Journal of Systems Architecture 122 (2022) 102339

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Priority 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, University of Cantabria, Santander, Spain

ARTICLE INFO

MSC:
00-01
99-00

Keywords:
Priority assignment
Schedulability
Time-partitioning
Multipath Flows

ABSTRACT

The increasing complexity in the design of industrial embedded systems represents a challenge in the development of scheduling algorithms for such systems, which are essential to guarantee that they meet their deadlines even in the worst-case situation. In this work, we propose a new collection of non-iterative priority assignment algorithms for multipath flows within hierarchical schedulers based on state-of-the-art scheduling algorithms, which have been adapted to this complex system model. They are applied to an industrial railway use case that has motivated this work, and then their performance is evaluated in different general synthetic scenarios, with the aim of providing a view on how they behave in a wider range of system configurations.

1. Introduction

The design of modern cyber–physical systems is facing the challenge of meeting both sophisticated functional and non-functional requirements, which has led to the increase of their complexity. This phenomenon can be observed in several industrial domains, such as avionics, whose architectures have evolved to time and space partitioned models [1,2], where applications can execute in isolation, i.e. without interfering each other. Railway companies are also willing to re-factor their application designs and to substitute traditional architectures by novel execution environments that allow the execution of several applications within embedded architectures, even when their criticality levels (i.e. the level of assurance that safety functions provide according to a safety certification standard) are different [3]. This is what the scientific community calls mixed-criticality systems. During their design, it is essential to keep strict temporal and spatial isolation among components, so that low criticality applications cannot jeopardize high criticality ones. This is usually achieved by implementing partitioning techniques. On the one hand, temporal partitioning guarantees that an application will be executed only during the specified amount of time, without compromising other applications by CPU or shared resource monopolization. On the other hand, keeping applications isolated within their memory addresses and avoiding unauthorized read/write operations are part of the space partitioning techniques. Partitioning also allows component integration when several stakeholders take part in system development. In [4] readers can find a complete survey on mixed-criticality systems, addressing different design aspects such as timing analysis or scheduling issues. Several European research projects have also made important contributions on the execution of partitioned systems [5,6].

In the development of safety critical systems, it is also a common practice that software components are executed in more than one instance, so that processing outcomes have a higher level of integrity. A well-known strategy to achieve this is via active redundancy and voting techniques [7], and in the context of real-time systems, these redundant architectures can be modelled as complex execution flows, such as multipath flows. In multipath flows, fork and join structures are used to describe precedence or data dependencies between the different functions that compose safety critical applications. Such applications must meet, even in the worst-case scenario, hard deadlines that are imposed on software, being the consequences of not meeting them potentially catastrophic. Therefore, scheduling, which refers to the specific task-execution order that guarantees meeting such deadlines, becomes a major concern. In [8] the authors perform a deep analysis and characterization of existing techniques for mapping and scheduling distributed real-time systems, and one of the main conclusions is that the scheduling of distributed real-time systems based on partitioning is a topic that has been scarcely addressed.

* 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.2021.102339
Received 26 October 2021; Accepted 13 November 2021
Available online 3 December 2021

1.1. Related work

There is a vast number of scheduling approaches for real-time systems, since [9] demonstrated that Rate Monotonic priority assignment was optimal for periodic and independent tasks whose deadlines are equal to their periods. Later, in [10] Deadline Monotonic priority assignment was proved to be optimal when tasks' deadlines are not longer than their periods, and in [11] the authors presented the algorithm called "Optimal Priority Assignment" (OPA), which is optimal for arbitrarily triggered and independent tasks. As architectural paradigms evolved, so did the scheduling parameter assignment techniques, developing specifically dedicated algorithms for certain problems (heuristic approaches) such as [12,13] as well as search and more general optimization algorithms (metaheuristic approaches) such as genetic algorithms [14] or simulated annealing [15]. In [16], some of the most relevant priority assignment techniques from the last 40 years are compiled; they can be applied to a wide range of real-time systems, from mixed-criticality ones to those using probabilistic analysis.

More recently, time-partitioned systems are scheduled via iterative algorithms in [17] by allocating tasks to time-partitions and producing a Time-Division Multiple Access (TDMA) schedule taking context-switch overheads into consideration, addressing single-core platforms and constrained ($D < T$) deadlines. They do not provide any priority assignment for the hierarchical scheduler they implement. In [18] mixed-criticality applications are mapped and scheduled in heterogeneous partitioned architectures using an iterative optimization algorithm, although it does not consider inter-partition communications [19], which are a key feature in redundant execution schemes like the one presented in our use case. The work in [20] proposes an iterative algorithm for fixed-priority assignment in both preemptive and non-preemptive scheduling policies. However, it does not consider time-partitioned architectures like the one addressed in this work. Finally, in [21] an ILP (Integer Linear Programming) formulation is proposed for scheduling real-time tasks in uniprocessor systems. This approach can be applied to hierarchical schedulers, although they do not address distributed architectures.

On the other hand, non-iterative algorithms, i.e. those that provide a unique solution without any optimization loop, have also been used in the literature when addressing priority assignment. For instance, UD, ED, PD [22], EQF and EQS [23] are deadline distribution algorithms that have been used both in Fixed Priorities (FP) and Earliest Deadline First (EDF) systems. In [24] these algorithms were used with the aim of assigning Scheduling Deadlines in EDF systems, and in [25] it is demonstrated that there are systems scheduled by FP where non-iterative algorithms may outperform iterative ones. These works however were proposed only for linear systems without time-partitioning.

The system model addressed in this work, as well as the developed optimization tools around it, can be directly transformed into the DAG (Directed Acyclic Graph) model. There is a bunch of research works addressing this system model, such as [26–28]. In these works, the authors propose several methods to decompose DAGs into multi-core architectures under different scheduling policies (Earliest Deadline First, Rate Monotonic...) and always considering implicit-deadline constraints. However, there is not any work, as far as the authors know, addressing the particular features targeted in this paper, say hierarchical scheduling combining time-partitioning and preemptive FP.

1.2. Objectives and manuscript organization

As a step forward on re-factoring railway signalling applications, the aim of this work is to find feasible priority assignment solutions for the kind of systems represented by the industrial use case described in this work. Rather than implementing complex priority assignment algorithms that might entail high computation times, we want to adapt several non-iterative priority assignment algorithms to our multipath and time-partitioned system model, as they have exhibited reasonably good behaviour in the literature for different scenarios [25]. We will analyse and compare their performance by applying them (1) to our real industrial use case, and (2) to a complex synthetic system that enables the exploration of their performance on a wide range of system configurations.

In this manuscript we address a real-time scheduling problem whose main features have been described in Section 1, along with an overview of the related work. The system model and also the schedulability analysis technique are described in Section 2, and in Section 3 the selected algorithms to be adapted to this model are described in detail. Section 4 shows how the interpretation of the proposed algorithms is performed and in Section 5 they are applied to the motivating industrial use case, which is described in detail. Then, they are applied to a synthetically generated system in order to show their performance. Finally, Section 6 draws the conclusions and suggests some future research lines.

2. Modelling and analysis

This work is based on a system model compliant with MAST (Modelling and Analysis Suite Tool for Real-Time Applications) [29], which is a GPL open source model and also a set of tools developed by the University of Cantabria. It enables the description of the temporal behaviour of computing systems, and provides several algorithms for scheduling parameter assignment, schedulability analysis and simulation. Its metamodel has been evolved to a second version, MAST2 [30], which adds some novel scheduling policies and modelling elements, such as time partitioning. The model used in this work is aligned with the OMGs MARTE standard [31].

2.1. Architecture

We address a distributed architecture, where one or more communication networks allow the inter-connection of different processors. Processors provide hardware (storage, actuators...) and software (libraries, programs...) resources for task execution, and they host a real-time operating system that enables partitioning, such as Integrity [32]. We will assume without loss of generality that we can obtain the minimum and maximum latencies that messages undergo at network level. Scheduling communications traffic is beyond the scope of this paper and it remains as future work.

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 distributed end-to-end flow (e2e flow from now on), 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 j-th step in the e2e flow Γ_i is denoted τ_{ij}, and it has a worst-case and a best-case execution time, C_{ij} and C_{ij}^b 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 t_{in}. Therefore, the release time for that step is in the range of $[t_{in} + \Phi_{ij}, t_{in} + max(\Phi_{ij}, J_{ij})]$.

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. The worst-case response time (WCRT) is denoted as R_{ij} and the best-case response time (BCRT) as R_{ij}^b, and both are obtained by schedulability analysis

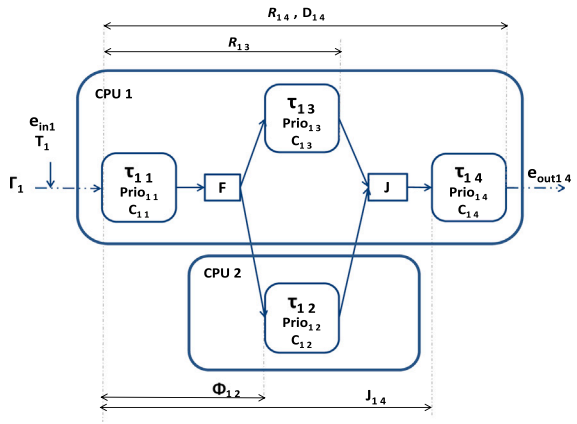

Fig. 1. Distributed multipath e2e flow.

