Interference Constraint Graph – A New Specification for Mixed-Criticality Systems

Pengcheng Huang, Pratyush Kumar, Nikolay Stoimenov, Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland
firstname.lastname@tik.ee.ethz.ch

Abstract

Current research in mixed-criticality systems assumes that any task of lower criticality levels can be dropped at anytime in order to guarantee the schedulability of tasks of higher criticality levels. However, in an industrial mixed-criticality system, tasks may interfere with each other only under certain scenarios. Currently a designer does not have any means to specify or control this. The paper proposes the Interference Constraint Graph (ICG) which specifies the allowed interferences between tasks. The new specification formalism generalizes and can easily express many of the existing mixed-criticality scheduling conditions. In spite of its generality, we show that standard fixed-priority scheduling can be efficiently applied. Experiments demonstrate that the ICG model enables systematic reduction of the number of tasks that can be dropped.

1. Introduction

The increasing pressure for reducing costs and physical resources has led to the design of embedded systems where applications with different safety-critical levels are integrated on a single computing platform. Typical examples are advanced driver assistance systems in the automotive industry and the Integrated Modular Architecture in the avionics industry [16]. Such systems have been called mixed-criticality. Since applications with different criticality levels share common hardware resources, they interfere with each other during runtime. For example, in an automotive adaptive cruise control system, heavy computation image-processing tasks may interfere with tasks responsible for controls of the engine. Traditionally, such interferences have been avoided by providing temporal and spatial isolation between the applications [18].

However, such strict isolation practices lead to underutilization of the hardware resources because applications rarely execute with their worst-case execution times (WCETs) and therefore, the temporal partitions are almost never used completely. Recently, a large body of research has been published which addresses this problem by providing heterogeneous scheduling guarantees to the applications according to their safety-criticality level. Such a strategy was initially proposed by Vestal [19] and subsequently developed by many others [5, 4, 8, 10, 3, 11, 9].

In the commonly used model of a mixed-criticality task set [19, 5, 4, 13], each task belongs to a particular safety-critical level, and it is also modeled for all criticality levels existing in the system by having a (possibly different) worst-case execution time for each level. The worst-case execution times of a task form a non-decreasing order when considered from the lowest to the highest criticality level. The reasoning behind this assumption is that the rigorousness and the conservatism of the methods used for deriving worst-case execution times increase with the increase of the considered criticality level.

Most of the recently proposed scheduling techniques make use of these different worst-case execution times by dropping tasks with a criticality level less or equal to χ, if a single task exceeds its worst-case execution time associated with level χ. This could mean that in an automotive system, tasks having lower criticality level such as image-processing and diagnostics are stopped when a single higher criticality task goes above a certain execution time threshold.

Such scenarios are rarely desirable in practice. A designer wants to be able to specify under what circumstances which tasks in which applications may be affected by an overrun of a task. For example, in an avionics Flight Management System, there are different tasks associated with the active, secondary, and temporary flightplans. Tasks for the active flightplan are certified at level B (when A is the highest, and E is the lowest), while tasks for the secondary and temporary flightplan are certified at level C. When a high criticality task executes above a certain execution time threshold, the system may stop only tasks responsible for the secondary, or for the temporary, but not for both even if they are of the same criticality level. Otherwise, the pilot does not have any means to insert a new flightplan and update the active one. A similar problem can occur if a high criticality task in one application can stop all low criticality tasks in other applications.

Most of the existing mixed-criticality scheduling strategies assume that they can stop a set of tasks based only on information about their criticality level without taking into account the actual functional requirements for these tasks. While this is commonly done in research

on schedulability, the required functionality of the safety-critical system may be compromised.

Currently mixed-criticality system designers do not have any means to specify which tasks can be affected if a certain task executes above a certain execution time threshold, and in what order the tasks can be affected. Moreover, stopping all tasks of a particular criticality level is often unnecessary as schedulability of the high criticality tasks can be guaranteed even when certain low criticality tasks continue executing. This is illustrated in the following example and confirmed by the experimental results in Section 6.

Example 1.1. Consider a simple dual-criticality task set with three periodic tasks and deadlines smaller or equal to periods. The parameters are given in Table 1 with time information in units of ms. C(LO) denotes the maximum execution time of a task in low criticality, C(HI) denotes the maximum execution time of a task in high criticality, and χ denotes the safety-criticality level of a task.

