Approximate Feasibility Analysis and Response-Time Bounds of Static-Priority Tasks with Release Jitters

Pascal Richard
LISI/ENSMA
University of Poitiers (France)
pascal.richard@univ-poitiers.fr

Joël Goossens
Computer Science Department
Université Libre de Bruxelles
joel.goossens@ulb.ac.be

Nathan Fisher
Department of Computer Science
University of North Carolina, Chapel Hill
fishern@cs.unc.edu

Abstract

We consider static-priority tasks with constrained-deadlines that are subjected to release jitter. We define an approximate worst-case response-time analysis *and propose a polynomial-time algorithm. For that purpose, we extend the Fully Polynomial-Time Approximation Scheme (*FPTAS*) presented in [6] to take into account release jitter constraints; this feasibility test is then used to define a polynomial time algorithm that approximate worst-case response times of tasks. Nevertheless, the approximate worst-case response time values have not been proved to have any bounded error in comparison with worst-case response times computed by an exact algorithm (with pseudo-polynomial time complexity).*

1 Introduction

Guaranteeing that tasks will always meet their deadlines is a major issue in the design of hard-real time systems. We consider the problem of ensuring that periodic tasks scheduled by a preemptive static-priority scheduler upon a uniprocessor platform meet all deadlines. Every execution of a given task is called a job. We consider tasks that have constrained-deadlines (i.e., deadlines are less than or equal to task periods) and are subjected to release jitter. A *release jitter* models an interval of time in which a task waits the next tick of the RTOS in order to start or is pending due to input communications.

Tasks are scheduled at run-time using a static-priority scheduling policy. Every task has a static priority and at any time the executed job has the highest priority among tasks awaiting execution. The feasibility problem consists of proving that tasks will always meet their deadlines at run-time. For the considered real-time systems, the feasibility problem is not known to be NP-hard, but only pseudo-polynomial time tests are known. However, pseudo-polynomial time complexity is too computationally expensive for performing on-line task admission or for analysing large distributed systems using classical methods such as the holistic analysis [19].

For a static-priority system, a task set is *feasible* on a given processing platform, if every task will always meet all deadlines when scheduled according to its given static-priority on the given platform. A feasibility test is an algorithm used to check if a task set is feasible or not. One can distinguish several approaches to designing a feasibility test for real-time task sets: *(i)* an exact feasibility test, *(ii)* a sufficient feasibility test (also known as pessimistic feasibility test) and *(iii)* an approximate feasibility test. We briefly describe their main characteristics.

An *exact feasibility test* can always correctly categorize task sets as either *feasible* or *infeasible* upon a specific hardware platform [10, 13, 15]. An exact test will label a periodic task set as "infeasible" if and only if the task set will miss a deadline at run-time. Neither a polynomial-time test nor NP-hardness result are known for static-priority tasks having constrained-deadlines.

A *sufficient feasibility test* always leads to an exact positive decision: if the test concludes that a task set is feasible then no deadline will be missed at run-time. But, when it concludes that a task is infeasible, then it may be a rather pessimistic decision (i.e., tasks may meet their deadlines at run-time). Sufficient feasibility tests have a lower computation complexity than corresponding exact feasibility tests. Numerous sufficient feasibility tests are known in the literature (e.g. [16, 11, 3, 9, 1, 4]).

An *approximate feasibility test* is based on the approximability theory of NP-hard optimization problems [7]. It reduces the gap between the two previous approaches to control the "unused processor capacity" for tests based on the processor-demand analysis. It runs in *polynomial*-time according to an accuracy parameter ϵ. An approximate feasibility test allows to conclude that a task set is [6, 5]:

- feasible (upon a unit-speed processor).

- infeasible upon a $(1-\epsilon)$-speed processor. That is, "we must effectively ignore ϵ of the processor capacity for the test to become exact" [6]. So, the pessimism introduced by the feasibility test is kept bounded by a constant multiplicative factor.

In [17], some numerical experiments are presented that show the practical interest of several approximate feasibility analysis in comparison with exact feasibility tests.

Most of feasibility tests produce a boolean decision: feasibility or infeasibility. However, an important qualitative measure for a task is its *worst-case response time* (i.e., the maximum size interval of time between a release of a task and its completion). Response-Time Analysis is often used to quantify the maximum earliness or tardiness of tasks and to bound release jitter of dependent tasks or messages in a distributed system. For synchronous static-priority systems, worst-case response times of tasks can be computed in *pseudo*-polynomial time.

This research. As far as we know, no approximation algorithm is known for approximating worst-case response times of tasks with a constant performance guarantee (i.e., upper bounds of worst-case response times). The aim of this paper is to introduce such an analysis and to try to show its relationship with approximate feasibility analysis. We present an FPTAS for analysing the feasibility of static priority tasks with release jitter constraints. We then show feasibility tests can be used to define upper bounds of worst-case response times based on a polynomial time algorithm. Lastly, we show that there exists some task systems such that ratio between the exact worst-case response time and the approximate worst-case response time is not bounded.

