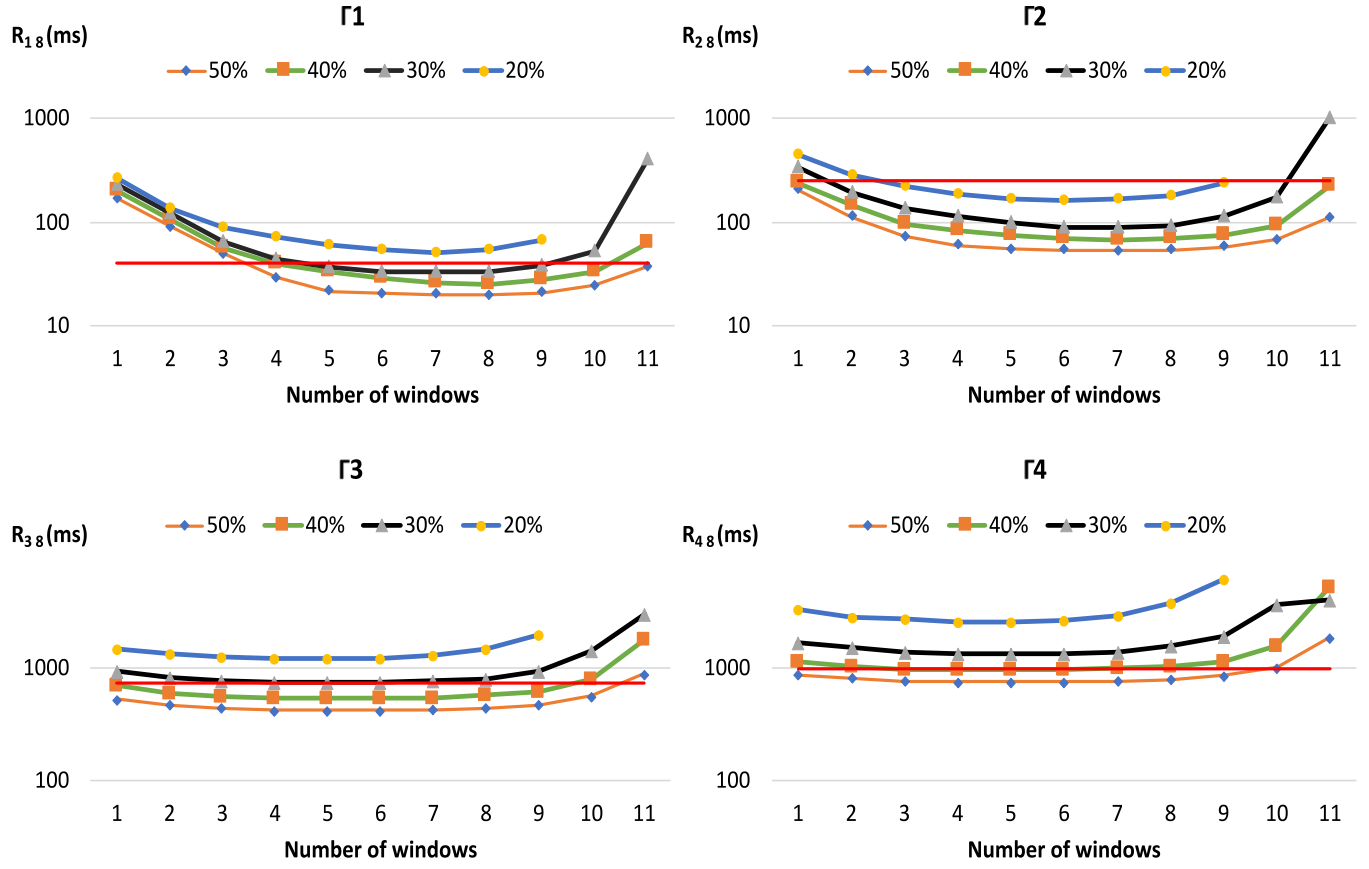

Fig. 13. e2e flows' worst-case response times for different numbers of windows (Scn1). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