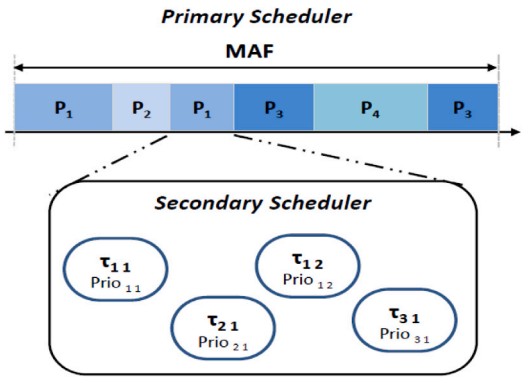

Fig. 2. Example of hierarchical scheduler.

techniques. As mentioned before, deadlines are imposed on software in order to guarantee that applications complete their duties within a bounded time. We identify such requirements as e2e deadlines, set at the output steps of e2e flows, and they are denoted as D_{ij}. 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 modelling 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.

Due to the multipath nature of these e2e flows, there may be more than one output step, each of them having a different (or even without) timing requirement. This is usual, for instance, when logging tasks are inserted: they do not have to meet any hard-deadline constraint but their scheduling must be determined along with the rest of the system. Each step may have more than one immediate predecessor and/or successor steps. The subset of steps immediately preceding the step τ_{ij} is named Γ_{ij}^{pred}, and similarly, Γ_{ij}^{succ} are those steps that are immediate successors of step τ_{ij}.

In Fig. 1, the workload event e_{in}, which is represented by a down-pointing arrow, forks and activates two steps, whose output events combine to activate a final step. Horizontal arrows represent precedence relations among event handlers.

2.2. Hierarchical scheduling

In this work hierarchically scheduled and time-partitioned systems are addressed. Hierarchical schedulers are composed of a primary scheduler and a secondary scheduler. A timetable-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). 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 highest number the highest priority. These 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, which is in essence, following the terminology just presented, the sum of the utilization of all the temporal windows within the MAF, so:

$$AU_{P_x} = \sum_{\forall Win_{xk} \in P_x} L_{xk}/MAF \tag{1}$$

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:

$$U_{P_x} = \sum_{\forall \tau_{ij} \in P_x} U_{ij} \tag{2}$$

Considering our target applications, we assume that steps are statically assigned to partitions, which is equivalent to the concept of partitioned scheduling used in multiprocessor systems; in contrast to global scheduling in which the migration of steps would be allowed.

2.3. Response-time analysis

The response-time analysis technique used to process the experimental evaluation of this work is [33], which is an offset-based technique [34]–[35] extended to support multipath e2e flows. It was demonstrated that it improves the results of the holistic approach [36] in general FP systems, so it is the most advanced tool available. Readers are encouraged to read the aforementioned references for a deeper understanding of the schedulability analysis.

3. Scheduling-parameter assignment overview

Scheduling-parameter assignment (say priorities in FP schedulers or scheduling-deadlines in EDF schedulers) is vital in the design and development of real-time systems. Even if some works such as [37]–[38] proposed optimal solutions in the field of multiprocessor scheduling, this problem is typically considered NP-hard for non-trivial cases [15]; the priority assignment to tasks is just a combinatorial problem where the number of possible solutions explodes quickly with the size of the system. That is why researchers have dedicated their efforts to developing algorithms that reach sub-optimal solutions in an acceptable computational time, and which are typically based on iterative optimization algorithms that improve their results at each iteration following some optimization criteria, which implies long computation times anyway.

In order to achieve the objectives mentioned in section 1.3, we select a collection of non-iterative algorithms proposed in the literature for different application domains. We will follow the same methodology as in [24], where the authors assign to each step what they call Virtual Deadlines (VDs). These VDs are not temporal requirements but just a mechanism to distribute the e2e deadline across all the steps of the e2e flow. We have selected the following algorithms:

- Ultimate Deadline (UD)
  It is the simplest scheduling-parameter assignment algorithm, where the e2e deadline is assigned to all steps composing the e2e flow [22]. It was used in [24] for VD assignment in linear e2e flows based on EDF schedulers.

$$VD_{ij} = D_i \tag{3}$$

  where D_i refers to the e2e deadline of the linear e2e flow Γ_i.
- 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 [22] . In [24] it was also used for VD assignment, considering linear e2e flows scheduled by EDF policy.

$$VD_{ij} = D_i - \sum_{k=j+1}^{N_i} C_{ik} \tag{4}$$

- Proportional Deadline (PD)
  The e2e deadline is distributed/dealt 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 [22].
  In [24] the authors used this algorithm in non-synchronized linear distributed systems based on EDF, therefore when distributing the e2e deadline they assigned local scheduling deadlines to steps. These deadlines are referred to the event that activates that step, and to be interpreted as local in the literature, it has been accepted that in linear e2e flows the sum of local deadlines should be the e2e deadline [39]. On the contrary, if there is a global clock and all scheduling deadlines are referred to the workload event that activates the e2e flow, they are global deadlines. The authors show that interpreting deadline distribution algorithms as local or global produces significant differences in response times.

$$VD_{ij} = \frac{C_{ij}}{\sum_{k=1}^{N_i} C_{ik}} * Di \tag{5}$$

- Normalized-Proportional Deadline (NPD)
  This algorithm is similar to PD, but it also considers the utilization of the processing element where it is hosted [22]. It was also used in [24] for scheduling-deadline assignment in EDF systems, based on linear e2e flows.

$$VD_{ij} = Di * \frac{C_{ij} * UP_{ij}}{\sum_{k=1}^{N_i} C_{ik} * UP_{ik}} \tag{6}$$

  where UP_{ij} is the utilization of the processor where τ_{ij} is hosted.
- Equal Slack (EQS)
  This algorithm was proposed for on-line deadline assignment in soft real-time distributed systems, based on EDF schedulers [23]. Deadline assignment is performed by equally dividing the slack, defined as the difference between the deadline and the worst-case response time. An interpretation of this algorithm was performed by [25] for off-line scheduling parameter assignment of distributed linear e2e flows. Since activation times are unknown for off-line schedulers, the authors assumed that such activations happened at time 0, and tasks' response times were assumed to be their worst-case execution times. Paradoxically this algorithm, which is non-iterative, produced better results than iterative algorithms when deadlines were larger than activation periods [25]. In that work, VDs are assigned as shown in Eq. (7):

$$VD_{ij} = C_{ij} + \frac{D_i - \sum_{k=j}^{N_i} C_{ik}}{N_i - j + 1} \tag{7}$$

  There are three main elements in Eq. (7): (1) the worst-case execution time of the step under assignment, (2) the numerator term, where the sum of the worst-case execution times of all the steps from the step under analysis till the end of the e2e flow are subtracted from the e2e deadline, and (3) the denominator term, which is the relative position of the step counted from the end of the e2e flow.
- Equal Flexibility (EQF)
  This algorithm was also originally proposed for on-line EDF scheduling [23], and it is also based on dividing the slack (considering the previously given definition), while the proportionality with respect of 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. This algorithm was also adapted for off-line scheduling of linear e2e flows in [25], and similarly to the previous algorithm, it outperformed other iterative algorithms when deadlines are higher than activation periods and in those scenarios where the e2e flows have to transit through the same processor more than once.

$$VD_{ij} = C_{ij} + [D_i - \sum_{k=j}^{Ni} C_{ik}] * [\frac{C_{ij}}{\sum_{k=j}^{N_i} C_{ik}}] \tag{8}$$

  Eq. (8) is composed of three main elements: (1) the worst-case execution time C_{ij} of the step under assignment, (2) a factor where the execution times of all the successor steps from the step under assignment till the end of the e2e flow are subtracted to the e2e deadline, and (3) a proportionality factor between the worst-case execution time of the step under assignment and the sum of all the successor steps from the step under assignment till the end of the e2e flow.

Supported by these results from the background literature presented, we propose to adapt these algorithms to the system model addressed in this work, which includes multipath e2e flows and time-partitions.

4. Priority assignment in multipath flows within time-partitions

In order to find schedulable solutions to our problem, we will follow a two-step strategy. First, based on the algorithms described in Section 3, we will formulate our new algorithms in order to apply them to multipath e2e flows within hierarchically scheduled time-partitioned architectures. These algorithms produce Virtual Deadlines. Then, VDs will be transformed into priorities in the second step of our proposal, following a Deadline Monotonic policy in the context of each partition.

4.1. Virtual deadline assignment

To illustrate the Virtual Deadline assignment process, a simple yet paradigmatic example is depicted in Fig. 3, where all of the challenging new features are contained: a single workload event triggers the execution of a multipath e2e flow with different timing constraints at its output events. The number within brackets represents the worst-case execution time of each step. For the sake of clarity, a single processor without time partitioning is assumed for this illustrative example. Distributed architectures and time-partitions in them are considered in the experiments performed in the paper, as the response-time analysis technique allows us to analyse both system models (time-partitioned and non time-partitioned).

Considering the illustrative example in Fig. 3, Table 1 details the results of applying the proposed algorithms in this illustrative example. The response times obtained by applying the analysis technique are also shown for all steps.

There are several remarkable conclusions to be mentioned. First, due to the nature of the UD algorithm, many VDs in the e2e flow are the same, which would lead to equal priorities if we did not solve these ties in some way. ED, PD_Local, NPD_Local and EQS also deal with ties, and they are solved as explained in the paper. Second, according to the worst-case response times of steps $\tau_{1,8}$ and $\tau_{1,9}$, the

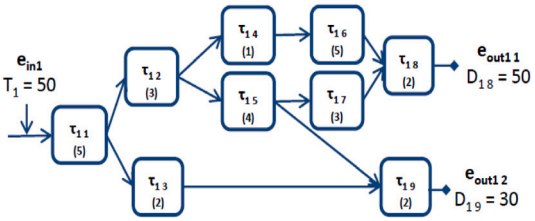

Fig. 3. Illustrative example.