Organization. The remainder of this paper is organized as follows. We first define a preliminary result for computing worst-case response times while performing a processor demand analysis (e.g., [13]), then we extend the FPTAS presented in [6] with release jitter constraints. These results are then combined to define for computing approximate worst-case response times. Nevertheless, we show via a counter-example that the computed approximate worst-case response times values are not guaranteed to be close to actual worst-case response times (i.e., with a bounded error).

2 Task Model and Exact Analysis

2.1 Task Model

In this paper, we assume that all tasks share a processor upon which all jobs must execute. Every job can be preempted and resumed later at no cost or penalty. Without loss of generality, we also assume the the rate of the processor is exactly one, since if it is not the case all processing requirements can be normalized to the processor speed.

A task τ_i, $1 \leq i \leq n$, is defined by a worst-case execution requirement C_i, a relative deadline D_i and a period T_i between two successive releases. Every task occurrence is called a job. We assume that deadlines are constrained: $D_i \leq T_i$. Such an assumption is realistic in many real-world applications and also leads to simpler algorithms for checking feasibility of task sets [12]. Moreover, we define the utilization factor of the periodic tasks as follows: $U \stackrel{\text{def}}{=} \sum_{i=1}^{n} C_i/T_i$. We consider a discrete scheduling model and thus we assume that all parameters are integers.

In order to model delay due to input data communications of tasks, we also consider that jobs are subjected to release jitter. A release jitter J_i of a task τ_i is a interval of time after the release of a job in which it waits before starting its execution. In the following, we assume that $0 \leq J_i \leq D_i$ (otherwise the system is obviously not schedulable). Release jitter constraints model delays introduced by the RTOS in presence of system ticks or input communications. For this latter case, dependencies among distributed tasks are modeled using the parameters J_i, $1 \leq i \leq n$. Using such a model, a distributed system can be analysed processor by processor, separately using for instance an holistic based schedulability analysis [19].

For a given processor, we assume that all tasks are independent and synchronously released. All tasks have static priorities that are set before starting the application and are never changed at run-time. At any time, the highest priority task is selected for execution among ready tasks. Without loss of generality, we assume next that tasks are indexed according to priorities: τ_1 is the highest priority task and τ_n is the lowest priority one.

2.2 Known Results

2.2.1 Request-Bound and Workload Functions

In presence of release jitter constraints, the request-bound function of a task τ_i at time t (denoted $\text{RBF}(\tau_i, t)$) and the cumulative processor demand (denoted $W_i(t)$) of tasks at time t of tasks having priorities greater than or equal to i are respectively (see [19] for details):

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i \quad (1)$$

$$W_i(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, t) \quad (2)$$

Informally, the request-bound function for a task τ_i and positive t is the maximum execution requirement of jobs of τ_i released in any continuous interval of length t.

Using these functions, two distinct (but linked) exact feasibility tests can be defined. We restate both results that will be reused in the remainder.

2.2.2 Time-Demand Analysis

The time-demand approach checks that the processor capacity is always less than or equal to the processor capacity required by task executions. [13] presents a processor-

demand schedulability test for constrained-deadline systems (but the test was extended for arbitrary deadline systems in [12]). It can be also easily extended to tasks subjected to release jitter as stated in the following result (a proof can be found in [8]):

Theorem 1 *[13, 15] A static-priority system with release jitter contraints is feasible iff* $\max_{i=1\ldots n}\left\{\min_{t\in S_i}\frac{W_i(t)}{t}\right\} \leq 1$, *where* S_i *is the set of scheduling points defined as follows:* $S_i \stackrel{\text{def}}{=} \{aT_j - J_j \mid j = 1\ldots i, a = 1\ldots\left\lfloor\frac{D_i-J_i+J_j}{T_j}\right\rfloor\} \cup\{D_i - J_i\}$.

Note that schedulability points correspond to a set of time instants in the schedule where a task can start its execution, after the delay introduced by its release jitter. From a computational complexity point of view, for any integer k, there is a task system with two tasks such that the time complexity of the time-demand analysis is at least $O(k)$ (Lemma 1, [15]).

2.2.3 Response-Time Analysis

An alternative approach for checking the feasibility of a static-priority task set is to compute the worst-case response time R_i. The worst-case response time of τ_i is formally defined as:

Definition 1 *The worst-case response time of a task* τ_i *subjected to a release jitter is:* $R_i \stackrel{\text{def}}{=} (\min\{t > 0 \mid W_i(t) = t\}) + J_i$.

Note that for infeasible tasks R_i does not necessarily correspond to the worst case response time, but instead only corresponds to the worst-case response time of the first job of τ_i.

Exact algorithms for calculating the worst-case response time of periodic tasks are known (e.g., see [10] for a recursive definition of the following method). Using successive approximations starting from a lower bound of R_i, we can compute the smallest fixed point of $W_i(t) = t$ with the following sequence. By Definition 1, this smallest fixed point is the worst-case response time for feasible task τ_i.

$$W_i^{(0)} = \sum_{j=1}^{i} C_j$$

$$W_i^{(k+1)} = C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, W_i^{(k)})$$

The recursion terminates (assuming that $U \leq 1$) for the smallest integer k such that: $W_i^{(k+1)} = W_i^{(k)}$ (i.e., the smallest fixed point of the equation $W_i(t) = t$ has been reached).

