Uniprocessor Schedulability and Sensitivity Analysis of Multiple Criticality Tasks with Fixed-Priorities

François DORIN, Pascal RICHARD, Michaël RICHARD
LISI
ENSMA - Université de Poitiers
1 rue Clément Ader, BP 40109,
86961 Chasseneuil du Poitou Cedex, France
{dorinfr, richardp, richardm}@ensma.fr

Joël GOOSSENS
Computer Science Department
Université Libre de Bruxelles
Boulevard du Triomphe - C.P. 212
1050 Bruxelles, Belgium
joel.goossens@ulb.ac.be

Abstract

Safety-critical real-time standards define several criticality levels for the tasks (e.g., DO-178B - Software Considerations in Airborne Systems and Equipment Certification). Classical models do not take into account these levels. Vestal introduced a new multiple criticality model, to model more precisely existing real-time systems, and algorithms to schedule such systems. Such task model represents a potentially very significant advance in the modeling of safety-critical real-time systems. Baruah and Vestal continues this investigation, with a new algorithm under fixed and dynamic priority policies.

In this paper, we provide some results about the optimality of Vestal's algorithm and analyze an interesting property of this algorithm. We also adapt sensitivity analysis developed by Bini et al. for multiple criticality systems.

1. Introduction

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, the used WCET is an upper bound 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 [7], 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.

Some software standards define several criticality levels which define several levels of required confidence. For example, the RTCA DO-178B software standard [3] defines 5 levels of criticity, 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 take into account these different levels of confidence is to perform a time partitioning between the different software applications which allows to enforce temporal isolation of tasks like described, for example, in the ARINC 653 standard [1]. ARINC 653 is an Application Programming Interface that provides time partitioning among applications having different required Design Assurance Levels. The timeline 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 classical 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

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.

Table 1. The required Design Assurance Level in the DO-178B.

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.

Another way to take into account different levels of confidence was discussed by Vestal in a recent paper [13]. Vestal introduced a new formal model for representing real-time task sets. This model, based on the consideration of several WCETs instead of a single one, allows to require more or less confidence depending on the criticality of the tasks. Baruah and Vestal gave this definition in [4]: 'the more confidence one needs in a task execution time bound, the larger and more conservative that bound tends to be in practice'.

In [13], Vestal provided two fixed priority algorithms in order to schedule such systems: one based on period transformation [12] and another based on the Audsley's algorithm [2]. In [4], these works were completed by Vestal and Baruah. They established a link between classical 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 occurences 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 - the 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 [11]. 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. In other words, FTP scheduling algorithms and EDF cannot be compared.

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 [4] is that this hybrid-priority scheduling is not optimal in the FJP algorithm class.

This Research. In this paper, we give a modest step in the study of multiple criticality task systems. Precisely, we provide a complete proof that the original Audsley's algorithm already is optimal for this kind of problem. We then analyse 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 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 in [6] for analyzing multi-criticality task systems scheduled under a FTP scheduling policy.

Organization. The paper is organized as follow: Section 2 introduces the multiple criticality model as well as some known results we will discuss later. We prove, in Section 3, the optimality of the original Audsley's algorithm [2] for the kind of independent task systems with constrained-deadlines under fixed priority 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. In Section 5, we performed sensitivity analysis on multiple criticality based systems followed by an example.

2. Task Model and known results

2.1. Task Model

The model developed by Vestal in [13] is based on the classical Liu and Layland's one [11]. Let τ denote a task system composed of n tasks. Each task τ_i for $i = 1, \ldots, n$ is composed of:

- a worst-case execution time C_i, which corresponds to the required processor time per instance of the task,
- a period T_i, which is the minimum inter-arrival separation time between two consecutive instances of the task τ_i,
- a relative deadline D_i, which corresponds to the maximum autorized amount of time between the activation and the end of an instance of the task τ_i.

For the multiple criticality model, Vestal introduced the following parameters:

- A WCET function $C_i : \mathbb{N}^+ \rightarrow \mathbb{R}^+$, which specifies the WCET for different criticality levels. We can notice that C_i is no more a constant for a given task but a function. Thus, the WCET for the criticality level ℓ is denoted by $C_i(\ell)$.
- A criticality level L_i, $L_i \in \mathbb{N}^+$, which specifies the required confidence for the task τ_i. By convention, it is assumed that the level 1 is the lowest critical level.

