Real-Time Syst (2010) 46: 305–331
DOI 10.1007/s11241-010-9107-4

Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities

François Dorin · Pascal Richard · Michaël Richard · Joël Goossens

Published online: 23 September 2010

Abstract Safety-critical real-time standards define several criticality levels for the tasks. In this paper we consider the real-time systems designed under the DO-178B safety assessment process (i.e., Software Considerations in Airborne Systems and Equipment Certification). Vestal introduced a new multiple criticality task model to efficiently take into account criticality levels in the schedulability analysis of such systems. Such a task model represents a potentially very significant advance in the modeling of safety-critical real-time softwares. Baruah and Vestal continue this investigation, with a new scheduling algorithm combining fixed and dynamic priority policies. Another major design issue is to allow a system developer to determine how sensitive is the schedulability analysis to changes in execution time of various software components.

In this paper, we first prove that the well-known Audsley's algorithm is optimal for assigning priorities to tasks with multiple criticality levels. We then provide a proof on the optimality of Vestal's algorithm for optimizing the resource requirements to schedule tasks with multiple criticality levels. We then present a sensitivity analysis for multiple criticality tasks that is based on Bini et al. results on sporadic tasks.

Keywords Uniprocessor scheduling · Multiple criticality tasks · Sensitivity analysis

F. Dorin · P. Richard (✉) · M. Richard
LISI/ENSMA, Avenue Clément Ader, Téléport 2, BP 40109, 86961 Chasseneuil duPoitou, France
e-mail: pascal.richard@univ-poitiers.fr

F. Dorin
e-mail: francois.dorin@ensma.fr

M. Richard
e-mail: richardm@ensma.fr

J. Goossens
Département d'Informatique, Université Libre de Bruxelles, Boulevard du Triomphe – C.P. 212, 1050 Brussels, Belgium
e-mail: joel.goossens@ulb.ac.be

Table 1 The required design assurance level in the DO-178B

Level	Failure condition	Description
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload

1 Introduction

Avionic software standards define several criticality levels which define several levels of required confidence. For example, the RTCA DO-178B software standard (Authority 1992) defines 5 levels of criticality, denoted from A to E. A failure of a A-criticality task can have catastrophic results (i.e., crash of an airplane) whereas a failure of a E-criticality task has no effect on the safety of the airplane. The failure conditions, reported in Table 1, are categorized by their effects on the aircraft, crew, and passengers.

A way to achieve a higher assurance on worst-case execution time is to enforce time partitioning at run-time (Vestal 2007), as in the ARINC 653 standard (ARINC 1997). ARINC 653 is an Application Programming Interface that provides time partitioning among applications having different required Design Assurance Levels. The time line is defined as a set of time partitions. Each partition has a fixed predetermined amount of time. Each task (or a set of dependent tasks) is attached to a partition and a popular scheduling algorithm is executed on each partition. Since each partition has a fixed predetermined amount of allocated time, a partition cannot interfere with another one. In other words, a task, which belongs to a partition A, cannot interfere with a task which belongs to a partition B. Moreover, by affecting task with the same required level of confidence on the same partition, it is possible to ensure temporal isolation between tasks requiring different levels of confidence.

In practice, execution times of a recurring task are different from one execution to another. Schedulability analysis of real-time systems is based on the worst-case execution time (WCET). The execution time of a task never exceeds its WCET otherwise it is impossible to guarantee the system schedulability. Determining an exact WCET value for every task occurrence is a very difficult problem. So in practice, WCETs are upper bounds of execution requirements.

Since computing WCET is a complex problem, two different approaches can be considered:

– The first one is to allow some WCET exceedance (for instance, due to a optimistic approximation of the WCETs). Some models allow to take into account this kind of problem. For example, Bougueroua, in Bougueroua et al. (2007), introduced the notion of allowance to achieve this aim.

– The second one is to consider several levels of confidence for WCET. A high required confidence task have to never miss a deadline whereas a low required confidence task can miss some deadline sometimes without great consequences on the safety of the whole system. In such cases, the WCET of high required confidence tasks have to be evaluated with the maximum possible precision because an underestimated value can cause the task to miss a deadline, which can be very critical for the system, and an overestimated value can lead a feasibility test to conclude that a task is not feasible whereas no deadline miss can occur at run-time. So, the idea is to perform tight evaluation of the WCET for tasks having a high confidence level and to allow more approximate (i.e., average) evaluation for tasks with low confidence levels.

Next we will focus on the second approach defined in Vestal (2007). Vestal conjectures that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice" (Vestal 2007). For validating low criticality tasks, the worst-time observed during tests of normal operational scenarios might be used rather than sophisticated methods for computing WCET. In that way, resource requirement for scheduling multiple criticality systems can be highly decreased if different degrees of execution time assurance are considered, thus software development costs as well as computation power of hardware platforms can be reduced. Recently, in Vestal (2007), Baruah and Vestal (2008) is introduced a new formal model for task sets that considers *several* WCETs instead of a single one for each task. For a given task, these WCETs are computed according to several confidence levels.

Baruah et al. (2010) presented another multiple criticality model dealing with a collection of jobs (i.e., tasks are not periodically released). They proved that the schedulability problem is NP-hard, even if two criticality levels are considered. In Baruah et al. (2010) is presented on-line scheduling strategies for minimizing the makespan (i.e., the length of the schedule) and provides several competitive analysis results. This scheduling problem corresponds to existing problems taken from the domain of UAV (Unmanned Aerial Vehicles) used for defense reconnaissance and surveillance. In such real-world problems, the tasks are classified into two categories:

– flight-critical tasks: tasks require to ensure the UAV can fly,
– mission-critical tasks: tasks require to ensure the reconnaissance and surveillance objectives.

In order to allow the UAV to fly, it has to be certified by a civilian Certification Authorities. To deliver this certification, it is not necessary to take into account mission-critical tasks and it can be done using multiple criticality task model.

The second major issue in the design of real-time systems is the sensitivity analysis in order to determine how sensitive is the feasibility of the system according to changes in the execution times of software components (e.g., tasks). Popular feasibility analysis only provides little more than a yes or no answer and there is still a lack of flexibility as far as the designer is concerned. For overcoming such results, sensitivity analysis have been developed to estimate what changes task parameters can be made while preserving system feasibility.

All seminal works concern Rate Monotonic scheduling and execution time analysis to determine how may be increased WCET without causing deadlines to be missed (Lehoczky et al. 1989; Klein et al. 1993; Katcher et al. 1993; Punnekkat et al. 1997; Vestal 1994). These works have then been extended for analyzing more complex real-time systems as in Racu et al. (2008) and to other possible changes in task timing characteristics: WCET, deadlines or periods (Bini et al. 2008; Bini and Buttazzo 2009; Balbastre et al. 2009) for citing only some relevant recent works on these topics. We think that such an issue is quite important in time-partitioned systems for allocating time budgets for softwares to be executed in different time-partitions (e.g., developed by independent software developer teams) as well as for estimating possible software updates. To the best of our knowledge, such sensibility analysis has never been investigated for multiple criticality task sets.

1.1 This research

In this paper, we consider the seminal Vestal's multiple-criticality task model. In Baruah and Vestal (2008) is claimed that Vestal's algorithm (i.e., defined in Vestal 2007) is optimal. We provide a complete proof of a refined result showing that the original Audsley's algorithm is actually optimal for this kind of problem. We then analyze the sensitivity of system parameters from processor speed and task execution requirements:

- What is the required processor speed so that a multiple criticality task set is schedulable under Vestal's algorithm. Precisely, we show that Vestal's algorithm can be easily adapted to compute such a processor speed.
- What is the allowed variations of WCETs of a task so that it is still schedulable. For that purpose, we adapt the sensitivity analysis introduced by Bini et al. (2008) for analyzing multiple criticality task systems scheduled under a FTP scheduling policy.

Lastly, we extend our previous results by adding release jitters into the task model. We will give several motivations why release jitter definition must depends on task criticality levels.

