# Worst-case analysis of feasibility tests for self-suspending tasks

Frédéric Ridouard, Pascal Richard

# Worst-case analysis of feasibility tests for self-suspending tasks

Frédéric Ridouard, Pascal Richard
LISI-ENSMA
Av C. Ader, Téléport 2 BP 40109
86961 Futuroscope Cedex, France
{frederic.ridouard,pascal.richard}@ensma.fr

## Abstract

*In most real-time systems, tasks invoke external operations processed upon dedicated processors. External operations introduce self-suspension delays in the task behaviors. In such task systems, checking that deadlines will be meet at run-time is $\mathcal{NP}$-Hard in the strong sense. For that reason, known response time analysis (RTA) only compute upper bounds of worst-case response times. These pessimistic estimations lead in practice the designers of a real-time system to oversize the computer features. The aim of this paper is to quantify the pessimism used in known RTA methods. We propose an exact exponential time feasibility test and define upper bounds of competitive ratio of three known RTA techniques.*

**Keywords:** Real-time, On-line scheduling, self-suspension, Maximum response time.

## 1 Introduction

A real-time system is a system in which the correctness of the system depends not only on correctness of computations, but also on the time at which the results are produced (if a result is late, it is a fault). A real-time system can be seen as a task system where each task must respect its constraints. A task meets its deadline if it completes its execution before its deadline otherwise the task misses its deadline. There exists a feasible schedule for a task system if all deadlines are met.

Several models of recurring real-time tasks have been defined. The simplest but also the most fundamental model is provided by the *periodic task model* of Liu and Layland [8]. In this model, a periodic task $\tau$ has only two characteristics $\tau = (C, T)$: $C$ is the worst-case execution requirement of task $\tau$ and $T$ its period between two successive releases. Consequently, an instance of the periodic task $\tau$ (a job) is generated and released in the system after $T$ units of times with an execution requirement equals to $C$. A job must complete its execution before the next release ($T$ units of time later). Tasks are assumed to be independent.

Most of real-time systems contain tasks with self-suspension. A task with a self-suspension is a task that during its execution prepares specifics computations (e.g. In/Out operations or *FFT* on a digital signal processor). The task is self-suspended to execute the specifics computations upon external dedicated processors. External operations introduce self-suspension delays in the behavior of tasks. The task waits until the completion of the external operations to finish its execution. Generally, the execution requirement of external operations can be integrated in the execution requirement of the task. But, if self-suspension delays are large, then such an approach cannot be used to achieve a schedulable system. Thus self-suspension must be explicitely considered in the task model.

We have already proved [13] that the feasibility problem of scheduling task systems is $\mathcal{NP}$-Hard in the strong sense. We have also shown the presence of scheduling anomalies under *EDF* for scheduling independent tasks with self-suspension upon an uniprocessor platform when preemption is allowed. We have proved [14] that classical on-line scheduling algorithms are not better than 2 competitive to minimize the maximum response time. In this paper, we show that on-line and deterministic scheduling algorithms are not optimal to schedule tasks with self-suspension. The Response Time Analysis (*RTA*) can only compute upper bounds of worst-case response times in a reasonable amount of time. These pessimistic estimations lead in practice to oversize the computer features. The aim of this paper is to quantify the pessimism used in three known *RTA* methods based on fixed-priority task systems.

Several feasibility tests are presented and defined for analysing tasks allowed to self-suspend. For fixed-priority task systems, there exist tests based on the computation of

worst-case response time: Kim *et al.* [7], Jane W. S. Liu [9] and Palencia *et al.* [11, 12]. The latter approach can be used for *EDF* scheduling [12]. There exists also a test based on the utilization factor of the processor [4]. But, no study concerning the quality of these tests are known to exhibit relative merits of these methods. Consequently, our approach is to analyze the relevance and quality of these tests.

We next analyse the feasibility tests of Kim *et al.* [7] and Liu [9] to schedule tasks with self-suspension. Before, we define the task model (Section 2). In Section 3, the feasibility tests of Kim and Liu are presented. In Section 4, we present the main technique to evaluate the on-line algorithms. In Section 5, we show that it is impossible to define an optimal on-line algorithm to schedule tasks systems when tasks are allowed to self-suspend. Lastly, the feasibility tests are analyzed to determine their pessimism.

## 2 Task model

We consider that task systems are based on a collection of periodic and independent tasks. Let $I$ be a task system of $n$ tasks. Every occurrence of a task is called a job. Every task $\tau_i$ ($1 \leq i \leq n$) arrives in the system at time 0, its relative deadline is denoted $D_i$ and its period $T_i$. If its relative deadline is equal to the period, the task has a implicit deadline else if just $D_i \leq T_i$ constrained deadline. The maximum execution requirement of a task $\tau_i$ is $C_i$.

In the system, preemption of tasks is allowed. Consequently, a job can be suspended at any time to allow the execution of others jobs and later on will be resume to continue its execution.