Table 1
Virtual deadlines, priorities and response times for each algorithm, applied to the multipath e2e flow depicted in Fig. 3.

Simple example		τ_{11}	τ_{12}	τ_{13}	τ_{14}	τ_{15}	τ_{16}	τ_{17}	τ_{18}	τ_{19}
UD	VD_{ij}	30	30	30	50	30	50	50	50	30
	$Prio_{ij}$	9	8	7	4	6	3	2	1	5
	R_{ij}	5	8	10	17	14	22	25	**27**	**16**
ED	VD_{ij}	21	24	28	43	28	48	48	50	30
	$Prio_{ij}$	9	8	7	4	6	3	2	1	5
	R_{ij}	5	8	10	17	14	22	25	**27**	**16**
PD_Global	VD_{ij}	10.71	17.14	15	26.47	25.71	41.17	44.11	50	30
	$Prio_{ij}$	9	7	8	5	6	3	2	1	4
	R_{ij}	5	10	7	15	14	22	25	**27**	**17**
PD_Local	VD_{ij}	10.71	6.42	4.28	9.32	8.57	14.7	18.4	5.88	4.28
	$Prio_{ij}$	3	6	9	4	5	2	1	7	8
	R_{ij}	19	28	21	37	36	45	48	**50**	**38**
NPD_Global	VD_{ij}	10.71	17.14	15	26.47	25.71	41.17	44.11	50	30
	$Prio_{ij}$	9	7	8	5	6	3	2	1	4
	R_{ij}	5	10	7	15	14	22	25	**27**	**17**
NPD_Local	VD_{ij}	10.71	6.42	4.28	9.32	8.57	14.7	18.4	5.88	4.28
	$Prio_{ij}$	3	6	9	4	5	2	1	7	8
	R_{ij}	19	28	21	37	36	45	48	**50**	**38**
EQS	VD_{ij}	11.6	12.5	15	15	17.66	26.5	25.5	50	30
	$Prio_{ij}$	9	8	7	6	5	3	4	1	2
	R_{ij}	5	8	10	11	15	23	18	**27**	**25**
EQF	VD_{ij}	18.81	19.57	19.33	23.9	23.2	40.83	36.75	50	30
	$Prio_{ij}$	9	7	8	5	6	2	3	1	4
	R_{ij}	5	10	7	15	14	25	20	**27**	**17**

best priority assignment algorithms are UD and ED, PD_Global and EQF algorithms also obtain good results. However, as will be shown later, this should not be taken as representative for all situations, as there are other algorithms that exhibit better behaviour in other different system configurations. Finally, it should be noticed that the VDs and hence the priority assignments given by PD_Global and PD_Local are the same as the ones produced by NPD_Global and NPD_Local respectively. This is an expected result for this simple example since all the steps are hosted in the same partition, and therefore the normalization factor has no effect. Experiments at the evaluation section show that none of those algorithms produce the same VDs when e2e flows traverse more than one partition.

In the following lines the algorithms proposed for Virtual Deadline assignment are shown. Each algorithm is presented with its pseudocode, showing how it has been implemented.

4.1.1. Ultimate Deadline (UD)

Due to the multipath nature of our use case, there may be more than one timing requirement at different outputs. Therefore, an adequate propagation of such deadlines must be done, guaranteeing that the effect of the most restrictive one is propagated through all the paths where it has an influence. This propagation can be seen in Algorithm 1.

Algorithm 1 Ultimate Deadline

```
Initialize all VD_ij to inf.
for j ← N_i to 1 in each Γ_i do
  if ∄Γ_ij^succ then
    VD_ij = D_ij
  else
    for each τ_ik ∈ Γ_ij^succ do
      if VD_ik < VD_ij then
        VD_ij = VD_ik
      end if
    end for
  end if
end for
```

4.1.2. Effective Deadline (ED)

In addition to the issue of the different end-to-end deadlines, which applies here too, the execution time of the successor steps also influences the calculation of each VD. Therefore, for each step, the VD will be the most restrictive value of the difference between the successors VD and its worst-case execution time, as shown in Algorithm 2.

Algorithm 2 Effective Deadline

```
Initialize all VD_ij to inf.
for j ← N_i to 1 in each Γ_i do
  if ∄Γ_ij^succ then
    VD_ij = D_ij
  else
    for each τ_ik ∈ Γ_ij^succ do
      if VD_ik − C_ij < VD_ij then
        VD_ij = VD_ik − C_ik
      end if
    end for
  end if
end for
```

4.1.3. Proportional Deadline (PD)

As our experience in applying this algorithm with local and global deadlines corroborates that remarkable differences can be obtained in the schedulability of linear e2e flows [24], we will propose two versions of this algorithm for our system model as well.

- Global version (PD_Global):
  Based on the algorithm proposed in [22] described in Section 3, we propose the global version of the proportional deadline algorithm through the following equation, which retains the essence of the original algorithm:

$$VD_{ij} = Load_{ij} * F_{ij} \tag{9}$$

  where:

  - $Load_{ij}$
    Represents the accumulated load (sum of C_{ij}s) from the workload event to each step. When there is more than one path, the highest possible value will be considered. This propagation is shown in step 1 of Algorithm 3.
  - F_{ij}
    Represents the proportionality factor between the e2e deadline and the accumulated load at each step. To determine this value, the e2e deadline D_{ij} is divided by the term $Load_{ij}$ at all output steps with an e2e deadline. Then, it is propagated backwards, and in those steps with more than one predecessor step (where different e2e deadlines may have effect) the highest value of F_{ij} is propagated (so that the most restrictive VD is produced). This is shown in step 2 of Algorithm 3.

Algorithm 3 Proportional Deadline (PD_Global)

Step 1:
Initialize all $Load_{ij} = 0$
for j ← 1 to N_i in each Γ_i **do**
  if $\nexists \Gamma_{ij}^{pred}$ **then**
    $Load_{ij} = C_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ **do**
      if $Load_{ik} + C_{ij} > Load_{ij}$ **then**
        $Load_{ij} = Load_{ik} + C_{ij}$
      end if
    end for
  end if
end for
Step 2:
Initialize all F_{ij} to inf.
for j ← N_i to 1 in each Γ_i **do**
  if $\nexists \Gamma_{ij}^{succ}$ **then**
    $F_{ij} = D_{ij}/Load_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ **do**
      if $F_{ik} < F_{ij}$ **then**
        $F_{ij} = F_{ik}$
      end if
    end for
  end if
end for
Step3:
for each τ_{ij} in each Γ_i **do**
  Calculate VDs by Eq. (9)
end for

- Local version (PD_Local)
  This algorithm turns global deadlines obtained by Algorithm 3 into local deadlines. To do so, we invert the notion of local deadlines explained before, where the summation of local deadlines provided the e2e deadline in linear e2e flows: from each of the output steps in the e2e flow, the local VD of each step is obtained by subtracting the value of the global VD from the predecessor step. If there is more than one, the most restrictive (lowest value) VD is assigned. The algorithm that turns global VDs into local ones is described in Algorithm 4.

Algorithm 4 Turn Global VD into Local VD

for j ← N_i to 1 in each Γ_i **do**
  for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ **do**
    if $VD_{ij} - VD_{ik} < VD_{ij}$ **then**
      $VD_{ij} = VD_{ij} - VD_{ik}$
    end if
  end for
end for

4.1.4. Normalized Proportional Deadline (NPD)

This algorithm is similar to the PD algorithm; in this case a normalization factor that considers the utilization of the resource where steps are hosted is applied. Two versions of the algorithm are proposed too, considering Global and Local VDs:

- Global Version (NPD_Global)
  Following the same criterion applied to the PD algorithms, we propose an equation based on the original formulation and we explain how to calculate each factor that composes it in Algorithm 5.

$$VD_{ij} = Load'_{ij} * F'_{ij} \tag{10}$$

where

  - $Load'_{ij}$
    Is the accumulated load from the workload event to each step. Since we are addressing partitioned systems, this factor will refer to the utilization of partition where the step is allocated (P_x). Thus, the accumulated value is $C_{ij} * U_{P_x}$. When there is more than one predecessor step we propagate the maximum value, as shown in the step 1 of Algorithm 5.
  - F'_{ij}
    Is the proportionality factor between the e2e deadline and the accumulated load at each step. It is calculated in the same way as in PD_Global algorithm, as described in the step 2 of Algorithm 5.

Algorithm 5 Normalized Proportional Deadline (NPD_Global)

Step 1:
Initialize all $Load'_{ij} = 0$
for j ← 1 to N_i in each Γ_i **do**
  for each P_x **do**
    if $\tau_{ij} \in P_x$ **then**
      if $\nexists \Gamma_{ij}^{pred}$ **then**
        $Load'_{ij} = C_{ij} * U_{P_x}$
      else
        for each $\tau_{ik} \in \Gamma_{ij}^{pred}$ **do**
          if $Load'_{ik} + C_{ij} * U_{P_x} > Load'_{ij}$ **then**
            $Load'_{ij} = Load'_{ik} + C_{ij} * U_{P_x}$
          end if
        end for
      end if
    end if
  end for
end for
Step 2:
Initialize all F'_{ij} to inf.
for j ← N_i to 1 in each Γ_i **do**
  if $\nexists \Gamma_{ij}^{succ}$ **then**
    $F'_{ij} = D_{ij}/Load'_{ij}$
  else
    for each $\tau_{ik} \in \Gamma_{ij}^{succ}$ **do**
      if $F'_{ik} < F'_{ij}$ **then**
        $F'_{ij} = F'_{ik}$
      end if
    end for
  end if
end for
Step3:
for each τ_{ij} in each Γ_i **do**
  Calculate VDs by Eq. (10)
end for

- Local Version (NPD_Local)
  In order to develop the Local version of the Normalized Proportional Deadline algorithm we will apply Algorithm 4, which turns Global VDs into Local ones.