Table 1: Task parameters for Example 1

	period	*deadline*	χ	C(LO)	C(HI)
τ_1	10	10	high	3	5
τ_2	6	6	low	3	3
τ_3	15	11	low	1	1

This task set is schedulable by existing scheduling techniques, for example Vestal's approach [19] and EDF-VD [3]. However, it is not hard to observe that when τ_1 executes at C(HI), dropping only τ_3 is sufficient to guarantee the schedulability of τ_1. Therefore, the deadline of τ_2 can always be guaranteed under all runtime scenarios. □

This has been observed already in [15, 17], and the authors propose scheduling mechanisms that improve the guarantees given to low criticality tasks by not unnecessarily stopping them. However, these approaches do not have any means to specify in what order and which tasks may interfere with each other and under what runtime scenarios. A designer needs to be able to specify which task interferences are allowed by the functionality of the system and only they should be observable in any runtime scheduling scenario. Therefore, a new scheduling algorithm is needed that is able to take a task interference specifications and produce only schedules that meet this specification at runtime.

The motivation of this work is to propose a generalized specification of the allowed interferences in mixed-criticality task sets. The new specification overcomes the limitations described above. Moreover, it can easily be used to express a special case - the common practice that any high criticality task can interfere with all lower criticality tasks. Task sets specified with this new specification can be tested for schedulability with a fixed priority method which is presented and discussed. In particular, the contributions of this paper are as follows:

- A new specification for mixed-criticality systems is proposed which is driven by the needs of designers of industrial mixed-criticality systems. The allowed interferences between tasks are specified as an Interference Constraint Graph (ICG). In this graph, each task maps to a node, and an edge quantifies the interference between a pair of tasks.
- We then investigate how to design a schedule to satisfy a given ICG specification. We show that the Audsley's algorithm [1] can be used to design a fixed-priority scheduling policy which can meet a given ICG specification. This result resembles many previous results on mixed-criticality scheduling policies [19, 6]. It demonstrates that the more generic ICG specification is still susceptible to a simple and efficient scheduling policy.
- Experimental results based on randomly generated task sets show that in many cases stopping all low criticality tasks is not necessary during runtime in order to guarantee schedulability of high criticality tasks. In particular, we show that interferences that tasks suffer can be reduced systematically, and an algorithm that removes edges from an ICG graph is proposed.

Organization. Section 3 describes the used system model. In Section 4, we propose the Interference Constraint Graph (ICG), we discuss its relation to the standard mixed-criticality task model and its properties regarding schedulability. Section 5 presents how to schedule a task set specified with an ICG by fixed-priority scheduling. Experimental results in Section 6 show how to systematically reduce interferences between tasks. The paper concludes with Section 7.

2. Related Work

Vestal et al. presented in [19] a seminal work on mixed-criticality scheduling by proposing the current dominant mixed-criticality task model and a scheduling technique based on fixed task level priority. This approach was later on extended in [8] by formally proving that Vestal's approach is actually optimal among task level fixed-priority scheduling techniques. [6] further extends the work in [19] by introducing the idea of mode switch, where complete rejection of lower criticality tasks is assumed at each mode switch due to the overrun of critical tasks.

Job level fixed priority scheduling techniques were also proposed in the mixed-criticality scheduling domain. The OCBP (Own Criticality Based Priority) [4] is a job level priority extension of the original technique in [19]. This technique was extended in [12, 10] to perform online job level priority assignment for sporadic mixed-criticality task sets.

The migration of EDF to the domain of mixed-criticality scheduling was presented in [3]. In the EDF-VD approach, the authors tried to tune the deadlines of all critical tasks in a uniform way. With the new deadlines, they guarantee that whenever critical tasks overrun, they can still meet the original deadlines by rejecting all low criticality tasks. The same idea was later on extended

in [9] by allowing all critical tasks to tune deadlines non-uniformly, the authors claimed that this technique is so far the best in terms of mixed-criticality schedulability.

The authors in [17] identified runtime efficiency as a primary goal of mixed-criticality scheduling. An extension of task level fixed priority scheduling [19] was presented, that can determine which lower criticality tasks should be dropped when a high-criticality task exceeds its low-criticality execution time. Other scheduling techniques such as [3], propose a time-triggered scheduling technique based on OCBP. Different time-triggered schedules are computed offline, and the runtime system switches between those schedules in reaction to task runtimes.

However, as discussed in the previous section, the existing scheduling approaches implicitly assume that tasks can be stopped during runtime based only on information about their criticality level. A designer currently does not have any means to formally specify and control when such behaviors at runtime are correct.

In [18] the authors present a solution to map mixed-criticality task sets onto partitioned heterogeneous multiprocessors. They use a notion of separation graph to constrain the tasks that can share the same partition. However, this separation graph does not specify exactly when and how tasks can interfere with each other as considered in the current paper.

3. System Model

Given is a set of n sporadic tasks denoted as $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ executed on a uniprocessor. Each task τ_i is characterized by the following parameters:

- A minimum inter-arrival time $T_i \in \mathbb{R}^+$ between any two successive jobs.
- A relative deadline $D_i \in \mathbb{R}^+$ where $D_i \leq T_i$.