To simplify our results , we consider that tasks are allowed to self-suspend at most once. The Figure 1 presents this model. Every task $\tau_i$ ($1 \leq i \leq n$) has two subtasks (with a maximum execution requirement $C_{i,k}$, $1 \leq k \leq 2$) separated by a maximum self-suspension delay $X_i$ between the completion of the first subtask and the start of the second subtask. Such delays change from one execution to another since they model execution requirements of external operations. Consequently every task $\tau_i$ is denoted: $\tau_i : (C_{i,1}, X_i, C_{i,2}, D_i)$.

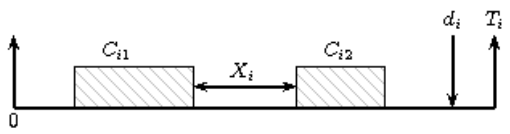


**Figure 1. Le modŁle des tâches**

The utilization factor of a periodic task $\tau_i$, is the ratio of its execution requirement to its period: $U(\tau_i) = C_i/T_i$. The utilization factor of a task system $\tau$ is the sum of the utilization factors of all tasks: $U(\tau) = \sum_{i=1}^{n} U(\tau_i)$.

The maximum response time $R_i$ of a task $\tau_i$ is equal to the difference between the completion time and the release date. To minimize the maximum response time of a task set is to minimize $\max R_i$.

A task set is said *feasible* if there exists a schedule such that all tasks are completed by their deadlines at run-time.

## 3 Presentation of feasibility tests

In the following section,we present three feasibility tests:

- *Kim et al. [7]*: To define their feasibility tests, they use the works of Wellings [16] and Ming *et al.* [10]. They define two tests based on the same principle : to consider a task with a self-suspension in two independent tasks without any suspension delay.
- *Jane W. S. Liu [9]* : This feasibility test determines the blocking time due to self-suspension and higher-priority tasks.

### 3.1 Feasibility tests of Kim *et al.* [7]

Wellings *et al.* [16] studied the tasks with self-suspension but with $C_{i,1} = 0$. The self-suspension is called release jitter [3, 16]. A release jitter for a task is the difference of time between arrival and release time. Consequently, they use task set in which each task has a release jitter. To determine the response time of a task $\tau_i$, they use the following recurrence relation:

$$R_i^0 = C_i$$
$$R_i^{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_j \qquad (1)$$

The recurrence stops if $R_i^{n+1} = R_i^n$. And the worst-case response time of $\tau_i$ is $R_i^n + X_i$. To prove that the task $\tau_i$ is schedulable, $R_i^n + X_i$ must be less than or equal to $D_i$.

Ming *et al* (cf [10]) have modified the recurrence relation of Wellings (1) to take into account any task with a self-suspension:

$$R_i^0 = C_i + X_i$$
$$R_i^{n+1} = C_i + X_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_j \qquad (2)$$

However, Ming *et al.* consider the suspension delay as a part of execution requirement. But external operations are scheduled upon dedicated processors. Consequently, such an approach can increase unnecessarily the worst-case response times of tasks. Kim *et al.* (cf [7]) define two new feasibility tests to compute worst-case response times of tasks with self-suspensions.

#### 3.1.1 Method A of Kim

They consider that $D_i \leq T_i$ for all $i$ and tasks can be preempted. This first method subdivide each task $\tau_i$ with self-suspension in two independent tasks without suspension :

- $\tau_{i,1}$, released at time $r_i$ without release jitter and with a processing requirement of $C_{i,1}$.
- $\tau_{i,2}$, released at time $r_i$, its jitter $J_{i,2}$ equals $X_i$ and a processing requirement equal to $C_{i,2}$.

The two generated tasks inherit the period and the deadline of $\tau_i$.

To prove the schedulability of task $\tau_i$, we must transform $\tau_i$ into $\tau_{i,1}$ and $\tau_{i,2}$, and we then calculate the worst-case response time of the generated tasks. $\tau_{i,1}$ has a release jitter equal to 0 and $\tau_{i,2}$ has one equal to $X_i$. The worst-case of $\tau_{i,1}$ and $\tau_{i,2}$ are calculated independently. To calculate the worst-case response time of $\tau_{i,1}$, the Wellings's formula (1) is used:

$$R_{i,1}^{n+1} = C_{i,1} + \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,1}^n}{T_j} \right\rceil C_{j,1} + \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,1}^n + X_j}{T_j} \right\rceil C_{j,2}$$

Computations stop for the smallest positive integer $n$ satisfies $R_{i,1}^{n+1} = R_{i,1}^n$ and the worst-case response time $R_{i,1}^*$ of $\tau_{i,1}$ is equal to $R_{i,1}^n$. If $R_{i,1}^* \leq D_i$ then $\tau_{i,1}$ is schedulable. Otherwise, we cannot conclude that $\tau_{i,1}$ is schedulable.