4.1.5. Equal Slack (EQS)

When Eq. (8) was formulated in [25], the authors considered linear systems where there is only a single e2e deadline. However in our model, the relative position of the step within the e2e flow becomes non-trivial due to the multipath structures. Therefore, we propose an equivalent equation which is similar in its structure, and in Algorithm 6 we show how each of the terms that compose the equation are calculated.

$$VD_{ij} = C_{ij} + \frac{H1_{ij}}{H2_{ij}} \tag{11}$$

where $H1_{ij}$ refers to the numerator term from Eq. (8) and $H2_{ij}$ reflects the denominator term. When calculating these terms, if there is more

than one predecessor step we will consider the one that produces the highest $H1_{ij}/H2_{ij}$ value, with the aim of obtaining the most restrictive VD.

Algorithm 6 Equal Slack

```
Initialize all H1 to inf. and H2 to 0.0
for j ← N_i to 1 in each Γ_i do
  if ∄Γ_ij^succ then
    H1_ij = D_ij − C_ij
    H2_ij = 1
    Calculate VD_ij through Eq. 11
  else
    H1_ij = H1_ij − C_ij
    H2_ij = H2_ij + 1
    Calculate VD_ij through Eq. (11)
  end if
  for each τ_ik ∈ Γ_ij^pred do
    if H1_ik/H2_ik > H1_ij/H2_ij then
      H1_ik = H1_ij
      H2_ik = H2_ij
    end if
  end for
end for
```

4.1.6. Equal Flexibility (EQF)

The formulation in Eq. (9) also considered linear e2e flows, and the sum of worst-case execution times from the step under assignment becomes non-trivial too, as there is more than a single path to take into account. Therefore, we propose an equation that retains the structure of Eq. (9), and in Algorithm 7 we detail how this interpretation is performed. We will also choose the $Q1_{ij} * Q2_{ij}$ that produces the lowest VD in those cases where there is more than one predecessor step.

$$VD_{ij} = C_{ij} + Q1_{ij} * Q2_{ij} \tag{12}$$

Algorithm 7 Equal Flexibility

```
Initialize all Q_1 and Q_2 to inf.
for j ← N_i to 1 in each Γ_i do
  if ∄Γ_ij^succ then
    Q1_ij = D_ij − C_ij
    Q2_ij = 1
    Calculate VD_ij through Eq. (12)
  else
    Q1_ij = Q1_ij − C_ij
    Q2_ij = C_ij/Q2_ij + C_ij
    Calculate VD_ij through Eq. (12)
  end if
  for each τ_ik ∈ Γ_ij^pred do
    if Q1_ik * Q2_ik > Q1_ij * Q2_ij then
      Q1_ik = Q1_ij
      Q2_ik = Q2_ij
    end if
  end for
end for
```

4.2. Virtual deadline transformation into priorities

As said before, the second stage of our priority assignment strategy is to transform the Virtual Deadlines into priorities. To do so, we assign priorities in the context of each partition following a deadline monotonic order, which assigns the highest priority to the step with the lowest Virtual Deadline.

Being a multipath architecture, it is likely that the same Virtual Deadline is assigned to more than one step, mostly depending on worst-case execution times of steps and also the e2e deadlines. In fact, there are cases where this kind of tie happens in a generalized manner, for instance: in the ED algorithm, the same VD will be assigned to all the steps preceding a Join event handler. If these steps are hosted in the same partition, assigning the same VD implies assigning the same priority, which is undesirable in the response-time analysis like the one applied in this work. In the absence of a clear criterion to solve such ties we propose the following approach: steps are processed following their index order, and they are sorted in a non-decreasing order according to their VDs. Then, priorities are assigned following this order decreasingly. Thus, the same priority is never assigned to two or more different steps. An optimized strategy for solving ties in priority assignment is a subject for future work.

5. Evaluation of the priority assignment algorithms

In this section the proposed algorithms are evaluated in different scenarios. First, a real industrial use case from the railway domain is presented. Then, a more general evaluation is performed by generating synthetic e2e flows with a wide range of activation patterns and deadline requirements.

5.1. Industrial use case

The scheduling problem addressed in this work is motivated by the need of train manufacturers, whose concern in scheduling of signalling applications motivated them to explore and develop novel response-time analysis and scheduling techniques [33]. In this section the motivational use case is presented, modelling a real application by means of the system model described previously and using real data provided by the application developers. However, there is some information omitted due to confidentiality issues, which does not lead to a loss of generality.

Fig. 4 shows the modelling of a railway signalling application, which supervises the driving and also provides information to drivers [40]. Supervision in performed through the execution of several functionalities, which are activated when the train goes through a balise and receives a message with driving instructions. This triggering event is represented by the workload event e_{in1}, and τ_{11} represents the capturing task. After that, three safety functionalities are executed: (1) application of the Emergency Brake (EB functionality), (2) establishing a communication session with a centralized control centre called Radio-Block Center (RBC-CS functionality), and (3) parameter visualization in Human–Machine-Interface (PV-DMI functionality). According to railway safety certification standards, all the functionalities must be executed within 1 s of receiving the message from the balise [41]. For the sake of clarity, only the EB functionality has been depicted in Fig. 4, 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 $e_{out_1.2}$ in Fig. 4. There are two partitions in each processor: P_1 for processing tasks and P_2 for communication tasks, and they are scheduled in a 10 ms MAF where partition windows are distributed uniformly. P_1 is composed of four 0.05 ms windows: $Win_{11} = \{0, 0.05\}$, $Win_{12} = \{2.5, 0.05\}$, $Win_{13} = \{5, 0.05\}$ and $Win_{14} = \{7.5, 0.05\}$, and P_2 is composed of eight 0.025 ms windows $Win_{21} = \{1, 0.025\}$, $Win_{22} = \{2.25, 0.025\}$, $Win_{23} = \{3.5, 0.025\}$, $Win_{24} = \{4.75, 0.025\}$, $Win_{25} = \{6, 0.025\}$, $Win_{26} = \{7.25, 0.025\}$, $Win_{27} = \{8.5, 0.025\}$ and $Win_{28} = \{9.75, 0.025\}$.

The communications network that connects both processors is considered a black box where messages are characterized by a minimum and a maximum latency, which based on our experience are assumed to be 40 μs and 400 μs, respectively. The manufacturer provided the measured worst-case execution times of each function, but the exact values cannot be shown in order to maintain confidentiality. However, the ones used in this test, shown in Table 2, are of the same magnitude

Table 2
Priority assignment for the train signalling application (C_{ij} in μs)

Functionality	Emergency Brake - 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
Priority assignment													
UD	10	9	9	9	8	9	8	7	7	8	7	8	7
ED	10	7	3	3	2	7	1	2	1	4	3	4	3
PD_Global	10	9	7	3	3	7	1	2	1	4	3	4	3
PD_Local	10	6	2	2	1	8	1	3	3	3	9	3	9
NPD_Global	10	9	9	9	6	9	6	5	5	8	3	7	3
NPD_Local	10	4	1	3	2	8	1	2	3	3	9	3	9
EQS	10	9	7	7	4	9	1	4	1	6	3	6	3
EQF	9	6	4	6	9	6	1	9	1	9	3	10	3
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	
Priority assignment													
UD	6	6	6	5	6	5	4	4	5	4	5	4	
ED	8	6	6	5	8	4	5	4	5	2	5	2	
PD_Global	8	8	8	5	8	5	4	2	5	2	5	2	
PD_Local	5	5	5	4	5	4	6	6	2	7	2	8	
NPD_Global	8	8	8	4	7	4	3	3	5	2	5	2	
NPD_Local	5	4	5	5	5	4	6	6	2	7	2	8	
EQS	8	8	8	5	8	2	5	2	5	2	5	2	
EQF	5	3	7	8	5	2	5	5	8	2	8	2	
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	
Priority assignment													
UD	3	3	3	2	3	2	1	1	2	1	2	1	
ED	9	9	9	8	9	7	8	7	6	1	6	1	
PD_Global	7	9	9	6	9	7	6	4	6	1	6	1	
PD_Local	4	8	8	7	4	7	9	9	1	6	1	7	
NPD_Global	6	7	7	2	6	2	1	1	4	1	4	1	
NPD_Local	6	9	9	7	4	7	8	8	1	6	1	7	
EQS	7	9	9	6	7	3	6	3	4	1	4	1	
EQF	4	2	8	7	4	3	4	6	7	1	7	1	