1.2 Organization

The paper is organized as follows: Sect. 2 introduces the multiple criticality model as well as some known results we will use later. We prove, in Sect. 3, the optimality of the original Audsley's algorithm (Audsley 1991) for the kind of independent task systems with constrained-deadlines under fixed-task-priority scheduling policy. Section 4 deals with Vestal's algorithm, and the fact the returned schedule has the highest critical scaling factor among all the possible schedules. We provide also experimental results to compare the both algorithms. In Sect. 5, we performed sensitivity analysis on multiple criticality based systems followed by an example. In Sect. 6, we extend our results to take into account release jitters in the multiple-criticality model.

2 Task model, schedulability and sensitivity analysis

2.1 Task model

The model developed by Vestal (2007) is an extension of traditional sporadic task model that takes into account several assurance levels for defining worst-case execution times (e.g., RTCA DO-178B defines 5 criticality levels).

Definition 1 $L = \{L_1, \ldots, L_m\}$ is an ordered set of m design assurance levels, where L_m is the highest design assurance.

Each task will be assigned to a precise assurance level, but several execution times will be defined for each task, one per assurance level, in order to perform efficient schedulability analysis.

Definition 2 A sporadic multiple criticality task τ_i is defined by:

- a criticality level $L_i \in L$,
- a relative deadline D_i,
- a minimal period T_i between two successive releases,
- m worst-case execution times $C_i(\ell), \ell \in L$ that satisfies:

$$\begin{cases} C_i(\ell) = C_i(\ell+1) & \text{if } L_1 \leq \ell < L_i \\ C_i(\ell) \leq C_i(\ell+1) & \text{if } L_i \leq \ell \leq L_m \end{cases} \tag{1}$$

In the previous definition of a task τ_i, the condition $C_i(\ell) \leq C_i(\ell+1), L_i \leq \ell \leq L_m$ enforces Vestal's conjecture that "the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice." We add the condition $C_i(\ell) = C_i(\ell+1), L_1 \leq \ell < L_i$ in the definition of a multiple-criticality task for avoiding problems while performing schedulability analysis of multiple criticality task sets. This specific point will be developed in the next section. Notice that the multiple criticality task model is a strict generalization of the traditional sporadic task model (Baruah and Vestal 2008) in which all tasks are specified with identical degree of assurance.

In the remainder, we consider fixed-task-priority scheduling policies: every task has a fixed-priority that is never changed at run-time: π_i is an integer that defines the priority level assigned to τ_i. According to a fixed-priority scheduler, at any time, the highest priority task is selected for execution among ready ones. We assume without loss of generality that tasks are indexed according to priority levels (i.e., τ_1 is assigned to the highest priority level and τ_n to the lowest one). We also assume that tasks have constrained-deadlines (i.e., $D_i \leq T_i, i = 1, \ldots, n$). $u_i(\ell) \stackrel{\text{def}}{=} C_i(\ell)/T_i$ denotes the processor utilization factor of task τ_i and the system utilization factor is the sum of task utilization factors. Any task set having a utilization factor greater that 1 is said overloaded and it is well known that such system cannot be scheduled by any scheduling algorithm upon uniprocessor (e.g., Earliest Deadline First). We define the *worst-case response time* R_i as the longest delay between the arrival of a request and a completion among all τ_i executions.

Table 2 Task set that do not satisfies Definition 1

τ_i	T_i	D_i	L_i	$C_i(1)$	$C_i(2)$
1	5	5	1	1	2
2	5	5	2	2	5

2.2 Feasibility analysis

In Baruah and Vestal (2008) is presented a simple method for feasibility analysis of multiple criticality tasks. From a multiple criticality task set τ can be defined the *corresponding sporadic task system* τ' as follows:

Definition 3 From a multiple criticality sporadic task τ_i, we define a corresponding traditional sporadic task τ_i' defined by the 3-tuple (WCET, Relative Deadline, Period): $\tau_i'(C_i(L_i), D_i, T_i)$.

If no restrictions are placed upon the scheduling algorithms that may be used, feasibility analysis can be done using the following result:

Theorem 1 (Baruah and Vestal 2008) *A multiple criticality task set is feasible if and only if the corresponding traditional sporadic system is feasible.*

In Baruah and Vestal (2008) is presented a proof sketch of the previous theorem. But, the result is related to the basic underlying assumption that $C_i(\ell) = C_i(\ell+1), L_1 \leq \ell < L_i$. That is why we add that condition in Definition 2 in comparison with multiple criticality task definitions presented in Vestal (2007) and Baruah and Vestal (2008). Let us present a simple example exhibiting that such an assumption is required for using Theorem 1 and schedulability tests that will be used in the remainder. Let us consider the task set presented in Table 2. The classical feasibility analysis of the corresponding task set concludes that it is infeasible if all tasks are considered with the highest degree of assurance, since no algorithm can schedule a task set having the utilization factor is greater than 1:

$$\frac{C_1(L_1)}{T_1} + \frac{C_2(L_2)}{T_2} = 1.2 > 1 \tag{2}$$

However, from a multiple criticality schedulability analysis, this task system is schedulable when assigning the highest priority to the task τ_2 and the lowest priority to the task τ_1. The corresponding worst-case response time R_1 and R_2 of tasks τ_1 and τ_2, respectively:

$$R_1 = C_1(L_1) = 5 \leq D_1 \tag{3}$$

$$R_2 = C_1(L_2) + C_2(L_2) = 3 \leq D_2 \tag{4}$$

Thus, all deadlines seem to be met according to the feasibility test on the corresponding traditional sporadic system which is obviously impossible in any overloaded system. Next, we assume that task execution requirements must satisfy conditions defined in (1) for every multiple criticality task.

2.3 Scheduling algorithms

For these multiple criticality systems, in Vestal (2007) is provided two fixed-task-priority algorithms in order to schedule such systems: one based on period transformation based on Sha et al. (1986) and another based on a modification of the Audsley's priority assignment algorithm (Audsley 1991) (i.e., in fact a tie-breaker rule is introduced in Audsley's algorithm). In Baruah and Vestal (2008), these works were completed to establish a link between popular sporadic task systems and multiple criticality task systems. The corresponding sporadic task system is defined as the initial multiple criticality task set in which only the WCET corresponding to its critical confidence level is considered for every task. They proved an interesting property for the feasibility analysis: a multiple criticality sporadic task system is feasible if and only if the corresponding traditional sporadic task system is feasible (i.e., schedulable when temporal isolation of task executions is enforced by the operating system).

On-line scheduling algorithms can be classified into three different categories: fixed-task-priority (FTP, all occurrences of a given task have the same priority as for Rate Monotonic (RM) or Deadline Monotonic (DM) priority assignment policies); fixed-job-priority (FJP, every job has a fixed priority, but subsequent jobs of a given task can have different priorities—Earliest Deadline First (EDF) is such an algorithm); and lastly, Dynamic Priority (DP, the most general class of scheduling algorithms). For Liu and Layland's task systems, a classical result is that FTP scheduling algorithms are dominated by EDF (Liu and Layland 1973). That is to say, if a task system is schedulable by an FTP scheduling algorithm, then it is schedulable by EDF. This result does not hold for multiple criticality task system since Baruah and Vestal gave a counter-example of a task system which can be scheduled by FTP algorithm and cannot be scheduled by EDF (Baruah and Vestal 2008). In other words, FTP scheduling algorithms and EDF are incomparable.

To overcome the fact that EDF and FTP algorithms are not comparable, Vestal and Baruah proposed an hybrid-priority scheduling algorithm able to schedule any task system schedulable by Vestal's algorithm and/or by EDF, that is to say by any FTP algorithm or by EDF, since Vestal's algorithm is optimal for the FTP algorithm class. This hybrid-priority scheduling belongs to the class of the fixed job-priority (FJP) scheduling. A last result provided in Baruah and Vestal (2008) is that this hybrid-priority scheduling is not optimal in the FJP algorithm class.

We next detail Vestal's modified version of the Audsley's priority assignment algorithm (Audsley et al. 1993) in order to define a feasible fixed-priority schedule. Vestal's algorithm is claimed to be optimal in the category of the fixed-task-priority algorithms for independent task systems with constrained-deadlines (Baruah and Vestal 2008), but no proof is provided (we will see that it is correct, but the algorithm can be actually simplified).