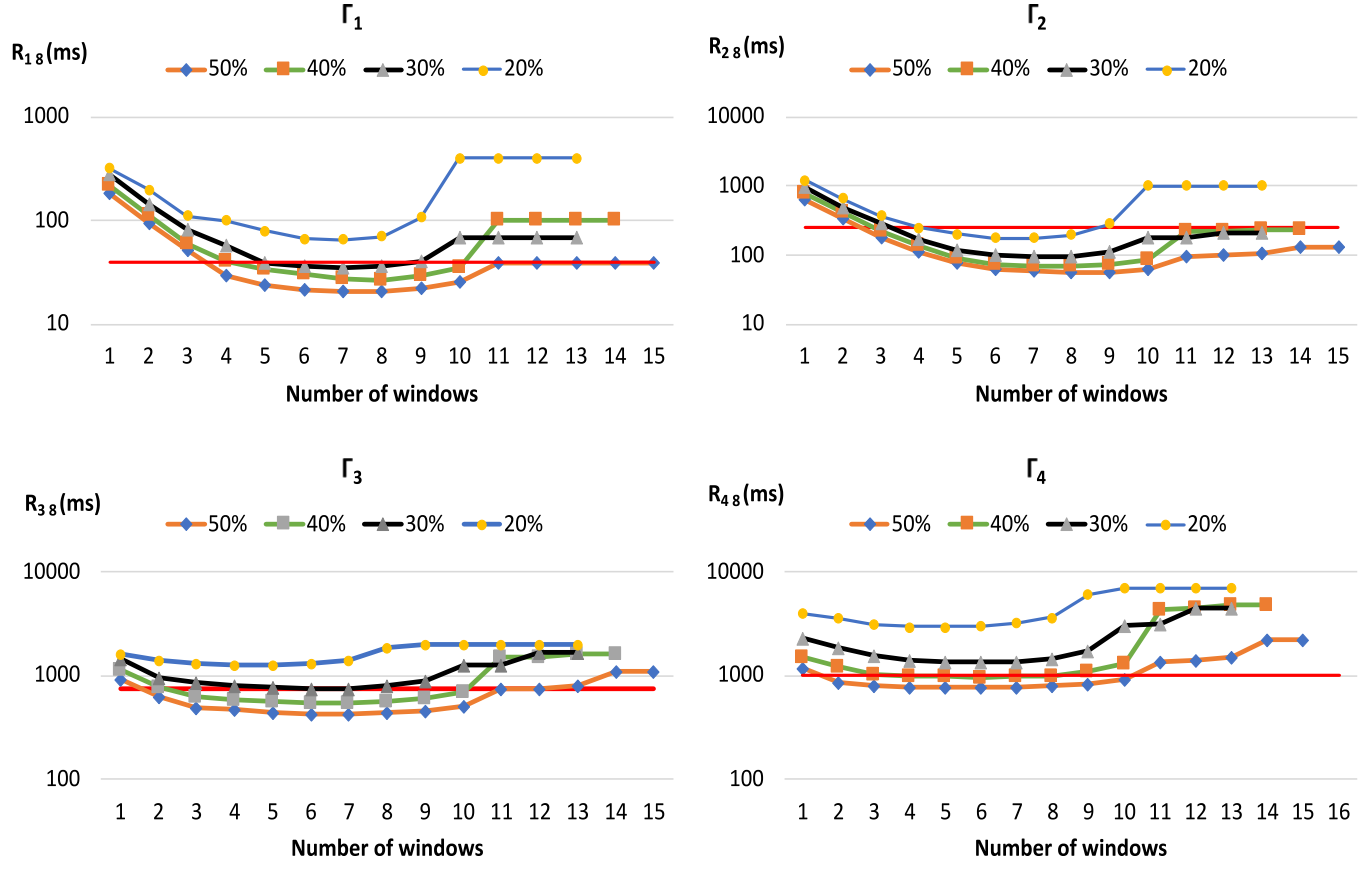

Fig. 14. e2e flows' worst-case response times for different numbers of windows (Scn2). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