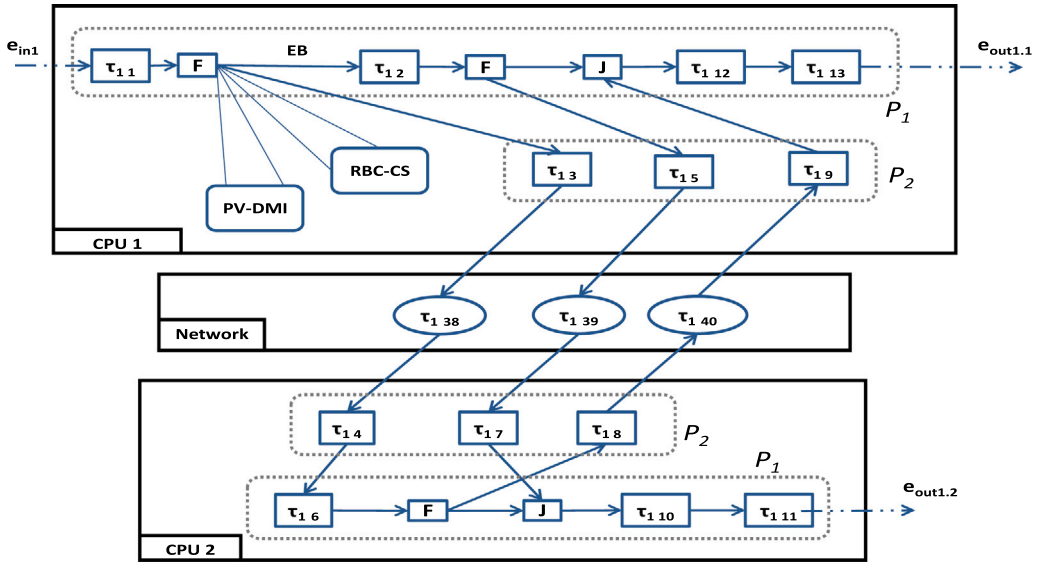

Fig. 4. Real-time application model (RBC-CS & PV-DMI not depicted for the sake of clarity).

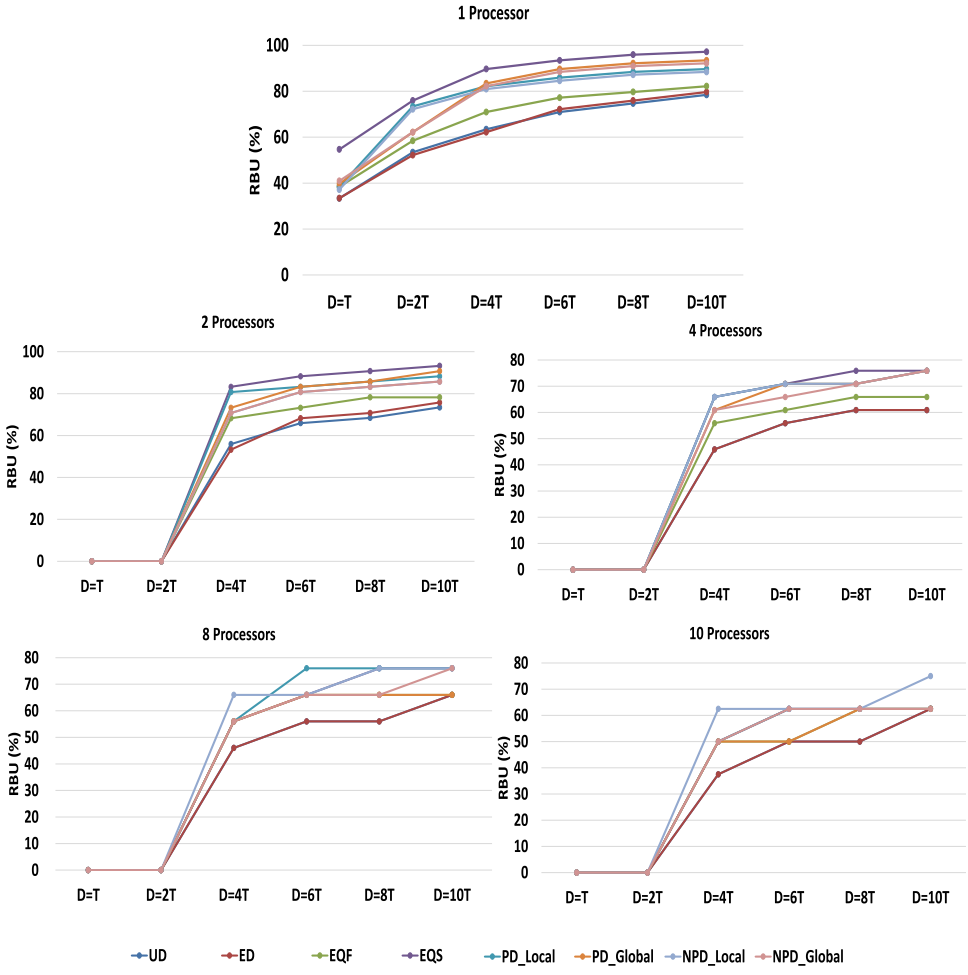

Fig. 5. RBU evolution of all algorithms.

Table 3
Worst-case response times of the railway signalling application.

Railway signalling app	EB		RBC-CS		PV-DMI	
	$\tau_{1\ 11}$	$\tau_{1\ 13}$	$\tau_{1\ 23}$	$\tau_{1\ 25}$	$\tau_{1\ 35}$	$\tau_{1\ 37}$
Worst-case response times (ms)						
UD	12.28	14.74	12.37	14.83	19.85	22.31
ED	17.34	19.78	17.34	19.79	17.37	19.82
PD_Global	17.37	19.78	17.34	19.79	17.37	19.82
PD_Local	14.84	17.31	16.12	18.59	21.11	23.58
NPD_Global	14.89	17.33	14.9	17.34	17.37	19.81
NPD_Local	16.09	18.52	17.35	18.6	22.32	23.59
EQS	14.91	17.34	14.92	17.35	17.39	19.82
EQF	21.16	23.63	21.17	23.64	23.65	26.11

Table 4
Slack factor for each algorithm applied in the industrial use case.

	Slack factor
UD	81.87
ED	88.7
PD_Global	88.33
PD_Local	66.25
NPD_Global	87.5
NPD_Local	68.75
EQS	86.25
EQF	51.14

order as the real ones. We assume that best-case execution times are half of the worst-case ones.

The resulting priority assignments are shown in Table 2. For all cases, the highest number means the highest priority, and as explained before, priorities are valid in the context of each partition. As in the previous example, here the different assignments obtained by the algorithms can be seen. The schedulability analysis introduced in Section 3 is applied for each priority assignment, and the worst-case response times obtained for the steps with an e2e deadline have been compiled in Table 3. These steps are $\tau_{1\ 11}$ and $\tau_{1\ 13}$ for the emergency brake functionality, $\tau_{1\ 23}$ and $\tau_{1\ 25}$ for the RBC Communication-session establishment functionality and $\tau_{1\ 35}$ and $\tau_{1\ 37}$ for the parameter visualization functionality.

Looking at Table 3, the lowest response time for the whole application is obtained by the NPD_Global algorithm, as it completes it execution in 19.81 ms in the worst-case scenario, and ED, PD_Global and EQS show a very similar performance (19.82 ms in the worst-case scenario). Taking a look at each functionality independently, it is remarkable that with UD, PD_Local and NPD_Local algorithms the EB functionality finishes its execution several ms before the others. This, however, is achieved by penalizing the execution of the other two functionalities, so in those cases where applications must meet different deadline requirements, the collection of algorithms proposed in this work can provide different alternatives for their design.

Focusing only on the response times may not provide an adequate view on the system schedulability if there are several outputs in response to the same input event, which is actually the case described in this application. It is convenient to perform a sensitivity analysis through the calculation of Slack times. These times can be associated to a single task, to a single e2e flow or to the whole system under analysis.

Table 5
Highest RBU results (in %) with time-partitioning.

	$D = T$	$D = 2T$	$D = 4T$	$D = 6T$	$D = 8T$	$D = 10T$
1 Processor	EQS (54)	EQS (75)	EQS (89)	EQS (93)	EQS (95)	EQS (97)
2 Processors	NS –	NS –	EQS (83)	EQS (88)	EQS (90)	EQS (93)
4 Processors	NS –	NS –	EQS PD_Local NPD_Local (66)	EQS,PD_Global PD_Local,NPD_Local (71)	EQS (76)	EQS,PD_Local NPD_Local PD_Global, NPD_Global (76)
8 Processors	NS	NS – –	PD_Local (66)	NPD_Local (76)	EQS,PD_Local NPD_Local (76)	EQS,PD_Local NPD_Local NPD_Global (76)
10 Processors	NS –	NS –	NPD_Local (62)	EQS,PD_Local, NPD_Local NPD_Global (62)	EQF, EQF, PD_Local PD_Global, NPD_Local NPD_Global (62)	NPD_Local (75)

Table 6
Highest BU results (in %).