The processor-demand analysis and the response-time analysis are both based on the cumulative request-bound function (i.e., Equation 2).

Nevertheless, to the best of our knowledge, no direct link is known between these methods for validating static-priority task sets. In this section, we propose combining the aforementioned analysis techniques in an algorithm that calculates the response time of a periodic task in the presence of release jitter constraints. The initial value (e.g., $W_i^{(0)}$) plays an important role to limit the number of required iterations to reach the smallest fixed point of equation $W_i(t) = t$. Different approaches have been proposed in [18, 2] and are quite useful in practice to reduce computation time. Nevertheless, such improvements are not necessary to present our results and for that reason are not developed in the remainder.

As in the processor-demand approach, the worst-case response-time computation can be done in pseudo-polynomial time. Furthermore, for any integer k, there is a task system with two tasks such that the time complexity of the response-time analysis is at least $O(k)$ (Lemma 2, [15]).

2.3 A Preliminary Result

We show that worst-case response times of tasks can be computed using a Time-Demand Analysis (i.e., using Theorem 1), for every feasible task set. For a feasible task τ_i, it is sufficient to check the following testing set [13]:

$$S_i \stackrel{\text{def}}{=} \{aT_j - J_j \mid j = 1\ldots i, a = 1\ldots\left\lfloor\frac{D_i-J_i+J_j}{T_j}\right\rfloor\} \cup\{D_i - J_i\}$$

We first define the critical scheduling point that facilitates the computation of the worst-case response time of τ_i (under the assumption that the task τ_i will meet its deadline at execution time).

Definition 2 *The critical scheduling point for a feasible task* τ_i *is:* $t^* \stackrel{\text{def}}{=} \min\{t \in S_i \mid W_i(t) \leq t\}$.

We now prove if t^* exists, then $W_i(t^*) + J_i$ defines the worst-case response time of τ_i.

Theorem 2 *The worst-case response time of a feasible task* τ_i *is exactly* $R_i = W_i(t^*) + J_i$.

Proof:

Since task τ_i is feasible then we verify that $W_i(t^*) \leq t^*$. Let $S_i = \{t_{i1}, t_{i2}, \ldots, t_{i\ell}\}$ with $t_{i1} < t_{i2} < \cdots < t_i^* < \cdots < t_{i\ell} = D_i - J_i$. By Definition 2, there exists $t^* = t_{ij}$, where $1 \leq j \leq \ell$, is the first scheduling point verifying $W_i(t^*) \leq t^*$: $W_i(t) > t$ for all $t \in \{t_{i1}, \ldots, t_{ij-1}\}$ and $W_i(t_{ij}) \leq t_{ij}$.

Since $W_i(t)$ is non-decreasing between subsequent scheduling points $\{t_{ia}, t_{ia+1}\}$, $1 \leq a \leq \ell - 1$, then there exists a time $t \in (t_{ij-1}, t_{ij}]$ such that $W_i(t) = t$. Since scheduling points in S_i corresponds to task releases, then no new task is released between t and t^* and as a consequence we have $W_i(t) = W_i(t^*)$. The worst-case response time of τ_i is then defined as $W_i(t^*) + J_i$. ■

Tasks	C_i	D_i	T_i	J_i
τ_1	1	3	3	2
τ_2	2	5	5	1
τ_3	1	12	12	2

Table 1. Static-priority task set with release jitter constraints

$t \in S_i$	1	4	7	9
$W_1(t)/t$	1			
$W_2(t)/t$	3	1		
$W_3(t)/t$	4	1.25	1.14	1

Table 2. Exact Time-Demand Analysis

Thus, for all feasible tasks, we can compute their worst-case response times. But, t^* is not defined for an infeasible task τ_i , thus there is no scheduling point $t \in S_i$ such that $W_i(t) \leq t$. For this latter case, the presented method cannot be used to compute a worst-case response time (i.e., some scheduling points after the deadline must be considered).

Since the size of S_i depends on $\sum_{j=1}^{i-1} \lfloor \frac{D_i - J_i + J_j}{T_j} \rfloor$, then the algorithm runs in *pseudo*-polynomial time. Note that computing the smallest fixed-point $W_i(t) = t$ using successive approximation is also performed in pseudo-polynomial time.

Let us take an example, consider the task set presented in Table 1. The utilization factor is $U = 0.81$. The computations associated with the exact tests are given in Table 2. Figure 1 presents $W_3(t)$ and the processor capacity (i.e., $f(t) = t$). Notice that for every task $\tau_i, 1 \leq i \leq n$ the first value such that $W_i(t)/t \leq 1$ leads to its exact worst-case response time:

- for τ_1, $R_1 = W_1(1) + J_1 = 1 + 2 = 3$,
- for τ_2, $R_2 = W_2(4) + J_2 = 4 + 1 = 5$,
- for τ_3, $R_3 = W_3(9) + J_3 = 9 + 2 = 11$.