The Audsley's algorithm is based on the following observation: the response time of a task depends only of the set of the higher priority tasks, and it is unnecessary to know the exact priority assignment. So, the principle of the Audsley's algorithm is to enumerate each priority level from the lowest to the highest. At each priority level is assigned the first task which is schedulable at this priority level (ties are broken arbitrarily). If there is at least one priority level with no task which can be assigned to it, then the task system is unschedulable using a fixed-task-priority algorithm.

Vestal modified this algorithm in the following way: instead of taking the first task which can be scheduled at a given priority level, Vestal's algorithm assigns the task with the highest critical scaling factor as defined in Lehoczky et al. (1989). This critical scaling factor Δ_i corresponds to the maximum scaling factor Δ for task computation times (i.e., factor by which all C_i can be multiplied without a deadline failure if a factor Δ less than or equal to Δ_i is considered). Thus, if Δ is increased beyond the factor Δ_i, a deadline will be missed. The utilization factor corresponding to the critical scaling factor defines the breakdown utilization of a task set. In the next definition, we precisely define the critical scaling factor associated to a task τ_i assigned to criticality level L_i (please notice that this definition is only a restatement of Lehoczky's definition presented (Lehoczky et al. 1989) in order to take into account that tasks have several execution requirements):

Definition 4 The critical scaling factor associated to a task τ_i is a threshold value of the scaling factor for all task execution times:

$$\Delta_i \stackrel{\text{def}}{=} \left[\min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j(L_i)}{t} \left\lceil \frac{t}{T_j} \right\rceil \right]^{-1} \tag{5}$$

where S_i is the set of scheduling points as defined in Lehoczky et al. (1989):

$$S_i \stackrel{\text{def}}{=} \left\{ kT_j \,|\, j = 1, \ldots, i-1; k = 1, \ldots, \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \cup D_i \tag{6}$$

We give an example of Vestal's assignment algorithm in Fig. 1. The upper table summarizes the task characteristics. The bottom table is a trace of Vestal's algorithm. For example, when we are looking for a task to assign at the priority level 3, we compute the critical scaling factor of each task, and we choose the one having the highest critical scaling factor which is, in this case, τ_3. So, we continue this process at the priority level 2 without forget to remove task τ_3. The task with the highest critical scaling factor at this level is τ_0, so τ_0 is assigned at the priority level 2. And so on.

The critical scaling factor of the system is given by the minimum of the critical scaling factor of each task when all tasks are assigned a priority. In this case, the critical scaling factor of the system is determined by the critical scaling factor of τ_3.

Baruah and Vestal (2008) claimed that this algorithm is optimal for scheduling independent task sets with constrained-deadlines under a fixed-task-priority policy without providing a complete proof. We will prove in Sect. 3 that the original Audsley's algorithm already is optimal for this schedulability analysis problem (i.e., without considering the critical scaling factors as a tie breaking rule). Furthermore, our proof is also valid for establishing that Vestal's algorithm is optimal. We will also show in Sect. 4 that Vestal's algorithm returns a schedule having the highest possible critical scaling factor.

Fig. 1 Vestal's priority assignment trace

τ_i	T_i	D_i	L_i	$C_i(1)$	$C_i(2)$
0	164	104	1	7	17
1	89	44	2	4	4
2	191	80	1	12	16
3	283	283	2	85	85

Priority	Trace
3	$\left.\begin{array}{l}\Delta_{\tau_0} = 0.928571\\ \Delta_{\tau_1} = 0.360656\\ \Delta_{\tau_2} = 0.740741\\ \Delta_{\tau_3} = 1.69461\end{array}\right\} \Rightarrow \pi_3 = 3$
2	$\left.\begin{array}{l}\Delta_{\tau_0} = 3.86957\\ \Delta_{\tau_1} = 1.18919\\ \Delta_{\tau_2} = 3.47826\end{array}\right\} \Rightarrow \pi_0 = 2$
1	$\left.\begin{array}{l}\Delta_{\tau_1} = 2.2\\ \Delta_{\tau_2} = 5\end{array}\right\} \Rightarrow \pi_2 = 1$
0	$\Delta_{\tau_1} = 11 \} \Rightarrow \pi_1 = 0$

$\Delta = \min_{\Delta_i} = 1.69461$

2.4 Sensitivity analysis

In this section we present known results on sensibility analysis of sporadic tasks. These works will be reused in Sect. 5 for developing the sensibility analysis of multiple criticality tasks.

From an historical point of view, the definition of the critical scaling factor was the first step for performing sensitivity analysis of independent task systems under Rate Monotonic scheduling policy. The idea of Lehoczky (1990) considers the changes of all execution times by a give factor up to a breakdown utilization (i.e., a deadline will be missed). Vestal (1994) reused the critical scaling factor definition for real-time systems where task deadlines are equal to their periods and blocking times are linear combination of resource access times. The idea is to introduce slack variables in each Lehoczky's inequality for converting them into equalities that are then trivially solved to yield the value of the slack variable. The obtained values are then used to derive upper bounds of the task execution times. The method is generalized to the case of tasks composed of several modules, by replacing the computation time C_i with a linear combination of the computation times of individual modules and reused for determining upper bounds of task WCET.

Previous approaches are based on analytical approaches for determining execution time limits. In Punnekkat et al. (1997), Racu et al. (2008) are used numerical algorithms for determining these bounds by combining a binary search (e.g., branch and bound, depth search, etc.) with a slightly modified version of a schedulability test (e.g., response time analysis). Enhanced stochastic algorithms based on evolu-

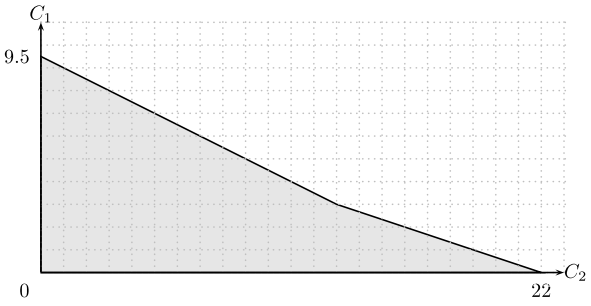

Fig. 2 Example of the representation of the $\mathbb{C}$-space of a system composed of 2 tasks, with $T_1 = D_1 = 9.5$ and $T_2 = D_2 = 22$

tionary search techniques (e.g. genetic algorithms) have been proposed in Racu et al. (2008) for performing multi-dimensional sensitivity analysis. While providing interesting results in practice for realistic systems, these approaches do not allow to define an exact characterization of feasible regions of task parameters.

Bini et al. (2008) performed a multi-dimensional sensitivity analysis which extends the Lehoczky's one. Two methods are described: one to perform schedulability in the $\mathbb{C}$-space (i.e., studying the modification of the execution time C_i of the tasks), and an other in the f-space (i.e., studying the modification of the period T_i of the tasks). This method allows to represent graphically these feasibility spaces of task parameters (i.e., Fig. 2 for an example of a $\mathbb{C}$-space graphically represented).

In the following, we focus on schedulability in $\mathbb{C}$-space since we are interested by the impact of using a model with several WCETs per task instead a single one. That method will be extended to multiple criticality tasks in Sect. 5.

The method to perform sensitivity analysis on the $\mathbb{C}$-space allows to choose the direction in which we want to perform the analysis, that is to say to choose which subset of tasks we want to study, and the weighting for each task.

The starting point of the method is the fact that a task system is schedulable if, and only if:

$$\max_{1\leq i\leq n} \min_{t\in S_i} \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil \leq t \tag{7}$$

or, in a vectored form:

$$\max_{1\leq i\leq n} \min_{t\in S_i} \mathbb{C}_i n_i(t) \leq t \tag{8}$$

where $\mathbb{C}_i$ is a vector of the i highest priority task $\mathbb{C}_i = (C_1, C_2, \ldots, C_i)$, and $n_i(t) = (\lceil \frac{t}{T_1} \rceil, \lceil \frac{t}{T_2} \rceil, \ldots, \lceil \frac{t}{T_{i-1}} \rceil, 1)$.