The worst-case response time of $\tau_{i,2}$ is calculated with the following recurrent formula:

$$R_{i,2}^{n+1} = C_{i,2} + \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,2}^n}{T_j} \right\rceil C_{j,1} + \sum_{j=1}^{i-1} \left\lceil \frac{R_{i,2}^n + X_j}{T_j} \right\rceil C_{j,2}$$

The worst-case response time $R_{i,2}^*$ of $\tau_{i,2}$ is calculated. To finish, if $(R_{i,1}^* + X_i + R_{i,2}^*) \leq D_i$, then $\tau_i$ is schedulable, otherwise we cannot conclude.

#### 3.1.2 Method B of Kim

This approach is an improvement of Ming's method (*cf.* Formula 2). This method consider the suspension delays as part of processing requirement of tasks. But without this assumption, during the interval of time $X_i$, other tasks can be scheduled. To calculate the worst-case response time of a task, $X_i$ can be reduced and furthermore the worst-case response time of $\tau_i$ can be shortened. Consequently, to calculate the worst-case response time of a task $\tau_i$, the following recurrent formula is used:

$$R_i^{n+1} = C_i + M_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_{j,1} + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^n + X_j}{T_j} \right\rceil C_{j,2}$$

Where $M_i = X_i - \sum_{j=1}^{i-1} \left\lfloor \frac{X_i}{T_j} \right\rfloor C_j$

If $R_i^{n+1} = R_i^n$ and $R_i^n \leq D_i$ then $\tau_i$ is schedulable. Otherwise, we cannot conclude if it is schedulable or not.

**Remark 1** *Since $M_i \leq X_i$, if $\tau_i$ is schedulable with the Ming's method (cf. Formula 2), then the task is schedulable with the method B of kim.*

### 3.2 The Liu's method [9]

To take into account the extra delay suffered by a task $\tau_i$ due to its own self-suspension and the suspension of higher-priority tasks, Liu [9] considers this delay as a factor of blocking time of $\tau_i$, denoted $b_i(ss)$.

The blocking time of a task due to its own suspension is not more than $X_i$. To determine the blocking time due to a higher-priority task $\tau_k$, we must study two cases:

- $\tau_k$ cannot delay $\tau_i$ during more than $C_k$ units of time since the task $\tau_k$ can be scheduled (or partially scheduled) during the suspension of $\tau_i$ because the processor is idle.
- Moreover, if $X_k < C_k$ then the blocking time cannot be more than $X_k$ units of time.

Consequently, the blocking factor due to each higher-priority tasks, $\tau_k$ is never more than the suspension delay of $\tau_k$ and never more than $C_k$.

Finally, the blocking time $b_i(ss)$ is equal to:

$$b_i(ss) = X_i + \sum_{k=1}^{i-1} \min(C_k, X_k)$$

Note that Liu's method is not expected to perform as well as the Kim's methods, since it does not specify where the suspension occurs within the task.

# 4 Validation of on-line algorithms

## 4.1 Introduction

This paper is interested by the validation of on-line algorithms. For any objective function, we wish to know the quality of the solution obtained with an on-line scheduling algorithm (hereafter referred to as the performance guarantee of the algorithm). This quality will not be better than the quality obtained by an optimal off-line algorithm. Two commonly used methods to evaluate the performance of an on-line algorithm are known:

- The simulation : The on-line scheduling algorithms are compared and evaluated in the confine of a stochastic model.
- The competitive analysis : The on-line algorithm is compared with an optimal off-line algorithm for the same problem so that the on-line algorithm achieves its worst-case results.

## 4.2 The simulation

The simulation allows to compare the on-line algorithms. To evaluate the performance of an on-line algorithm, this method defines a stochastic model by assuming a certain probabilistic distribution to compute task features. With this model, a task system is generated and it is submitted to every on-line algorithm.
However, the on-line algorithm is then evaluated within the confine of the stochastic model. Moreover, this approach is inconsistent with the environments of on-line algorithms. Because the probabilistic distribution model based on past observations will always model the future arrivals of jobs. But, as pointed out by Karp [6], this assumption is inconsistent with the nature of on-line algorithms unless the future resemble to the past.

## 4.3 The competitive analysis

The first results of this approach are the results obtained by *Sleator and Tarjan* [15] in 1985. This approach compares the on-line algorithm to an optimal clairvoyant algorithm in the worst-case. The optimal off-line algorithm (said *the adversary*) defines the instances of problem to compare the two algorithms. But a good adversary defines instances of problem so that the on-line algorithm achieves its worst-case performance. To analyse deterministic algorithms, two equivalent adversaries can be used:

- The oblivious adversary defines the task system in advance based on the characteristics of the on-line algorithm, and serves it optimally.
- The adaptive on-line adversary defines the next request of tasks according to the decision taken by the on-line algorithm, but serves it immediately.