3 A FPTAS for Feasibility Analysis of a Task

3.1 Approximating the Request-Bound Function

For synchronous task systems without release jitter, the worst-case activation scenario for the tasks occurs when they are simultaneously released [14]. When tasks are subjected to release jitter, then the worst-case processor workload occurs when all higher-priority tasks are simultaneously available after J_i units of time (e.g., when their input data are available). Notice that deadline failures of τ_i (if any) occur necessarily in an interval of time where only tasks with a priority higher or equal to i are running. Such an interval of time is defined as a level-i busy period [13]. When analysing a task τ_i, if we assume that the analysed processor busy period starts at time 0, then the worst-case workload in that busy period is defined by the release of task τ_j at time $-J_j$, $j \leq i$. According to such a scenario, the total execution time requested at time t by a task τ_i is defined by [19]: $\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i$.

The RBF function is a discontinuous function with a "step" of height C_i every T_i units of time. In order to approximate the request bound function according to an error bound ϵ (accuracy parameter, $0 < \epsilon < 1$), we use the same principle as in [6, 5]: we consider the first $(k-1)$ steps of $\text{RBF}(\tau_i, t)$, where k is defined as $k = \lceil 1/\epsilon \rceil - 1$ and a linear approximation, thereafter. The approximate request bound function is defined as follow:

$$\delta(\tau_i, t) = \begin{cases} \text{RBF}(\tau_i, t) & \text{for } t \leq (k-1)T_i - J_i, \\ C_i + (t + J_i)\frac{C_i}{T_i} & \text{otherwise.} \end{cases} \quad (3)$$

Notice that up to $(k-1)T_i - J_i$ the approximate request-bound function is equivalent to the exact request-bound function of τ_i, and after that it is approximated by a linear function with a slope equal to the utilization factor of τ_i. The next subsection describes how we use the approximation to the request-bound function to obtain an approximation scheme for feasibility analysis of static-priority tasks subjected to release jitter constraints.

3.2 Approximation Scheme

[19] shows that a static-priority task system with release jitter constraints is feasible, iff, worst-case response times of tasks are not greater than their relative deadlines. This problem is known as the *release jitter problem*. An alternative way is to define a time-demand approach for solving the release jitter problem using the principles of the well-known exact feasibility test presented for the rate monotonic scheduling algorithm in [13].

As presented in Theorem 1, the cumulative request bound function at time t is defined by: $W_i(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \text{RBF}(\tau_j, t)$. A task τ_i is feasible (with a constrained relative deadline) iff, there exists a time t, $0 \leq t \leq D_i - J_i$, such that $W_i(t) \leq t$. Since request bound functions are step functions, then $W_i(t)$ is also a step function that increases for every scheduling point in the following set $S_i = \{t = bT_a - J_a; a = 1 \ldots i, b = 1 \ldots \left\lfloor \frac{D_i - J_i + J_a}{T_a} \right\rfloor\} \cup \{D_i - J_i\}$. The feasibility test can then be formulated as follows: if there exists a scheduling point $t \in S_i$, such that $W_i(t)/t \leq 1$ then the task is feasible.

To define an approximate feasibility test, we define an approximate cumulative request bound function as:

$$\widehat{W}_i(t) \stackrel{\text{def}}{=} C_i + \sum_{j=1}^{i-1} \delta(\tau_j, t)$$

According to the error bound ϵ leading to $k = \lceil 1/\epsilon \rceil - 1$, we define the following testing set $\widehat{S}_i \subseteq S_i$:

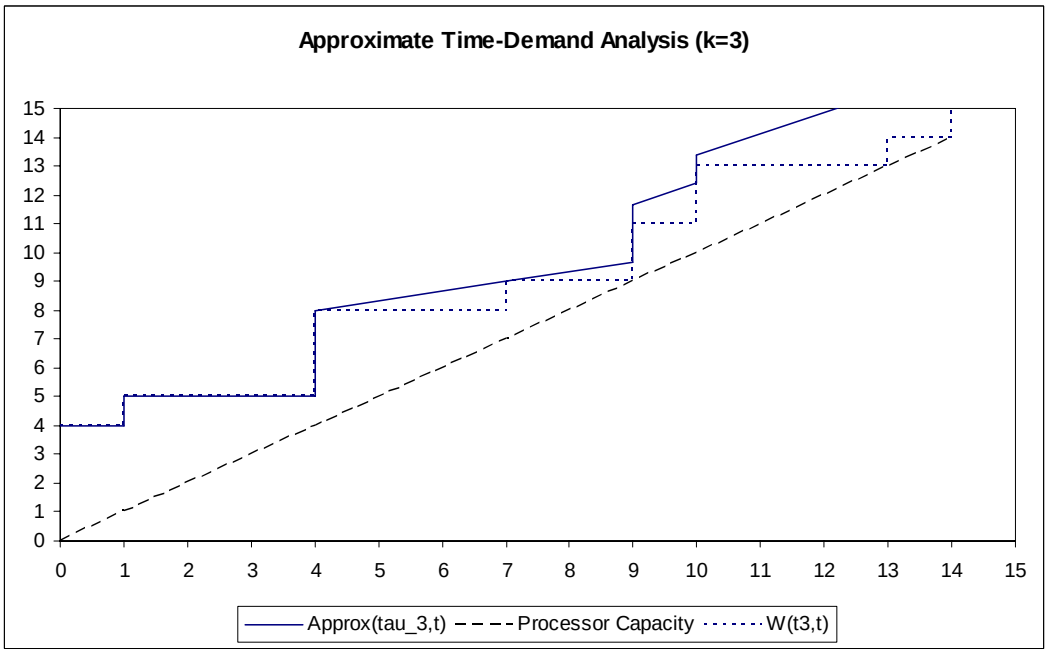

Figure 1. Exact and approximate cumulative request bound functions $W_3(t)$ and $\widehat{W_3}(t)$ with $\epsilon = 0.3$ leading to $k = 3$. Steps occurs at time $aT_i - J_i$ where $0 < a \leq k-1$ and $0 \leq i \leq n$ before starting linear approximations. The approximate test concludes that τ_3 is not feasible upon a $(1-\epsilon)$-speed processor.

$$\widehat{S_i} \stackrel{\text{def}}{=} \{t = bT_a - J_a; a = 1 \ldots i-1, b = 1 \ldots k-1\} \cup \{D_i - J_i\}$$

We consider the task set presented in Table 1, the cumulative request bound function $\widehat{W_3}(t)$ is presented in Figure 1 using $\epsilon = 0.3$. This means exactly three steps will be considered for every task (i.e., $k = 3$) before approximating the request bound function using a linear function. We indicate without providing computation details that worst-case response times of τ_1 and τ_2 can be exactly computed since they are achieved before approximating request bound functions. But as shown in Figure 1, the approximate feasibility test concludes that τ_3 is not feasible because $\widehat{W_3}(t) > t$ for all scheduling points (i.e., for all $t \in \widehat{S_3}$).

This is a FPTAS since the algorithm is polynomial according to the input size and the input parameter $1/\epsilon$. We now prove the correctness of this approximate feasibility test.

3.3 Correctness of Approximation

The key point to ensure the correctness is: $\delta(\tau_i, t)/\text{RBF}(\tau_i, t) \leq (1+\epsilon)$. This result will then be used to prove that if a task set is stated infeasible by the FPTAS, then it is infeasible under a $(1-\epsilon)$ speed processor.

Theorem 3 *$\forall t \geq 0$, we verify that: $\text{RBF}(\tau_i, t) \leq \delta(\tau_i, t) \leq (1+\frac{1}{k})\text{RBF}(\tau_i, t)$ where $k = \lceil \frac{1}{\epsilon} \rceil - 1$.*

Proof: We first prove the first inequality: for all $t \in [0, (k-1)T_i - J_i]$

$$\delta(\tau_i, t) = \text{RBF}(\tau_i, t)$$

For $t > (k-1)T_i - J_i$:

$$\delta(\tau_i, t) = C_i + (t + J_i)\frac{C_i}{T_i} = C_i\left(1 + \frac{t+J_i}{T_i}\right)$$

As a consequence:

$$\delta(\tau_i, t) \geq \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i = \text{RBF}(\tau_i, t)$$

We now prove the second inequality of the statement: If $\delta(\tau_i, t) > \text{RBF}(\tau_i, t)$ then since $t > (k-1)T_i - J_i$ then $k-1$ steps before approximating the request bound function, we verify:

$$\text{RBF}(\tau_i, t) \geq kC_i \tag{4}$$

Furthermore,

$$\delta(\tau_i, t) - \text{RBF}(\tau_i, t) \leq C_i$$

This is obvious if $t \in [0, (k-1)T_i - J_i]$ since $\delta(\tau_i, t) = \text{RBF}(\tau_i, t)$, and if $t > (k-1)T_i - J_i$, then:

$$\begin{aligned} \delta(\tau_i, t) - \text{RBF}(\tau_i, t) &= C_i + (t + J_i)\frac{C_i}{T_i} - \left\lceil \frac{t+J_i}{T_i} \right\rceil C_i \\ &\leq C_i \end{aligned}$$

As a consequence: $\delta(\tau_i, t) \leq \text{RBF}(\tau_i, t) + C_i$ and using inequality (4), we obtain the result:

$$\delta(\tau_i, t) \leq (1 + \frac{1}{k})\text{RBF}(\tau_i, t)$$

As a consequence, both inequalities are verified. ■

Using the same approach presented in [6, 5], we can establish the correctness of approximation.