The actual execution time of a job of a task is not known until runtime when the job signals that it has finished executing. Based on this assumption, for the further discussion, we denote with $s(t)$ the runtime *scenario* at time t, $t \in \mathbb{R}^+$. A scenario is defined as the tuple $(c_1(t), c_2(t), \ldots, c_n(t))$, where $c_i(t)$ is the maximum of the run times of all jobs of task τ_i up to and including time t.

Each scenario $s(t)$ has at least one associated trace $tr(t)$ which leads to this scenario. The trace is characterized simply by the deadlines, arrival and finishing times of jobs up to and including time t.

Note that a scenario $s(t)$ and a corresponding trace $tr(t)$ depend on the arrival and execution times of jobs as well as on the selected scheduling strategy. In general, a task set and scheduling policy may exhibit at runtime many scenarios and associated traces.

We restrict the scope of the paper to consider only online scheduling strategies which cannot know the execution time of a job until running the job to completion.

4. Interference Constraint Graph

In this section, we propose the Interference Constraint Graph (ICG) as a specification of mixed-criticality task sets where the designer can specify the allowed interferences between tasks during runtime. We will discuss the syntax, semantics, and the properties of such a graph. We will present an example to illustrate the construction of such a graph specification from a commonly used mixed-criticality system model.

Definition 4.1. Interference Constraint Graph. An Interference Constraint Graph (ICG) is a directed graph $G(V, E, \sigma)$, where V is a set of vertices which corresponds to the set of tasks, i.e., $V \stackrel{def}{=} \tau$, E is a set of directed edges with $e_{i,j} \in E$ being an edge from τ_i to τ_j, i.e., $e_{i,j} = (\tau_i, \tau_j)$, and $\sigma(e)$ is a function $\sigma : E \rightarrow \mathbb{R}^+$ defined for all edges $e \in E$.

If there exists an edge $e_{i,j} = (\tau_i, \tau_j)$, then we say that task τ_i can *interfere with* task τ_j, meaning that if during runtime the execution time of a job of task τ_i exceeds $\sigma(e_{i,j})$ at time $t \geq 0$, then after time t, jobs of task τ_j do not need to be executed anymore. For all $e_{i,j} \in E$, we have $\sigma(\tau_i, \tau_j) \leq D_i$. Formally, we have the following schedulability definition.

Definition 4.2. ICG-Schedulability. Given a task set τ and a corresponding ICG G, the task set is ICG-schedulable with a particular scheduling strategy if in any scenario $s(t)$ and any corresponding trace $tr(t)$ that can be produced by the task set and the strategy, jobs of any task τ_j with absolute deadlines smaller or equal to t must meet their deadlines if for no task τ_i with $e_{i,j} \in E$ we have that:

$$c_i(t) > \sigma(\tau_i, \tau_j)\,.$$

In other words, ICG-Schedulability requires that deadlines of a task τ_j to be met in any scenario $s(t)$, only if each interfering task τ_i does not execute for more than $\sigma(\tau_i, \tau_j)$ in this scenario. If a task τ_j does not have any interfering tasks, i.e. there are no edges $e_{i,j}$, then jobs of task τ_j should meet their deadlines in any scenario $s(t)$.

Since we consider only online scheduling strategies, the above schedulability condition must be met for all t', $0 \leq t' \leq t$ in a scenario $s(t)$ and trace $tr(t)$.

Note that the ICG definition does not pose any restrictions on the topology of the graph. In particular, self-loops are quite useful as they can specify the maximum allowed execution time for a job of a task.

The graph does not need to be complete, i.e., there may be no edges between some vertices. In particular, the graph may have no edges at all, meaning that no tasks can interfere with each other.

Checking the consistency of ICGs is outside of the scope of the current paper.

4.1. ICG Representation of a Mixed-criticality Specification

Let us consider a mixed-criticality system model which is commonly used in literature [19, 5, 4, 13, 14, 2, 9]. Each task in the system is associated with a single safety-criticality level assigned from the set $\boldsymbol{\chi} \subseteq N^+$. The system may have an arbitrary number of criticality levels. In addition to the task parameters presented in Section 3, we have that each task τ_i is characterized by the following parameters:

- A safety-criticality level $\chi_i \in \boldsymbol{\chi}$, where a bigger value denotes a higher safety-critical level.
- A set of worst-case execution time estimates $\hat{C}_i = \{C_i(\chi) \mid C_i(\chi) \in R^+, \chi \in \boldsymbol{\chi}, C_i(\chi) = C_i(\chi_i) \forall \chi \geq \chi_i\}$. We assume that $C_i(\chi)$ is monotonically non-decreasing with increasing χ.

The criticality level of a scenario $s(t)$ is determined as the smallest value l ($l \in \boldsymbol{\chi}$) that satisfies the inequality $c_i(t) \leq C_i(l)$ for all i, $1 \leq i \leq n$. We assume that l always exists, otherwise the scenario is considered erroneous.