An algorithm that minimizes a measure of performance, is $c$-competitive if the performance obtained by the on-line algorithm is less than or equal to $c$ times the value of the optimal algorithm. More formally, given an on-line algorithm $A$ and a task system $I$, the performance obtained by the on-line algorithm $A$ (*Resp.* the adversary) in scheduling $I$ is denoted $\sigma_A(I)$ (*Resp.* $\sigma^*(I)$). Consequently, $A$ is $c$-competitive if there exists a task system $I$ and a constant $c$ so that $\sigma_A(I) \leq c\sigma^*(I)$.

The competitive ratio $c_A$ of an on-line algorithm $A$ is the worst-case ratio while considering any instance $I$.

**Definition 1** *The competitive ratio, $c_A$, of the on-line algorithm $A$ to minimize a performance criterion while considering any instance $I$ is:*

$$c_A = \sup_{tout I} \frac{\sigma_A(I)}{\sigma^*(I)}$$

# 5 On-line algorithms are not optimal

In this section, we demonstrate that there exists no on-line optimal algorithm to schedule task systems when tasks are allowed to self-suspend upon uniprocessor systems.

**Theorem 1** *No on-line deterministic algorithms are optimal to schedule tasks systems when tasks are allowed to self-suspend upon a uniprocessor system.*

**Proof :**
To prove this theorem, we use the competitive analysis with an adaptative adversary (cf. Section 4.3). Hence, we define a task system and according to the scheduling decision of any on-line and deterministic algorithm, the adversary defines the next request of tasks so that the on-line algorithm misses a deadline and the adversary serves it optimally. We define a task system $I$ and we show that no on-line deterministic algorithm can schedule optimally $I$. We consider that at time $0$ two tasks are available:

$$\tau_1 : C_{1,1} = 1, X_1 = 7, C_{1,2} = 1, D_1 = 10, T_1 = 10$$
$$\tau_2 : C_{2,1} = 1, X_2 = 4, C_{2,2} = 1, D_2 = 9, T_2 = 10$$

Let $A$ be an on-line algorithm. At time $0$, to make its scheduling decision, $A$ has two choices:

1. *The on-line algorithm* $A$ does not schedule $\tau_2$ at time $0$ (this schedule is presented Figure 2.a). Either it schedules $\tau_1$ or it lefts the machine idle. In the two cases, it schedules $\tau_2$ at time $t$, with $0 < t \leq 3$ to respect the deadline of $\tau_2$ (For the Figure 2.a, $t = 2$). But at time 3, an other task $\tau_3$ with a period equals to 10 is released:

$$\tau_3 : C_{1,1} = 1, X_1 = 2, C_{1,2} = 3, D_1 = 9$$

At time 3, the on-line algorithm $A$ schedules $\tau_3$ since it has not laxity. But $A$ has not enough time to complete $\tau_2$ and $\tau_3$ before their common deadline at time 9. Consequently $A$ has done a bad choice and hence, the scheduling of $I$ under $A$ is not feasible.

*The optimal off-line algorithm* schedules at time $0$, $\tau_2$ and at time $1$, $\tau_1$. At time 3, $\tau_3$ is released and immediately run. At time 5, $\tau_2$ is resumed from its self-suspension and completed at time 6. Finally, $\tau_3$ is completed at time 9 and $\tau_1$ at time 10. Figure 2.b presents the scheduling of $I$ under an optimal off-line algorithm.

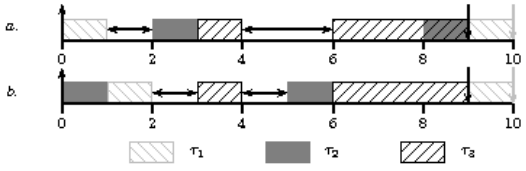


**Figure 2. The on-line algorithm $A$ does not schedule $\tau_2$ at time $0$ but at time $t = 2$**

Consequently, we show that if an on-line and deterministic algorithm chooses to not run $\tau_2$ at times $0$, it is not optimal since there exits a feasible schedule of $I$.

2. When $A$ schedules $\tau_2$ at time $0$, at time $1$, it must schedule $\tau_1$ to respect its deadline. At time 2 arrives $\tau_4$ with implicit deadline ($T_4 = D_4$):

$$\tau_4 : C_{1,1} = 4, X_1 = 3, C_{1,2} = 1, T_1 = 10$$

$A$ schedules $\tau_4$ at time 2 (to respect its deadline) and at time 6, $\tau_2$ is resumed from its self-suspension and scheduled. But between time 9 and 10, $A$ must complete $\tau_1$ and $\tau_4$, hence it is impossible. $A$ cannot schedule the task system $I$. Figure 3.a presents the scheduling of $I$ under $A$.

The optimal off-line algorithm schedules at time $0$, $\tau_1$, at time $1$, $\tau_2$ and at time 2, $\tau_4$. At time 6, $\tau_2$ is resumed and completed at time 7. Finally, $\tau_1$ is completed at time 9 and $\tau_4$ at time 10. Figure 3 presents the scheduling of $I$ obtained by the adversary.