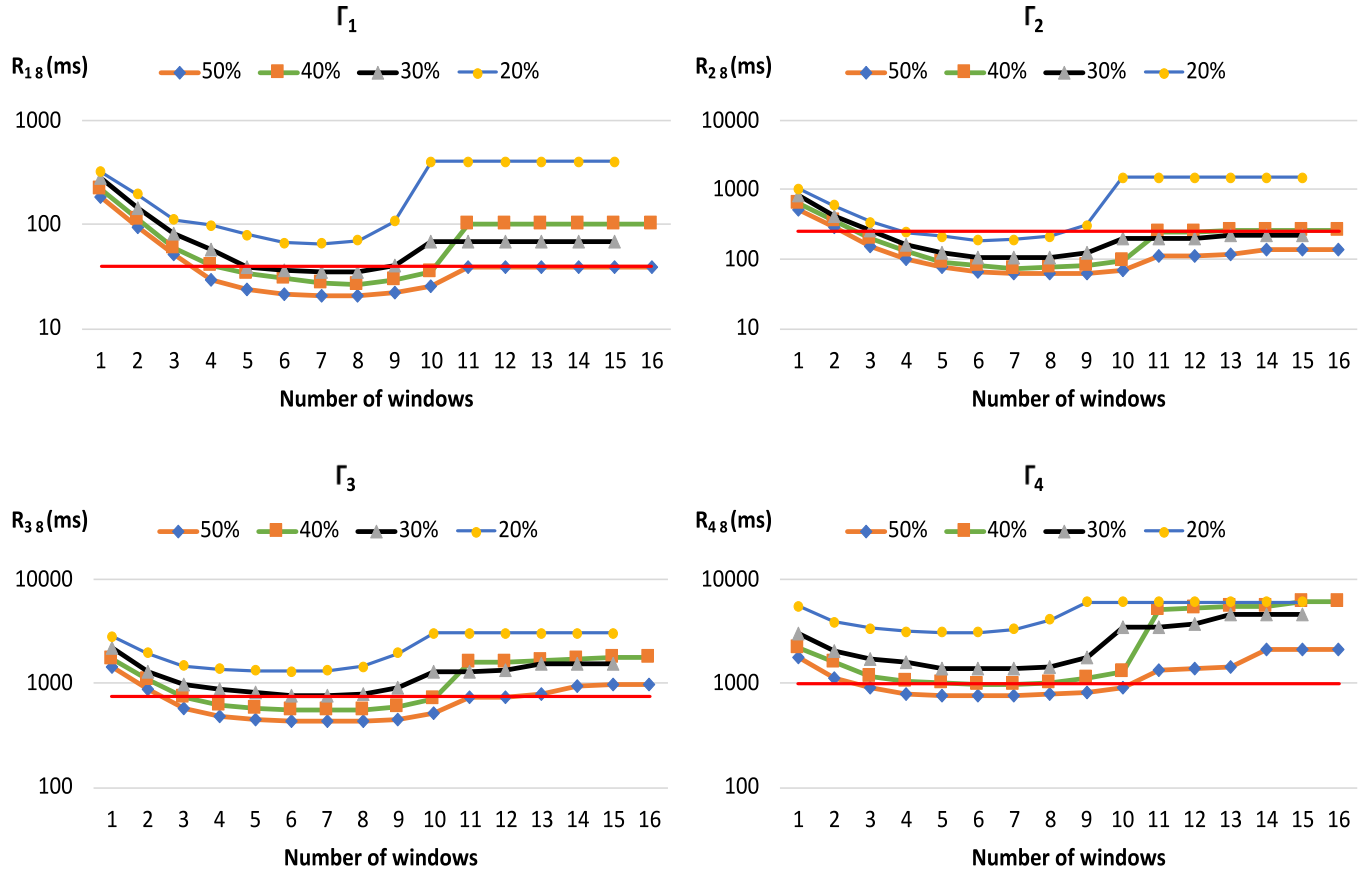

Fig. 15. e2e flows' worst-case response times for different numbers of windows (Scn3). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