By replacing $\mathbb{C}_i$ by $\mathbb{C}_i + \lambda d_i$ in (8), we obtain (the complete proof can be found in Bini et al. 2008):

$$\lambda = \min_{i=1,\ldots,n} \max_{t\in \text{sched}(P_i)} \frac{t - n_i(t)\mathbb{C}_i}{n_i(t)d_i} \tag{9}$$

where λ is a scaling factor and $\text{sched}(P_i)$ is a subset of S_i.

The vector d_i corresponds to the studied direction. If we want to perform schedulability analysis on τ_k only, then d_i is equal to $(\underbrace{(0,\ldots,0,\overbrace{1}^{k^{th}\text{ element}},0,\ldots,0)}_{i\text{ elements}})$.

If we want to perform a sensitivity analysis on the whole system, then d_i must be equal to $\mathbb{C}_i$. The corresponding analysis leads to define the critical scaling factor of the system.

The schedulability in the $\mathbb{C}$-space is a generalization of the schedulability analysis introduced by Lehoczky (1990) in the sense that the computation of a critical factor for a single task or for the whole tasks system are particular cases of the Bini's method. Indeed, Bini's method allows to choose the direction on which the sensitivity analysis is performed. Thus, it is possible to study only one task, the whole task system or any subset of tasks of the system.

In this paper, one of our contributions is to adapt this algorithm to multiple criticality task systems (see Sect. 5) in the case of sensitivity analysis on the $\mathbb{C}$-space.

3 Optimality of Audsley's algorithm

We first prove that Vestal's algorithm is optimal. Note that neither in Vestal (2007) nor in Baruah and Vestal (2008) give such a complete proof. More precisely, we prove a refined result: the tie breaker that exploit critical scaling factors in Vestal's algorithm can be removed without losing optimality in the priority assignment process. By removing the tie breaker, Vestal's algorithm is equivalent to Audsley's algorithm. Thus, if there exists a feasible priority assignment for multiple criticality task, then Audsley's algorithm will find it.

Theorem 2 *The Audsley's algorithm is optimal for scheduling multiple criticality independent task systems with constrained-deadlines under a fixed-task-priority policy.*

To prove this theorem, we will use the lemmas described next:

Lemma 1 *When studying a specific task τ_i, we can consider corresponding task system instead of a multiple criticality task system, with the WCETs corresponding to the ones on critical level L_i, the criticality level of the studied task τ_i.*

Proof This lemma can be deduced from the definition of a multiple criticality task system. When we compute the worst-case response time (WCRT) of the task τ_i, we consider only the WCET of the criticality level of τ_i as we can see in the following equation, which is the modified version of the Joseph and Pandya's equation (Joseph and Pandya 1986) introduced by Vestal (2007) to compute the WCRT for multiple criticality systems:

$$R_i = \sum_{j=1}^{i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i) \tag{10}$$

Thus, when we are studying the task τ_i, we can consider only a classical task system with the WCETs corresponding to the WCET of the criticality level of τ_i, that is to say L_i. □

If we have a look to the task system given in Fig. 1, we can see that the critical scaling factor of task τ_1, when assigned at the priority 2, is greater than the critical scaling factor of the task τ_1, when assigned at the priority 3 (i.e., at a lower priority level). This intuitive result is summarized in the following lemma:

Lemma 2 *Let τ_i to be a task which has a critical factor of $\Delta_{i,j}$ when assigned of the priority j. If τ_i is assigned of the priority $j-1$ then the critical factor of τ_i for this priority verifies $\Delta_{i,j} < \Delta_{i,j-1}$.*

Proof For the following proof, we will consider a task τ_i which can be assigned at the level priority j or $j-1$. It is important to notice that the only difference between these two assignments is that the set of higher priority tasks, when τ_i is assigned at the priority level j contains one additional task than the set of higher priority tasks when τ_i is assigned at the priority level $j-1$. By convenience, we suppose the additional task to be τ_j, but since the task set of higher priority tasks are not ordered, it can be any higher priority task.

By definition, from Lehoczky (1990)

$$\Delta_{i,j} \stackrel{\text{def}}{=} \left[\min_{t \in S_{i,j}} \frac{1}{t} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil \right]^{-1} \tag{11}$$

$$\Delta_{i,j-1} \stackrel{\text{def}}{=} \left[\min_{t \in S_{i,j-1}} \frac{1}{t} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil \right]^{-1} \tag{12}$$

These definitions were just adapted to multiple criticality task systems, replacing classical WCET C_k by multiple criticality task WCET at level L_i which is equal to $C_k(L_i)$.

$S_{i,j}$ denotes the set of scheduling points for the task τ_i when assigned of the priority j. This set is defined by the following equation:

$$S_{i,j} \stackrel{\text{def}}{=} \left\{ kT_m | m = 1, \ldots, j; k = 1, \ldots, \left\lfloor \frac{D_i}{T_m} \right\rfloor \right\} \cup \{D_i\} \tag{13}$$

We were aware that Bini et al. introduced in Bini and Buttazzo (2004) a sufficient subset of scheduling points, but for our proof, we need to consider the set of all scheduling points.

So, according to (11) and (12), there exists t_j and t_{j-1} such as

$$\Delta_{i,j} = \left[\frac{1}{t_j} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil \right]^{-1} \tag{14}$$

$$\Delta_{i,j-1} = \left[\frac{1}{t_{j-1}} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_{j-1}}{T_k} \right\rceil \right]^{-1} \tag{15}$$

One can remark than $S_{i,j-1} \subset S_{i,j}$. So, we have two cases to take into account: $t_j \in S_{i,j-1}$ and $t_j \notin S_{i,j-1}$:

– If $t_j \in S_{i,j-1}$. It is obvious that:

$$\forall t, \quad \frac{1}{t} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil > \frac{1}{t} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil \tag{16}$$

So, if $t = t_j$ then:

$$\frac{1}{t_j} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil > \frac{1}{t_j} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil \tag{17}$$

Since $t_j \in S_{i,j-1}$ and t_{j-1} minimize $\frac{1}{t} \sum_{k=1}^{j-1} C_k(L_i) \lceil \frac{t}{T_k} \rceil$ (see definition of t_{j-1}, (15)), we have:

$$\frac{1}{t_{j-1}} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_{j-1}}{T_k} \right\rceil \leq \frac{1}{t_j} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil \tag{18}$$

Equations (17) and (18) give:

$$\frac{1}{t_{j-1}} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_{j-1}}{T_k} \right\rceil < \frac{1}{t_j} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil \tag{19}$$

That is to say:

$$\Delta_{i,j-1} > \Delta_{i,j} \tag{20}$$

– Now, we consider the case when $t_j \notin S_{i,j-1}$.

By definition, we have to notice that $D_i = \max(S_{i,j})$ and $D_i = \max(S_{i,j-1})$. Since $t_j \notin S_{i,j-1}$, we have $t_j \neq D_i$. So,

$$\exists t_k \in S_{i,j-1}, t_j < t_k \tag{21}$$

We can notice than $\sum_{k=1}^{j-1} C_k(L_i) \lceil \frac{t}{T_k} \rceil$ is a piecewise function and t_j is not a point of discontinuity since $t_j \notin S_{i,j-1}$, so:

$$\exists t_k \in S_{i,j-1}, \begin{cases} t_k > t_j \\ \sum_{k=1}^{j-1} C_k(L_i) \lceil \frac{t_j}{T_k} \rceil = \sum_{k=1}^{j-1} C_k(L_i) \lceil \frac{t_k}{T_k} \rceil \end{cases} \tag{22}$$

Moreover,

$$\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_j}{T_k} \right\rceil < \sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k} \right\rceil \tag{23}$$

So, (22) and (23) lead to:

$$\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k} \right\rceil < \sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k} \right\rceil \tag{24}$$

Since $t_k > t_j$, we have $\frac{1}{t_k} < \frac{1}{t_j}$. And, if we use (24), we have:

$$\frac{1}{t_k}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k} \right\rceil < \frac{1}{t_j}\sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k} \right\rceil \tag{25}$$

By definition of t_{j-1} (i.e., (15)), we have

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k} \right\rceil = \min_{t\in S_{i,j-1}} \frac{1}{t}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t}{T_k} \right\rceil \tag{26}$$

And then, since $t_k \in S_{i,j-1}$:

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k} \right\rceil \leq \frac{1}{t_k}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k} \right\rceil \tag{27}$$

If we combine (25) and (27), we obtain:

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k} \right\rceil < \frac{1}{t_j}\sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k} \right\rceil \tag{28}$$

That is to say,

$$\Delta_{i,j-1} > \Delta_{i,j} \tag{29}$$

We proved that in both cases ($t_j \in S_{i,j-1}$ and $t_j \notin S_{i,j-1}$), $\Delta_{i,j-1} > \Delta_{i,j}$. This prove the lemma. □

Now, we have the material to prove Theorem 1.