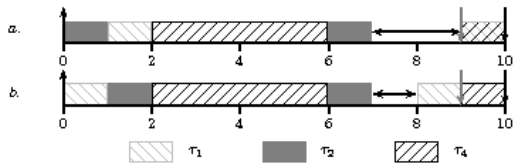


**Figure 3. The on-line algorithm $A$ schedules $\tau_2$ at time $0$**

Consequently, there exits no optimal on-line and deterministic algorithm to schedule task systems upon uniprocessor system when tasks are allowed to self-suspend.

□

# 6 Analysis of feasibility tests

## 6.1 Introduction

In this section, the feasibility tests presented in the Section 3 are analyzed. We use the two validation techniques presented in Section 4. These feasibility tests compute the upper bound of the maximum response time of every task. At first time, we establish the pessimism of these estimations. To determine this pessimism, we use the approach presented by Epstein and Rob Van Stee in [5]. They provided lower bounds for on-line deterministic (or randomized) algorithms for several optimization criteria. They studied problems in term of competitive analysis. They automatically generated a huge number of synthetic task sets. For each task set, they computed the competitive ratio for the on-line algorithm studied. Finally, they kept the task set with the worst competitive ratio. In our work, the optimality criteria is the minimization of the maximum response time. We use the same method: we generate, with a brute force generation (as done in [1] for non-preemptive system), a huge number of task sets and for each feasibility test we keep the task set leading to the worst competitive ratio. To complete this analysis, we define a stochastic model to generate a lot of task systems. With these task systems, statistics are defined to compare these tests.
The feasibility tests presented in this paper are based on fixed-priority task systems. Consequently, to use the competitive analysis (*cf.* Section 4.3), we don't use an optimal off-line algorithm as adversary but we use the fixed priority algorithm (*RM*).

## 6.2 Simulation environment

The first constraint is to obtain schedulable task systems. The utilization factor of generated task systems is bounded by $0.7$. Decreasing the utilization factor is an important parameter for generating feasible task systems.

The presence of anomalies for scheduling tasks with self-suspension in fixed-priority task system [13, 14] increases the costs of computations since reducing processing requirement can lead to worst-case response times of tasks.

The feasibility tests are based on the fixed-priority scheduling algorithm *RM*. But, we proved [13, 14] that scheduling anomalies can occur while scheduling tasks with self-suspension under *RM*. Consequently, if the execution requirement of a task is decreased of one unit of time, the response time can increase and a deadline can be missed. Hence, to determine the exact worst-case response time of task systems where tasks are allowed to self-suspend, we must test all possible processing requirements (and suspension delays) for each job of each task.

**Remark 2** *$C_i$ (resp. $X_i$) is the upper limit to its processing requirement (resp. worst-case suspension delay) of task $\tau_i$. Consequently, we consider that the execution requirement (resp. suspension delay) of a task can vary between $1$ and $C_i$ (resp. $X_i$) since all parameters are integers. Moreover, $C_{i,1}$, $X_i$ and $C_{i,2}$ belong to the interval $[1, 4]$.*

We define two rules to reduce the hyperperiod length:

- To minimize the length of the hyper period, the tasks are synchronous.
- To minimize the computations and the length of the hyper period, tasks have harmonic periods (Definition 2).

**Definition 2** *Let $I : (\tau_1, \tau_2, \ldots, \tau_n)$ be a task system. $I$ has harmonic periods if and only if the two following properties are respected:*

$$T_1 \leq T_2 \leq \cdots \leq T_n$$
$$\forall i, i \in \{2, \ldots, n\}, T_i \mod T_{i-1} = 0$$

**Remark 3** *We assume that tasks are indexed in increasing order of periods. The second property limits the length of the feasibility interval and the number of jobs within it. Consequently the task with the smallest priority is $\tau_n$. Since we use the fixed priority scheduling algorithm, $\tau_n$ has the longest period.*

To generate a task: first, the executive requirements and the suspension delays are computed. Finally, to determine the period, the period of the task previously generated, is multiplied until the utilization factor (of the task system) is less than $0.7$.
Finally, we consider tasks with implicit deadlines. Moreover, the generated task systems contain only two or three tasks to firstly limit time while computing exact response times, and secondly to exhibit task systems leading to worst-case performance guarantees.

## 6.3 Lower bounds

### 6.3.1 Introduction

Next subsections detail task sets automatically generated by our simulator leading to the worst-case performance of the three considered feasibility tests.