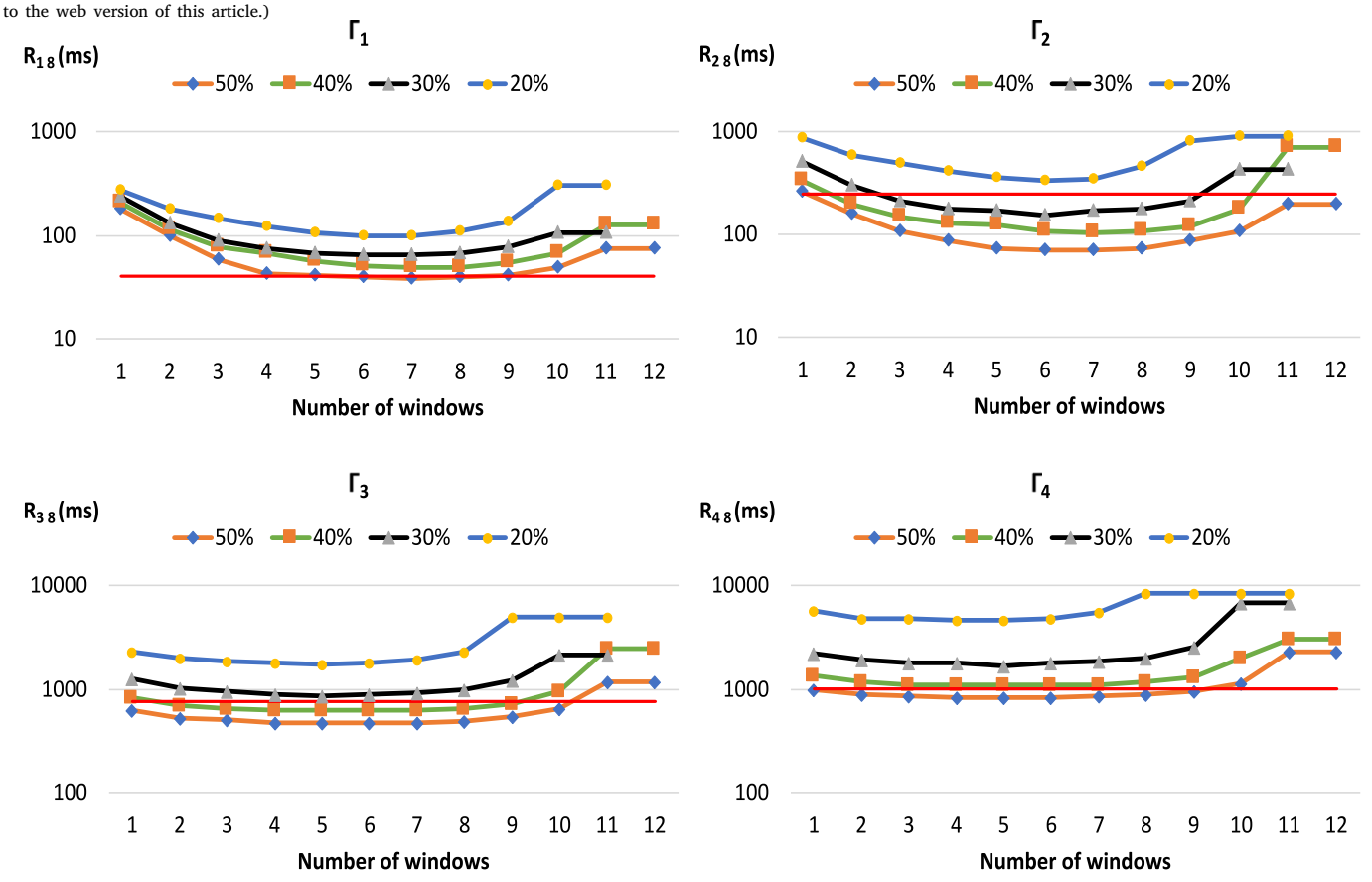

Fig. 16. e2e flows' worst-case response times for different numbers of windows (Scn4). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
Schedulable solutions found by WinAs: MAF and worst-case response times.

AU_{Px}	MAF (ms)				R_{ij} (ms)			
	CPU_1	CPU_2	CPU_3	CPU_4	$R_{1\ 8}$	$R_{2\ 8}$	$R_{3\ 8}$	$R_{4\ 8}$
Scn1								
40%	5	5	5	5	39.1	82.2	545.1	969.4
50%	5	5	5	5	29.08	60.1	423.8	757.03
Scn2								
40%	31.25	5	5	31.25	39.6	134.1	583.2	995.5
50%	31.25	5	5	31.25	29.58	110.1	467.1	773.4
Scn3								
40%	23.43	1.25	1.25	7.81	28.76	78.73	551.9	987.35
50%	93.75	5	5	31.25	29.58	102.01	485.29	795.07
Scn4								
50%	1.25	1.25	1.25	1.25	39.6	70.83	469.4	835.58