Theorem 4 *If there exists a time instant $t \in (0, D_i - J_i]$, such that $\widehat{W}_i(t) \leq t$, then τ_i is feasible (i.e., $W_i(t) \leq t$).*

Proof: Directly follows from Theorem 3. ■

Theorem 5 *If $\forall t \in (0, D_i - J_i]$, $\widehat{W}_i(t) > t$, then τ_i is infeasible on a processor of $(1-\epsilon)$ capacity.*

Proof: Assume that $\forall t \in (0, D_i - J_i]$, $\widehat{W}_i(t) > t$, but τ_i is still feasible on a $(1-\epsilon)$ speed processor. Since assuming τ_i to be feasible upon a $(1-\epsilon)$ speed processor, then there must exist a time t_0 such that τ_i: $W_i(t_0) \leq (1-\epsilon)t_0$. But, using Theorem 3 we verify that $\widehat{W}_i(t) \leq (1 + \frac{1}{k})W_i(t)$, where $k = \lceil \frac{1}{\epsilon} \rceil - 1$, then for all $t \in (0, D_i - J_i]$, the condition $\widehat{W}_i(t) > t$ implies that $\forall t \in (0, D_i - J_i]$:

$$W_i(t) > \frac{t}{1 + \frac{1}{k}} > \frac{k}{k+1}t \geq (1-\epsilon)t.$$

As a consequence, a time t_0 such that $W_i(t_0) \leq (1-\epsilon)t_0$ cannot exist and τ_i is infeasible. ■

To conclude the correctness, we must prove that scheduling points are sufficient.

Theorem 6 *For all $t \in \widehat{S}_i$ such that $\widehat{W}_i(t) > t$, then we also verify that: $\forall t \in (0, D_i - J_i], \widehat{W}_i(t) > t$.*

Proof: Let t_1 and t_2 be two *adjacent* points in $\widehat{S}_i$ (i.e., $\nexists\ t \in \widehat{S}_i$ such that $t_1 < t < t_2$). Since $\widehat{W}_i(t_1) > t_1$, $\widehat{W}_i(t_2) > t_2$ and the fact that $\widehat{W}_i(t)$ is an non-decreasing step left-continuous function we conclude that $\forall t \in (t_1, t_2)\ \widehat{W}_i(t) > t$ (see Figure 2 for details). The property follows. ■

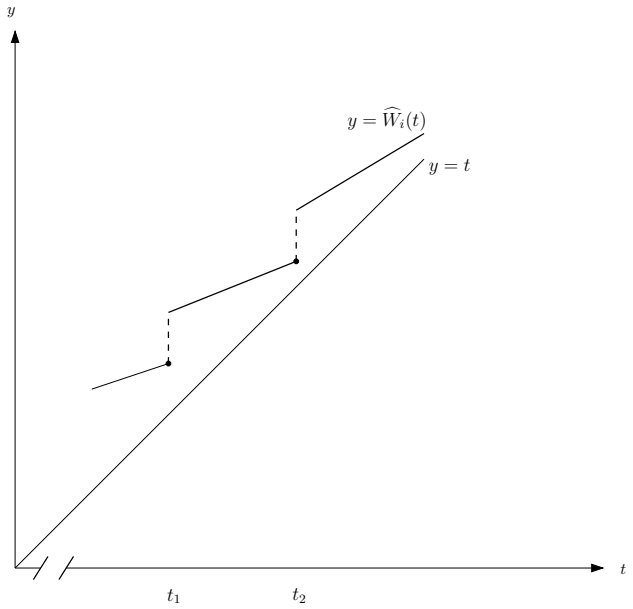

Figure 2. The scheduling points $\widehat{S}_i$ are sufficient

4 Approximate Response-Time Analysis with Release Jitter

4.1 Approximate worst-case response time upper bound

According to a accuracy parameter ϵ, we define approximate worst-case response times as in the classical Combinatorial Optimization Problem theory [7]:

Definition 3 *Let ϵ be a constant and R_i be the worst-case response time of a task τ_i, then the approximate worst-case responses time $\widehat{R}_i$ satisfies: $R_i \leq \widehat{R}_i \leq (1+\epsilon)R_i$.*

We shall combine results presented in Sections 2 and 3, in order to define approximate worst-case response times. Using the FPTAS presented in Section 3, we can check that a task is feasible or not. If it is feasible, then we are able to compute an upper bound of the worst-case response time of a task as presented in Section 2.

Definition 4 *Consider a task τ_i such that there exists a time t satisfying $\widehat{W}_i(t) \leq t$, then an approximate worst-case response time is defined by:*
$t^* \stackrel{\text{def}}{=} \min\left(t \in \widehat{S}_i \mid \widehat{W}_i(t) \leq t\right)$ *and* $\widehat{R_i} \stackrel{\text{def}}{=} \widehat{W}_i(t^*) + J_i$.

We now prove that such a method defines an upper bound of the worst-case response time of task τ_i.