### 6.3.2 Method A of Kim

**Lower bound 1** *The lower bound of the competitive ratio for the feasibility test of the method A of Kim to minimize the maximum response time while scheduling tasks allowed to self-suspend at most once is $2,91667$.*

**Proof:**
Let $I_A$ be the following task system containing three tasks:

$$\tau_1 : C_{1,1} = 3, X_1 = 2, C_{1,2} = 3, T_1 = 12$$
$$\tau_2 : C_{2,1} = 3, X_2 = 1, C_{2,2} = 1, T_2 = 96$$
$$\tau_3 : C_{3,1} = 1, X_3 = 1, C_{3,2} = 1, T_3 = 96$$

The upper bound of the maximum response time obtained with the method A of Kim denoted $\sigma_i^A$ for each task $\tau_i$ of $I_A$ is:

$$\tau_1 : \sigma_1^A = 8, \tau_2 : \sigma_2^A = 17, \tau_3 : \sigma_3^A = 35$$

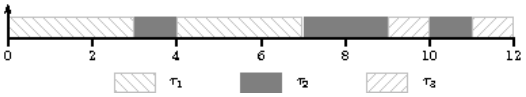


**Figure 4. The exact maximum response time obtained by *RM* while scheduling $I_A$.**

Figure 4 presents the exact maximum response time obtained with the fixed-priority scheduling algorithm *RM*. There are no scheduling anomalies and for that reason, tasks are scheduled with their worst-case execution requirements and suspension delays. At time $0$, *RM* schedules $\tau_1$ and during its suspension it schedules partially $\tau_2$. At time $7$, $\tau_2$ is scheduled. At time $9$, $\tau_2$ is suspended and $\tau_3$ scheduled. Finally at time $11$, $\tau_2$ finishes its execution and $\tau_3$, at

time 12. Hence, the exact maximum responses time of tasks (denoted $\sigma_i^{RM}$) are:

$$\tau_1 : \sigma_1^{RM} = 8, \tau_2 : \sigma_2^{RM} = 11, \tau_3 : \sigma_3^{RM} = 12$$

Consequently, the worst-case competitive ratio for $I$ is:

$$\begin{aligned} c_A^{RM} &= \sup_{anyI} \frac{\sigma_A(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_A(I_A)}{\sigma_{RM}(I_A)} \\ &\geq \max\left(\frac{\sigma_1^A}{\sigma_1^{RM}}, \frac{\sigma_2^A}{\sigma_2^{RM}}, \frac{\sigma_3^A}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_3^A}{\sigma_3^{RM}} = \frac{35}{12} = 2.91667 \end{aligned}$$

□

#### 6.3.3 Method B of Kim

**Lower bound 2** *The lower bound to minimize the maximum response time for the method B of Kim is equal to* $2,75$.

**Proof:**
Let $I_B$ be the following task system:

$$\begin{aligned} &\tau_1 : C_{1,1} = 1, X_1 = 1, C_{1,2} = 3, T_1 = 6 \\ &\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 2, T_2 = 270 \\ &\tau_3 : C_{3,1} = 3, X_3 = 2, C_{3,2} = 3, T_3 = 810 \end{aligned}$$

The upper bound obtained with the second method of Kim for each task $\tau_i$ of $I_B$ and denoted $\sigma_i^B$ is equal to:

$$\tau_1 : \sigma_1^B = 5, \tau_2 : \sigma_2^B = 22, \tau_3 : \sigma_3^B = 35$$

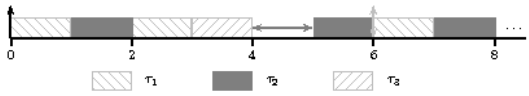


**Figure 5. The exact maximum response time obtained by *RM* while scheduling $I_B$.**

Figure 5 presents the exact maximum response time of task $\tau_2$. At time 0, ***RM*** schedules $\tau_1$ since it has the highest priority. At time 1, $\tau_1$ is suspended and $\tau_2$ scheduled. At time 2, $\tau_2$ is suspended and $\tau_1$ is completed. At time 3, $\tau_3$ is scheduled during the suspension of $\tau_2$. At time 6, $\tau_1$ is released and at time 8, $\tau_2$ is completed. Consequently, we obtain the following exact response times:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 8, \tau_3 : \sigma_3^{RM} = 24$$

Consequently, the competitive competitive ratio for the second method of Kim is:

$$\begin{aligned} c_B^{RM} &= \sup_{anyI} \frac{\sigma_B(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_B(I_B)}{\sigma_{RM}(I_B)} \\ &\geq \max\left(\frac{\sigma_1^B}{\sigma_1^{RM}}, \frac{\sigma_2^B}{\sigma_2^{RM}}, \frac{\sigma_3^B}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_2^B}{\sigma_2^{RM}} = \frac{22}{8} = 2.75 \end{aligned}$$

□

#### 6.3.4 Jane W. S. Liu's method

**Lower bound 3** *The competitive ratio on RM obtained with the method of Liu is* $2,875$ *to minimize the maximum response time for tasks are allowed to self-suspend at most once.*

**Proof:**
We use the instance $I_B$ defined in the Theorem 2. The upper bound of maximum response time obtained with the method of Liu for each task $\tau_i$ of $I_B$ and denoted $\sigma_i^L$ are:

$$\tau_1 : \sigma_1^L = 5, \tau_2 : \sigma_2^L = 23, \tau_3 : \sigma_3^L = 47$$

Figure 5 presents the exact maximum response time obtained with ***RM*** while scheduling $I_B$. These results are:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 8, \tau_3 : \sigma_3^{RM} = 24$$