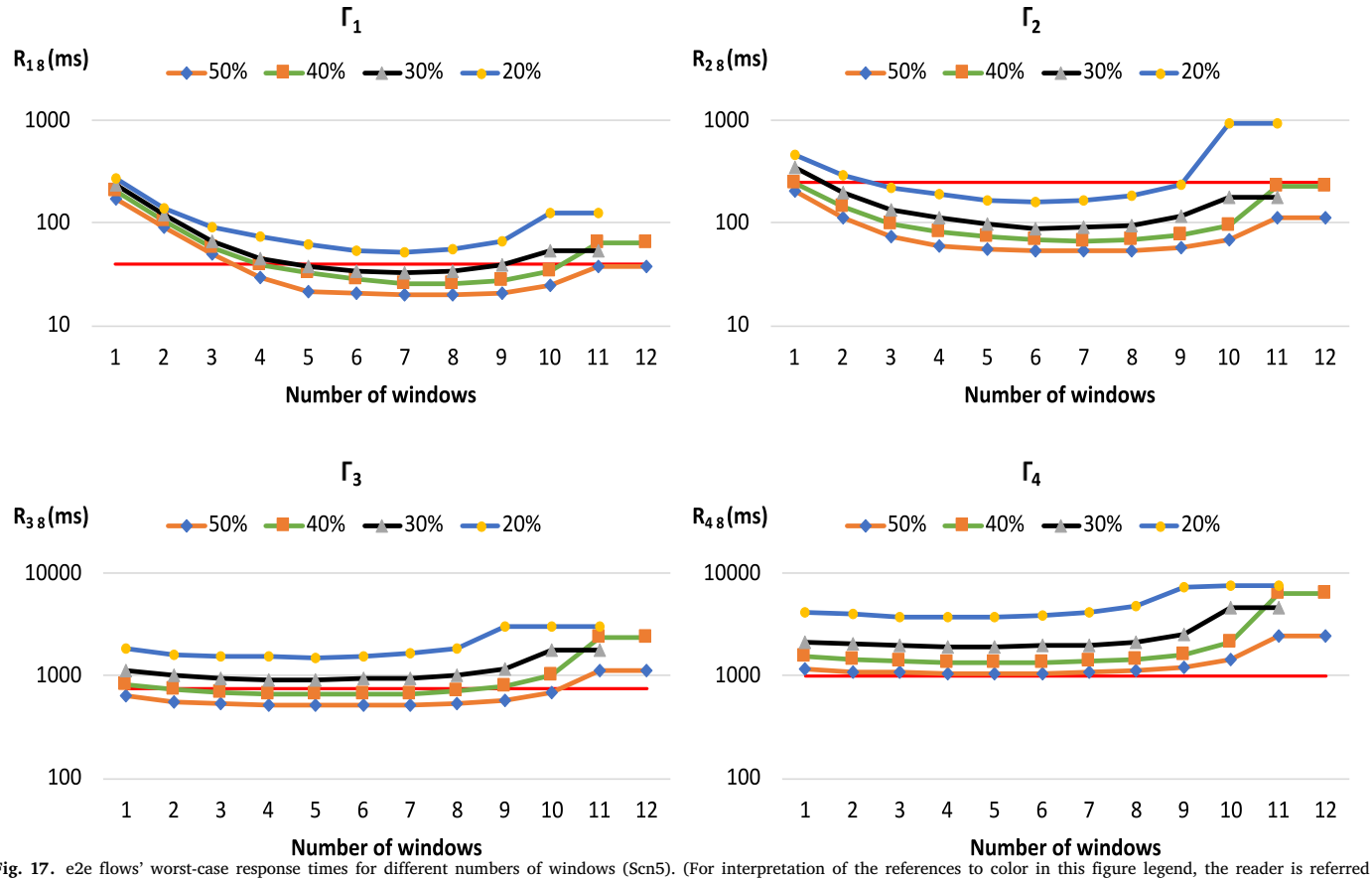

Fig. 17. e2e flows' worst-case response times for different numbers of windows (Scn5). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the message, three different safety functionalities are executed: (1) application of the Emergency Brake (EB functionality), (2) establishing a communication session with a centralized control center called Radio-Block Center (RBC-CS functionality), and (3) parameter visualization in Human–Machine-Interface (PV-DMI functionality). According to railway safety certification authorities, the execution of all functionalities must not take longer than 1 s from the reception of the message from the balise [38]. In order to provide a clear picture, only the EB functionality has been depicted in Fig. 18, although the others exhibit the same logical structure. This safety application is executed redundantly in two CPUs, and therefore there are two output events for each functionality that must satisfy temporal constraints: $e_{out1.1}$ and $e_{out1.2}$ for the EB functionality in Fig. 18.

Both processors are connected through a communication network, which is modeled as a black box where messages are characterized by a minimum and a maximum latency, which are assumed to be 40 μs and 400 μs, respectively. The manufacturer provided the worst-case execution times of each function, which were obtained through complex measurement techniques, although in order to maintain confidentiality, the exact values cannot be shown. However, the ones used in this experiment, shown in Table 3, are of the same magnitude order as the real ones. We assume that best-case execution times are half of the worst-case ones.

In order to apply WinAs to this industrial use-case, we will focus on the experiments from [13], where tentative available utilizations, partition windows and priorities were assigned, subject to future optimization. In this work we will perform several experiments where different fixed available utilization values are considered, in order to assess how this input parameter affects WinAs in order to schedule this industrial use-case. In Experiments 1 and 2 the same available utilization is allocated to all partitions, which should be enough to host the targeted application in both cases. Then these values are slightly

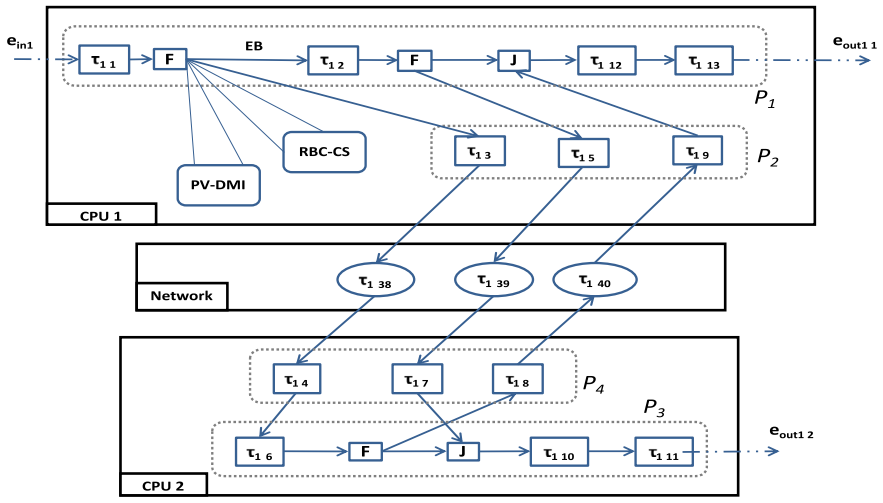

Fig. 18. Real-time application model (RBC-CS & PV-DMI not depicted for the sake of clarity).