Conventionally, a mixed-criticality task set is considered *schedulable* with a particular scheduling strategy if in any runtime scenario $s(t)$ with criticality level l and corresponding trace $tr(t)$ that the task set and the strategy can produce, the strategy will give sufficient execution to each job that belongs to a task with $\chi_i \geq l$ and has a deadline smaller or equal to t, such that the job can signal finishing before its deadline. This schedulability definition has been widely used in literature [19, 5, 4, 13, 15, 10, 14, 2, 9, 17].

We can now demonstrate that the conventional schedulability condition can be easily expressed with an ICG. The following corollary follows from the definition of an ICG, and shows that it is possible to systematically construct an ICG for a mixed-criticality task set that is subject to the above schedulability condition.

Corollary 4.1. A task set which is subject to the above system model and schedulability condition can be specified with an ICG graph $G = (V, E, \sigma)$ as follows: (1) $V = \tau$. (2) For every task, τ_i, add edge $e_{i,i} = (\tau_i, \tau_i)$, with $\sigma(e_{i,i}) = C_i(\chi_i)$. (3) For every pair of tasks τ_i and τ_j with $\chi_i > \chi_j$, add an edge $e_{i,j} = (\tau_i, \tau_j)$ with $\sigma(e_{i,j}) = C_i(\chi_j)$.

The following example illustrates the results of the corollary and shows that the ICG specification is general enough to capture the commonly used mixed-criticality system model.

Example 4.1. Consider a task set of five tasks with three different safety-criticality levels. The parameters of the tasks are given in Table 2.

By Corollary 4.1, this commonly used task set specification can be represented with a corresponding ICG which is shown in Figure 1. Notice again that, the ICG specification is not limited to the standard specification, e.g., if τ_2 is not allowed to interfere with τ_4, it can be easily modeled by removing the edge between the two in the ICG. □

Table 2: Task parameters for Example 2

τ	T	D	χ	$C(3)$	$C(2)$	$C(1)$
τ_1	15	15	3	5	3	2
τ_2	10	10	2	2	2	1
τ_3	20	20	2	3	3	2
τ_4	30	30	1	4	4	4
τ_5	8	8	1	1	1	1

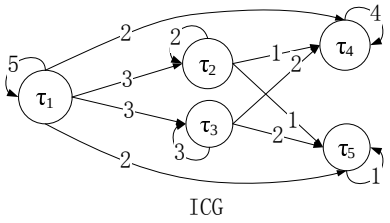

Figure 1: ICG representation for Example 2

Most mixed-criticality scheduling policies implicitly specify which criticality levels are stopped under certain runtime scenarios. On the other hand, the ICG specification explicitly defines, between pairs of tasks, that a task may be dropped if a certain condition is satisfied. The current proposal considers only one type for the condition: exceeding a certain execution time. However, it is not hard to see that this can be extended to include more general conditions such as deadline miss ratios, missing k deadlines in n consecutive instances, and others. On the other hand, one may not only consider dropping tasks, but also adjusting the Quality of Service provided to them, e.g., adjusting their periods or execution times at runtime. However, this is outside of the scope of the current paper.

4.2. Properties of ICG-Schedulability

Based on the ICG-Schedulability Definition 4.2, we can derive two useful properties with respect to the schedulability of an ICG.

The first one shows that whenever new interferences are introduced between tasks in an ICG-Schedulable task set, the ICG-Schedulability is maintained. Formally, this is specified as the Additivity property.

Property 4.1. Additivity. Given a task set τ specified with ICG $G = (V, E, \sigma)$ which is ICG-Schedulable. If we add more edges to G, i.e., we obtain $G' = (V', E', \sigma')$, where $V' = V$, $E \subset E'$, and $\forall e \in E : \sigma'(e) = \sigma(e)$, ICG-Schedulability is maintained, i.e., G' is also ICG-Schedulable.

Proof. Consider an ICG-Schedulable graph G where tasks τ_i and τ_j cannot interfere with each other, i.e., $(\tau_i, \tau_j) \notin E$. Let us consider a feasible trace tr where both tasks τ_i and τ_j execute, τ_i has finished before τ_j has arrived, and the execution time of τ_i is c_i.

Let us now consider the ICG G' with the added edge (τ_i, τ_j). Considering the value of $\sigma'(\tau_i, \tau_j)$, we have two

scenarios. In the first one, suppose that $\sigma'(\tau_i, \tau_j)$ is greater or equal to c_i, then the execution trace is not modified. In the second scenario, suppose that $\sigma'(\tau_i, \tau_j)$ is less than c_i. In this scenario, the trace tr is modified by dropping instances of τ_j when the run time of τ_i exceeds $\sigma'(\tau_i, \tau_j)$, which does not change the feasibility of the trace. □

On the other hand, one may show that removing edges from an ICG may make the task set unschedulable. In reality, a designer may be interested in removing interferences between tasks while still maintaining the ICG-Schedulability property satisfied. A heuristic algorithm that removes the maximum number of edges from a given ICG while keeping schedulability satisfied is presented in Section 6.