Hence, the competitive ratio obtained for the method of Liu to minimize the maximum response time is equal to:

$$\begin{aligned} c_L^{RM} &= \sup_{anyI} \frac{\sigma_L(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_L(I_B)}{\sigma_{RM}(I_B)} \\ &\geq \max\left(\frac{\sigma_1^L}{\sigma_1^{RM}}, \frac{\sigma_2^L}{\sigma_2^{RM}}, \frac{\sigma_3^L}{\sigma_3^{RM}}\right) \\ &\geq \frac{\sigma_2^L}{\sigma_2^{RM}} = \frac{23}{8} = 2.875 \end{aligned}$$

□

#### 6.3.5 Comparison of feasibility tests

These results show that the method B of *Kim* obtained the best results. But, we cannot conclude that the best feasibility test is the method B. Because, the only possible conclusion is that these feasibility tests are not comparable. We cannot conclude since it is possible for each feasibility test to determine tasks sets where the feasibility test is the best test but it is the worst for another. Kim *et al.* have already proved that their two methods are not comparable [7]. Now to prove that all the tests are not comparable, we show that

the method A of *Kim* and the method of *Liu* are not comparable:
Let $I$ be the following task set:

$$\tau_1 : C_{1,1} = 2, X_1 = 3, C_{1,2} = 1, T_1 = 7$$
$$\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 2, T_2 = 56$$
$$\tau_3 : C_{3,1} = 3, X_3 = 1, C_{3,2} = 2, T_3 = 392$$

The competitive ratio obtained for the three tests:

$$\sigma_A^{RM}(I) = 1.47$$
$$\sigma_B^{RM}(I) = 1.30$$
$$\sigma_L^{RM}(I) = 1.80$$

The ratio obtained with the method A of *Kim* is better than the ratio obtained with the method of *Liu*.

Let $I'$ be the following task set:

$$\tau_1 : C_{1,1} = 2, X_1 = 3, C_{1,2} = 1, T_1 = 9$$
$$\tau_2 : C_{2,1} = 2, X_2 = 3, C_{2,2} = 3, T_2 = 45$$
$$\tau_3 : C_{3,1} = 2, X_3 = 2, C_{3,2} = 1, T_3 = 90$$

The ratios on *RM* are:

$$\sigma_A^{RM}(I') = 1.69$$
$$\sigma_B^{RM}(I') = 1.06$$
$$\sigma_L^{RM}(I') = 1.56$$

But with this task set, the method of *Liu* is better than the method A of Kim. To conclude all tests are not comparable. Consequently, we can define a last test: the best method.

#### 6.3.6 The Best Method

This method consist in applying for **every task** all tests and to store the smallest computed response time. Such a method can help to decrease the competitive ratio (but we have no formal proof of that fact).

**Lower bound 4** *The competitive competitive ratio obtained while considering for each task system the best feasibility test is* $2,16667$ *to minimize the maximum response time for tasks allowed to self-suspend at most once.*

**Proof:**
Let $I_C$ be the following task system:

$$\tau_1 : C_{1,1} = 1, X_1 = 1, C_{1,2} = 3, T_1 = 9$$
$$\tau_2 : C_{2,1} = 1, X_2 = 3, C_{2,2} = 1, T_2 = 72$$
$$\tau_3 : C_{3,1} = 3, X_3 = 2, C_{3,2} = 1, T_3 = 648$$

The exact maximum response time obtained with the algorithm *RM* are:

$$\tau_1 : \sigma_1^{RM} = 5, \tau_2 : \sigma_2^{RM} = 6, \tau_3 : \sigma_3^{RM} = 14$$

The Table 1 presents for each task, the competitive ratio obtained with each feasibility test.

| Tasks | Method A of Kim | Method B of Kim | Method of Liu |
|---|---|---|---|
| $\tau_1$ | 1.00 | 1.00 | 1.00 |
| $\tau_2$ | 2.17 | 2.17 | 2.33 |
| $\tau_3$ | 1.57 | 1.43 | 1.64 |

**Table 1. Competitive ratio for each task of $I_C$**

Consequently, the competitive competitive ratio ,$C_{Bst}$ for this method is:

$$\begin{aligned} c_{Bst}^{RM} &= \sup_{anyI} \frac{\sigma_{Bst}(I)}{\sigma_{RM}(I)} \geq \frac{\sigma_L(I_C)}{\sigma_{RM}(I_C)} \\ &\geq \sup_{1 \leq i \leq 3} \{\inf\{c_A^{RM}(\tau_i), c_B^{RM}(\tau_i), c_L^{RM}(\tau_i)\}\} \\ &\geq 2.16667 \end{aligned}$$

□

### 6.4 Simulation results

#### 6.4.1 Introduction

In this section, we present numerical results obtained during the brute force generation described in Section 6.2. All tests (upper bounds and the exact test) have been applied to every generated task set. We are aware that such a simulation environment is not sufficient (*cf.* [2]) to exhibit relative merits of the considered feasibility tests that they are only valid in the confine of our stochastic model (see Section 6.2).