Theorem 7 *For every task τ_i such that there exists a time t satisfying $\widehat{W}_i(t) \leq t$, then: $R_i \leq \widehat{R_i}$*

Proof: Let t be a scheduling point such that $\widehat{W}_i(t) \leq t$. From the approximate feasibility test, we verify that τ_i is feasible: there exists a time t^* such that $W_i(t^*) \leq t^*$ and $t^* \leq t$. Since $R_i = W_i(t^*) + J_i$ and $\widehat{R_i} = \widehat{W}_i(t) + J_i$ then, it follows from properties of the approximate feasibility test that $R_i \leq \widehat{R_i}$. ■

4.2 The Algorithm

The complete algorithm for computing approximate worst-case response time of a task τ_i is presented in Algorithm 1. The algorithm contains three nested loops. The first loop and the last one are bounded by n (i.e., the number of tasks). The second one is related to k, thus on the value $1/\epsilon$. Thus, this implementation of the approximate

feasibility test for a given task leads to a $O(n^2/\epsilon)$ algorithm. This algorithm is eligible to be a FPTAS since it is polynomial in the size of the task set and the accuracy parameter $1/\epsilon$. But, as we will prove in the next section, it does not lead to bounded performance guarantee on computed response times in comparison with an exact response time analysis (performed with a pseudo-polynomial time algorithm).

4.3 Worst-case analysis of the algorithm performance guarantee

We now show that this method does not lead to an approximation algorithm (i.e., with the expected bounded error presented in Definition 3) even if the approximate feasibility analysis returns a positive answer.

Theorem 8 *There exist some task systems for which* $cR_i \leq \widehat{R_i}$ *for any integer* c*.*

Proof: Let us consider a task system with two tasks with the following parameters: τ_1 with $C_1 = 1-\lambda$ and $T_1 = 1$ and τ_2 with $C_2 = k\lambda$ and $T_2 = k+1/\lambda$, where $0 < \lambda < 1$ and k is an arbitrary integer. (Both tasks have their jitter parameter equal to zero). With these parameters and the Rate-Monotonic scheduling policy, the task τ_2 can only be executed λ unit of time within any interval of length one in the schedule. The τ_2 completes at time k. The approximate feasibility analysis leads to the following computations:

$$\begin{aligned} \widehat{W}_2(t) &= k\lambda + \delta(\tau_1, t) \\ &= k\lambda + (1-\lambda) + t(1-\lambda) \end{aligned}$$

The corresponding approximate worst-case response time will be achieved for $\widehat{W}_2(t) = t$. The approximation switches to a linear approximation at time $(k-1)T_1 = k-1$. The corresponding fixed-point t is:

$$\begin{aligned} t &= k\lambda + (1-\lambda) + t(1-\lambda) \\ t &= k - 1 + \frac{1}{\lambda} \end{aligned}$$

As a consequence the approximate worst-case response time is: $\widehat{R_2} = k - 1 + 1/\lambda$. The approximate feasibility analysis always predicts that the task system is feasible for any integer k since the approximate worst-case response time is strictly less than the deadline of task τ_2. Therefore, the approximate response time is strictly larger than the exact, and can be made arbitrarily large: the ratio between the exact worst-case response time and the approximate one is exactly:

$$\frac{\widehat{R_2}}{R_2} = 1 - \frac{1}{k} + \frac{1}{\lambda k}$$

This ratio increases without any bound as λ approaches zero. So, for any arbitrary integer c, we can find a lambda sufficiently small such that $\widehat{R_2}/R_2 \geq c$. ■

The Figure 3 presents an example of this counterexample with $k = 10, \lambda = 0.1, \epsilon = 0.33$. The exact worst-case response time of τ_2 is 10 and the approximate worst-case response time is 19 (thus τ_2 completes by its deadline equal to 20). Note that the slope of the approximated cumulative request bound function tends to one when λ tends to zero. and thus becomes nearly parallel to the line representing the processor capacity. That is why a performance guarantee can not be achieved using our method.

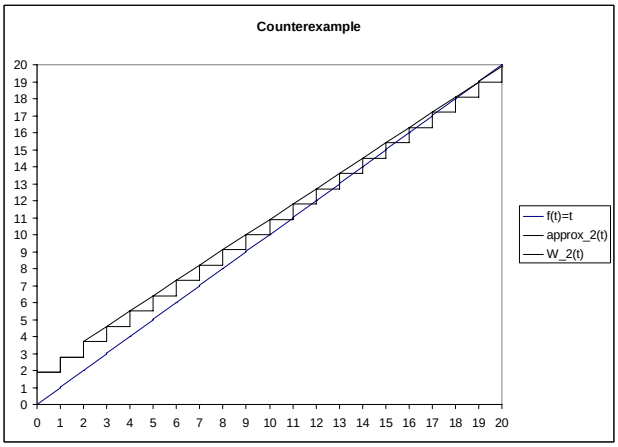

Figure 3. Counterexample with $k = 3, \lambda = 0.1, \epsilon = 0.3$.The exact worst-case response time is 10 and the approximate one is achieved when lines intersect at time 19. Thus, the approximate value is near 2 times greater than the exact wors-case response time. Reducing λ to an arbitrary small value lead to an unbounded performance ratio.