The next property shows that increasing existing interferences between tasks of an ICG-Schedulable task set will keep ICG-Schedulability property satisfied. In particular, increasing the interference that a task τ_j can suffer from another task τ_i, i.e., decreasing the value of $\sigma(\tau_i, \tau_j)$, will keep ICG-Schedulability satisfied. Formally, this is specified as the Monotonicity property.

Property 4.2. Monotonicity. Given a task set τ specified with ICG $G = (V, E, \sigma)$ which is ICG-Schedulable. If we decrease the values of the function σ, i.e., we obtain $G' = (V', E', \sigma')$, where $V' = V$, $E' = E$, and $\forall e \in E : \sigma'(e) \leq \sigma(e)$, ICG-Schedulability is maintained, i.e., G' is also ICG-Schedulable.

Proof. Consider an ICG-Schedulable graph G, where task τ_i can interfere with task τ_j with $\sigma(\tau_i, \tau_j)$. Let us consider a feasible trace tr where both tasks τ_i and τ_j execute, τ_i executes before τ_j, and the execution time of τ_i is c_i. Let us consider two cases. *Case 1:* $c_i > \sigma(\tau_i, \tau_j)$, in this case, when the run time of task τ_i exceeds $\sigma(\tau_i, \tau_j)$, task τ_j is dropped. Decreasing the value of $\sigma(\tau_i, \tau_j)$ will only cause task τ_j to be dropped earlier and the newly obtained trace is still feasible. *Case 2:* $c_i \leq \sigma(\tau_i, \tau_j)$, in this case, task τ_j is not dropped because of interferences from task τ_i. When decreasing $\sigma(\tau_i, \tau_j)$, we have two scenarios. Suppose that the new value is $\sigma'(\tau_i, \tau_j)$. If $\sigma'(\tau_i, \tau_j)$ is greater or equal to c_i, then the trace will not be modified. On the other hand, if $\sigma'(\tau_i, \tau_j)$ is less than c_i, then task τ_j will be dropped when the run time of τ_i exceeds $\sigma'(\tau_i, \tau_j)$, which does not change the feasibility of the trace. □

5. Fixed Priority Scheduling under ICG

The ICG specification as introduced in the previous section increases the expressiveness of existing mixed-criticality specifications by modeling explicitly the interferences between tasks. One intuition on the ICG semantics is that this more detailed specification would lead to increased complexity in the required scheduling strategy. In this section, we show that a fixed-priority preemptive scheduling strategy is applicable. Tasks are scheduled preemptively based on their statically assigned priorities.

Let us denote the set of higher priority tasks for a task τ_i as $hp(\tau_i)$. Further, define $\sigma(\tau_i, \tau_j) \stackrel{def}{=} +\infty$ when no directed edge from τ_i to τ_j exists in the ICG. We can adapt previous results on response time analysis for preemptive fixed priority scheduling [19, 3] and derive a schedulability test which takes into account interferences between tasks. An ICG-Schedulability test under fixed-priority preemptive scheduling is given in the following lemma.

Lemma 5.1. Given a task set τ and a corresponding ICG $G = (V, E, \sigma)$, τ is ICG-Schedulable if $\forall \tau_i \in \tau$, we have that $R_i \leq D_i$, where R_i is a fixed-point solution of the following recurrence relation:

$$R_i = \sigma(\tau_i, \tau_i) + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\}. \quad (1)$$

Proof Sketch. According to Definition 4.2, a task τ_i is guaranteed to meet its deadline only if each interfering task τ_j does not execute for more than $\sigma(\tau_j, \tau_i)$. We can consider two cases:

1. If a higher priority task τ_j can interfere with τ_i, it only needs to be assumed to execute for not more than $\min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\}$ because:
   - If $\sigma(\tau_j, \tau_j) \geq \sigma(\tau_j, \tau_i)$, τ_i is dropped if the runtime of τ_j exceeds $\sigma(\tau_j, \tau_i)$.
   - If $\sigma(\tau_j, \tau_j) < \sigma(\tau_j, \tau_i)$, τ_j cannot trigger the dropping of τ_i, since it is not allowed to execute for more than $\sigma(\tau_j, \tau_j)$.
2. If a higher criticality task τ_j cannot interfere with τ_i, then its worst-case execution time is assumed to be $\sigma(\tau_j, \tau_j)$ when testing the schedulability of τ_i.

Both cases can be compactly represented as $\min\{\sigma(\tau_j, \tau_j), \sigma(\tau_j, \tau_i)\}$. □

Let us illustrate now the results of Lemma 5.1 with a simple example. First, let us denote that a task τ_i has higher priority than τ_j with $\tau_i \succ \tau_j$.

Example 5.1. Consider a task set consisting of four tasks. Task parameters are given in Table 3 with $D_i = T_i, \forall i$. The corresponding ICG is shown in Figure 2.