Proof of Theorem 1 Using Lemma 1, studying the schedulability of a multiple criticality task can be done by studying the schedulability of the equivalent task system on the criticality level of the studied task. And taking into account Lemma 2, the critical scaling factor of a task can only increase when we assign the task to a higher priority level. In other word, the interference due to higher priority tasks can only decrease.

Thus, if a task is schedulable at a priority level j, then it is schedulable when assigned of a higher priority. Since the hypothesis of the classical task model are also respected in the case of the multiple criticality task model, we can deduce that the Audsley's algorithm is also optimal for multiple criticality task systems. □

And having the previous theorem, we can easily state the following theorem:

Theorem 3 *The Vestal's algorithm is optimal to schedule a set of independent tasks with constrained-deadlines under a fixed-task-priority scheduling policy.*

Proof Since Vestal's algorithm is a particular case of Audsley's algorithm (i.e., task critical scaling factors are used for breaking ties), and since Audsley's algorithm is optimal due to Theorem 1, we can conclude that Vestal's algorithm is also optimal to schedule independent task systems with constrained-deadlines under fixed-task-priority policy. □

4 Processor speed

4.1 Maximization of the critical scaling factor

For multiple criticality task systems, Audsley's algorithm is optimal. But, if the system is not schedulable, then computing the minimum amount of supplementary processor speed so that the system becomes schedulable under a FTP assignment is an important issue for system designers.

Clearly, for sporadic tasks with constrained-deadlines, priority assignment (i.e., DM) and speed up factor computation are independent problems. We prove next that such a result is also valid for multiple criticality task system and furthermore that both problem can be solved simultaneously (i.e., the speed up factor can be computed in a greedy manner while performing the priority assignment).

Algorithm 1 presents an implementation of our algorithm in pseudo-code. It computes a priority assignment and a critical scaling factor Δ^*. The function $\Delta(\tau_i, \tau)$ computes the critical scaling factor of the task τ_i when the higher or equal priority task set is equal to τ.

If the critical scaling factor Δ^* is greater than 1, then it corresponds to the maximum factor by which we can divide the processor speed without having deadline failure. If $\Delta^* < 1$, then the initial task set is not schedulable and Δ^* corresponds to the minimum factor by which the processor speed must be accelerated to lead to a schedulable task system.

The main result (i.e., Theorem 4) will be based on the following property:

Lemma 3 *Let τ denote a task system and τ_i and τ_j be two tasks with τ_i having a higher priority than τ_j. If the critical scaling factor of the task τ_i at the priority level of τ_j is greater than the critical factor of the task τ_j at the same level, then inserting the task τ_i at the priority level of the task τ_j can only increase the critical factor Δ of the task system.*

Algorithm 1 Processor speed modulation and priority assignment

Require: τ^* = set of tasks to schedule
Ensure: Δ^* = maximum scaling factor
Ensure: $\tilde{\tau}$ = scheduled task system

```
τ ⇐ τ*
τ̃ ⇐ ∅
for j from n to 1 do
  τ_Vestal = ∅
  for τ_A ∈ τ do
    if τ_Vestal = ∅ then
      τ_Vestal ⇐ τ_A
      Δ* = Δ(τ_A, τ)
    else
      if Δ(τ_Vestal, τ) < Δ(τ_A, τ) then
        τ_Vestal ⇐ τ_A
      end if
    end if
  end for
  π(τ_Vestal) ⇐ j
  τ ⇐ τ − {τ_Vestal}
  τ̃ ⇐ τ̃ ∪ {τ_Vestal}
  if Δ(τ_Vestal, τ) < Δ* then
    Δ* ⇐ Δ(τ_Vestal, τ)
  end if
end for
```

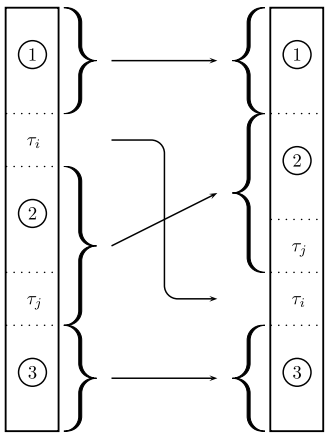

Fig. 3 Scheme of the transformation

Proof We will use an interchange argument to prove the result. Figure 3 represents the basis of the transformation. Each zone corresponds to the following:

– Zone 1 is composed of tasks with higher priority than task τ_i,
– Zone 2 is composed of tasks with intermediate priority, that is to say with lower priority than τ_i but higher priority than τ_j,
– Zone 3 is composed of tasks with lower priority than the task τ_j.

If we study the evolution of the critical scaling factor of each task when performing the transformation, we can observe that:

– The critical factor of tasks in Zone 1 are not modified by the priority modifications of tasks with lower priority,
– The critical factor of tasks in Zone 3 are not modified by the modifications of the priority order of tasks of higher priority,
– The critical factor of tasks in Zone 2 can only increase due to Lemma 2.

And if we perform the transformation, it is, by hypothesis because task τ_i has a higher critical scaling factor at priority level of τ_j than τ_j.

In other words, in all the cases, the critical scaling factor of each task can be either unchanged or increased, except for task τ_i. But by assumption the new critical scaling factor of task τ_i is greater than the old critical scaling factor of task τ_j. The result follows. □

Now, using this lemma, it is easy to prove the following theorem.

Theorem 4 *Vestal's algorithm returns a priority assignment with the greatest critical scaling factor of tasks* (*i.e.*, *minimum speed up factor if the system is not schedulable under a unit-speed processor*).

Proof Let τ denote the task system. This task system is composed of n tasks, $\tau_1, \ldots, \tau_n$, and each task is assigned to a priority. To prove the result, we build-up Vestal's schedule from τ using Lemma 3. The method is straightforward: we are looking for the task having the highest critical scaling factor at the priority level n among the tasks having a priority higher or equal to n. Then, we insert this task to this level. Due to Theorem 2, the critical scaling factor of the new task system τ' is greater or equal to the critical scaling factor of Δ. We repeat this operation, replacing τ by τ' and looking for the task to insert at the level priority $n-1$, and so on until the studied priority task level is equal to 1.

By this way, we construct a new schedule from the initial one, which is the same than this one produced by Vestal's algorithm because in both cases, the same task selection is performed. Since the transformation used can only increase the critical scaling factor of the initial task set τ and since the initial task set τ can represent any task set, we can conclude that the task set resulting of Vestal's algorithm has the highest possible critical scaling factor for fixed-task-priority policy. This proves the Theorem 4. □

So, Vestal's algorithm, by providing a schedule with the highest possible critical scaling factor, has a great interest since it offers a simple way to define the minimum processor requirement so that a multiple criticality task set is schedulable.