Table 3
Train signaling application (times in μs).

Functionality	Emergency-brake application — EB												
τ_{ij}	$\tau_{1\ 1}$	$\tau_{1\ 2}$	$\tau_{1\ 3}$	$\tau_{1\ 4}$	$\tau_{1\ 5}$	$\tau_{1\ 6}$	$\tau_{1\ 7}$	$\tau_{1\ 8}$	$\tau_{1\ 9}$	$\tau_{1\ 10}$	$\tau_{1\ 11}$	$\tau_{1\ 12}$	$\tau_{1\ 13}$
C_{ij}	5	3	6	6	6	3	6	6	6	8	2	8	2
P_x	P_1	P_1	P_2	P_4	P_2	P_3	P_4	P_4	P_2	P_3	P_3	P_1	P_1
Functionality	RBC Communication-session establishment — RBC-CS												
τ_{ij}	–	$\tau_{1\ 14}$	$\tau_{1\ 15}$	$\tau_{1\ 16}$	$\tau_{1\ 17}$	$\tau_{1\ 18}$	$\tau_{1\ 19}$	$\tau_{1\ 20}$	$\tau_{1\ 21}$	$\tau_{1\ 22}$	$\tau_{1\ 23}$	$\tau_{1\ 24}$	$\tau_{1\ 25}$
C_{ij}	–	15	6	6	6	15	6	6	6	40	10	40	10
P_x	–	P_1	P_2	P_4	P_2	P_3	P_4	P_4	P_2	P_3	P_3	P_1	P_1
Functionality	Parameter visualization — PV-DMI												
τ_{ij}	–	$\tau_{1\ 26}$	$\tau_{1\ 27}$	$\tau_{1\ 28}$	$\tau_{1\ 29}$	$\tau_{1\ 30}$	$\tau_{1\ 31}$	$\tau_{1\ 32}$	$\tau_{1\ 33}$	$\tau_{1\ 34}$	$\tau_{1\ 35}$	$\tau_{1\ 36}$	$\tau_{1\ 37}$
C_{ij}	–	30	6	6	6	30	6	6	6	80	20	80	20
P_x	–	P_1	P_2	P_4	P_2	P_3	P_4	P_4	P_2	P_3	P_3	P_1	P_1

modified in order to see the impact of different distributions of available utilization among partitions, first by unbalancing the available utilization among partitions of the same processor (Experiment 3), and then by unbalancing the available utilization among partitions that are hosted in different processors (Experiment 4). The values of the available utilizations used are shown in Table 4.

Although the safety standards oblige completion of the execution of functionalities within 1 s, in this section a wider range of deadlines will be evaluated, in order to determine the behavior of the algorithm under more constrained and diverse timing requirements. Therefore, four different values of the e2e deadlines (50, 100, 500 and 1000 ms) will be explored.

Table 5 shows the results obtained by WinAs when it is applied to the industrial use-case for each experiment and deadline requirement. Each line of the table corresponds to a different e2e deadline and it shows the MAF obtained by the proposed algorithm, the worst-case response times of the output steps of the application, as well as the priority assignment technique that got these results. As can be seen, the proposed algorithm is capable of finding schedulable solutions for a wide range of deadline requirements and available utilization values. As expected, the worst-case response times always meet the deadlines of the e2e flow, and the more available utilization is assigned to a partition, the lower is the worst-case response time of the application. Experiment 1 replicates the experiment from [13]. In that paper, the authors tentatively set a 10 ms MAF in order to meet a 1000 ms deadline requirement. Applying WinAs, we obtain a schedulable solution by setting a 125 ms MAF, thus producing a much lower context-switch overhead.

Table 4
Available utilization of each partition (in %) in each experiment.

Experiment ID	AU_{P1}	AU_{P2}	AU_{P3}	AU_{P4}
1	2	2	2	2
2	8	8	8	8
3	6	1	6	1
4	3	3	1	1

6. Conclusions and future work

In this work, the scheduling of time partitions has been addressed. First, a study of the effect of the size and number of partition windows taking into account context switch overheads has been performed. It is concluded that increasing the number of windows of a partition is beneficial, up to a point, for the worst-case response times. The knowledge coming from this study has been leveraged to develop an algorithm, called WinAs, to search for schedulable solutions by adjusting the MAF with one window per partition for a fixed available utilization. The proposed algorithm has been applied to a representative set of synthetic experiments and also to the industrial use-case that motivates this work. It is shown to be capable of finding schedulable solutions in many different scenarios, including applications composed of a wide range of deadline requirements.

As highlighted in the paper, there are quite a few research paths to consider. Exploring a non-uniform window assignment would be one of the first tasks to address, together with different strategies of MAF reduction inside WinAs, depending on specific application features. All the knowledge gathered during the study, in combination with

Table 5
Results of applying WinAs to the railway application under the different experiments and deadlines tested (all times in ms).