In addition to these parameters, we introduce the priority π_i of a task τ_i, which allows to determine which task have to be executed at a given time: the task with the highest priority is executed first. By convention, a high numerical value for π_i denotes a low priority task. Thus, the task having a priority equal to 0 has the highest priority.

In this paper, we assume that tasks have constrained-deadlines (i.e., $D_i \leq T_i$ for each task). $u_i(\ell) \stackrel{\text{def}}{=} C_i(\ell)/T_i$ 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 DP scheduling algorithm. Moreover, it is supposed that tightness of WCET increases according to critical levels. Thus, for all task τ_i and for all criticality level l, the following relation is verified [13]:

$$C_i(\ell) \leq C_i(\ell + 1) \tag{1}$$

From a multi-criticality task τ set can be defined the *corresponding sporadic task system* τ' as follows: to every multi-critical task τ_i is defined a corresponding sporadic task $\tau'_i(C_i(L_i), D_i, T_i)$. The key assumption to enforce (i.e., Theorem 1 [4]) that a multiple criticality sporadic task system is schedulable if and only if the corresponding traditional sporadic task system is feasible, it must be enforced that:

$$\forall i \in [1, n], \forall j \in [1, L_i], C_i(j) = C_i(L_i) \tag{2}$$

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

Table 2. Example of violation

Let us consider the task set presented in Table 2. The classical schedulability analysis of the corresponding task set concludes that it is unschedulable, since no DP algorithm can schedule a task set having the utilization factor is greater than 1:

$$\frac{C_1(L_1)}{T_1} + \frac{C_2(L_2)}{T_2} = 1.2 > 1 \tag{3}$$

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 of τ_1 and τ_2 are:

$$Tr_1 = C_1(L_1) = 5 \leq D_1 \tag{4}$$

$$Tr_2 = C_1(L_2) + C_2(L_2) = 3 \leq D_2 \tag{5}$$

Thus, all deadlines seem to be met which is obviously impossible in any overloaded system. Remember that task execution requirements must satisfy the Equation (2) for every multiple criticality task.

2.2. Known results

Scheduling algorithm In [13], Vestal introduced a modified version of the Audsley's algorithm [2]. The Vestal's algorithm is optimal in the category of the fixed priority algorithms for independent task systems with constrained-deadlines [4].

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-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 i.e., factor which corresponds to the maximum factor by which we can multiply the WCET of the task without the task τ_i missed a deadline [9]. We recall the previse definition of the critical scaling factor of a system because it will be reused hereafter:

$$\Delta^* \stackrel{\text{def}}{=} \left[\max_{1 \leq i \leq n} \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \right]^{-1} \tag{6}$$

where S_i is the set of scheduling points as defined in [10]:

$$S_i \stackrel{\text{def}}{=} \left\{ kT_j | j = 1, \ldots, i; k = 1, \ldots, \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \quad (7)$$

This critical scaling factor corresponds to the maximum factor by which we can multiply all C_i of the tasks without a deadline failure. If we consider tasks separately, the critical factor of a task can be defined as follows:

$$\Delta_i \stackrel{\text{def}}{=} \left[\min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \right]^{-1} \quad (8)$$

In [4], Baruah and Vestal claimed that this algorithm is optimal for scheduling independent task sets with constrained-deadlines under a fixed priority policy without providing a complete proof.

We show next that the original Audsley's algorithm already is optimal for this kind of problem (i.e., without considering the critical scaling factors as a tie braking rule) and, as a consequence, that Vestal's algorithm also is optimal. We also show that Vestal's algorithm returns a schedule having the highest possible critical scaling factor.

We give an example of Vestal's assignment algorithm in Figure 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 of a priority. In this case, the critical scaling factor of the system is determined by the critical scaling factor of τ_3.

Schedulability analysis. In [9], the critical scaling factor is a basic sensitivity analysis on independent task systems under fixed priority policy.

In [6], Bini et al. performed a 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 spaces (i.e., Figure 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. So, readers interested by schedulability in f-space can report themself to the original paper from Bini et al. [6].