4.2 Experimental results

We compute the scaling factor of a system of tasks scheduled by Vestal's algorithm and by Audsley's algorithm in order to compare computational power of required computing platform. For this purpose, we performed a statistical analysis, based on the following characteristics:

- the number of tasks was chosen in the set {10, 20, 30}
- the utilization factor of the whole systems vary from 0.05 to 0.95 using a step of 0.05
- we perform 10 000 runs per configurations

The generation of task systems follow the following procedure:

- the utilization factor of each task is generated using UUniFast algorithm of Bini and Buttazzo (2004).
- the period T_i of the task τ_i is randomly generated in the range [100, 3000].
- the deadline of the task τ_i are set to its period T_i.
- the criticality level L_i of each task τ_i is randomly chosen in the set $\{1, 2, 3, 4, 5\}$.
- the execution time $C_i(L_i)$ at the level L_i of the task τ_i is computed from the utilization factor and the period of the task.
- the execution times for lower criticality levels (i.e., $\ell < L_i$) are set to $C_i(L_i)$.
- the execution times for higher criticity levels (i.e., $\ell > L_i$) are set randomly in the range $[C_i(L_i), D_i]$ so that constraints defined in (1) are both satisfied.

The column gain corresponds to the relative gain in percent of the critical scaling factor obtained by Vestal's algorithm against the Audsley's one. It corresponds to the following formula:

$$\text{gain} = \frac{\Delta_v - \Delta_a}{\Delta_a} * 100 \tag{30}$$

where Δ_a corresponds to the critical scaling factor when the Audsley's algorithm is considered and Δ_v corresponds to the Vestal's one. Since we have shown (i.e., in Theorem 4) that the schedule obtained by Vestal's algorithm has the highest critical scaling factor for a given task system, we are sure that the following condition is always satisfied:

$$\Delta_a \leq \Delta_v \tag{31}$$

We can see in Fig. 5 that the gain is more important when the utilization factor of the task system is low. It can be easily explained: for task systems with a small workload, there exists a lot of schedulable configurations and Audsley's algorithm picks one of them without any additional criteria. Thus, since the number of feasible solution is huge, the solution computed by Audsley's algorithm can be far away from the optimal critical factor obtained by Vestal's algorithm.

Whereas for a high loaded task systems, there exists few feasible schedules and then the chance of Audsley's algorithm to pick up an nearly optimal schedule is high. Thus, for such task systems, the gain will be small.

U	10 tasks		20 tasks		30 tasks	
	Gain	Identical schedule	Gain	Identical schedule	Gain	Identical schedule
0.05	807.06%	0.11%	751.18%	0.00%	726.76%	0.00%
0.10	351.53%	0.44%	305.64%	0.04%	275.53%	0.00%
0.15	209.14%	1.32%	173.22%	0.23%	153.29%	0.14%
0.20	142.76%	2.46%	115.85%	0.85%	97.63%	0.68%
0.25	102.97%	3.99%	83.05%	1.82%	67.56%	1.19%
0.30	77.70%	5.56%	62.30%	2.68%	50.60%	2.04%
0.35	59.39%	7.41%	47.78%	3.92%	38.54%	2.85%
0.40	45.71%	8.86%	37.81%	4.72%	30.46%	3.03%
0.45	35.71%	11.26%	29.75%	5.76%	24.80%	4.25%
0.50	27.65%	12.96%	23.72%	7.45%	20.80%	5.01%
0.55	21.16%	16.16%	18.90%	8.52%	16.31%	6.65%
0.60	15.81%	19.66%	14.96%	9.31%	12.76%	6.44%
0.65	11.45%	23.94%	10.82%	13.21%	9.72%	8.66%
0.70	7.75%	30.16%	7.65%	14.27%	7.22%	10.06%
0.75	4.41%	41.63%	4.75%	19.40%	4.58%	12.13%
0.80	2.03%	57.83%	2.16%	34.56%	1.89%	27.24%
0.85	0.88%	70.58%	0.82%	55.98%	0.46%	58.33%
0.90	0.29%	82.03%	0.12%	71.42%	0.00%	100.00%
0.95	0%	100.00%	0.00%	100.00%	0.00%	100.00%

Fig. 4 Comparison results

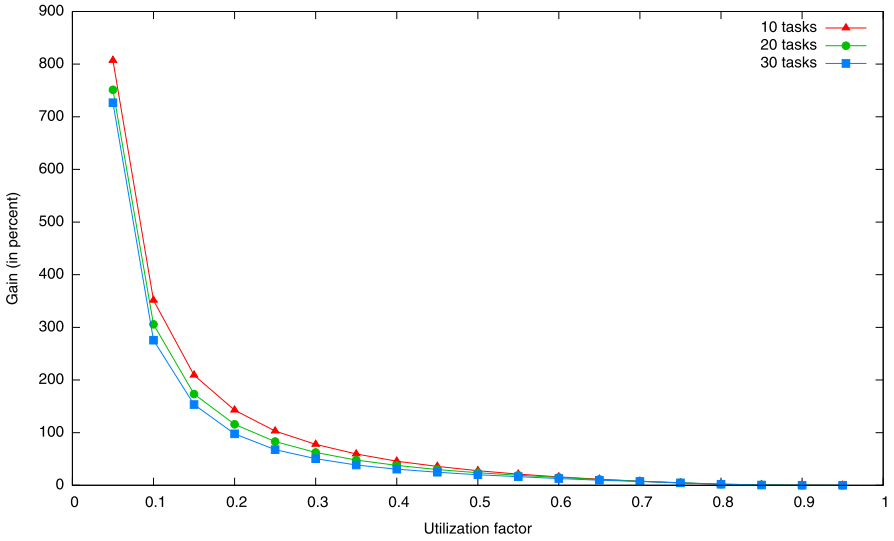

Fig. 5 Evolution of the gain depending of the number of tasks

These results are also confirmed with the ratio of identical schedules obtained by Vestal's algorithm and Audsley's algorithm in Fig. 4:

– for low loaded task systems, the chance to have the same schedule is low: less than 1% for ten tasks and an utilization factor of 0.10,
– for high loaded task systems, the chance to have the same schedule is high: greater than 50% for ten tasks and a utilization factor of 0.8.

If we want to reduce the power consumption of low loaded systems, then we can see we can reduce the processor speed by at least a factor of 2 in average if the utilization factor of the system is less than 0.2. Even if the gain decreases with the load of the system, it is still noticeable for medium loaded system. However, for systems with high utilization factor, the gain is insignificant (less than 2% for systems with a load greater than 0.8).

We also check the influence of the number of criticality levels on the gain (cf. Fig. 6). We can notice that the number of criticality levels has a weak influence. It is just noticeable that, for systems with very low utilization factor (less than 0.2), higher the number of criticality levels is, then lower the gain of Vestal's algorithm on Audsley's one is.

5 Sensitivity analysis on WCET

We next adapt the Bini et al. sensitivity analysis (i.e., initially developed for classical real-time task systems Bini et al. 2008) to multiple criticality task systems. We only focus to the sensitivity analysis in the $\mathbb{C}$-space, since the multiple criticality task model distinguishes from classical sporadic task systems by considering a set of WCETs for every task.

5.1 Sensitivity analysis in the $\mathbb{C}$-space

We extend the sensitivity analysis in the $\mathbb{C}$-space by analyzing tasks at the same critical level. Instead of having one λ in the studied direction d, we define one λ_ℓ per criticality level ℓ.

$$\lambda_\ell \stackrel{\text{def}}{=} \min_{\substack{i=1,\ldots,n \\ L_i=\ell}} \max_{t \in \text{sched}(P_i)} \frac{t - n_i(t)\mathbb{C}_i(\ell)}{n_i(t)d_i} \tag{32}$$

A particular attention must be focused on the modified C_i. Indeed, the modifications can break a basic assumption of multiple criticality system expressed by (1) (a complete example is detailed in the next section). In practice, such a problem can be easily solved by setting (1) as a constraint in Bini et al. sensitivity analysis method. Precisely, it is necessary to normalize execution requirements of every task so that the assumption on execution time stated in the task model is respected (i.e., (1)).