D_{ij}	MAF	Priority	$\tau_{1\ 11}$	$\tau_{1\ 13}$	$\tau_{1\ 23}$	$\tau_{1\ 25}$	$\tau_{1\ 35}$	$\tau_{1\ 37}$
Experiment 1								
1000	125	ED	735.18	980.18	735.18	980.19	735.16	980.17
500	62.5	ED	367.68	490.18	367.68	490.19	367.66	490.17
100	12.5	PD_G	73.69	98.18	73.69	98.19	73.67	98.16
50	3.125	PD_G	24.69	30.81	21.64	27.68	21.66	27.77
Experiment 2								
1000	125	ED	690.18	920.18	690.185	920.19	690.16	920.17
500	62.5	ED	345.18	460.18	345.18	460.19	345.16	460.17
100	12.5	PD_G	69.19	92.18	69.19	92.19	69.17	92.15
50	6.25	PD_G	34.69	46.18	34.69	46.19	34.67	46.15
Experiment 3								
1000	125	ED	717.68	965.18	717.68	965.19	717.6	965.17
500	62.5	ED	358.93	482.68	358.93	482.69	358.91	482.67
100	12.5	PG_G	71.94	96.68	71.94	96.69	71.92	96.65
50	6.25	PD_G	36.06	48.43	36.07	48.44	36.05	48.41
Experiment 4								
1000	125	ED	613.92	977.68	737.67	977.69	737.65	977.67
500	62.5	ED	368.92	488.93	368.92	488.94	368.9	488.92
100	12.5	PD_G	86.3	97.93	73.94	97.94	73.95	97.91
50	6.25	PD_G	49.42	49.06	43.26	49.07	43.28	49.03

the proposed heuristic algorithm, provides a solid basis for developing more complex search and optimization algorithms, such as simulated annealing or genetic algorithms.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the "Doctorados Industriales 2018" program from Universidad de Cantabria, Spain and MCIN/ AEI /10.13039/501100011033/ FEDER "Una manera de hacer Europa" under grant TIN2017-86520-C3-3-R (PRECON-I4).

References

[1] H. Fang, R. Obermaisser, Execution environment for mixed-criticality train applications based on an integrated architecture, in: 2017 International Conference on Promising Electronic Technologies (ICPET), IEEE, 2017, pp. 1–7.
[2] Airlines Electronic Engineering Committee, Aeronautical Radio INC, Avionics Application Software Standard Interface. ARINC Specification 653-1, Vol. 2551, Aeronautical Radio, Inc, 2010, pp. 21401–27435.
[3] Aeronautical Radio INC, ARINC Specification 664P7: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network, Vol. 2551, Aeronautical Radio, Inc, 2009, pp. 21401–27435.
[4] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International, IEEE, 2007, pp. 239–243, http://dx.doi.org/10.1109/RTSS.2007.47.
[5] A. Burns, R.I. Davis, A survey of research into mixed criticality systems, ACM Comput. Surv. 50 (6) (2017) 82, http://dx.doi.org/10.1145/3131347.
[6] European Commission. Information Society and Media Directorate-General Unit G3/Computing Systems Research Objective, Mixed criticality systems, Report from the Workshop on Mixed Criticality Systems, 2012.
[7] S. Baruah, H. Li, L. Stougie, Towards the design of certifiable mixed-criticality systems, in: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, IEEE, 2010, pp. 13–22, http://dx.doi.org/10.1109/RTAS.2010.10.
[8] K. Goossens, A. Azevedo, K. Chandrasekar, M.D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A.B. Nejad, et al., Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow, ACM SIGBED Rev. 10 (3) (2013) 23–34, http://dx.doi.org/10.1145/2544350.2544353.
[9] A. Crespo, A. Alonso, M. Marcos, A. Juan, P. Balbastre, Mixed criticality in control systems, IFAC Proc. Vol. 47 (3) (2014) 12261–12271, http://dx.doi.org/10.3182/20140824-6-ZA-1003.02004.
[10] S. Trujillo, A. Crespo, A. Alonso, J. Pérez, Multipartes: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems, Microprocess. Microsyst. 38 (8) (2014) 921–932, http://dx.doi.org/10.1016/j.micpro.2014.09.004.
[11] IEC, IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3, 2010.
[12] D.J. Smith, K.G. Simpson, Functional Safety: A Straightforward Guide to Applying IEC 61508 and Related Standards, Routledge, 2004.
[13] A. Amurrio, E. Azketa, J.J. Gutierrez, M. Aldea, M.G. Harbour, Response-time analysis of multipath flows in hierarchically-scheduled time-partitioned distributed real-time systems, IEEE Access 8 (2020) 196700–196711, http://dx.doi.org/10.1109/ACCESS.2020.3033461.
[14] A. Amurrio, J.J. Gutiérrez, M. Aldea, E. Azketa, Priority assignment in hierarchically scheduled time-partitioned distributed real-time systems with multipath flows, J. Syst. Archit. 122 (2022) 102339, http://dx.doi.org/10.1016/j.sysarc.2021.102339.
[15] B. Woolley, S. Mengel, A. Ertas, An evolutionary approach for the hierarchical scheduling of safety-and security-critical multicore architectures, Computers 9 (3) (2020) 71.
[16] J. Holland, Adaptation in Artificial and Natural Systems, The University of Michigan Press, Ann Arbor, 1975.
[17] D. Tămaş-Selicean, P. Pop, Design optimization of mixed-criticality real-time embedded systems, ACM Trans. Embedded Comput. Syst. (TECS) 14 (3) (2015) 50, http://dx.doi.org/10.1145/2700103.
[18] D. Tămaş-Selicean, P. Pop, Task mapping and partition allocation for mixed-criticality real-time systems, in: Dependable Computing (PRDC), 2011 IEEE 17th Pacific Rim International Symposium on, IEEE, 2011, pp. 282–283, http://dx.doi.org/10.1109/PRDC.2011.42.
[19] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (5) (1986) 533–549, http://dx.doi.org/10.1016/0305-0548(86)90048-1.
[20] J. Chen, C. Du, P. Han, Scheduling independent partitions in integrated modular avionics systems, PLoS One 11 (12) (2016) http://dx.doi.org/10.1371/journal.pone.0168064.
[21] M. Minoux, Mathematical Programming: Theory and Algorithms, John Wiley & Sons, 1986.
[22] R.B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, 1997.
[23] E. Deroche, J.-L. Scharbarg, C. Fraboul, Mapping real-time communicating tasks on a distributed ima architecture, in: Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on, IEEE, 2016, pp. 1–8, http://dx.doi.org/10.1109/ETFA.2016.7733586.
[24] Y. Zhou, S. Samii, P. Eles, Z. Peng, Partitioned and overhead-aware scheduling of mixed-criticality real-time systems, in: Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019, pp. 39–44.
[25] Green Hills Software, Integrity RTOS, URL http://www.ghs.com/.
[26] M. González Harbour, J.J. Gutiérrez, J.C. Palencia, J.M. Drake, MAST: Modeling and analysis suite for real time applications, in: Proceedings of the 13th Euromicro Conference on Real-Time Systems, IEEE, 2001, pp. 125–134.
[27] Object Management Group, UML profile for MARTE: Modeling and analysis of real time embedded systems, version 1.1, OMG Document Formal (2011).