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	$\left.\Delta_{\tau_1} = 11\right\} \Rightarrow \pi_1 = 0$

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

Figure 1. Vestal's priority assignment trace

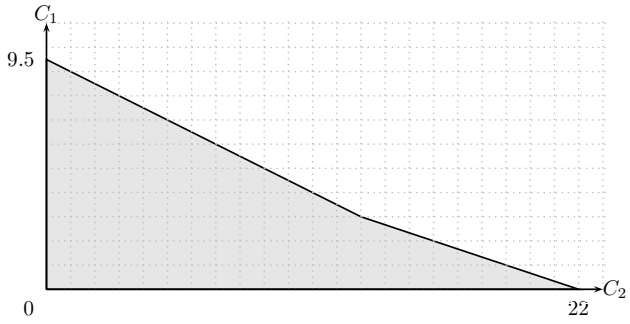

Figure 2. Example of the representation of the $\mathbb{C}$-space of a system composed of 2 tasks, with $T_1 = D_1 = 9.5$ and $T_2 = D_2 = 22$

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:

$$\max_{1 \le i \le n} \min_{t \in S_i} \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil \le t \quad (9)$$

or, in a vectorial form:

$$\max_{1 \le i \le n} \min_{t \in S_i} \mathbb{C}_i n_i(t) \le t \quad (10)$$

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

By replacing $\mathbb{C}_i$ by $\mathbb{C}_i + \lambda d_i$ in the Equation 10, we

obtain (the complete proof can be found in [6]):

$$\lambda = \min_{i=1,\ldots,n} \max_{t \in sched(P_i)} \frac{t - n_i(t)\mathbb{C}_i}{n_i(t)d_i} \quad (11)$$

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

The vector d_i correspond 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 in [9] 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 Section 5) in the case of sensitivity analysis on the $\mathbb{C}$-space.

3. Optimality of Audsley's algorithm

Our first contribution corresponds to the following result:

Theorem 1. *The Audsley's algorithm is optimal for scheduling multiple criticality independent task systems with constrained-deadlines under a fixed-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 [8] introduced by Vestal in [13] to compute the WCRT for multiple criticality systems:

$$Tr_i = \sum_{j=1}^{i} \left\lceil \frac{Tr_i}{T_j} \right\rceil C_j(L_i) \quad (12)$$

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 Figure 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 [9]

$$\Delta_{i,j} \stackrel{\text{def}}{=} \left[\min_{t \in S_{i,j}} \frac{1}{t} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil \right]^{-1} \quad (13)$$

$$\Delta_{i,j-1} \stackrel{\text{def}}{=} \left[\min_{t \in S_{i,j-1}} \frac{1}{t} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t}{T_k} \right\rceil \right]^{-1} \quad (14)$$

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:

$$S_{i,j} \stackrel{\text{def}}{=} \left\{ kT_m | m = 1,\ldots,j; k = 1,\ldots,\left\lfloor \frac{D_i}{T_m} \right\rfloor \right\} \cup \{D_i\} \quad (15)$$

We were aware that Bini et al. introduced in [5] a sufficient subset of scheduling points, but for our proof, we need to consider the set of all scheduling points.

So, according to Equations 13 and 14, there exists t_j and t_{j-1} such as

$$\Delta_{i,j} = \left[\frac{1}{t_j} \sum_{k=1}^{j} C_k(L_i) \left\lceil \frac{t_j}{T_k} \right\rceil \right]^{-1} \quad (16)$$

$$\Delta_{i,j-1} = \left[\frac{1}{t_{j-1}} \sum_{k=1}^{j-1} C_k(L_i) \left\lceil \frac{t_{j-1}}{T_k} \right\rceil \right]^{-1} \quad (17)$$

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:

$$\forall t, \frac{1}{t}\sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t}{T_k}\right\rceil > \frac{1}{t}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t}{T_k}\right\rceil \tag{18}$$

So, if $t = t_j$ then:

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

Since $t_j \in S_{i,j-1}$ and t_{j-1} minimize $\frac{1}{t}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t}{T_k}\right\rceil$ (see definition of t_{j-1}, Equation 17), we have:

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k}\right\rceil \leq \frac{1}{t_j}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_j}{T_k}\right\rceil \tag{20}$$

Equations 19 and 20 give:

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k}\right\rceil < \frac{1}{t_j}\sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k}\right\rceil \tag{21}$$