Table 3: Task parameters for Example 5.1

τ	τ_1	τ_2	τ_3	τ_4
T	15	22	12	6

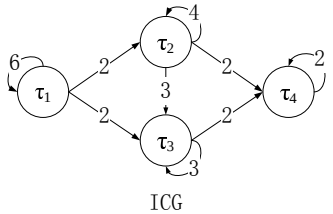

Figure 2: ICG for the task set in Example 5.1

In this example, task τ_1 is not allowed to interfere directly with τ_4. Hence, when considering the schedulability of τ_4, τ_1 must be assumed to take 6 units of execution

time, if the schedulability of τ_4 depends on the execution time of τ_1 (e.g. τ_1 is assigned a higher priority than τ_4). In contrast, τ_1 is allowed to interfere with τ_2, hence when testing the schedulability of τ_2, τ_1 only needs to be assumed executing for 2 time units.

Let us consider a particular priority ordering: $\tau_4 \succ \tau_1 \succ \tau_2 \succ \tau_3$. For the schedulability test of τ_3, following Lemma 5.1, the fixed-point of the following recursive equation gives the worst-case response time of τ_3:

$$R_3 = 3 + \left\lceil \frac{R_3}{15} \right\rceil \times 2 + \left\lceil \frac{R_3}{22} \right\rceil \times 3 + \left\lceil \frac{R_3}{6} \right\rceil \times 2. \quad (2)$$

Here, the execution times of τ_1 and τ_2 are assumed to be 2 and 3 units, since τ_3 does not need to be guaranteed when τ_1 and τ_2 execute more than 2 and 3 units, respectively. Task τ_4 is assumed to execute its worst-case execution time since it is not allowed to interfere with τ_3. Equation 2 has a fixed-point of 12, hence τ_3 is schedulable on the lowest priority level. In fact, this task set is ICG-Schedulable with the priority assignment given above. □

The schedulability test in Lemma 5.1 is sufficient but not necessary. The presented simple test does not take into account: 1) the interferences higher priority tasks suffer from low priority tasks, and 2) the interferences among the higher priority tasks. An exact schedulability test under fixed priority scheduling considering all interferences will need to combine all possible interference scenarios that can happen during runtime. However, this is outside of the scope of this paper.

5.1. Priority Assignment

Following Lemma 5.1, the schedulability of a task is independent of the relative priority orderings of its higher priority tasks. Based on this, one can immediately test which tasks can be assigned on the lowest priority level using relation (1). Once the task on the lowest priority level is determined, it is removed from the task set. We continue to find the task that can be assigned the second lowest priority level from the reduced task set, and so on. This process is recursively called until: 1) no task is left to be assigned a priority; or 2) at some step no task can be assigned on the current priority level (in this case the process fails to find a feasible schedule). This approach is known as the Audsley's algorithm [1].

It has been shown in [7] that Audsley's Optimal Priority Assignment algorithm is compatible with a schedulability test, if the following three conditions are satisfied:

1. The schedulability of a task is independent of the relative priority ordering of higher priority tasks.
2. The schedulability of a task is independent of the relative priority ordering of lower priority tasks.
3. If we swap the priorities of any two tasks of adjacent priority, then the task assigned higher priority cannot become unschedulable, if it was previously schedulable at the lower priority.

In particular, for a schedulability test complying with these three conditions, Audsley's algorithm is an optimal priority assignment approach, i.e. it will always find a feasible priority ordering as long as there exists one. This leads us to the following result on priority assignment for the ICG schedulability test presented in Lemma 5.1.

Theorem 5.1. Audsley's algorithm [1] is an optimal priority assignment approach with respect to the schedulability test given in Lemma 5.1.

Proof Sketch. We need to show that the test in Lemma 5.1 satisfies the three conditions described above.

1. The summation in relation (1) does not take into account the relative priority ordering of higher priority tasks.
2. Relation (1) does not consider any lower priority task because of the pre-emptive fixed priority strategy.
3. Consider two tasks τ_i and τ_j with priorities k and $k+1$, respectively. The upper bound on the response time of task τ_j cannot increase when it is assigned priority k, as the only change in the computation in relation (1) is the removal of task τ_i from the set of higher priority tasks.

□

6. Evaluation

We demonstrate in this section the advantage of ICG specification by showing that for mixed-criticality task sets with corresponding ICG specifications, a systematic reduction of interferences tasks suffer can be achieved. We quantitatively measure the reduction of interferences by the percentage of edges that remain after edge removal in an ICG. For this purpose, an algorithm that removes edges from a given ICG is proposed, and numerical results on randomly generated task sets are presented.

6.1. Random Task-set Generation

The first step of our experiments is to randomly generate task sets with corresponding ICG graphs. This is done by first generating standard sporadic mixed-criticality task sets with implicit deadlines. The corresponding ICG specifications of the generated task sets are produced using the results of Corollary 4.1.