[28] M. González Harbour, J.J. Gutiérrez, J.M. Drake, P. López, J.C. Palencia, Modeling distributed real-time systems with MAST 2, J. Syst. Archit. 59 (6) (2013) 331–340.
[29] J.C. Palencia, M. González Harbour, Schedulability analysis for tasks with static and dynamic offsets, in: Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279), IEEE, 1998, pp. 26–37.
[30] J.C. Palencia, M. González Harbour, J.J. Gutiérrez, J.M. Rivas, Response-time analysis in hierarchically-scheduled time-partitioned distributed systems, IEEE Trans. Parallel Distrib. Syst. 28 (7) (2016) 2017–2030.
[31] J.J. Gutiérrez, J.C. Palencia, M. González Harbour, Schedulability analysis of distributed hard real-time systems with multiple-event synchronization, in: Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000, 2000, pp. 15–24, http://dx.doi.org/10.1109/EMRTS.2000.853988.
[32] J. Liu, Real-Time Systems, Vol. 48, Prentice Hall, 2000, p. 42.
[33] H. Kao, H. Garcia-Molina, Deadline assignment in a distributed soft real-time system, in: Proceedings. the 13th International Conference on Distributed Computing Systems, IEEE, 1993, pp. 428–437.
[34] A. Amurrio, E. Azketa, M. Aldea, J.J. Gutíerrez, How windows size and number can influence the schedulability of hierarchically-scheduled time-partitioned distributed real-time systems, in: 25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021), Work in Progress Session, in Ada User Journal 42, Ada Europe, 2021, pp. 101–104.
[35] E. Hamelin, M.A. Hmid, A. Naji, Y. Mouafo-Tchinda, Selection and evaluation of an embedded hypervisor: application to an automotive platform, in: Proceedings of the 10th Embedded Real-Time Systems International Congress (ERTS 2020).
[36] M. Masmano, I. Ripoll, A. Crespo, J. Metge, Xtratum: a hypervisor for safety critical embedded systems, in: Proceedings of the 11th Real-Time Linux Workshop 2009, pp. 263-272.
[37] ERTMS/ETCS, European rail traffic management system/European train control system release notes to system requirements specification, 2006, Subset 026 Version 2.3.0.
[38] UNISIG, ERTMS/ETCS -SUBSET-041- performance requirements for interoperability, 2015.