That is to say:

$$\Delta_{i,j-1} > \Delta_{i,j} \tag{22}$$

- 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,

$$\exists t_k \in S_{i,j-1}, t_j < t_k \tag{23}$$

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

$$\exists t_k \in S_{i,j-1}, \begin{cases} t_k > t_j \\ \sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_j}{T_k}\right\rceil = \sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k}\right\rceil \end{cases} \tag{24}$$

Moreover,

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

So, Equations 24 and 25 lead to:

$$\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k}\right\rceil < \sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k}\right\rceil \tag{26}$$

Since $t_k > t_j$, we have $\frac{1}{t_k} < \frac{1}{t_j}$. And, if we use Equation 26, we have:

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

By definition of t_{j-1} (i.e., Equation 17), we have

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k}\right\rceil = \min_{t\in S_{i,j-1}} \frac{1}{t}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t}{T_k}\right\rceil \tag{28}$$

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

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k}\right\rceil \leq \frac{1}{t_k}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_k}{T_k}\right\rceil \tag{29}$$

If we combine Equations 27 and 29, we obtain:

$$\frac{1}{t_{j-1}}\sum_{k=1}^{j-1} C_k(L_i)\left\lceil \frac{t_{j-1}}{T_k}\right\rceil < \frac{1}{t_j}\sum_{k=1}^{j} C_k(L_i)\left\lceil \frac{t_j}{T_k}\right\rceil \tag{30}$$

That is to say,

$$\Delta_{i,j-1} > \Delta_{i,j} \tag{31}$$

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 2. *The Vestal's algorithm is optimal to schedule a set of independent tasks with constrained-deadlines under a fixed 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 braking 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 priority policy. □

4. Processor speed

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 FP 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 Processor speed modulation and priority assignment

Require: τ^* = set of tasks to schedule
Ensure: $\Delta^* =$ maximum scaling factor
Ensure: $\tilde{\tau} =$ scheduled task system
  $\tau \Leftarrow \tau^*$
  $\tilde{\tau} \Leftarrow \emptyset$
  for j from n to 1 **do**
    $\tau_{\text{Vestal}} = \emptyset$
    for $\tau_A \in \tau$ **do**
      if $\tau_{\text{Vestal}} = \emptyset$ **then**
        $\tau_{\text{Vestal}} \Leftarrow \tau_A$
        $\Delta^* = \Delta(\tau_A, \tau)$
      else
        if $\Delta(\tau_{\text{Vestal}}, \tau) < \Delta(\tau_A, \tau)$ **then**
          $\tau_{\text{Vestal}} \Leftarrow \tau_A$
        end if
      end if
    end for
    $\pi(\tau_{\text{Vestal}}) \Leftarrow j$
    $\tau \Leftarrow \tau - \{\tau_{\text{Vestal}}\}$
    $\tilde{\tau} \Leftarrow \tilde{\tau} \cup \{\tau_{\text{Vestal}}\}$
    if $\Delta(\tau_{\text{Vestal}}, \tau) < \Delta^*$ **then**
      $\Delta^* = \Delta(\tau_{\text{Vestal}}, \tau)$
    end if
  end for

The 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 3) 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.*

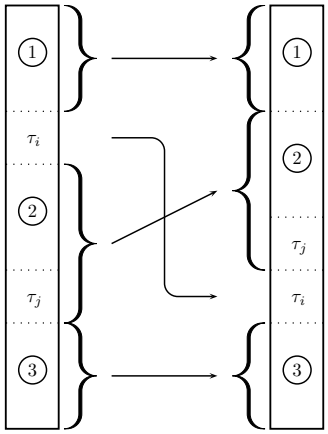

Figure 3. Scheme of the transformation

Proof. We shall use an interchange argument to prove the result. The 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 3. *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 priority policy. This proves the Theorem 3. □

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.

5. Sensitivity analysis on WCET

We next adapt the Bini et al. sensitivity analysis (i.e., initially developed for classical real-time task systems [6]) 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 criticity level ℓ.

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

Table 3. Example of a multiple criticality tasks system

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

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 Equation 1 (a complete example is detailed in the next section). In practice, such a problem can be easily solved by setting Equation 2 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., Equation 1).

For that purpose every time that Equation 1 is not satisfied:

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

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

$$C_i(\ell) \leftarrow C_i(\ell+1) \quad (34)$$

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 Table 3.