For that purpose every time that (1) is not satisfied:

$$\exists \ell, \quad C_i(\ell) > C_i(\ell+1) \tag{33}$$

then, we assign the value of C_i at criticality level $\ell+1$ to the C_i at criticality level ℓ

$$C_i(\ell) \leftarrow C_i(\ell+1) \tag{34}$$

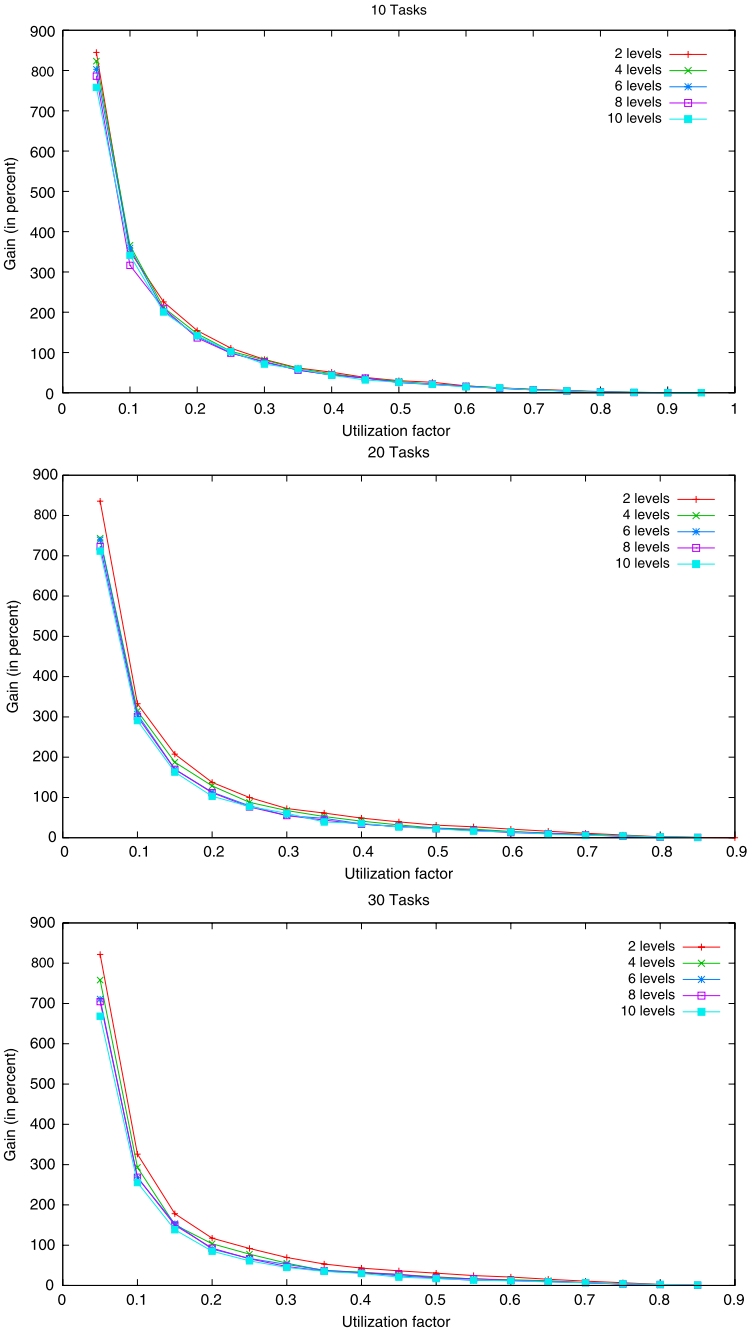

Fig. 6 Influence of the number of criticality levels

Table 3 Example of a multiple criticality tasks system

τ_i	T_i	D_i	L_i	$C_i(1)$	$C_i(2)$
τ_1	137	65	1	9	29
τ_2	286	139	2	86	86
τ_3	248	168	1	32	160

5.2 Example

After this simple normalization step, Bini et al. sensitivity analysis can be easily performed. Let study the example of multiple criticality task system where characteristics are given in Table 3.

And let focus on the task τ_2 on which we will perform the sensitivity analysis. Bini et al. (2008) showed that when the schedulability analysis is performed only on a single task, (32)[1] can be rewritten in:

$$\delta C_k^{\max} = \min_{i=k,\ldots,n} \max_{t \in \text{sched}(P_i)} \frac{t - n_i(t)\mathbb{C}_i}{\lceil \frac{t}{T_k} \rceil} \tag{35}$$

To apply the sensitivity analysis, scheduling points must be computed. Bini and Buttazzo (2004) use these recursive definition to find them:

$$\begin{cases} \text{sched}(P_i) \stackrel{\text{def}}{=} \mathcal{P}_{i-1}(D_i) \\ \quad \mathcal{P}_0(t) \stackrel{\text{def}}{=} \{t\} \\ \quad \mathcal{P}_i(t) \stackrel{\text{def}}{=} \mathcal{P}_{i-1}(\lfloor \frac{t}{T_i} \rfloor T_i) \cup \mathcal{P}_{i-1}(t) \end{cases} \tag{36}$$

Applying (36) to τ_2 and τ_3 to have their scheduling points give us the following sets:

$$\text{sched}(P_2) = \{T_1, D_2\} \tag{37}$$

$$\text{sched}(P_3) = \{T_1, D_3\} \tag{38}$$

So, we can now compute the critical scheduling factor for task τ_2 and τ_3 (a trace of the computations can be found in Table 4):

$$\delta C_2 = \max(22, -5) = 22 \tag{39}$$

$$\delta C_3 = \max(10, 32) = 32 \tag{40}$$

[1] We do not use the Bini's notation $\Delta C_k^{\max}$ to avoid possible confusion with the critical scaling factor Δ. We use $\delta C_k^{\max}$ instead.

Table 4 Trace of the δC_i

τ_2		τ_3	
t	δC_2	t	δC_3
137	22	137	10
139	−5	168	32

Table 5 Sensitivity analysis before normalization step

τ_i	T_i	D_i	L_i	$C_i(1)$	$C_i(2)$
τ_1	137	65	1	9	29
τ_2	286	139	2	118	108
τ_3	248	168	1	32	160

Table 6 Sensitivity analysis after normalization step

τ_i	T_i	D_i	L_i	$C_i(1)$	$C_i(2)$
τ_1	137	65	1	9	29
τ_2	286	139	2	108	108
τ_3	248	168	1	32	160

Having these δC_i, we can now compute the critical scaling factor per criticality level:

$$\delta C_2^{\max}(1) = \min_{i=1,\ldots,n \wedge L_i=1}(\delta C_i) = \min(\{\delta C_3\}) \tag{41}$$

$$\delta C_2^{\max}(2) = \min_{i=1,\ldots,n \wedge L_i=2}(\delta C_i) = \min(\{\delta C_2\}) \tag{42}$$

If we apply the modification to the task system, we obtain the system shows in Table 5. We can easily see that the basic hypothesis of multiple criticality task system (1) is not satisfied for task τ_2 since $C_2(1) > C_2(2)$. So, we have to perform a normalization step, as describe in the previous section.

After normalization, we obtain the task system describes in Table 6. Figure 7 shows the multiple criticality $\mathbb{C}$-space for the task τ_2, that is to say the possible value for $C_2(1)$ and $C_2(2)$ in order to satisfy (1).

6 Release jitters

6.1 Traditional release jitter

Audsley et al. (1993) introduced the notion of release jitter for assigning priorities to static priority tasks. The aim of the release jitter is to model a delay between the arrival of a task and its release (i.e., overhead introduced by the real-time kernel or delay introduced by input communications). Release jitters needs to be considered as soon as the assumption that a task is always released as soon as it arrives. So, taking

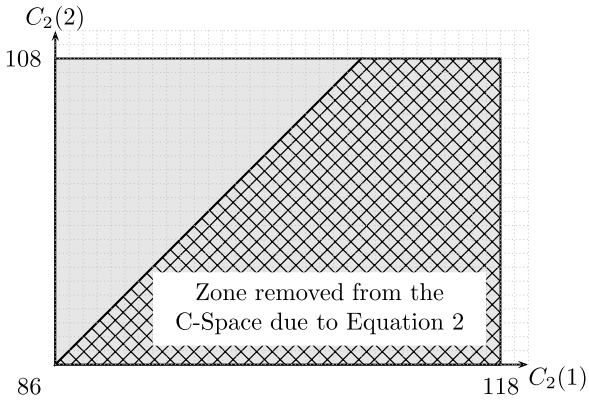

Fig. 7 Multiple criticality $\mathbb{C}$-space for task τ_2

into account task release jitters allow us to have tasks which may be delayed after its arrival in the system. The release jitter of a task τ_i is denoted by J_i.