For generating the standard mixed-criticality task-sets, we use a modified version of the random task generator as proposed in [14, 2]. The random task generator is controlled by the following parameters:

- $[U_-, U_+]$: utilizations of tasks are uniformly drawn from this range, $0 < U_- < U_+ \leq 1$;
- U_{bound}: the system utilization bound, which is defined as $U_{\text{bound}} \stackrel{def}{=} \max_{\chi \in \boldsymbol{\chi}} \{ \sum_{\{\tau_i | \chi_i \geq \chi\}} \frac{C_i(\chi)}{T_i} \}$;
- $[R_-, R_+]$: the ratio of worst-case execution times of tasks at two consecutive criticality levels ($\frac{\text{higher}}{\text{lower}}$) is uniformly drawn from this range, $1 \leq R_- < R_+$;

Algorithm 1: Interference Minimization

Input: $\tau, G(V, E, \sigma)$
Output: $G'(V, E', \sigma')$

```
for e ∈ E do
    calculate the minimum speedup factor s_e of the processor with only e in G, such that the task-set is ICG-Schedulable; set the weight of edge e (contribution to schedulability): w_e ← 1/s_e ;
end
sort edges E in decreasing order of their weights;
E' ← ∅;
while τ is not ICG-Schedulable under G'(V, E', σ') do
    add the first edge e in E to E' and remove e from E, σ'(e) ← σ(e);
end
return G'(V, E', σ') ;
```

- $[T_-, T_+]$: the periods of tasks are uniformly drawn from this range;
- $\pi : \boldsymbol{\chi} \rightarrow \mathbb{R}^{\geq 0}$: the probability of any task being of criticality level χ; $\sum_{\chi \in \boldsymbol{\chi}} \pi(\chi) = 1$.

6.2. Edge Removing

As shown by Property 4.1, adding new edges in the ICG specification will maintain ICG-Schedulability. On the other hand, removing edges while the task set stays schedulable can be used to reduce the interferences tasks suffer. We use the percentage of remaining edges in an ICG after edge removal as a quantitative measure of the improved expressiveness of the proposed specification.

The problem then is to maximally remove edges from a given ICG. We view the optimality criteria simply as removal of the maximum number of edges in the ICG specification while ensuring schedulability. In order to solve this problem, one intuition is that different edges in an ICG have different contributions to schedulability. Hence, we can first evaluate their weights in terms of such contributions. Once this is done, we can add edges in decreasing weights until ICG-Schedulability is satisfied. The proposed heuristic algorithm is shown in Algorithm 1.

Specifically, for each edge we measure the minimal speedup factor of the processor such that the task set is ICG-schedulable if only this edge exists in the ICG. Clearly, a lower speedup factor means a higher contribution to schedulability. Hence, the weight of an edge can be measured by the inverse of the corresponding speedup factor. The algorithm then tries greedily to add the minimum number of edges in the ICG by adding one edge at a time in decreasing order of their weights until the task set becomes schedulable. For calculation of the speedup factor under fixed priority scheduling in Algorithm 1, we use an existing technique as proposed in [8]. We omit here the details for space reasons.

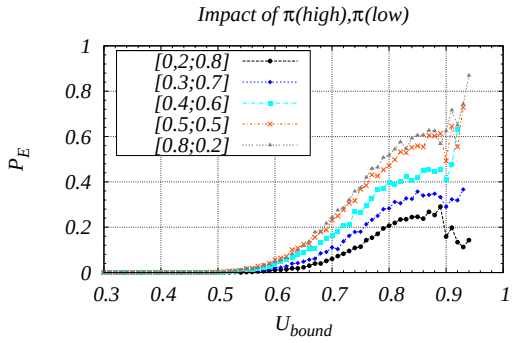

Figure 3: P_E measured with: $T_- = 4, T_+ = 16, U_- = 0.02, U_+ = 0.2, R_- = 1, R_+ = 4$.

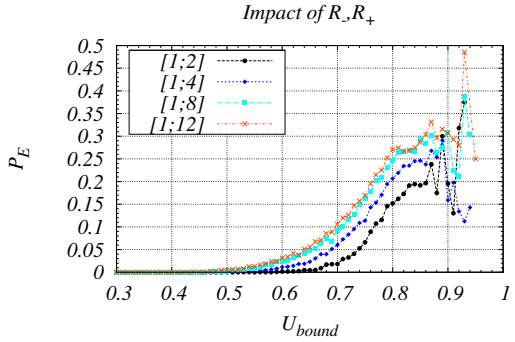

Figure 4: P_E measured with: $T_- = 4, T_+ = 16, U_- = 0.02, U_+ = 0.2, \pi(\text{high}) = 0.2, \pi(\text{low}) = 0.8$.