And let focus on the task τ_2 on which we will perform the sensitivity analysis. Bini et al. showed in [6] that when the schedulability analysis is performed only on a single task, Equation 32[1] can be rewritten in:

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

To apply the sensitivity analysis, scheduling points must be computed. In [5], Bini uses these recursive definition to find them:

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

[1] We do not use the Bini's notation ΔC_k^{max} to avoid possible confusion with the critical scaling factor Δ. We use δC_k^{max} instead.

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

Table 4. Trace of the δC_i

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 5. Sensitivity analysis before normalization step

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

$$\text{sched}(\text{P}_2) = \{T_1, D_2\} \quad (37)$$
$$\text{sched}(\text{P}_3) = \{T_1, D_3\} \quad (38)$$

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):

$$\delta C_2 = \max(22, -5) = 22 \quad (39)$$
$$\delta C_3 = \max(10, 32) = 32 \quad (40)$$

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

$$\delta C_2^{\max}(1) = \min_{i=1,\ldots,n \wedge L_i=1}(\delta C_i) = \min(\{\delta C_3\}) \quad (41)$$
$$\delta C_2^{\max}(2) = \min_{i=1,\ldots,n \wedge L_i=2}(\delta C_i) = \min(\{\delta C_2\}) \quad (42)$$

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 (Equation 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 4 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 Equation 2.

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

Table 6. Sensitivity analysis after normalization step

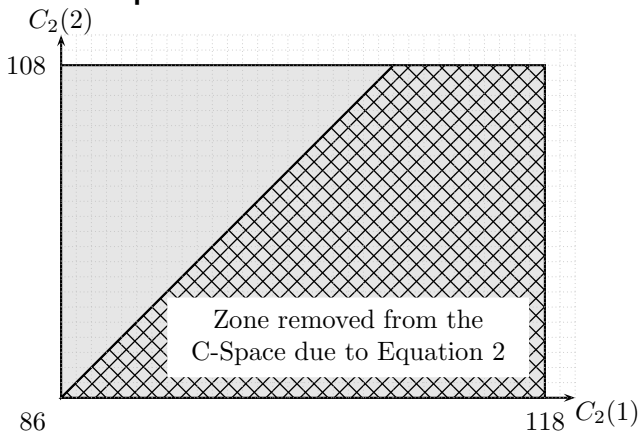

Figure 4. Multiple criticality $\mathbb{C}$-space for task τ_2

6. Conclusion and future work

In this article, we investigate the multiple criticality task scheduling model introduced in [13] and [4]. 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-priority algorithm for scheduling independent task systems with constrained-deadlines, and as a consequence that the tie braking rule used in Vestal's algorithm is not useful for assigning fixed-priority to multi 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.

We also show how to adapt the sensitivity analysis in the $\mathbb{C}$-space originally developed by Bini in [6] for the case of multiple criticality task systems. Such an extension allows to analyse a subset of tasks. From a practical point of view, it is particularly useful to analyse all tasks belonging to the specific critical level.

Future work Lehockzy, in [9] performed a sensitivity analysis for a single task and the whole task system. Bini et al., in [6] extends 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.

References

[1] ARINC. Avionics application software standard interface. *ARINC Spec*, 653, 1997.

[2] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. *Real-Time Systems*, 1991.

[3] F. Authority. *Software Considerations in Airborne Systems and Equipment Certification*. RTCA Inc: EUROCAE, 1992.

[4] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS '08: Proceedings of the 2008 Euromicro Conference on Real-Time Systems*, pages 147–155, Washington, DC, USA, 2008. IEEE Computer Society.

[5] E. Bini and G. Buttazzo. Schedulability analysis of periodic fixed priority systems. *Computers, IEEE Transactions on*, 53(11):1462–1473, Nov. 2004.

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

[7] L. Bougueroua, L. George, and S. Midonnet. 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)*, April 2007.

[8] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[9] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings of the 11th Real-Time Systems Symposium*, pages 201–209, Dec 1990.

[10] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm-exact characterization and average case behavior. In *in Proc. ZEEE Real-Time Svst. - SvmD*, 1989.

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

[12] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings IEEE Real-Time Systems Symposium*, pages 181–191. IEEE Computer Society Press, 1986.

[13] S. Vestal. 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*, pages 239–243, Washington, DC, USA, 2007. IEEE Computer Society.