	$D = T$	$D = 2T$	$D = 4T$	$D = 6T$	$D = 8T$	$D = 10T$
1 Processor	NPD_Local (63)	EQS (78)	EQS (89)	EQS (93)	EQS (95)	EQS (96)
2 Processors	PD_Local, NPD_Local (61)	EQS (76)	EQS (89)	EQS (93)	EQS (94)	EQS (95)
4 Processors	PD_Local, NPD_Local (57)	PD_Local (72)	EQS (85)	EQS (91)	EQS (93)	EQS (94)
8 Processors	PD_Local, PD_Global NPD_Local (49)	EQS, PD_Local, NPD_Local (64)	EQS, PD_Local NPD_Local NPD_Global (78)	EQS PD_Local NPD_Local (84)	EQS (88)	EQS (96)
10 Processors	NPD_Local (46)	EQS (63)	NPD_Local (80)	EQS (86)	EQS (89)	EQS (91)

According to MAST,[1] a Slack time is the value by which the associated element may be increased while maintaining the system schedulable. If the calculated slack is negative, this means that the system is not schedulable and the percentage obtained should be subtracted in order to achieve schedulability. Based on this, we define the Slack Factor as the maximum factor that can be applied to the execution times of all tasks until the boundary of schedulability is reached. In our use case, this factor can give designers an insight into how much systems could grow (in terms of utilization) when applying the different priority assignment algorithms. The Slack Factors calculated for each algorithm are shown in Table 4: even if NPD_Global seemed to be the best algorithm for our use case in terms of response times, the priority assignment that lets the system load grow the most is ED, closely followed by PD_Global and NPD_Global. The fact that the Slack Factors are so high can be explained by highlighting that the system represents a very low load and therefore execution times can be greatly increased.

5.2. Performance evaluation

After applying all the proposed algorithms to our industrial application, we need to assess their behaviour when external conditions differ from the ones that characterize this use case, in order to get a deeper view of their behaviour. Instead of generating a huge number of synthetic task sets to perform our evaluation, we are committed to making all our experiments reproducible by the research community and therefore we will generate a small-sized task set that includes most of the representative features relevant for the system model we are targeting in this work. We will generate synthetic DAGs with the tool TGFF [42], which can be directly transformed to our system model and then processed by our tools. We will use the TGFF tool to randomly generate ten different step-sequences, and we will assign steps' random worst-case execution times in a 0-20 ms range. These e2e flows will allow us to test the following features:

- Activation periods: We will test a wide range of activation periods, from 50 ms to 1s. These values are within the ranges of the sampling frequency of automotive subsystems [43] and the typical minimum inter-arrival times of balises in the railway domain.
- Deadlines: Different deadline requirements (relative to the periods) will be evaluated. The most restrictive requirements are normally found in certification standards, although we will consider more relaxed deadlines as distributed systems normally have deadlines larger than periods.
- Number of processors: We will analyse the behaviour of our algorithms first by assigning 80% of available utilization to a single partition allocated to a single processor, and then by distributing this available utilization into 2, 4, 8 and 10 processors (with one partition each) having the same total available utilization (this means that when testing 2 processors, partitions within them will get 40% of available utilization, and so on). Step-to-core mapping is performed randomly, but two consecutive steps will not be allowed to be assigned to the same processors, if possible. The utilization of all partitions shall be kept in a fair balance so that the pessimism that unbalanced loads produce is minimized.

In order to determine which of the proposed algorithms shows the best performance, we introduce the term Relative Breakdown Utilization (RBU) for time partitioned systems. It is based on the Breakdown

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

Table 7
Execution time of some of the experiments.

Experiment	Execution time (s)
Simple example	26
Railway signalling use case	28
1 Processor - D=T	36
1 Processor - D=10T	27
10 Processors - D=10T	31

Utilization (BU) introduced in [44], extended to be applied to partitioned systems where CPU time is not fully available for a partition. Thus, the RBU term (in percentage) for a partition is the value calculated as $U_{P_X}/AU_{P_X} * 100$, reached when all the execution times are scaled up to a point at which a deadline is first missed. To do so, we apply a scaling factor to all execution times in the context of each partition, until the system reaches the boundary of schedulability. For those tests with more than one partition, we will take the average RBU to show the experimental results, which is possible because step-to-processor mapping is kept fairly balanced in all cases. As a reference, we will consider a non-partitioned system where the CPU is fully available (general FP systems). Thus, we will replicate all the experiments considering a single partition with $AU = 100\%$, so we will show the BU for each algorithm.

When deadlines are too restrictive or the system is barely schedulable, and hence for calculating RBU/BU, the execution times of tasks have to be drastically reduced, we will consider that when partitions reach a utilization lower than 1%, the system is not schedulable (NS). This happens because the response-time analysis computes all the time-window gaps where the execution is not allowed and thus worst-case response times increase. Proposing an optimized partition window assignment is beyond the scope of this work.

The most relevant results have been compiled in Tables 5 and 6. They show the algorithm or algorithms that obtain the highest RBU/BU (expressed in % in brackets) for each combination of number of processors and deadline/period rates. Generally comparing Tables 5 and 6, we can directly see the penalization occurring when 100% of CPU-time is not available, i.e. without time partitioning, those scenarios where D=T are schedulable, whereas with time partitioning only the scenario with a single processor is schedulable. Even if the RBU term is relative to the CPU availability, the CPU unavailability and the effect it has in the analysis technique for time-partitioned systems provokes that utilizations in all scenarios are always lower than their counterparts in non-partitioned ones.

Regarding Table 5, which corresponds to time-partitioned systems, when the computing is performed in a single processor, the algorithm that obtains the highest RBU is always EQS. When the number of processors is increased some other algorithms seem to behave better, such as NPD_Local that produces the highest RBU when deadlines are 6, 8 and 10 times the period. Moreover, PD_Local also exhibits the a high performance in distributed systems where deadlines are 4, 6 or 8 times the period. In systems without time-partitioning (Table 6) NPD_Local always obtains the highest BU when deadlines are within the periods, and in this scenario PD_Local also exhibits good performance when steps are mapped in more than one processor. When deadlines tend to relax, EQS is the most suitable in most of the experiment-configurations. Note that the UD and ED algorithms, which had obtained fairly good results in the simple example from Section 4, never appear in any of these tables as the best ones.

In order to complete the qualitative analysis of these tests Fig. 5 shows, for each number of processors, the evolution of the RBU obtained by the proposed algorithms as a function of the D/T ratio. This allows readers to track the performance of those algorithms that are not present in the tables, which only presented the outstanding algorithms in each experiment configuration.

As a general conclusion, it can be stated that there is no single algorithm that behaves the best in all cases. This reinforces our idea of proposing simple and non-iterative algorithms so that all of them can be applied, and then the most suitable result can be selected. Table 7 shows the execution times of some of the experiments conducted in this work: the simple example from Section 4, the railway signalling use case from Section 5.1 and a subset of the synthetic scenarios from Section 5.2. In each experiment, the presented execution time includes the execution of the eight priority assignment algorithms plus their worst-case response-time analysis.

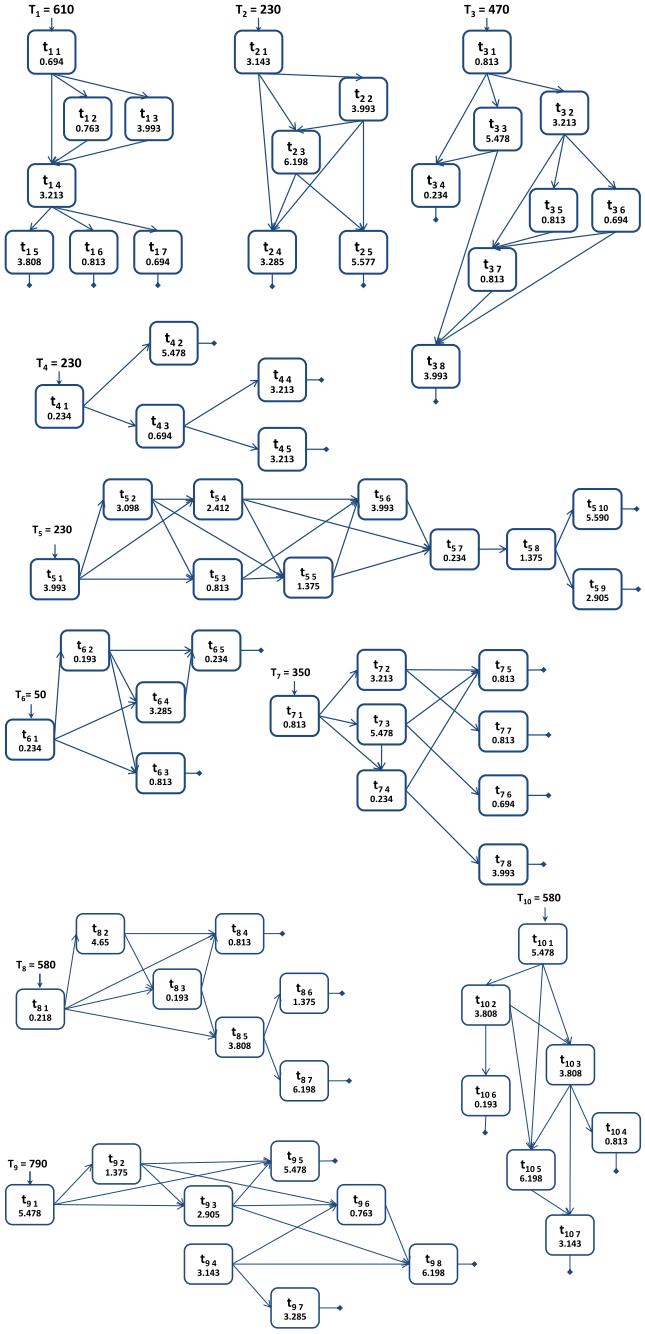

Fig. 6. Generated e2e flows.

6. Conclusions and future work

In this paper we have presented a collection of algorithms for priority assignment to multipath e2e flows in hierarchically scheduled and time-partitioned distributed real-time systems. All of them are non-iterative algorithms that can provide feasible solutions in a short time,

Table A.8
Step-to-Processor mapping of the synthetic e2e flows.