6.3. Results

We now apply Algorithm 1 to randomly generated mixed-criticality task sets, and we are interested in observing how many edges we can remove from the ICGs while maintaining schedulability. The experiments are extensively conducted on 1000 random task sets generated with specific group of controlling parameters.

Let us denote the set of schedulable task sets as Sched, then the average percentage of edges (P_E) *remaining* after edge removal for a set of task sets can be defined as follows:

$$P_E = \sum_{\text{Sched}} \frac{|E'|}{|E|} \Big/ |\text{Sched}| \ . \tag{3}$$

Effect of ratio of high criticality tasks. Here we consider randomly generated dual-criticality (high and low) task sets. In particular, we evaluate how the criticality-probability distribution (π) will affect P_E. As shown in Figure 3, for the typical case when the high criticality tasks only constitute a small portion of the dual-criticality task set, considerable number of edges in the ICG graphs can be removed. For instance, when $\pi(\text{high}) = 0.2$, we may be able to keep at most 30% of the edges in order to ensure schedulability. Intuitively if we increase $\pi(\text{high})$, P_E would increase since more interferences with the low criticality tasks are needed to accommodate the extra workloads of the high criticality tasks.

Effect of ratio of worst-case execution times. Here we show how the ratio of worst-case execution times between two consecutive criticality levels will affect P_E. The experiments are conducted on random dual-criticality

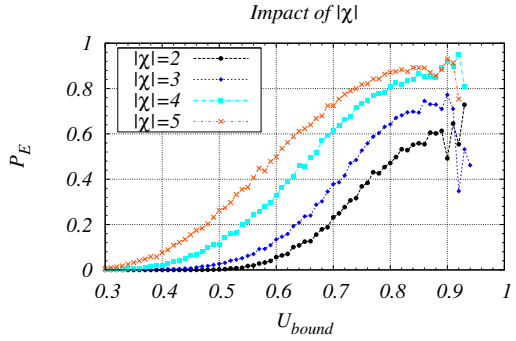

Figure 5: P_E measured with: $T_- = 4, T_+ = 16, U_- = 0.02, U_+ = 0.2, R_- = 1, R_+ = 4$.

task sets where $\pi(\text{high}) = 0.2$. Intuitively, for smaller worst-case execution time ratios between consecutive criticality levels, the interferences low criticality tasks suffer should be less: for the extreme case when the ratio is equal to 1, interference with low criticality tasks will not improve schedulability and all edges in the ICGs can be removed. As can be seen in Figure 4, for task sets generated with ratios drawn from a wider range, the percentages of remaining edges after edge removal are increased.

Effect of number of criticality levels. Here we present how the number of criticality levels will affect P_E. The larger the number of criticality levels in the system, the more interferences tasks can suffer. This is because the degree of uncertainties in worst-case execution times is higher. We evaluate the impacts of the number of criticality levels in our experiments by generating ICG graphs for random task sets with 2 to 5 criticality levels, with equal criticality probabilities. As shown in Figure 5, the percentage of edges we have to keep in the ICG graphs increases with increasing number of criticality levels. Furthermore, we can observe, for higher number of criticality levels, tasks need to interfere with each other even when the system utilization bound is low, e.g., for 4 criticality levels, interferences between tasks are needed to ensure schedulability already when $U_{bound} = 0.35$.

7. Conclusion

We propose the Interference Constraint Graph (ICG) for specifying mixed-criticality systems. The new specification models in detail the possible interactions between tasks and generalizes the standard mixed-criticality specification. It allows us to model real-life mixed-criticality systems more accurately. Properties of the ICG formalism are examined and a scheduling technique based on preemptive task level fixed priority is presented. Numerical results on randomly generated mixed-criticality task sets are presented to demonstrate the improved expressiveness of the ICG specification. Moreover, the new specification allows to use metrics that quantify the interferences between tasks and to use algorithms for their minimization.

References

[1] N. Audsley and Y. Dd. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, 1991.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS*, pages 145–154, 2012.

[3] S. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *RTSS*, pages 3–12, 2011.

[4] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *RTAS*, pages 13–22, 2010.

[5] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[6] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *RTSS*, pages 34–43, 2011.

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

[8] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems*, 46(3):305–331, 2010.

[9] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, pages 135–144, 2012.

[10] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 13–23, 2011.

[11] H.-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks. In *RTAS*, pages 23–32, 2012.

[12] H. Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 183–192, 2010.

[13] H. Li and S. Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *EMSOFT*, pages 99–108, 2010.

[14] H. Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *ECRTS*, pages 166–175, 2012.

[15] T. Park and S. Kim. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems. In *EMSOFT*, pages 253–262, 2011.

[16] P. J. Prisaznuk. Integrated modular avionics. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, pages 39–45, 1992.

[17] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *ECRTS*, pages 155–165, July 2012.

[18] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *RTSS*, pages 24–33, 2011.

[19] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, pages 239–243, 2007.