5 Conclusion and Further Work

We presented a method for approximating worst-case response times of static-priority tasks with release jitter constraints. The method is based on a FPTAS performing a feasibility test based on a Time-Demand Analysis. According to an accuracy parameter ϵ, if the approximate feasibility test concludes that a task τ_i is feasible (i.e., meets its deadline) then we can compute an approximate worst-case response time, but without any constant performance guarantee. But, when the approximate feasibility test cannot conclude that τ_i is feasible, we know that τ_i will not be feasible under a processor with capacity $(1-\epsilon)$; however, the proposed approach cannot guarantee that the approximate worst-case response times are within a constant multiplicative factor of the actual worst-case response time. Even if our results are not complete, they allow to define a sufficient feasibility analysis that can be used for analysing a component in a QoS Optimization method or encapsulated within a holistic analysis for analysing distributed real-time systems.

The existence of an approximation scheme (or weakly

Algorithm 1. Approximate worst-case response time of τ_i

input :
ϵ : real /* The FPTAS accuracy parameter */;
i : integer /* Index of the analysed task */;
n : integer /* Size of the task set */;
$C[n], T[n], D[n], J[n]$: array of integers /* Task parameters */;
output: Approximate response time of τ_i or 'not feasible upon a $(1-\epsilon)$-speed processor';

$k = \lceil 1/\epsilon \rceil - 1$ /* k is the number of steps considered in $rbf(\tau_i, t)$ */;
for $j = 1$ **to** $i-1$ **do**
  for $\ell = 1$ **to** k **do** /* for each scheduling point$t$ */
    if $(\ell = k$ **and** $j = i-1)$ **then** $t = D[i] - J[i]$; /* t is the last scheduling point */
    else $t = \ell \times T[j] - J[j]$; /* t is another scheduling points */
    $w = C_i$ /* w is $\delta(\tau_i, t)$ */;
    for $m = 1$ **to** $i-1$ **do** /* for all higher priority tasks */
      if $(t \leq (k-1)T[m] - J[m])$ **then** $w+ = C[m]\lceil t/T[m] \rceil$; /* compute $rbf(\tau_m; t)$ */
      else $w+ = C[m] + (t + J[i])C[m]/T[m]$; /* compute linear approximation */
    end
    if $(t \geq w)$**) then return** $(w + J[i])$; /* approximate response time of τ_i */
  end
end
return *("not feasible upon a* $(1-\epsilon)$*-speed processor")*;

an approximation algorithm) is still an interesting open issue. If such a result exists for the worst-case response time analysis, it will exactly quantify the pessimism of the corresponding sufficient feasibility test.

Acknowledgments

The authors would like to thank anonymous reviewers for their helpful comments that allow to improve the presentation of this paper.

References

[1] E. Bini, G. Buttazzo, and G. Buttazzo. Rate monotonic scheduling: The hyperbolic bound. *IEEE Transactions on Computers*, 2003.

[2] R. Bril, W. Verhaege, and E. Pol. Initial values for on-line response time calculations. *proc. Int Euromicro Conf. on Real-Time Systems (ECRTS'03), Porto*, 2003.

[3] A. Burchard and J. Liebeherr. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 1995.

[4] D. Chen, A. Mok, and T. Kuo. Utilization bound revisited. *IEEE Transactions on Computers*, 2003.

[5] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In I. C. Society, editor, *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 117–126, 2005.

[6] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. *Proceedings of the 13th International Conference on Real-Time Systems,Paris, France*, pages 233–249, 2005.

[7] M. Garey and D. Johnson. Computers and intractability: a guide to the theory of np-completeness. *WH Freeman and Company*, 1979.

[8] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. *Technical Report 2966, Institut National de Recherche en Informatique et Automatique (INRIA), France*, 1996.

[9] C. Han and H. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. *proc 18th IEEE Real-Time Systems Symposium (RTSS'97)*, 1997.

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

[11] S. Lauzac, R.Melhem, and D. Mossé. An improved rate-monotonic admission control and its applications. *IEEE Transactions on Computers*, 2003.

[12] J. Lehoczky. Fixed-priority scheduling of periodic task sets with arbitrary deadlines. *proc. Real-Time System Symposium (RTSS'90)*, pages 201–209, 1990.

[13] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. *proc. Real-Time System Symposium (RTSS'89)*, pages 166–171, 1989.

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

[15] Y. Manabee and S. Aoyagi. A feasible decision algorithm for rate monotonic and deadline monotonic scheduling. *Real-Time Systems Journal*, pages 171–181, 1998.

[16] D.-W. Park, S. Natarajan, A. Kanavsky, and M. Kim. A generalized utilization bound test for fixed-priority real-time scheduling. *proc. 2sd Workshop on Real-Time Systems and Applications*, 1995.

[17] P. Richard. Polynomial time approximate schedulability tests for fixed-priority real-time tasks: some numerical experimentations. *14th Real-Time and Network Systems, Poitiers (France)*, 2006.

[18] M. Sjodin and H. Hansson. Improved response time analysis calculations. *proc. IEEE Int Symposium on Real-Time Systems (RTSS'98)*, 1998.

[19] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, 1994.