#### 6.4.2 Results

To obtain relevant results from a statistical point of view, we generated one million of tasks sets. The tasks sets are generated with the procedure defined in Section 6.2. The Table 2 presents statistical results obtained by the simulator for the feasibility tests.
The first row of the Table 2 presents the percentage of times where every feasibility test has been the best one (while scheduling task sets). The method B of *Kim* leads to the best results.
The average competitive ratios in row 2 of the Table 2 allows us to remark that the method B of *Kim* is the feasibility test arriving in first position. But even if the percentage of the method of *Liu* is equal to zero, this feasibility test has a average less than the average of the method A of *Kim*.

With the standard deviations (row 3 of the Table 2), the feasibility test with the smallest standard deviation is the method A of *Kim*.

| Feasibility tests | Method A of Kim | Method B of Kim | Method of Liu |
|---|---|---|---|
| Best method | 3.64% | 99.8% | $\approx$ 0.00% |
| Average ratios | 1.65 | 1.21 | 1.50 |
| Standard deviations of ratios | 0.18 | 0.20 | 0.22 |

**Table 2. Results of simulation for the feasibility tests for task systems with $2$ or $3$ tasks**

## 7 Conclusion

In this paper, we have presented some results on tasks allowed to self-suspend at most once. For such task systems there exists no on-line optimal algorithm. We also presented the performances of three different feasibility tests. For these tests, our aim was to compute their pessimisms since they compute an upper bound of the exact maximum response time of tasks. To determine this pessimism, we use the approach of Epstein and Van Stee [5] and also the competitive analysis. But the feasibility tests are not compared to an optimal algorithm, but to the fixed-priority on-line algorithm *RM*. Hence, we shown that the competitive ratio of feasibility tests are between $2.75$ and $2.91667$ implying the designers of real-time system to oversize the computer features. We also shown that feasibility tests are not comparable. But, if for each task, we apply every feasibility test and we retain the best then the competitive ratio decreases to $2.16667$. Finally, for task sets with a small number of tasks and exactly one self-suspension per task, the method B of Kim *et al.* is the best one.

In further works, an interesting issue is to analyze others feasibility tests and to consider a more general stochastic environment (with task sets having a larger number of jobs). Also to extend the approach to the *EDF* scheduling policy and the feasibility test of Palencia [12] (based on *EDF*). An other interesting issue can be to consider dependent tasks (tasks with shared resources or precedence constraints) [11].

## References


[1] I. Alzeer, P. Molinaro, and Y. Trinquet. Calcul exhaustif du temps de rponse de tches et messages dans un systme temps rel rparti. *In Proceedings of the* $13^th$ *Real-Time Systems*, 2005.

[2] E. Bini and GC. Buttazzo. Biasing effects in schedulability measures. *IEEE Proceedings of the* $16^th$ *Euromicro Conference on Real-Time Systems (ECRTS04), Catania, Italy*, July 2004.

[3] A. Burns. Preemptive priority-based scheduling: An appropriate engineering approach. *in Advances in Real-Time Systems, S.H. Son, Ed.,Prentice Hall, New Jersey*, pages 225–248, 1995.

[4] U. C. Devi. An improved schedulability tast for uniprocessor periodic task systems. *proc. Euromicro Conference on Real-Time Systems (ECRTS'03)*, pages 23–30, 2003.

[5] L. Epstein and R. Van Stee. On non-preemptive scheduling of periodic and sporadic tasks. *Theoretical Computer Science*, 299:439–450, 2003.

[6] R. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? *Algorithms, Software, Architecture, IFIP Transactions A-12*, Information processing:1:416–429, 1992.

[7] I-G. Kim, K-H. Choi, S-K. Park, D-Y. Kim, and M-P. Hong. Real-time scheduling of tasks that contain the external blocking intervals. *Real-Time and Embedded Computing Systems and Applications(RTCSA'95)*, 1995.

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

[9] Jane W. S. Liu. *Real-Time Systems*, chapter Priority-Driven Scheduling of Periodics Tasks, pages 164–165. Prentice Hall, 2000.

[10] L. Ming. Scheduling of the inter-dependent messages in real-time communication. *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, Dec. 1994.

[11] J.C. Palencia and M. Gonzales-Harbour. Schedulability analysis for tasks with static and dynamic offsets. *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 1998.

[12] J.C. Palencia and M. Gonzales-Harbour. Offset-based response time analysis of distributed systems scheduled under edf. *Proceedings of the IEEE Real-Time Systems Symposium*, 2003.

[13] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, 1, December 2004.

[14] F. Ridouard, P. Richard, and F. Cottet. Ordonnancement de tches indpendantes avec suspension. *Proceedings of the 13rd RTS Embedded Systems (RTS'05)*, 1, April 2005.

[15] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM 28*, 2:202–208, 1985.

[16] A.J. Wellings, M. Richardson, A. Burns, N. Audsley, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 1993.