Number of Processors	Step-to-Processor Mapping									
1	$\tau_{1\,1}$ $\tau_{1\,2}$ $\tau_{1\,3}$ $\tau_{1\,4}$ $\tau_{1\,5}$ $\tau_{1\,6}$ $\tau_{1\,7}$ $\tau_{2\,1}$ $\tau_{2\,2}$ $\tau_{2\,3}$ $\tau_{2\,4}$ $\tau_{2\,5}$ $\tau_{3\,1}$ $\tau_{3\,2}$ $\tau_{3\,3}$ $\tau_{3\,4}$ $\tau_{3\,5}$ $\tau_{3\,6}$ $\tau_{3\,7}$ $\tau_{3\,8}$ $\tau_{4\,1}$ $\tau_{4\,2}$ $\tau_{4\,3}$ $\tau_{4\,4}$ $\tau_{4\,5}$ $\tau_{5\,1}$ $\tau_{5\,2}$ $\tau_{5\,3}$ $\tau_{5\,4}$ $\tau_{5\,5}$ $\tau_{5\,6}$ $\tau_{5\,7}$ $\tau_{5\,8}$ $\tau_{5\,9}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,2}$ $\tau_{6\,3}$ $\tau_{6\,4}$ $\tau_{6\,5}$ $\tau_{7\,1}$ $\tau_{7\,2}$ $\tau_{7\,3}$ $\tau_{7\,4}$ $\tau_{7\,5}$ $\tau_{7\,6}$ $\tau_{7\,7}$ $\tau_{7\,8}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{8\,3}$ $\tau_{8\,4}$ $\tau_{8\,5}$ $\tau_{8\,6}$ $\tau_{8\,7}$ $\tau_{9\,1}$ $\tau_{9\,2}$ $\tau_{9\,3}$ $\tau_{9\,4}$ $\tau_{9\,5}$ $\tau_{9\,6}$ $\tau_{9\,7}$ $\tau_{9\,8}$ $\tau_{10\,1}$ $\tau_{10\,2}$ $\tau_{10\,3}$ $\tau_{10\,4}$ $\tau_{10\,5}$ $\tau_{10\,6}$ $\tau_{10\,7}$									
2	$\tau_{1\,1}$ $\tau_{1\,3}$ $\tau_{1\,5}$ $\tau_{1\,7}$ $\tau_{2\,2}$ $\tau_{2\,4}$ $\tau_{3\,1}$ $\tau_{3\,3}$ $\tau_{3\,5}$ $\tau_{3\,7}$ $\tau_{4\,2}$ $\tau_{4\,4}$ $\tau_{5\,1}$ $\tau_{5\,3}$ $\tau_{5\,5}$ $\tau_{5\,7}$ $\tau_{5\,9}$ $\tau_{6\,2}$ $\tau_{6\,4}$ $\tau_{7\,1}$ $\tau_{7\,3}$ $\tau_{7\,5}$ $\tau_{7\,7}$ $\tau_{8\,2}$ $\tau_{8\,4}$ $\tau_{8\,6}$ $\tau_{9\,1}$ $\tau_{9\,3}$ $\tau_{9\,5}$ $\tau_{9\,7}$ $\tau_{10\,2}$ $\tau_{10\,4}$ $\tau_{10\,6}$									$\tau_{1\,2}$ $\tau_{1\,4}$ $\tau_{1\,6}$ $\tau_{2\,1}$ $\tau_{2\,3}$ $\tau_{2\,5}$ $\tau_{3\,2}$ $\tau_{3\,4}$ $\tau_{3\,6}$ $\tau_{3\,8}$ $\tau_{4\,1}$ $\tau_{4\,3}$ $\tau_{4\,5}$ $\tau_{5\,2}$ $\tau_{5\,4}$ $\tau_{5\,6}$ $\tau_{5\,8}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,3}$ $\tau_{6\,5}$ $\tau_{7\,2}$ $\tau_{7\,4}$ $\tau_{7\,6}$ $\tau_{7\,8}$ $\tau_{8\,1}$ $\tau_{8\,3}$ $\tau_{8\,5}$ $\tau_{8\,7}$ $\tau_{9\,2}$ $\tau_{9\,4}$ $\tau_{9\,6}$ $\tau_{9\,8}$ $\tau_{10\,1}$ $\tau_{10\,3}$ $\tau_{10\,5}$ $\tau_{10\,7}$							
4	$\tau_{1\,1}$ $\tau_{1\,5}$ $\tau_{2\,2}$ $\tau_{3\,3}$ $\tau_{4\,1}$ $\tau_{5\,1}$ $\tau_{5\,5}$ $\tau_{6\,4}$ $\tau_{7\,3}$ $\tau_{7\,8}$ $\tau_{8\,2}$ $\tau_{8\,5}$ $\tau_{9\,1}$ $\tau_{10\,4}$ $\tau_{10\,5}$				$\tau_{1\,2}$ $\tau_{1\,6}$ $\tau_{2\,1}$ $\tau_{2\,5}$ $\tau_{3\,2}$ $\tau_{3\,4}$ $\tau_{3\,8}$ $\tau_{4\,2}$ $\tau_{4\,3}$ $\tau_{5\,2}$ $\tau_{5\,6}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{6\,5}$ $\tau_{7\,4}$ $\tau_{7\,7}$ $\tau_{8\,3}$ $\tau_{8\,6}$ $\tau_{9\,2}$ $\tau_{9\,6}$ $\tau_{10\,1}$ $\tau_{10\,6}$				$\tau_{1\,3}$ $\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,1}$ $\tau_{3\,5}$ $\tau_{3\,7}$ $\tau_{4\,4}$ $\tau_{5\,3}$ $\tau_{5\,7}$ $\tau_{5\,9}$ $\tau_{6\,2}$ $\tau_{7\,1}$ $\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{8\,7}$ $\tau_{9\,3}$ $\tau_{9\,7}$ $\tau_{9\,8}$ $\tau_{10\,2}$ $\tau_{10\,7}$				$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{3\,6}$ $\tau_{4\,5}$ $\tau_{5\,8}$ $\tau_{6\,3}$ $\tau_{7\,6}$ $\tau_{8\,1}$ $\tau_{9\,4}$ $\tau_{9\,5}$ $\tau_{10\,3}$			
8	$\tau_{1\,1}$ $\tau_{3\,7}$ $\tau_{5\,5}$ $\tau_{6\,4}$ $\tau_{7\,3}$ $\tau_{9\,1}$	$\tau_{1\,2}$ $\tau_{2\,1}$ $\tau_{3\,8}$ $\tau_{4\,3}$ $\tau_{5\,6}$ $\tau_{7\,4}$ $\tau_{8\,3}$ $\tau_{9\,2}$ $\tau_{10\,1}$ $\tau_{6\,5}$	$\tau_{1\,3}$ $\tau_{2\,2}$ $\tau_{3\,1}$ $\tau_{4\,4}$ $\tau_{5\,7}$ $\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{9\,3}$ $\tau_{10\,2}$	$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{4\,1}$ $\tau_{5\,8}$ $\tau_{7\,6}$ $\tau_{8\,5}$ $\tau_{9\,4}$ $\tau_{10\,3}$	$\tau_{1\,5}$ $\tau_{3\,3}$ $\tau_{4\,5}$ $\tau_{5\,1}$ $\tau_{5\,9}$ $\tau_{7\,7}$ $\tau_{8\,6}$ $\tau_{9\,5}$ $\tau_{10\,4}$	$\tau_{1\,6}$ $\tau_{2\,5}$ $\tau_{3\,4}$ $\tau_{5\,2}$ $\tau_{5\,10}$ $\tau_{6\,1}$ $\tau_{7\,8}$ $\tau_{9\,6}$ $\tau_{10\,5}$	$\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,5}$ $\tau_{5\,3}$ $\tau_{6\,2}$ $\tau_{7\,1}$ $\tau_{8\,7}$ $\tau_{9\,7}$ $\tau_{10\,6}$	$\tau_{3\,6}$ $\tau_{4\,2}$ $\tau_{5\,4}$ $\tau_{6\,3}$ $\tau_{7\,2}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{9\,8}$ $\tau_{10\,7}$
10	$\tau_{1\,1}$ $\tau_{4\,2}$ $\tau_{5\,7}$ $\tau_{6\,1}$ $\tau_{7\,5}$ $\tau_{8\,4}$ $\tau_{9\,3}$	$\tau_{1\,2}$ $\tau_{2\,1}$ $\tau_{4\,3}$ $\tau_{10\,1}$ $\tau_{5\,8}$ $\tau_{7\,6}$ $\tau_{8\,5}$ $\tau_{9\,4}$	$\tau_{1\,3}$ $\tau_{3\,1}$ $\tau_{4\,4}$ $\tau_{10\,2}$ $\tau_{7\,7}$ $\tau_{8\,6}$ $\tau_{9\,5}$	$\tau_{1\,4}$ $\tau_{2\,3}$ $\tau_{3\,2}$ $\tau_{4\,1}$ $\tau_{10\,3}$ $\tau_{5\,9}$ $\tau_{7\,8}$ $\tau_{8\,7}$ $\tau_{9\,6}$	$\tau_{1\,5}$ $\tau_{3\,3}$ $\tau_{5\,1}$ $\tau_{10\,4}$ $\tau_{9\,7}$	$\tau_{1\,6}$ $\tau_{2\,5}$ $\tau_{3\,4}$ $\tau_{5\,2}$ $\tau_{5\,10}$ $\tau_{10\,5}$ $\tau_{9\,8}$	$\tau_{1\,7}$ $\tau_{2\,4}$ $\tau_{3\,5}$ $\tau_{5\,3}$ $\tau_{6\,2}$ $\tau_{7\,1}$ $\tau_{10\,6}$ $\tau_{2\,2}$	$\tau_{3\,6}$ $\tau_{5\,4}$ $\tau_{6\,3}$ $\tau_{7\,2}$ $\tau_{8\,1}$ $\tau_{8\,2}$ $\tau_{10\,7}$ $\tau_{7\,3}$ $\tau_{9\,1}$	$\tau_{3\,7}$ $\tau_{5\,5}$ $\tau_{6\,4}$	$\tau_{3\,8}$ $\tau_{4\,5}$ $\tau_{5\,6}$ $\tau_{6\,5}$ $\tau_{7\,4}$ $\tau_{8\,3}$ $\tau_{9\,2}$

in contrast to iterative algorithms that might require high computation times, so designers and developers have the possibility of applying all of them in order to check which one best suits the target system. Results show that there is no single algorithm that stands out clearly from the others. Therefore, the algorithm that best suits a certain configuration should be chosen from this collection, although an insight into the behaviour can be obtained by comparing the targeted scenario with the ones evaluated in this work. The proposed algorithms have been applied to an industrial use case, then they have been evaluated and their performance has been ranked by means of a synthetic representative application.