We will see that introducing the release jitters in multiple-criticality task systems will be as simple as adding the release jitters in the Liu and Layland seminal task model. It is well known that introducing release jitters for computing worst-case response time is obtained by modifying (10) as follows:

$$R_i = \sum_{j=1}^{i} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j(L_i) \tag{43}$$

6.2 Multiple criticality jitter

In the following, we made no assumption about the link between task release jitters and task criticality levels. Precisely, release jitters model overheads due to the real-time kernel that can be evaluated using different methods for estimating the corresponding worst-case execution times. Thus, we can consider that the release jitter is no more a constant but can depends on the criticality levels of tasks. For low critical tasks, release jitters can be ignored, and then set to zero whereas for high critical tasks, a non-null release jitter can be assigned with the same constraints that for worst-case execution times:

$$J_i(\ell) \leq J_i(\ell + 1), \quad \forall \ell \tag{44}$$

In this case, to take into account this multiple criticality task jitter, formulas as extended in the same as it is done for classical task systems. The main difference is the criticality of the studied task that must be taken into account. As a consequence, (10) becomes:

$$R_i = \sum_{j=1}^{i} \left\lceil \frac{R_i + J_j(L_i)}{T_j} \right\rceil C_j(L_i) \tag{45}$$

where $L_j(\ell)$ denotes the release jitter of the task τ_j for the criticality level ℓ.

7 Conclusion and future work

In this article, we investigate the multiple criticality task scheduling model introduced in Vestal (2007) and Baruah and Vestal (2008). Such task model represents a potentially very significant advance in the modeling of safety-critical real-time systems. We first formally proved the original Audsley's algorithm is already optimal in the class of fixed-task-priority algorithm for scheduling independent task systems with constrained-deadlines, and as a consequence that the tie breaking rule used in Vestal's algorithm is not useful for assigning fixed-task-priority to multiple criticality tasks.

Moreover, we performed two kind of sensitivity analysis: we first showed that Vestal's algorithm can be extended to compute the minimum processor speed so that a multiple criticality task set is schedulable. For that purpose, Lehoczky's critical scaling factor is used as a tie breaker at each task priority level. Our experimental study to compare Vestal's algorithm and Audsley's algorithm shown that the gain of using Vestal's algorithm is higher when the load of the task system is low.

We also show how to adapt the sensitivity analysis in the $\mathbb{C}$-space originally developed by Bini et al. (2008) for the case of multiple criticality task systems. Such an extension allows to analyze a subset of tasks. From a practical point of view, it is particularly useful to analyze all tasks belonging to the specific critical level.

We complete our analysis by introducing release jitters in the multiple-criticality task model. We shown that the previous works of Audsley (1993) can be easily adapted to the multiple criticality model. For that purpose, we introduced the notion of multiple criticality jitters (i.e., introduced delays are evaluating according to criticality levels).

7.1 Future work

Lehoczky (1990) performed a sensitivity analysis for a single task and the whole task system. Bini et al. (2008) extend this method to allow a task sensitivity analysis according to a given direction. Future works may concern the sensitivity analysis of a task system to draw the $\mathbb{C}$-space without considering any particular direction.

Most of critical systems are in fact large scale dependable distributed computing systems. Thus, introducing release jitters in the task model is a first step to extend the presented works to the analysis of distributed systems. Furthermore, we must extend the system model (i.e., computing platform and task model) to cope with dependability issues.

References

ARINC: Avionics application software standard interface. ARINC Spec **653** (1997)

Audsley N (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report YCS 164, Dept. Computer Science, University of York, UK

Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ (1993) Applying new scheduling theory to static priority pre-emptive scheduling. Softw Eng J 8:284–292

Authority F (1992) Software considerations in airborne systems and equipment certification. RTCA Inc: EUROCAE

Balbastre P, Ripoll I, Crespo A (2009) Period sensitivity analysis and d-p domain feasibility region in dynamic priority systems. J Syst Softw 82:1098–1111

Baruah S, Li H, Stougie L (2010) Scheduling for certifiability in mixed criticality systems. In: The real-time technology and applications symposium. Pre-publication

Baruah S, Vestal S (2008) Schedulability analysis of sporadic tasks with multiple criticality specifications. In: ECRTS '08: Proceedings of the 2008 Euromicro conference on real-time systems. IEEE Comput Soc, Washington, pp 147–155

Bini E, Buttazzo G (2004) Schedulability analysis of periodic fixed priority systems. Comput IEEE Trans 53(11):1462–1473

Bini E, Buttazzo G (2009) The space of EDF deadlines: the exact region and a convex approximation. Real-Time Syst 41:27–51

Bini E, Di Natale M, Buttazzo G (2008) Sensitivity analysis for fixed-priority real-time systems. Real-Time Syst 39(1–3):5–30

Bougueroua L, George L, Midonnet S (2007) Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled fp and edf. In: The second international conference on systems (ICONS'07)

Joseph M, Pandya P (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Katcher D, Arakawa H, Strosnider J (1993) Engineering and analysis of fixed priority scheduler. IEEE Trans Softw Eng 19(9):920–934

Klein M, Ralaya T, Pollak B, Obebza R, Harbour M (1993) Guide to rate monotonic analysis for real-time systems. Kluwer Academic, Norwell

Lehoczky J (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of the 11th real-time systems symposium, pp 201–209

Lehoczky JP, Sha L, Ding Y The rate monotonic scheduling algorithm-exact characterization and average case behavior. In: Real-time system symposium (1989)

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Punnekkat S, Davis R, Burns A (1997) Sensitivity analysis of real-time task sets. In: Advances in computing science, LNCS, vol 1345, Springer, Berlin, pp 72–82

Racu R, Hamann A, Ernst R (2008) Sensitivity analysis of complex embedded real-time systems. Real-Time Syst 39:31–72

Sha L, Lehoczky JP, Rajkumar R (1986) Solutions for some practical problems in prioritized preemptive scheduling. In: Proceedings IEEE real-time systems symposium. IEEE Comput Soc, Los Alamitos, pp 181–191.

Vestal S (1994) Fixed-priority sensitivity analysis for linear compute time models. IEEE Trans Softw Eng 20(4):308–317

Vestal S (2007) Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: RTSS '07: proceedings of the 28th IEEE international real-time systems symposium. IEEE Comput Soc, Washington, pp 239–243

François Dorin received the engineering degree from ENSMA (National Engineering School for mechanics and Aerotechnics) located at Poitiers and the M.Sc. degree from University of Poitiers, France in 2007. He currently works toward the Ph.D. degree in the Laboratory of Applied Computer Science at the University of Poitiers and ENSMA, France. His research interests include real-time scheduling, on-line algorithms and combinatorial optimization.

Pascal Richard is a professor in Computer Science at the University of Poitiers, France. He received the Ph.D. degree in Computer Science from the University of Tours (France) in 1997. His research interests include realtime systems, scheduling theory, on-line algorithms and combinatorial optimization.

Michaël Richard is Assistant Professor at the National Engineering School for mechanics and Aerotechnics (ENSMA) and University of Poitiers since September 2003. Michaël Richard received his Ph.D. degree in computer science in November 2002 form the University of Poitiers, France. He teaches algorithm and programming, object oriented Design and programming, Operating Systems, and Embedded Systems. His main research interests are in real-time scheduling theory, real-time distributed systems and embedded systems.

Joël Goossens is Associate Professor at the Université Libre de Bruxelles, since October 2006. Joël Goossens received his M.Sc. degree in computer science in 1992, his M.Sc. degree in network and management in 1993 and his Ph.D. degree in computer science in 1999, all from the Université Libre de Bruxelles, Belgium. He teaches algorithms and programming, operating systems and real-time scheduling. His main research interests are presently in real-time scheduling theory, real-time operating systems and embedded systems.