The following stage of this research includes the development of a heuristic algorithm for scheduling time-partition windows, in order to optimize the response times of the applications addressed in this work. Moreover, a more elegant strategy for solving the aforementioned ties in priority assignment could be explored. Finally, adequate task-to-partition strategies might be explored as part of a holistic optimization stage.

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.

Acknowledgements

This work was supported in part by the “Doctorados Industriales 2018” program from the University of Cantabria and the Spanish Government and FEDER funds (AEI/FEDER, UE) under grant TIN2017-86520-C3-3-R (PRECON-I4).

Annex

In this annex more details for reproducing the performance evaluation experiments from Section 5.2 are provided:

- Listing 1 shows the input code for generating synthetic e2e flows with TGFF.
- Fig. 6 shows the generated synthetic e2e flows, including activation periods of all e2e flows and worst-case execution times for all steps. Note that a 1/3 scaling factor is later applied to the generated execution times in order to produce feasible utilizations for the experiments.
- Table A.8 shows the Step-to-Processor mapping for the different number of processors evaluated.

```
tg_cnt 10
task_cnt 7 3
task_degree 3 3
period_mul 1,0.85,1.02
tg_write
eps_write


table_cnt 1
table_label Processor
type_attrib WCET 10 10
trans_write
```

Listing 1: Input code for TGFF

References

[1] Airlines Electronic Engineering Committee, Aeronautical Radio INC, Avionics application software standard interface. arinc specification 653-1, 2010, pp. 21401–27435, AERONAUTICAL RADIO, INC 2551.

[2] Aeronautical Radio INC, Arinc specification 664p7: Aircraft data network, part 7 - avionics full duplex switched ethernet (afdx) network, 2009, pp. 21401–27435, AERONAUTICAL RADIO, INC 2551.

[3] 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.

[4] A. Burns, R.I. Davis, A survey of research into mixed criticality systems, ACM Comput. Surv. 50 (6) (2017) 82.

[5] Safe4rail, URL https://safe4rail.eu/partners.

[6] 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.

[7] 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.

[8] A. Amurrio, E. Azketa, J. Javier Gutierrez, M. Aldea, J. Parra, A review on optimization techniques for the deployment and scheduling of distributed real-time systems, Rev. Iberoam. Autom. Inform. Ind. (in Spanish) 16 (3) (2019) 249–263.

[9] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1) (1973) 46–61.

[10] N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings, Hard real-time scheduling: The deadline-monotonic approach, IFAC Proc. Vol. 24 (2) (1991) 127–132.

[11] N.C. Audsley, Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times, Citeseer, 1991.

[12] J.J. Gutiérrez, M. González Harbour, Optimized priority assignment for tasks and messages in distributed hard real-time systems, in: Proceedings of Third Workshop on Parallel and Distributed Real-Time Systems, IEEE, 1995, pp. 124–132.

[13] R. Garibay-Martínez, G. Nelissen, L.L. Ferreira, L.M. Pinho, Task partitioning and priority assignment for distributed hard real-time systems, J. Comput. System Sci. 81 (8) (2015) 1542–1555.

[14] E. Azketa, J.P. Uribe, M. Marcos, L. Almeida, J.J. Gutierrez, Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems, in: 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2011, pp. 958–965.

[15] K.W. Tindell, A. Burns, A.J. Wellings, Allocating hard real-time tasks: an np-hard problem made easy, Real-Time Syst. 4 (2) (1992) 145–165, http://dx.doi.org/10.1007/BF00365407.

[16] R.I. Davis, L. Cucu-Grosjean, M. Bertogna, A. Burns, A review of priority assignment in real-time systems, J. Syst. Archit. 65 (2016) 64–82.

[17] Y. Zhou, S. Samii, P. Eles, Z. Peng, Scheduling optimization with partitioning for mixed-criticality systems, J. Syst. Archit. 98 (2019) 191–200.

[18] D. Tămaş-Selicean, P. Pop, Task mapping and partition allocation for mixed-criticality real-time systems, in: Dependable computing (PRDC), in: 2011 IEEE 17th Pacific Rim International Symposium on, IEEE, 2011, pp. 282–283, http://dx.doi.org/10.1109/PRDC.2011.42.

[19] R. Shah, Y.-H. Lee, D. Kim, Sharing i/o in strongly partitioned real-time systems, in: International Conference on Embedded Software and Systems, Springer, 2004, pp. 502–507.

[20] Y. Zhao, H. Zeng, The concept of maximal unschedulable deadline assignment for optimization in fixed-priority scheduled real-time systems, Real-Time Syst. 55 (3) (2019) 667–707.

[21] A. Guasque, H. Tohidi, P. Balbastre, J.M. Aceituno, J. Simó, A. Crespo, Integer programming techniques for static scheduling of hard real-time systems, IEEE Access 8 (2020) 170389–170403.

[22] J. Liu, Real-Time Systems, Vol. 48, Prentice Hall, 2000, p. 42.

[23] H. Kao, H. Garcia-Molina, Deadline assignment in a distributed soft real-time system, in: [1993] Proceedings. the 13th International Conference on Distributed Computing Systems, IEEE, 1993, pp. 428–437.

[24] J.M. Rivas, J.J. Gutierrez, J.C. Palencia, M.G. Harbour, Deadline assignment in edf schedulers for real-time distributed systems, IEEE Trans. Parallel Distrib. Syst. 26 (10) (2014) 2671–2684.

[25] J.M. Rivas Concepción, J.J. Gutiérrez García, et al., Interpretación de dos algoritmos edf on-line para la optimización de sistemas distribuidos de tiempo real, (In Spanish). URL http://hdl.handle.net/10902/17634.

[26] A. Saifullah, J. Li, K. Agrawal, C. Lu, C. Gill, Multi-core real-time scheduling for generalized parallel task models, Real-Time Syst. 49 (4) (2013) 404–435.

[27] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, C.D. Gill, Parallel real-time scheduling of dags, IEEE Trans. Parallel Distrib. Syst. 25 (12) (2014) 3242–3252.

[28] F. Guan, J. Qiao, Y. Han, Dag-fluid: A real-time scheduling algorithm for dags, IEEE Trans. Comput. 70 (3) (2020) 471–482.

[29] 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.

[30] M.G. 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.

[31] Object Management Group, Uml profile for marte: Modeling and analysis of real time embedded systems, version 1.1., OMG Document Formal.

[32] Green Hill Software, IntegrityRTOS, URL http://www.ghs.com/.

[33] 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.

[34] 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.

[35] 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.

[36] J.C. Palencia, J.J. Gutiérrez, M.González Harbour, On the schedulability analysis for distributed hard real-time systems, in: Proceedings Ninth Euromicro Workshop on Real Time Systems, IEEE, 1997, pp. 136–143.

[37] A. Srinivasan, J.H. Anderson, Optimal rate-based scheduling on multiprocessors, J. Comput. System Sci. 72 (6) (2006) 1094–1117.

[38] R.I. Davis, A. Burns, Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems, Real-Time Syst. 47 (1) (2011) 1–40.

[39] N. Serreli, G. Lipari, E. Bini, The distributed deadline synchronization protocol for real-time systems scheduled by edf, in: 2010 IEEE 15th Conference on Emerging Technologies & Factory Automation, ETFA 2010, IEEE, 2010, pp. 1–8.

[40] ERTMS/ETCS, European Rail Traffic Management System/European Train Control System release notes to system requirements specification, Subset 026 version 2.3.0.

[41] Ertms/etcs - subset-041: Performance requirements for interoperability, 2015.

[42] R.P. Dick, D.L. Rhodes, W. Wolf, Tgff: task graphs for free, in: Proceedings of the Sixth International Workshop on Hardware/Software Codesign, CODES/CASHE'98, IEEE, 1998, pp. 97–101.

[43] A. Sangiovanni-Vincentelli, M. Di Natale, Embedded system design for automotive applications, Computer 40 (10) (2007) 42–51.

[44] J. Lehoczky, L. Sha, Y. Ding, The rate monotonic scheduling algorithm: Exact characterization and average case behavior, in: RTSS, Vol. 89, 1989, pp. 166–171.